Introducing Adafruit CLUE

Created by Kattni Rembor

https://learn.adafruit.com/adafruit-clue
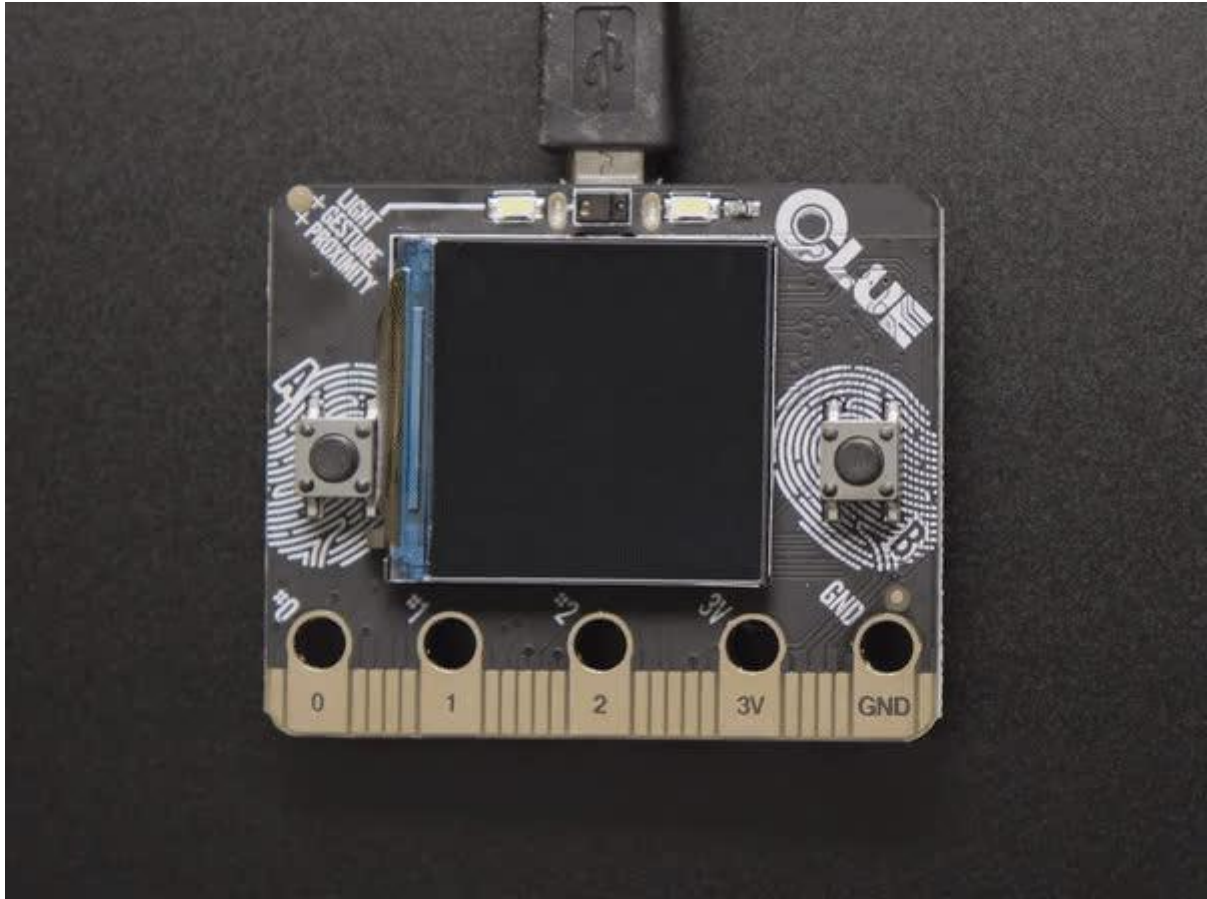
Last updated on 2023-08-29 03:18:22 PM EDT

Table of Contents

Overview

> Your board's bootloader must be 0.6.1 or later to be able to load CircuitPython 8.2.0 or later. See the Update Bootloader section in this guide.

Do you feel like you just don't have a CLUE? Well, we can help with that - get a CLUE here at Adafruit by picking up this sensor-packed development board. We wanted to build some projects that have a small screen and a lot of sensors. To make it compatible with existing projects, we made it the same shape and size as the BBC micro:bit () and with the same edge-connector on the bottom with 5 big pads so it will fit into your existing robot kit or 'bit add-on.
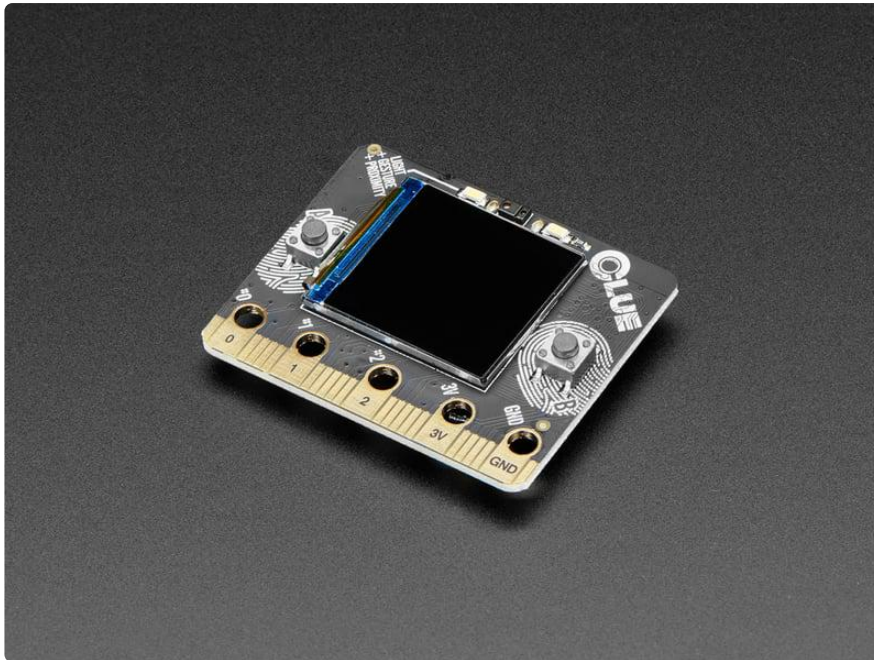
While the CLUE looks a bit like a 'bit it has totally redesigned-from-scratch technology:

- Nordic nRF52840 Bluetooth LE processor - 1 MB of Flash, 256KB RAM, 64 MHz Cortex M4 processor
- 1.3″ 240×240 Color IPS TFT display for high resolution text and graphics
- Power it from any 3-6V battery source (internal regulator and protection diodes)
- Two A / B user buttons and one reset button
- Tons of sensors!
    - ST Micro series 9-DoF motion - LSM6DS33 Accel/Gyro () + LIS3MDL () magnetometer ()
    - APDS9960 Proximity, Light, Color, and Gesture Sensor ()
    - PDM Microphone sound sensor ()
    - SHT Humidity ()
    - BMP280 temperature and barometric pressure/altitude ()
- RGB NeoPixel indicator LED
- 2 MB internal flash storage for datalogging, images, fonts or CircuitPython code
- Buzzer/speaker for playing tones and beeps
- Two bright white LEDs in front for illumination / color sensing.
- Qwiic / STEMMA QT connector for adding more sensors, motor controllers, or displays over I2C. You can plug in GROVE I2C sensors by using an adapter cable ().
- Programmable with Arduino IDE or CircuitPython
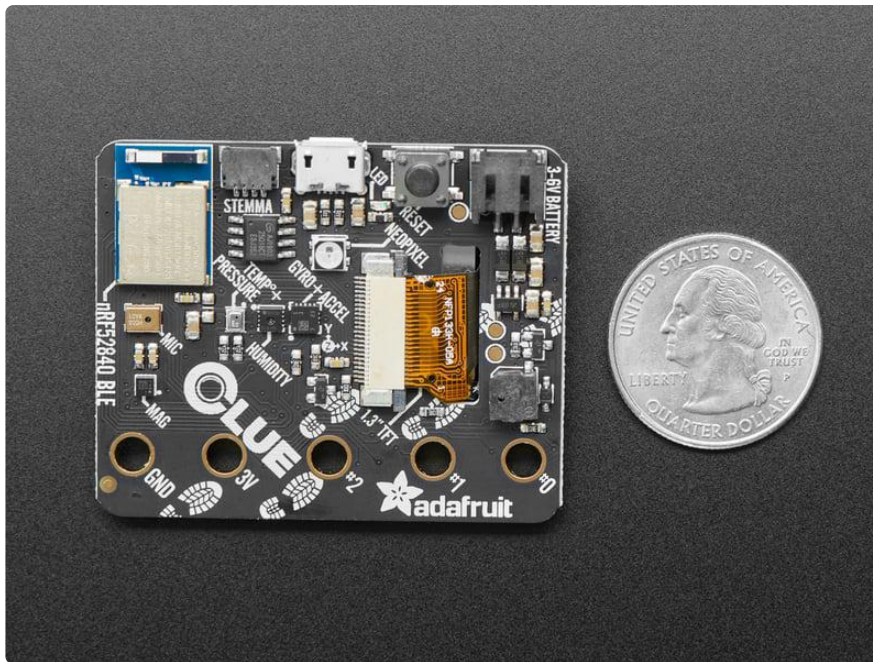
Please note that at this time there is no MakeCode or Scratch support for the nRF52840 chipset (of course, we'd love to see MakeCode but there is no ETA when it may be added). While the CLUE is the same outline and we did our best to make the edge-connector pins match up, most cases for the 'bit wont fit the CLUE, and code may not be immediately compatible without adjustment, especially since only Arduino and CircuitPython are supported at this time.
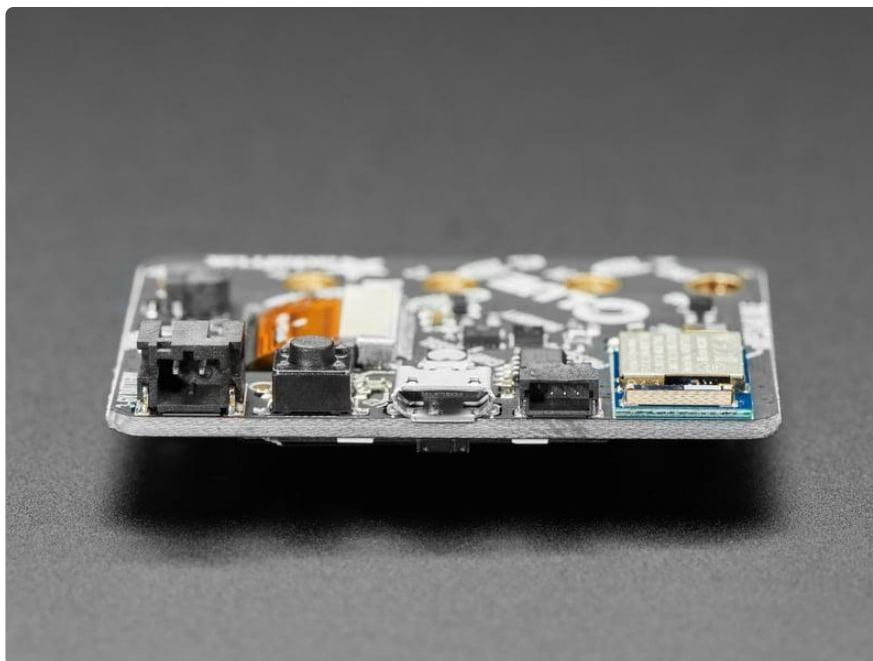
The CLUE is designed for projects that use a ton of sensors - and they're all built in! So you can start exploring your world, measuring, logging and learning. You can transmit data over Bluetooth to a computer or mobile device for data plotting and logging, or save it to the built in storage.

Update Bootloader

> Your board's bootloader must be 0.6.1 or later to be able to load CircuitPython 8.2.0 or later.

Older versions of Adafruit's nRF52840 boards were shipped with a bootloader that does not handle large UF2's, including CircuitPython 8.2.0 and later, and has other issues. To check whether you need to update the bootloader, double-click the reset button, and look in the `...BOOT` drive for INFO_UF2.TXT. Inside that file, check the version number. It should be 0.6.1 or newer. This one is older:

```
UF2 Bootloader 0.2.9 lib/nrfx (v1.1.0-1-g096e770) lib/tinyusb (legacy-755-
g55874813) s140 6.1.1
Model: Adafruit Feather nRF52840 Express
Board-ID: nRF52840-Feather-revD
Bootloader: s140 6.1.1
Date: Feb 22 2019
```

The dafruit nRF52 Bootloader can be upgraded/downgraded without any additional hardware. There are 3 different ways to update bootloader, each has its pros and cons:

- Use UF2: This is the fastest and safest way to update bootloader. However, it requires your existing bootloader version is at least 0.4.0 and only work with nRF52840 (not nRF52832).

- Use Arduino IDE: work with all Adafruit nRF52 boards and bootloader version, typo free but may not be the latest bootloader version due to the BSP release cycle.
- Use Command Line: work with all boards and bootloader version (including 3rd party one). This command line uses the back-end of the Arduino IDE method above.

Use UF2

This is the fastest and safest way to update bootloader. However, it requires your existing bootloader is at least 0.4.0 and only work with nRF52840 (not nRF52832) since UF2 make use of USB MSC interface.

Download update-bootloader UF2

You need a `.uf2` file containing the latest version of the bootloader. Use the link below to download it.

<div style="text-align:center">

Latest .uf2 of Feather nRF52840 bootloader

</div>

Enter Bootloader Mode

Double-click the Reset button on your board, and you will see the NeoPixel RGB LED turn green (identified by the arrow in the image). If it turns red, check the USB cable, try another USB port, etc.

Note: on nRF52840 USB Key with TinyUF2 (PID 5199) you need to hold its button while plugging into your PC.

Drap & Drop UF2

You will see a new BOOT disk drive appear e.g FTHR840BOOT. Drag the downloaded `.uf2` file to FTHR840BOOT. The LED will flash. Then, the FTHR840BOOT drive will disappear. That's it, you have successfully update your board to the latest version.
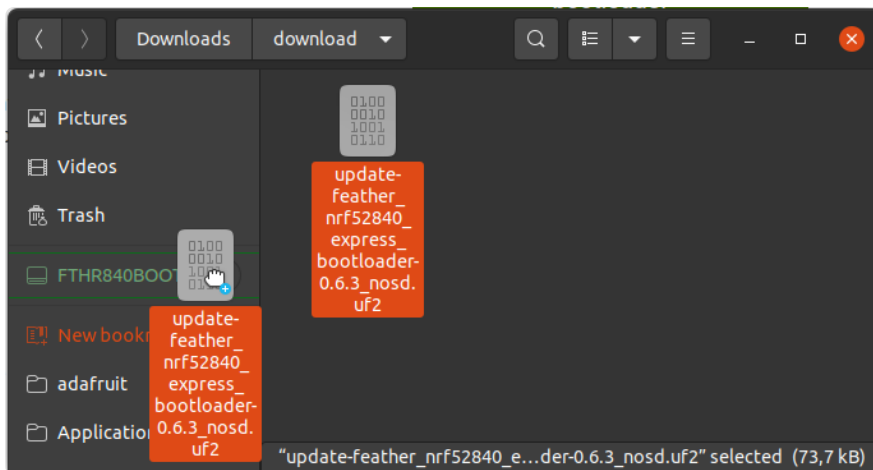
Use Arduino IDE

Arduino IDE Burn Bootloader menu option will pick the correct bootloader binary for your selected board and prevent any command typos or other common errors.

> Close the Serial Monitor before you click "Burn Bootloader". Afterwards, you shouldn't close the Arduino IDE, unplug the board, launch Serial Monitor etc ... to abort the process. There is a high chance it will brick your device! Do this with care and caution.

> This is an advanced method: if your existing bootloader version is 0.4.0 or later, you can use the UF2 method, which is faster and safer.

Make sure no other program is attached to the serial port on the board, such as Mu, the Serial Monitor in Arduino, or a terminal program. Then, to start, select the correct board you are using under Tools -> Board
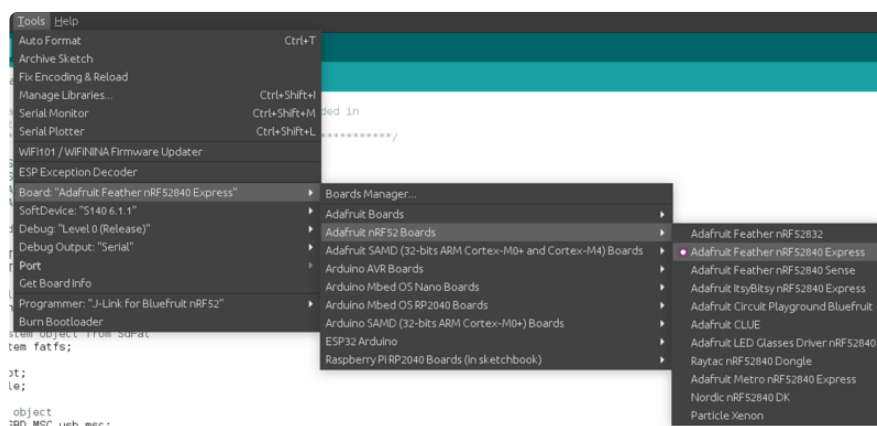
Then select "Bootloader DFU for Bluefruit nRF52" for Tools->Programmer
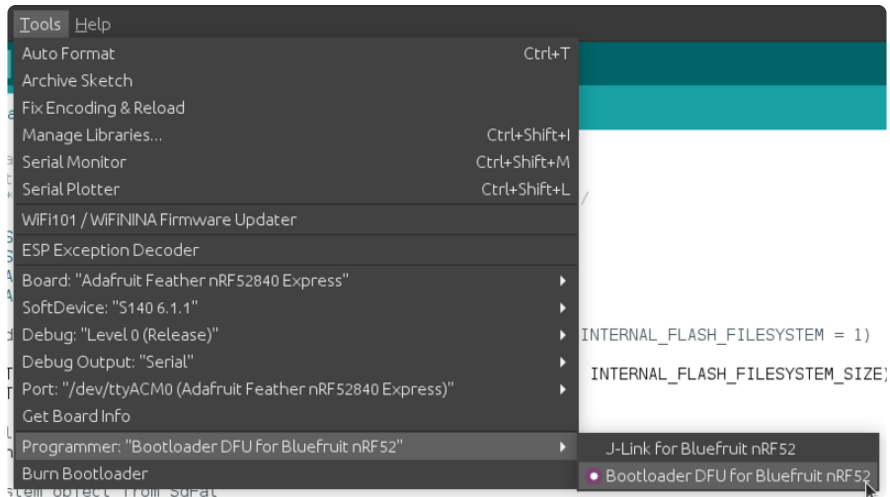
Double check all of the following: Board, Programmer...

Then select Tools->Burn Bootloader to start the upgrade.
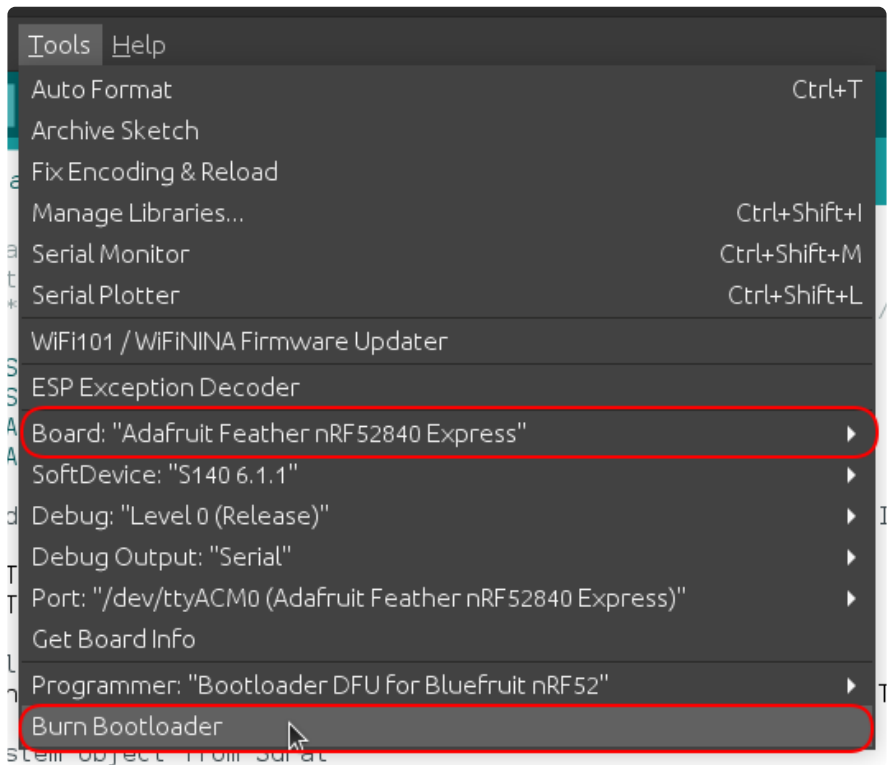
After receiving the new Bootloader over the serial connection, the old Bootloader will erase itself! The new bootloader will then be flashed. The process typically takes 30-60 seconds to complete. Make sure you see the "Device programmed" in the output log before launching Serial monitor or uploadinga new sketch.

```
Upgrading target on /dev/ttyACM0 with DFU package /Adafruit_nRF52_Arduino/
bootloader/feather_nrf52840_express/
feather_nrf52840_express_bootloader-0.6.2_s140_6.1.1.zip. Flow control is disabled,
Dual bank, Touch 1200
Touched serial port /dev/ttyACM0
Opened serial port /dev/ttyACM0
Starting DFU upgrade of type 3, SoftDevice size: 151016, bootloader size: 39000,
```

```
application size: 0
Sending DFU start packet
Sending DFU init packet
Sending firmware file
######################################
######################################
######################################
######################################
######################################
######################################
######################################
######################################
######################################
############
Activating new firmware

DFU upgrade took 20.50154972076416s
Device programmed.
```

Drawbacks of this method are that you will need to install the Arduino IDE, and the bundled bootloader may not be the latest one. Check out the next page for a more advanced command line version.

Use Command Line

This is advanced method: if your existing bootloader version is 0.4.0 or later, you can use the UF2 method, which is faster and safer.

Download Bootloader Package

To update the bootloader, you need a `.zip` file containing the latest version of the bootloader. Use the link below to download it. Then unzip the `.zip` you downloaded.

Latest .zip of Feather nRF52840
bootloader and associated files

Download adafruit-nrfutil

You will also need the utility program "adafruit-nrfutil", which has slightly different names on different platforms.

On Linux, you can download and install `adafruit-nrfutil` by doing:

```
$ pip3 install --user adafruit-nrfutil
```

On MacOS, if you have `python3` and `pip3` installed, you can install via `pip3`, as above for Linux. Otherwise download `adafruit-nrfutil-macos` from this link:

adafruit-nrfutil-macos for MacOS

Then make it executable by doing:

```
chmod +x adafruit-nrfutil-macos
```

On Windows, download `adafruit-nrfutil.exe` from this link:

adafruit-nrfutil.exe (for Windows)

Update Bootloader

To update the bootloader, first connect the board, and then double-click to get `NRF52 BOOT`. In the commands below, substitute the name of the `.zip` file you downloaded for file given there.

On Linux, run this command:

```
adafruit-nrfutil --verbose dfu serial --package
feather_nrf52840_express_bootloader-0.2.9_s140_6.1.1.zip -p /dev/ttyACM0 -b 115200
--singlebank --touch 1200
```

On MacOS, find out the device name for the connected board, by doing:

```
ls /dev/cu.*
```

The device name will be something like `/dev/cu.usbmodem411`. Use this command, substituting the device name you've discovered. If you are running it other than where you downloaded `adafruit-nrfutil-macos`, change the path to the command accordingly. If you installed it via pip3, it's just called `adafruit-nrfutil`, and it should be in your PATH.

```
./adafruit-nrfutil-macos --verbose dfu serial --package
feather_nrf52840_express_bootloader-0.2.9_s140_6.1.1.zip -p /dev/cu.usbmodem411 -b
115200 --singlebank --touch 1200
```

On Windows, use this command, in the folder where you downloaded the other two files above. You'll need to specify the correct COM port, instead of `COMxx`. Look in De vice Manager for the name of the COM port (it will be listed as a "USB Serial Device" on Windows 10) after you have double-clicked the reset button.

```
adafruit-nrfutil.exe --verbose dfu serial --package
feather_nrf52840_express_bootloader-0.2.9_s140_6.1.1.zip --port COMxx -b 115200 --
singlebank --touch 1200
```

Once done, click the reset button again - the bootloader will be running!

Pinouts

There's all kinds of features packed into CLUE. Let's take a look!
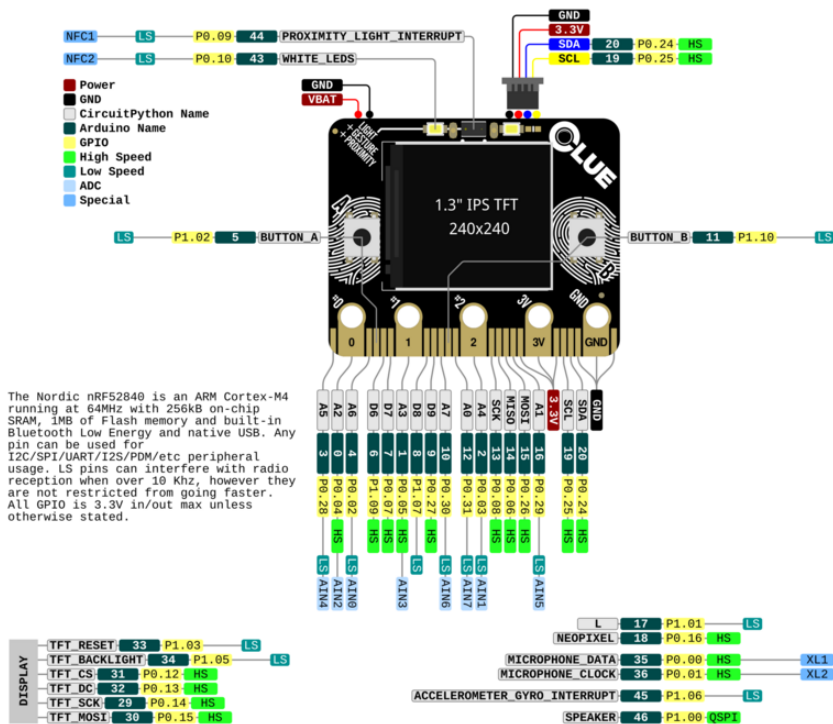
[Click here to view a PDF version of the pinout diagram]() ()
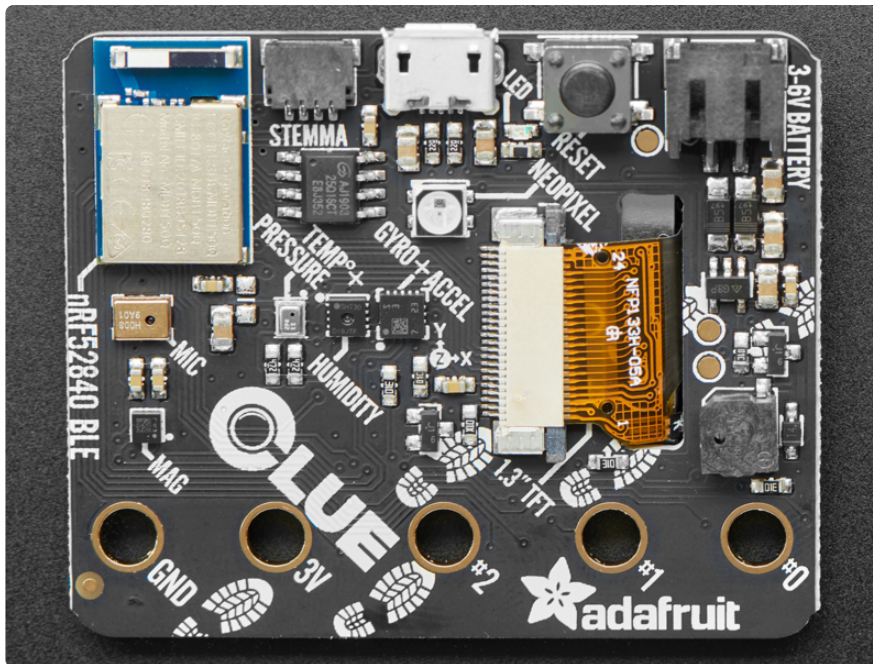
Microcontroller and QSPI
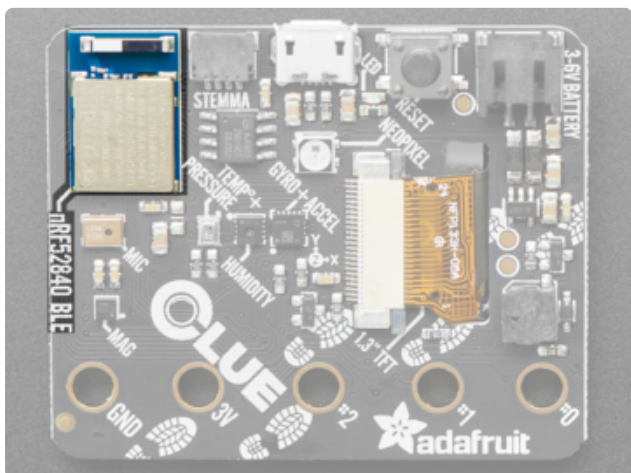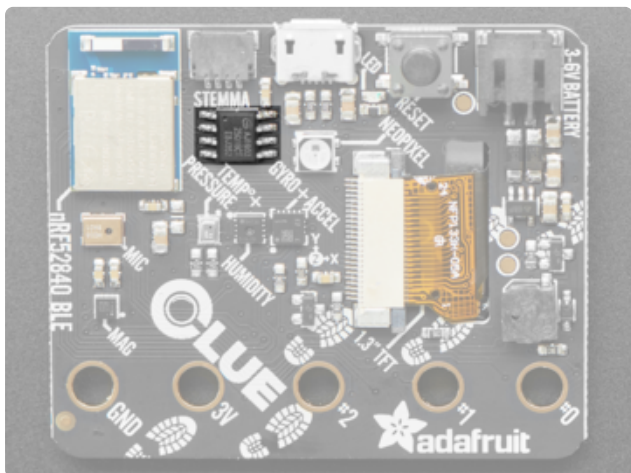
Nordic nRF52840 Bluetooth LE processor - 1 MB of Flash, 256KB RAM, 64 MHz Cortex M4 processor.
QSPI flash - 2MB of internal flash storage for datalogging, images, fonts or CircuitPython code.

Display
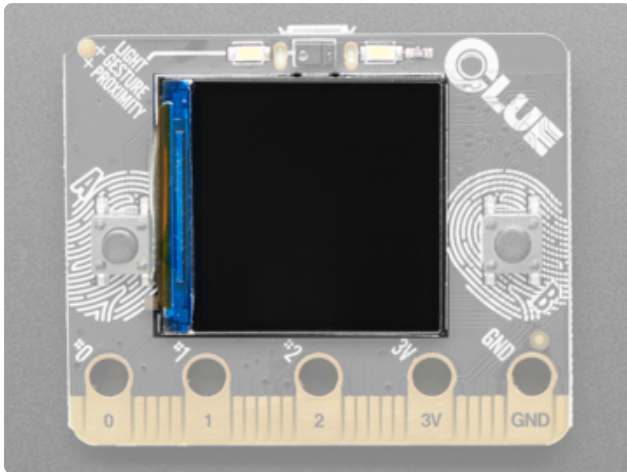
1.3″ 240×240 Color IPS TFT display - Display high resolution text and graphics. The cable goes through a slot in the board to the back to the display connector. The front of the TFT has a controller chip embedded in the connector cable (you can see it as a thin rectangle to the left of the display). This chip is light sensitive, so if you use a xenon strobe you may disable the display. If you need to use the CLUE In a strobe/very-high-brightness setup, cover up the chip with a strip of black electrical tape
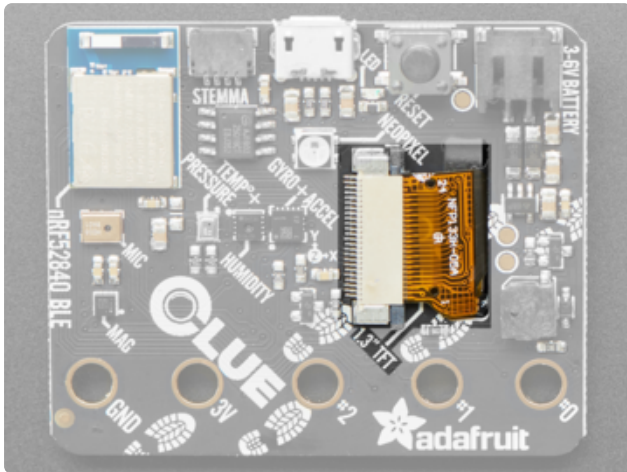
Sensors
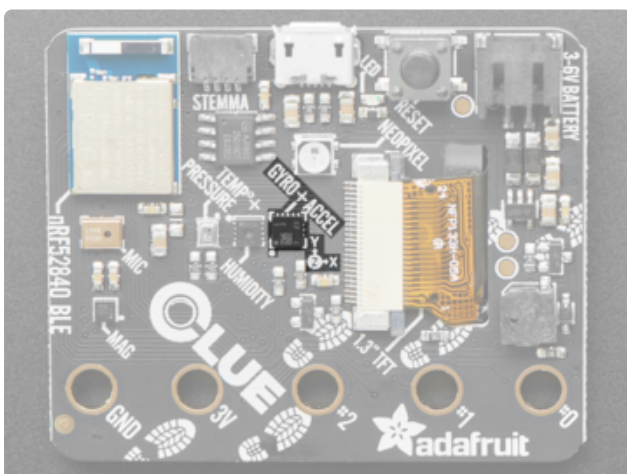
Gyro + Accel: LSM6DS33 - This sensor is a 6-DoF IMU accelerometer + gyroscope. The 3-axis accelerometer, can tell you which direction is down towards the Earth (by measuring gravity) or how fast the CLUE is accelerating in 3D space. The 3-axis gyroscope that can measure spin and twist. Pair with a triple-axis magnetometer to create a 9-DoF inertial measurement unit that can detect its orientation in real-space thanks to Earth's stable magnetic field. Sensor is I2C on standard pins on address 0x6A.
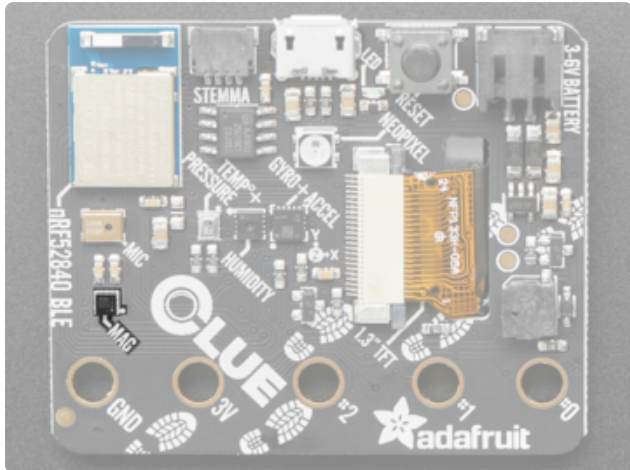
Magnetometer: LIS3MDL - Sense the magnetic fields that surround us with this handy triple-axis magnetometer (compass) module. Magnetometers can sense where the strongest magnetic force is coming from, generally used to detect magnetic north, but can also be used for measuring magnetic fields. This sensor tends to be paired with a 6-DoF (degree of freedom) accelerometer/gyroscope to create a 9-DoF inertial measurement unit that can detect its orientation in real-space thanks to Earth's stable magnetic field. Sensor is I2C on standard pins on address 0x1C.
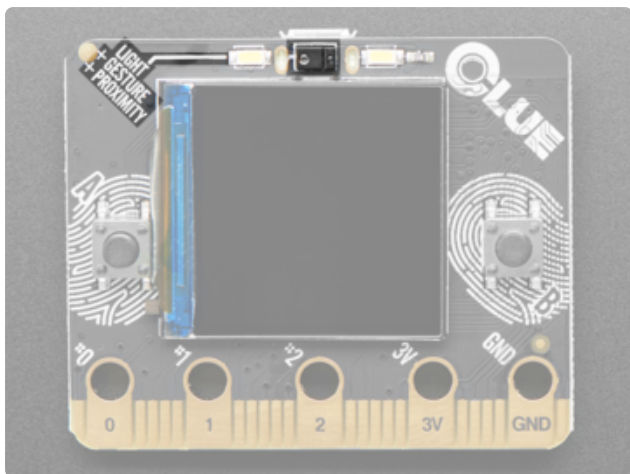
Light + Gesture + Proximity: APDS9960 - Detect simple gestures (left to right, right to left, up to down, down to up are currently supported), return the amount of red, blue, green, and clear light, or return how close an object is to the front of the sensor. This sensor has an integrated IR LED and driver, along with four directional photodiodes that sense reflected IR energy from the LED. Since there are four IR sensors, you can measure the changes in light reflectance at each of the cardinal locations over time and turn those changes into gestures. Sensor is I2C on standard pins on address 0x39.
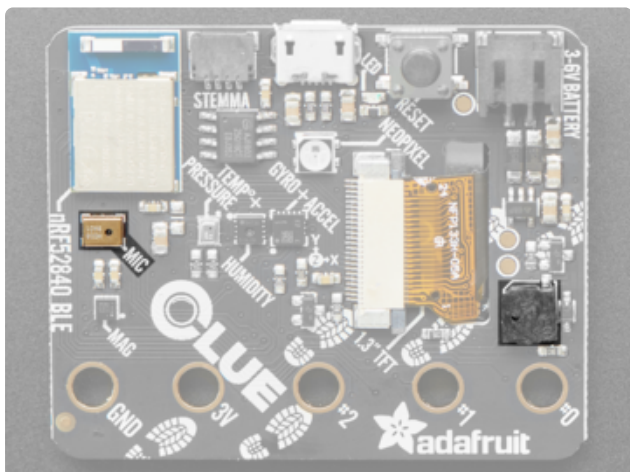
PDM Microphone sound sensor: MP34DT01-M - PDM sound sensor. In CircuitPython, `board.MICROPHONE_DATA` is PDM data, and `board.MICROPHONE_CLOCK` is PDM clock. In Arduino, `D35` is PDM data, and `D36` is PDM clock.
Speaker/buzzer - This tiny buzzer is good for playing back beeps and tones. It's not suitable for playing back audio files. Addressable in CircuitPython as `board.SPEAKER`, and in Arduino as `D46`.
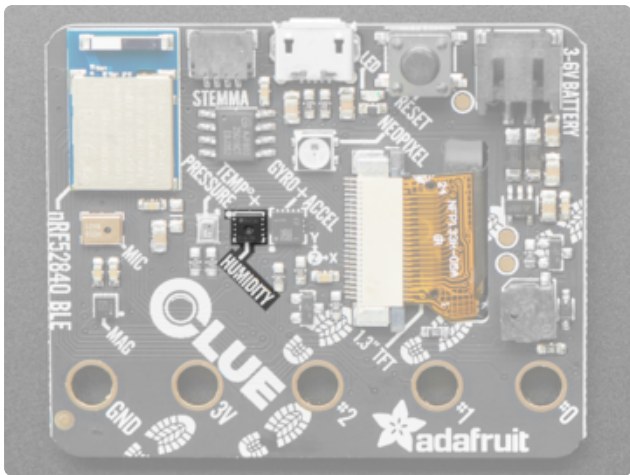
Humidity: SHT30 - This sensor has an excellent ±2% relative humidity and ±0.5°C accuracy for most uses. Sensor is I2C on standard pins on address 0x44.
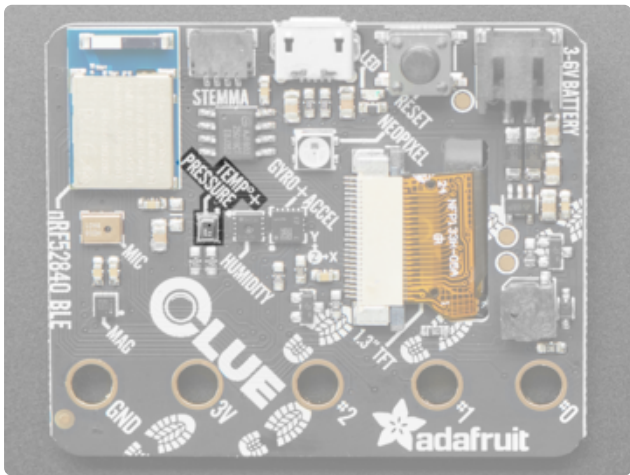
Temp + Pressure: BMP280 - This sensor is a precision sensing solution for measuring barometric pressure with ±1 hPa absolute accuraccy, and temperature with ±1.0°C accuracy. Because pressure changes with altitude, and the pressure measurements are so good, you can also use it as an altimeter with ±1 meter accuracy. It has a a low altitude noise of 0.25m and a fast conversion time. Sensor is I2C on standard pins on address 0x77.

USB and Battery

Like the micro:bit, the CLUE does not have built in LiPoly battery charging. This is for your safety so you can use Alkaline or NiMH batteries without damaging them! You can use LiPoly batteries but you will need an external charger
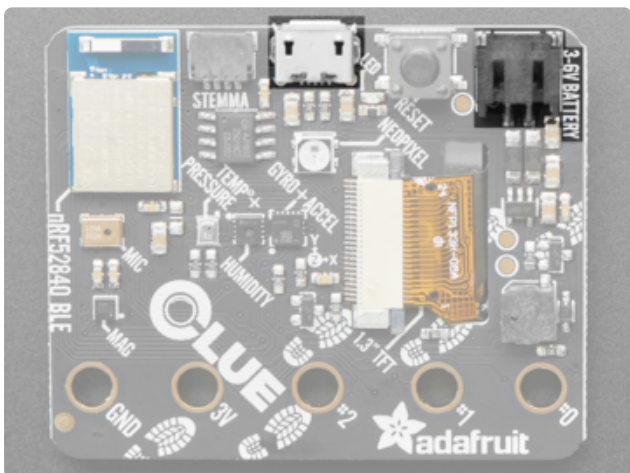
USB Micro - This USB port is used for programming and/or powering the CLUE. It is a standard USB Micro connector. Battery - 2-pin JST PH connector for a battery. Power the CLUE from any 3V-6V power source, as it has internal regulator and protection diodes.

Buttons
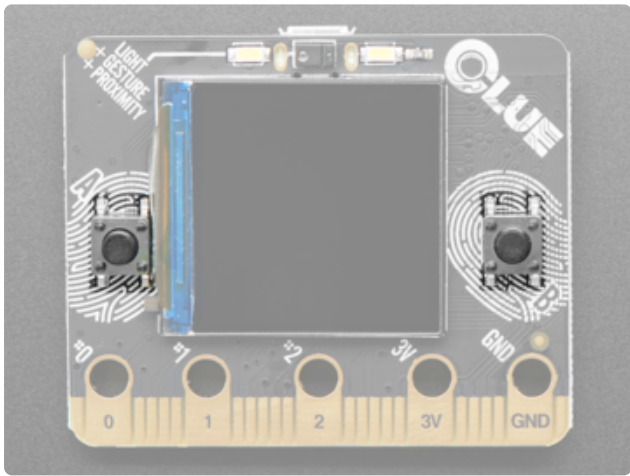
A and B buttons - The CLUE has two user-programmable buttons on the front, labeled A and B. Use them as inputs to control your code. These are unconnected when not pressed, and connected to GND when pressed, so they read LOW. Set the pins to use an internal pull-UP when reading these pins so they will read HIGH when not pressed. Buttons can be addressed in CircuitPython using `board.BUTTON_A` and `board.BUTTON_B`, and in Arduino as `D5` (left button) and `D11` (right button).
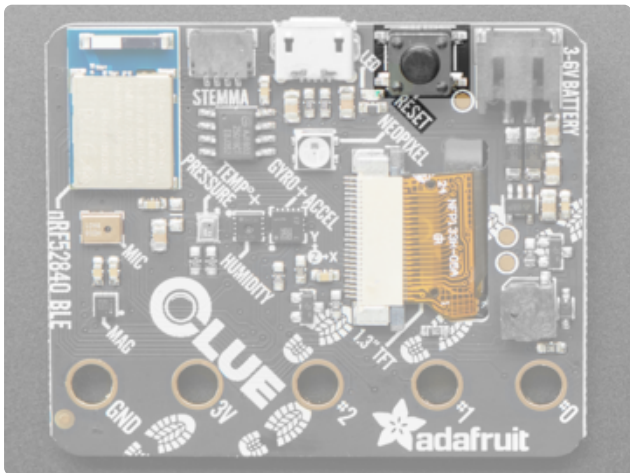
Reset button - This button resets the board. Press once to reset. Quickly press twice to enter the bootloader.

STEMMA QT
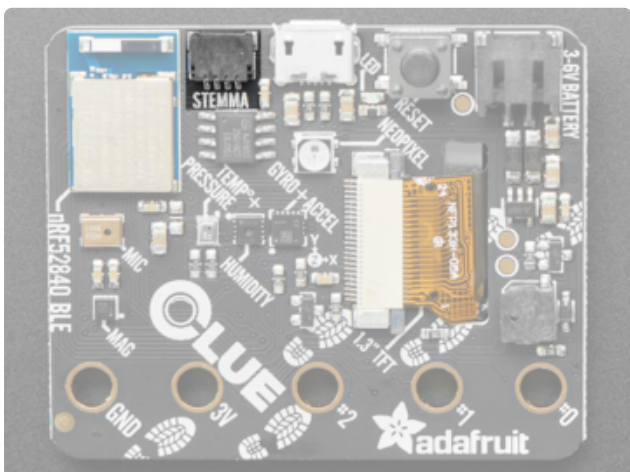
Qwiic / STEMMA QT connector - Use to add more sensors, motor controllers, or displays over I2C. You can plug in GROVE I2C sensors by using an adapter cable (). In CircuitPython, you can use the STEMMA connector with `board.SCL` and `board.SDA`, or `board.STEMMA_I2C()`.

LEDs
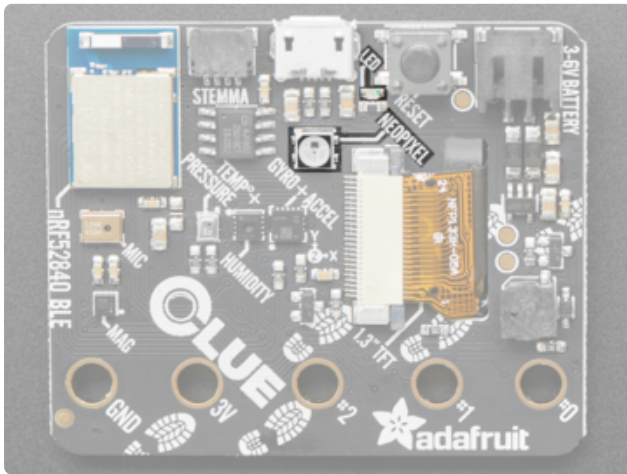
NeoPixel - The addressable RGB NeoPixel LED is used as a status LED by the bootloader and CircuitPython, but is also controllable using code. Control it using `board.NEOPIXEL` in CircuitPython and `D18` in Arduino.

Status LED - This little red LED works as a status LED in the bootloader. Otherwise, it is controllable using code by addressing `board.D17` in CircuitPython, and `D17` in Arduino.
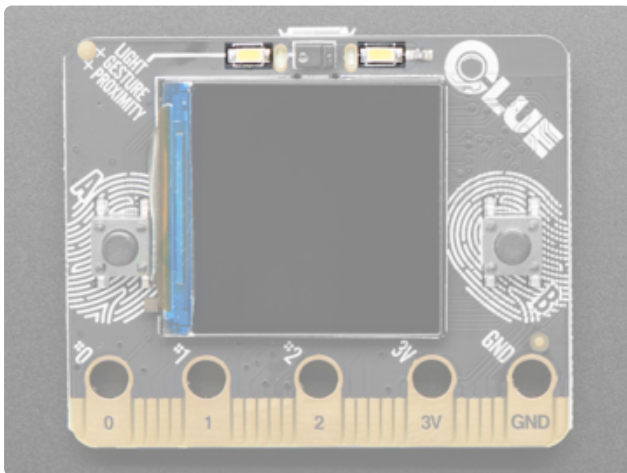
Bright white LEDs - On the front of the board are two bright white LEDs for illumination and color sensing. Control them in CircuitPython using `board.WHITE_LEDS`, and in Arduino using `D43`.

GPIO and Power Pads
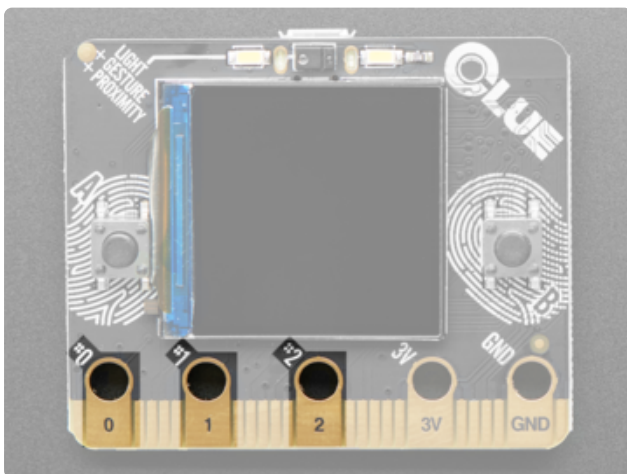
Pads 0, 1 and 2 - These pads are used for connecting external sensors etc, typically using alligator clips. They also work as inputs using capacitive touch. In CircuitPython they are `board.D0`, `board.D1`, and `board.D2`. In Arduino, they are `D0`, `D1` and `D2`.
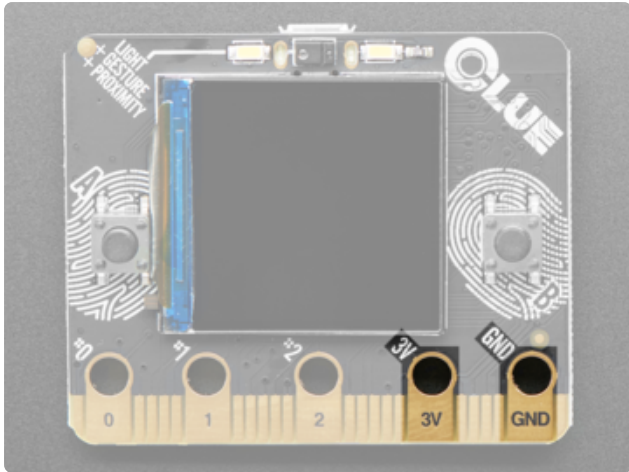
3V and GND - These are the power and ground pads used when connecting external sensors etc. typically using alligator clips.

Edge Connector
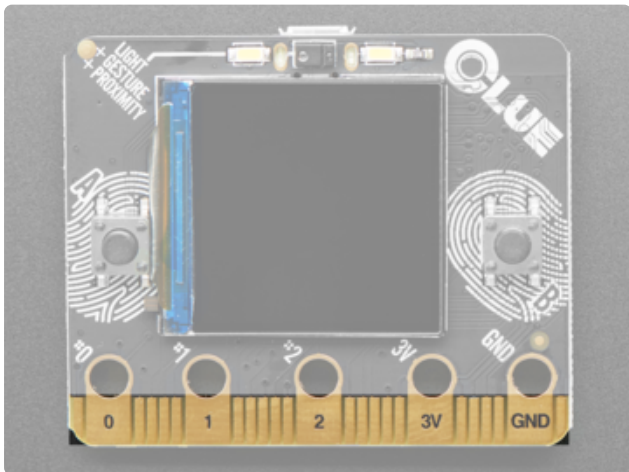
Micro:Bit compatible edge connector - This is the Micro:Bit compatible edge connector, used to break out all of the other features of this microcontroller. Compatible with other Micro:Bit-compatible hardware. While the CLUE is the same outline and we did our best to make the edge-connector pins match up, code may not be immediately compatible without adjustment, especially since only Arduino and CircuitPython are supported at this time.

Here's the pinout diagram for the micro:bit - the CLUE has the same pinout with some extras!

- The I2C pins are on on the same P19/P20 (we like to use D19/D20 naming)
- The SPI pins are on on the same P13-P15 (we like to use D13-D15 naming)
- There are analog pins on P0 (Arduino A2), P1 (Arduino A3), P2 (Arduino A4), P3 (Arduino A5), P4 (Arduino A6), P10 (Arduino A7) just like the micro:bit
- There are additional analog pins on D12 (Arduino A0) and P16 (Arduino A1)
- Button A and B are on the same P5 and P11 pins
- Since we don't have an LED matrix, you can use P3, P4, P6, P7, P9, P10, P11 without worrying about conflicting with an LED grid
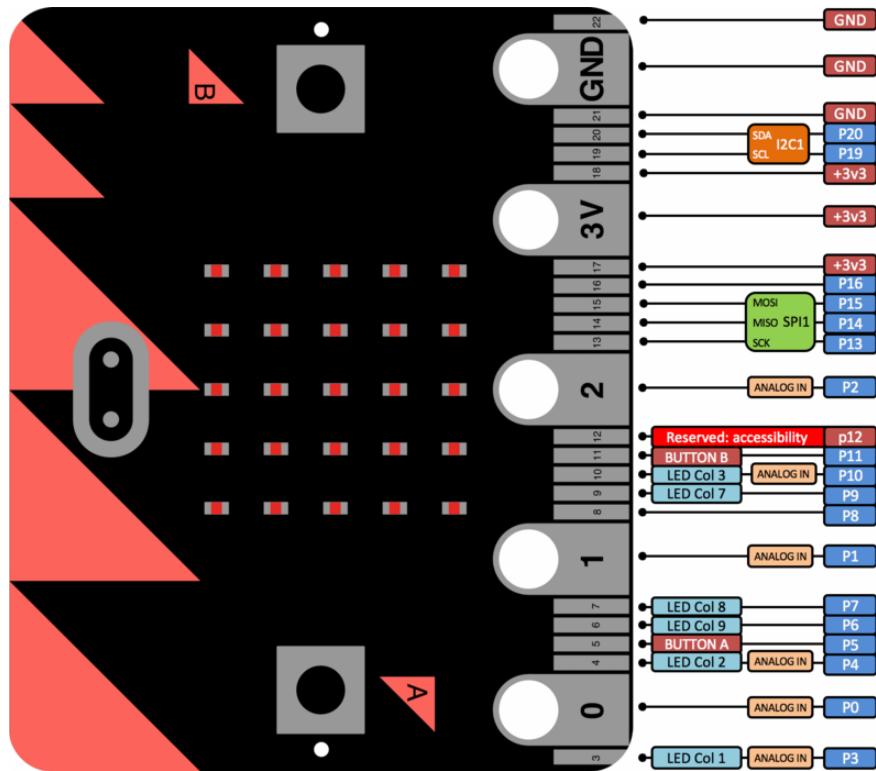
Debug Pads
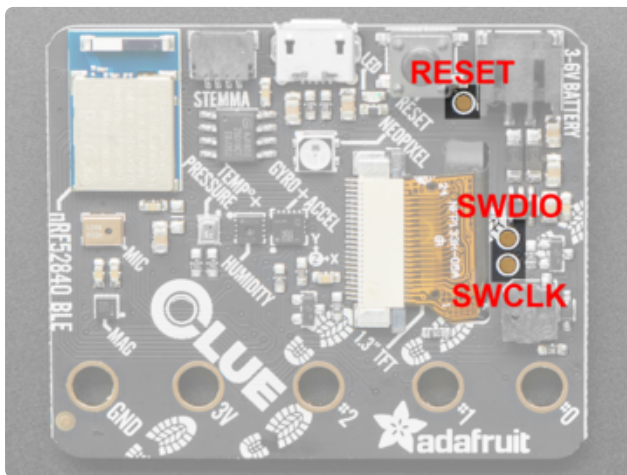
On the bottom of the board are three pads, one near the reset button, and two to the right of the display cable. The pad near the reset button is reset. Of the two pads near the display cable, the top is SWDIO and on the bottom is SWCLK. On the off chance you want to reprogram your CLUE or debug it using a debug/programmer, you will need to solder/connect to these pads.

Powering Your CLUE
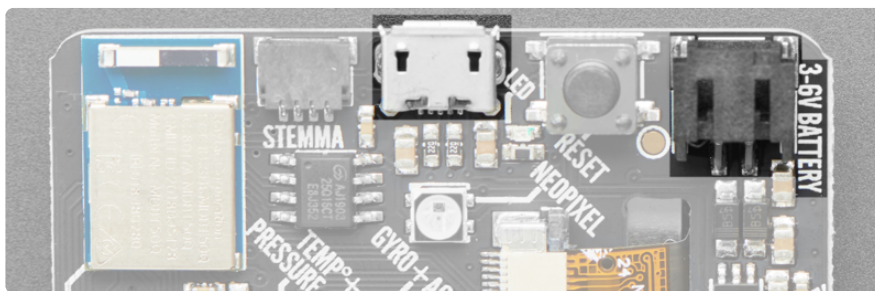
To use your CLUE board you'll have to provide it with a power source - and this is where CLUE is different than the micro:bit so we want to make it super clear to avoid confusion.

micro:bit Power

The BBC micro:bit can be powered from USB or it can be powered from a JST 2-PH battery connector in the corner. When powering from USB, use any USB power bank or port. When powering from the battery connector, there is no regulator and the voltage cannot be more than 3.3 volts. For that reason, the micro:bit folks warn users to:

- Only use 2 x AA or AAA battery holders with alkaline batteries for 2 x 1.5V = 3V power.

Can't use:

- You can't use 3 x AA because that will be 3 x 1.5 = 4.5 Volts - too much!
- You can't use 2 x AA NiMH rechargeable because that would be 2 x 1.2 = 2.4 - too low!
- You can't use 1 x LiPoly battery - when charged these provide 4.2V - too much!

CLUE Power

The Adafruit CLUE can also be powered from USB or it can be powered from a JST 2-PH battery connector in the corner. When powering from USB, use any USB power bank or port. When powering from the battery connector, you can use any battery from 3 to 6V because we have a regulator to safely bring the voltage to a safe level. Because of this, you can use:

- 3 x AA or AAA battery holders with alkaline or NiMH batteries for 3 x 1.2~1.5V = 3.6~4.5V power - recommended!
- 1 x LiPoly or LiIon battery. Just remember that the CLUE does not have built in battery charging so you will need to charge separately!

Not recommended (you can use them, we just don't suggest it)

- 2 x AA or AAA battery holders with alkaline batteries for 2 x 1.5V = 3V power. Not recommended, because the voltage will drop as the batteries die and you might get poor behavior.
- 4 x AA or AAA battery holders with NiMH batteries only! We think the voltage is a little high if you were to use Alkalines, and you may forget to use rechargeable, so we don't recommend it.

HELP! Accel/Gyro Not Working?

If you're getting 0's from the LSM6DS33 (Accelerometer/Gyro) - it's not broken! Some (not all) LSM's would lock up when we perform a firmware reset a certain way, which our original test code would do by default.

To fix, visit

https://learn.adafruit.com/adafruit-clue/arduino-test ()

to download the new UF2 and install it onto your CLUE (double click to enter BOOT mode, then drag the new UF2 over to the disk drive)

Update to the latest version of the Arduino/CircuitPython LSM6DS33 libraries to fix the bug.

Arduino Support Setup

You can install the Adafruit Bluefruit nRF52 BSP (Board Support Package) in two steps:

> nRF52 support requires at least Arduino IDE version 1.8.15! Please make sure you have an up to date version before proceeding with this guide!

> Please consult the FAQ section at the bottom of this page if you run into any problems installing or using this BSP!

1. BSP Installation

Recommended: Installing the BSP via the Board Manager

- [Download and install the Arduino IDE ()](#) (At least v1.8)
- Start the Arduino IDE
- Go into Preferences
- Add `https://adafruit.github.io/arduino-board-index/package_adafruit_index.json` as an 'Additional Board Manager URL' (see image below)
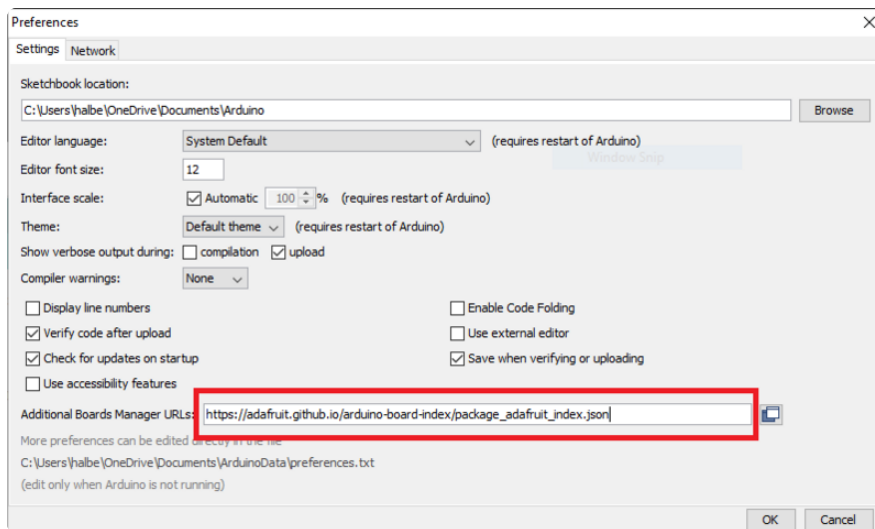
- Restart the Arduino IDE
- Open the Boards Manager option from the Tools -> Board menu and install 'Adafruit nRF52 by Adafruit' (see image below)
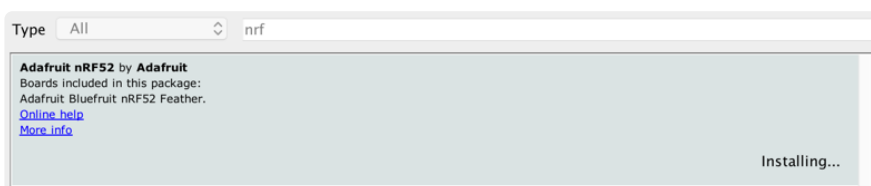
It will take up to a few minutes to finish installing the cross-compiling toolchain and tools associated with this BSP.

The delay during the installation stage shown in the image below is normal, please be patient and let the installation terminate normally:
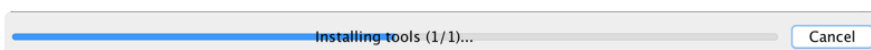
Once the BSP is installed, select

- Adafruit Bluefruit nRF52832 Feather (for the nRF52 Feather)
- Adafruit Bluefruit nRF52840 Feather Express (for the nRF52840 Feather)
- Adafruit ItsyBitsy nRF52840 (for the Itsy '850)
- Adafruit Circuit Playground Bluefruit (for the CPB)
- etc...

from the Tools -> Board menu, which will update your system config to use the right compiler and settings for the nRF52:
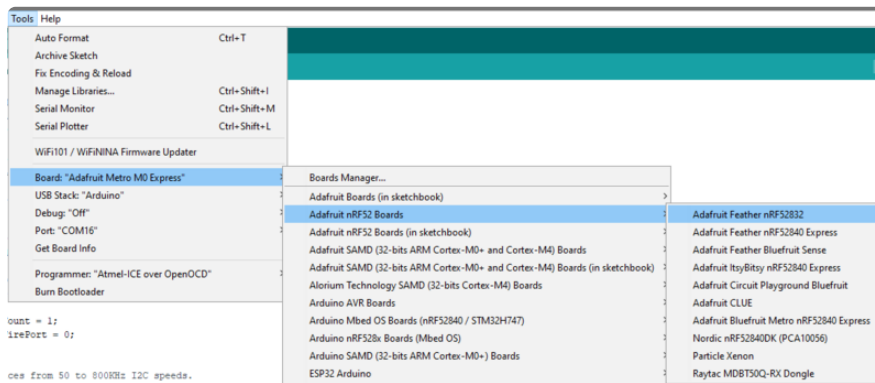
2. LINUX ONLY: adafruit-nrfutil Tool Installation

adafruit-nrfutil () is a modified version of Nordic's nrfutil (), which is used to flash boards using the built in serial bootloader. It is originally written for python2, but have been migrated to python3 and renamed to adafruit-nrfutil since BSP version 0.8.5.

> This step is only required on Linux, pre-built binaries of adafruit-nrfutil for Windows and MacOS are already included in the BSP. That should work out of the box for most setups.

Install python3 if it is not installed in your system already

```
$ sudo apt-get install python3
```

Then run the following command to install the tool from PyPi

```
$ pip3 install --user adafruit-nrfutil
```

Add pip3 installation dir to your PATH if it is not added already. Make sure adafruit-nrfutil can be executed in terminal by running

```
$ adafruit-nrfutil version
adafruit-nrfutil version 0.5.3.post12
```

3. Update the bootloader (nRF52832 ONLY)

To keep up with Nordic's SoftDevice advances, you will likely need to update your bootloader if you are using the original nRF52832 based Bluefruit nRF52 Feather boards.

Follow this link for instructions on how to do that

This step ISN'T required for the newer nRF52840 Feather Express, which has a different bootloader entirely!

Update the nRF52832 Bootloader

Advanced Option: Manually Install the BSP via 'git'

If you wish to do any development against the core codebase (generate pull requests, etc.), you can also optionally install the Adafruit nRF52 BSP manually using 'git', as decribed below:

Adafruit nRF52 BSP via git (for core development and PRs only)

1. Install BSP via Board Manager as above to install compiler & tools.
2. Delete the core folder nrf52 installed by Board Manager in Adruino15, depending on your OS. It could be
   macOS: `~/Library/Arduino15/packages/adafruit/hardware/nrf52`
   Linux: `~/.arduino15/packages/adafruit/hardware/nrf52`
   Windows: `%APPDATA%`
   `\Local\Arduino15\packages\adafruit\hardware\nrf52`
3. Go to the sketchbook folder on your command line, which should be one of the following:
   macOS: `~/Documents/Arduino`

Linux: `~/Arduino`
Windows: `~/Documents/Arduino`

4. Create a folder named `hardware/Adafruit`, if it does not exist, and change directories into it.
5. Clone the [Adafruit_nRF52_Arduino]() repo in the folder described in step 2:
   `git clone --recurse-submodules` [`git@github.com:adafruit/`](#) [`Adafruit_nRF52_Arduino.git`](#)

6. This should result in a final folder name like `~/Documents/Arduino/hardware/` `Adafruit/Adafruit_nRF52_Arduino` (macOS).

7. Restart the Arduino IDE

Arduino Board Testing

Once you have the Bluefruit nRF52 BSP setup on your system, you need to select the appropriate board, which will determine the compiler and expose some new menus options:

1. Select the Board Target

- Go to the Tools menu
- Select Tools > Board > Adafruit Bluefruit nRF52 Feather for nRF52832-based boards
- Select Tools > Board > Adafruit Bluefruit nRF52840 Feather Express for nRF528 40-based boards
- Select Tools > Board > Adafruit CLUE for the Adafruit CLUE
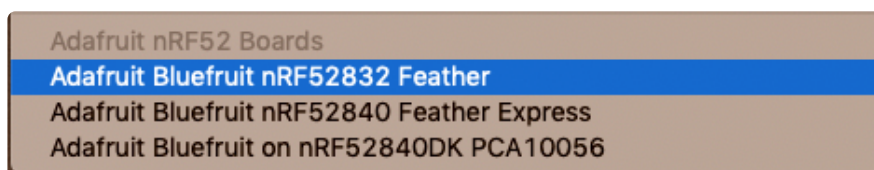
2. Select the USB CDC Serial Port

Finally, you need to set the serial port used by Serial Monitor and the serial bootloader:

- Go to Tools > Port and select the appropriate device
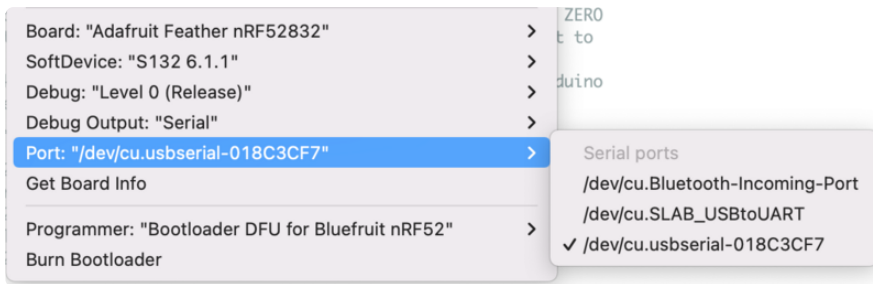
2.1 Download & Install CP2104 Driver (nRF52832)

For Feather nRF52832 If you don't see the serial ports device listed, you may need to install the SiLabs CP2104 driver () on your system.

On MacOS If you see this dialog message while installing driver
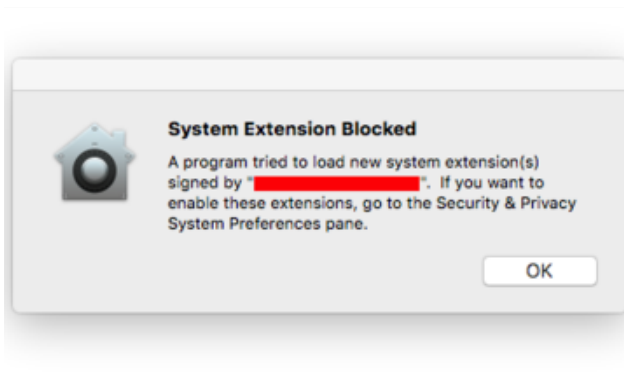
On MacOS If you see this dialog message while installing driver, System Extension Blocked

And cannot find the serial port of CP2104, it is highly possible that driver is blocked.
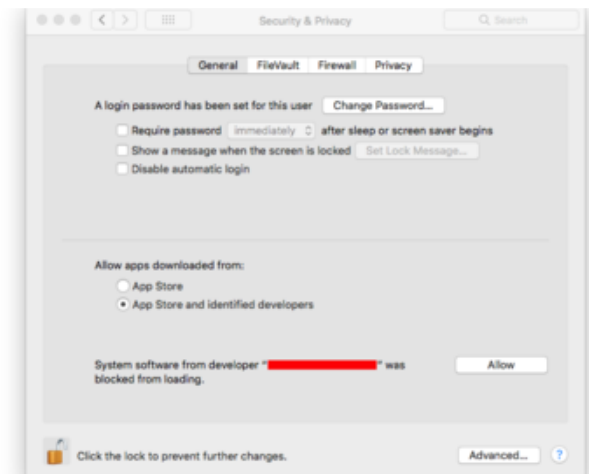
To enable it go to System Preferences -> Security & Privacy and click allow if you see Silab in the developer name.

After installing cp210x driver, If feather nRF52832 appear as 2 serial ports on your macos. e.g "/dev/cu.SLAB_USBtoUART" and "/dev/cu.usbserial-1234", the correct port to use is "/dev/cu.usbserial-1234"

2.2 Download & Install Adafruit Driver (nRF52840 Windows)

For Feather nRF52840, If you are using Windows, you will need to follows [Windows Driver Installation ()](#) to download and install driver.

3. Update the bootloader (nRF52832 Feather Only)

To keep up with Nordic's SoftDevice advances, you will likely need to update your bootloader

Follow this link for instructions on how to do that

> This step is only necessary on the nRF52832-based devices, NOT on the newer nRF52840 Feather Express.

<div style="text-align:center">

Update the Bootloader

</div>

4. Run a Test Sketch

At this point, you should be able to run a test sketch from the Examples folder, or just flash the following blinky code from the Arduino IDE:

> #include <Adafruit_TinyUSB.h> is required when using with nRF52840 based board for Serial port implementation.

```
#if defined(USE_TINYUSB)
#include <Adafruit_TinyUSB.h> // for Serial
#endif

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

This will blink the red LED beside the USB port on the Feather, or the red LED labeled "LED" by the corner of the USB connector on the CLUE.

Arduino sketch failed to compile with error: ld returned 1 exit status

If the sketch fails to compile and reports an error message like this:

`collect2.exe: error: ld returned 1 exit status`

`exit status 1`

`Compilation error: exit status 1`

Look in the rest of the compile message output (turn on verbose output if needed) and check if the actual error(s) look something like:

`undefined reference to ‘Adafruit_USBD_CDC::begin(unsigned long)`

`undefined reference to ‘Adafruit_USBD_CDC::write(unsigned char const*, unsigned int)`

If so, the required `#include` is missing. See the example code above and add the required lines to your sketch.

If Arduino failed to upload sketch to the Feather

If you get this error:

`Timed out waiting for acknowledgement from device.`

```
Failed to upgrade target. Error is: No data received on serial
port. Not able to proceed.
Traceback (most recent call last):
  File "nordicsemi\__main__.py", line 294, in serial
  File "nordicsemi\dfu\dfu.py", line 235, in dfu_send_images
  File "nordicsemi\dfu\dfu.py", line 203, in _dfu_send_image
  File "nordicsemi\dfu\dfu_transport_serial.py", line 155, in
send_init_packet
  File "nordicsemi\dfu\dfu_transport_serial.py", line 243, in
send_packet
  File "nordicsemi\dfu\dfu_transport_serial.py", line 282, in
get_ack_nr
```

`nordicsemi.exceptions.NordicSemiException: No data received on serial port. Not able to proceed.`

This is probably caused by the bootloader version mismatched on your feather and installed BSP. Due to the difference in flash layout (more details ()) and Softdevice API (which is bundled with bootloader), sketch built with selected bootloader can only upload to board having the same version. In short, you need to upgrade/burn bootloader to match on your Feather, follow above Update The Bootloader () guide

It only has to be done once to update your Feather

On Linux I'm getting 'arm-none-eabi-g++: no such file or directory', even though 'arm-none-eabi-g++' exists in the path specified. What should I do?

This is probably caused by a conflict between 32-bit and 64-bit versions of the compiler, libc and the IDE. The compiler uses 32-bit binaries, so you also need to have a 32-bit version of libc installed on your system (details ()). Try running the following commands from the command line to resolve this:

```
sudo dpkg --add-architecture i386

sudo apt-get update

sudo apt-get install libc6:i386
```

Arcada Libraries

OK now that you have Arduino IDE set up, drivers installed if necessary and you've practiced uploading code, you can start installing all the Libraries we'll be using to program it.

There's a lot of libraries!

Install Libraries

Open up the library manager...
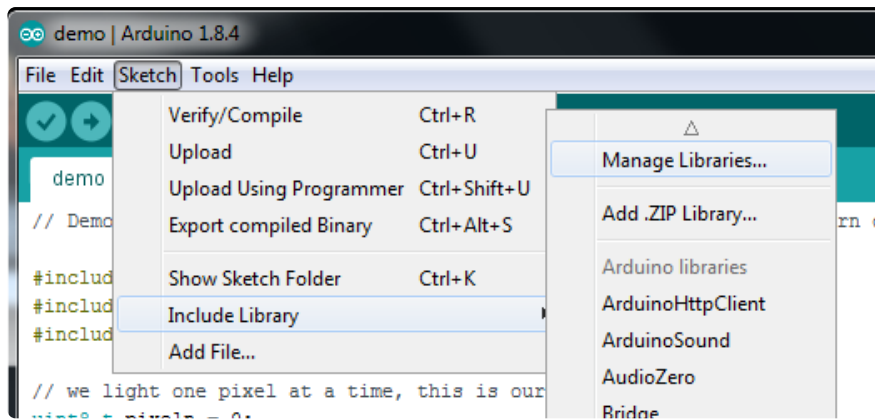
And install the following libraries:

Adafruit Arcada

This library generalizes the hardware for you so you can read the joystick, draw to the display, read files, etc. without having to worry about the underlying methods
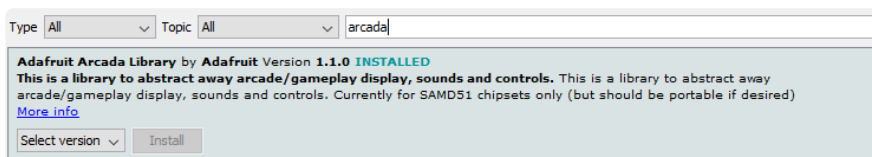
If you use Arduino 1.8.10 or later, the IDE will automagically install all the libraries you need to run all the Arcada demos when you install Arcada. We strongly recommend using the latest IDE so you don't miss one of the libraries!

If you aren't running Arduino IDE 1.8.10 or later, you'll need to install all of the following!

Adafruit NeoPixel

This will let you light up the status LEDs on the front/back
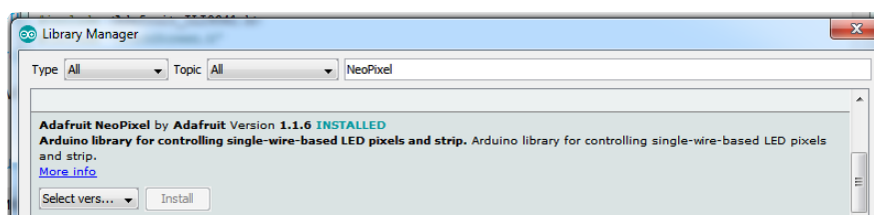
Adafruit FreeTouch

This is the open source version of QTouch for SAMD21 boards
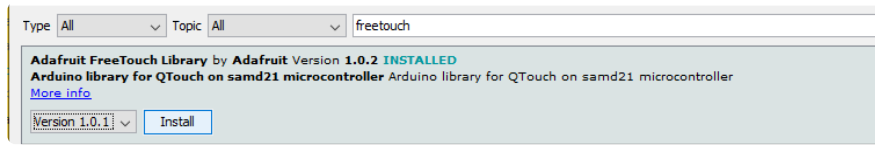
Adafruit Touchscreen

Used by Adafruit Arcada for touchscreen input (required even if your Arcada board does not have a touchscreen)
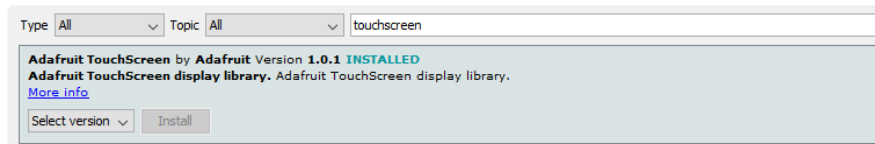
Adafruit SPIFlash

This will let you read/write to the onboard FLASH memory with super-fast QSPI support

Adafruit Zero DMA

This is used by the Graphics Library if you choose to use DMA
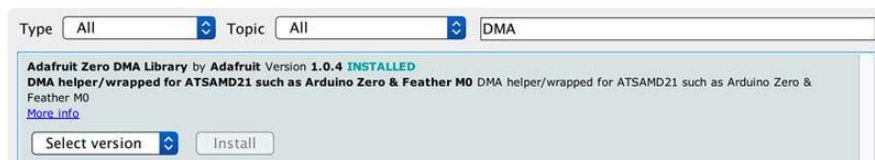
Adafruit GFX

This is the graphics library used to draw to the screen
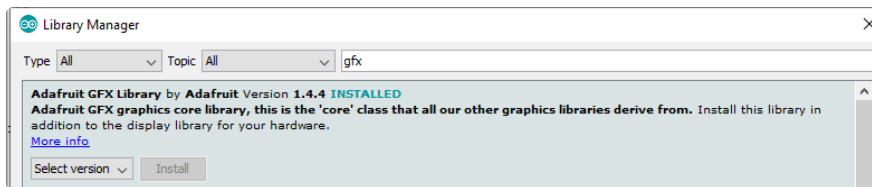
If using an older (pre-1.8.10) Arduino IDE, locate and install Adafruit_BusIO (newer versions do this one automatically).

Adafruit ST7735

The display on the PyBadge/PyGamer & other Arcada boards

Adafruit ILI9341

The display on the PyPortal & other Arcada boards
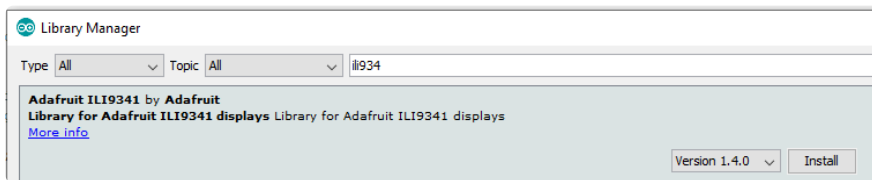
Adafruit LIS3DH

For reading the accelerometer data, required even if one is not on the board
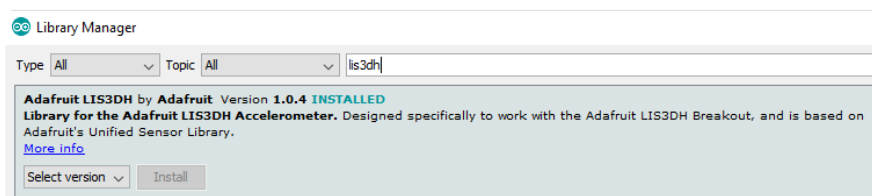
Adafruit Sensor

Needed by the LIS3DH Library, required even if one is not on the board
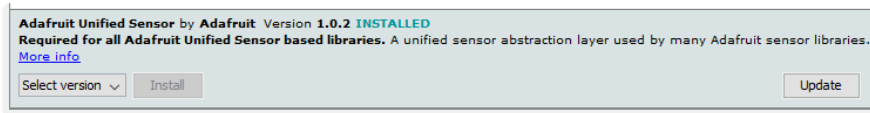
Adafruit ImageReader

For reading bitmaps from SPI Flash or SD and displaying
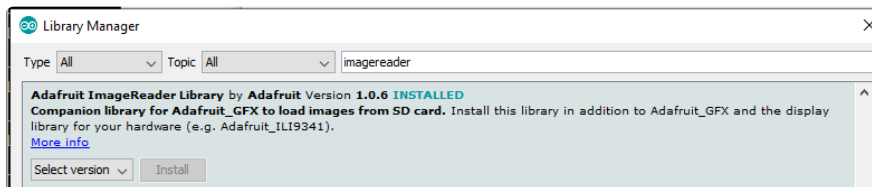
ArduinoJson

We use this library to read and write configuration files
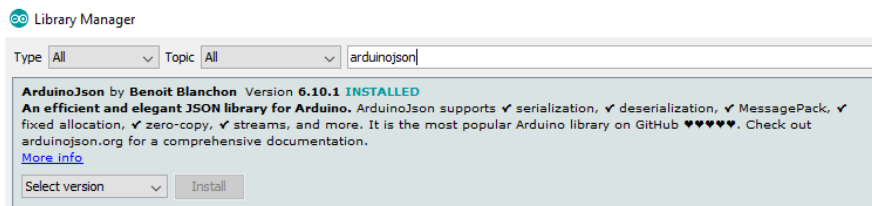
Adafruit ZeroTimer

We use this library to easily set timers and callbacks on the SAMD processors
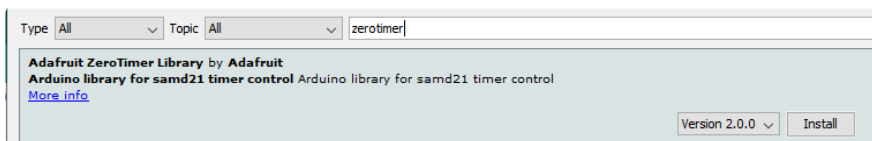
Adafruit TinyUSB

This lets us do cool stuff with USB like show up as a Keyboard or Disk Drive
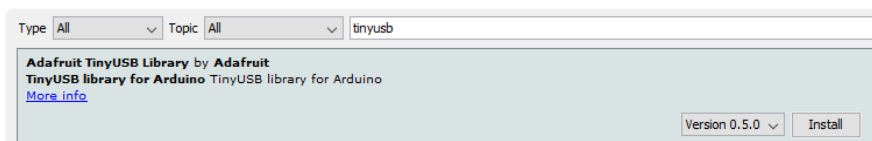
Adafruit WavePlayer

Helps us play .WAV sound files.
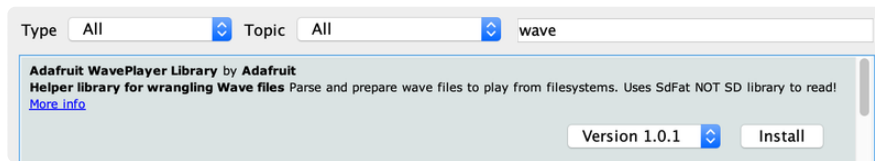
SdFat (Adafruit Fork)

The Adafruit fork of the really excellent SD card library that gives a lot more capability than the default SD library
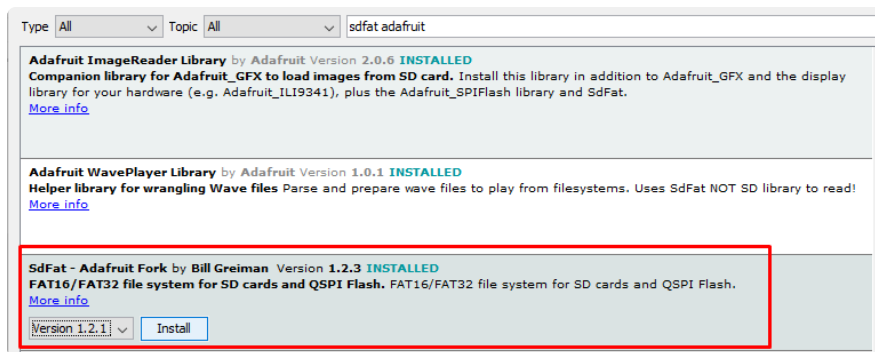
Audio - Adafruit Fork

Our fork of the Audio library provides a toolkit for building streaming audio projects.
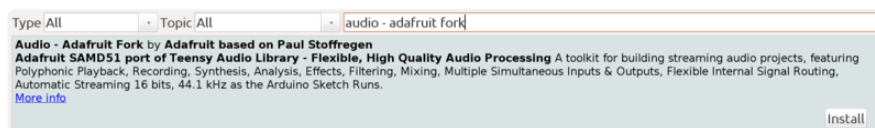
Sensor Libraries

To read and manage all the sensors on your CLUE board, we will need libraries to control each and every one of them.

Use the library manager to install them.
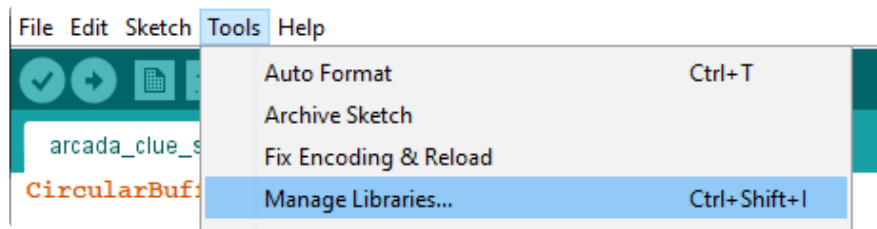
Adafruit Sensor Lab

This library generalizes sensor reading for you so you can search for and use various sensors without knowing the specifics - great for starting out with sensor readings in Arduino IDE

If you use Arduino 1.8.10 or later, the IDE will automagically install all the libraries you need to run all the sensor lab demos when you install Sensor Lab. We strongly recommend using the latest IDE so you don't miss one of the libraries!

If you aren't running Arduino IDE 1.8.10 or later, you'll need to install all of the following!

Search for and install the following:

Adafruit Unified Sensor
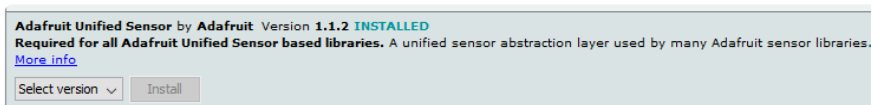
Adafruit ADXL343
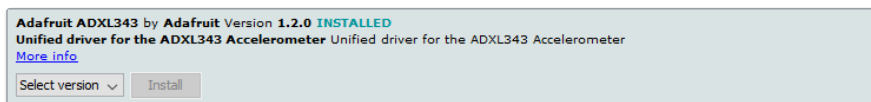
Adafruit APDS9660

Adafruit APDS9960 Library by **Adafruit** Version 1.1.2 INSTALLED
This is a library for the Adafruit APDS9960 gesture/proximity/color/light sensor. This is a library for the Adafruit APDS9960 gesture/proximity/color/light sensor.
More info

Adafruit BMP280

Adafruit BMP280 Library by **Adafruit** Version 2.0.1 INSTALLED
Arduino library for BMP280 sensors. Arduino library for BMP280 pressure and altitude sensors.
More info
Select version ∨ Install

Adafruit BME280

Adafruit BME280 Library by **Adafruit** Version 2.0.1 INSTALLED
Arduino library for BME280 sensors. Arduino library for BME280 humidity and pressure sensors.
More info
Select version ∨ Install

Adafruit DPS310

Adafruit DPS310 by **Adafruit** Version 1.0.2 INSTALLED
Library for the Adafruit DPS310 barometric pressure sensor. Designed specifically to work with the Adafruit DPS310 Breakout, and is based on Adafruit's Unified Sensor Library.
More info
Select version ∨ Install

Adafruit LIS2MDL

Adafruit LIS2MDL by **Adafruit** Version 2.1.1 INSTALLED
Unified Magnetometer sensor driver for Adafruit's LIS2MDL Breakout Unified Magnetometer sensor driver for Adafruit's LIS2MDL Breakout
More info
Select version ∨ Install

Adafruit LIS3MDL

Adafruit LIS3MDL by **Adafruit** Version 1.0.4 INSTALLED
Library for the Adafruit LIS3MDL magnetometer. Designed specifically to work with the Adafruit LIS3MDL Breakout, and is based on Adafruit's Unified Sensor Library.
More info
Select version ∨ Install   Update

Adafruit LSM6DS

Adafruit MSA301

Adafruit SHT31

Adafruit AHRS & Adafruit Sensor Calibration

Arduino Test

Once you've got the IDE installed and both Arcada and SensorLab libraries in place you can compile and run the test sketch. This will check all the hardware, and display it on the screen, its sort of a universal test because every part is checked. It's also a great reference if you want to know how to read the sensors or buttons, or control the screen.

If you don't want to compile the example, and want to just get to the hardware test, download CLUE_TEST.UF2 button here to download, and install it by double-clicking until you see a BOOT disk appear, then drag the UF2 over

CLUE_TEST.UF2

You can find it as an example in the Adafruit Arcada library (check the previous pages for all the libraries you need to install!)
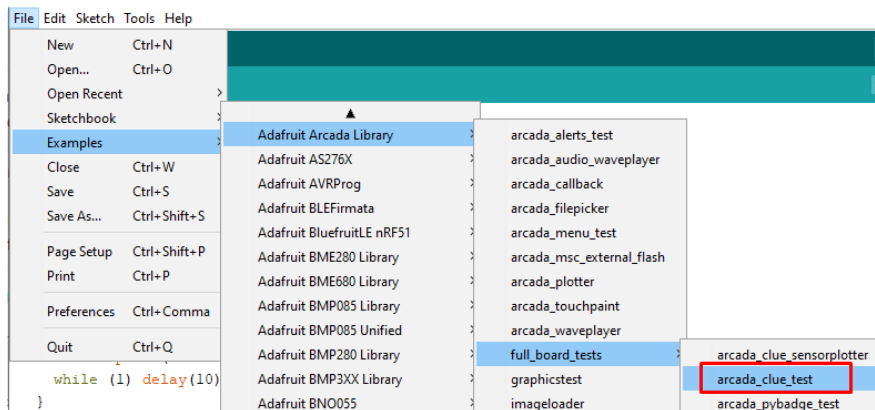
The test code

1. Checks the QSPI flash chip initialised correctly, and displays the manufacturer/ device ID if so
2. Checks if the QPI flash has a filesystem on it (if not, try loading CircuitPython which will create a filesystem)
3. Tests if all the sensors are found. You should see APDS9960 LSM6DS33 LISS3MDL SHT30 and BMP280 all in green text. If any are in red text, that means there was difficulty detecing the sensor. Try disconnecting it from power completely, waiting a few seconds, and plugging it back in.
4. Print the ambient temperature from the BMP280
5. Print the barometric pressure from the BMP280
6. Print the humidity from the SHT30
7. Print the light level from the APDS9960
8. Print the accelerometer output from the LSM6DS33
9. Print the gyroscope output from the LSM6DS33
10. Print the magnetometer output from the LIS3MDL
11. Print the audio level detected by the microphone (you can try blowing on the mic to see the number increase)
12. When the left button is held down, you will hear a beep to test the piezo
13. When the right button is held down, the front white LEDs will light up
14. The NeoPixel on the back will display colors in the rainbow
15. The red LED will pulse in and out.

To test Arcada's callback functionality, we pulse pin #13 red LED so you'll see it ramp up 4 times a second.
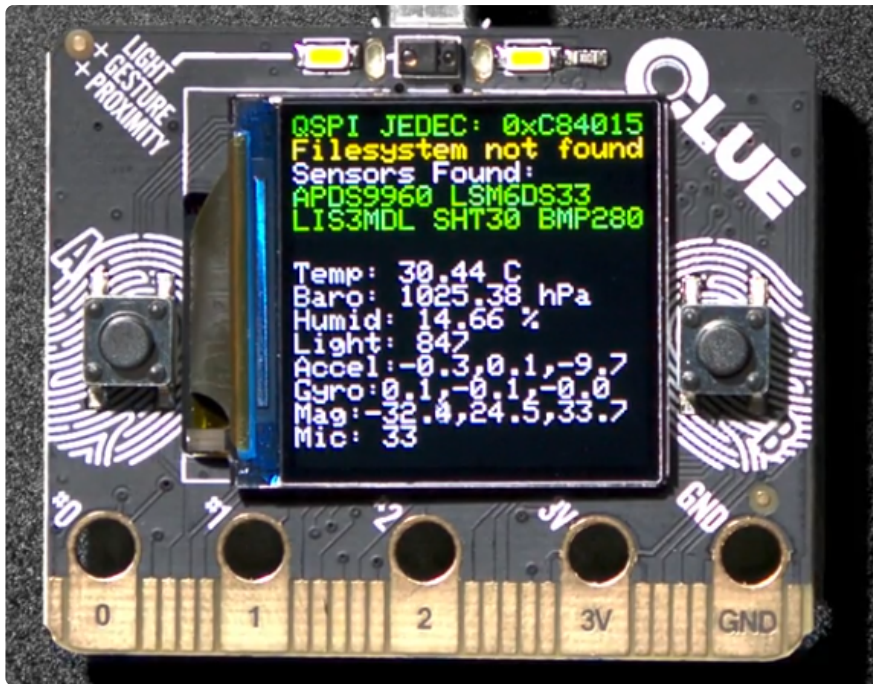
Animated GIF Player

The little 240x240 screen on the CLUE can be used to display simple animated GIFs, a great way to make a project from an existing GIF or video!

[We have a full guide on the animated GIF code here](), its an Arduino sketch (CircuitPython does not yet have the ability to play animated GIFs).

Here's the CLUE quickstart:

- Make sure you have a 'filesystem' on the QSPI flash. If you aren't sure, [simply load CircuitPython once, that will create the 2 MB disk drive]().
- Load the GIF player UF2 from this button:

> CLUE_gifplayer.UF2

- On the CIRCUITPY disk drive that appears, create a gifs folder, and drag the two demo GIFs from this zip into the CIRCUITPY/gifs folder.

> 240x240_demo_gifs.zip

Use the A and B buttons to go forward/back through the collection of gifs in the folder. For more info on how to configure and customize behavior - [check the guide!]()

Arduino Bluefruit nRF52 API

Arduino Bluefruit nRF52 API ()

Arduino BLE Examples

Arduino BLE Examples ()

CircuitPython on CLUE

CircuitPython () is a derivative of MicroPython () 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 CLUE from circuitpython.org

adafruit-circuitpython-clue_nrf52840_...0.0-beta.4.uf2
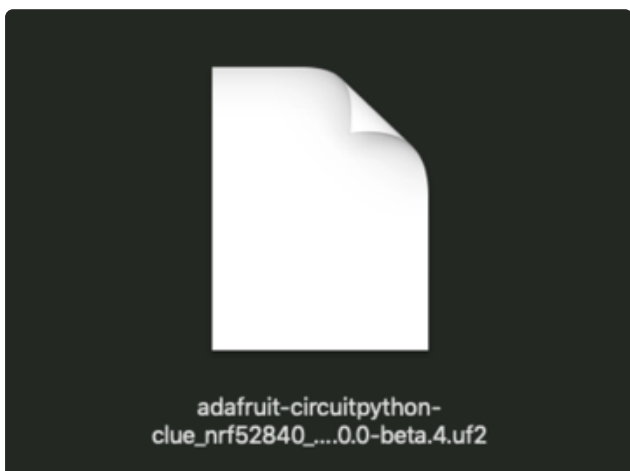
Click the link above to download the latest version of CircuitPython for the CLUE.

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

Plug your CLUE 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 (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!
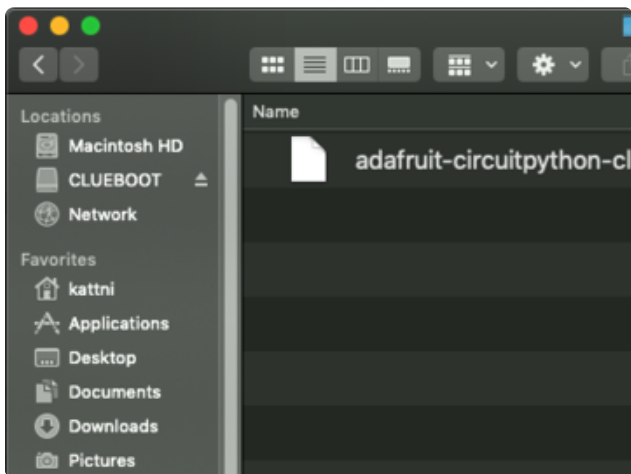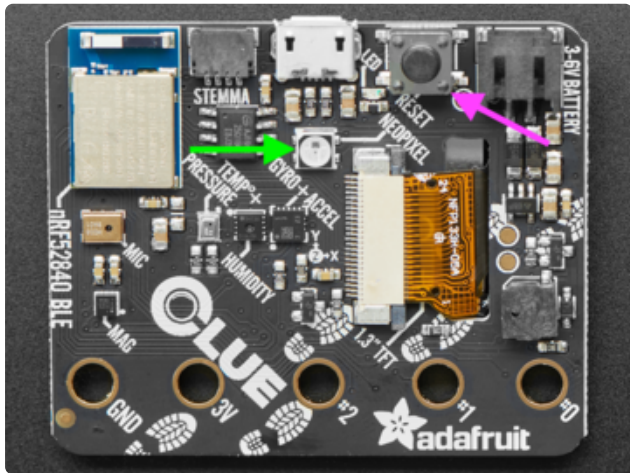
You will see a new disk drive appear called CLUEBOOT.

Drag the adafruit-circuitpython-clue-etc.uf2 file to CLUEBOOT.
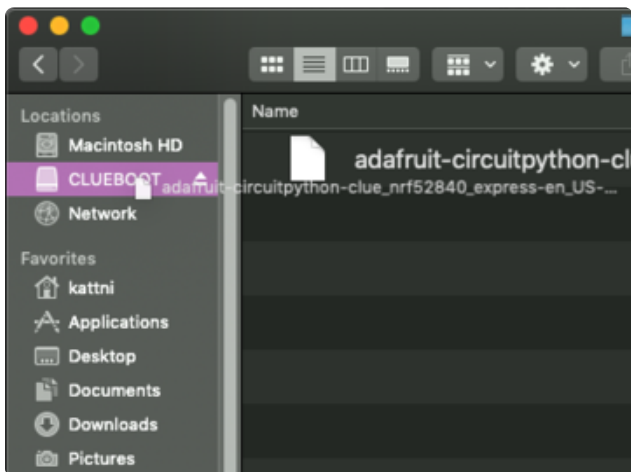
The LED will flash. Then, the CLUEBOOT drive will disappear and a new disk drive called CIRCUITPY will appear.
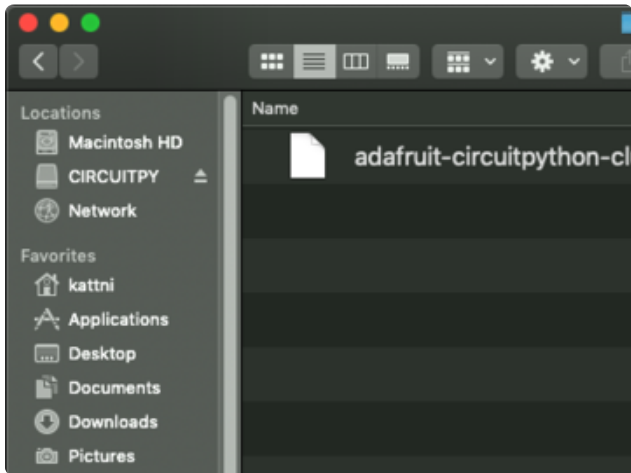
If this is the first time you're installing CircuitPython or you're doing a completely fresh install after erasing the filesystem, you will have two files - boot_out.txt, and code.py, and one folder - lib on your CIRCUITPY drive.

If CircuitPython was already installed, the files present before reloading CircuitPython should still be present on your CIRCUITPY drive. Loading CircuitPython will not create new files if there was already a CircuitPython filesystem present.

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

CLUE CircuitPython Libraries

The CLUE is packed full of features like a display and a ton of sensors. Now that you have CircuitPython installed on your CLUE, you'll need to install a base set of CircuitPython libraries to use the features of the board with CircuitPython.

Follow these steps to get the necessary libraries installed.

Installing CircuitPython Libraries on your CLUE

If you do not already have a lib folder on your CIRCUITPY drive, create one now.

Then, download the CircuitPython library bundle that matches your version of CircuitPython from CircuitPython.org.

Download the latest library bundle
from circuitpython.org
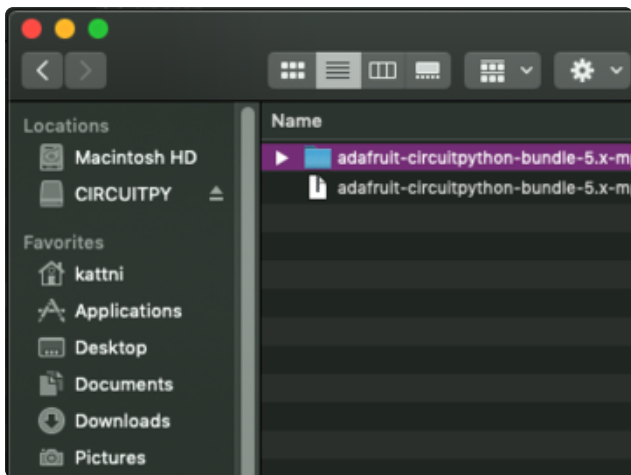
The bundle downloads as a .zip file.
Extract the file. Open the resulting folder.
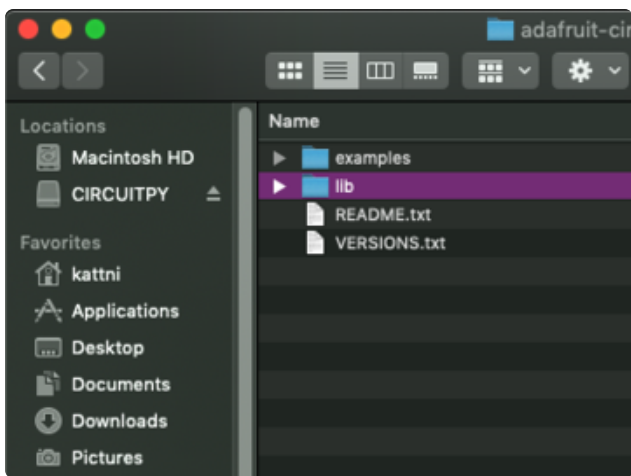
Open the lib folder found within.
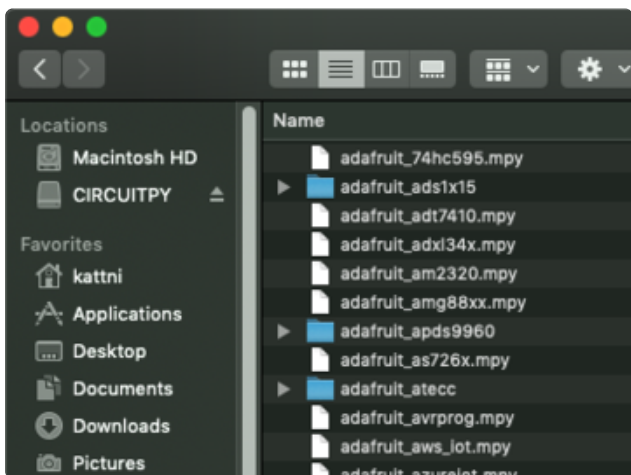
Once inside, you'll find a lengthy list of folders and .mpy files. To install a CircuitPython library, you drag the file or folder from the bundle lib folder to the lib folder on your CIRCUITPY drive.

Copy the following folders and files from the bundle lib folder to the lib folder on your CIRCUITPY drive:
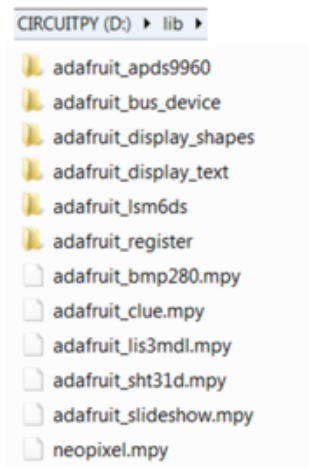
adafruit_apds9960
adafruit_bmp280.mpy
adafruit_bus_device
adafruit_clue.mpy
adafruit_display_shapes
adafruit_display_text
adafruit_lis3mdl.mpy
adafruit_lsm6ds
adafruit_register
adafruit_sht31d.mpy
adafruit_slideshow.mpy
neopixel.mpy

Your lib folder should look like the image on the left. These libraries will let you run the demos in the CLUE guide.

Getting Started with BLE and CircuitPython Guides

- Getting Started with CircuitPython and Bluetooth Low Energy () - Get started with CircuitPython, the Adafruit nRF52840 and the Bluefruit LE Connect app.
- BLE Light Switch with Feather nRF52840 and Crickit () - Control a robot finger from across the room to flip on and off the lights!
- Color Remote with Circuit Playground Bluefruit () - Mix NeoPixels wirelessly with a Bluetooth LE remote control!
- MagicLight Bulb Color Mixer with Circuit Playground Bluefruit () - Mix colors on a MagicLight Bulb wirelessly with a Bluetooth LE remote control.
- Bluetooth Turtle Bot with CircuitPython and Crickit () - Build your own Bluetooth controlled turtle rover!
- Wooden NeoPixel Xmas Tree () - Cut a Christmas tree of wood and mount some NeoPixels in the tree to create a festive yuletide light display.
- Bluefruit TFT Gizmo ANCS Notifier for iOS () - Circuit Playground Bluefruit displays your iOS notification icons so you know when there's fresh activity!
- Bluefruit Playground Hide and Seek () - Use Circuit Playground Bluefruit devices to create a colorful signal strength-based proximity detector!

- Snow Globe with Circuit Playground Bluefruit () - Make your own festive (or creatively odd!) snow globe with custom lighting effects and Bluetooth control.
- Bluetooth Controlled NeoPixel Lightbox () - Great for tracing and writing, this lightbox lets you adjust color and brightness with your phone.
- Circuit Playground Bluefruit NeoPixel Animation and Color Remote Control () - Control NeoPixel colors and animation remotely over Bluetooth with the Circuit Playground Bluefruit!
- Circuit Playground Bluetooth Cauldron () - Build a Bluetooth Controlled Light Up Cauldron.
- NeoPixel Badge Lanyard with Bluetooth LE () - Light up your convention badge and control colors with your phone!
- CircuitPython BLE Controlled NeoPixel Hat () - Wireless control NeoPixels on your wearables!
- Bluefruit nRF52 Feather Learning Guide () - Get started now with our most powerful Bluefruit board yet!
- CircusPython: Jump through Hoops with CircuitPython Bluetooth LE () - Blinka jumps through a ring of fire, controlled via Bluetooth LE and the Bluefruit LE Connect app!
- A CircuitPython BLE Remote Control On/Off Switch () - Make a remote control on/off switch for a computer with CircuitPython and BLE.
- NeoPixel Infinity Cube () - Build a 3D printed, Bluetooth controlled Mirrored Acrylic and NeoPixel Infinity cube.
- CircuitPython BLE Crickit Rover () - Purple Robot with Feather nRF52840 and Crickit plus NeoPixel underlighting!
- Circuit Playground Bluefruit Pumpkin with Lights and Sounds () - Add the Circuit Playground Bluefruit and STEMMA speaker to an inexpensive plastic pumpkin.
- No-Solder LED Disco Tie with Bluetooth () - Build an LED tie controlled by Bluetooth LE.
- Bluetooth Remote Control for the Lego Droid Developer Kit () - Reinvigorating the Lego Star Wars Droid Developer Kit with an Adafruit powered remote control using Bluetooth LE.

Installing the Mu Editor

Mu is a simple code editor that works with the Adafruit CircuitPython boards. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in so you get immediate feedback from your board's serial output!

Mu is our recommended editor - please use it (unless you are an experienced coder with a favorite editor already!).

Download and Install Mu

Download Mu from https://codewith.mu ().

Click the Download link for downloads and installation instructions.

Click Start Here to find a wealth of other information, including extensive tutorials and and how-to's.

Windows users: due to the nature of MSI installers, please remove old versions of Mu before installing the latest version.

Starting Up Mu
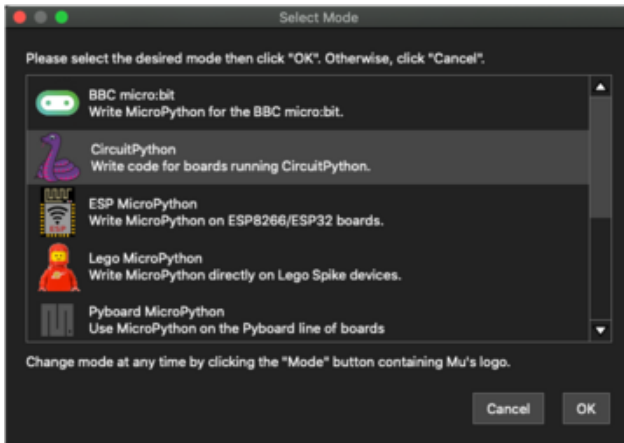
The first time you start Mu, you will be prompted to select your 'mode' - you can always change your mind later. For now please select CircuitPython!

The current mode is displayed in the lower right corner of the window, next to the "gear" icon. If the mode says "Microbit" or something else, click the Mode button in the upper left, and then choose "CircuitPython" in the dialog box that appears.
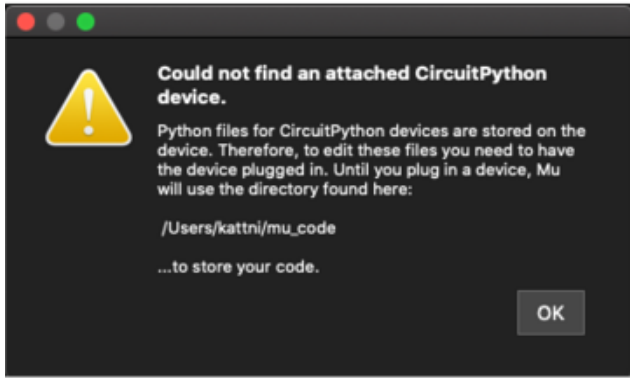
Mu attempts to auto-detect your board on startup, so if you do not have a CircuitPython board plugged in with a CIRCUITPY drive available, Mu will inform you where it will store any code you save until you plug in a board.

To avoid this warning, plug in a board and ensure that the CIRCUITPY drive is mounted before starting Mu.

Using Mu

You can now explore Mu! The three main sections of the window are labeled below; the button bar, the text editor, and the serial console / REPL.
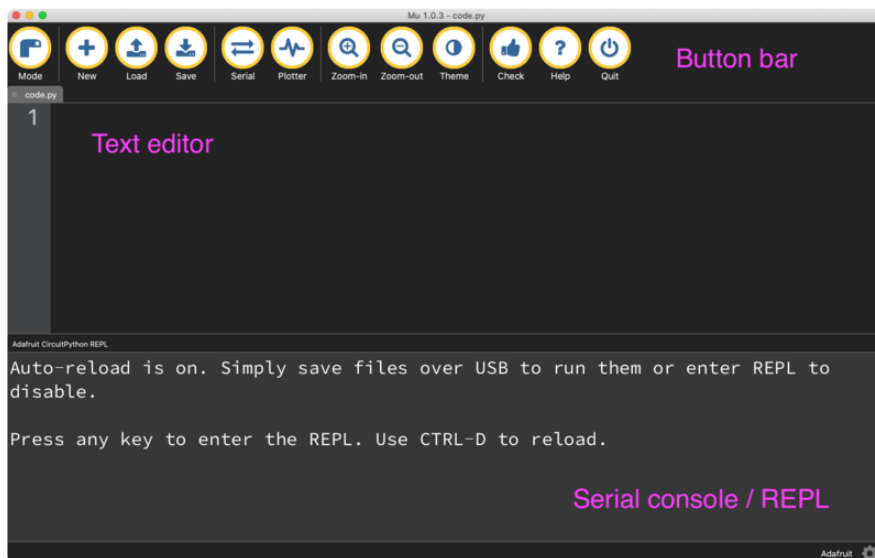
Now you're ready to code! Let's keep going...

Creating and Editing Code

One of the best things about CircuitPython is how simple it is to get code up and running. This section covers how to create and edit your first CircuitPython program.

To create and edit code, all you'll need is an editor. There are many options. Adafruit strongly recommends using Mu! It's designed for CircuitPython, and it's really simple and easy to use, with a built in serial console!

If you don't or can't use Mu, there are a number of other editors that work quite well. The Recommended Editors page () has more details. Otherwise, make sure you do "Eject" or "Safe Remove" on Windows or "sync" on Linux after writing a file if you aren't using Mu. (This is not a problem on MacOS.)

Creating Code
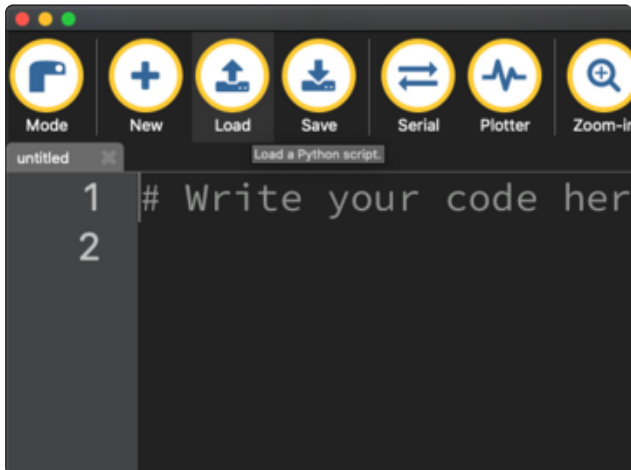
Installing CircuitPython generates a code.py file on your CIRCUITPY drive. To begin your own program, open your editor, and load the code.py file from the CIRCUITPY drive.

If you are using Mu, click the Load button in the button bar, navigate to the CIRCUITPY drive, and choose code.py.

Copy and paste the following code into your editor:
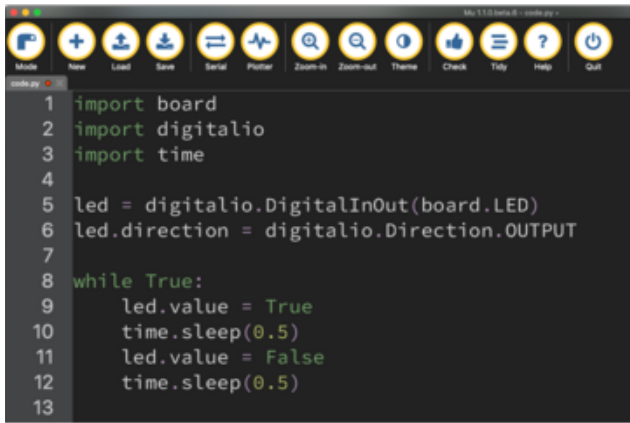
```python
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

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

The KB2040, QT Py and the Trinkeys do not have a built-in little red LED! There is an addressable RGB NeoPixel LED. The above example will NOT work on the KB2040, QT Py or the Trinkeys!
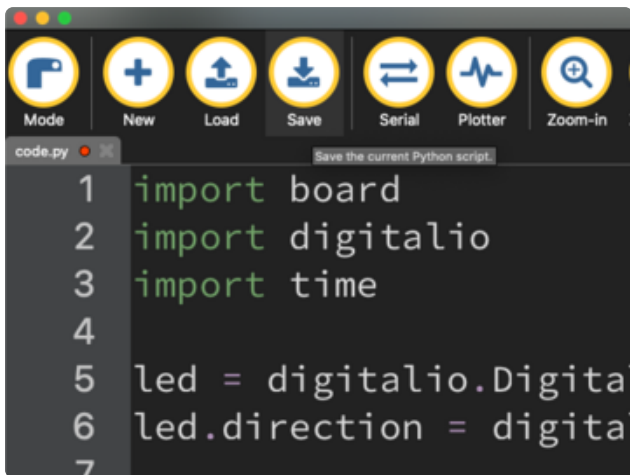
If you're using a KB2040, QT Py or a Trinkey, please download the NeoPixel blink example ().

The NeoPixel blink example uses the onboard NeoPixel, but the time code is the same. You can use the linked NeoPixel Blink example to follow along with this guide page.

It will look like this. Note that under the `while True:` line, the next four lines begin with four spaces to indent them, and they're indented exactly the same amount. All the lines before that have no spaces before the text.

Save the code.py file on your CIRCUITPY drive.

The little LED should now be blinking. Once per half-second.

Congratulations, you've just run your first CircuitPython program!

> On most boards you'll find a tiny red LED.
>
> On the ItsyBitsy nRF52840, you'll find a tiny blue LED.
>
> On QT Py M0, QT Py RP2040, and the Trinkey series, you will find only an RGB NeoPixel LED.

Editing Code
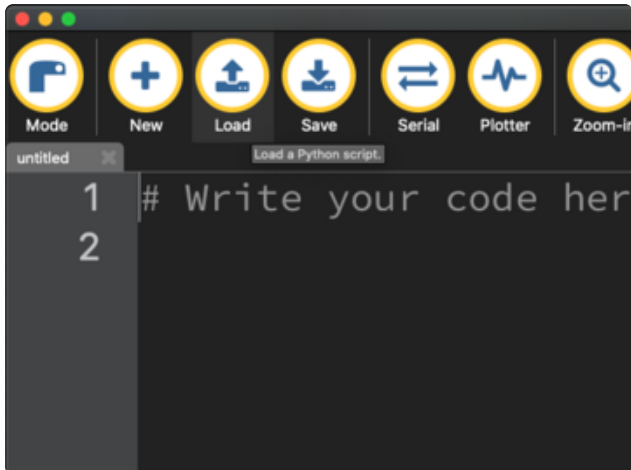
To edit code, open the code.py file on your CIRCUITPY drive into your editor.

Make the desired changes to your code. Save the file. That's it!

Your code changes are run as soon as the file is done saving.

There's one warning before you continue...

> ### Don't click reset or unplug your board!

The CircuitPython code on your board detects when the files are changed or written and will automatically re-start your code. This makes coding very fast because you save, and it re-runs. If you unplug or reset the board before your computer finishes writing the file to your board, you can corrupt the drive. If this happens, you may lose the code you've written, so it's important to backup your code to your computer regularly.

There are a couple of ways to avoid filesystem corruption.

1. Use an editor that writes out the file completely when you save it.

Check out the Recommended Editors page () for details on different editing options.

> If you are dragging a file from your host computer onto the CIRCUITPY drive, you still need to do step 2. Eject or Sync (below) to make sure the file is completely written.

2. Eject or Sync the Drive After Writing

If you are using one of our not-recommended-editors, not all is lost! You can still make it work.

On Windows, you can Eject or Safe Remove the CIRCUITPY drive. It won't actually eject, but it will force the operating system to save your file to disk. On Linux, use the sync command in a terminal to force the write to disk.

You also need to do this if you use Windows Explorer or a Linux graphical file manager to drag a file onto CIRCUITPY.

Oh No I Did Something Wrong and Now The CIRCUITPY Drive Doesn't Show Up!!!

Don't worry! Corrupting the drive isn't the end of the world (or your board!). If this happens, follow the steps found on the Troubleshooting () page of every board guide to get your board up and running again.

Back to Editing Code...

Now! Let's try editing the program you added to your board. Open your code.py file into your editor. You'll make a simple change. Change the first `0.5` to `0.1`. The code should look like this:

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
    time.sleep(0.1)
    led.value = False
    time.sleep(0.5)
```

Leave the rest of the code as-is. Save your file. See what happens to the LED on your board? Something changed! Do you know why?

You don't have to stop there! Let's keep going. Change the second `0.5` to `0.1` so it looks like this:

```
while True:
    led.value = True
```

```
    time.sleep(0.1)
    led.value = False
    time.sleep(0.1)
```

Now it blinks really fast! You decreased the both time that the code leaves the LED on and off!

Now try increasing both of the `0.1` to `1`. Your LED will blink much more slowly because you've increased the amount of time that the LED is turned on and off.

Well done! You're doing great! You're ready to start into new examples and edit them to see what happens! These were simple changes, but major changes are done using the same process. Make your desired change, save it, and get the results. That's really all there is to it!

Naming Your Program File

CircuitPython looks for a code file on the board to run. There are four options: code.txt, code.py, main.txt and main.py. CircuitPython looks for those files, in that order, and then runs the first one it finds. While code.py is the recommended name for your code file, it is important to know that the other options exist. If your program doesn't seem to be updating as you work, make sure you haven't created another code file that's being read instead of the one you're working on.

Connecting to the Serial Console

One of the staples of CircuitPython (and programming in general!) is something called a "print statement". This is a line you include in your code that causes your code to output text. A print statement in CircuitPython (and Python) looks like this:

```
print("Hello, world!")
```

This line in your code.py would result in:

```
Hello, world!
```

However, these print statements need somewhere to display. That's where the serial console comes in!

The serial console receives output from your CircuitPython board sent over USB and displays it so you can see it. This is necessary when you've included a print statement in your code and you'd like to see what you printed. It is also helpful for

troubleshooting errors, because your board will send errors and the serial console will display those too.

The serial console requires an editor that has a built in terminal, or a separate terminal program. A terminal is a program that gives you a text-based interface to perform various tasks.

Are you using Mu?

If so, good news! The serial console is built into Mu and will autodetect your board making using the serial console really really easy.
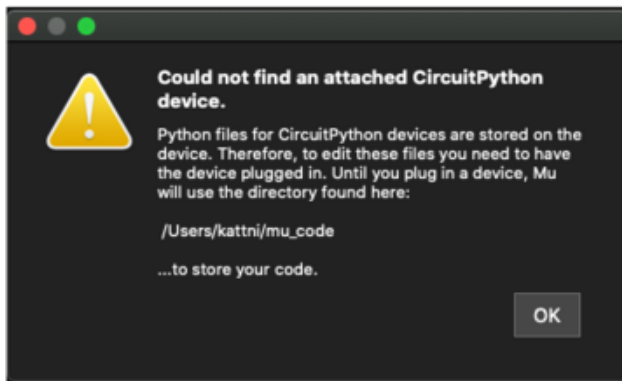
First, make sure your CircuitPython board is plugged in.

If you open Mu without a board plugged in, you may encounter the error seen here, letting you know no CircuitPython board was found and indicating where your code will be stored until you plug in a board.

If you are using Windows 7, make sure you installed the drivers ().

Once you've opened Mu with your board plugged in, look for the Serial button in the button bar and click it.
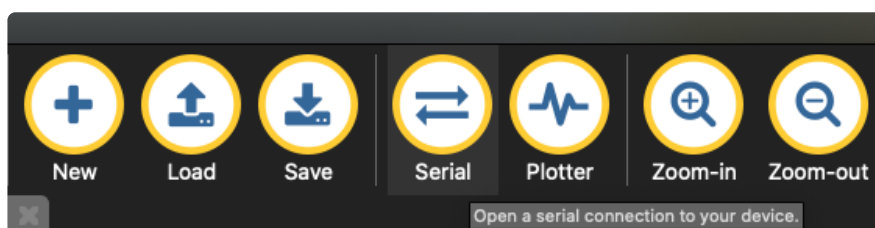
The Mu window will split in two, horizontally, and display the serial console at the bottom.
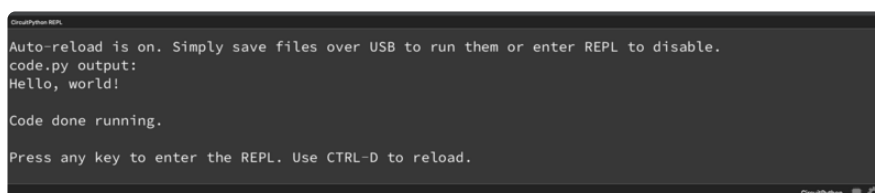
> If nothing appears in the serial console, it may mean your code is done running or has no print statements in it. Click into the serial console part of Mu, and press CTRL+D to reload.

Serial Console Issues or Delays on Linux

If you're on Linux, and are seeing multi-second delays connecting to the serial console, or are seeing "AT" and other gibberish when you connect, then the `modemmanager` service might be interfering. Just remove it; it doesn't have much use unless you're still using dial-up modems.

To remove `modemmanager`, type the following command at a shell:

```
sudo apt purge modemmanager
```

Setting Permissions on Linux

On Linux, if you see an error box something like the one below when you press the Serial button, you need to add yourself to a user group to have permission to connect to the serial console.

On Ubuntu and Debian, add yourself to the dialout group by doing:

```
sudo adduser $USER dialout
```

After running the command above, reboot your machine to gain access to the group. On other Linux distributions, the group you need may be different. See the [Advanced Serial Console on Linux ()](#) for details on how to add yourself to the right group.

Using Something Else?
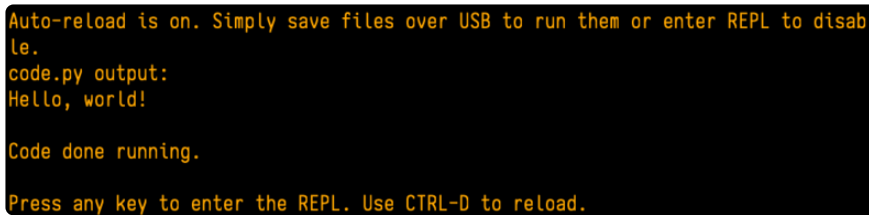
If you're not using Mu to edit, are using or if for some reason you are not a fan of its built in serial console, you can run the serial console from a separate program.

Windows requires you to download a terminal program. [Check out the Advanced Serial Console on Windows page for more details. ()](#)

MacOS has Terminal built in, though there are other options available for download. [Check the Advanced Serial Console on Mac page for more details. ()](#)

Linux has a terminal program built in, though other options are available for download. [Check the Advanced Serial Console on Linux page for more details. ()](#)

Once connected, you'll see something like the following.

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Hello, world!

Code done running.

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

Interacting with the Serial Console

Once you've successfully connected to the serial console, it's time to start using it.

The code you wrote earlier has no output to the serial console. So, you're going to edit it to create some output.

Open your code.py file into your editor, and include a `print` statement. You can print anything you like! Just include your phrase between the quotation marks inside the parentheses. For example:
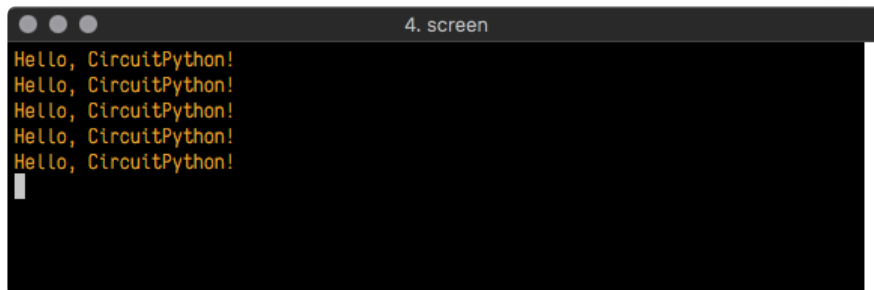
```python
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    print("Hello, CircuitPython!")
    led.value = True
    time.sleep(1)
```

```
    led.value = False
    time.sleep(1)
```

Save your file.

Now, let's go take a look at the window with our connection to the serial console.
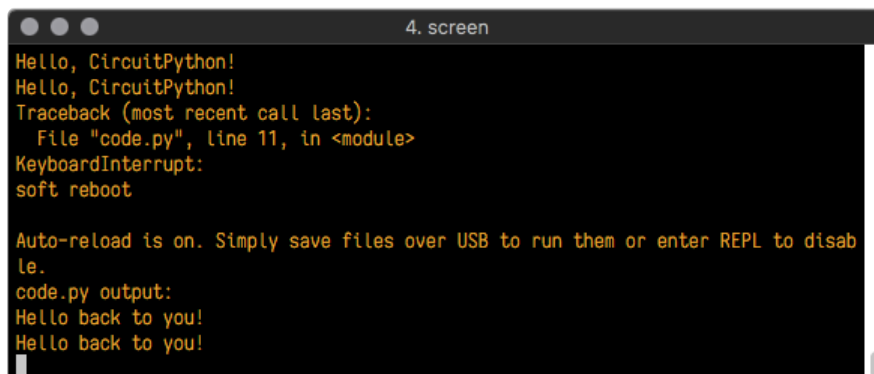
Excellent! Our print statement is showing up in our console! Try changing the printed text to something else.

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    print("Hello back to you!")
    led.value = True
    time.sleep(1)
    led.value = False
    time.sleep(1)
```

Keep your serial console window where you can see it. Save your file. You'll see what the serial console displays when the board reboots. Then you'll see your new change!

The `Traceback (most recent call last):` is telling you the last thing your board was doing before you saved your file. This is normal behavior and will happen every time the board resets. This is really handy for troubleshooting. Let's introduce an error so you can see how it is used.

Delete the `e` at the end of `True` from the line `led.value = True` so that it says `led.value = Tru`
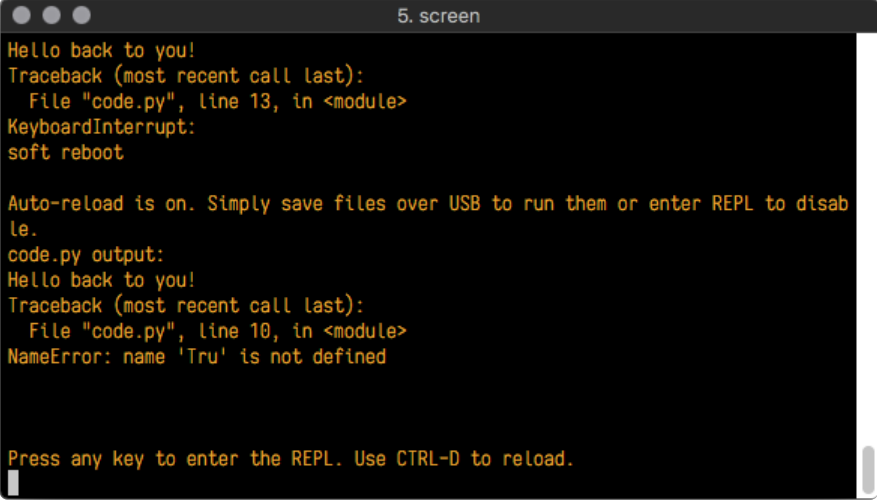
```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    print("Hello back to you!")
    led.value = Tru
    time.sleep(1)
    led.value = False
    time.sleep(1)
```

Save your file. You will notice that your red LED will stop blinking, and you may have a colored status LED blinking at you. This is because the code is no longer correct and can no longer run properly. You need to fix it!
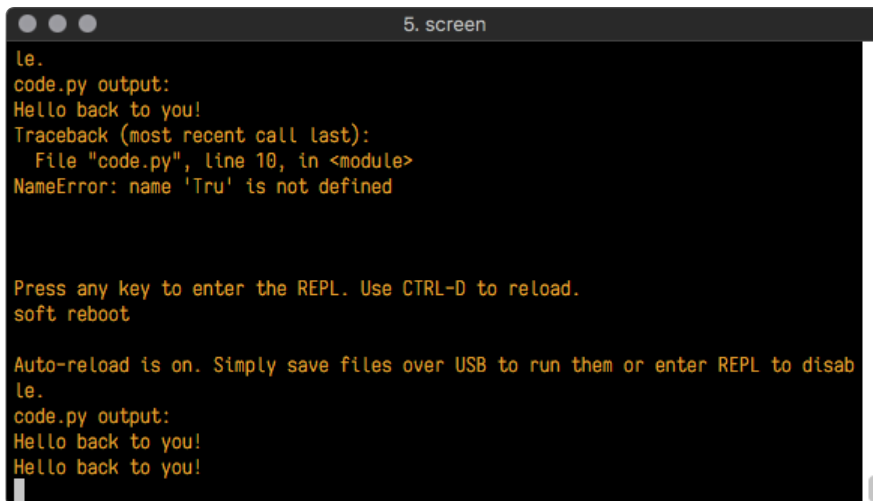
Usually when you run into errors, it's not because you introduced them on purpose. You may have 200 lines of code, and have no idea where your error could be hiding. This is where the serial console can help. Let's take a look!

The `Traceback (most recent call last):` is telling you that the last thing it was able to run was `line 10` in your code. The next line is your error: `NameError: name 'Tru' is not defined`. This error might not mean a lot to you, but combined with knowing the issue is on line 10, it gives you a great place to start!

Go back to your code, and take a look at line 10. Obviously, you know what the problem is already. But if you didn't, you'd want to look at line 10 and see if you could figure it out. If you're still unsure, try googling the error to get some help. In this case, you know what to look for. You spelled True wrong. Fix the typo and save your file.

```
le.
code.py output:
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 10, in <module>
NameError: name 'Tru' is not defined


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

Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.
code.py output:
Hello back to you!
Hello back to you!
```

Nice job fixing the error! Your serial console is streaming and your red LED Is blinking again.

The serial console will display any output generated by your code. Some sensors, such as a humidity sensor or a thermistor, receive data and you can use print statements to display that information. You can also use print statements for troubleshooting, which is called "print debugging". Essentially, if your code isn't working, and you want to know where it's failing, you can put print statements in various places to see where it stops printing.

The serial console has many uses, and is an amazing tool overall for learning and programming!

The REPL
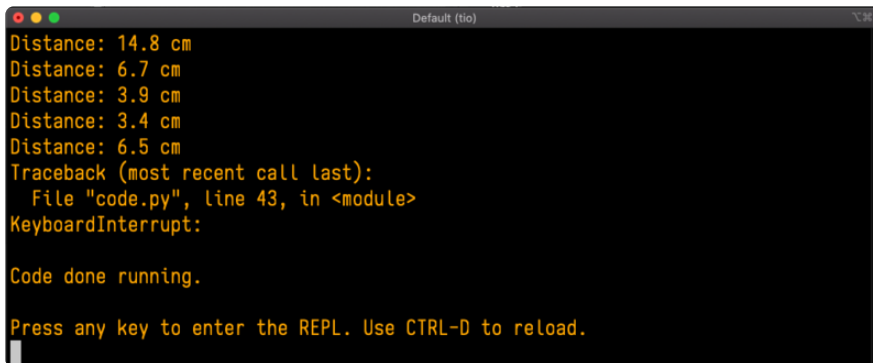
The other feature of the serial connection is the Read-Evaluate-Print-Loop, or REPL. The REPL allows you to enter individual lines of code and have them run immediately. It's really handy if you're running into trouble with a particular program and can't figure out why. It's interactive so it's great for testing new ideas.

Entering the REPL

To use the REPL, you first need to be connected to the serial console. Once that connection has been established, you'll want to press CTRL+C.

If there is code running, in this case code measuring distance, it will stop and you'll see `Press any key to enter the REPL. Use CTRL-D to reload.` Follow those instructions, and press any key on your keyboard.

The `Traceback (most recent call last):` is telling you the last thing your board was doing before you pressed Ctrl + C and interrupted it. The `KeyboardInterrupt` is you pressing CTRL+C. This information can be handy when troubleshooting, but for now, don't worry about it. Just note that it is expected behavior.
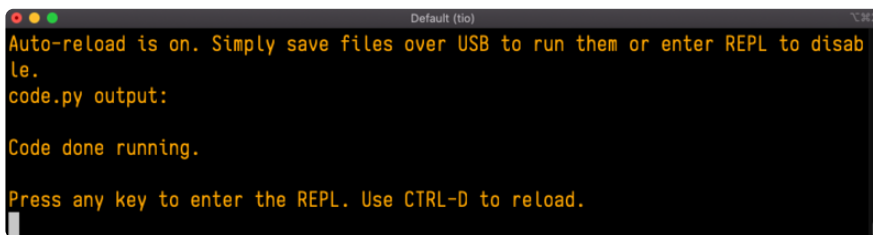
```
Distance: 14.8 cm
Distance: 6.7 cm
Distance: 3.9 cm
Distance: 3.4 cm
Distance: 6.5 cm
Traceback (most recent call last):
  File "code.py", line 43, in <module>
KeyboardInterrupt:

Code done running.

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

If your code.py file is empty or does not contain a loop, it will show an empty output and `Code done running.` . There is no information about what your board was doing before you interrupted it because there is no code running.
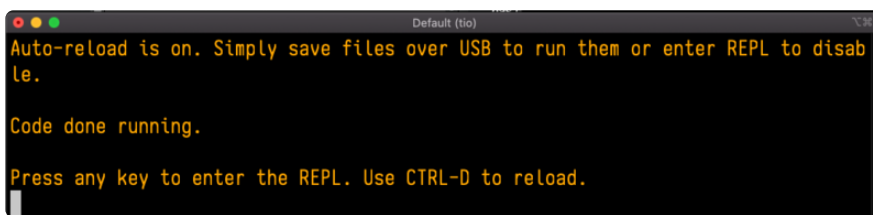
```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:

Code done running.

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

If you have no code.py on your CIRCUITPY drive, you will enter the REPL immediately after pressing CTRL+C. Again, there is no information about what your board was doing before you interrupted it because there is no code running.
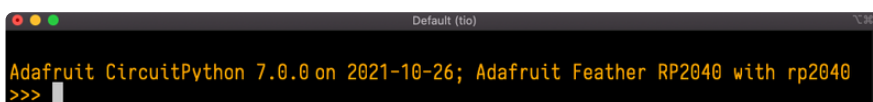
```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Code done running.

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

Regardless, once you press a key you'll see a `>>>` prompt welcoming you to the REPL!

```
Adafruit CircuitPython 7.0.0 on 2021-10-26; Adafruit Feather RP2040 with rp2040
>>>
```

If you have trouble getting to the `>>>` prompt, try pressing Ctrl + C a few more times.

The first thing you get from the REPL is information about your board.

`Adafruit CircuitPython 7.0.0 on 2021-10-26; Adafruit Feather RP2040 with rp2040`

This line tells you the version of CircuitPython you're using and when it was released. Next, it gives you the type of board you're using and the type of microcontroller the board uses. Each part of this may be different for your board depending on the versions you're working with.
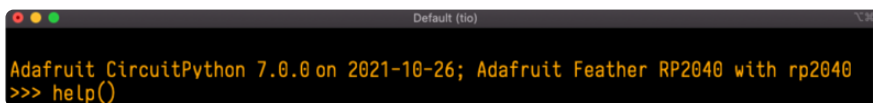
This is followed by the CircuitPython prompt.

`>>>`

Interacting with the REPL
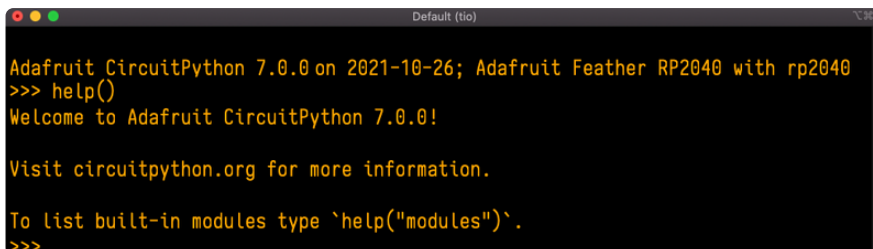
From this prompt you can run all sorts of commands and code. The first thing you'll do is run `help()`. This will tell you where to start exploring the REPL. To run code in the REPL, type it in next to the REPL prompt.

Type `help()` next to the prompt in the REPL.

Then press enter. You should then see a message.

First part of the message is another reference to the version of CircuitPython you're using. Second, a URL for the CircuitPython related project guides. Then... wait. What's this? `To list built-in modules type `help("modules")`.` Remember the modules you learned about while going through creating code? That's exactly what this is talking about! This is a perfect place to start. Let's take a look!

Type `help("modules")` into the REPL next to the prompt, and press enter.

```
>>> 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
Plus any modules on the filesystem
>>>
```

This is a list of all the core modules built into CircuitPython, including `board`. Remember, `board` contains all of the pins on the board that you can use in your code. From the REPL, you are able to see that list!

Type `import board` into the REPL and press enter. It'll go to a new prompt. It might look like nothing happened, but that's not the case! If you recall, the `import` statement simply tells the code to expect to do something with that module. In this case, it's telling the REPL that you plan to do something with that module.

```
>>> import board
>>>
```

Next, type `dir(board)` into the REPL and press enter.

```
>>> dir(board)
['__class__', '__name__', 'A0', 'A1', 'A2', 'A3', 'D0', 'D1', 'D10', 'D11', 'D12', 'D13',
'D24', 'D25', 'D4', 'D5', 'D6', 'D9', 'I2C', 'LED', 'MISO', 'MOSI', 'NEOPIXEL', 'RX', 'SCK
', 'SCL', 'SDA', 'SPI', 'TX', 'UART', 'board_id']
>>>
```

This is a list of all of the pins on your board that are available for you to use in your code. Each board's list will differ slightly depending on the number of pins available. Do you see `LED`? That's the pin you used to blink the red LED!

The REPL can also be used to run code. Be aware that any code you enter into the REPL isn't saved anywhere. If you're testing something new that you'd like to keep, make sure you have it saved somewhere on your computer as well!

Every programmer in every programming language starts with a piece of code that says, "Hello, World." You're going to say hello to something else. Type into the REPL:

`print("Hello, CircuitPython!")`

Then press enter.

```
>>> print("Hello, CircuitPython")
Hello, CircuitPython
>>>
```

That's all there is to running code in the REPL! Nice job!

You can write single lines of code that run stand-alone. You can also write entire programs into the REPL to test them. Remember that nothing typed into the REPL is saved.

There's a lot the REPL can do for you. It's great for testing new ideas if you want to see if a few new lines of code will work. It's fantastic for troubleshooting code by entering it one line at a time and finding out where it fails. It lets you see what modules are available and explore those modules.
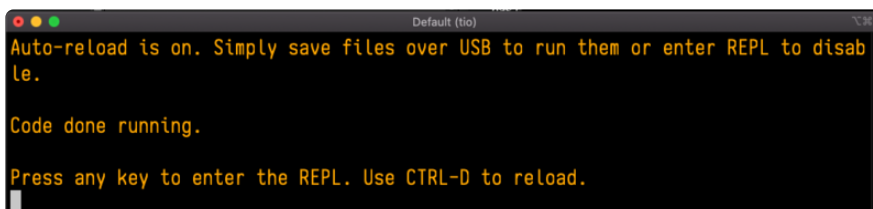
Try typing more into the REPL to see what happens!

> Everything typed into the REPL is ephemeral. Once you reload the REPL or return to the serial console, nothing you typed will be retained in any memory space. So be sure to save any desired code you wrote somewhere else, or you'll lose it when you leave the current REPL instance!

Returning to the Serial Console

When you're ready to leave the REPL and return to the serial console, simply press CTRL+D. This will reload your board and reenter the serial console. You will restart the program you had running before entering the REPL. In the console window, you'll see any output from the program you had running. And if your program was affecting anything visual on the board, you'll see that start up again as well.

You can return to the REPL at any time!

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Code done running.

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

CircuitPython Libraries

> As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. Visit https://circuitpython.org/downloads to

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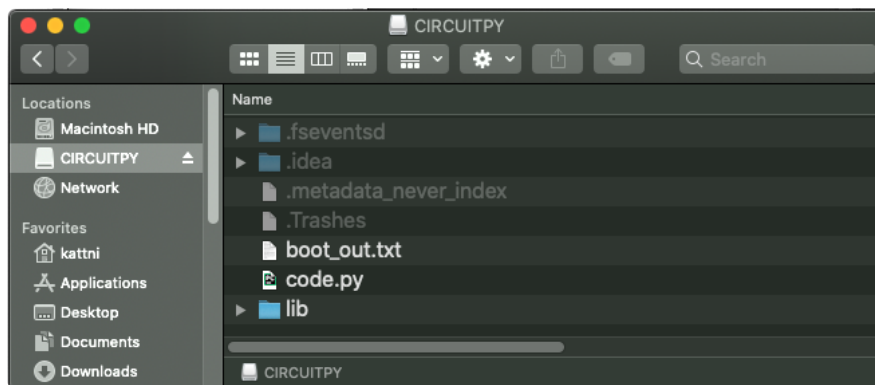
CircuitPython libraries work in the same way as regular Python modules so the Python docs () are an excellent reference for how it all should work. In Python terms, you can place our library files in the lib directory because it's part of the Python path by default.

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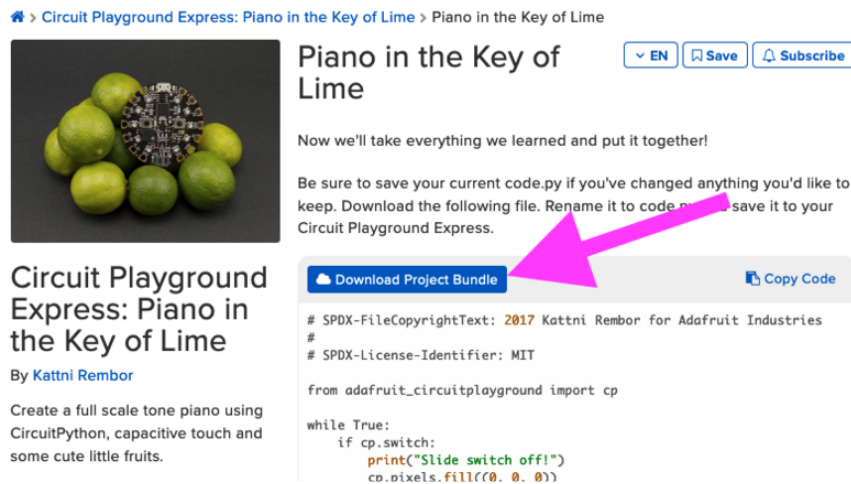
Either way, as you start to explore CircuitPython, you'll want to know how to get libraries on board.

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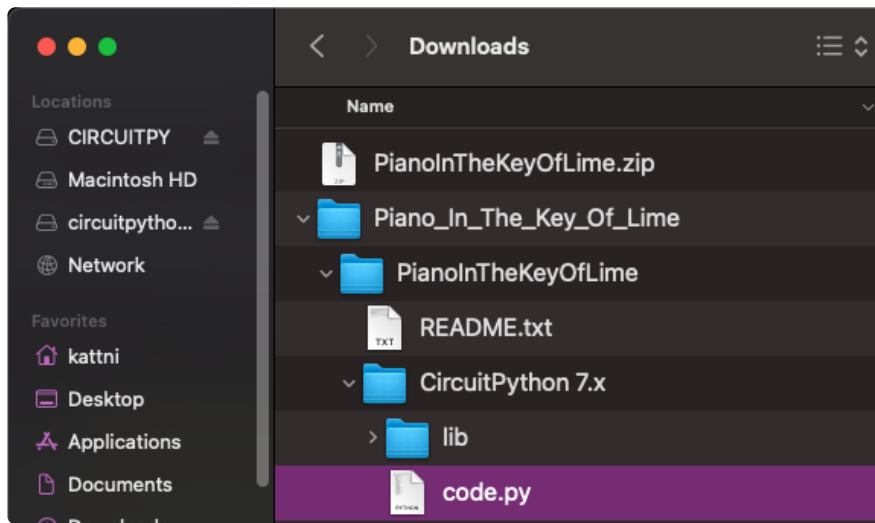
The Piano in the Key of Lime guide was chosen as an example. That guide is specific to Circuit Playground Express, and cannot be used on all boards. Do not expect to download that exact bundle and have it work on your non-CPX microcontroller.

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**

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.

<div style="text-align:center">

Click for the latest CircuitPython Community Library Bundle release

</div>

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.
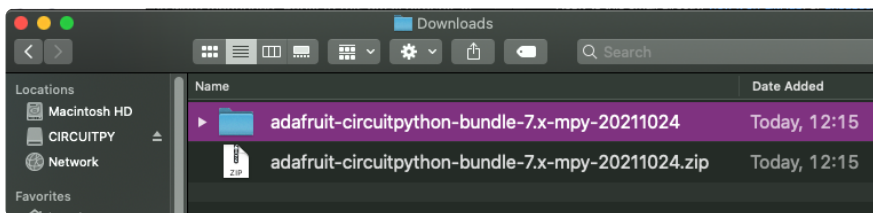
Open the bundle folder. Inside you'll find two information files, and two folders. One folder is the lib bundle, and the other folder is the examples bundle.
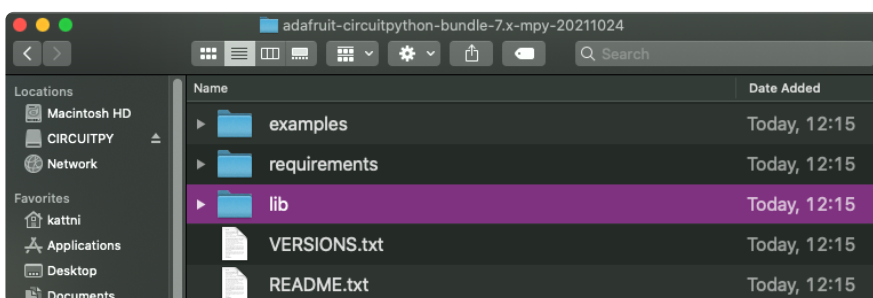
Now open the lib folder. When you open the folder, you'll see a large number of .mpy files, and folders.
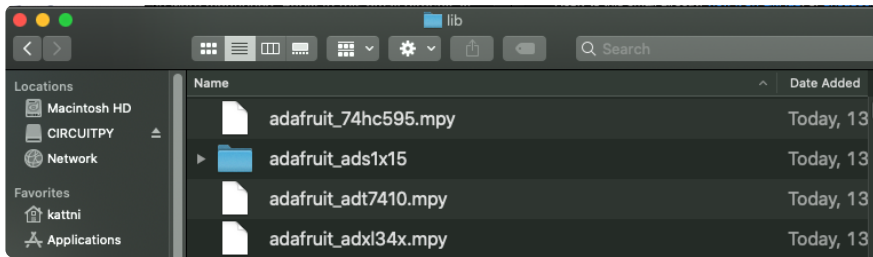
Example Files

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.
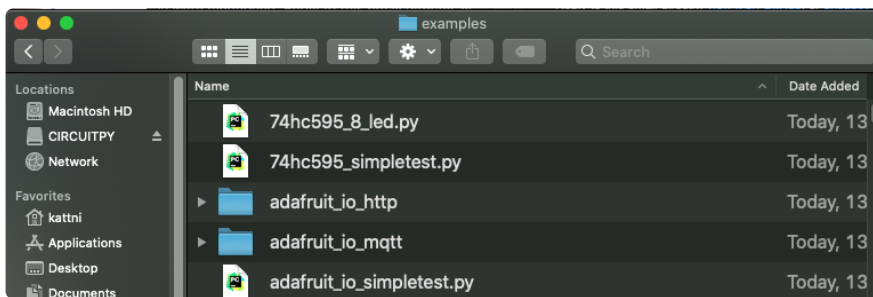
Copying Libraries to Your Board

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 () on The REPL page () 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.
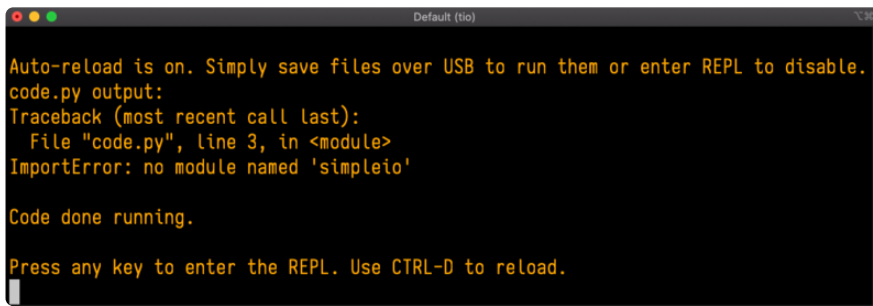
Let's use a modified version of the Blink example.

```
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 ().

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 () 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 (). 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 () for a full list of functionality

CircuitPython Pins and Modules

CircuitPython is designed to run on microcontrollers and allows you to interface with all kinds of sensors, inputs and other hardware peripherals. There are tons of guides showing how to wire up a circuit, and use CircuitPython to, for example, read data from a sensor, or detect a button press. Most CircuitPython code includes hardware setup which requires various modules, such as `board` or `digitalio`. You import these modules and then use them in your code. How does CircuitPython know to look for hardware in the specific place you connected it, and where do these modules come from?

This page explains both. You'll learn how CircuitPython finds the pins on your microcontroller board, including how to find the available pins for your board and what each pin is named. You'll also learn about the modules built into CircuitPython, including how to find all the modules available for your board.

CircuitPython Pins

When using hardware peripherals with a CircuitPython compatible microcontroller, you'll almost certainly be utilising pins. This section will cover how to access your board's pins using CircuitPython, how to discover what pins and board-specific objects are available in CircuitPython for your board, how to use the board-specific objects, and how to determine all available pin names for a given pin on your board.

`import board`

When you're using any kind of hardware peripherals wired up to your microcontroller board, the import list in your code will include `import board`. The `board` module is built into CircuitPython, and is used to provide access to a series of board-specific objects, including pins. Take a look at your microcontroller board. You'll notice that next to the pins are pin labels. You can always access a pin by its pin label. However, there are almost always multiple names for a given pin.

To see all the available board-specific objects and pins for your board, enter the REPL (`>>>`) and run the following commands:

```
import board
dir(board)
```

Here is the output for the QT Py SAMD21. You may have a different board, and this list will vary, based on the board.

The following pins have labels on the physical QT Py SAMD21 board: A0, A1, A2, A3, SDA, SCL, TX, RX, SCK, MISO, and MOSI. You see that there are many more entries available in `board` than the labels on the QT Py.

You can use the pin names on the physical board, regardless of whether they seem to be specific to a certain protocol.

For example, you do not have to use the SDA pin for I2C - you can use it for a button or LED.

On the flip side, there may be multiple names for one pin. For example, on the QT Py SAMD21, pin A0 is labeled on the physical board silkscreen, but it is available in CircuitPython as both `A0` and `D0`. For more information on finding all the names for a given pin, see the [What Are All the Available Pin Names? ()](#) section below.

The results of `dir(board)` for CircuitPython compatible boards will look similar to the results for the QT Py SAMD21 in terms of the pin names, e.g. A0, D0, etc. However, some boards, for example, the Metro ESP32-S2, have different styled pin names. Here is the output for the Metro ESP32-S2.

```
>>> import board
>>> dir(board)
['__class__', 'A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'DEBUG_RX', 'DEBUG_TX', 'I2C',
 'IO1', 'IO10', 'IO11', 'IO12', 'IO13', 'IO14', 'IO15', 'IO16', 'IO17', 'IO18',
 'IO2', 'IO21', 'IO3', 'IO33', 'IO34', 'IO35', 'IO36', 'IO37', 'IO4', 'IO42', 'IO
45', 'IO5', 'IO6', 'IO7', 'IO8', 'IO9', 'LED', 'MISO', 'MOSI', 'NEOPIXEL', 'RX',
 'SCK', 'SCL', 'SDA', 'SPI', 'TX', 'UART']
```

Note that most of the pins are named in an IO# style, such as IO1 and IO2. Those pins on the physical board are labeled only with a number, so an easy way to know how to access them in CircuitPython, is to run those commands in the REPL and find the pin naming scheme.

> If your code is failing to run because it can't find a pin name you provided, verify that you have the proper pin name by running these commands in the REPL.

I2C, SPI, and UART

You'll also see there are often (but not always!) three special board-specific objects included: `I2C`, `SPI`, and `UART` - each one is for the default pin-set used for each of the three common protocol busses they are named for. These are called singletons.

What's a singleton? When you create an object in CircuitPython, you are instantiating ('creating') it. Instantiating an object means you are creating an instance of the object with the unique values that are provided, or "passed", to it.

For example, When you instantiate an I2C object using the `busio` module, it expects two pins: clock and data, typically SCL and SDA. It often looks like this:

```
i2c = busio.I2C(board.SCL, board.SDA)
```

Then, you pass the I2C object to a driver for the hardware you're using. For example, if you were using the TSL2591 light sensor and its CircuitPython library, the next line of code would be:

```
tsl2591 = adafruit_tsl2591.TSL2591(i2c)
```

However, CircuitPython makes this simpler by including the `I2C` singleton in the `board` module. Instead of the two lines of code above, you simply provide the singleton as the I2C object. So if you were using the TSL2591 and its CircuitPython library, the two above lines of code would be replaced with:

```
tsl2591 = adafruit_tsl2591.TSL2591(board.I2C())
```

> The board.I2C(), board.SPI(), and board.UART() singletons do not exist on all boards. They exist if there are board markings for the default pins for those devices.

This eliminates the need for the `busio` module, and simplifies the code. Behind the scenes, the `board.I2C()` object is instantiated when you call it, but not before, and on subsequent calls, it returns the same object. Basically, it does not create an object until you need it, and provides the same object every time you need it. You can call `board.I2C()` as many times as you like, and it will always return the same object.

> The UART/SPI/I2C singletons will use the 'default' bus pins for each board - often labeled as RX/TX (UART), MOSI/MISO/SCK (SPI), or SDA/SCL (I2C). Check your board documentation/pinout for the default busses.

What Are All the Available Names?

Many pins on CircuitPython compatible microcontroller boards have multiple names, however, typically, there's only one name labeled on the physical board. So how do you find out what the other available pin names are? Simple, with the following script! Each line printed out to the serial console contains the set of names for a particular pin.

On a microcontroller board running CircuitPython, first, connect to the serial console.

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 CircuitPython_Essentials/Pin_Map_Script/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

Your CIRCUITPY drive should now look similar to the following image:
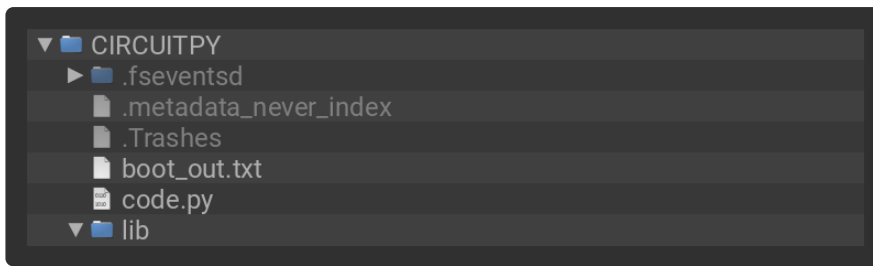
```
# SPDX-FileCopyrightText: 2021-2023 Kattni Rembor for Adafruit Industries
# SPDX-FileCopyrightText: 2023 Dan Halbert for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Pin Map Script"""
import board
import microcontroller

board_pins = []
for pin in dir(microcontroller.pin):
    if isinstance(getattr(microcontroller.pin, pin), microcontroller.Pin):
        pins = []
        for alias in dir(board):
            if getattr(board, alias) is getattr(microcontroller.pin, pin):
                pins.append(f"board.{alias}")
        # Add the original GPIO name, in parentheses.
        if pins:
            # Only include pins that are in board.
            pins.append(f"({str(pin)})")
            board_pins.append(" ".join(pins))

for pins in sorted(board_pins):
    print(pins)
```

Here is the result when this script is run on QT Py SAMD21:
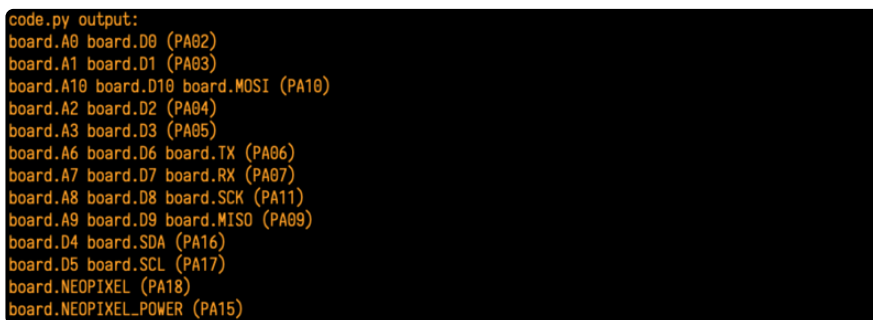
Each line represents a single pin. Find the line containing the pin name that's labeled on the physical board, and you'll find the other names available for that pin. For example, the first pin on the board is labeled A0. The first line in the output is `board.A0 board.D0 (PA02)`. This means that you can access pin A0 in CircuitPython using both `board.A0` and `board.D0`.

The pins in parentheses are the microcontroller pin names. See the next section for more info on those.

You'll notice there are two "pins" that aren't labeled on the board but appear in the list: `board.NEOPIXEL` and `board.NEOPIXEL_POWER` . Many boards have several of these special pins that give you access to built-in board hardware, such as an LED or an on-board sensor. The QT Py SAMD21 only has one on-board extra piece of hardware, a NeoPixel LED, so there's only the one available in the list. But you can also control whether or not power is applied to the NeoPixel, so there's a separate pin for that.

That's all there is to figuring out the available names for a pin on a compatible microcontroller board in CircuitPython!

Microcontroller Pin Names

The pin names available to you in the CircuitPython `board` module are not the same as the names of the pins on the microcontroller itself. The board pin names are aliases to the microcontroller pin names. If you look at the datasheet for your microcontroller, you'll likely find a pinout with a series of pin names, such as "PA18" or "GPIO5". If you want to get to the actual microcontroller pin name in CircuitPython, you'll need the `microcontroller.pin` module. As with `board` , you can run `dir(microcontroller.pin)` in the REPL to receive a list of the microcontroller pin names.

```
>>> import microcontroller
>>> dir(microcontroller.pin)
['__class__', 'PA02', 'PA03', 'PA04', 'PA05', 'PA06', 'PA07', 'PA08', 'PA09',
'PA10', 'PA11', 'PA15', 'PA16', 'PA17', 'PA18', 'PA19', 'PA22', 'PA23']
```

CircuitPython Built-In Modules

There is a set of modules used in most CircuitPython programs. One or more of these modules is always used in projects involving hardware. Often hardware requires installing a separate library from the Adafruit CircuitPython Bundle. But, if you try to find `board` or `digitalio` in the same bundle, you'll come up lacking. So, where do these modules come from? They're built into CircuitPython! You can find an comprehensive list of built-in CircuitPython modules and the technical details of their functionality from CircuitPython [here](#) () and the Python-like modules included [here](#) (). However, not every module is available for every board due to size constraints or hardware limitations. How do you find out what modules are available for your board?

There are two options for this. You can check the [support matrix](#) (), and search for your board by name. Or, you can use the REPL.

Plug in your board, connect to the serial console and enter the REPL. Type the following command.

```
help("modules")
```


That's it! You now know two ways to find all of the modules built into CircuitPython for your compatible microcontroller board.

Advanced Serial Console on Mac

Connecting to the serial console on Mac does not require installing any drivers or extra software. You'll use a terminal program to find your board, and `screen` to connect to it. Terminal and `screen` both come installed by default.
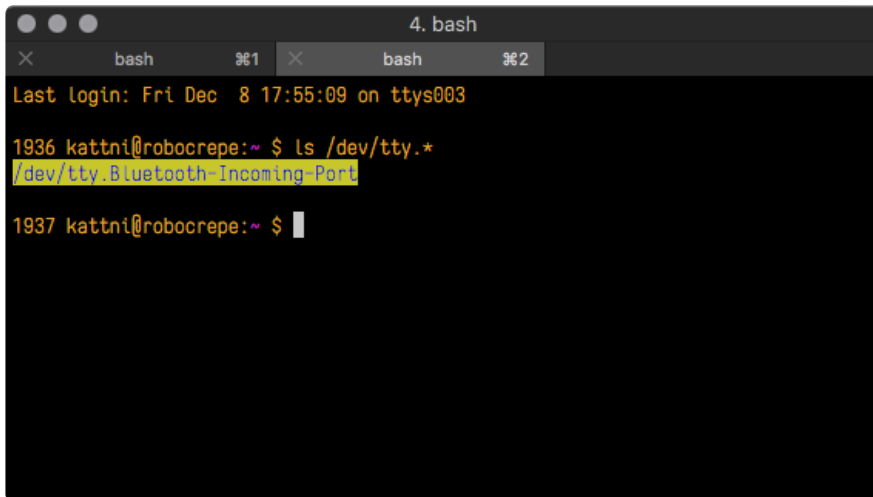
What's the Port?

First you'll want to find out which serial port your board is using. When you plug your board in to USB on your computer, it connects to a serial port. The port is like a door through which your board can communicate with your computer using USB.

The easiest way to determine which port the board is using is to first check without the board plugged in. Open Terminal and type the following:

```
ls /dev/tty.*
```

Each serial connection shows up in the `/dev/` directory. It has a name that starts with `tty.` . The command `ls` shows you a list of items in a directory. You can use `*` as a wildcard, to search for files that start with the same letters but end in something different. In this case, you're asking to see all of the listings in `/dev/` that start with `tty.` and end in anything. This will show us the current serial connections.
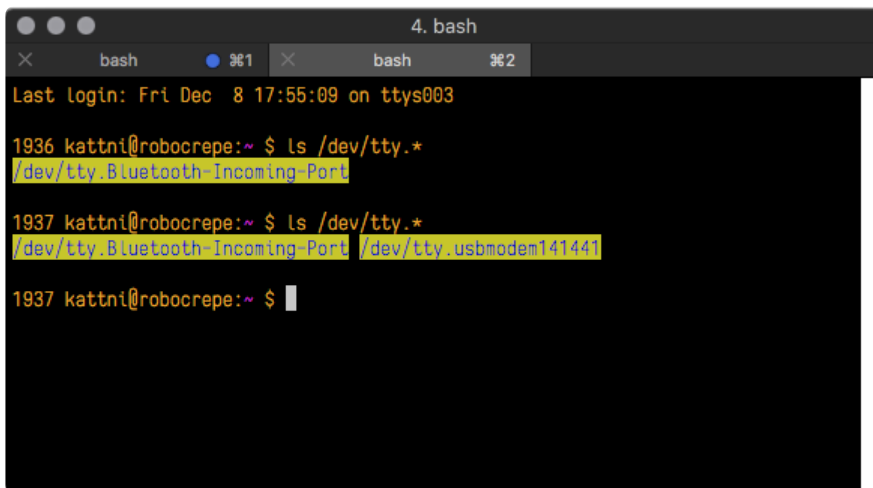
Now, plug your board. In Terminal, type:

```
ls /dev/tty.*
```

This will show you the current serial connections, which will now include your board.

A new listing has appeared called `/dev/tty.usbmodem141441`. The `tty.usbmodem141441` part of this listing is the name the example board is using. Yours will be called something similar.

Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.
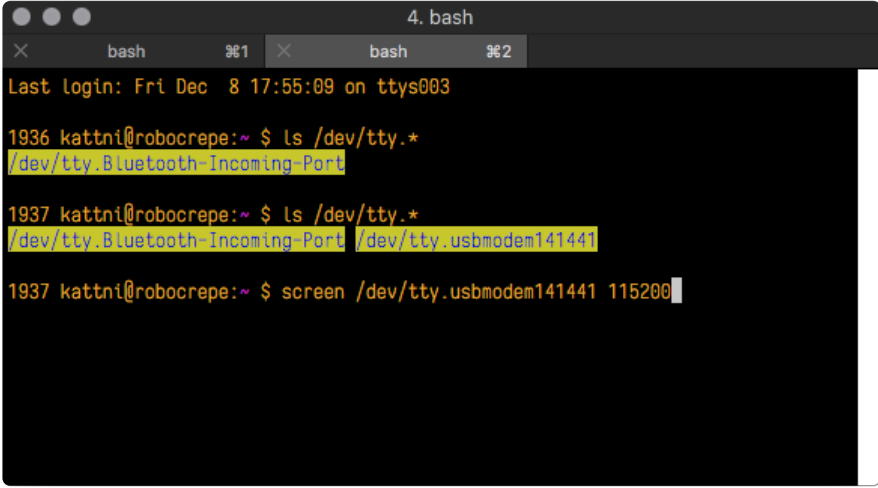
Connect with screen

Now that you know the name your board is using, you're ready connect to the serial console. You're going to use a command called `screen`. The `screen` command is included with MacOS. To connect to the serial console, use Terminal. Type the

following command, replacing `board_name` with the name you found your board is using:

```
screen /dev/tty.board_name 115200
```

The first part of this establishes using the `screen` command. The second part tells screen the name of the board you're trying to use. The third part tells screen what baud rate to use for the serial connection. The baud rate is the speed in bits per second that data is sent over the serial connection. In this case, the speed required by the board is 115200 bits per second.

Press enter to run the command. It will open in the same window. If no code is running, the window will be blank. Otherwise, you'll see the output of your code.

Great job! You've connected to the serial console!

Advanced Serial Console on Windows

Windows 7 and 8.1

If you're using Windows 7 (or 8 or 8.1), you'll need to install drivers. See the Windows 7 and 8.1 Drivers page () for details. You will not need to install drivers on Mac, Linux or Windows 10.

You are strongly encouraged to upgrade to Windows 10 if you are still using Windows 7 or Windows 8 or 8.1. Windows 7 has reached end-of-life and no longer receives security updates. A free upgrade to Windows 10 is still available ().

What's the COM?

First, you'll want to find out which serial port your board is using. When you plug your board in to USB on your computer, it connects to a serial port. The port is like a door through which your board can communicate with your computer using USB.

You'll use Windows Device Manager to determine which port the board is using. The easiest way to determine which port the board is using is to first check without the board plugged in. Open Device Manager. Click on Ports (COM & LPT). You should find something already in that list with (COM#) after it where # is a number.
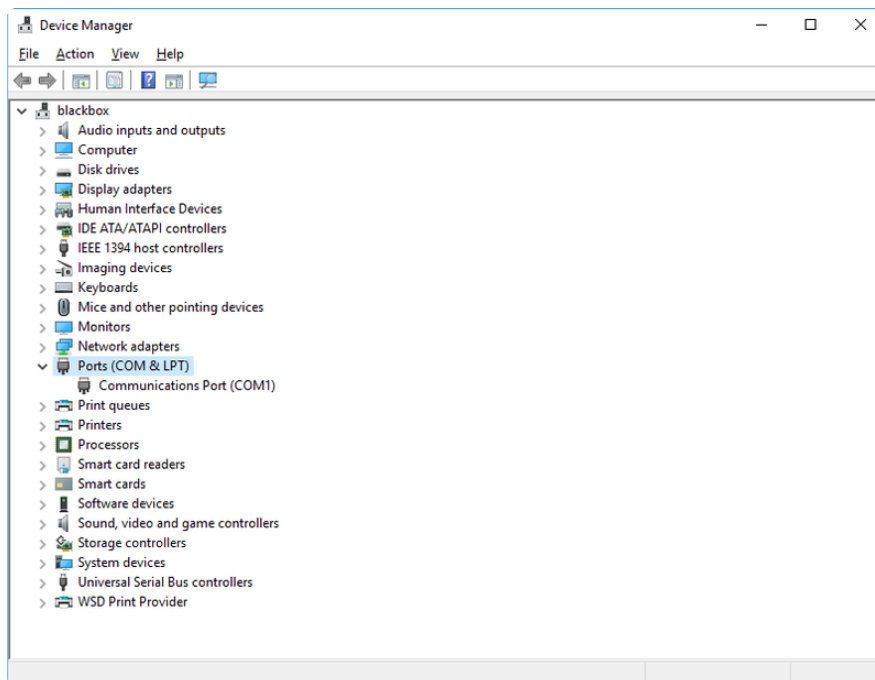
Now plug in your board. The Device Manager list will refresh and a new item will appear under Ports (COM & LPT). You'll find a different (COM#) after this item in the list.
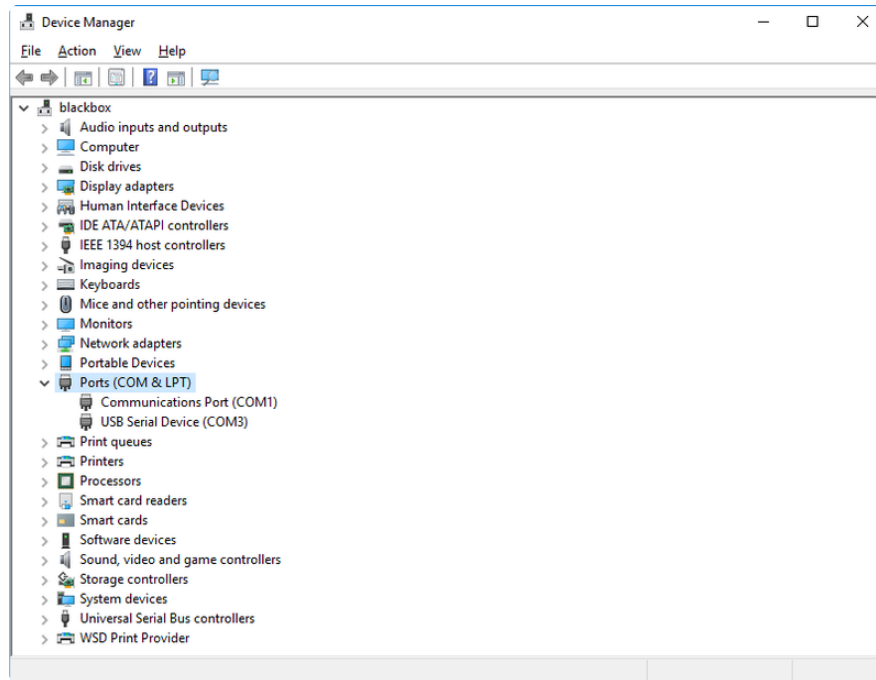
Sometimes the item will refer to the name of the board. Other times it may be called something like USB Serial Device, as seen in the image above. Either way, there is a new (COM#) following the name. This is the port your board is using.

Install Putty

If you're using Windows, you'll need to download a terminal program. You're going to use PuTTY.

The first thing to do is download the latest version of PuTTY (). You'll want to download the Windows installer file. It is most likely that you'll need the 64-bit version. Download the file and install the program on your machine. If you run into issues, you can try downloading the 32-bit version instead. However, the 64-bit version will work on most PCs.

Now you need to open PuTTY.

- Under Connection type: choose the button next to Serial.
- In the box under Serial line, enter the serial port you found that your board is using.
- In the box under Speed, enter 115200. This called the baud rate, which is the speed in bits per second that data is sent over the serial connection. For boards with built in USB it doesn't matter so much but for ESP8266 and other board with a separate chip, the speed required by the board is 115200 bits per second. So you might as well just use 115200!

If you want to save those settings for later, use the options under Load, save or delete a stored session. Enter a name in the box under Saved Sessions, and click the Save button on the right.
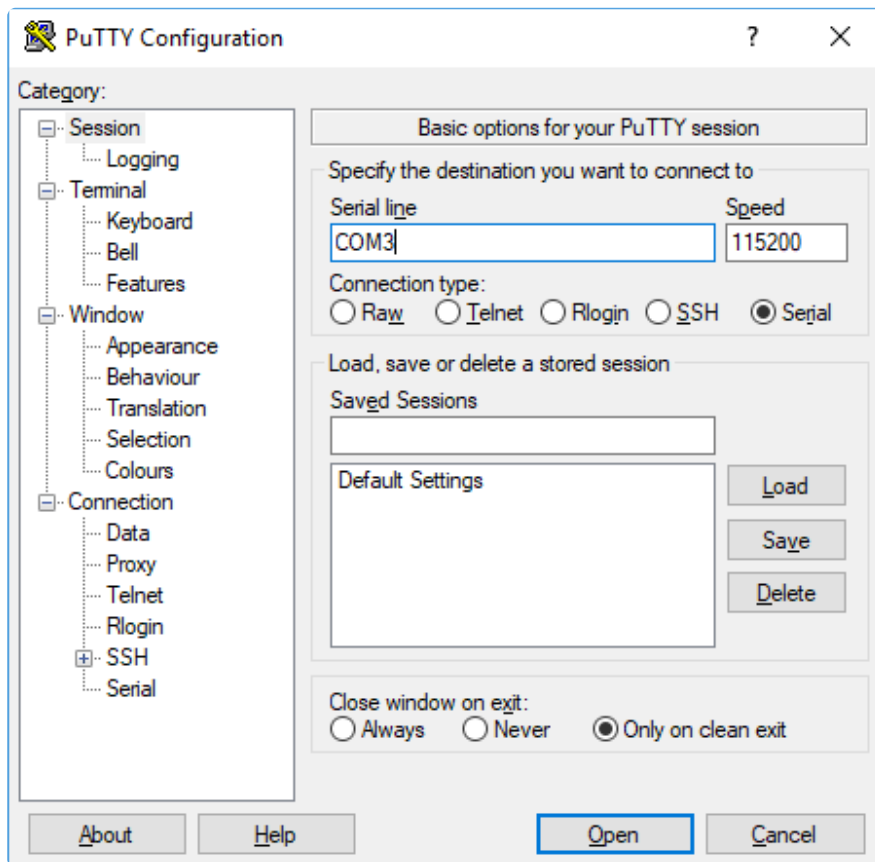
Once your settings are entered, you're ready to connect to the serial console. Click "Open" at the bottom of the window. A new window will open.
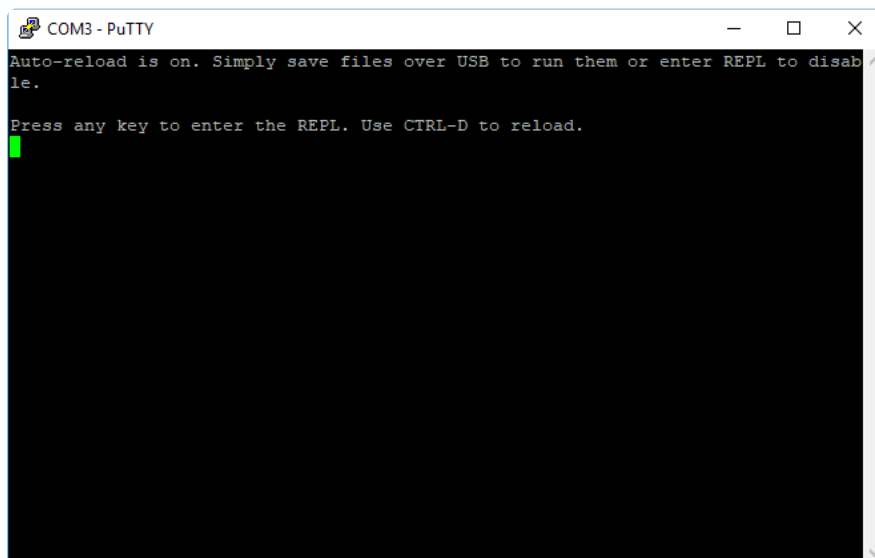
If no code is running, the window will either be blank or will look like the window above. Now you're ready to see the results of your code.

Welcome to the Community!
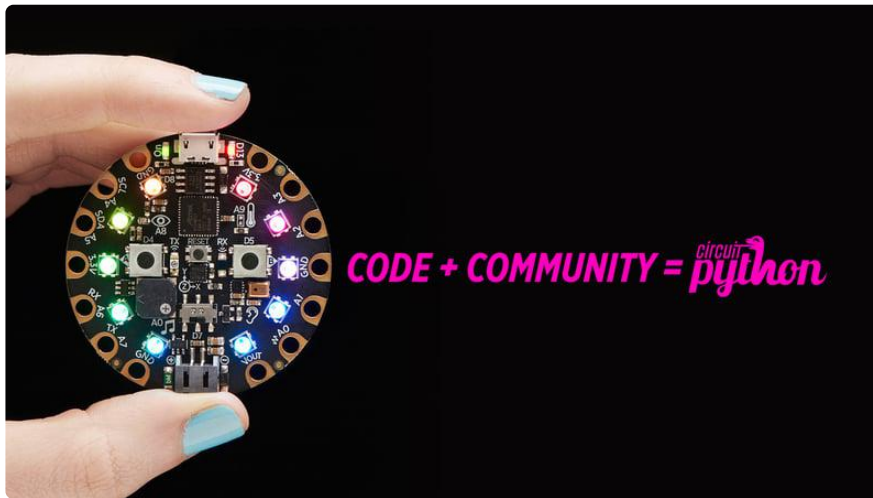
CircuitPython is a programming language that's super simple to get started with and great for learning. It runs on microcontrollers and works out of the box. You can plug it in and get started with any text editor. The best part? CircuitPython comes with an amazing, supportive community.

Everyone is welcome! CircuitPython is Open Source. This means it's available for anyone to use, edit, copy and improve upon. This also means CircuitPython becomes better because of you being a part of it. Whether this is your first microcontroller board or you're a seasoned software engineer, you have something important to offer the Adafruit CircuitPython community. This page highlights some of the many ways you can be a part of it!

Adafruit Discord
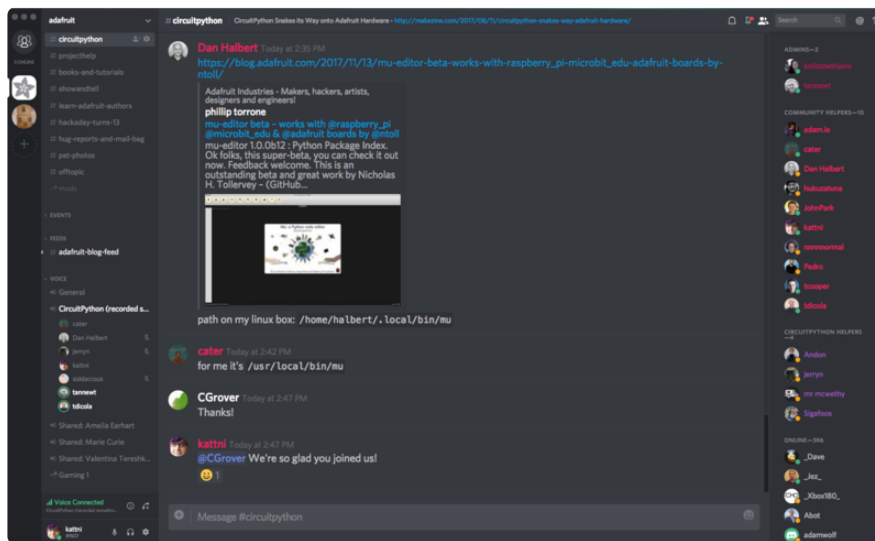
The Adafruit Discord server is the best place to start. Discord is where the community comes together to volunteer and provide live support of all kinds. From general discussion to detailed problem solving, and everything in between, Discord is a digital maker space with makers from around the world.

There are many different channels so you can choose the one best suited to your needs. Each channel is shown on Discord as "#channelname". There's the #help-with-projects channel for assistance with your current project or help coming up with ideas for your next one. There's the #show-and-tell channel for showing off your newest creation. Don't be afraid to ask a question in any channel! If you're unsure, #general is a great place to start. If another channel is more likely to provide you with a better answer, someone will guide you.

The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. Your contributions are important! The #circuitpython-dev channel is available for development discussions as well.

The easiest way to contribute to the community is to assist others on Discord. Supporting others doesn't always mean answering questions. Join in celebrating successes! Celebrate your mistakes! Sometimes just hearing that someone else has gone through a similar struggle can be enough to keep a maker moving forward.

The Adafruit Discord is the 24x7x365 hackerspace that you can bring your granddaughter to.

Visit https://adafru.it/discord ()to sign up for Discord. Everyone is looking forward to meeting you!

CircuitPython.org
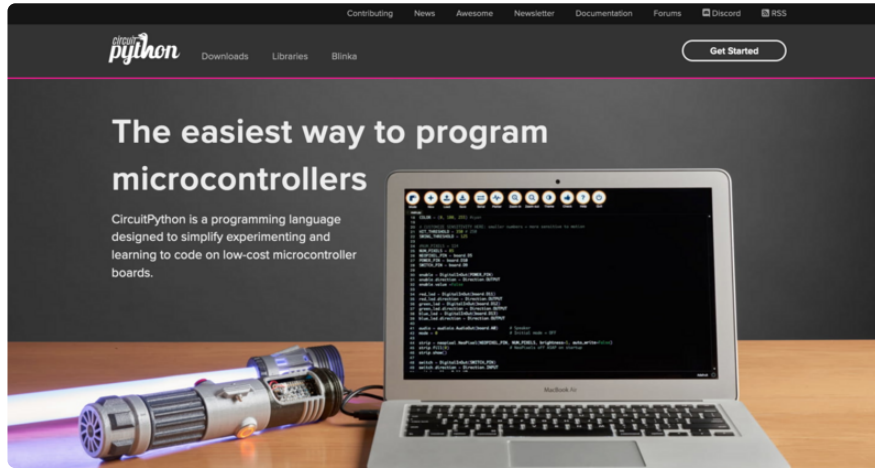
Beyond the Adafruit Learn System, which you are viewing right now, the best place to find information about CircuitPython is circuitpython.org (). Everything you need to get started with your new microcontroller and beyond is available. You can do things like download CircuitPython for your microcontroller () or download the latest CircuitPython Library bundle (), or check out which single board computers support Blinka (). You can also get to various other CircuitPython related things like Awesome CircuitPython or the Python for Microcontrollers newsletter. This is all incredibly useful, but it isn't necessarily community related. So why is it included here? The Contributing page ().

CircuitPython itself is written in C. However, all of the Adafruit CircuitPython libraries are written in Python. If you're interested in contributing to CircuitPython on the Python side of things, check out circuitpython.org/contributing (). You'll find information pertaining to every Adafruit CircuitPython library GitHub repository, giving you the opportunity to join the community by finding a contributing option that works for you.

Note the date on the page next to Current Status for:

Current Status for Tue, Nov 02, 2021

If you submit any contributions to the libraries, and do not see them reflected on the Contributing page, it could be that the job that checks for new updates hasn't yet run for today. Simply check back tomorrow!

Now, a look at the different options.

Pull Requests

The first tab you'll find is a list of open pull requests.

GitHub pull requests, or PRs, are opened when folks have added something to an Adafruit CircuitPython library GitHub repo, and are asking for Adafruit to add, or merge, their changes into the main library code. For PRs to be merged, they must first be reviewed. Reviewing is a great way to contribute! Take a look at the list of open pull requests, and pick one that interests you. If you have the hardware, you can test code changes. If you don't, you can still check the code updates for syntax. In the case of documentation updates, you can verify the information, or check it for spelling and grammar. Once you've checked out the update, you can leave a comment letting us know that you took a look. Once you've done that for a while, and you're more comfortable with it, you can consider joining the CircuitPythonLibrarians review team. The more reviewers we have, the more authors we can support. Reviewing is a crucial part of an open source ecosystem, CircuitPython included.

Open Issues

The second tab you'll find is a list of open issues.

GitHub issues are filed for a number of reasons, including when there is a bug in the library or example code, or when someone wants to make a feature request. Issues are a great way to find an opportunity to contribute directly to the libraries by updating code or documentation. If you're interested in contributing code or documentation, take a look at the open issues and find one that interests you.

If you're not sure where to start, you can search the issues by label. Labels are applied to issues to make the goal easier to identify at a first glance, or to indicate the difficulty level of the issue. Click on the dropdown next to "Sort by issue labels" to see the list of available labels, and click on one to choose it.
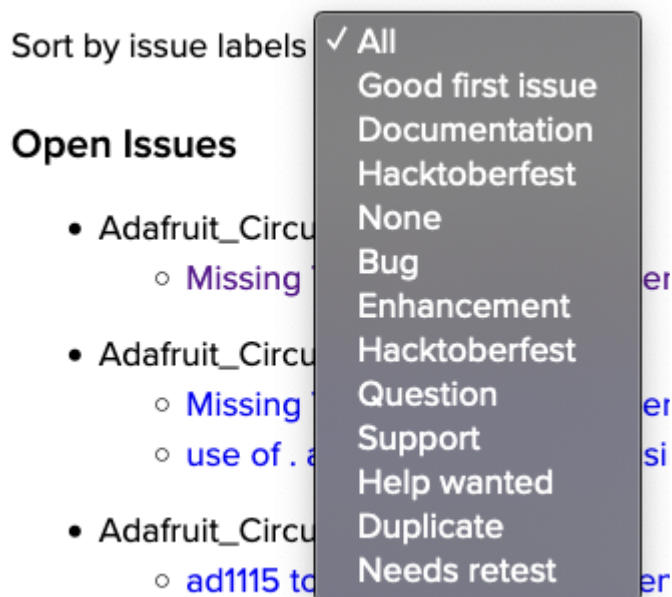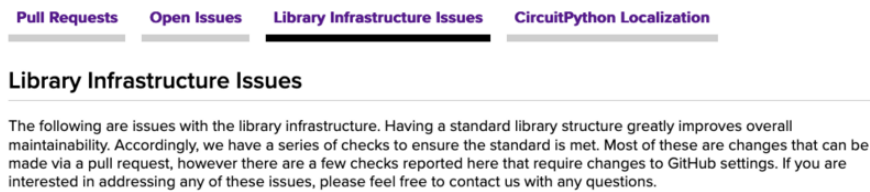
If you're new to everything, new to contributing to open source, or new to contributing to the CircuitPython project, you can choose "Good first issue". Issues with that label are well defined, with a finite scope, and are intended to be easy for someone new to figure out.

If you're looking for something a little more complicated, consider "Bug" or "Enhancement". The Bug label is applied to issues that pertain to problems or failures found in the library. The Enhancement label is applied to feature requests.

Don't let the process intimidate you. If you're new to Git and GitHub, there is a guide ()
to walk you through the entire process. As well, there are always folks available on Di
scord () to answer questions.

Library Infrastructure Issues

The third tab you'll find is a list of library infrastructure issues.

This section is generated by a script that runs checks on the libraries, and then
reports back where there may be issues. It is made up of a list of subsections each
containing links to the repositories that are experiencing that particular issue. This
page is available mostly for internal use, but you may find some opportunities to
contribute on this page. If there's an issue listed that sounds like something you could
help with, mention it on Discord, or file an issue on GitHub indicating you're working
to resolve that issue. Others can reply either way to let you know what the scope of it
might be, and help you resolve it if necessary.

CircuitPython Localization

The fourth tab you'll find is the CircuitPython Localization tab.
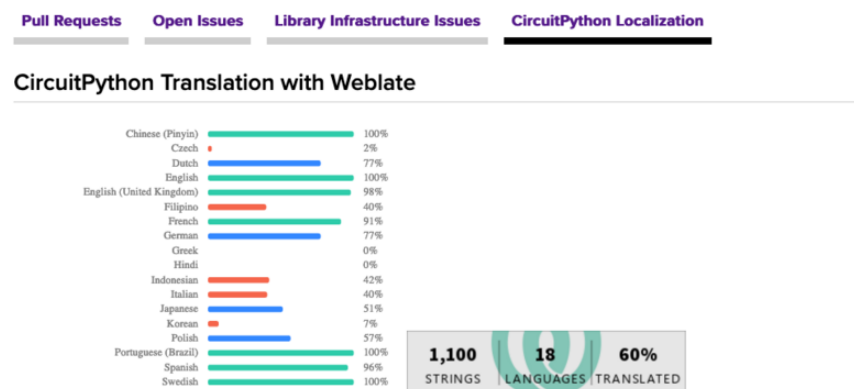
If you speak another language, you can help translate CircuitPython! The translations
apply to informational and error messages that are within the CircuitPython core. It
means that folks who do not speak English have the opportunity to have these
messages shown to them in their own language when using CircuitPython. This is
incredibly important to provide the best experience possible for all users.

CircuitPython uses Weblate to translate, which makes it much simpler to contribute translations. You will still need to know some CircuitPython-specific practices and a few basics about coding strings, but as with any CircuitPython contributions, folks are there to help.

Regardless of your skill level, or how you want to contribute to the CircuitPython project, there is an opportunity available. The Contributing page () is an excellent place to start!

Adafruit GitHub
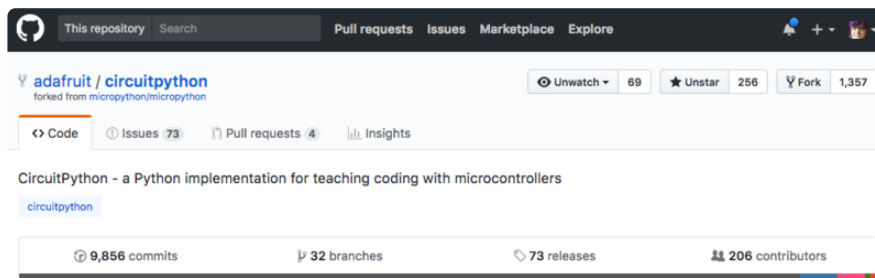
Whether you're just beginning or are life-long programmer who would like to contribute, there are ways for everyone to be a part of the CircuitPython project. The CircuitPython core is written in C. The libraries are written in Python. GitHub is the best source of ways to contribute to the CircuitPython core (), and the CircuitPython libraries (). If you need an account, visit https://github.com/ () and sign up.

If you're new to GitHub or programming in general, there are great opportunities for you. For the CircuitPython core, head over to the CircuitPython repository on GitHub, click on "Issues ()", and you'll find a list that includes issues labeled "good first issue ()" . For the libraries, head over to the Contributing page Issues list (), and use the drop down menu to search for "good first issue ()". These issues are things that have been identified as something that someone with any level of experience can help with. These issues include options like updating documentation, providing feedback, and fixing simple bugs. If you need help getting started with GitHub, there is an excellent guide on Contributing to CircuitPython with Git and GitHub ().
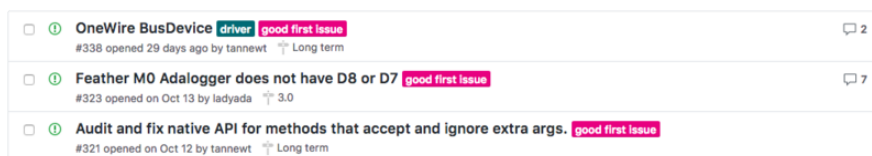
Already experienced and looking for a challenge? Checkout the rest of either issues list and you'll find plenty of ways to contribute. You'll find all sorts of things, from new driver requests, to library bugs, to core module updates. There's plenty of opportunities for everyone at any level!

When working with or using CircuitPython or the CircuitPython libraries, you may find problems. If you find a bug, that's great! The team loves bugs! Posting a detailed issue to GitHub is an invaluable way to contribute to improving CircuitPython. For CircuitPython itself, file an issue here (). For the libraries, file an issue on the specific library repository on GitHub. Be sure to include the steps to replicate the issue as well as any other information you think is relevant. The more detail, the better!

Testing new software is easy and incredibly helpful. Simply load the newest version of CircuitPython or a library onto your CircuitPython hardware, and use it. Let us know about any problems you find by posting a new issue to GitHub. Software testing on both stable and unstable releases is a very important part of contributing CircuitPython. The developers can't possibly find all the problems themselves! They need your help to make CircuitPython even better.

On GitHub, you can submit feature requests, provide feedback, report problems and much more. If you have questions, remember that Discord and the Forums are both there for help!

Adafruit Forums
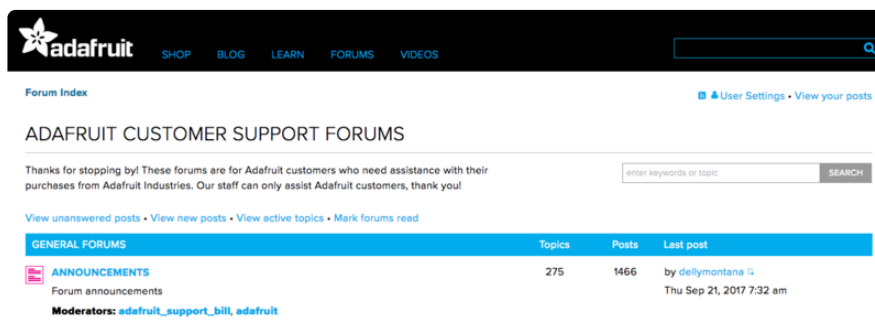
The Adafruit Forums () are the perfect place for support. Adafruit has wonderful paid support folks to answer any questions you may have. Whether your hardware is giving you issues or your code doesn't seem to be working, the forums are always there for you to ask. You need an Adafruit account to post to the forums. You can use the same account you use to order from Adafruit.

While Discord may provide you with quicker responses than the forums, the forums are a more reliable source of information. If you want to be certain you're getting an Adafruit-supported answer, the forums are the best place to be.

There are forum categories that cover all kinds of topics, including everything Adafruit. The Adafruit CircuitPython () category under "Supported Products & Projects" is the best place to post your CircuitPython questions.
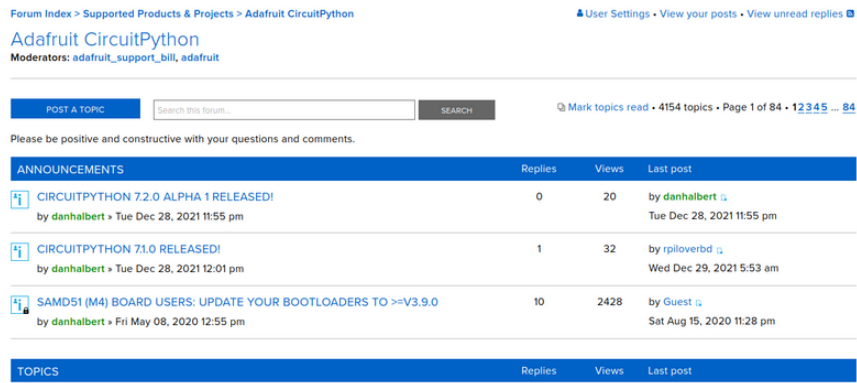
Be sure to include the steps you took to get to where you are. If it involves wiring, post a picture! If your code is giving you trouble, include your code in your post! These are great ways to make sure that there's enough information to help you with your issue.

You might think you're just getting started, but you definitely know something that someone else doesn't. The great thing about the forums is that you can help others too! Everyone is welcome and encouraged to provide constructive feedback to any of the posted questions. This is an excellent way to contribute to the community and share your knowledge!

Read the Docs
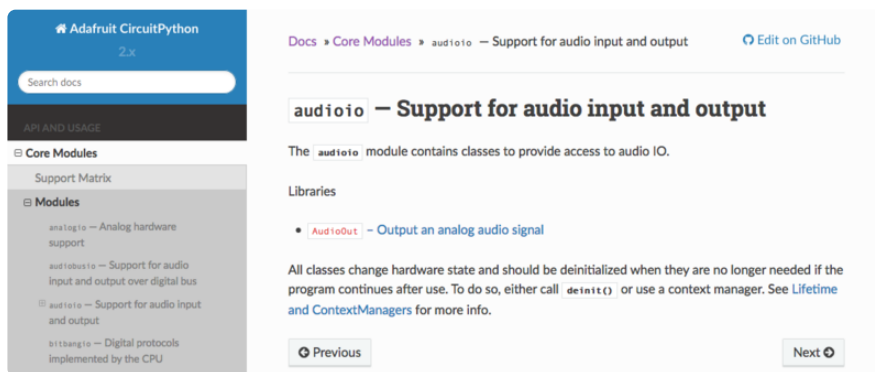
Read the Docs () is a an excellent resource for a more detailed look at the CircuitPython core and the CircuitPython libraries. This is where you'll find things like API documentation and example code. For an in depth look at viewing and understanding Read the Docs, check out the CircuitPython Documentation () page!

Here is blinky:

```
import time
import digitalio
import board

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT
while True:
    led.value = True
    time.sleep(0.1)
    led.value = False
    time.sleep(0.1)
```

Frequently Asked Questions

These are some of the common questions regarding CircuitPython and CircuitPython microcontrollers.

What are some common acronyms to know?

CP or CPy = CircuitPython ()
CPC = Circuit Playground Classic () (does not run CircuitPython)
CPX = Circuit Playground Express ()
CPB = Circuit Playground Bluefruit ()

Using Older Versions

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.

I have to continue using CircuitPython 6.x or earlier. Where can I find compatible libraries?

We are no longer building or supporting the CircuitPython 6.x or earlier library bundles. We highly encourage you to update CircuitPython to the latest version () and use the current version of the libraries (). However, if for some reason you cannot update, here are the last available library bundles for older versions:

- 2.x bundle ()
- 3.x bundle ()
- 4.x bundle ()

- [5.x bundle ()](())
- [6.x bundle ()](())

Python Arithmetic

Does CircuitPython support floating-point numbers?

All CircuitPython boards support floating point arithmetic, even if the microcontroller chip does not support floating point in hardware. Floating point numbers are stored in 30 bits, with an 8-bit exponent and a 22-bit mantissa. Note that this is two bits less than standard 32-bit single-precision floats. You will get about 5-1/2 digits of decimal precision.

(The broadcom port may provide 64-bit floats in some cases.)

Does CircuitPython support long integers, like regular Python?

Python long integers (integers of arbitrary size) are available on most builds, except those on boards with the smallest available firmware size. On these boards, integers are stored in 31 bits.

Boards without long integer support are mostly SAMD21 ("M0") boards without an external flash chip, such as the Adafruit Gemma M0, Trinket M0, QT Py M0, and the Trinkey series. There are also a number of third-party boards in this category. There are also a few small STM third-party boards without long integer support.

`time.localtime()` , `time.mktime()` , `time.time()` , and `time.monotonic_ns()` are available only on builds with long integers.

Wireless Connectivity

How do I connect to the Internet with CircuitPython?

If you'd like to include WiFi in your project, your best bet is to use a board that is running natively on ESP32 chipsets - those have WiFi built in!

If your development board has an SPI port and at least 4 additional pins, you can check out [this guide ()](()) on using AirLift with CircuitPython - extra wiring is required

and some boards like the MacroPad or NeoTrellis do not have enough available pins to add the hardware support.

For further project examples, and guides about using AirLift with specific hardware, check out the Adafruit Learn System ().

How do I do BLE (Bluetooth Low Energy) with CircuitPython?

The nRF52840 and nRF52833 boards have the most complete BLE implementation. Your program can act as both a BLE central and peripheral. As a central, you can scan for advertisements, and connect to an advertising board. As a peripheral, you can advertise, and you can create services available to a central. Pairing and bonding are supported.

ESP32-C3 and ESP32-S3 boards currently provide an incomplete () BLE implementation. Your program can act as a central, and connect to a peripheral. You can advertise, but you cannot create services. You cannot advertise anonymously. Pairing and bonding are not supported.

The ESP32 could provide a similar implementation, but it is not yet available. Note that the ESP32-S2 does not have Bluetooth capability.

On most other boards with adequate firmware space, BLE is available for use with AirLift () or other NINA-FW-based co-processors. Some boards have this coprocessor on board, such as the PyPortal (). Currently, this implementation only supports acting as a BLE peripheral. Scanning and connecting as a central are not yet implemented. Bonding and pairing are not supported.

Are there other ways to communicate by radio with CircuitPython?

Check out Adafruit's RFM boards ()for simple radio communication supported by CircuitPython, which can be used over distances of 100m to over a km, depending on the version. The RFM SAMD21 M0 boards can be used, but they were not designed for CircuitPython, and have limited RAM and flash space; using the RFM breakouts or FeatherWings with more capable boards will be easier.

Asyncio and Interrupts

Is there asyncio support in CircuitPython?

There is support for asyncio starting with CircuitPython 7.1.0, on all boards except the smallest SAMD21 builds. Read about using it in the Cooperative Multitasking in CircuitPython () Guide.

Does CircuitPython support interrupts?

No. CircuitPython does not currently support interrupts - please use asyncio for multitasking / 'threaded' control of your code

Status RGB LED

My RGB NeoPixel/DotStar LED is blinking funny colors - what does it mean?

The status LED can tell you what's going on with your CircuitPython board. Read more here for what the colors mean! ()

Memory Issues

What is a MemoryError?

Memory allocation errors happen when you're trying to store too much on the board. The CircuitPython microcontroller boards have a limited amount of memory available. You can have about 250 lines of code on the M0 Express boards. If you try to `import` too many libraries, a combination of large libraries, or run a program with too many lines of code, your code will fail to run and you will receive a `MemoryError` in the serial console.

What do I do when I encounter a MemoryError?

Try resetting your board. Each time you reset the board, it reallocates the memory. While this is unlikely to resolve your issue, it's a simple step and is worth trying.

Make sure you are using .mpy versions of libraries. All of the CircuitPython libraries are available in the bundle in a .mpy format which takes up less memory than .py format. Be sure that you're using the latest library bundle () for your version of CircuitPython.

If that does not resolve your issue, try shortening your code. Shorten comments, remove extraneous or unneeded code, or any other clean up you can do to shorten your code. If you're using a lot of functions, you could try moving those into a separate library, creating a .mpy of that library, and importing it into your code.

You can turn your entire file into a .mpy and `import` that into code.py. This means you will be unable to edit your code live on the board, but it can save you space.

Can the order of my `import` statements affect memory?

It can because the memory gets fragmented differently depending on allocation order and the size of objects. Loading .mpy files uses less memory so its recommended to do that for files you aren't editing.

How can I create my own .mpy files?

You can make your own .mpy versions of files with `mpy-cross`.

You can download `mpy-cross` for your operating system from here (). Builds are available for Windows, macOS, x64 Linux, and Raspberry Pi Linux. Choose the latest `mpy-cross` whose version matches the version of CircuitPython you are using.

To make a .mpy file, run `./mpy-cross path/to/yourfile.py` to create a yourfile.mpy in the same directory as the original file.

How do I check how much memory I have free?

Run the following to see the number of bytes available for use:

```
import gc
gc.mem_free()
```

Unsupported Hardware

Is ESP8266 or ESP32 supported in CircuitPython? Why not?

We dropped ESP8266 support as of 4.x - For more information please read about it here ()!

As of CircuitPython 8.x we have started to support ESP32 and ESP32-C3 and have added a WiFi workflow for wireless coding! ()

We also support ESP32-S2 & ESP32-S3, which have native USB.

Does Feather M0 support WINC1500?

No, WINC1500 will not fit into the M0 flash space.

Can AVRs such as ATmega328 or ATmega2560 run CircuitPython?

No.

Troubleshooting

From time to time, you will run into issues when working with CircuitPython. Here are a few things you may encounter and how to resolve them.

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.

Always Run the Latest Version of CircuitPython and Libraries

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. You need to update to the latest CircuitPython. ().

You need to download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then download the latest bundle ().

As new versions of CircuitPython are released, Adafruit will stop providing the previous bundles as automatically created downloads on the Adafruit CircuitPython Library Bundle repo. If you must continue to use an earlier version, you can still download the appropriate version of `mpy-cross` from the particular release of

CircuitPython on the CircuitPython repo and create your own compatible .mpy library files. However, it is best to update to the latest for both CircuitPython and the library bundle.

I have to continue using CircuitPython 5.x or earlier. Where can I find compatible libraries?

Adafruit is no longer building or supporting the CircuitPython 5.x or earlier library bundles. You are highly encourged to update CircuitPython to the latest version () and use the current version of the libraries (). However, if for some reason you cannot update, links to the previous bundles are available in the FAQ ().

Bootloader (boardnameBOOT) Drive Not Present

You may have a different board.

Only Adafruit Express boards and the SAMD21 non-Express boards ship with the UF2 bootloader ()installed. The Feather M0 Basic, Feather M0 Adalogger, and similar boards use a regular Arduino-compatible bootloader, which does not show a boardnameBOOT drive.

MakeCode

If you are running a MakeCode () program on Circuit Playground Express, press the reset button just once to get the CPLAYBOOT drive to show up. Pressing it twice will not work.

MacOS

DriveDx and its accompanything SAT SMART Driver can interfere with seeing the BOOT drive. See this forum post () for how to fix the problem.

Windows 10

Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 with the driver package installed? You don't need to install this package on Windows 10 for most Adafruit boards. The old version (v1.5) can interfere

with recognizing your device. Go to Settings -> Apps and uninstall all the "Adafruit" driver programs.

Windows 7 or 8.1

To use a CircuitPython-compatible board with Windows 7 or 8.1, you must install a driver. Installation instructions are available here ().

It is recommended () that you upgrade to Windows 10 if possible; an upgrade is probably still free for you. Check here ().

> The Windows Drivers installer was last updated in November 2020 (v2.5.0.0) . Windows 7 drivers for CircuitPython boards released since then, including RP2040 boards, are not yet available. The boards work fine on Windows 10. A new release of the drivers is in process.

You should now be done! Test by unplugging and replugging the board. You should see the CIRCUITPY drive, and when you double-click the reset button (single click on Circuit Playground Express running MakeCode), you should see the appropriate boardnameBOOT drive.

Let us know in the Adafruit support forums () or on the Adafruit Discord () if this does not work for you!

Windows Explorer Locks Up When Accessing boardnameBOOT Drive

On Windows, several third-party programs that can cause issues. The symptom is that you try to access the boardnameBOOT drive, and Windows or Windows Explorer seems to lock up. These programs are known to cause trouble:

- AIDA64: to fix, stop the program. This problem has been reported to AIDA64. They acquired hardware to test, and released a beta version that fixes the problem. This may have been incorporated into the latest release. Please let us know in the forums if you test this.
- Hard Disk Sentinel
- Kaspersky anti-virus: To fix, you may need to disable Kaspersky completely. Disabling some aspects of Kaspersky does not always solve the problem. This problem has been reported to Kaspersky.

- ESET NOD32 anti-virus: There have been problems with at least version 9.0.386.0, solved by uninstallation.

Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied

On Windows, a Western DIgital (WD) utility that comes with their external USB drives can interfere with copying UF2 files to the boardnameBOOT drive. Uninstall that utility to fix the problem.

CIRCUITPY Drive Does Not Appear or Disappears Quickly

Kaspersky anti-virus can block the appearance of the CIRCUITPY drive. There has not yet been settings change discovered that prevents this. Complete uninstallation of Kaspersky fixes the problem.

Norton anti-virus can interfere with CIRCUITPY. A user has reported this problem on Windows 7. The user turned off both Smart Firewall and Auto Protect, and CIRCUITPY then appeared.

Sophos Endpoint security software can cause CIRCUITPY to disappear () and the BOOT drive to reappear. It is not clear what causes this behavior.

Device Errors or Problems on Windows

Windows can become confused about USB device installations. This is particularly true of Windows 7 and 8.1. It is recommended () that you upgrade to Windows 10 if possible; an upgrade is probably still free for you: see this link ().

If not, try cleaning up your USB devices. Use Uwe Sieber's Device Cleanup Tool () (on that page, scroll down to "Device Cleanup Tool"). Download and unzip the tool. Unplug all the boards and other USB devices you want to clean up. Run the tool as Administrator. You will see a listing like this, probably with many more devices. It is listing all the USB devices that are not currently attached.
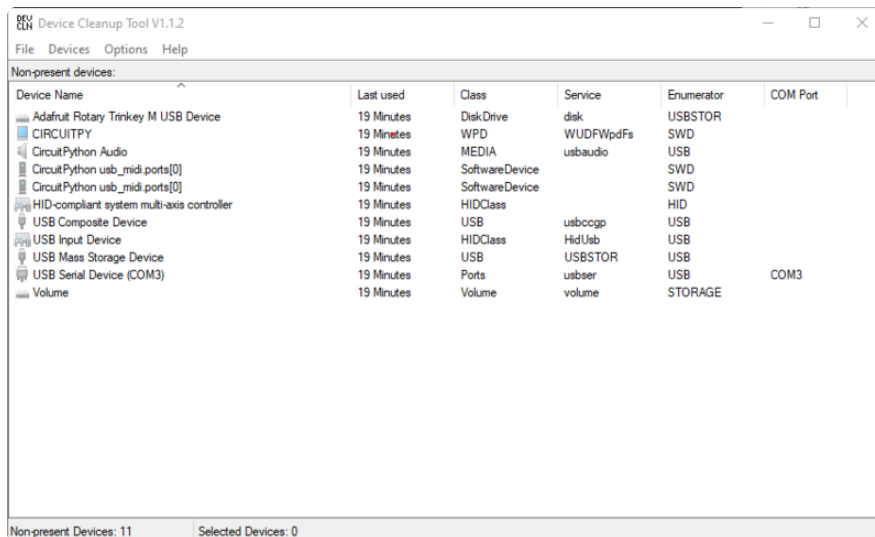
Select all the devices you want to remove, and then press Delete. It is usually safe just to select everything. Any device that is removed will get a fresh install when you plug it in. Using the Device Cleanup Tool also discards all the COM port assignments for the unplugged boards. If you have used many Arduino and CircuitPython boards, you have probably seen higher and higher COM port numbers used, seemingly without end. This will fix that problem.

Serial Console in Mu Not Displaying Anything

There are times when the serial console will accurately not display anything, such as, when no code is currently running, or when code with no serial output is already running before you open the console. However, if you find yourself in a situation where you feel it should be displaying something like an error, consider the following.

Depending on the size of your screen or Mu window, when you open the serial console, the serial console panel may be very small. This can be a problem. A basic CircuitPython error takes 10 lines to display!

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Traceback (most recent call last):
  File "code.py", line 7
SyntaxError: invalid syntax


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

More complex errors take even more lines!

Therefore, if your serial console panel is five lines tall or less, you may only see blank lines or blank lines followed by `Press any key to enter the REPL. Use CTRL-D to reload.` . If this is the case, you need to either mouse over the top of the panel to utilise the option to resize the serial panel, or use the scrollbar on the right side to scroll up and find your message.
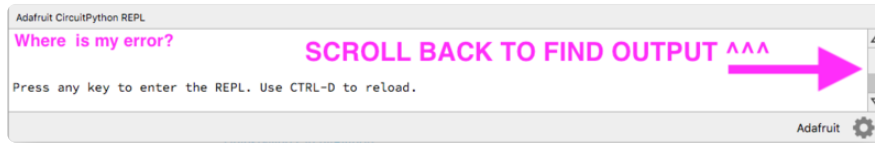
This applies to any kind of serial output whether it be error messages or print statements. So before you start trying to debug your problem on the hardware side, be sure to check that you haven't simply missed the serial messages due to serial output panel height.

code.py Restarts Constantly

CircuitPython will restart code.py if you or your computer writes to something on the CIRCUITPY drive. This feature is called auto-reload, and lets you test a change to your program immediately.

Some utility programs, such as backup, anti-virus, or disk-checking apps, will write to the CIRCUITPY as part of their operation. Sometimes they do this very frequently, causing constant restarts.

Acronis True Image and related Acronis programs on Windows are known to cause this problem. It is possible to prevent this by disabling the " ()Acronis Managed Machine Service Mini" ().

If you cannot stop whatever is causing the writes, you can disable auto-reload by putting this code in boot.py or code.py:

```
import supervisor

supervisor.runtime.autoreload = False
```

CircuitPython RGB Status Light

Nearly all CircuitPython-capable boards have a single NeoPixel or DotStar RGB LED on the board that indicates the status of CircuitPython. A few boards designed before CircuitPython existed, such as the Feather M0 Basic, do not.

Circuit Playground Express and Circuit Playground Bluefruit have multiple RGB LEDs, but do NOT have a status LED. The LEDs are all green when in the bootloader. In versions before 7.0.0, they do NOT indicate any status while running CircuitPython.

CircuitPython 7.0.0 and Later

The status LED blinks were changed in CircuitPython 7.0.0 in order to save battery power and simplify the blinks. These blink patterns will occur on single color LEDs when the board does not have any RGB LEDs. Speed and blink count also vary for this reason.

On start up, the LED will blink YELLOW multiple times for 1 second. Pressing the RESET button (or on Espressif, the BOOT button) during this time will restart the board and then enter safe mode. On Bluetooth capable boards, after the yellow blinks, there will be a set of faster blue blinks. Pressing reset during the BLUE blinks will clear Bluetooth information and start the device in discoverable mode, so it can be used with a BLE code editor.

Once started, CircuitPython will blink a pattern every 5 seconds when no user code is running to indicate why the code stopped:

- 1 GREEN blink: Code finished without error.
- 2 RED blinks: Code ended due to an exception. Check the serial console for details.
- 3 YELLOW blinks: CircuitPython is in safe mode. No user code was run. Check the serial console for safe mode reason.

When in the REPL, CircuitPython will set the status LED to WHITE. You can change the LED color from the REPL. The status indicator will not persist on non-NeoPixel or DotStar LEDs.

The CircuitPython Boot Sequence
Version 7.0 and later
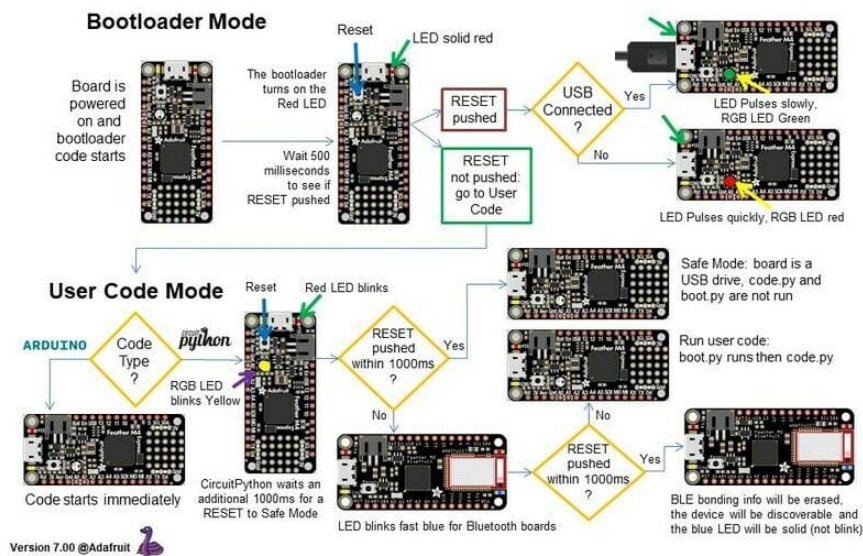
CircuitPython 6.3.0 and earlier

Here's what the colors and blinking mean:

- steady GREEN: code.py (or code.txt, main.py, or main.txt) is running
- pulsing GREEN: code.py (etc.) has finished or does not exist
- steady YELLOW at start up: (4.0.0-alpha.5 and newer) CircuitPython is waiting for a reset to indicate that it should start in safe mode
- pulsing YELLOW: Circuit Python is in safe mode: it crashed and restarted
- steady WHITE: REPL is running
- steady BLUE: boot.py is running

Colors with multiple flashes following indicate a Python exception and then indicate the line number of the error. The color of the first flash indicates the type of error:

- GREEN: IndentationError
- CYAN: SyntaxError
- WHITE: NameError
- ORANGE: OSError
- PURPLE: ValueError
- YELLOW: other error

These are followed by flashes indicating the line number, including place value. WHITE flashes are thousands' place, BLUE are hundreds' place, YELLOW are tens' place,

and CYAN are one's place. So for example, an error on line 32 would flash YELLOW three times and then CYAN two times. Zeroes are indicated by an extra-long dark gap.

The CircuitPython Boot Sequence
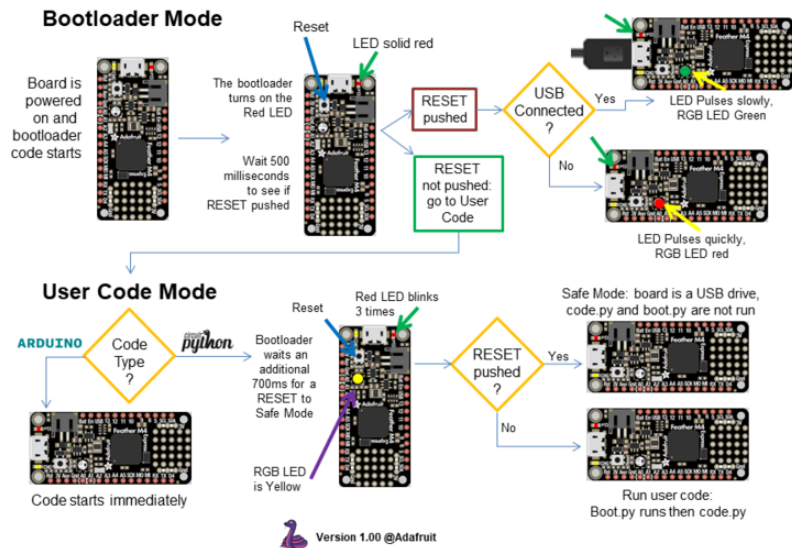
Serial console showing `ValueError: Incompatible .mpy file`

This error occurs when importing a module that is stored as a .mpy binary file that was generated by a different version of CircuitPython than the one its being loaded into. In particular, the mpy binary format changed between CircuitPython versions 6.x and 7.x, 2.x and 3.x, and 1.x and 2.x.

So, for instance, if you upgraded to CircuitPython 7.x from 6.x you'll need to download a newer version of the library that triggered the error on `import`. All libraries are available in the Adafruit bundle ().

CIRCUITPY Drive Issues

You may find that you can no longer save files to your CIRCUITPY drive. You may find that your CIRCUITPY stops showing up in your file explorer, or shows up as NO_NAME. These are indicators that your filesystem has issues. When the CIRCUITPY disk is not safely ejected before being reset by the button or being disconnected from USB, it may corrupt the flash drive. It can happen on Windows, Mac or Linux, though it is more common on Windows.

Be aware, if you have used Arduino to program your board, CircuitPython is no longer able to provide the USB services. You will need to reload CircuitPython to resolve this situation.

The easiest first step is to reload CircuitPython. Double-tap reset on the board so you get a boardnameBOOT drive rather than a CIRCUITPY drive, and copy the latest version of CircuitPython (.uf2) back to the board. This may restore CIRCUITPY functionality.

If reloading CircuitPython does not resolve your issue, the next step is to try putting the board into safe mode.

Safe Mode

Whether you've run into a situation where you can no longer edit your code.py on your CIRCUITPY drive, your board has gotten into a state where CIRCUITPY is read-only, or you have turned off the CIRCUITPY drive altogether, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in boot.py (where you can set CIRCUITPY read-only or turn it off completely). Second, it does not run the code in code.py. And finally, it does not automatically soft-reload when data is written to the CIRCUITPY drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the CIRCUITPY drive.

Entering Safe Mode in CircuitPython 7.x and Later

To enter safe mode when using CircuitPython 7.x, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a "slow" double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

Entering Safe Mode in CircuitPython 6.x

To enter safe mode when using CircuitPython 6.x, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 700ms. On some boards, the onboard status LED (highlighted in green above) will turn solid yellow during this time. If you press reset during that 700ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

Once you've entered safe mode successfully in CircuitPython 6.x, the LED will pulse yellow.

If you successfully enter safe mode on CircuitPython 7.x, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot.
Press again to exit safe mode.

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

You can now edit the contents of the CIRCUITPY drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

At this point, you'll want to remove any user code in code.py and, if present, the boot.py file from CIRCUITPY. Once removed, tap the reset button, or unplug and plug in your board, to restart CircuitPython. This will restart the board and may resolve your drive issues. If resolved, you can begin coding again as usual.

If safe mode does not resolve your issue, the board must be completely erased and CircuitPython must be reloaded onto the board.

> You WILL lose everything on the board when you complete the following steps. If possible, make a copy of your code before continuing.

To erase CIRCUITPY: `storage.erase_filesystem()`

CircuitPython includes a built-in function to erase and reformat the filesystem. If you have a version of CircuitPython older than 2.3.0 on your board, you can [update to the newest version ()](#) to do this.

1. [Connect to the CircuitPython REPL ()](#) using Mu or a terminal program.
2. Type the following into the REPL:

```
>>> import storage
>>> storage.erase_filesystem()
```

CIRCUITPY will be erased and reformatted, and your board will restart. That's it!

Erase CIRCUITPY Without Access to the REPL

If you can't access the REPL, or you're running a version of CircuitPython previous to 2.3.0 and you don't want to upgrade, there are options available for some specific boards.

The options listed below are considered to be the "old way" of erasing your board. The method shown above using the REPL is highly recommended as the best method for erasing your board.

> If at all possible, it is recommended to use the REPL to erase your CIRCUITPY drive. The REPL method is explained above.

For the specific boards listed below:

If the board you are trying to erase is listed below, follow the steps to use the file to erase your board.

1. Download the correct erase file:

Circuit Playground Express

Feather M0 Express

Feather M4 Express

Metro M0 Express

Metro M4 Express QSPI Eraser

Trellis M4 Express (QSPI)

Grand Central M4 Express (QSPI)

PyPortal M4 Express (QSPI)

Circuit Playground Bluefruit (QSPI)

Monster M4SK (QSPI)

PyBadge/PyGamer QSPI Eraser.UF2

CLUE_Flash_Erase.UF2

Matrix_Portal_M4_(QSPI).UF2

RP2040 boards (flash_nuke.uf2)

2. Double-click the reset button on the board to bring up the boardnameBOOT drive.

3. Drag the erase .uf2 file to the boardnameBOOT drive.

4. The status LED will turn yellow or blue, indicating the erase has started.

5. After approximately 15 seconds, the status LED will light up green. On the NeoTrellis M4 this is the first NeoPixel on the grid

6. Double-click the reset button on the board to bring up the boardnameBOOT drive.

7. Drag the appropriate latest release of CircuitPython () .uf2 file to the boardnameBOOT drive.

It should reboot automatically and you should see CIRCUITPY in your file explorer again.

If the LED flashes red during step 5, it means the erase has failed. Repeat the steps starting with 2.

If you haven't already downloaded the latest release of CircuitPython for your board, check out the installation page (). You'll also need to load your code and reinstall your libraries!

For SAMD21 non-Express boards that have a UF2 bootloader:

Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that have a UF2 bootloader include Trinket M0, GEMMA M0, QT Py M0, and the SAMD21-based Trinkey boards.

If you are trying to erase a SAMD21 non-Express board, follow these steps to erase your board.

1. Download the erase file:

<div style="text-align:center;">

SAMD21 non-Express Boards

</div>

2. Double-click the reset button on the board to bring up the boardnameBOOT drive.
3. Drag the erase .uf2 file to the boardnameBOOT drive.
4. The boot LED will start flashing again, and the boardnameBOOT drive will reappear.
5. Drag the appropriate latest release CircuitPython () .uf2 file to the boardnameBOOT drive.

It should reboot automatically and you should see CIRCUITPY in your file explorer again.

If you haven't already downloaded the latest release of CircuitPython for your board, check out the installation page () YYou'll also need to load your code and reinstall your libraries!

For SAMD21 non-Express boards that do not have a UF2 bootloader:

Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that do not have a UF2 bootloader include the Feather M0 Basic Proto, Feather Adalogger, or the Arduino Zero.

If you are trying to erase a non-Express board that does not have a UF2 bootloader, [follow these directions to reload CircuitPython using `bossac` ()](), which will erase and re-create CIRCUITPY.

Running Out of File Space on SAMD21 Non-Express Boards

Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. This includes boards like the Trinket M0, GEMMA M0, QT Py M0, and the SAMD21-based Trinkey boards.

The file system on the board is very tiny. (Smaller than an ancient floppy disk.) So, its likely you'll run out of space but don't panic! There are a number of ways to free up space.
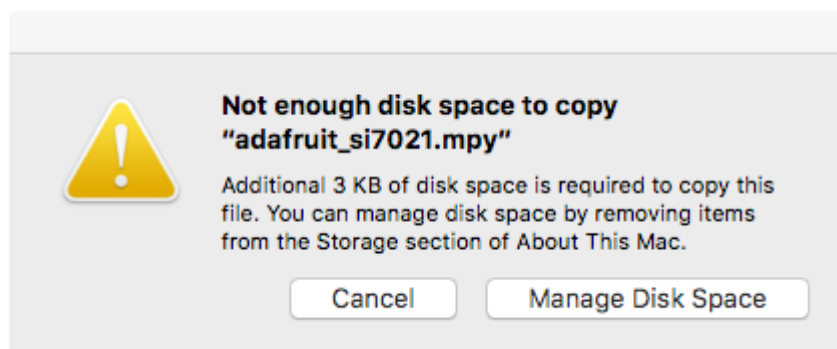
Delete something!

The simplest way of freeing up space is to delete files from the drive. Perhaps there are libraries in the lib folder that you aren't using anymore or test code that isn't in use. Don't delete the lib folder completely, though, just remove what you don't need.

The board ships with the Windows 7 serial driver too! Feel free to delete that if you don't need it or have already installed it. It's ~12KiB or so.

Use tabs

One unique feature of Python is that the indentation of code matters. Usually the recommendation is to indent code with four spaces for every indent. In general, that is recommended too. However, one trick to storing more human-readable code is to use a single tab character for indentation. This approach uses 1/4 of the space for indentation and can be significant when you're counting bytes.

On MacOS?

MacOS loves to generate hidden files. Luckily you can disable some of the extra hidden files that macOS adds by running a few commands to disable search indexing and create zero byte placeholders. Follow the steps below to maximize the amount of space available on macOS.

Prevent & Remove MacOS Hidden Files

First find the volume name for your board. With the board plugged in run this command in a terminal to list all the volumes:

```
ls -l /Volumes
```

Look for a volume with a name like CIRCUITPY (the default for CircuitPython). The full path to the volume is the /Volumes/CIRCUITPY path.

Now follow the [steps from this question ()](steps from this question) to run these terminal commands that stop hidden files from being created on the board:

```
mdutil -i off /Volumes/CIRCUITPY
cd /Volumes/CIRCUITPY
rm -rf .{,_.}{fseventsd,Spotlight-V*,Trashes}
mkdir .fseventsd
touch .fseventsd/no_log .metadata_never_index .Trashes
cd -
```

Replace /Volumes/CIRCUITPY in the commands above with the full path to your board's volume if it's different. At this point all the hidden files should be cleared from the board and some hidden files will be prevented from being created.

Alternatively, with CircuitPython 4.x and above, the special files and folders mentioned above will be created automatically if you erase and reformat the filesystem. WARNING: Save your files first! Do this in the REPL:

```
>>> import storage
>>> storage.erase_filesystem()
```

However there are still some cases where hidden files will be created by MacOS. In
particular if you copy a file that was downloaded from the internet it will have special
metadata that MacOS stores as a hidden file. Luckily you can run a copy command
from the terminal to copy files without this hidden metadata file. See the steps below.

Copy Files on MacOS Without Creating Hidden Files

Once you've disabled and removed hidden files with the above commands on macOS
you need to be careful to copy files to the board with a special command that
prevents future hidden files from being created. Unfortunately you cannot use drag
and drop copy in Finder because it will still create these hidden extended attribute
files in some cases (for files downloaded from the internet, like Adafruit's modules).

To copy a file or folder use the -X option for the cp command in a terminal. For
example to copy a file_name.mpy file to the board use a command like:

```
cp -X file_name.mpy /Volumes/CIRCUITPY
```

(Replace file_name.mpy with the name of the file you want to copy.)

Or to copy a folder and all of the files and folders contained within, use a command
like:

```
cp -rX folder_to_copy /Volumes/CIRCUITPY
```

If you are copying to the lib folder, or another folder, make sure it exists before
copying.

```
# if lib does not exist, you'll create a file named lib !
cp -X file_name.mpy /Volumes/CIRCUITPY/lib
# This is safer, and will complain if a lib folder does not exist.
cp -X file_name.mpy /Volumes/CIRCUITPY/lib/
```

Other MacOS Space-Saving Tips

If you'd like to see the amount of space used on the drive and manually delete hidden
files here's how to do so. First, move into the Volumes/ directory with `cd /Volumes/`,
and then list the amount of space used on the CIRCUITPY drive with the `df`
command.

```
Last login: Thu Oct 28 17:19:15 on ttys008

7039 kattni@robocrepe:~ $ cd /Volumes/

7040 kattni@robocrepe:Volumes $ df -h CIRCUITPY/
Filesystem     Size   Used  Avail Capacity iused ifree %iused  Mounted on
/dev/disk2s1   47Ki   46Ki  1.0Ki    98%     512     0   100%   /Volumes/CIRCUITPY

7041 kattni@robocrepe:Volumes $
```

That's not very much space left! The next step is to show a list of the files currently on the CIRCUITPY drive, including the hidden files, using the `ls` command. You cannot use Finder to do this, you must do it via command line!

```
7041 kattni@robocrepe:Volumes $ ls -a CIRCUITPY/
.                         ._trinket_code.py     code.py
..                        .fseventsd            lib
.Trashes                  .idea                 original_code.py
._code.py                 .metadata_never_index trinket_code.py
._original_code.py        boot_out.txt

7042 kattni@robocrepe:Volumes $
```

There are a few of the hidden files that MacOS loves to generate, all of which begin with a ._ before the file name. Remove the ._ files using the `rm` command. You can remove them all once by running `rm CIRCUITPY/._*`. The `*` acts as a wildcard to apply the command to everything that begins with ._ at the same time.

```
7042 kattni@robocrepe:Volumes $ rm CIRCUITPY/._*

7043 kattni@robocrepe:Volumes $
```

Finally, you can run `df` again to see the current space used.

```
7043 kattni@robocrepe:Volumes $ df -h CIRCUITPY/
Filesystem     Size   Used  Avail Capacity iused ifree %iused  Mounted on
/dev/disk2s1   47Ki   34Ki  13Ki     73%     512     0   100%   /Volumes/CIRCUITPY

7044 kattni@robocrepe:Volumes $
```

Nice! You have 12Ki more than before! This space can now be used for libraries and code!

Device Locked Up or Boot Looping

In rare cases, it may happen that something in your code.py or boot.py files causes the device to get locked up, or even go into a boot loop. A boot loop occurs when the board reboots repeatedly and never fully loads. These are not caused by your everyday Python exceptions, typically it's the result of a deeper problem within CircuitPython. In this situation, it can be difficult to recover your device if CIRCUITPY is not allowing you to modify the code.py or boot.py files. Safe mode is one recovery option. When the device boots up in safe mode it will not run the code.py or boot.py

scripts, but will still connect the CIRCUITPY drive so that you can remove or modify those files as needed.

The method used to manually enter safe mode can be different for different devices. It is also very similar to the method used for getting into bootloader mode, which is a different thing. So it can take a few tries to get the timing right. If you end up in bootloader mode, no problem, you can try again without needing to do anything else.

For most devices:
Press the reset button, and then when the RGB status LED blinks yellow, press the reset button again. Since your reaction time may not be that fast, try a "slow" double click, to catch the yellow LED on the second click.

For ESP32-S2 based devices:
Press and release the reset button, then press and release the boot button about 3/4 of a second later.

Refer to the diagrams above for boot sequence details.

"Uninstalling" CircuitPython

A lot of our boards can be used with multiple programming languages. For example, the Circuit Playground Express can be used with MakeCode, Code.org CS Discoveries, CircuitPython and Arduino.

Maybe you tried CircuitPython and want to go back to MakeCode or Arduino? Not a problem. You can always remove or reinstall CircuitPython whenever you want! Heck, you can change your mind every day!

There is nothing to uninstall. CircuitPython is "just another program" that is loaded onto your board. You simply load another program (Arduino or MakeCode) and it will overwrite CircuitPython.

Backup Your Code

Before replacing CircuitPython, don't forget to make a backup of the code you have on the CIRCUITPY drive. That means your code.py any other files, the lib folder etc. You may lose these files when you remove CircuitPython, so backups are key! Just drag the files to a folder on your laptop or desktop computer like you would with any USB drive.

Moving Circuit Playground Express to MakeCode

On the Circuit Playground Express (this currently does NOT apply to Circuit Playground Bluefruit), if you want to go back to using MakeCode, it's really easy. Visit makecode.adafruit.com () and find the program you want to upload. Click Download to download the .uf2 file that is generated by MakeCode.

Now double-click your CircuitPython board until you see the onboard LED(s) turn green and the ...BOOT directory shows up.
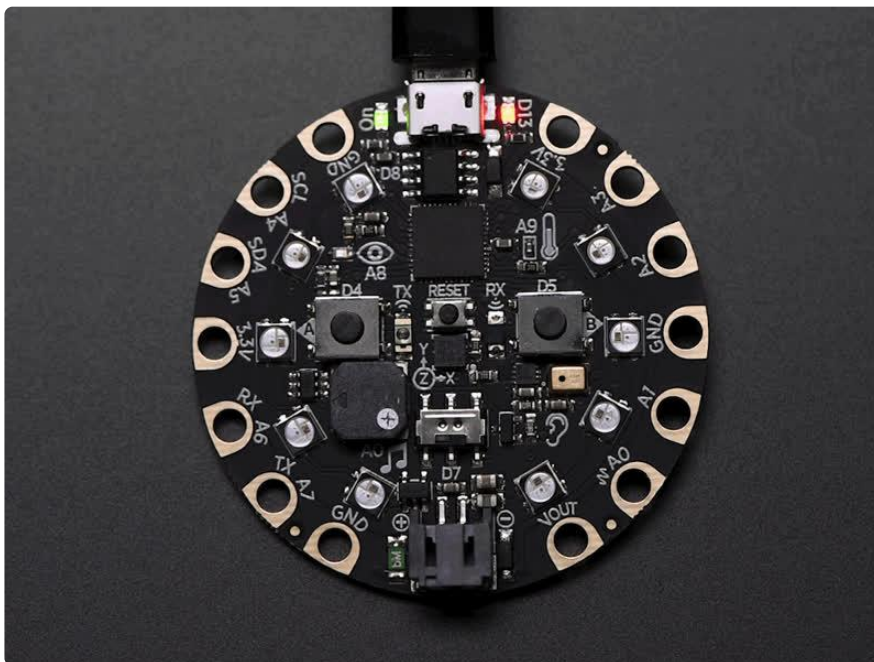
Then find the downloaded MakeCode .uf2 file and drag it to the CPLAYBOOT drive.
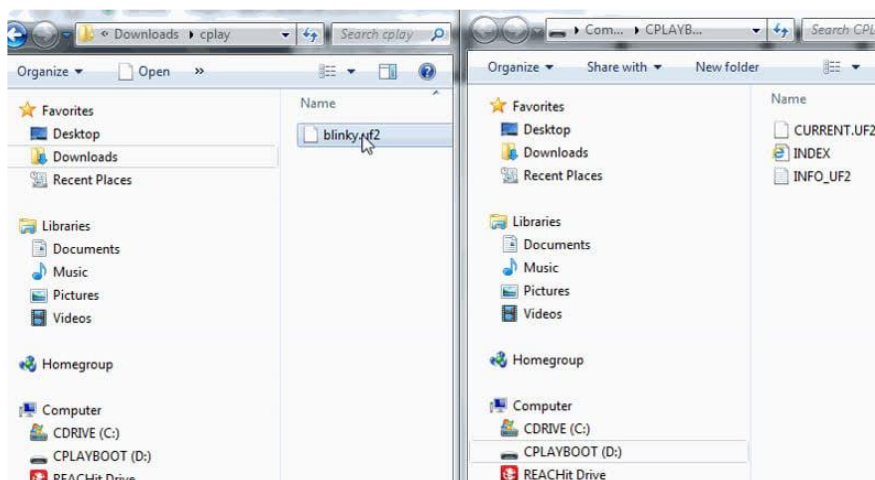
Your MakeCode is now running and CircuitPython has been removed. Going forward you only have to single click the reset button to get to CPLAYBOOT. This is an idiosyncrasy of MakeCode.

Moving to Arduino

If you want to use Arduino instead, you just use the Arduino IDE to load an Arduino program. Here's an example of uploading a simple "Blink" Arduino program, but you don't have to use this particular program.

Start by plugging in your board, and double-clicking reset until you get the green onboard LED(s).

Within Arduino IDE, select the matching board, say Circuit Playground Express.
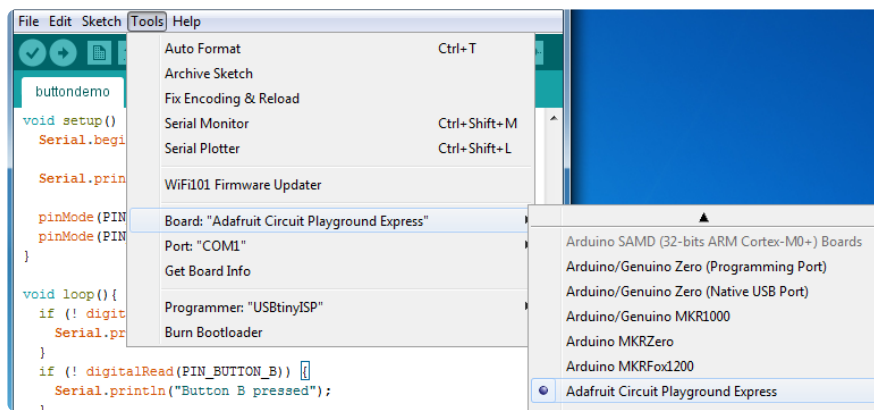
Select the correct matching Port:
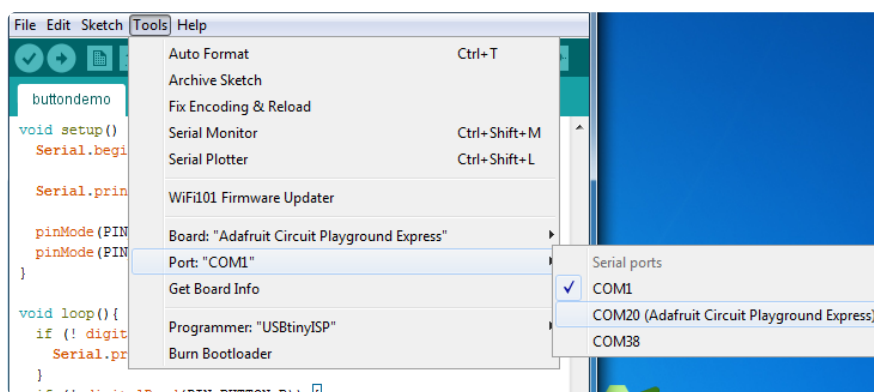
Create a new simple Blink sketch example:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}
```

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Make sure the LED(s) are still green, then click Upload to upload Blink. Once it has uploaded successfully, the serial Port will change so re-select the new Port!

Once Blink is uploaded you should no longer need to double-click to enter bootloader mode. Arduino will automatically reset when you upload.

CircuitPython Essentials

CircuitPython Essentials ()

Clue Library Documentation

Clue Library Documentation ()

CLUE CircuitPython Demos

The Adafruit CLUE is packed with tons of sensor and inputs. You can use these in many combinations to create all kinds of fun projects. We've included a series of CircuitPython demos for the CLUE to get you started. Take a look!

(For the high-level CLUE library, check out the docs here ().)

CLUE Spirit Level
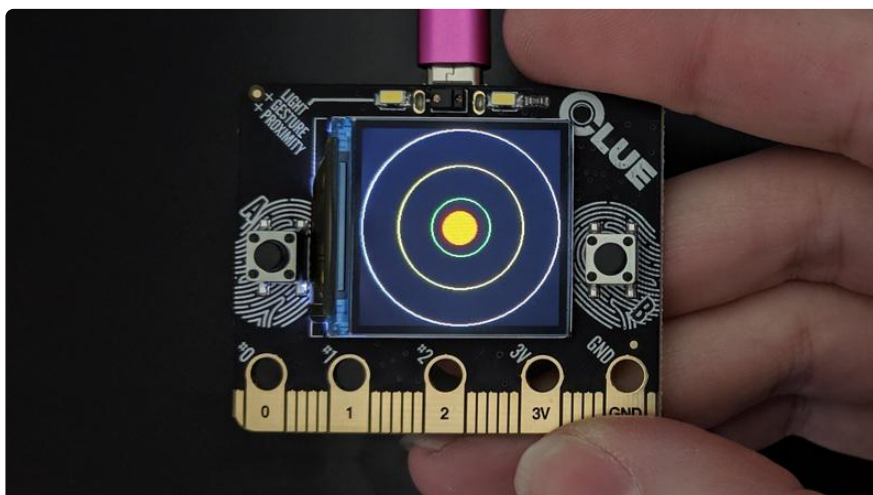
The Adafruit CLUE comes with a built in accelerometer for measuring acceleration. For more information on how that works, check out Wikipedia ().

Using CircuitPython, we can create a spirit level using generated shapes and the accelerometer data. This example generates circles as guide lines and and outline, and a dot as the "bubble". Move the board around to watch the bubble "float" to the top!

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 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
   ▼ 📁 clue_slideshow
        🖼 adafruit_products_Blinka_240x240.bmp
        📄 adafruit_products_Blinka_240x240.bmp.license
        🖼 adafruit_products_Blinka_Love_240x240.bmp
        📄 adafruit_products_Blinka_Love_240x240.bmp.license
        🖼 adafruit_products_Blinka_Ouroboros_240x240.bmp
        📄 adafruit_products_Blinka_Ouroboros_240x240.bmp.license
        📄 clue_slideshow.py
   ▼ 📁 lib
      ► 📁 adafruit_apds9960
      ► 📁 adafruit_bitmap_font
      ► 📁 adafruit_bus_device
      ► 📁 adafruit_display_shapes
      ► 📁 adafruit_display_text
      ► 📁 adafruit_lsm6ds
      ► 📁 adafruit_register
        📄 adafruit_bmp280.mpy
        📄 adafruit_clue.mpy
        📄 adafruit_lis3mdl.mpy
        📄 adafruit_pixelbuf.mpy
        📄 adafruit_sht31d.mpy
        📄 neopixel.mpy
```

```python
# SPDX-FileCopyrightText: 2019 Kattni Rembor, written for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""CLUE Spirit Level Demo"""
import board
import displayio
from adafruit_display_shapes.circle import Circle
from adafruit_clue import clue

display = board.DISPLAY
clue_group = displayio.Group()

outer_circle = Circle(120, 120, 119, outline=clue.WHITE)
middle_circle = Circle(120, 120, 75, outline=clue.YELLOW)
inner_circle = Circle(120, 120, 35, outline=clue.GREEN)
clue_group.append(outer_circle)
clue_group.append(middle_circle)
clue_group.append(inner_circle)

x, y, _ = clue.acceleration
bubble_group = displayio.Group()
level_bubble = Circle(int(x + 120), int(y + 120), 20, fill=clue.RED,
outline=clue.RED)
bubble_group.append(level_bubble)

clue_group.append(bubble_group)
display.show(clue_group)

while True:
    x, y, _ = clue.acceleration
    bubble_group.x = int(x * -10)
    bubble_group.y = int(y * -10)
```

Let's take a look at the code.

First, we import the necessary libraries and modules. Then we create the display object for later use.

Next we create the `clue_group` that will hold all of the objects we plan to display. Then we create our outline and guide lines, and add them to the group.

Next we get the initial acceleration values. For this, we only care about x and y, but as `acceleration` is an x, y, z value, we must unpack three values from it. The `_` takes the place of z which we never use.

Then we create the `bubble_group` to hold the bubble. With the circles, as they are static, we could simply append them into a group. The bubble needs to move, so it must be in its own group that we will add to the same group as the circles once created. Next we create the level bubble. Instead of a hard coded initial location, its initial location is based on the initial x, y value that we obtained. Then we add the bubble to the bubble group.

Finally, we add the `bubble_group` to the `clue_group`, and tell the display to `show()` everything contained within, which is all of the shapes we created.

Inside the loop, we get an updated x, y value. Since it is in the loop, it will continue to update. Then we set the `bubble_group` x and y locations to be the constantly updating x and y values from the accelerometer. We multiply them by -10 for two reasons. The multiplication by 10 is to extend the values to utilise the entire display - when moving slowly, x and y values are approximately -10 to 10, so without the multiplication, the dot would move in a small space in the center of the display. The negative is to cause the bubble to move "upwards" like an actual level bubble would, opposite the direction it would otherwise move based on the actual x and y values.

That's what goes into making a spirit level with CircuitPython and CLUE!

CLUE Temperature and Humidity Monitor
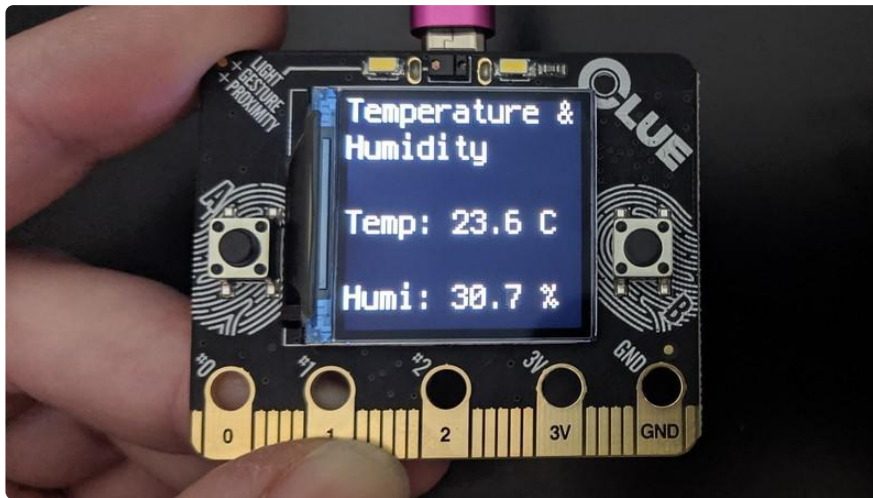
The Adafruit CLUE has a temperature and humidity sensor that reads the temperature in degrees Celsius and the relative humidity in percent. To learn more about relative humidity, check Wikipedia ().

With CircuitPython, it's easy to build a color-coded temperature and humidity monitor with customisable ranges and an optional audible alarm. Set the min and max temperature and humidity, optionally enable the audible alarm and you're all set to know when your environment is outside your comfort zone!

You'll want to set the `min_temperature` and `max_temperature` to your desired range in degrees Celsius, and the `min_humidity` and `max_humidity` to your desired range in percent. If you want the alarm to sound when the temperature or humidity is outside the specified range, enable it by setting `alarm_enable = True`. Then everything is ready to go!
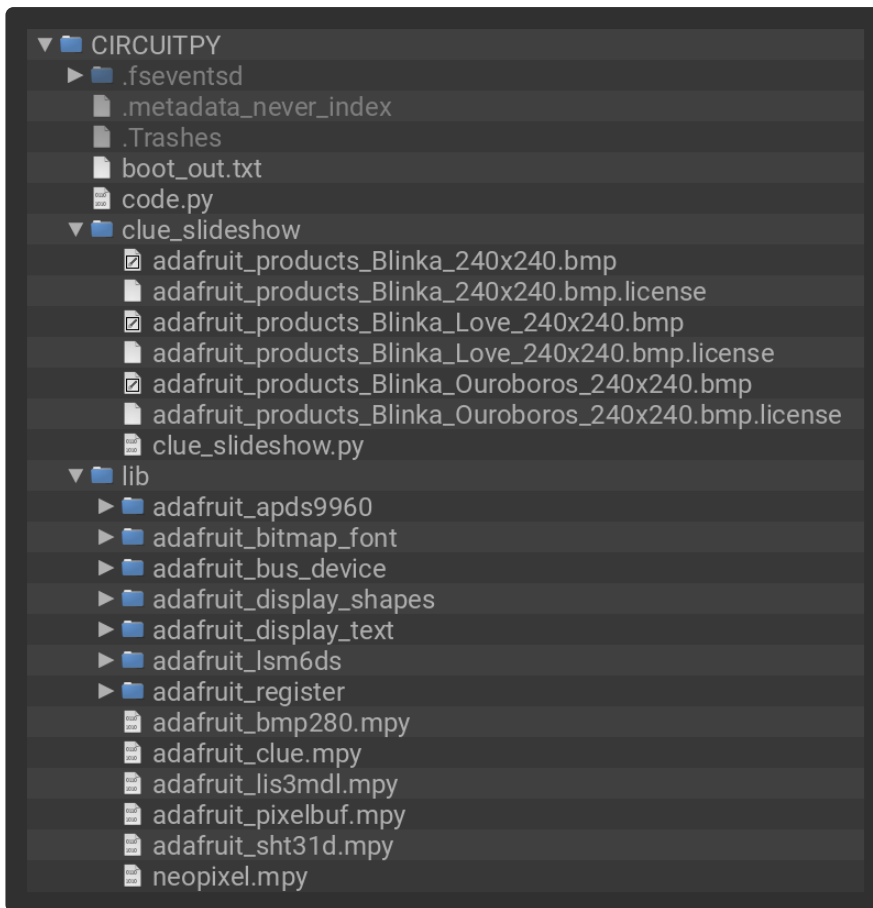
Installing Project Code

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 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
   ▼ 📁 clue_slideshow
       🖼 adafruit_products_Blinka_240x240.bmp
       📄 adafruit_products_Blinka_240x240.bmp.license
       🖼 adafruit_products_Blinka_Love_240x240.bmp
       📄 adafruit_products_Blinka_Love_240x240.bmp.license
       🖼 adafruit_products_Blinka_Ouroboros_240x240.bmp
       📄 adafruit_products_Blinka_Ouroboros_240x240.bmp.license
       📄 clue_slideshow.py
   ▼ 📁 lib
     ▶ 📁 adafruit_apds9960
     ▶ 📁 adafruit_bitmap_font
     ▶ 📁 adafruit_bus_device
     ▶ 📁 adafruit_display_text
     ▶ 📁 adafruit_lsm6ds
     ▶ 📁 adafruit_register
       📄 adafruit_bmp280.mpy
       📄 adafruit_clue.mpy
       📄 adafruit_lis3mdl.mpy
       📄 adafruit_pixelbuf.mpy
       📄 adafruit_sht31d.mpy
       📄 neopixel.mpy
```
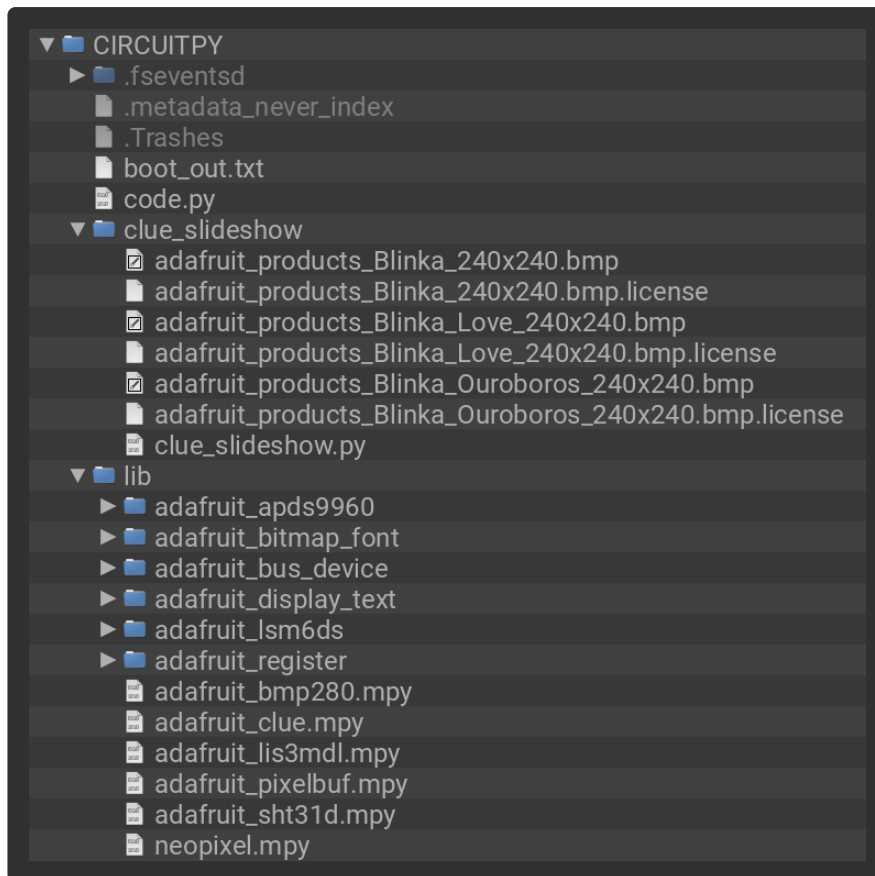
```python
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""Monitor customisable temperature and humidity ranges, with an optional audible
alarm tone."""
from adafruit_clue import clue

# Set desired temperature range in degrees Celsius.
min_temperature = 24
max_temperature = 30

# Set desired humidity range in percent.
min_humidity = 20
max_humidity = 65

# Set to true to enable audible alarm tone.
alarm_enable = False

clue_display = clue.simple_text_display(text_scale=3, colors=(clue.WHITE,))

clue_display[0].text = "Temperature &"
clue_display[1].text = "Humidity"

while True:
    alarm = False

    temperature = clue.temperature
    humidity = clue.humidity

    clue_display[3].text = "Temp: {:.1f} C".format(temperature)
    clue_display[5].text = "Humi: {:.1f} %".format(humidity)

    if temperature < min_temperature:
        clue_display[3].color = clue.BLUE
        alarm = True
    elif temperature > max_temperature:
```

```
            clue_display[3].color = clue.RED
            alarm = True
        else:
            clue_display[3].color = clue.WHITE

        if humidity < min_humidity:
            clue_display[5].color = clue.BLUE
            alarm = True
        elif humidity > max_humidity:
            clue_display[5].color = clue.RED
            alarm = True
        else:
            clue_display[5].color = clue.WHITE
        clue_display.show()

        if alarm and alarm_enable:
            clue.start_tone(2000)
        else:
            clue.stop_tone()
```

Let's take a look at the code.

First you set the temperature and humidity ranges that you'd like to monitor, and optionally enable the audible alarm. Next you set up the text display to prepare to add the lines of text. We scale it to 3 and set the initial color of all the text white.

Note that `colors` expects a tuple. Tuples in Python come in parentheses () with comma separators. If you have two values, a tuple would look like `(1.0, 3.14)` Since we have only one value, we need to have it print out like `(1.0,)` note the parentheses around the number, and the comma after the number. Therefore if you are only providing a single color for all lines of text, you must include the extra comma after the single color to make it a single member tuple, e.g. `colors=(clue.WHITE,)`.

Then we add the first two lines of text which are the title.

Inside the loop, we set the alarm variable to False. We'll use this variable throughout the program to determine whether the alarm should be going off. Next, we create a variable to hold the temperature data, and one to hold the humidity data. Then we create the next two lines of text to show the temperature and humidity data. Notice that we skip line 2 and line 4 - this is a feature to allow you to space out text on the display without creating empty lines.

Now we have two if blocks that look very similar, one for temperature and one for humidity. First we check to see if the temperature is less than the earlier-set minimum temperature, and if it is, we change the text to blue and set alarm to True. Then we check to see if the temperature is greater than the earlier-set maximum temperature, and if it is, change the text to red and set alarm to True. Otherwise, we keep the text white. Then we repeat the same steps for humidity.

Finally, we have one last if block. This block checks to see if BOTH `alarm` and `alarm_enable` are True. If so, play a 2000Hz tone which functions as the audible alarm. Otherwise, no tone is played.

That's what goes into making a temperature and humidity monitor using CircuitPython and CLUE!
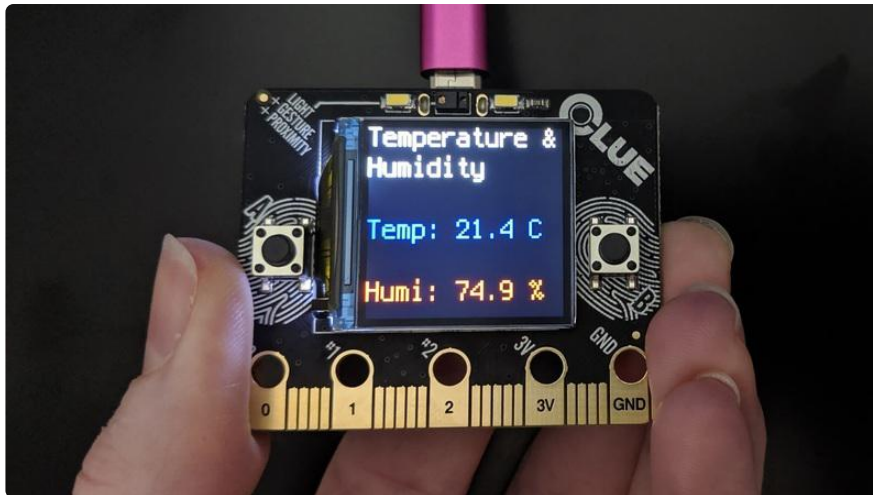
> Note: The temperature will read higher than ambient temperature because of the backlight on the CLUE display.

CLUE Height Calculator
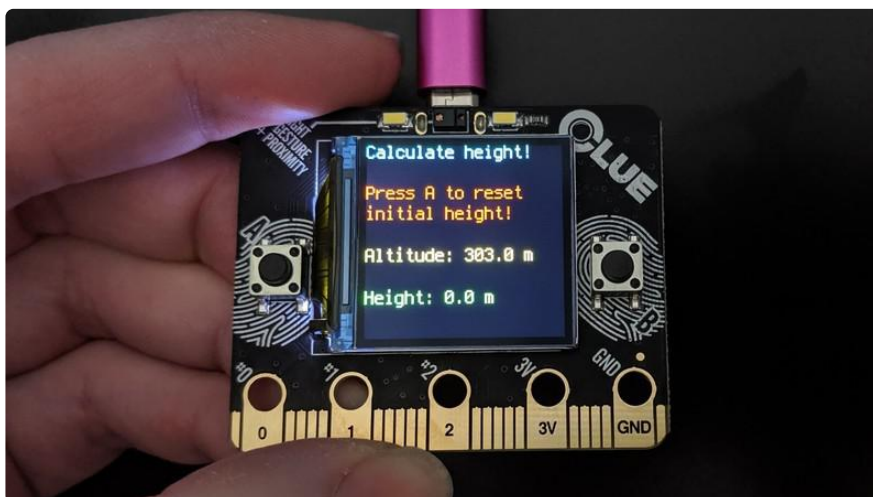
The Adafruit CLUE has a barometric pressure sensor that uses the pressure readings to calculate altitude. For more information on how this works, check out Wikipedia ().

With CircuitPython and a little math, it is possible to use the altitude reading to calculate the height of an object using your CLUE! Start with the CLUE at the bottom

of the object, and press button A to set the initial reading, and then lift your CLUE up to the top of the object to get your reading.

Remember that you must have the necessary libraries installed. Verify that your lib folder matches the list found on the CLUE CircuitPython Libraries page () before continuing.

To get the most accurate altitude reading, you'll want to find out the sea level pressure at your location. A google search should provide you with a number of options. Make sure you convert the results to hPa. Then set `clue.sea_level_pressure =` to the sea level pressure at your location in hPa.

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 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
  ▼ 📁 clue_slideshow
      🖼 adafruit_products_Blinka_240x240.bmp
      📄 adafruit_products_Blinka_240x240.bmp.license
      🖼 adafruit_products_Blinka_Love_240x240.bmp
      📄 adafruit_products_Blinka_Love_240x240.bmp.license
      🖼 adafruit_products_Blinka_Ouroboros_240x240.bmp
      📄 adafruit_products_Blinka_Ouroboros_240x240.bmp.license
      📄 clue_slideshow.py
  ▼ 📁 lib
    ▶ 📁 adafruit_apds9960
    ▶ 📁 adafruit_bitmap_font
    ▶ 📁 adafruit_bus_device
    ▶ 📁 adafruit_display_text
    ▶ 📁 adafruit_lsm6ds
    ▶ 📁 adafruit_register
      📄 adafruit_bmp280.mpy
      📄 adafruit_clue.mpy
      📄 adafruit_lis3mdl.mpy
      📄 adafruit_pixelbuf.mpy
      📄 adafruit_sht31d.mpy
      📄 neopixel.mpy
```

```python
# SPDX-FileCopyrightText: 2019 Kattni Rembor, written for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""Calculate the height of an object. Press button A to reset initial height and
then lift the
CLUE to find the height."""
from adafruit_clue import clue

# Set to the sea level pressure in hPa at your location for the most accurate
altitude measurement.
clue.sea_level_pressure = 1015

clue_display = clue.simple_text_display(
    text_scale=2,
    colors=(clue.CYAN, 0, clue.RED, clue.RED, 0, clue.YELLOW, 0, clue.GREEN),
)

initial_height = clue.altitude

clue_display[0].text = "Calculate height!"
clue_display[2].text = "Press A to reset"
clue_display[3].text = "initial height!"

while True:
    if clue.button_a:
        initial_height = clue.altitude
        clue.pixel.fill(clue.RED)
    else:
        clue.pixel.fill(0)

    clue_display[5].text = "Altitude: {:.1f} m".format(clue.altitude)
    clue_display[7].text = "Height: {:.1f} m".format(clue.altitude - initial_height)
    clue_display.show()
```

Let's take a look at the code.

First you set the sea level pressure at your location. Then you set up the text display to prepare to add your lines of text. Note that we don't use every line, so to set the colors on only the lines we used, we include a `0` as the color for the unused lines - this is easier and cleaner than setting them to a color! Then we get an initial height reading. The first three lines of text on the display are not dynamic in any way and can be set outside the loop.

Inside the loop, we check to see if button A is pressed, and if it is, we reset the initial height reading to the current altitude reading. This allows for you to reset the initial height reading while the program is running. As well, we turn the NeoPixel LED on the back of the board red when button A is pressed, otherwise we turn it off. Then we display the current altitude.

This is followed by the current height which is the change in altitude from the initial height reading, which is to say, our simple math: `altitude - initial_height`.

Then we call `show()` to make all of it show up on the display.

That's all there is to creating a height calculator with CircuitPython and CLUE!
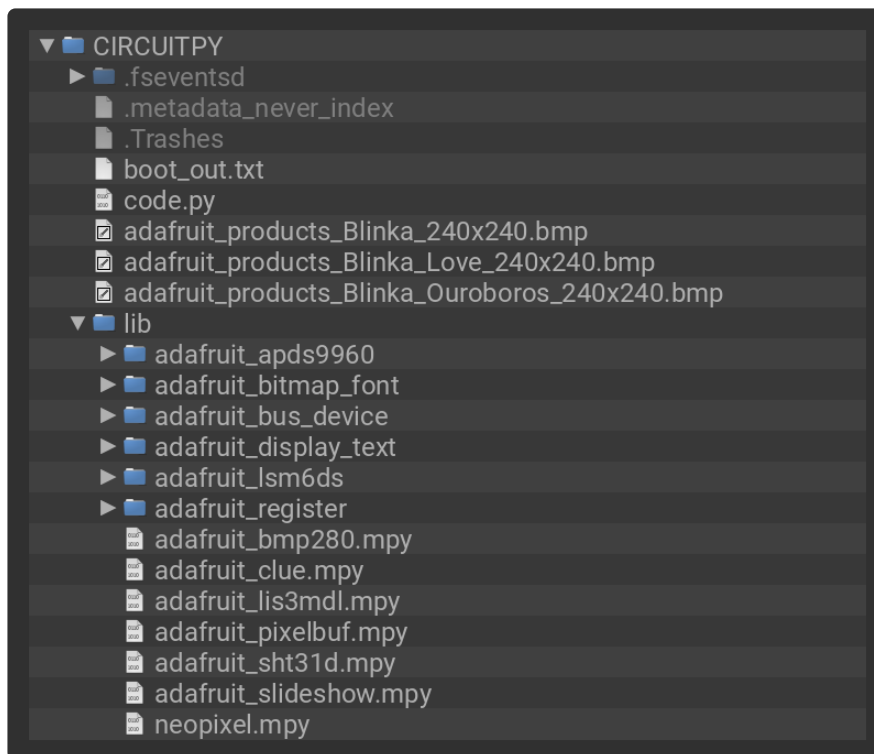

CLUE Slideshow

The Adafruit CLUE comes with built in buttons and a display. Using CircuitPython, the Adafruit CircuitPython CLUE and Adafruit CircuitPython Slideshow libraries, and the built in buttons and display, we can easily make an interactive slideshow.

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 examples/ clue_slideshow/ 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
    🖼 adafruit_products_Blinka_240x240.bmp
    🖼 adafruit_products_Blinka_Love_240x240.bmp
    🖼 adafruit_products_Blinka_Ouroboros_240x240.bmp
  ▼ 📁 lib
    ▶ 📁 adafruit_apds9960
    ▶ 📁 adafruit_bitmap_font
    ▶ 📁 adafruit_bus_device
    ▶ 📁 adafruit_display_text
    ▶ 📁 adafruit_lsm6ds
    ▶ 📁 adafruit_register
      📄 adafruit_bmp280.mpy
      📄 adafruit_clue.mpy
      📄 adafruit_lis3mdl.mpy
      📄 adafruit_pixelbuf.mpy
      📄 adafruit_sht31d.mpy
      📄 adafruit_slideshow.mpy
      📄 neopixel.mpy
```

```python
# SPDX-FileCopyrightText: 2019 Kattni Rembor, written for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""Display a series of bitmaps using the buttons to advance through the list. To
use: place
supported bitmap files on your CIRCUITPY drive, then press the buttons on your CLUE
to advance
through them.

Requires the Adafruit CircuitPython Slideshow library!"""

from adafruit_slideshow import SlideShow, PlayBackDirection
from adafruit_clue import clue

slideshow = SlideShow(clue.display, auto_advance=False)
```
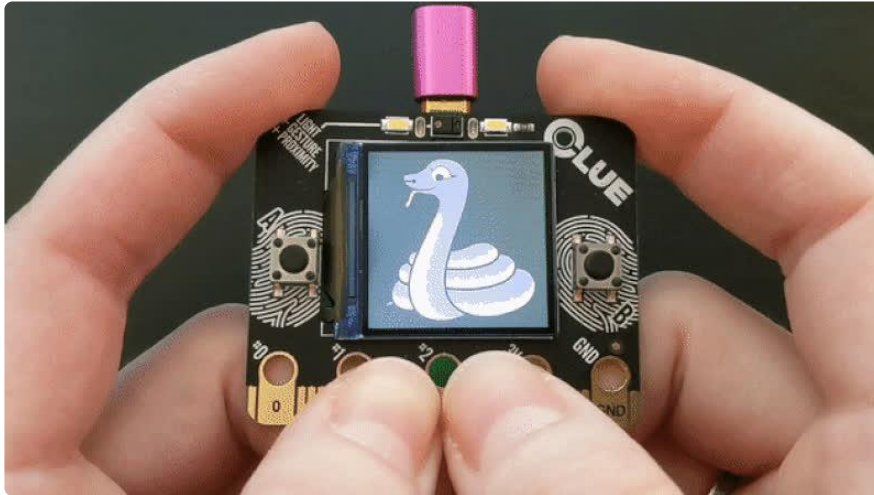
```
while True:
    if clue.button_b:
        slideshow.direction = PlayBackDirection.FORWARD
        slideshow.advance()
    if clue.button_a:
        slideshow.direction = PlayBackDirection.BACKWARD
        slideshow.advance()
```

Once the code and bitmaps are loaded, try pressing button B to move forward through displaying the images, and button A to move backward through displaying the images!

Let's take a look at the code.

First we import the CLUE library and the parts of the Slideshow library we intend to use: `SlideShow` and `PlayBackDirection`.

Then we create the slideshow. To create the slideshow, you must provide it the display object (`clue.display`). For this example, we've also set `auto_advance=False`. We don't want it to auto advance because we'll be using the buttons to advance through the images.

Inside the loop, we check for when each button is pressed. When button B is pressed, we set the playback direction to FORWARD and advance one image. When button A is pressed, we set the playback direction to BACKWARD, and advance one image.

That's all there is to creating a slideshow on your Adafruit CLUE using CircuitPython!

Using nRF52840 SPI on Battery Power

Summary: Don't use SPIM3 if you are not powering the board via USB.

The nRF52840 has three low-speed SPI peripherals, SPIM0, SPIM1, and SPIM2, and one high-speed SPI peripheral, SPIM3. SPIM0-2 have a maximum clock rate of 8 MHz, and SPIM3 has a maximum clock rate of 32 MHz. (SPIM0-2 are shared with the I2C peripherals, so not all SPI peripherals are always available.)

There appears to be a hardware bug with SPIM3: it does not operate properly when the board is not powered via the USB port. More precisely, it does not work when the VUSB pin to the nRF52840 is not energized.

In Arduino, avoid using SPIM3 if you are not powering the board via USB.

CircuitPython allocates which SPIM to use automatically. The first time you call `busio.SPI(),` or use `board.SPI()`, it will allocate SPIM3, so that you get the highest SPI speed possible.

To avoid getting the non-functional SPIM3 when not powering via USB, allocate the first SPI object, which will use SPIM3, and then just don't use it. Allocate a second one for your use, which will pick one of the other ones. For example:

```
import busio, board
# Allocate and then don't use the first SPI object.
# Choose pins you are not otherwise using. The pins below are just examples.
# You only need to specify a clock and a MOSI pin.

# Will allocate but not use SPIM3. SPIM3 doesn't work when not powered by USB.
do_not_use_this_spi = busio.SPI(clock=board.A0, MOSI=board.A1)

# Will allocate one of the working SPIM peripherals.
spi = board.SPI()

# ...
```
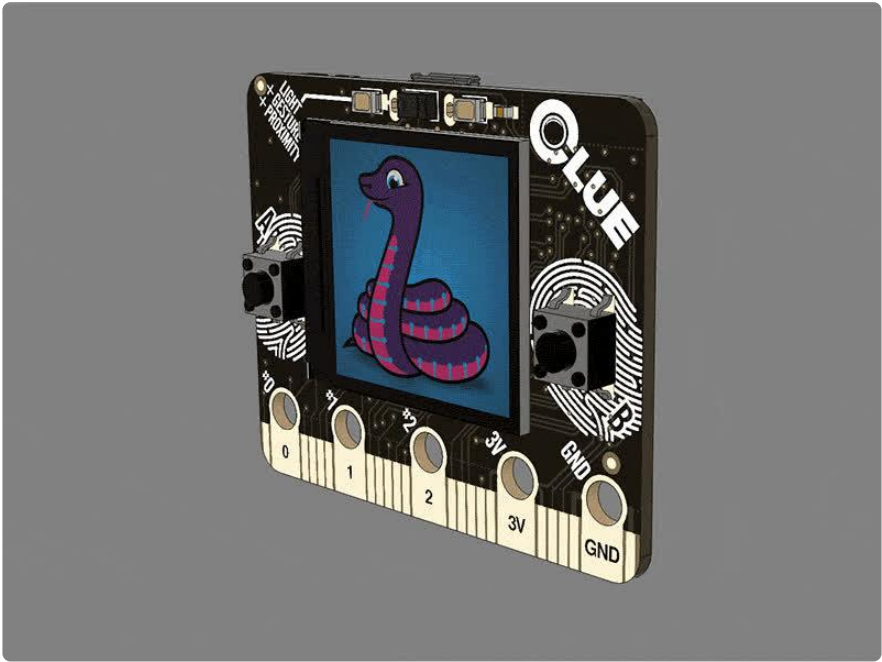
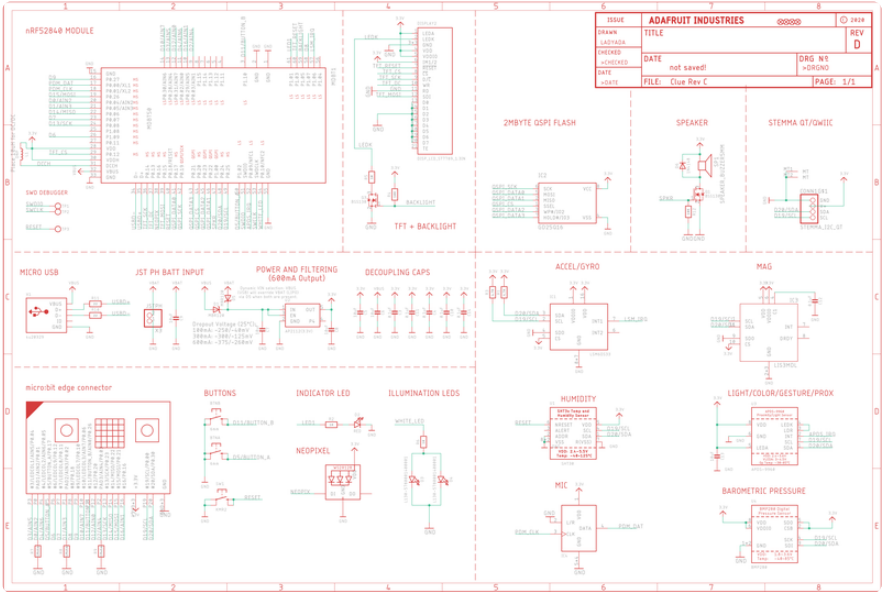This bug is being tracked in this CircuitPython issue ().

Downloads

Files:

- General nRF52840 Product Specification ()
- CLUE nRF52840 module: Raytac MDBT50Q details ()
- PDM mic: MP34DT01-M datasheet ()
- Humidity sensor: SHT3x-DIS dataheet ()
- Temperature and Barometric Pressure sensor: BMP280 datasheet ()
- Proximity, Light, Gesture, Color sensor: APDS9960 datasheet ()
- Gyroscope and Accelerometer: LSM6DS33 Datasheet ()
- Magnetometer: LIS3MDL Datasheet ()

- Fritzing object in the Adafruit Fritzing Library ()
- EagleCAD files on GitHub ()
- 3D Models on GitHub ()
- PDF for CLUE PrettyPins Pinout Diagram ()

SVG for CLUE PrettyPins Pinout Diagram

Schematic

Fab Print