PyPortal US Election Calendar

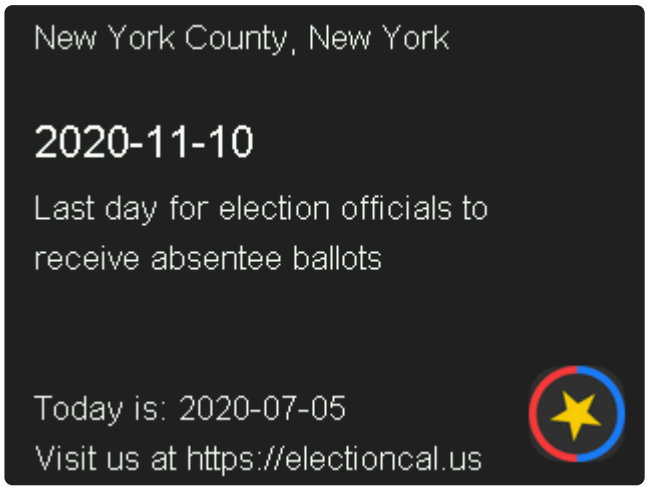
Created by Álvaro Figueroa

https://learn.adafruit.com/pyportal-electioncal-us

Last updated on 2024-04-09 10:48:10 AM EDT

Table of Contents

Overview

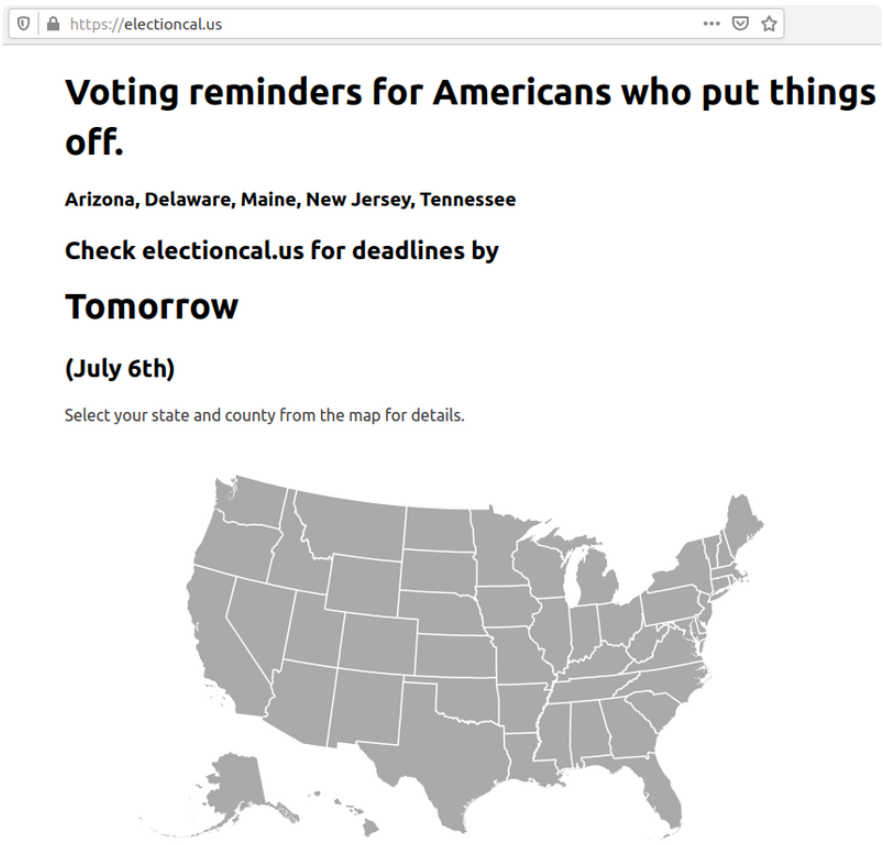


Voting is important to make the voice of the people as loud as possible.

But, it is also very complicated to remember all of the involved dates, so much so that you are almost certain to miss a registration date.

To try to make it easier for us with forgetful minds, a community driven effort has been gathering all of the hard to find information and dates, and compiling it on the Electioncal Website (https://adafru.it/M7F).

It's designed to share this information as an iCalendar url, which is super easy to add to your favorite desktop software or to your mobile phone.



But sometimes we need a little visual reminder, and a beautiful way to do it, is with a PyPortal.

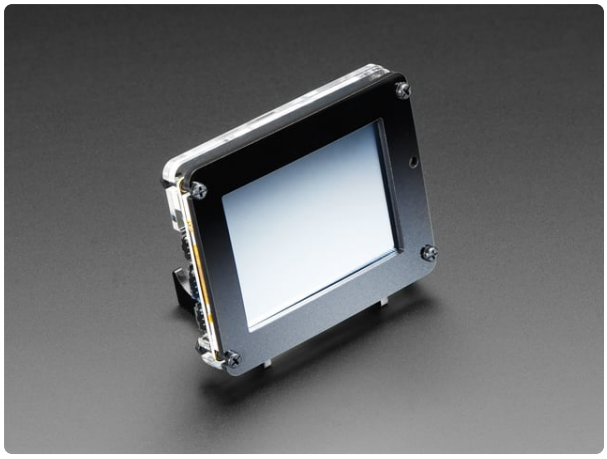
We are going to use CircuitPython to connect to the Internet, download and process the data, and then display it in a nice and clean way, so we can always keep the pulse on our Election Calendar.

Parts

Adafruit PyPortal - CircuitPython Powered Internet Display

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

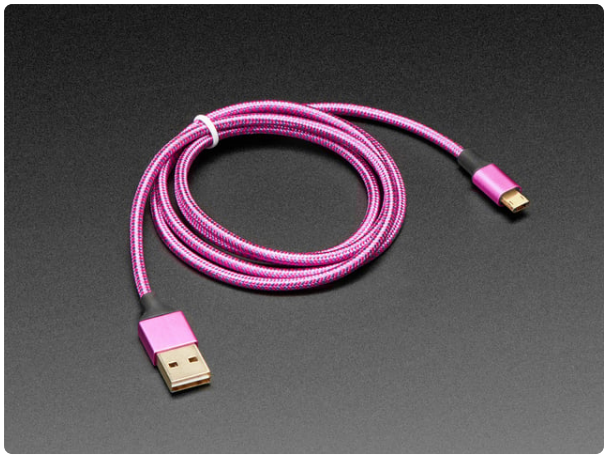
https://www.adafruit.com/product/4116



Adafruit PyPortal 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



Fully Reversible Pink/Purple USB A to micro B Cable - 1m long

This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also fully reversible! That's right, you will save seconds a day by...

https://www.adafruit.com/product/4111

5V 1A (1000mA) USB port power supply - UL Listed

Need a USB jack for charging or powering a project, but don't want to lug around a computer? This switching supply gives a clean regulated output at up to 1000mA! 110 or 240 input,...

https://www.adafruit.com/product/501

Install CircuitPython

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

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

Set up CircuitPython Quick Start!

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

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

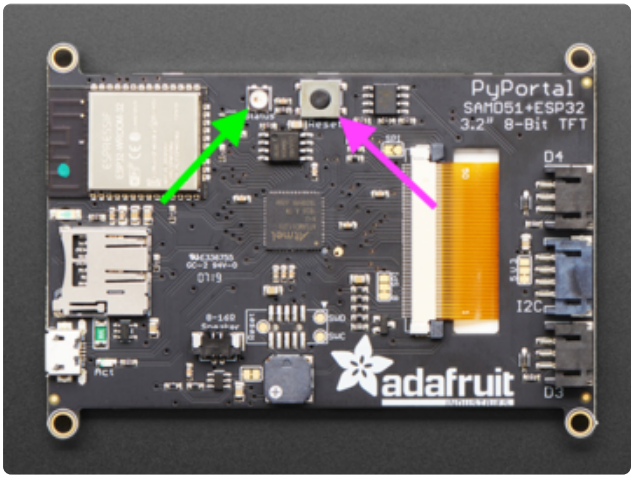
https://adafru.it/Egk

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

https://adafru.it/HFd

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

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

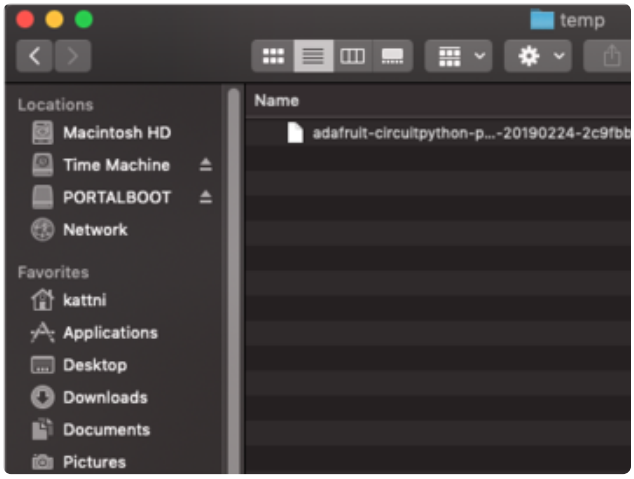


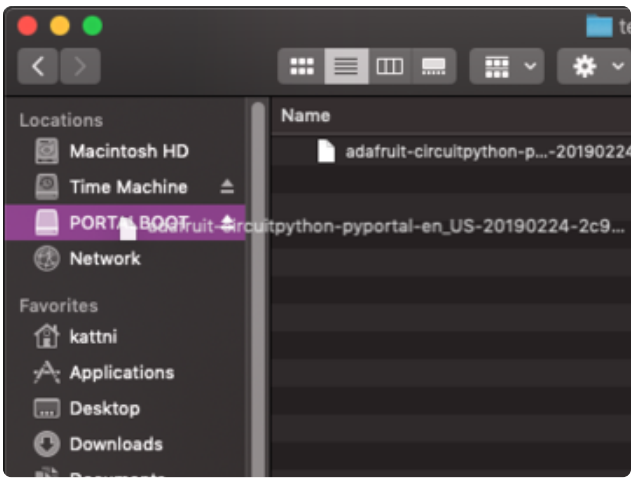
Plug your PyPortal into your computer using a known-good USB cable.

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

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

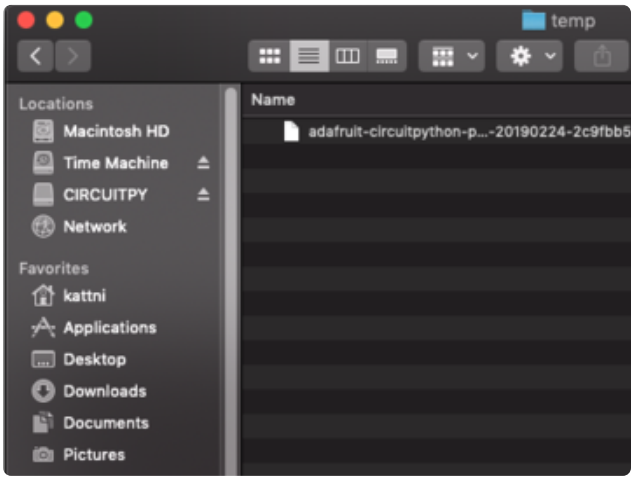
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!





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

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



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

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

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

PyPortal Default Files

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

PyPortal Default Files

https://adafru.it/UF-

PyPortal Pynt Default Files

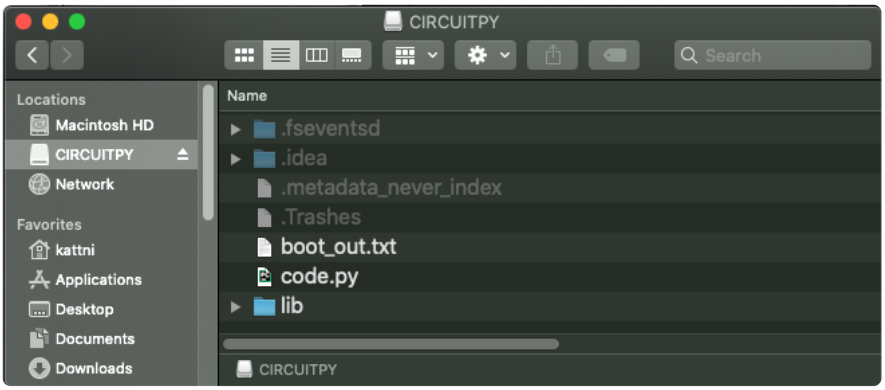
https://adafru.it/UGa

CircuitPython Libraries

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. Visit https://circuitpython.org/downloads to download the latest version of CircuitPython for your board. You must download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then visit https://circuitpython.org/libraries to download the latest Library Bundle.

Each CircuitPython program you run needs to have a lot of information to work. The reason CircuitPython is so simple to use is that most of that information is stored in other files and works in the background. These files are called libraries. Some of them are built into CircuitPython. Others are stored on your **CIRCUITPY** drive in a folder called **lib**. Part of what makes CircuitPython so great is its ability to store code separately from the firmware itself. Storing code separately from the firmware makes it easier to update both the code you write and the libraries you depend.

Your board may ship with a **lib** folder already, it's in the base directory of the drive. If not, simply create the folder yourself. When you first install CircuitPython, an empty **lib** directory will be created for you.



CircuitPython libraries work in the same way as regular Python modules so the Python docs (https://adafru.it/rar) are an excellent reference for how it all should work. In Python terms, you can place our library files in the **lib** directory because it's part of the Python path by default.

One downside of this approach of separate libraries is that they are not built in. To use them, one needs to copy them to the **CIRCUITPY** drive before they can be used. Fortunately, there is a library bundle.

The bundle and the library releases on GitHub also feature optimized versions of the libraries with the **.mpy** file extension. These files take less space on the drive and have a smaller memory footprint as they are loaded.

Due to the regular updates and space constraints, Adafruit does not ship boards with the entire bundle. Therefore, you will need to load the libraries you need when you begin working with your board. You can find example code in the guides for your board that depends on external libraries.

Either way, as you start to explore CircuitPython, you'll want to know how to get libraries on board.

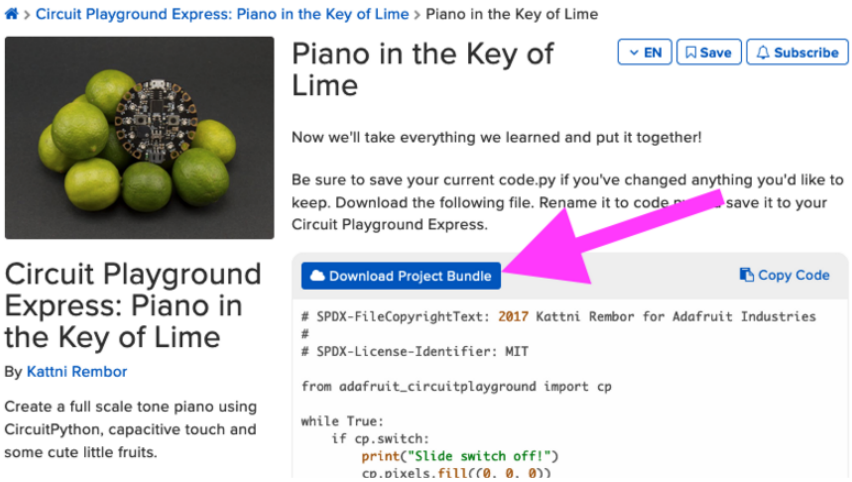
The Adafruit Learn Guide Project Bundle

The quickest and easiest way to get going with a project from the Adafruit Learn System is by utilising the Project Bundle. Most guides now have a **Download Project Bundle** button available at the top of the full code example embed. This button downloads all the necessary files, including images, etc., to get the guide project up

and running. Simply click, open the resulting zip, copy over the right files, and you're good to go!

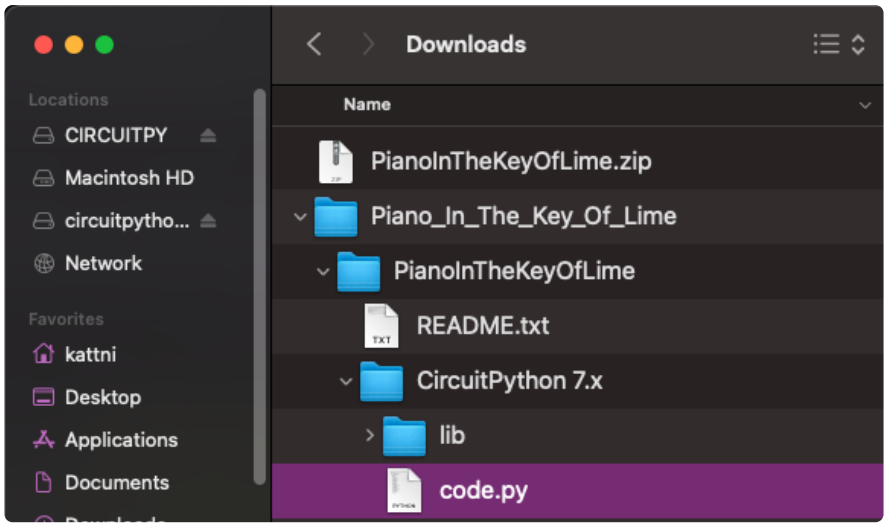
The first step is to find the Download Project Bundle button in the guide you're working on.

The Download Project Bundle button is only available on full demo code embedded from GitHub in a Learn guide. Code snippets will NOT have the button available.



When you copy the contents of the Project Bundle to your CIRCUITPY drive, it will replace all the existing content! If you don't want to lose anything, ensure you copy your current code to your computer before you copy over the new Project Bundle content!

The Download Project Bundle button downloads a zip file. This zip contains a series of directories, nested within which is the **code.py**, any applicable assets like images or audio, and the **lib/** folder containing all the necessary libraries. The following zip was downloaded from the Piano in the Key of Lime guide.



The Piano in the Key of Lime guide was chosen as an example. That guide is specific to Circuit Playground Express, and cannot be used on all boards. Do not expect to download that exact bundle and have it work on your non-CPX microcontroller.

When you open the zip, you'll find some nested directories. Navigate through them until you find what you need. You'll eventually find a directory for your CircuitPython version (in this case, 7.x). In the version directory, you'll find the file and directory you need: **code.py** and **lib/**. Once you find the content you need, you can copy it all over to your **CIRCUITPY** drive, replacing any files already on the drive with the files from the freshly downloaded zip.

In some cases, there will be other files such as audio or images in the same directory as code.py and lib/. Make sure you include all the files when you copy things over!

Once you copy over all the relevant files, the project should begin running! If you find that the project is not running as expected, make sure you've copied ALL of the project files onto your microcontroller board.

That's all there is to using the Project Bundle!

The Adafruit CircuitPython Library Bundle

Adafruit provides CircuitPython libraries for much of the hardware they provide, including sensors, breakouts and more. To eliminate the need for searching for each library individually, the libraries are available together in the Adafruit CircuitPython Library Bundle. The bundle contains all the files needed to use each library.

Downloading the Adafruit CircuitPython Library Bundle

You can download the latest Adafruit CircuitPython Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

Match up the bundle version with the version of CircuitPython you are running. For example, you would download the 6.x library bundle if you're running any version of CircuitPython 6, or the 7.x library bundle if you're running any version of CircuitPython 7, etc. If you mix libraries with major CircuitPython versions, you will get incompatible mpy errors due to changes in library interfaces possible during major version changes.

Click to visit circuitpython.org for the latest Adafruit CircuitPython Library Bundle

https://adafru.it/ENC

Download the bundle version that matches your CircuitPython firmware version. If you don't know the version, check the version info in **boot_out.txt** file on the **CIRCUITPY** drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.

There's also a **py** bundle which contains the uncompressed python files, you probably don't want that unless you are doing advanced work on libraries.

The CircuitPython Community Library Bundle

The CircuitPython Community Library Bundle is made up of libraries written and provided by members of the CircuitPython community. These libraries are often written when community members encountered hardware not supported in the Adafruit Bundle, or to support a personal project. The authors all chose to submit these libraries to the Community Bundle make them available to the community.

These libraries are maintained by their authors and are not supported by Adafruit. As you would with any library, if you run into problems, feel free to file an issue on the GitHub repo for the library. Bear in mind, though, that most of these libraries are supported by a single person and you should be patient about receiving a response.

Remember, these folks are not paid by Adafruit, and are volunteering their personal time when possible to provide support.

Downloading the CircuitPython Community Library Bundle

You can download the latest CircuitPython Community Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

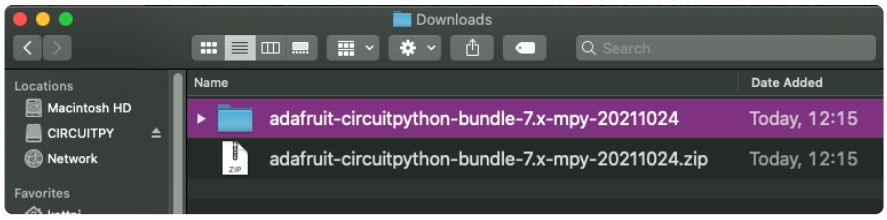
Click for the latest CircuitPython Community Library Bundle release

https://adafru.it/VCn

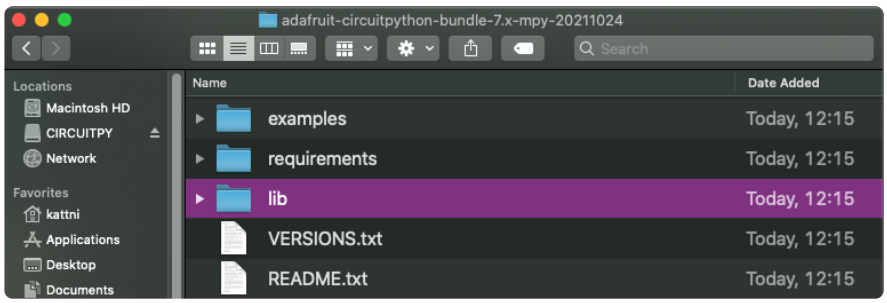
The link takes you to the latest release of the CircuitPython Community Library Bundle on GitHub. There are multiple versions of the bundle available. **Download the bundle version that matches your CircuitPython firmware version.** If you don't know the version, check the version info in **boot_out.txt** file on the **CIRCUITPY** drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.

Understanding the Bundle

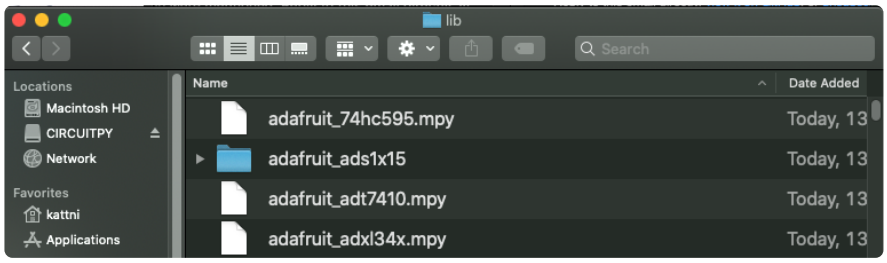
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



Open the bundle folder. Inside you'll find two information files, and two folders. One folder is the lib bundle, and the other folder is the examples bundle.



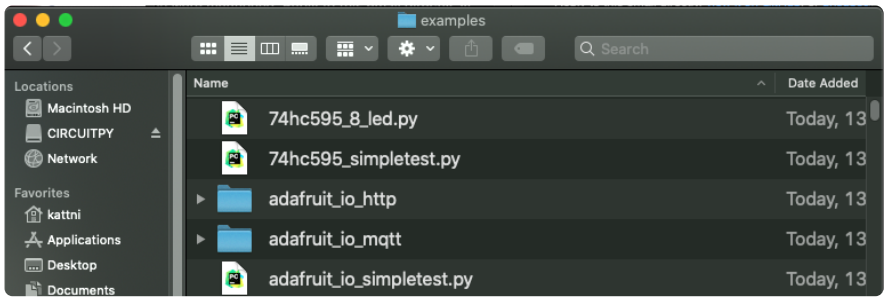
Now open the lib folder. When you open the folder, you'll see a large number of **.mpy** files, and folders.



Example Files

All example files from each library are now included in the bundles in an **examples** directory (as seen above), as well as an examples-only bundle. These are included for two main reasons:

- Allow for quick testing of devices.
- Provide an example base of code, that is easily built upon for individualized purposes.



Copying Libraries to Your Board

First open the **lib** folder on your **CIRCUITPY** drive. Then, open the **lib** folder you extracted from the downloaded zip. Inside you'll find a number of folders and **.mpy** files. Find the library you'd like to use, and copy it to the **lib** folder on **CIRCUITPY**.

If the library is a directory with multiple **.mpy** files in it, be sure to **copy the entire folder to CIRCUITPY/lib**.

This also applies to example files. Open the **examples** folder you extracted from the downloaded zip, and copy the applicable file to your **CIRCUITPY** drive. Then, rename it to **code.py** to run it.

If a library has multiple .mpy files contained in a folder, be sure to copy the entire folder to CIRCUITPY/lib.

Understanding Which Libraries to Install

You now know how to load libraries on to your CircuitPython-compatible microcontroller board. You may now be wondering, how do you know which libraries you need to install? Unfortunately, it's not always straightforward. Fortunately, there is an obvious place to start, and a relatively simple way to figure out the rest. First up: the best place to start.

When you look at most CircuitPython examples, you'll see they begin with one or more `import` statements. These typically look like the following:

- `import library_or_module`

However, `import` statements can also sometimes look like the following:

- `from library_or_module import name`
- `from library_or_module.subpackage import name`
- `from library_or_module import name as local_name`

They can also have more complicated formats, such as including a `try` / `except` block, etc.

The important thing to know is that **an `import` statement will always include the name of the module or library that you're importing**.

Therefore, the best place to start is by reading through the `import` statements.

Here is an example import list for you to work with in this section. There is no setup or other code shown here, as the purpose of this section involves only the import list.

```
import time
import board
import neopixel
import adafruit_lis3dh
import usb_hid
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode
```

Keep in mind, not all imported items are libraries. Some of them are almost always built-in CircuitPython modules. How do you know the difference? Time to visit the REPL.

In the Interacting with the REPL section (https://adafru.it/Awz) on The REPL page (https://adafru.it/Awz) in this guide, the `help("modules")` command is discussed. This command provides a list of all of the built-in modules available in CircuitPython for your board. So, if you connect to the serial console on your board, and enter the REPL, you can run `help("modules")` to see what modules are available for your board. Then, as you read through the `import` statements, you can, for the purposes of figuring out which libraries to load, ignore the statement that import modules.

The following is the list of modules built into CircuitPython for the Feather RP2040. Your list may look similar or be anything down to a significant subset of this list for smaller boards.

```
>>> help("modules")
__main__          board             micropython       storage
_bleio            builtins          msgpack           struct
adafruit_bus_device                 busio             neopixel_write    supervisor
adafruit_pixelbuf collections       onewireio         synthio
aesio             countio           os                sys
alarm             digitalio         paralleldisplay   terminalio
analogio          displayio         pulseio           time
array             errno             pwmio             touchio
atexit            fontio            qrio              traceback
audiobusio        framebufferio     rainbowio         ulab
audiocore         gc                random            usb_cdc
audiomixer        getpass           re                usb_hid
audiomp3          imagecapture      rgbmatrix         usb_midi
audiopwmio        io                rotaryio          vectorio
binascii          json              rp2pio            watchdog
bitbangio         keypad            rtc
bitmaptools       math              sdcardio
bitops            microcontroller   sharpdisplay
```

Now that you know what you're looking for, it's time to read through the import statements. The first two, `time` and `board`, are on the modules list above, so they're built-in.

The next one, `neopixel`, is not on the module list. That means it's your first library! So, you would head over to the bundle zip you downloaded, and search for **neopixel**. There is a **neopixel.mpy** file in the bundle zip. Copy it over to the **lib** folder on your **CIRCUITPY** drive. The following one, `adafruit_lis3dh`, is also not on the module list. Follow the same process for **adafruit_lis3dh**, where you'll find **adafruit_lis3dh.mpy**, and copy that over.

The fifth one is `usb_hid`, and it is in the modules list, so it is built in. Often all of the built-in modules come first in the import list, but sometimes they don't! Don't assume

that everything after the first library is also a library, and verify each import with the modules list to be sure. Otherwise, you'll search the bundle and come up empty!

The final two imports are not as clear. Remember, when `import` statements are formatted like this, the first thing after the `from` is the library name. In this case, the library name is `adafruit_hid`. A search of the bundle will find an **adafruit_hid folder**. When a library is a folder, you must copy the **entire folder and its contents as it is in the bundle** to the **lib** folder on your **CIRCUITPY** drive. In this case, you would copy the entire **adafruit_hid** folder to your **CIRCUITPY/lib** folder.

Notice that there are two imports that begin with `adafruit_hid`. Sometimes you will need to import more than one thing from the same library. Regardless of how many times you import the same library, you only need to load the library by copying over the **adafruit_hid** folder once.

That is how you can use your example code to figure out what libraries to load on your CircuitPython-compatible board!

There are cases, however, where libraries require other libraries internally. The internally required library is called a dependency. In the event of library dependencies, the easiest way to figure out what other libraries are required is to connect to the serial console and follow along with the `ImportError` printed there. The following is a very simple example of an `ImportError`, but the concept is the same for any missing library.

Example: `ImportError` Due to Missing Library

If you choose to load libraries as you need them, or you're starting fresh with an existing example, you may end up with code that tries to use a library you haven't yet loaded. This section will demonstrate what happens when you try to utilise a library that you don't have loaded on your board, and cover the steps required to resolve the issue.

This demonstration will only return an error if you do not have the required library loaded into the **lib** folder on your **CIRCUITPY** drive.

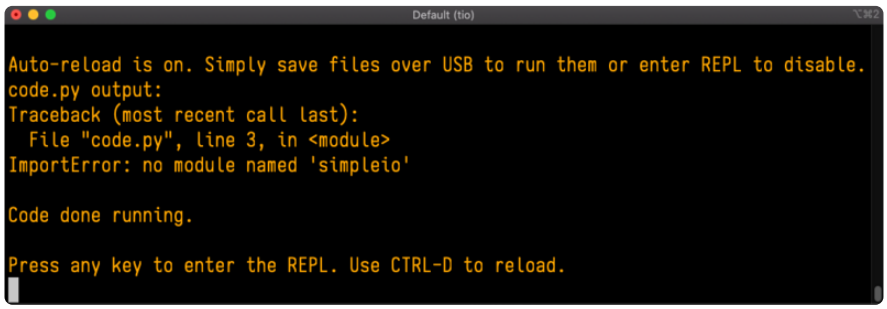
Let's use a modified version of the Blink example.

```
import board
import time
import simpleio
```

```
led = simpleio.DigitalOut(board.LED)

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

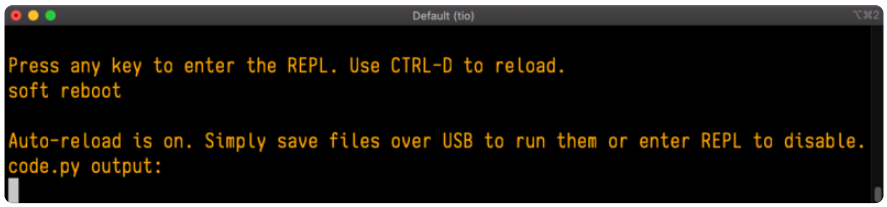
Save this file. Nothing happens to your board. Let's check the serial console to see what's going on.



You have an `ImportError`. It says there is `no module named 'simpleio'`. That's the one you just included in your code!

Click the link above to download the correct bundle. Extract the lib folder from the downloaded bundle file. Scroll down to find **simpleio.mpy**. This is the library file you're looking for! Follow the steps above to load an individual library file.

The LED starts blinking again! Let's check the serial console.



No errors! Excellent. You've successfully resolved an `ImportError`!

If you run into this error in the future, follow along with the steps above and choose the library that matches the one you're missing.

Library Install on Non-Express Boards

If you have an M0 non-Express board such as Trinket M0, Gemma M0, QT Py M0, or one of the M0 Trinkeys, you'll want to follow the same steps in the example above to install libraries as you need them. Remember, you don't need to wait for an `ImportError` if you know what library you added to your code. Open the library

bundle you downloaded, find the library you need, and drag it to the **lib** folder on your **CIRCUITPY** drive.

You can still end up running out of space on your M0 non-Express board even if you only load libraries as you need them. There are a number of steps you can use to try to resolve this issue. You'll find suggestions on the Troubleshooting page (https://adafru.it/Den).

Updating CircuitPython Libraries and Examples

Libraries and examples are updated from time to time, and it's important to update the files you have on your **CIRCUITPY** drive.

To update a single library or example, follow the same steps above. When you drag the library file to your lib folder, it will ask if you want to replace it. Say yes. That's it!

A new library bundle is released every time there's an update to a library. Updates include things like bug fixes and new features. It's important to check in every so often to see if the libraries you're using have been updated.

CircUp CLI Tool

There is a command line interface (CLI) utility called CircUp (https://adafru.it/Tfi) that can be used to easily install and update libraries on your device. Follow the directions on the install page within the CircUp learn guide (https://adafru.it/-Ad). Once you've got it installed you run the command `circup update` in a terminal to interactively update all libraries on the connected CircuitPython device. See the usage page in the CircUp guide (https://adafru.it/-Ah) for a full list of functionality

---

Code PyPortal with CircuitPython

Location

In the **code.py** file (which you will have renamed from **electioncal.py**), you can change the location for which you want to display the election data in this lines:

```
# Change this to your state and county, replacing spaces for underscores and in
lowercase
```

```
STATE="new_york"
COUNTY="new_york"
```

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).

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

Once you have logged into your account, there are two pieces of information you'll need to place in your **secrets.py** file: `Adafruit IO username`, and `Adafruit IO key`. Head to io.adafruit.com (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 secrets.py file like this:

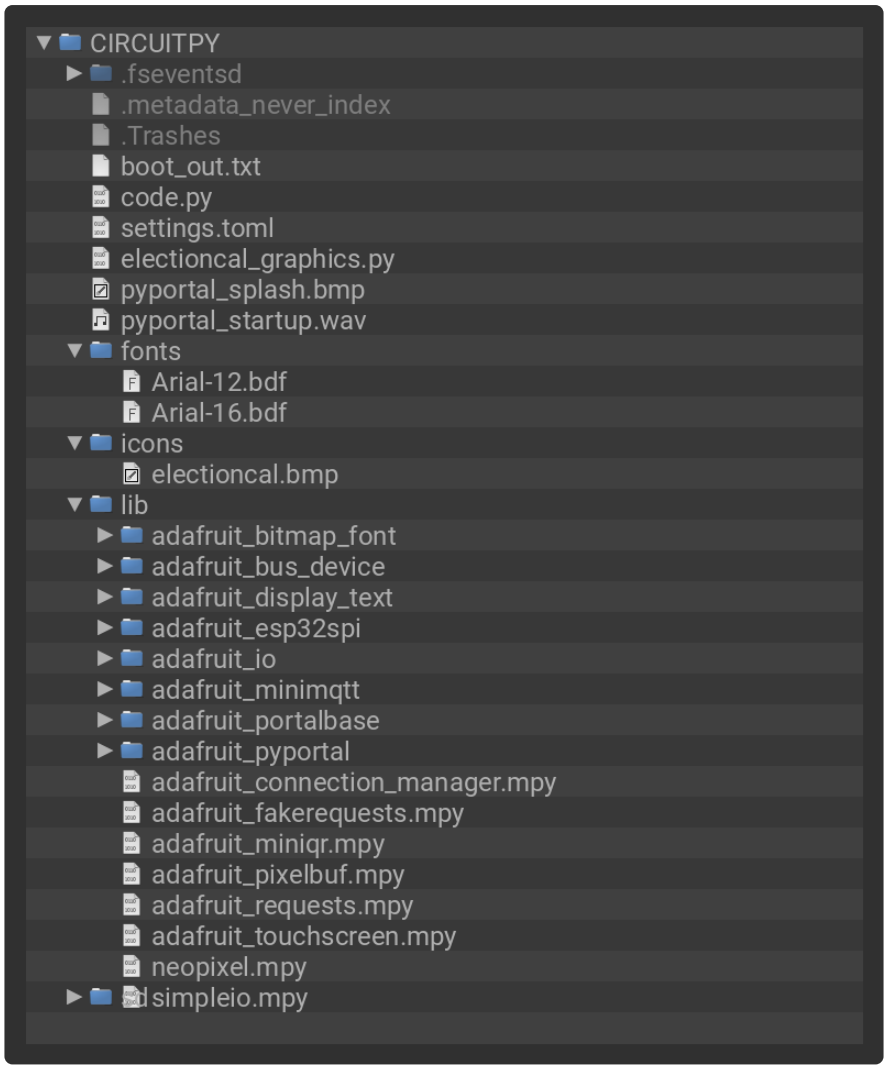
```
secrets = {
    'ssid' : 'your_wifi_ssid',
    'password' : 'your_wifi_password',
    'aio_username' : '_your_aio_username_',
    'aio_key' : '_your_big_huge_super_long_aio_key_'
}
```

Installing Project Code

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 **PyPortal_Electioncal_US/** 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: 2020 Alvaro Figueroa for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import sys
import time
import board
from adafruit_pyportal import PyPortal
cwd = ("/"+__file__).rsplit('/', 1)[0] # the current working directory (where this
file is)
sys.path.append(cwd)
import electioncal_graphics  # pylint: disable=wrong-import-position

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

# Change this to your state and county, replacing spaces for underscores and in
lowercase
STATE="new_york"
COUNTY="new_york"

DATA_SOURCE = "https://electioncal.us/en/" + STATE +"/" + COUNTY + "/voter.json"
DATA_LOCATION = []

# Initialize the pyportal object and let us know what data to fetch and where
```

```
# to display it
pyportal = PyPortal(url=DATA_SOURCE,
                    json_path=DATA_LOCATION,
                    status_neopixel=board.NEOPIXEL,
                    default_bg=0x000000)


gfx = electioncal_graphics.Electioncal_Graphics(pyportal.splash, am_pm=True)
display_refresh = None
while True:
    # only query the online time once per hour (and on first run)
    if (not display_refresh) or (time.monotonic() - display_refresh) > 3600:
        try:
            print("Getting time from internet!")
            pyportal.get_local_time()
            display_refresh = time.monotonic()
        except RuntimeError as e:
            print("Some error occured, retrying! -", e)
            continue

        try:
            value = pyportal.fetch()
            #print("Response is", value)
            gfx.load_data(value)
        except RuntimeError as e:
            print("Some error occured, retrying! -", e)
            continue
    try:
        gfx.elections_cycle()
    except RuntimeError as e:
        print("Some error ocurred, retrying! -", e)
        continue
```

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 Electioncal US has a few steps it takes to provide you with the information desired. It has a boot-up screen, a nice simple background, and multiple fonts for displaying the election dates.

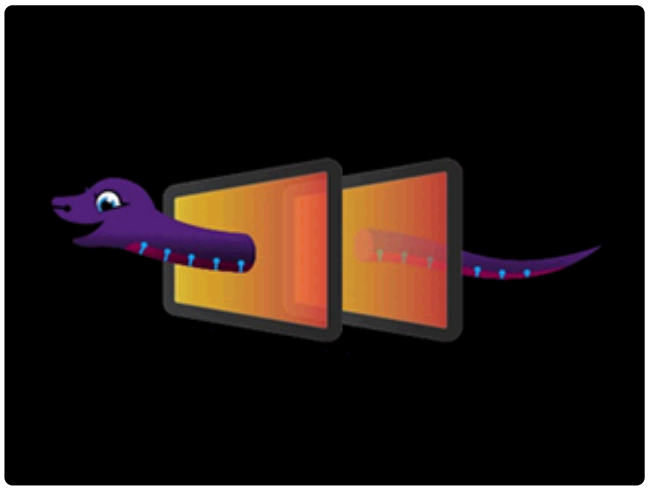
PyPortal Constructor

When setting up the `pyportal` constructor, we are providing it with these things:

- `url` to query
- `json_path` to traverse and find the key:value pairs we need
- `default_bg` default background color

Background

First, it displays a bitmap graphic as the screen's startup background while getting ready to download the Election dates. This is a 320 x 240 pixel RGB 16-bit raster graphic in **.bmp** format.



Time

Next, the program connects through the WiFi to get the local time via the adafruit.io server, for which we will display today's date at the bottom of the screen, to give you better context of the election dates.

API Query and JSON

At the top of this page, you configured:

`DATA_SOURCE = "https://electioncal.us/en/" + STATE +"/" + COUNTY + "/voter.json"`

This, with the STATE and COUNTY you defined, will turn into something like (New York County, New York State in this example):

`https://electioncal.us/en/new_york/new_york/voter.json`

When this query is complete, it returns a JSON file that looks like this:

```
{
  "version": 0,
  "name": null,
  "description": null,
  "dates":
```

```
    [
      {
        "date": "2020-04-28",
        "name": "Presidential Primary",
        "original_date": "2020-04-28",
        "state": "New York",
        "county": "New York County",
        "key": "20200428",
        "type": "election"
      },
      {
        "state": "New York",
        "county": "New York County",
        "name": "Mail voter registration",
        "type": "reminder",
        "election_key": "20200623",
        "subtype": "registration.received_by",
        "date": "2020-05-26",
        "postmark_too_late": true,
        "deadline_date": "2020-06-03",
        "explanation": "Voter registration must be received by June 3rd."
      }
    ]
}
```


Here is the same file beautified with the Firefox browser's built in tools (You can also use online code "beautifiers" such as https://codebeautify.org/jsonviewer (https://adafru.it/Eb5) or http://jsonviewer.stack.hu (https://adafru.it/Eb6)) :

Fetch

With the PyPortal set up, we can then use `pyportal.fetch()` and then from the screen object we call `load_data()` to load the JSON data to a way that is easy to pass through all the dates.

All of the heavy lifting of parsing that data and displaying it as text or bitmaps is done in the **electioncal_graphics.py** code.

JSON Traversal

The JSON file is formatted in a way that makes it easy to traverse the hierarchy and parse the data. In it, you'll see keys, such as `state`, `county`, `name`, `type` and `date`, and their respective values. So, here are some **key : value** pairs we care about:

- `state: "New York"`
- `county: "New York County"`
- `name: "Presidential Primary"`
- `date: "2020-04-28"`

In order to fetch this data from the file, we need to be able to describe their locations in the file hierarchically. This is helpful, for example, in differentiating between the `'name'` value that you have in the Presidential Primary and the `'name'` of the Mail voter registration. To avoid name clashing, we rely on JSON traversal.

In the **electioncal_graphics.py** file, you'll see how this is done. For example, the first date is found in this hierarchy of the JSON file: `["dates"][0]["date"]`.

In this case the number `0` is quite important, as it has the sub-tree that is relevant for the first, date, but if you change it to a number such as `1` or `2`, you will print the second and third date. Please note that the counter starts at zero.

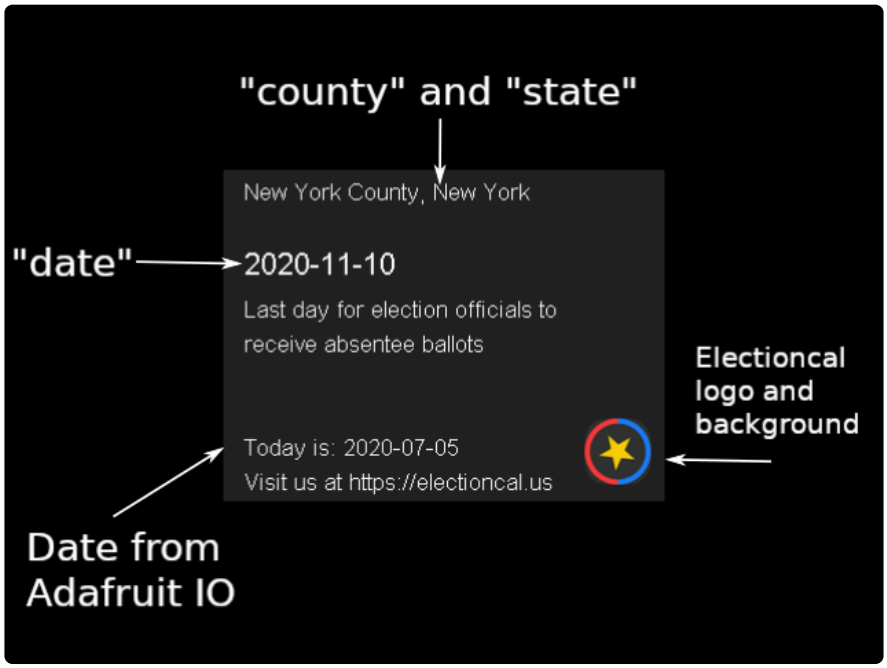
Depending of the STATE and COUNTY you configured, you will have several election dates, so the program then tries to replace this number by `[i]`, so that we can display not one date but many, and cycle them through the screen.

Font

The data is displayed as text created with bitmapped fonts to overlay on top of the background. The fonts used here are bitmap fonts made from the Arial typeface. You can learn more about converting type in this guide (https://adafru.it/E7E).

Graphics

Let's have a look at how the **electioncal_graphics.py** code places the elements on screen. Below, we can see the icon and text that are displayed. The items in quotes are the key names from the JSON file, and their values are what we see displayed using the CircuitPython `label` from the `displayio` library.



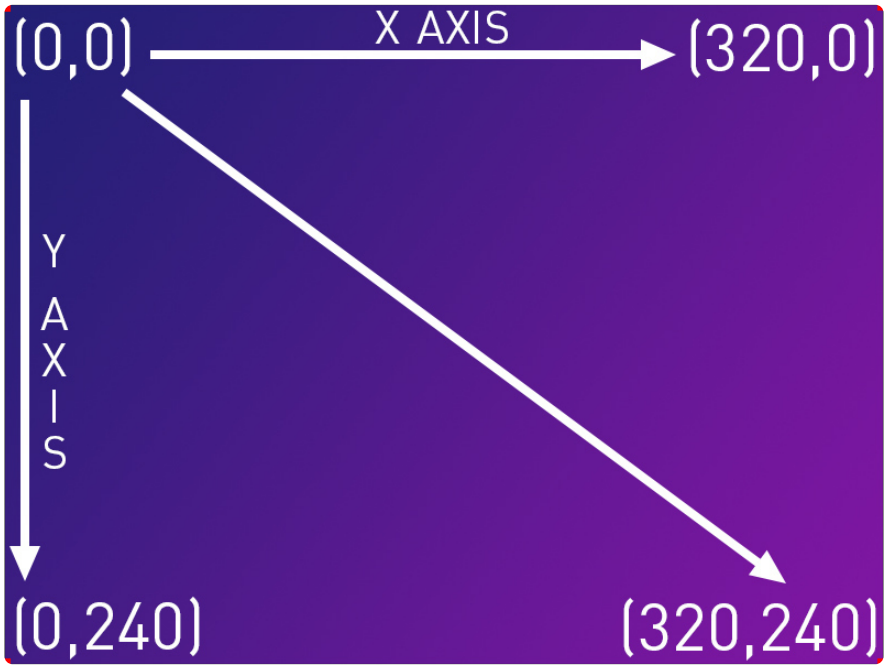

```
Temporarily unable to load content:
```

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).

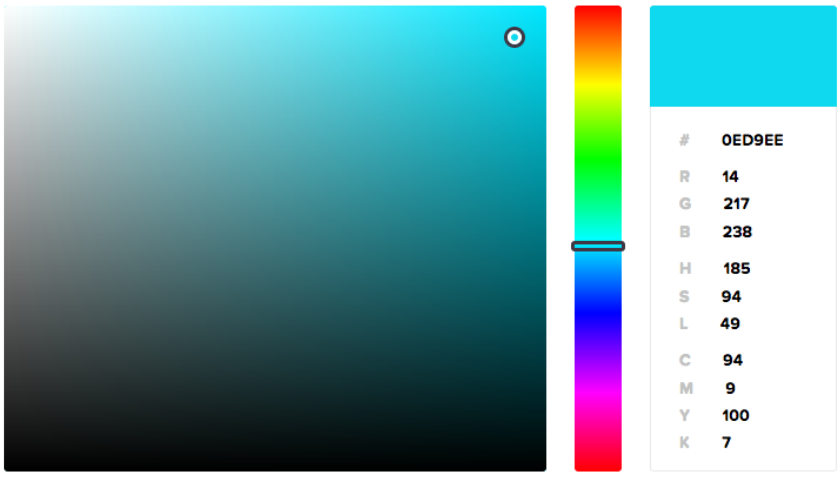


So, if you wanted to move the subscriber count text to the right and up closer to the top, your code may look like this for that part of the pyportal constructor: `text_position=(250, 10)`

Text Color

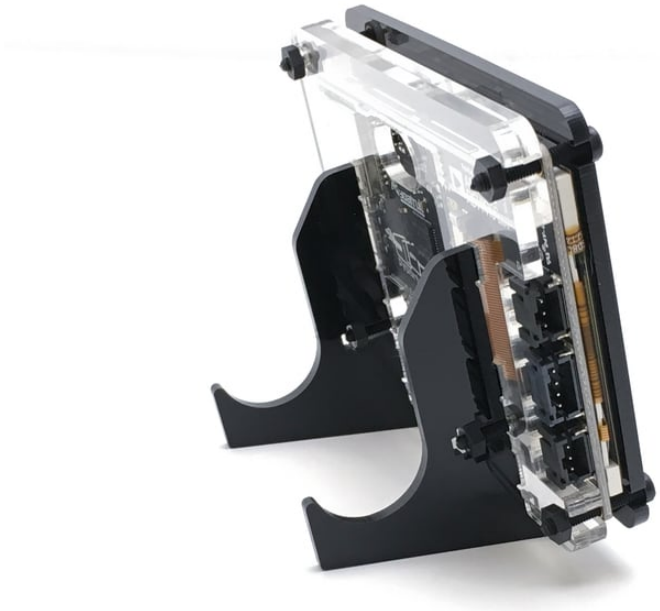
Another way to customize your display is to adjust the color of the text. The line `text_color=0xFFFFFF` in the constructor shows how. You will need to use the hexadecimal value for any color you want to display.

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`

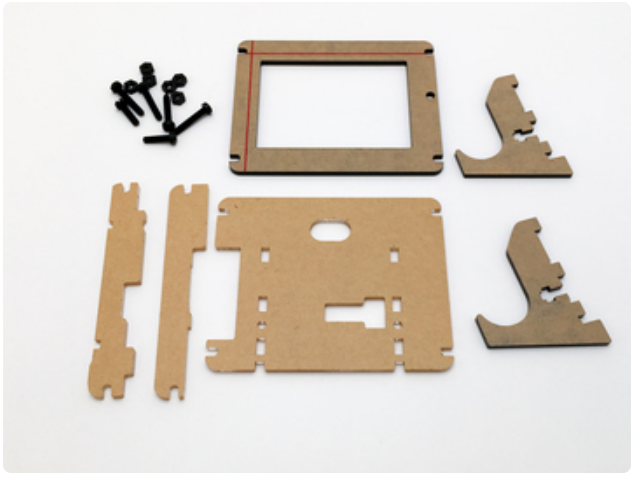


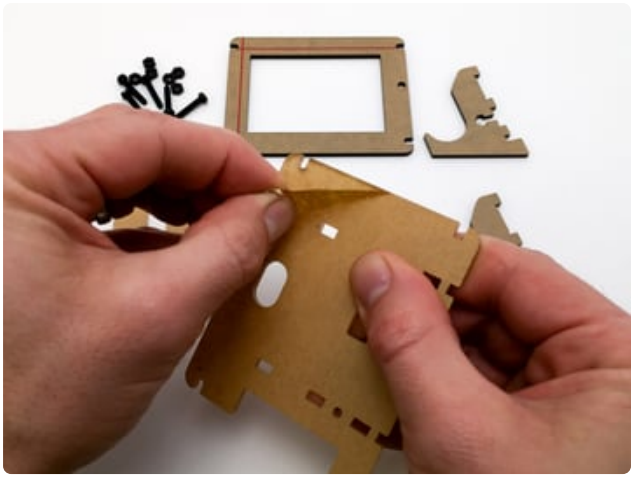
Now, we get ready to assemble the case on the next page.

Build The PyPortal Stand



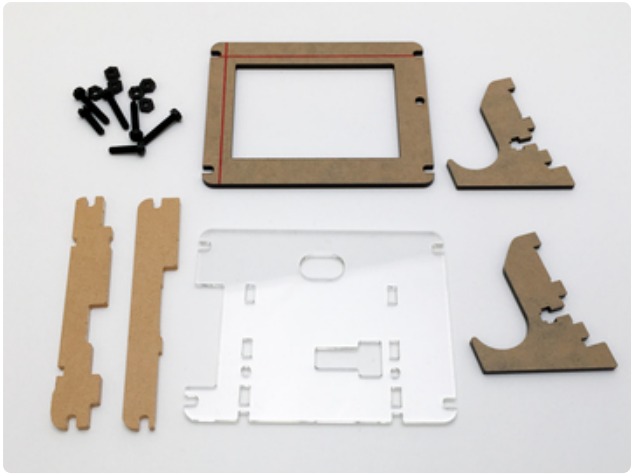
Here's how to assemble the laser cut acrylic stand for the PyPortal. The kit comes with six pieces of acrylic and six nylon screws and nuts.

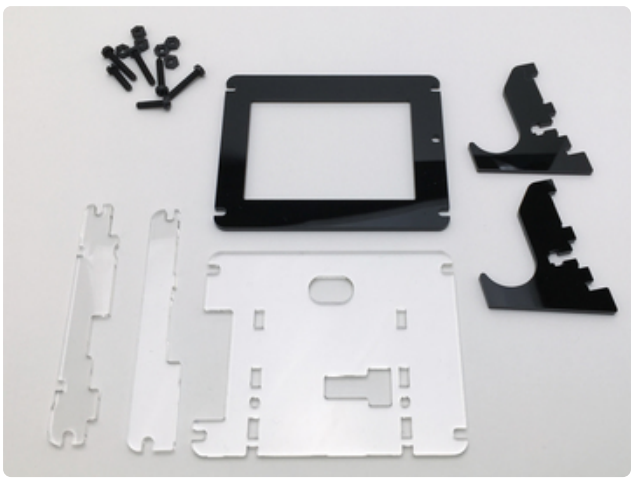


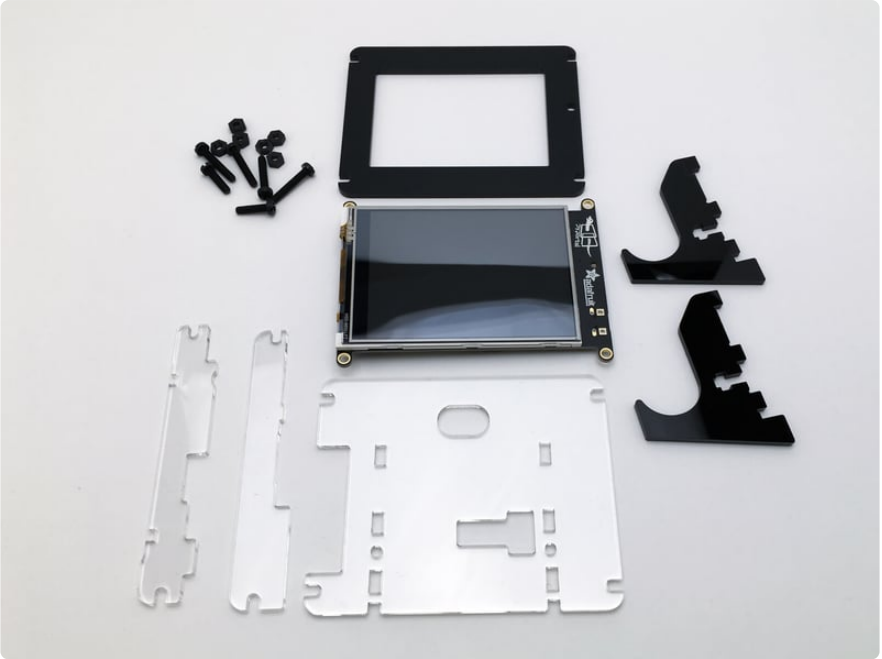


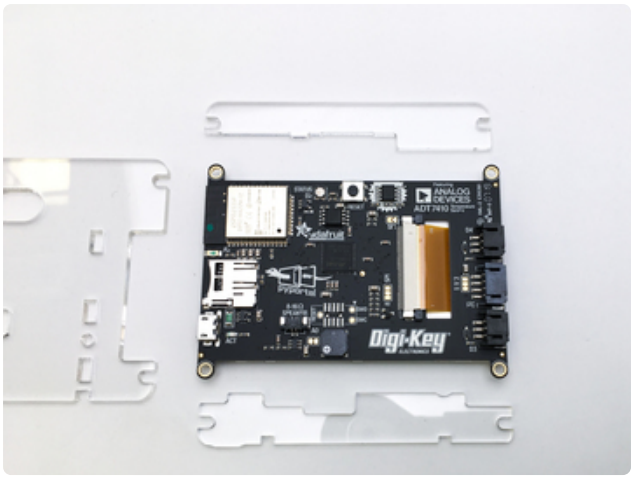
Prep

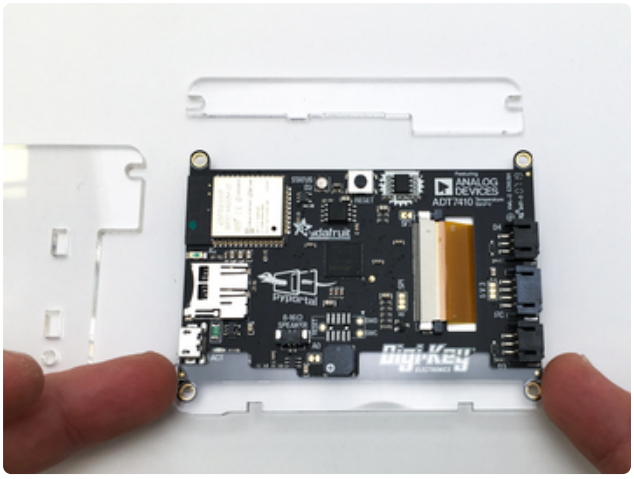
First, remove the protective paper from all of the acrylic pieces.











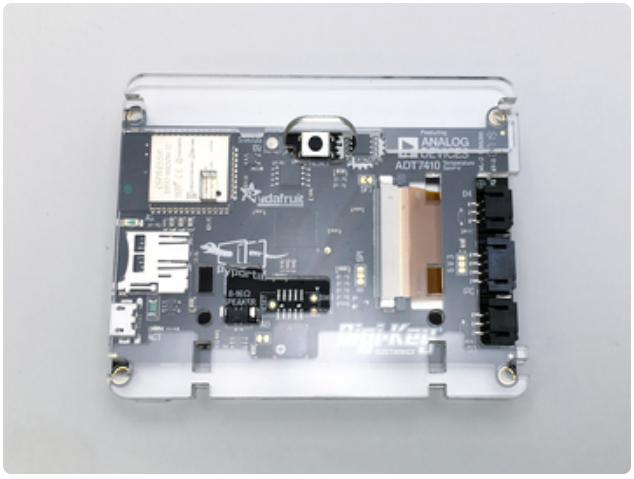
Sandwich

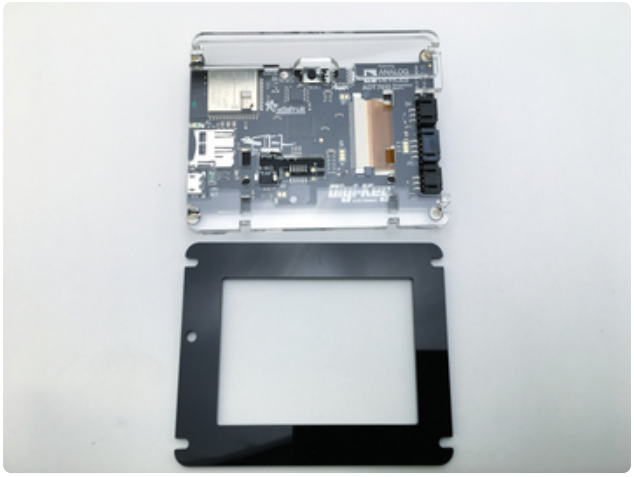
Next, do a dry fit of the three clear piece of acrylic on the back side of the PyPortal to get everything oriented properly.

The two small pieces are used as spacers to allow clearance around some of the larger parts. Lay them onto the board first, as shown.

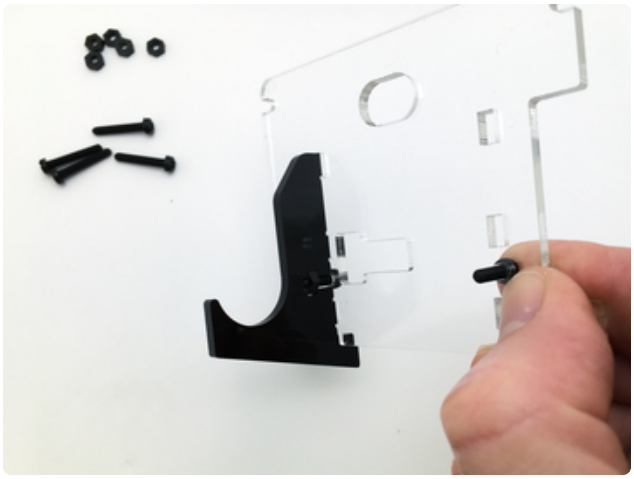
Then, place the large clear piece on top, making sure to align the hole for the reset and the cutout for the three JST ports.

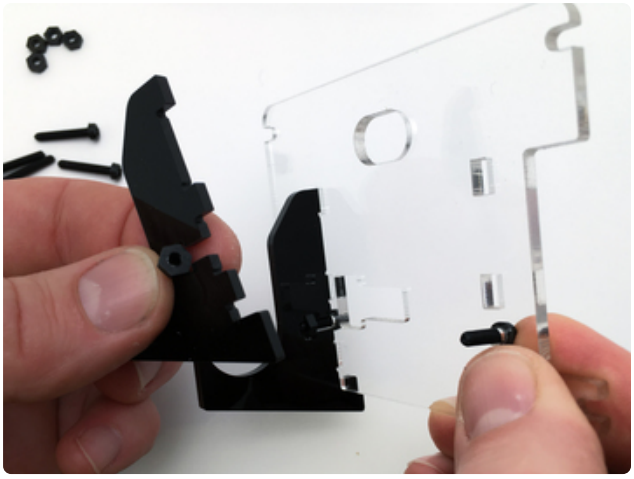
Complete the sandwich by placing the stack on top of the black front bezel with the hole for the light sensor oriented as shown here.

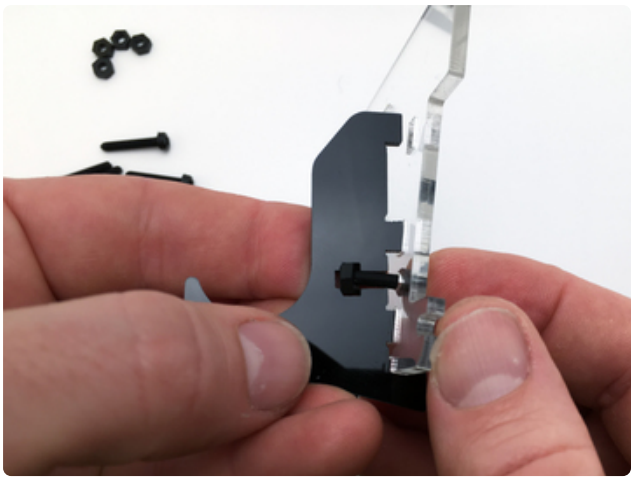





STATUS
D2
Featuring
ANALOG
DEVICES
ADT7410
adafruit
SPI
5V3
I2C
8-16Ω
SPEAKER
RESET
SWD
SWC
ACT
Digi-Key
ELECTRONICS





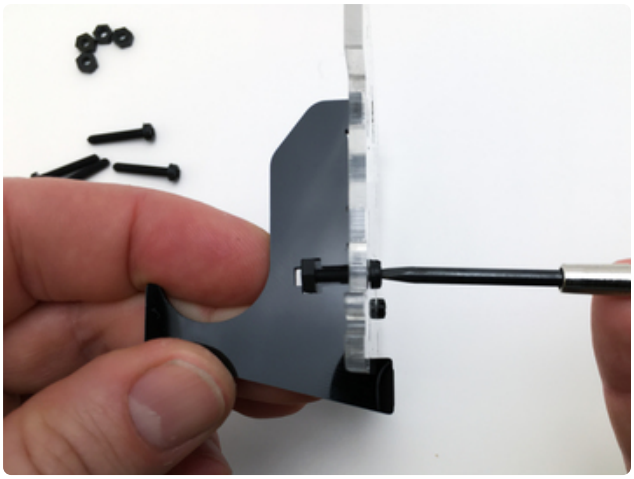


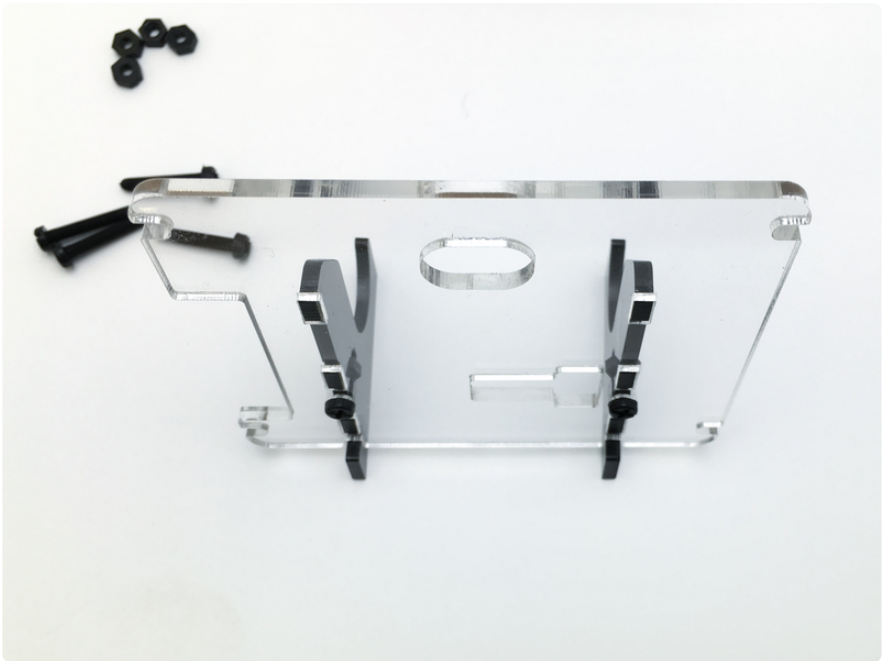
Legs

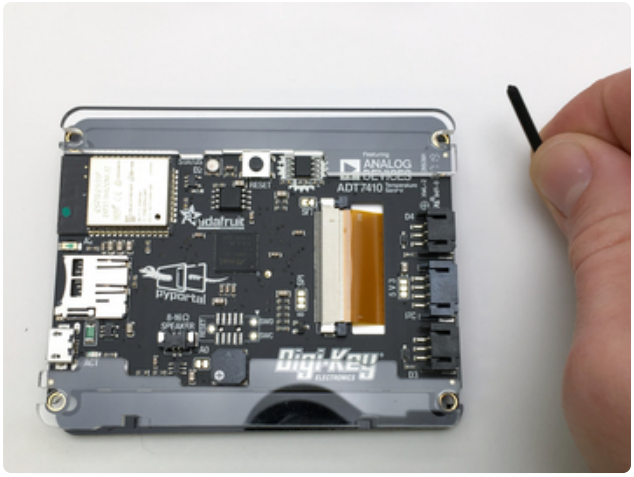
Now that the fit and orientation have been established, we'll install the legs.

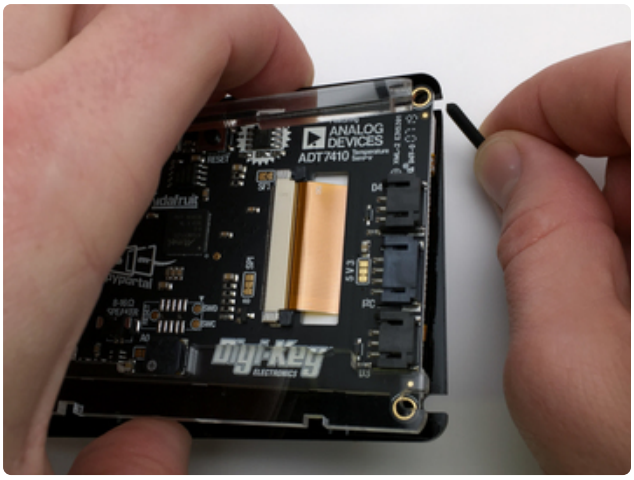
The two legs are identical. Pick one and slot it into the case back as shown.

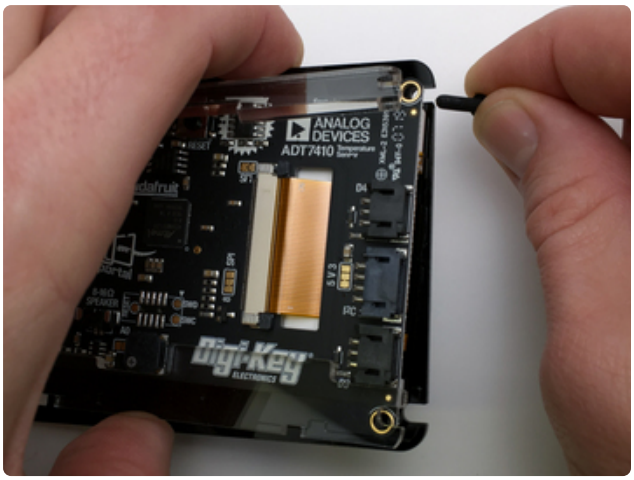
Place a nut into the captive slot of the leg and then feed a short screw through from the front of the clear acrylic case back. Fasten the screw (not too tight!) and then repeat for the second leg.







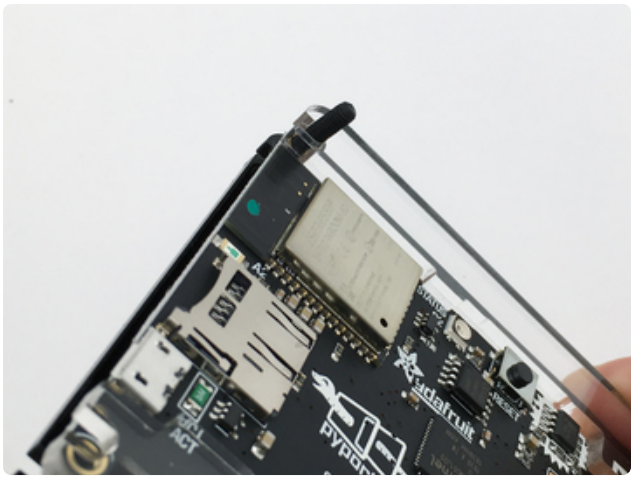


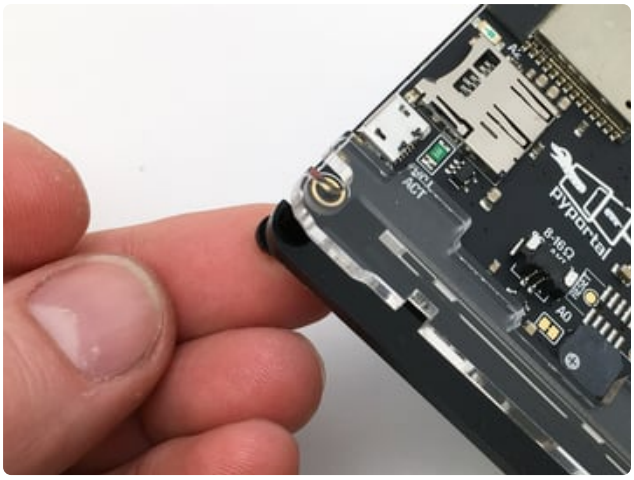


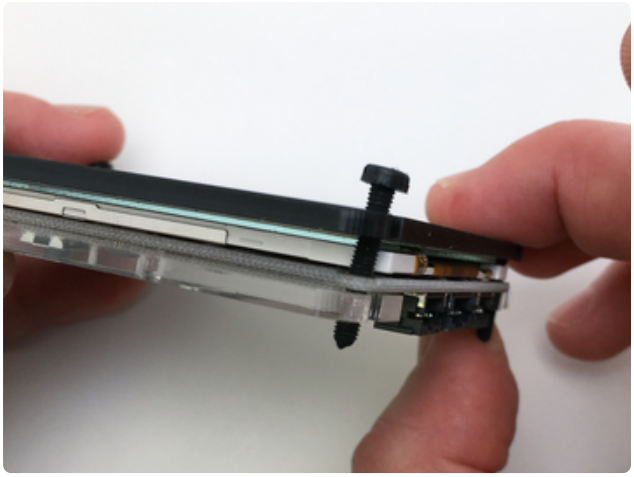
Add Long Screws

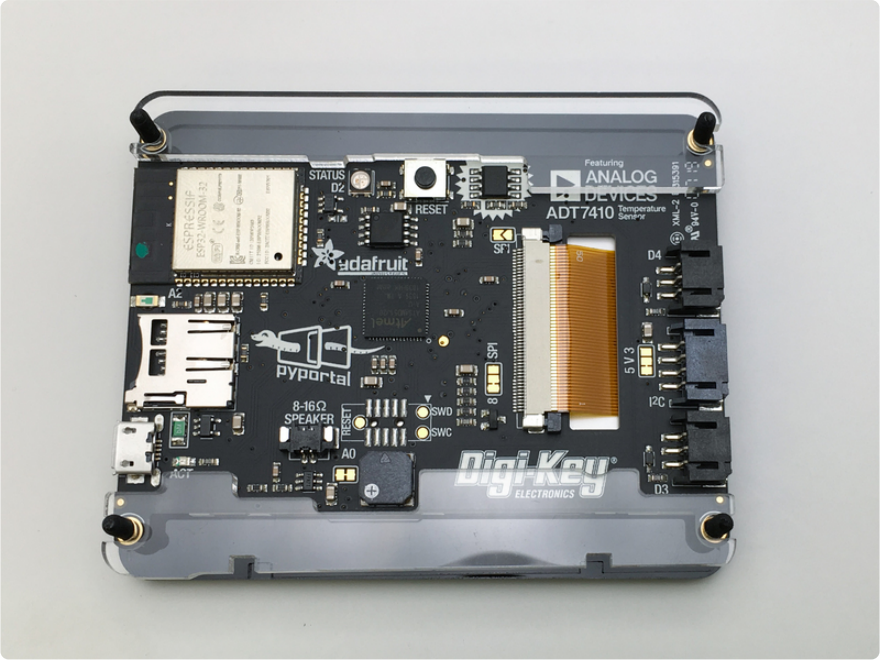
To put it all together, we'll use the four long screws to secure the entire acrylic - PyPortal - acrylic - acrylic sandwich!

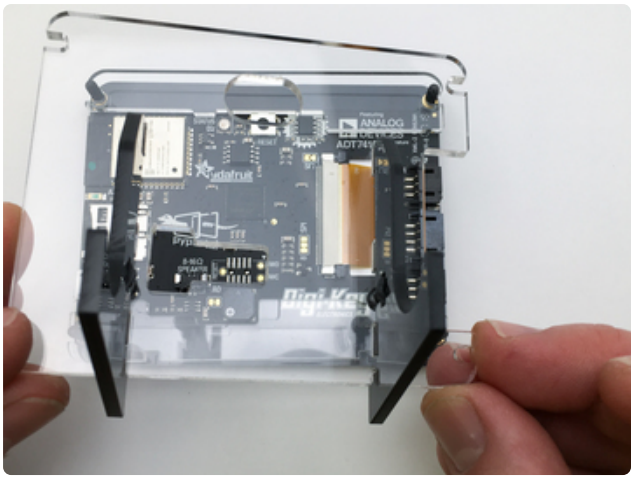
Run the four long screws from the front to the back, as shown.

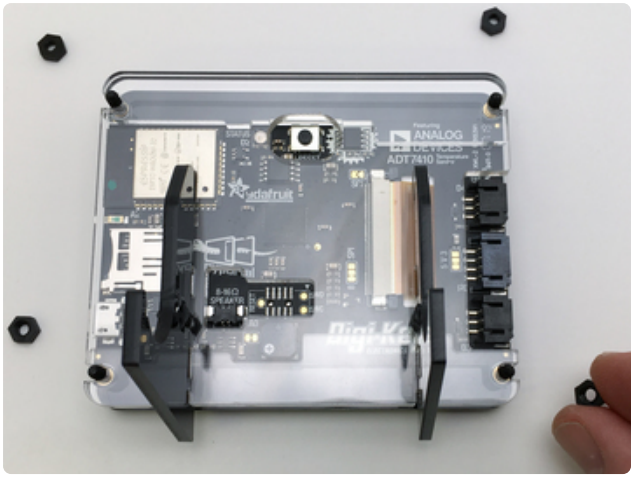

adafruit
pyportal

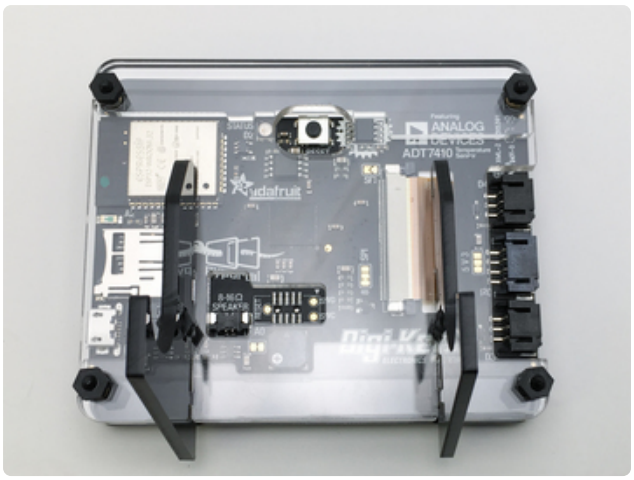

pyportal




STATUS
D2
RESET
Featuring
ANALOG
DEVICES
ADT7410
Temperature
Sensor
SFT
adafruit
D4
A2
pyportal
SPI
5 V 3
I2C
8-16Ω
SPEAKER
SWD
SWC
A0
Digi-Key
ELECTRONICS
D3



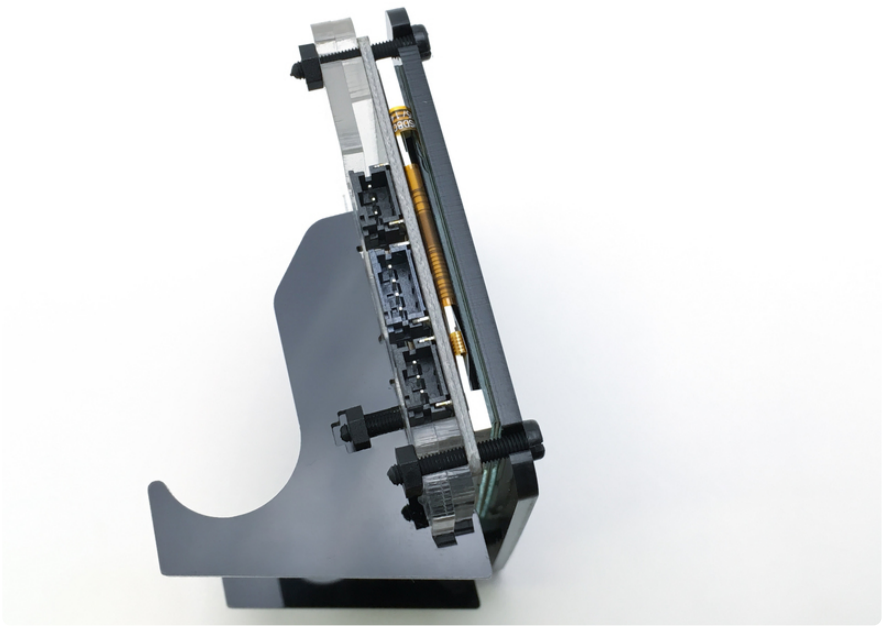




Screw It All Together

Finally, add the case back and legs assemblage and then thread on the four nuts to secure it all in place.

Be **careful not to over-tighten** the screws. Doing so can potentially crack the Pyportal display!



Bonus! Penny Roll Weight

If you'd like to give your PyPortal a bit of extra heft so it won't get pushed around on your desk, you can make a great weight for $0.50. A roll of 50 pennies does the trick! The legs are designed to hold a roll of coins perfectly!

Laser Cutter Files for PyPortal Stand

If you need to replace a piece or just want to make a spare for another PyPortal, here are the vector files for 1/8" (3mm) acrylic, in Adobe Illustrator format:

pyPortal_CUT_Black.ai

https://adafru.it/EqN

pyPortal_CUT_Clear.ai

https://adafru.it/EqO

Troubleshooting

While setting up our PyPortal with this project for Electioncal US, you might run into a bit of trouble. Nothing to worry, all should be easy to solve.

Problems with the WiFi

If something is wrong with the WiFi connection, one way to recognize this problem is by connecting to the REPL Serial Console, where you might find messages like:

```
Connecting to AP my-access-point
Could not connect to internet ('Failed to connect to ssid', b'my-
```

`access-point')`
`Retrying in 3 seconds...`

Here are some things you might look at to solve this:

- Checking the name of the access point and it's password on the **secrets.py** file
- Getting closer to the access point if it is far away

Problems setting up the county/state

If the JSON data cannot be found, you will see an error such as:

`Error loading JSON data: Please check the configuration of county and state, in code.py`

`Traceback (most recent call last):`

`File "code.py", line 60, in`

`File "code.py", line 57, in`

`File "electioncal_graphics.py", line 76, in load_data`

`File "electioncal_graphics.py", line 72, in load_data`

`ValueError: syntax error in JSON`

If this happens:

- Check the spelling of the state and county
- If the name of your state or county has a space, replace it with an underscore _
- Remember to lowercase both names

PyPortal library on other devices

The PyPortal is a wonderful device, but it is not the only device that has a screen, such as the larger and smaller versions of the PyPortal such as the PyPortal Titano and the PyPortal Pynt.

Everything in the code should work well, but the laying of the text and background might be a bit off. Nothing to worry here.

Make gradual changes to **electioncal_graphics.py** in the `__init__` section, where `x` and `y` are defined for all of the displayed objects.