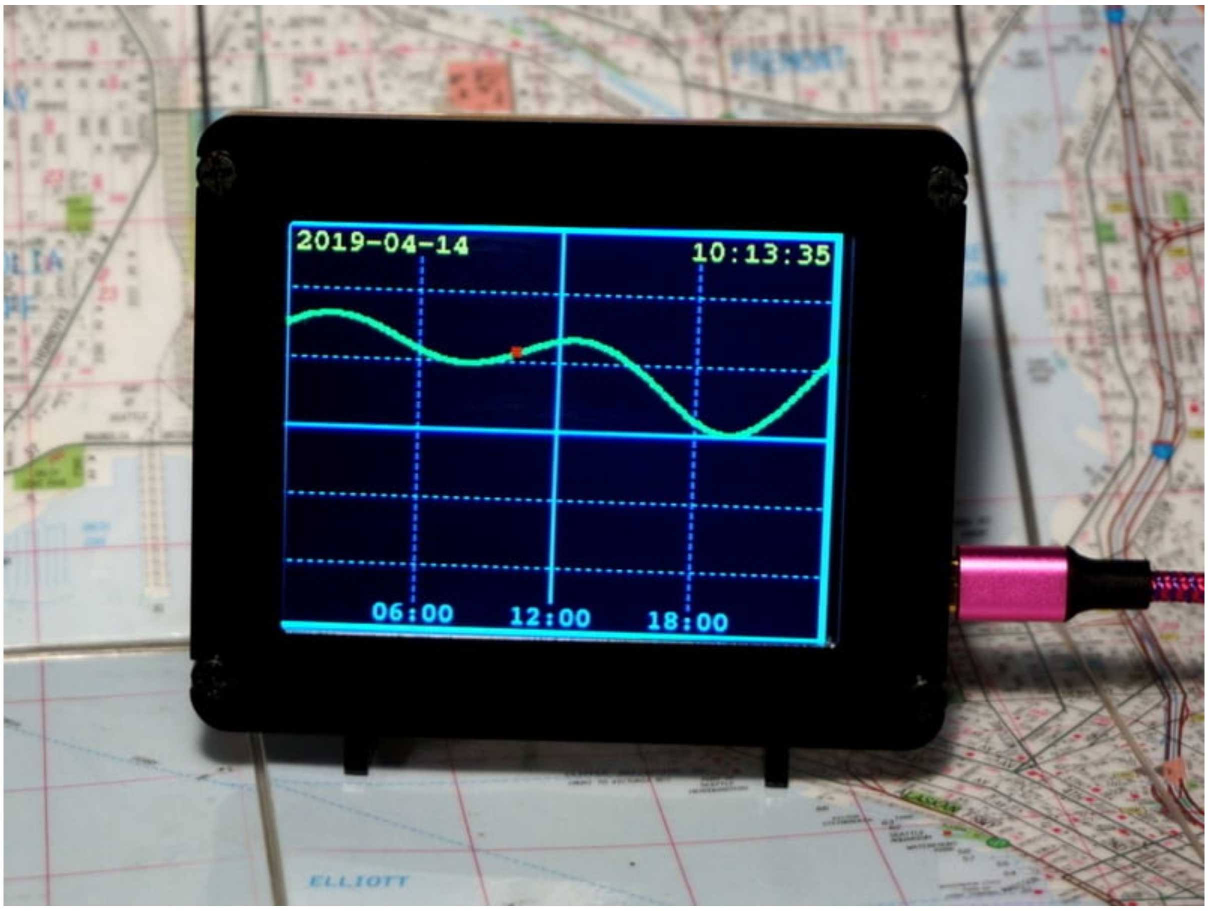
PyPortal Tides Viewer

Created by Carter Nelson



https://learn.adafruit.com/pyportal-tides-viewer

Last updated on 2025-02-26 02:59:52 PM EST

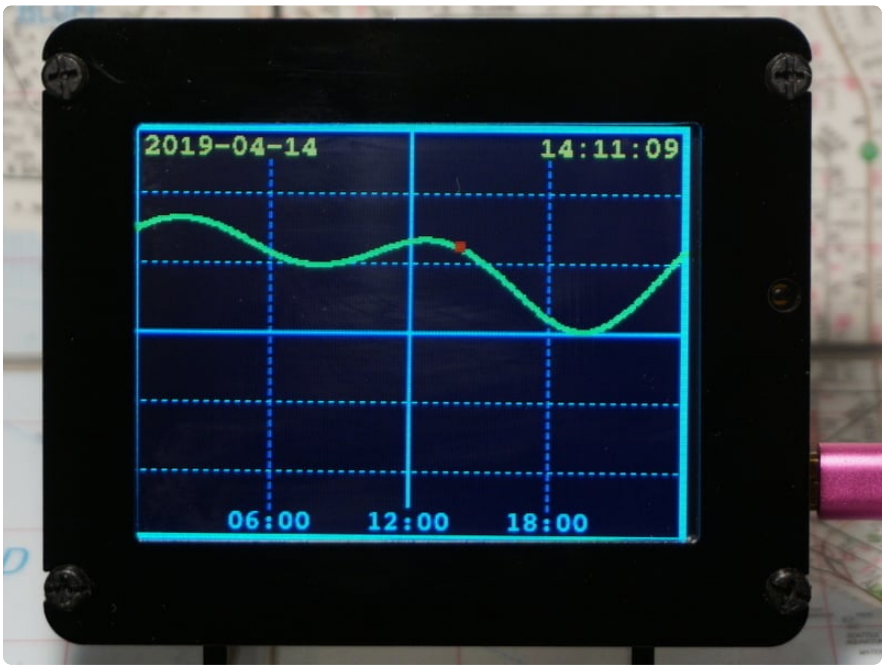
Table of Contents

Overview

Surfs up! Or is it?

Do you live near an ocean or some other tidal body of water? Do you want to know what time high tide is? Low tide? Or even the overall tide level throughout the day? This guide will show how you can use an Adafruit PyPortal smart display to easily fetch tide information from the Internet and display it.

CircuitPython is used for the code and the PyPortal library does all the heavy lifting. All that's left is some initial data entry and it's ready for display.



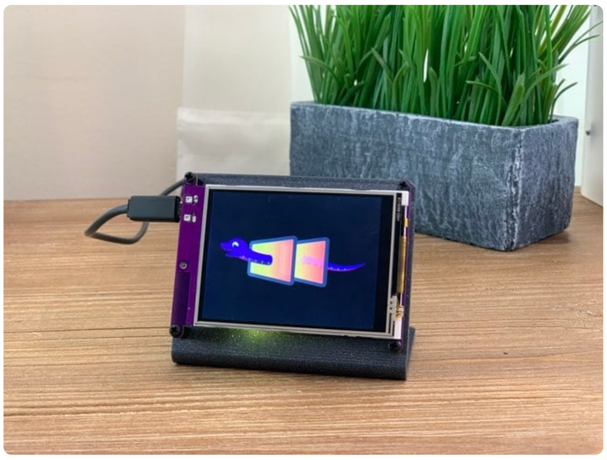
Parts

The parts for this project are available on AdaBox 011 or individually:

AdaBox011 - PyPortal

Reach out beyond your desk - to the stars and beyond - with PyPortal! This ADABOX features a new, easy-to-use IoT device that allows you to customize and create your...

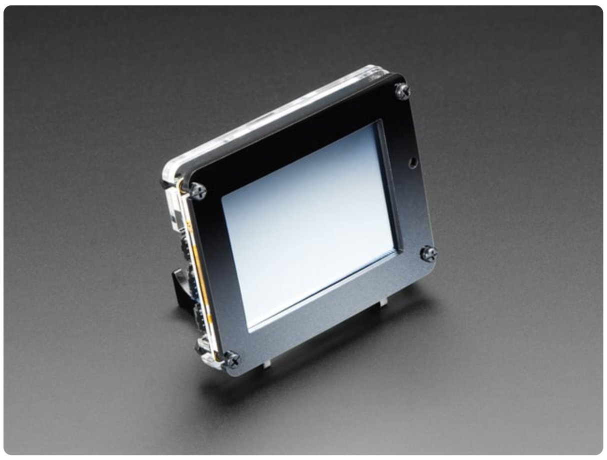
https://www.adafruit.com/product/4061



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...

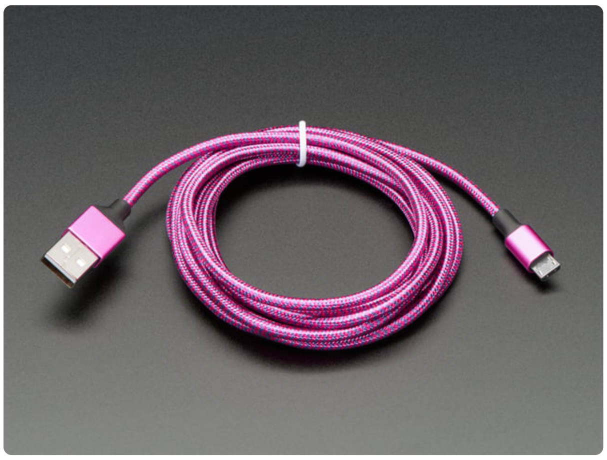
https://www.adafruit.com/product/4116



Adafruit PyPortal 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



Pink and Purple Braided USB A to Micro B Cable - 2 meter long

This cable is super-fashionable with a woven pink and purple Blinka-like pattern! First let's talk about the cover and over-molding. We got these in custom colors,...

https://www.adafruit.com/product/4148

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

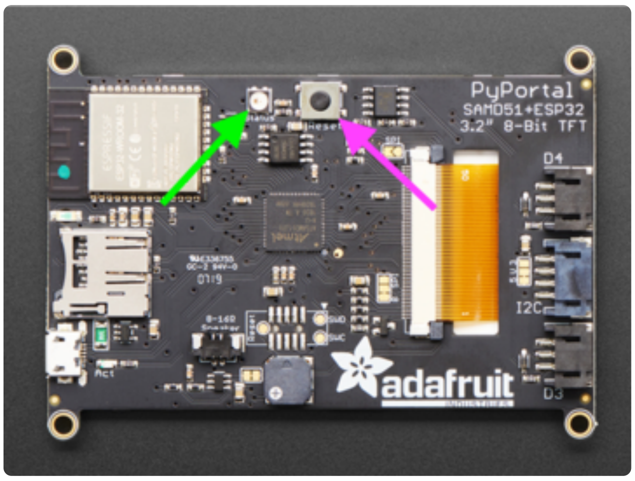
https://adafru.it/HFd

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).

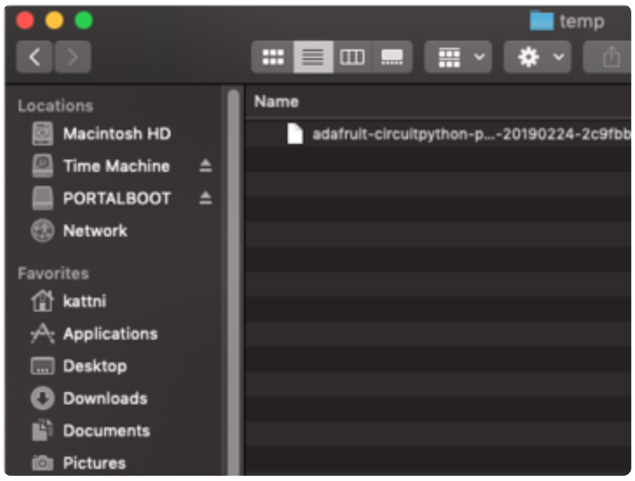
Plug your PyPortal into your computer using a known-good USB cable.

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.

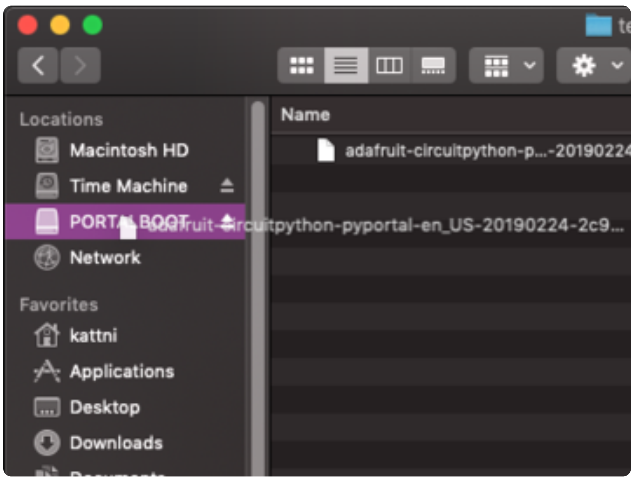


Double-click the **Reset** button on the top in the middle (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

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



adafruit-circuitpython-p...-20190224-2c9fbb

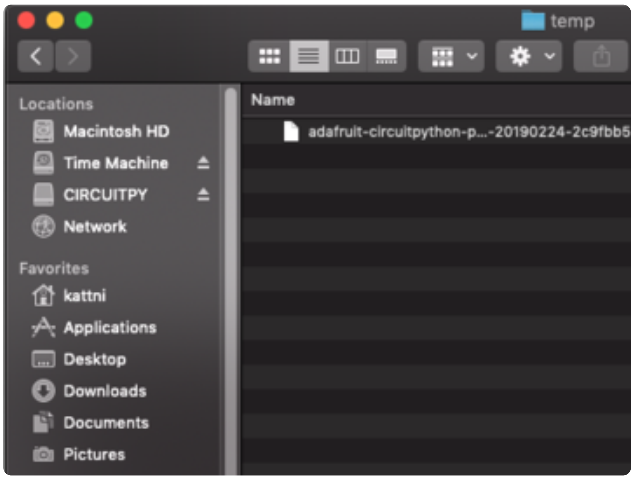


adafruit-circuitpython-p...-20190224

adafruit-circuitpython-pyportal-en_US-20190224-2c9...

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

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



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

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

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

PyPortal Default Files

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

PyPortal Default Files

https://adafru.it/UF-

PyPortal Pynt Default Files

https://adafru.it/UGa

PyPortal CircuitPython Setup

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

Adafruit CircuitPython Bundle

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

Latest Adafruit CircuitPython Library Bundle

https://adafru.it/ENC

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

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

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

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

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of CircuitPython 8 (https://adafru.it/Em8), 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.

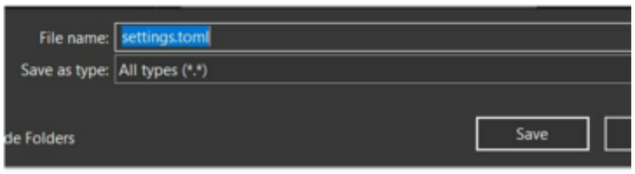
settings.toml File Tips

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

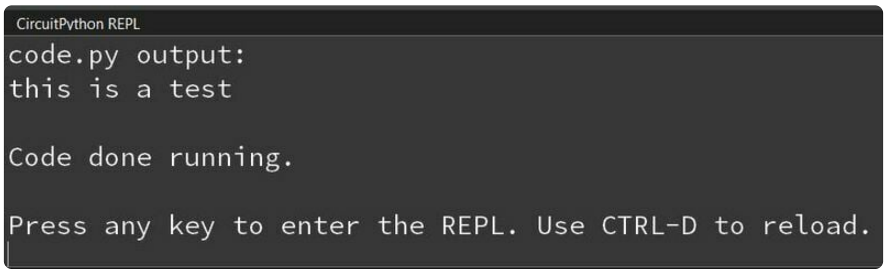


When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

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"))
```



```
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!

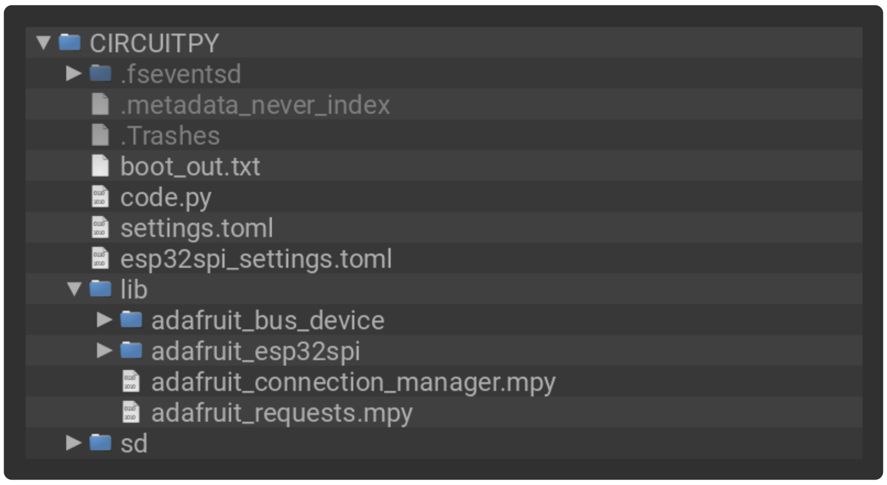
Connect to WiFi

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:



```
▼ CIRCUITPY
  ▶ .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    settings.toml
    esp32spi_settings.toml
  ▼ lib
    ▶ adafruit_bus_device
    ▶ adafruit_esp32spi
      adafruit_connection_manager.mpy
      adafruit_requests.mpy
  ▶ sd
```

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

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

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

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

print("ESP32 SPI webclient test")

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


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

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

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

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

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

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

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

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

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

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

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

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

print("Done!")
```

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

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

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

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

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

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

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

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

#...

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

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

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

- Verify an ESP32 is found, checks the firmware and MAC address

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

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

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

- Connect to the AP we've defined here, then print out the local IP address. Then attempt to do a domain name lookup and ping google.com to check network connectivity. (Note sometimes the ping fails or takes a while; this isn't a big deal.)

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

Now we're getting to the really interesting part of the example program. We've written a library for web fetching web data, named adafruit_requests (https://adafru.it/FpW). 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.

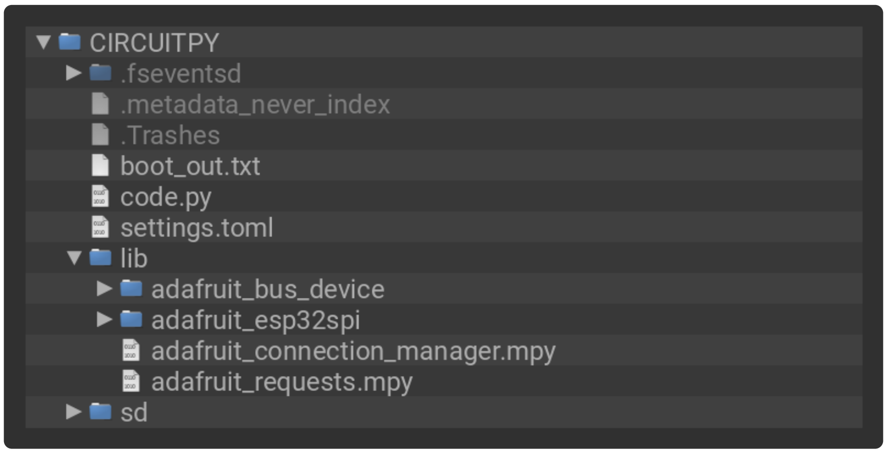
To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

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.

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

import os

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

import adafruit_requests

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

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

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

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

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

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

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

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

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

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

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

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

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

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



```
▼ CIRCUITPY
  ▶ .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    settings.toml
  ▼ lib
    ▶ adafruit_bus_device
    ▶ adafruit_esp32spi
      adafruit_connection_manager.mpy
      adafruit_requests.mpy
  ▶ sd
```

WiFi Manager

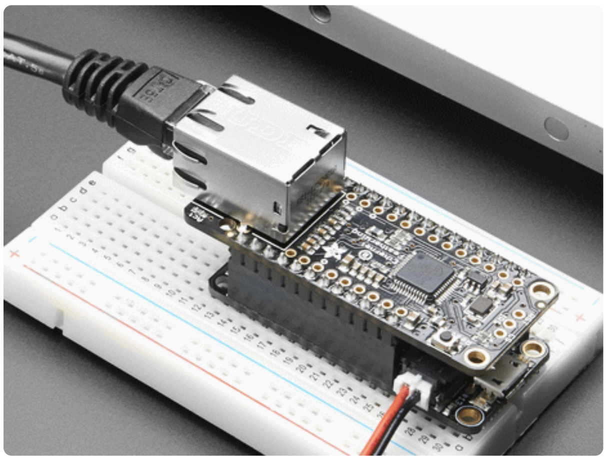
The way the examples above connect to WiFi works but it's a little finicky. Since WiFi is not necessarily so reliable, you may have disconnects and need to reconnect. For more advanced uses, we recommend using the `WiFiManager` class. It will wrap the connection/status/requests loop for you - reconnecting if WiFi drops, resetting the ESP32 if it gets into a bad state, etc.

Here's a more advanced example that shows using the `WiFiManager` and also how to fetch the current time from a web source.

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

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

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

print("ESP32 local time")

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

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

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

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

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

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

the_rtc = rtc.RTC()

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

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

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

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

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

Further Information

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



Networking in CircuitPython
By Anne Barela

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

NOAA Tides Web Service

We will use a web service provided by the National Oceanic and Atmospheric Administration (NOAA (https://adafru.it/ExM)) to get our tide information. One benefit of this is that no API key is needed, since it's your tax dollars at work. However, since this is a part of the US government, the service only covers regions of the United States.

If you live outside the United States, you may have to find a service which sends out tide data as a JSON feed and adapt the code to your new data source. The framework laid out here should provide you a great starting point.

NOAA CO-OPS API

Wow, acronyms! We already covered NOAA above. CO-OPS (https://adafru.it/19kf) stands for Center for Operational Oceanographic Products and Services. Why is there a dash? Don't know, it's just what the government does. But also, it doesn't really matter. If you want to know more, read about it here (https://adafru.it/19kf).

API stands for Application Programming Interface, a general term used a lot in programming. The CO-OPS API is one of many web services that NOAA provides. There's a complete list here (https://adafru.it/ExO).

The main documentation for the web service we will use is here:

CO-OPS API For Data Retrieval

https://adafru.it/ExP

There are a lot of options and types of data that can be returned. We've worked out the magic invocation needed for the PyPortal and you can find it in the code later. The only item you will need to worry about is finding your closest tide monitoring station ID. This is how you will set your location in the code.

So let's see how you can figure that out next.

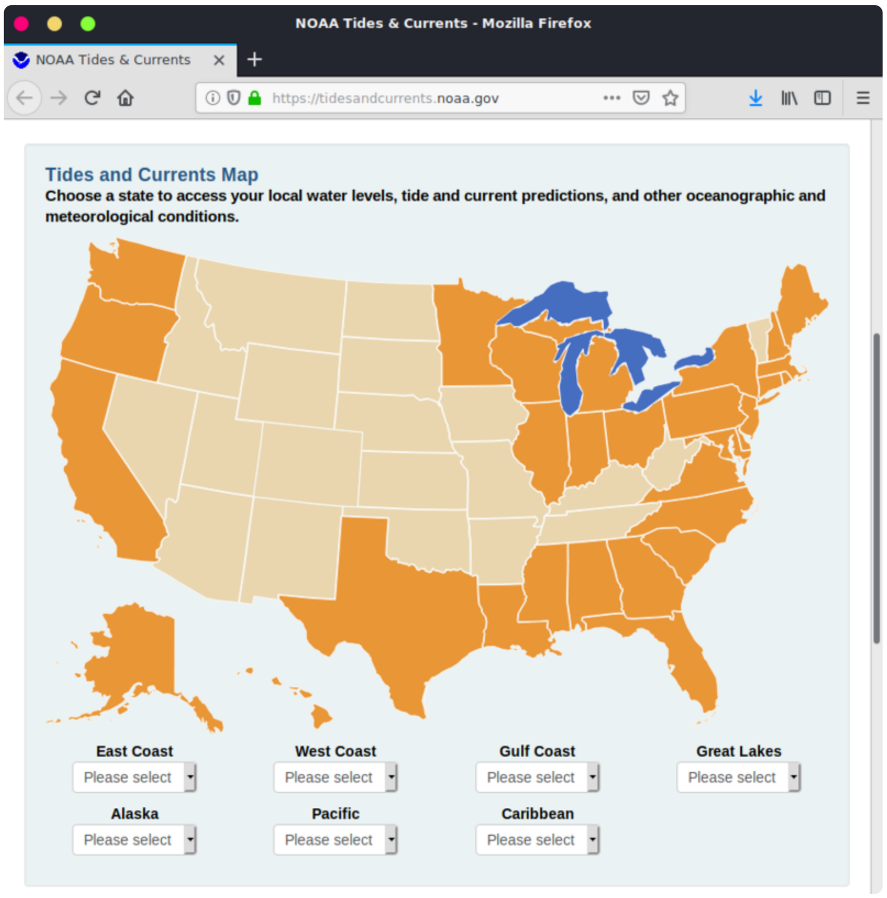
Find Your Tide Station ID

To find you station ID, start by going to this webpage:

Tides and Currents Map

https://adafru.it/ExQ

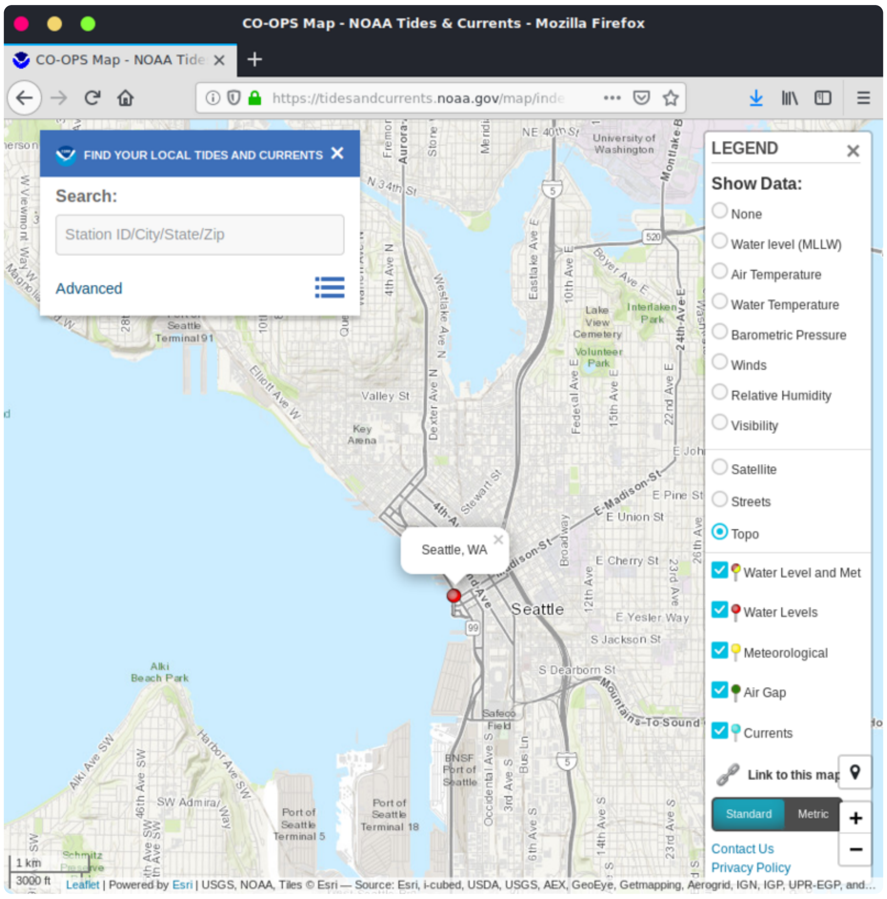
Scroll down a bit and you'll see a map. You can simply click on the state of interest. Note that some states are not clickable. That's because they don't have tides :(


NOAA Tides & Currents - Mozilla Firefox
NOAA Tides & Currents
https://tidesandcurrents.noaa.gov

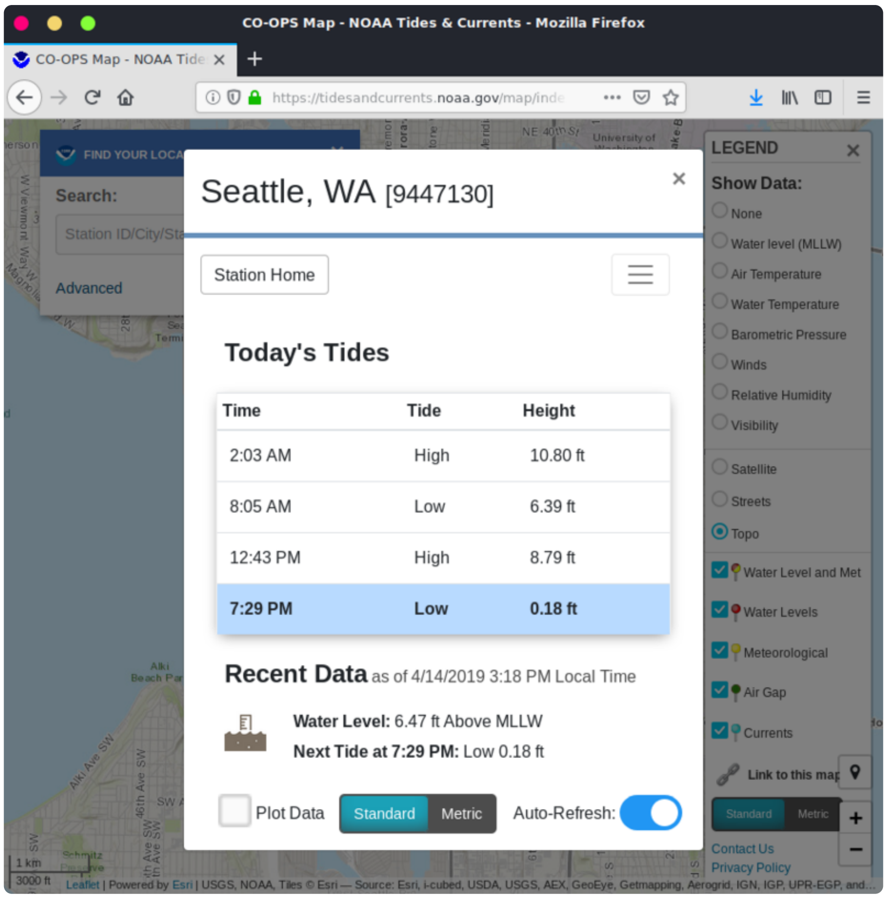
Tides and Currents Map

Choose a state to access your local water levels, tide and current predictions, and other oceanographic and meteorological conditions.

Once you've clicked on a state you'll get a familiar map like interface that you can drag and zoom around. Use that to zoom in and find the station marker that seems like it will work best for your location.



Click on the marker and it will bring up information about that marker.



Seattle, WA [9447130]

Station Home

Today's Tides

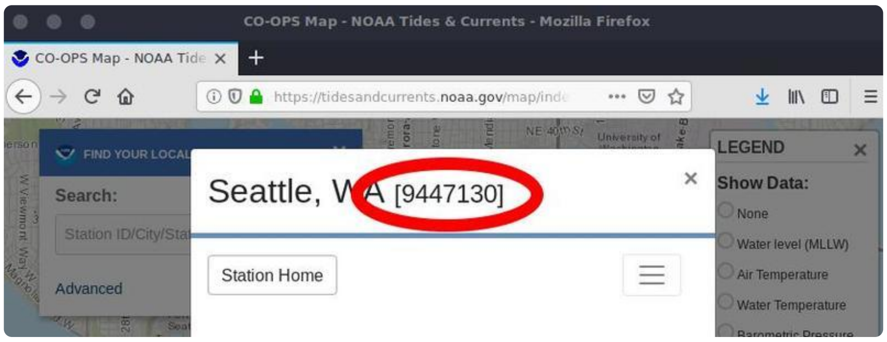
Time	Tide	Height
2:03 AM	High	10.80 ft
8:05 AM	Low	6.39 ft
12:43 PM	High	8.79 ft
7:29 PM	Low	0.18 ft

Recent Data as of 4/14/2019 3:18 PM Local Time

Water Level: 6.47 ft Above MLLW

Next Tide at 7:29 PM: Low 0.18 ft

The station ID will be the number shown near the top. This is what you will enter into your code.

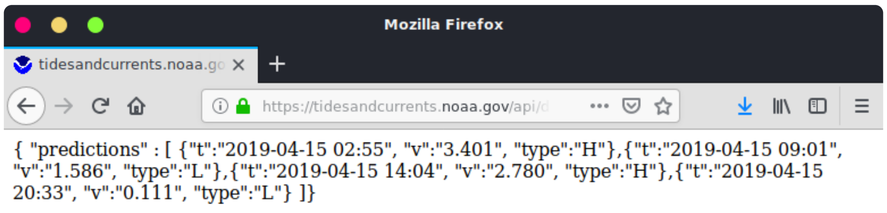


Basic Tide Time Info

Here's the URL that gets the high and low times for the current day at the station location. The station ID is the very last thing in the URL, so you can change it for your location if you want.

```
https://tidesandcurrents.noaa.gov/api/datagetter?
date=today&amp;product=predictions&amp;datum=mllw&amp;interval=hilo&amp;format=json&amp;units=
```

If you put that address in your web browser you'll get something that looks like this:



```
{ "predictions" : [ {"t":"2019-04-15 02:55", "v":"3.401", "type":"H"},{"t":"2019-04-15 09:01",
"v":"1.586", "type":"L"},{"t":"2019-04-15 14:04", "v":"2.780", "type":"H"},{"t":"2019-04-15
20:33", "v":"0.111", "type":"L"} ]}
```

Not a very pretty web page. That's because it's just JSON data. The key **predictions** is the root for all the data. Then there are several entries that look like:

```
{"t":"2019-04-15 02:55", "v":"3.401", "type":"H"}
```

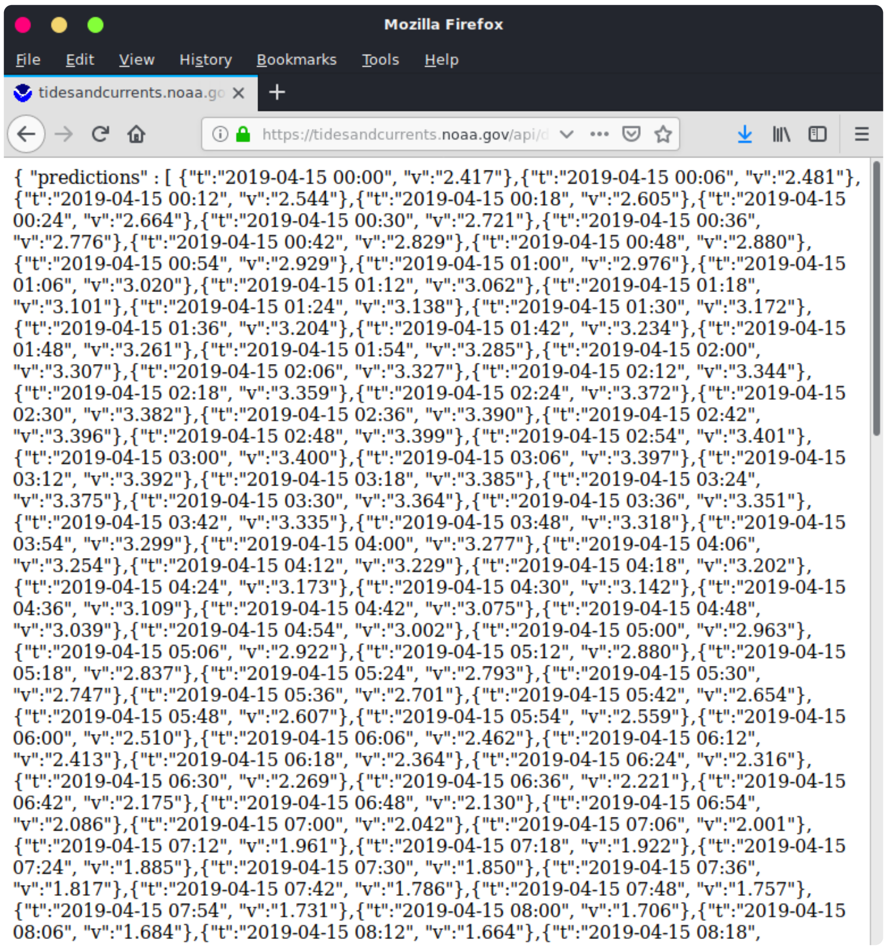
The **t** entry gives us the date and time, the **v** entry gives us the tide level, and the **type** is **H** for high tide or **L** for low tide.

So we have what we need - the time for each of the high and low tides for the given day. You'll see how this is parsed out later in the code.

Tide Levels Throughout the Day

With a slight modification of the URL, we can get predicted water levels in 6 minute increments over the period of the current day. Here's the URL for that:

```
https://tidesandcurrents.noaa.gov/api/datagetter?
date=today&amp;product=predictions&amp;datum=mllw&amp;format=json&amp;units=metric&amp;time_zo
```


Mozilla Firefox
File Edit View History Bookmarks Tools Help
tidesandcurrents.noaa.go
https://tidesandcurrents.noaa.gov/api/d

{ "predictions" : [{"t":"2019-04-15 00:00", "v":"2.417"},{"t":"2019-04-15 00:06", "v":"2.481"},
{"t":"2019-04-15 00:12", "v":"2.544"},{"t":"2019-04-15 00:18", "v":"2.605"},{"t":"2019-04-15
00:24", "v":"2.664"},{"t":"2019-04-15 00:30", "v":"2.721"},{"t":"2019-04-15 00:36",
"v":"2.776"},{"t":"2019-04-15 00:42", "v":"2.829"},{"t":"2019-04-15 00:48", "v":"2.880"},
{"t":"2019-04-15 00:54", "v":"2.929"},{"t":"2019-04-15 01:00", "v":"2.976"},{"t":"2019-04-15
01:06", "v":"3.020"},{"t":"2019-04-15 01:12", "v":"3.062"},{"t":"2019-04-15 01:18",
"v":"3.101"},{"t":"2019-04-15 01:24", "v":"3.138"},{"t":"2019-04-15 01:30", "v":"3.172"},
{"t":"2019-04-15 01:36", "v":"3.204"},{"t":"2019-04-15 01:42", "v":"3.234"},{"t":"2019-04-15
01:48", "v":"3.261"},{"t":"2019-04-15 01:54", "v":"3.285"},{"t":"2019-04-15 02:00",
"v":"3.307"},{"t":"2019-04-15 02:06", "v":"3.327"},{"t":"2019-04-15 02:12", "v":"3.344"},
{"t":"2019-04-15 02:18", "v":"3.359"},{"t":"2019-04-15 02:24", "v":"3.372"},{"t":"2019-04-15
02:30", "v":"3.382"},{"t":"2019-04-15 02:36", "v":"3.390"},{"t":"2019-04-15 02:42",
"v":"3.396"},{"t":"2019-04-15 02:48", "v":"3.399"},{"t":"2019-04-15 02:54", "v":"3.401"},
{"t":"2019-04-15 03:00", "v":"3.400"},{"t":"2019-04-15 03:06", "v":"3.397"},{"t":"2019-04-15
03:12", "v":"3.392"},{"t":"2019-04-15 03:18", "v":"3.385"},{"t":"2019-04-15 03:24",
"v":"3.375"},{"t":"2019-04-15 03:30", "v":"3.364"},{"t":"2019-04-15 03:36", "v":"3.351"},
{"t":"2019-04-15 03:42", "v":"3.335"},{"t":"2019-04-15 03:48", "v":"3.318"},{"t":"2019-04-15
03:54", "v":"3.299"},{"t":"2019-04-15 04:00", "v":"3.277"},{"t":"2019-04-15 04:06",
"v":"3.254"},{"t":"2019-04-15 04:12", "v":"3.229"},{"t":"2019-04-15 04:18", "v":"3.202"},
{"t":"2019-04-15 04:24", "v":"3.173"},{"t":"2019-04-15 04:30", "v":"3.142"},{"t":"2019-04-15
04:36", "v":"3.109"},{"t":"2019-04-15 04:42", "v":"3.075"},{"t":"2019-04-15 04:48",
"v":"3.039"},{"t":"2019-04-15 04:54", "v":"3.002"},{"t":"2019-04-15 05:00", "v":"2.963"},
{"t":"2019-04-15 05:06", "v":"2.922"},{"t":"2019-04-15 05:12", "v":"2.880"},{"t":"2019-04-15
05:18", "v":"2.837"},{"t":"2019-04-15 05:24", "v":"2.793"},{"t":"2019-04-15 05:30",
"v":"2.747"},{"t":"2019-04-15 05:36", "v":"2.701"},{"t":"2019-04-15 05:42", "v":"2.654"},
{"t":"2019-04-15 05:48", "v":"2.607"},{"t":"2019-04-15 05:54", "v":"2.559"},{"t":"2019-04-15
06:00", "v":"2.510"},{"t":"2019-04-15 06:06", "v":"2.462"},{"t":"2019-04-15 06:12",
"v":"2.413"},{"t":"2019-04-15 06:18", "v":"2.364"},{"t":"2019-04-15 06:24", "v":"2.316"},
{"t":"2019-04-15 06:30", "v":"2.269"},{"t":"2019-04-15 06:36", "v":"2.221"},{"t":"2019-04-15
06:42", "v":"2.175"},{"t":"2019-04-15 06:48", "v":"2.130"},{"t":"2019-04-15 06:54",
"v":"2.086"},{"t":"2019-04-15 07:00", "v":"2.042"},{"t":"2019-04-15 07:06", "v":"2.001"},
{"t":"2019-04-15 07:12", "v":"1.961"},{"t":"2019-04-15 07:18", "v":"1.922"},{"t":"2019-04-15
07:24", "v":"1.885"},{"t":"2019-04-15 07:30", "v":"1.850"},{"t":"2019-04-15 07:36",
"v":"1.817"},{"t":"2019-04-15 07:42", "v":"1.786"},{"t":"2019-04-15 07:48", "v":"1.757"},
{"t":"2019-04-15 07:54", "v":"1.731"},{"t":"2019-04-15 08:00", "v":"1.706"},{"t":"2019-04-15
08:06", "v":"1.684"},{"t":"2019-04-15 08:12", "v":"1.664"},{"t":"2019-04-15 08:18",

More data like before, but now much more of it!

With data every 6 minutes, that's 10 per hour, or 240 for the entire day. And now each entry contains simply **t** and **v**. So for the given time **t**, we get the predicted water level **v**. We can use that to make a neat little plot of water level vs. time for the day. This will give us a graphical representation of the tidal activity.

High Low Tide Times Viewer

This version is just a basic tide time information viewer. It shows the times when the daily high and low times will occur. Typically there will be two of each per day.

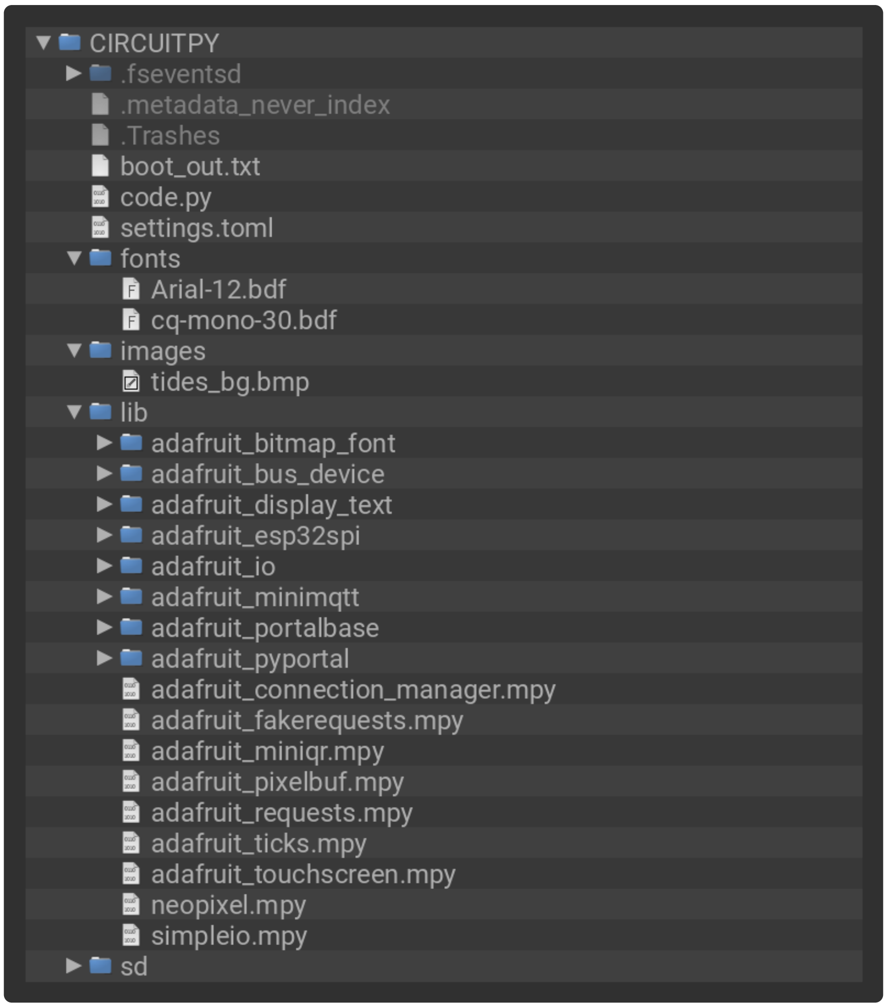
Let's get the code loaded up and running first, since that's more fun. We'll discuss how the parsing works after that.

Add 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_Tides**.

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



```
▼ CIRCUITPY
  ▶ .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    settings.toml
  ▼ fonts
      Arial-12.bdf
      cq-mono-30.bdf
  ▼ images
      tides_bg.bmp
  ▼ 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
```

Editing the Code

At the top of the code, find the line that sets the station ID and change it for your location:

```
STATION_ID = "9447130"   # tide location, find yours here: https://
tidesandcurrents.noaa.gov/
```

Note that it is entered as a string, not a number. So don't remove the quotation marks.

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

import time
import board
from adafruit_pyportal import PyPortal
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text.label import Label

# --| USER CONFIG |--------------------------
STATION_ID = (
    "9447130"  # tide location, find yours here: https://tidesandcurrents.noaa.gov/
)
HI_COLOR = 0x00FF00  # high tide times color
LO_COLOR = 0x11FFFF  # low tide times color
DATE_COLOR = 0xFFFFFF  # date and time color
```

```
# ---------------------------------------------

# pylint: disable=line-too-long
DATA_SOURCE = (
    "https://api.tidesandcurrents.noaa.gov/api/prod/datagetter?
date=today&product=predictions&datum=mllw&interval=hilo&format=json&units=metric&time_zone=lst
    + STATION_ID
)
DATA_LOCATION = ["predictions"]

# gotta have one of these
pyportal = PyPortal(status_neopixel=board.NEOPIXEL, default_bg="/images/
tides_bg.bmp")

# Connect to the internet and get local time
pyportal.get_local_time()

# Setup tide times font
tide_font = bitmap_font.load_font("/fonts/cq-mono-30.bdf")
tide_font.load_glyphs(b"1234567890:")

# Setup date and time font
date_font = bitmap_font.load_font("/fonts/Arial-12.bdf")
date_font.load_glyphs(b"1234567890-")

# Labels setup
HI_LABELS = [
    Label(tide_font, text="00:00", color=HI_COLOR, x=40, y=80),
    Label(tide_font, text="00:00", color=HI_COLOR, x=40, y=165),
]
LO_LABELS = [
    Label(tide_font, text="00:00", color=LO_COLOR, x=180, y=80),
    Label(tide_font, text="00:00", color=LO_COLOR, x=180, y=165),
]
DATE_LABEL = Label(date_font, text="0000-00-00 00:00:00", color=DATE_COLOR, x=75,
y=228)

# Add all the labels to the display
for label in HI_LABELS + LO_LABELS + [DATE_LABEL]:
    pyportal.graphics.splash.append(label)


def get_tide_info():
    """Fetch JSON tide time info and return it."""

    # Get raw JSON data
    raw_info = pyportal.network.fetch_data(DATA_SOURCE, json_path=DATA_LOCATION)

    # Return will be a dictionary of lists containing tide times
    new_tide_info = {"H": [], "L": []}

    # Parse out the tide time info
    for info in raw_info[0]:
        tide_type = info["type"]
        tide_time = info["t"].split(" ")[1]
        new_tide_info[tide_type].append(tide_time)

    return new_tide_info


def update_display(time_info, update_tides=False):
    """Update the display with current info."""

    # Tide time info
    if update_tides:
        # out with the old
        for tide_label in HI_LABELS + LO_LABELS:
            tide_label.text = ""
        # in with the new
        for i, hi_time in enumerate(tide_info["H"]):
            HI_LABELS[i].text = hi_time
        for i, lo_time in enumerate(tide_info["L"]):
            LO_LABELS[i].text = lo_time

    # Date and time
    DATE_LABEL.text = "{:04}-{:02}-{:02} {:02}:{:02}:{:02}".format(
        time_info.tm_year,
        time_info.tm_mon,
        time_info.tm_mday,
        time_info.tm_hour,
        time_info.tm_min,
        time_info.tm_sec,
    )


# First run update
tide_info = get_tide_info()
current_time = time.localtime()
update_display(current_time, True)
current_yday = current_time.tm_yday

# Update daily
while True:
    current_time = time.localtime()
    new_tides = False
    if current_time.tm_yday != current_yday:
        # new day, time to update
        tide_info = get_tide_info()
        new_tides = True
        current_yday = current_time.tm_yday
    update_display(current_time, new_tides)
    time.sleep(0.5)
```

How It Works

As mentioned previously, the JSON data we need to deal with looks like this:

```
{ "predictions" : [ {"t":"2019-04-15 02:55", "v":"3.401", "type":"H"},
{"t":"2019-04-15 09:01", "v":"1.586", "type":"L"},{"t":"2019-04-15 14:04",
"v":"2.780", "type":"H"},{"t":"2019-04-15 20:33", "v":"0.111", "type":"L"} ]}
```

All of the data mangling work is done in the function `get_tide_info()`. So let's walk through that.

When we setup our `PyPortal` object, we told it we want the information rooted at `predictions`. Once that is setup, along with providing the URL, to connect to the Internet and retrieve the data we simply call `fetch()` from the PyPortal library.

```
raw_info = pyportal.fetch()
```

Now we need to parse out the data in `raw_info`. We could come up with various ways of storing the results, but here we use a dictionary of lists. This will have an "H" entry which will contain the times for the high tides and an "L" entry which will contain the times for the low tides.

This choice of keys for the dictionary is not arbitrary. It was chosen to match the **type** values in the return results. That way we can use those directly as the dictionary keys.

We set this up initially with blank entries for the lists:

```
new_tide_info = {"H":[], "L":[]}
```

Then we loop over each entry and parse the data. We use `type` as is for the key. We don't want the date part of the **t** entry, so we split it on the space and save only the second part - the time.

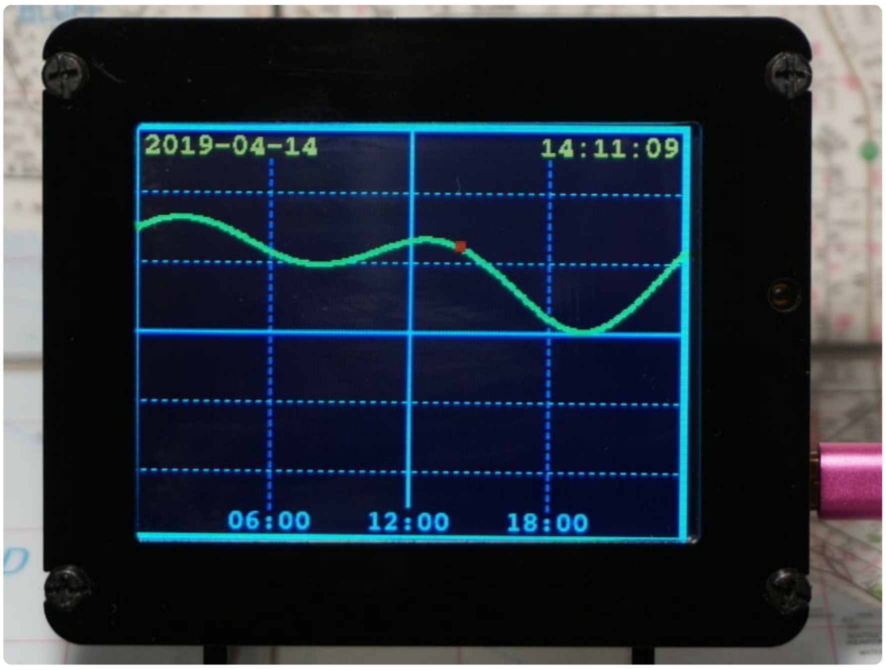
```
for info in raw_info:
    tide_type = info["type"]
    tide_time = info["t"].split(" ")[1]
    new_tide_info[tide_type].append(tide_time)
```

Done! Now `new_tide_info` has what we want, so we return it.

```
return new_tide_info
```

The rest of the code just displays these results. The very bottom of the code is a loop that runs for ever. Once a day it will go and fetch the new tide data.

Graphical Tide Level Viewer



This version is a little fancier. It provides a graphical plot of the predicted tide level over the 24 hour span of the current day.

It works pretty much the same as the simple version - go grab the data, parse it, display it. Here we just have more data to deal with and we display it a little fancier.

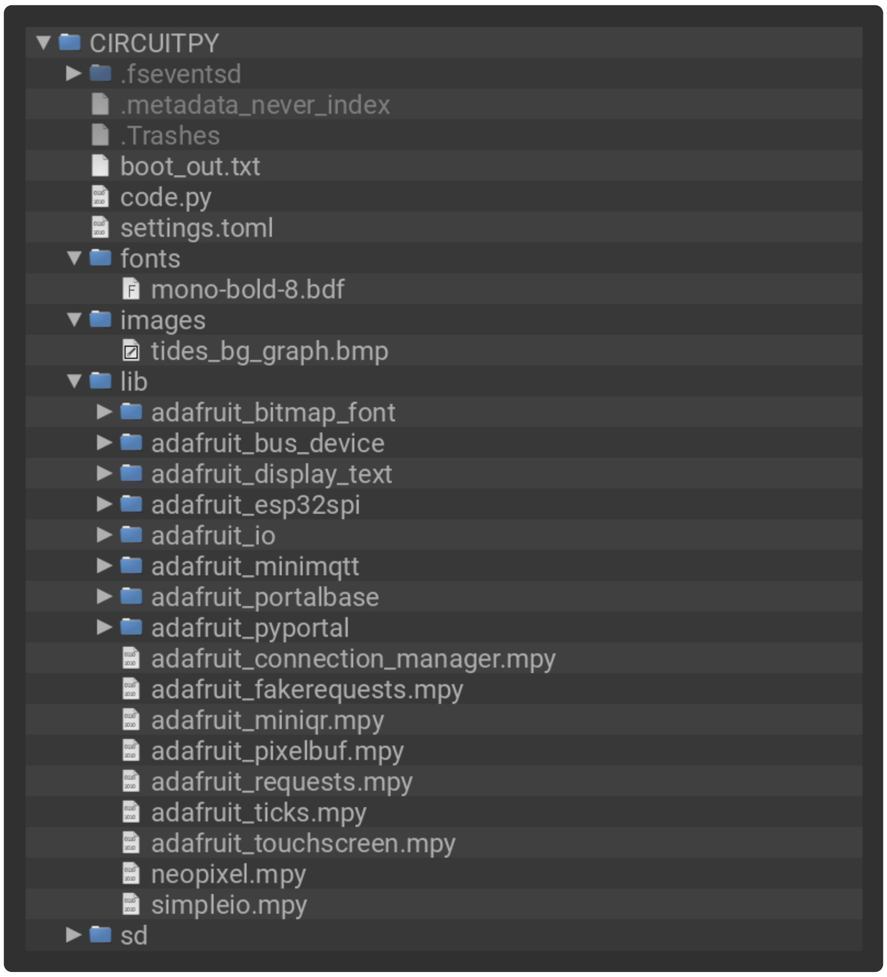
Let's get the code loaded up and running first...

Add 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_Tides**.

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



```
▼ CIRCUITPY
  ▶ .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    settings.toml
  ▼ fonts
      mono-bold-8.bdf
  ▼ images
      tides_bg_graph.bmp
  ▼ 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
```

Editing the Code

At the top of the code, find the line that sets the station ID and change it for your location:

```
STATION_ID = "9447130"   # tide location, find yours here: https://
tidesandcurrents.noaa.gov/
```

Note that it is entered as a string, not a number.

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

import time
import board
import displayio
from adafruit_pyportal import PyPortal
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text.label import Label

# --| USER CONFIG |--------------------------
STATION_ID = (
    "9447130"  # tide location, find yours here: https://tidesandcurrents.noaa.gov/
)
PLOT_SIZE = 2  # tide plot thickness
PLOT_COLOR = 0x00FF55  # tide plot color
MARK_SIZE = 6  # current time marker size
MARK_COLOR = 0xFF0000  # current time marker color
DATE_COLOR = 0xE0CD1A  # date text color
TIME_COLOR = 0xE0CD1A  # time text color
VSCALE = 20  # vertical plot scale
# ------------------------------------------

# pylint: disable=line-too-long
DATA_SOURCE = (
    "https://api.tidesandcurrents.noaa.gov/api/prod/datagetter?
date=today&product=predictions&datum=mllw&format=json&units=metric&time_zone=lst_ldt&station="
    + STATION_ID
)
DATA_LOCATION = ["predictions"]

WIDTH = board.DISPLAY.width
HEIGHT = board.DISPLAY.height

# gotta have one of these
pyportal = PyPortal(status_neopixel=board.NEOPIXEL, default_bg="/images/
tides_bg_graph.bmp")

# Connect to the internet and get local time
pyportal.get_local_time()

# Setup palette used for plot
palette = displayio.Palette(3)
palette[0] = 0x0
palette[1] = PLOT_COLOR
palette[2] = MARK_COLOR
palette.make_transparent(0)

# Setup tide plot bitmap
tide_plot = displayio.Bitmap(WIDTH, HEIGHT, 3)
pyportal.graphics.splash.append(displayio.TileGrid(tide_plot, pixel_shader=palette))

# Setup font used for date and time
date_font = bitmap_font.load_font("/fonts/mono-bold-8.bdf")
date_font.load_glyphs(b"1234567890-")

# Setup date label
date_label = Label(date_font, text="0000-00-00", color=DATE_COLOR, x=7, y=14)
pyportal.graphics.splash.append(date_label)

# Setup time label
time_label = Label(date_font, text="00:00:00", color=TIME_COLOR, x=234, y=14)
pyportal.graphics.splash.append(time_label)

# Setup current time marker
time_marker_bitmap = displayio.Bitmap(MARK_SIZE, MARK_SIZE, 3)
time_marker_bitmap.fill(2)
```

```
time_marker = displayio.TileGrid(
    time_marker_bitmap, pixel_shader=palette, x=-MARK_SIZE, y=-MARK_SIZE
)
pyportal.graphics.splash.append(time_marker)


def get_tide_data():
    """Fetch JSON tide data and return parsed results in a list."""

    # Get raw JSON data
    raw_data = pyportal.network.fetch_data(DATA_SOURCE, json_path=DATA_LOCATION)

    # Results will be stored in a list that is display WIDTH long
    new_tide_data = [None] * WIDTH

    # Convert raw data to display coordinates
    for data in raw_data[0]:
        _, t = data["t"].split(" ")  # date and time
        h, m = t.split(":")  # hours and minutes
        v = data["v"]  # water level
        x = round((WIDTH - 1) * (60 * float(h) + float(m)) / 1440)
        y = (HEIGHT // 2) - round(VSCALE * float(v))
        y = 0 if y < 0 else y
        y = HEIGHT - 1 if y >= HEIGHT else y
        new_tide_data[x] = y

    return new_tide_data


def draw_data_point(x, y, size=PLOT_SIZE, color=1):
    """Draw data point on to the tide plot bitmap at (x,y)."""
    if y is None:
        return
    offset = size // 2
    for xx in range(x - offset, x + offset + 1):
        for yy in range(y - offset, y + offset + 1):
            try:
                tide_plot[xx, yy] = color
            except IndexError:
                pass


def draw_time_marker(time_info):
    """Draw a marker on the tide plot for the current time."""
    h = time_info.tm_hour
    m = time_info.tm_min
    x = round((WIDTH - 1) * (60 * float(h) + float(m)) / 1440)
    y = tide_data[x]
    if y is not None:
        x -= MARK_SIZE // 2
        y -= MARK_SIZE // 2
        time_marker.x = x
        time_marker.y = y


def update_display(time_info, update_tides=False):
    """Update the display with current info."""

    # Tide data plot
    if update_tides:
        # out with the old
        for i in range(WIDTH * HEIGHT):
            tide_plot[i] = 0
        # in with the new
        for x in range(WIDTH):
            draw_data_point(x, tide_data[x])

    # Current location marker
    draw_time_marker(time_info)
    # Date and time
    date_label.text = "{:04}-{:02}-{:02}".format(
        time_info.tm_year, time_info.tm_mon, time_info.tm_mday
    )
    time_label.text = "{:02}:{:02}:{:02}".format(
        time_info.tm_hour, time_info.tm_min, time_info.tm_sec
    )


# First run update
tide_data = get_tide_data()
current_time = time.localtime()
update_display(current_time, True)
current_yday = current_time.tm_yday

# Run forever
while True:
    current_time = time.localtime()
    new_tides = False
    if current_time.tm_yday != current_yday:
        # new day, time to update
        tide_data = get_tide_data()
        new_tides = True
        current_yday = current_time.tm_yday
    update_display(current_time, new_tides)
    time.sleep(0.5)
```

How It Works

This code also does all the data mangling in a single function - `get_tide_data()`. The general idea is to take the time vs. tide level information and map it into the display's (x, y) coordinates. That way all we have to do is draw pixels at all the (x, y) locations and it will generate a plot.

But first, we go get the data. Same as before:

```
    raw_data = pyportal.fetch()
```

We store this in a list that has an entry for each x pixel on the display. The index of the list corresponds to the x pixel. The entry itself is the y value. So we know we need as many entries as the display has pixels across, i.e. its `WIDTH`:

```
    new_tide_data = [None]*WIDTH
```

And then we loop and parse the data again. We split out the date and time. We further split out the time into hours and minutes, so we can do some math on what will become our x value. The tide level v is what will become our y value. After some math, the results are stored in our list and finally returned.

```
    for data in raw_data:
        _, t = data["t"].split(" ") # date and time
        h, m = t.split(":")         # hours and minutes
        v = data["v"]               # water level
        x = round( (WIDTH - 1) * (60 * float(h) + float(m)) / 1440 )
        y = (HEIGHT // 2) - round(VSCALE * float(v))
        y = 0 if y &lt; 0 else y
        y = HEIGHT-1 if y &gt;= HEIGHT else y
        new_tide_data[x] = y

    return new_tide_data
```

Let's talk about that math a little more. First, the math for the horizontal or x position:

```
        x = round( (WIDTH - 1) * (60 * float(h) + float(m)) / 1440 )
```

The display is **WIDTH** pixels across. A day has 24 hours which is 24*60=**1440** total minutes. So there are 1440 minutes per `WIDTH` pixels. To get the **x** coordinate for any given minute, we just multiply by that ratio:

x = minutes * (WIDTH / 1440)

That's all that's happening. There's a little more in the code to compute total minutes for hours+minutes time data. And the **- 1** is to deal with the 0 based indexing of the x pixels - they start at 0, not 1.

Now the math for the vertical or y position:

```
        y = (HEIGHT // 2) - round(VSCALE * float(v))
```

We want the vertical plot to vary above/below the middle of the display. So we just compute the middle of display with `HEIGHT / 2`. From the this we subtract the tide level value **v**.

y = (HEIGHT / 2) - v

And that's pretty much it.

The extra things being done are to use // instead of / to force integer math, since the **y** pixel needs to be an integer. We also scale the **v** value by multiplying by `VSCALE`, which is just an arbitrary value to make the plot spread out more. We wrap that in `round()` to also make sure it ends up being an integer.

The other two lines just make sure the bounds are with the actual display values of `0` to `HEIGHT-1`.

What's That Little Red Mark?

That's the location of the current time on the tide plot. It should move along the plot as the day progresses. If it doesn't show up when you initially run the code, wait a bit. It should eventually show up.

Customizing

Simple Stuff

At the top of each version of the code there is a section with some user changeable settings. It looks like this:

```
#--| USER CONFIG |--------------------------
STATION_ID = "9447130"    # tide location, find yours here: https://
tidesandcurrents.noaa.gov/
HI_COLOR   = 0x00FF00     # high tide times color
LO_COLOR   = 0x11FFFF     # low tide times color
DATE_COLOR = 0xFFFFFF     # date and time color
#-------------------------------------------
```

The most important is the station ID, which sets the location. But there are a few others as well. You can change the color used for the text labels, for example.

Not So Simple Stuff

Most of the graphics used for the tides display were created ahead of time and saved as a BMP image file. These were then simply set to be the background image for the PyPortal and the tide information was added on top of that.

What if you want to change the background image for the tides time display? Or you don't like the blue tinted graph and want to change that? To do so, you will need the source files used for generating the BMP images. These are provided below in SVG format. You can use something like Inkscape (https://adafru.it/oEf) to edit these and output new BMP files.

tides_bg.svg

https://adafru.it/ExR

tides_bg_graph.svg

https://adafru.it/ExS

12 Hour Time Format

Not a fan of the 24 hour time format? Wish it could be 12 hour AM/PM instead? The current version of the code does not support this. Only 24 hour time format. This is mainly due to wanting to keep things simple. The times shown are simply what is returned from the NOAA web service, which are strings in a 24 hour time format.

It wouldn't be too difficult to add additional parsing, logic, and math to add this. Most of the work would be in adding some form of AM/PM indication to the display. Doing so would be a fun exercise in learning more about Python and CircuitPython.