Adafruit Metro M4 Express AirLift (WiFi)

Created by Brent Rubell

https://learn.adafruit.com/adafruit-metro-m4-express-airlift-wifi
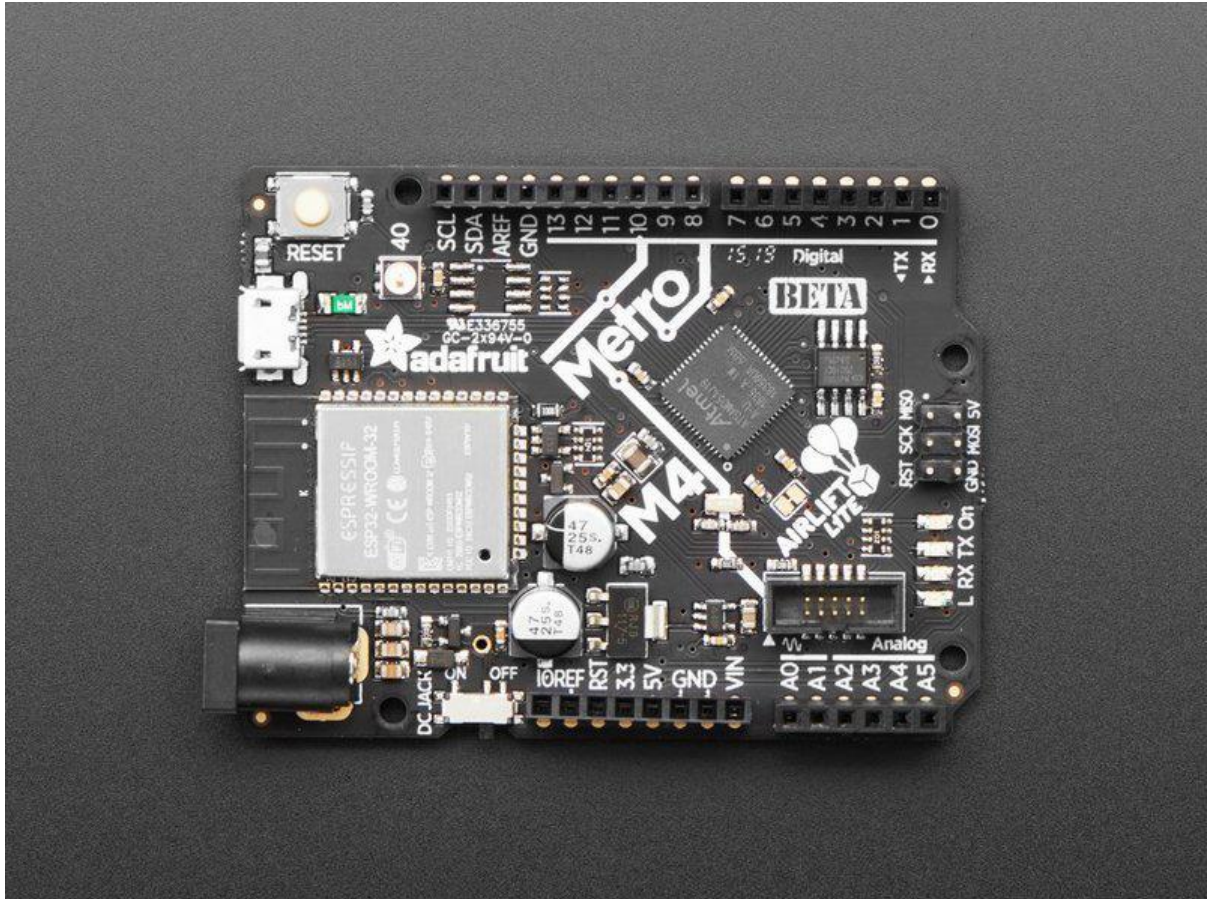
Last updated on 2022-12-12 04:37:41 PM EST

Table of Contents

Overview

Give your next project a lift with AirLift - our witty name for the ESP32 co-processor that graces this Metro M4. You already know about the Adafruit Metro M4 featuring the Microchip ATSAMD51, with it's 120MHz Cortex M4 with floating point support. With a train-load of FLASH and RAM, your code will be fast and roomy. And what better way to improve it than to add wireless? Now cooked in directly on board, you get a certified WiFi module that can handle all your TLS and socket needs, it even has root certificates pre-loaded.
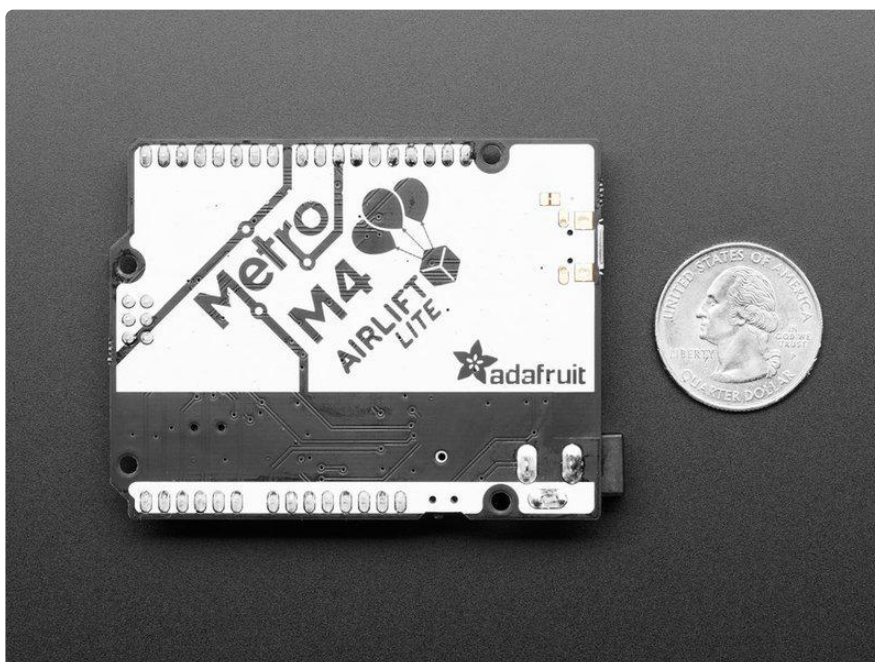
This Metro is the same size as the others, and is compatible with all our shields. It's got analog pins where you expect, and SPI/UART/I2C hardware support in the same spot as the Metro 328 and M0. But! It's powered with an ATSAMD51J19:

- Cortex M4 core running at 120 MHz
- [Floating point support with Cortex M4 DSP instructions ()](#)
- 512 KB flash, 192 KB RAM
- 32-bit, 3.3V logic and power
- Dual 1 MSPS DAC (A0 and A1)
- Dual 1 MSPS ADC (8 analog pins)
- 6 x hardware SERCOM (I2C, SPI or UART)
- 22 x PWM outputs
- Stereo I2S input/output with MCK pin
- 10-bit Parallel capture controller (for camera/video in)
- Built in crypto engines with AES (256 bit), true RNG, Pubkey controller
- 64 QFN
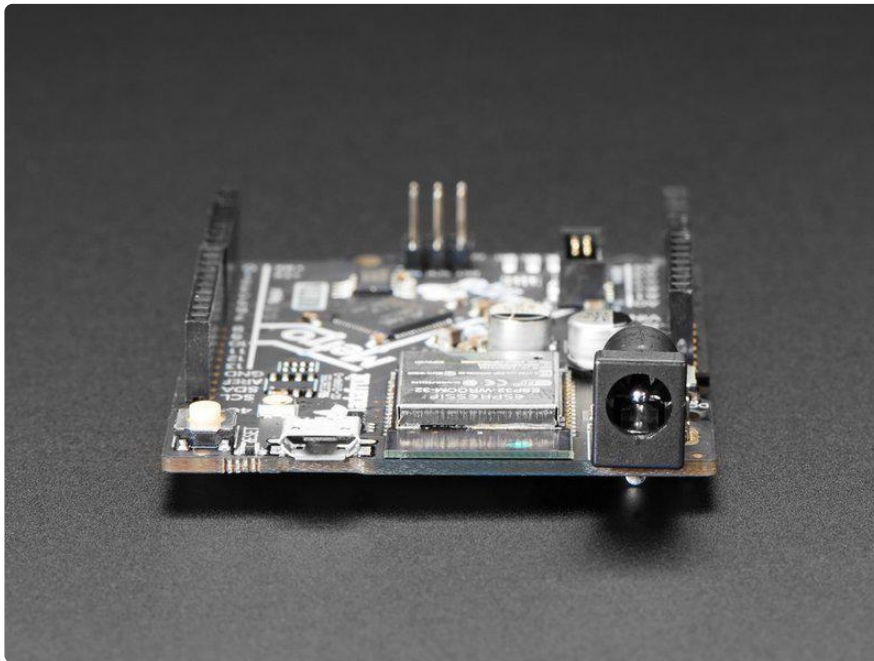
Pretty good start right? So we put this chip on a PCB with all these nice extras:

- Power the METRO M4 with 7-9V polarity protected DC or the micro USB connector to any 5V USB source. The 2.1mm DC jack has an on/off switch next to it so you can turn off your setup easily. The METRO will automagically switch between USB and DC.
- METRO has 25 GPIO pins, 8 of which are analog in, and two of which is a true analog out. There's a hardware SPI port, hardware I2C port and hardware UART. Logic level is 3.3V

- Native USB, there's no need for a hardware USB to Serial converter as the Metro M4 has built in USB support. When used to act like a serial device, the USB interface can be used by any computer to listen/send data to the METRO, and can also be used to launch and update code via the bootloader. It can also act like an HID keyboard or mouse.
- AirLift WiFi Co-processor, for WiFi and BLE support. WiFi includes TLS/SSL support, plenty of RAM for sockets, communication for Wifi is over SPI and has Arduino and CircuitPython libraries ready to go for fast wireless integration.
- Four indicator LEDs and one NeoPixel, on the back edge of the PCB, for easy debugging. One green power LED, two RX/TX LEDs for data being sent over USB, and a red LED connected. Next to the reset button there is an RGB NeoPixel that can be used for any purpose.
- 2 MB QSPI Flash storage chip is included on board. You can use the SPI Flash storage like a very tiny hard drive. When used in Circuit Python, the 2 MB flash acts as storage for all your scripts, libraries and files. When used in Arduino, you can read/write files to it, like a little datalogger or SD card, and then with our helper program, access the files over USB.
- Easy reprogramming, comes pre-loaded with the UF2 bootloader (), which looks like a USB storage key. Simply drag firmware on to program, no special tools or drivers needed! It can be used to load up CircuitPython or Arduino IDE (it is bossa v1.8 compatible)
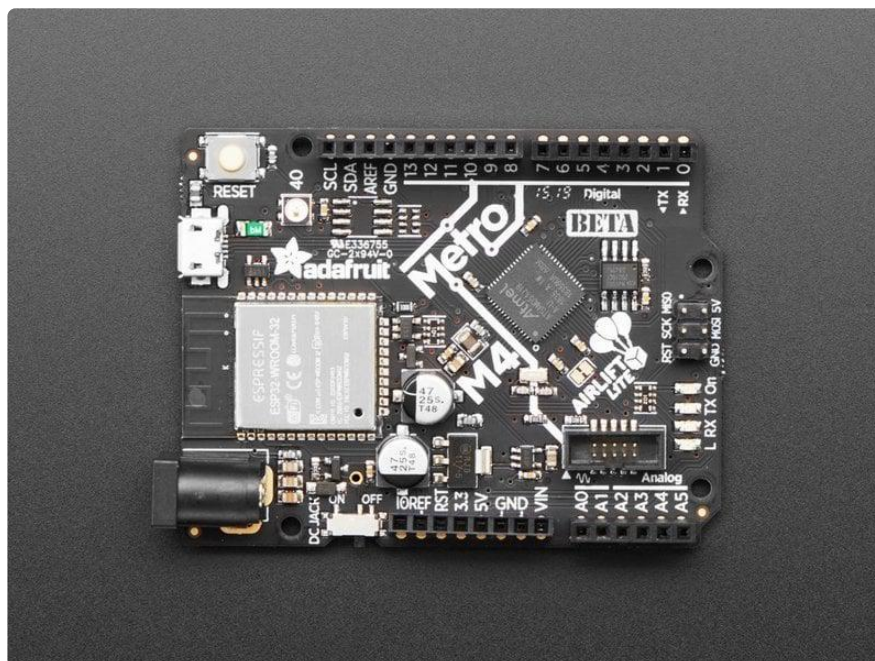
We have a working Arduino board support package, with lots of stuff working, but our primary target for this board is CircuitPython - with 120 MHz, and 192KB of RAM CircuitPython runs really well on this chip! CircuitPython's built in support for JSON

parsing plus our requests compatible library makes it incredibly easy to build secure IoT projects in just a few minutes.

So what are you waiting for? Pick up a Metro M4 AirLift today and be amazed at how easy and fast it is to get started with IoT projects using CircuitPython!

Update the UF2 Bootloader

Update the Bootloader on your SAMD51 M4 board to prevent a somewhat rare problem of parts of internal flash being overwritten on power-up.

Your SAMD51 M4 board bootloader may need to be updated to fix an intermittent bug that can erase parts of internal flash.

Updating Your Bootloader

To see if you need to update your bootloader, get the UF2 boot drive to appear as a mounted drive on your computer, in a file browser window. If you're running MakeCode, click the reset button once. If you're running CircuitPython or an Arduino program, double-click the reset button.

When you see the ...BOOT drive (FEATHERBOOT, METROM4BOOT, ITSYM4BOOT, PORTALBOOT, etc.) , click the drive in the file browser window and then double-click the INFO_UF2.TXT file to see what's inside.

The example screenshots below are for a PyGamer. What you see for your board will be largely the same except for the board name and the BOOT drive name.
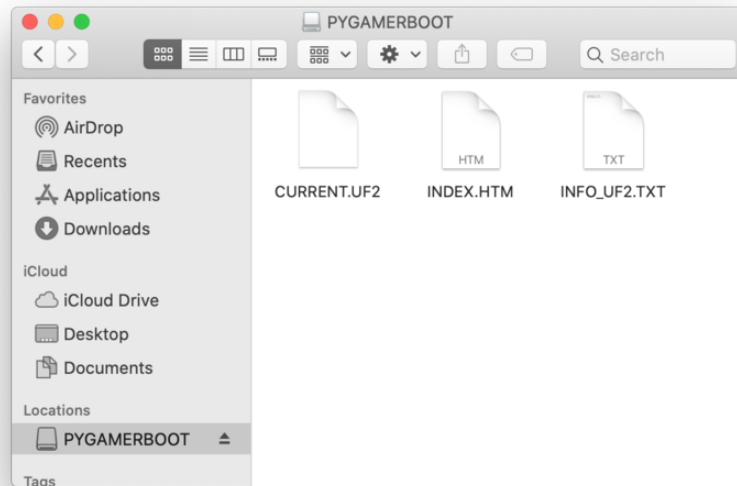
The bootloader version is listed in INFO_UF2.TXT. In this example, the version is v3.6. 0.
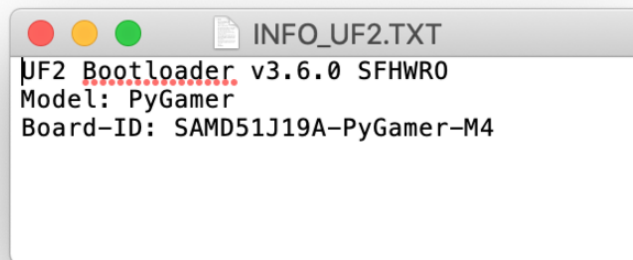
If the bootloader version you see is older than v3.9.0, you need to update. For instance, the bootloader above needs to be upgraded.

Download the latest version of the bootloader updater from the circuitpython.org Downloads page for your board.

Latest board downloads

- The bootloader updater will be named update-bootloader-name_of_your_board-v3.9.0.uf2 or some later version. Drag that file from your Downloads folder onto the BOOT drive:
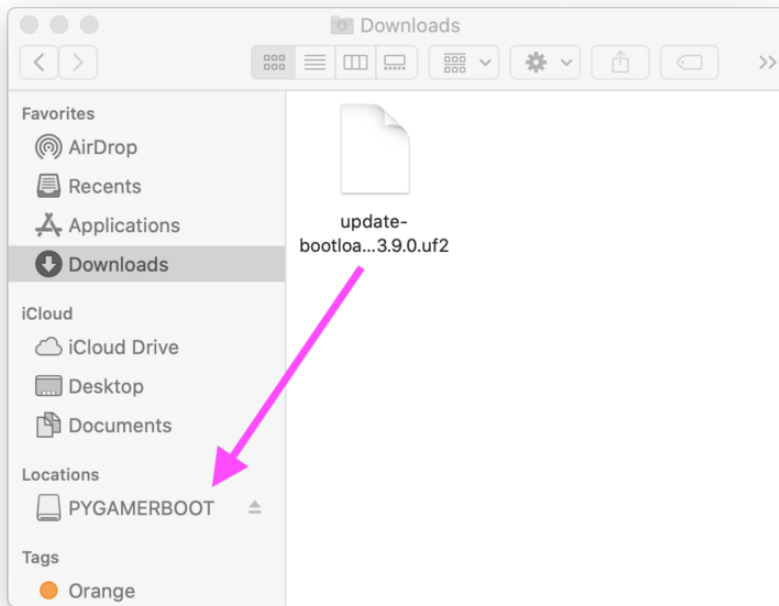
After you drag the updater onto the boot drive, the red LED on the board will flicker and then blink slowly about five times. A few seconds later, the BOOT will appear in the Finder. After that, you can click on the BOOT drive and double-click INFO_UF2.TXT again to confirm you've updated the bootloader.

UF2 Bootloader v3.9.0 SFHWRO
Model: PyGamer
Board-ID: SAMD51J19A-PyGamer-M4
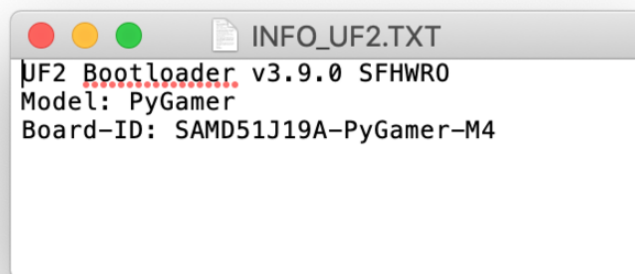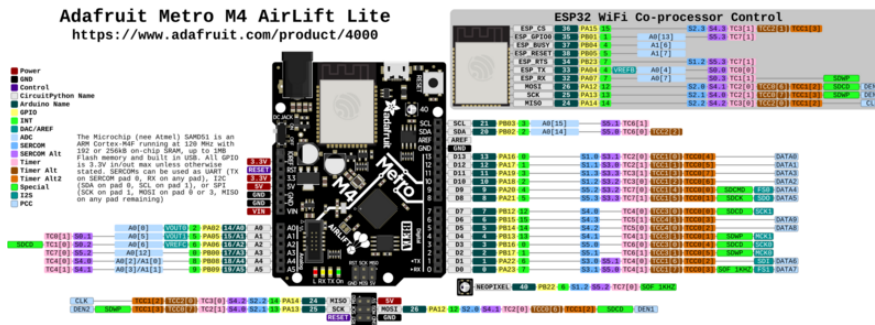
Pinouts

[Click here to view a PDF version of the pinout diagram](#) ()
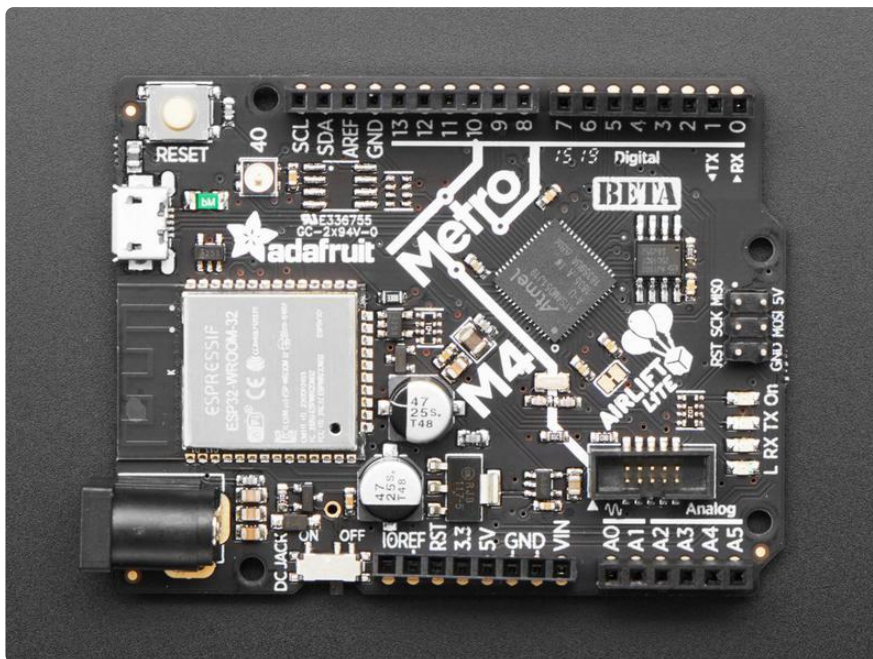
The Metro M4 Express AirLift is chock-full of microcontroller goodness. There's also a lot of pins and ports. We'll take you a tour of them now!

Power Connections
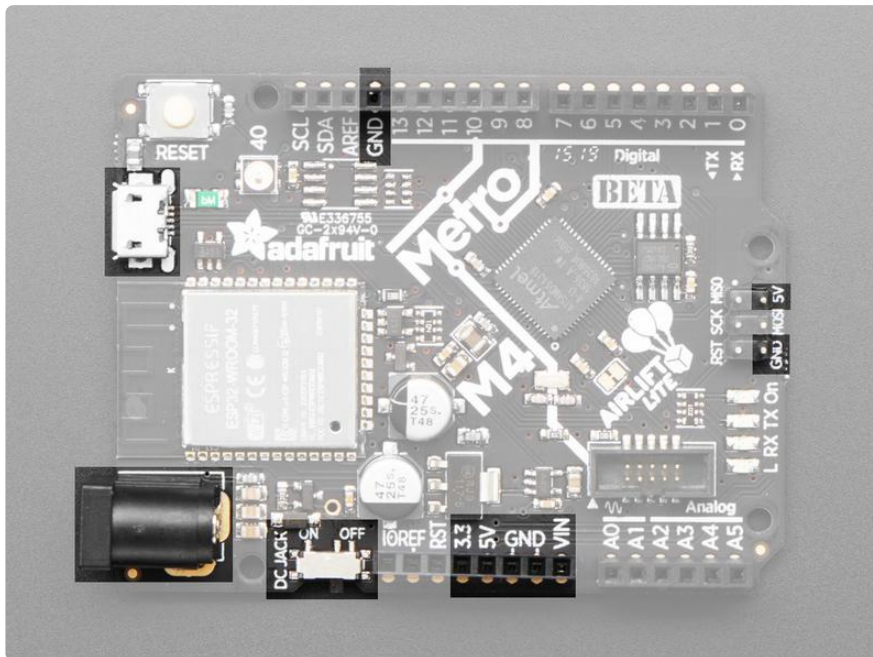
There's a lot of ways to power the Metro M4 Express AirLift, and a lot of ways to get power out as well.

There are two primary ways to power the Metro:

- Through the Micro USB port up at the top left
- Through the DC jack at the bottom left

The MicroUSB jack provides 5V at 500mA or so, there is a fuse that will shut off temporarily when more than 1000mA is drawn, this is to protect a computer USB port. You can plug this into any computer or USB charger with a USB cable. You can draw up to 500mA between the Vin, 5V and 3.3V supplies (combined).

The DC Jack is a 5.5mm/2.1mm center-positive DC connector, which is the most common available. Provide about 6V-12V here to power the Metro. There is no fuse on this connection so you can draw more current, up to 800mA between the 5V and 3.3V supplies, and 2A from Vin.

Onboard regulators take the USB or DC power and linearly convert it to 3.3V and 5V:

- 3V - this is the output from the 3.3V regulator, it can supply 500mA peak
- 5V - this is the output from the 5V regulator (when DC jack is used), or from USB. It can supply ~500mA peak from USB and ~800mA peak from DC

- GND - this is the common ground for all power and logic
- Vin - this is the higher of the DC jack or USB voltage. So if the DC jack is plugged in and 9V, Vin is 9V. If only USB connected, this will be 5V.

There is also an on/off switch. This switch is only for the DC jack and does not affect powering via USB

ESP32 WiFi Co-Processor
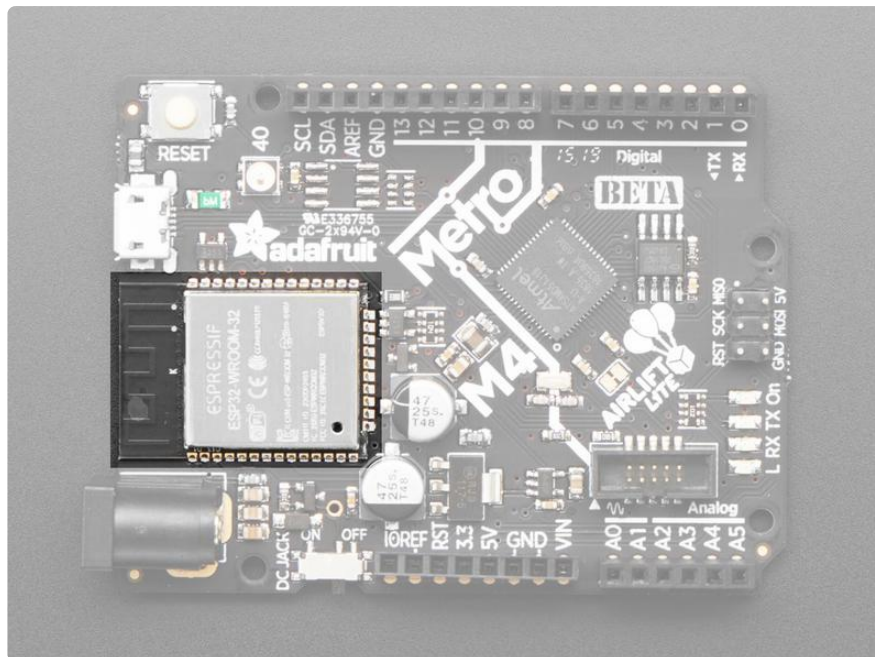
The WiFi capability uses an Espressif ESP32 Wi-Fi coprocessor with TLS/SSL support built-in.

The ESP32 uses the main SPI port for data, and also uses a CS pin (`board.ESP_CS` or Arduino `SPIWIFI_SS` or `36`), Ready/Busy pin (`board.ESP_BUSY` or Arduino `SPIWIFI_ACK` or `37`), and reset pin (`board.ESP_RESET` or Arduino `SPIWIFI_RESET` or `38`)

- For advanced use or reprogramming, we also connect the main RX/TX UART to the ESP32 - on Arduino that's `Serial2` and in CircuitPython use `board.ESP_RX` and `board.ESP_TX` . This is a unique/unshared Serial port that is not the same as the RX/TX pins on the Metro breakout pins.
- You can also connect to the ESP32 RTS pin (used in some serial contexts) on `board.ESP_RTS` or Arduino `34` .
- The ESP32 GPIO0 pin for bootloader enable is connected to `board.ESP_GPIO0` or Arduino `ESP32_GPIO0` or `35`

Logic pins
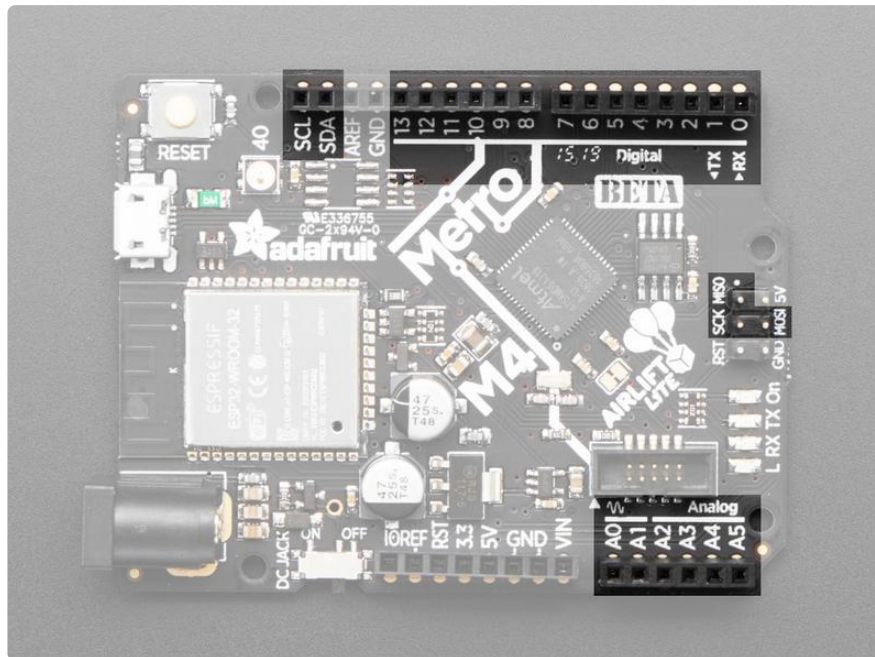
This is the general purpose I/O pin set for the microcontroller.

All logic is 3.3V

All of the pins except SCL can do PWM output (A0 has true analog output instead)

All pins can be interrupt inputs

Top Row

- #0 / RX - GPIO #0, also receive (input) pin for Serial1 (hardware UART)
- #1 / TX - GPIO #1, also transmit (output) pin for Serial1
- #2 through #12 - These are general purpose GPIO
- #13 - GPIO #13 and is connected to the red LED marked L next to the USB jack.
- SDA - the I2C (Wire) data pin. There's a 10K pull up on this pin to 3V already installed
- SCL - the I2C (Wire) clock pin. There's a 10K pull up on this pin to 3V already installed

Bottom Row

- A0 - This pin is analog input A0 but is also an analog output due to having a DAC (digital-to-analog converter). This is the first DAC, and is 'independent' of A1. You can set the raw voltage to anything from 0 to 3.3V, unlike PWM outputs this is a true analog output

- A1 - This pin is analog input A1 but is also an analog output due to having a DAC (digital-to-analog converter). This is the second DAC, and is 'independent' of A0. You can set the raw voltage to anything from 0 to 3.3V, unlike PWM outputs this is a true analog output
- A2 thru A5 - These are each analog input as well as digital I/O pins.

Please note, A3 functions as expected but happens to be a different microcontroller pin than the one on the Metro M4 non-WiFi version. This shouldn't affect you but if you're doing something special with that pin, like a timer or SERCOM config, check the schematic for the underlying hardware pin name

Right side

- SCK/MOSI/MISO - These are the hardware SPI pins, are are connected to the 2x3 header on the right hand side. These are also used by the ESP32 so they should not be used for anything but SPI connectivity - not for GPIO.

Additional analog inputs

In addition to the A0-A5 pins, there are extra analog inputs available. There's not a ton because we ended up going with prioritizing the PCC camera peripheral pins rather than try to stuff as many analog inputs as possible

- SDA is also A6
- SCL is also A7

These pins are available in CircuitPython under the `board` module. Names that start with # are prefixed with D and other names are as is. So #0 / RX above is available as `board.D0` and `board.RX` for example.

I2S Peripheral

For quality audio input and output, there is an I2S peripheral available on select pins. Those pins cannot be changed! These use I2S device 0 and clock generator 3

- Pin 8 - SDO (data out)
- Pin 1 - SDI (data in)
- Pin 3 - Bit Clock

- Pin 9 - Word Select
- Pin 2 - Master Clock (not used by our libraries but it is available)

Parallel Capture Peripheral

There's a 'camera' input peripheral you can use with some camera chips to capture video with 10-bit data width. We thought this was neat so we made sure all those pins were available. Here are the PCC pins (left) and the Metro M4 Express AirLift pins it's mapped to. Unlike other peripherals, you cannot mux these signals to other pins!

- DEN1: MOSI
- DEN2: SCK
- CLK: MISO
- D0: D13
- D1: D12
- D2: D10
- D3: D11
- D4: D9
- D5: D8
- D6: D1
- D7: D0
- D8: D5
- D9: D6

QSPI Flash and NeoPixel

As part of the 'Express' series of boards, the Metro M4 Express AirLift is designed for use with CircuitPython. To make that easy, we have added two extra parts to this Metro AirLift: a mini NeoPixel (RGB LED) and a 2 MB QSPI (Quad SPI) Flash chip
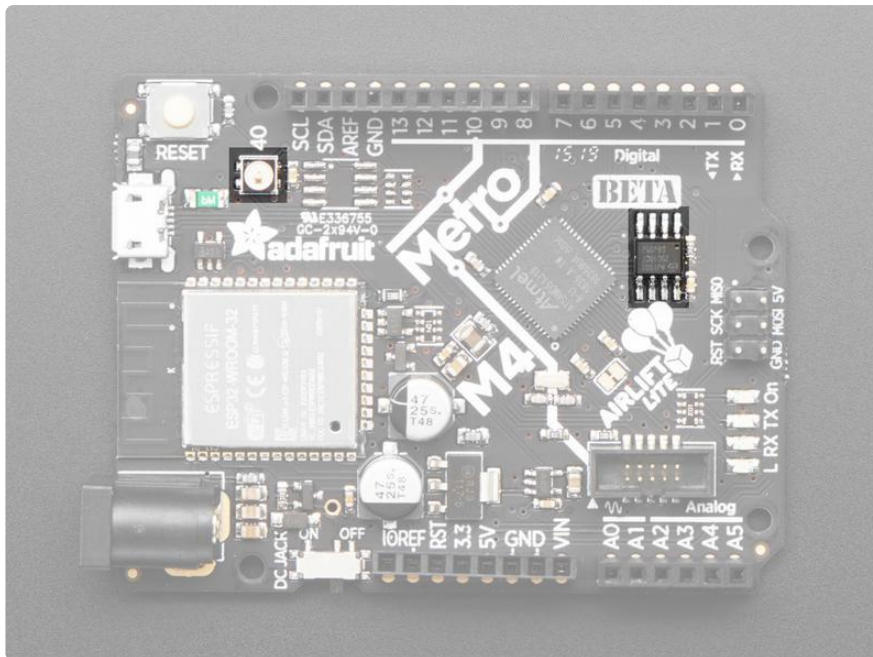
The NeoPixel is connected to pin #40 in Arduino, so just use our NeoPixel library () and set it up as a single-LED strand on pin 40. In CircuitPython, the NeoPixel is `board .NEOPIXEL` and the library for it is here () and in the bundle (). The NeoPixel is powered by the 3.3V power supply but that hasn't shown to make a big difference in brightness or color. The NeoPixel is also used by the bootloader to let you know if the device has enumerated correctly (green) or USB failure (red). In CircuitPython, the LED is used to indicate the runtime status.

The QSPI Flash is connected to 6 pins that are not brought out on the GPIO pads. This way you don't have to worry about the SPI flash colliding with other devices on the main SPI connection.

QSPI is neat because it allows you to have 4 data in/out lines instead of just SPI's single line in and single line out. This means that QSPI is at least 4 times faster. But in reality is at least 10x faster because you can clock the QSPI peripheral much faster than a plain SPI peripheral

However, the QSPI port is not also on an SERCOM. So, you have to either use the QSPI peripheral or bitbang SPI if you want to talk to the chip. We have an Arduino library here which provides QSPI interfacing for Arduino (). In CircuitPython, the QSPI flash is used natively by the interpretter and is read-only to user code, instead the Flash just shows up as the writeable disk drive!

Other Pins!

RST - this is the Reset pin, tie to ground to manually reset the ATSAMD51, as well as launch the bootloader manually

ARef - the analog reference pin. Normally the reference voltage is the same as the chip logic voltage (3.3V) but if you need an alternative analog reference, connect it to this pin and select the external AREF in your firmware. Can't go higher than 3.3V!

On the Metro M4 Express AirLift, at least for now, AREF is tied to 3.3V due to a silicon v0 bug that does not allow the DACs to work unless AREF is connected to 3.3V. You cut the top jumper next to the balloons logo if you need a different AREF voltage but note that this may change DAC range!

Debug Interface

If you'd like to do more advanced development, trace-debugging, or not use the bootloader, we have the SWD interface exposed.

You can use any 2x5 0.05" pitch SWD interface to connect. We suggest a J-Link, its supported by every IDE/software that can program the ATSAMD51.

SEGGER J-Link EDU - JTAG/SWD Debugger
The SEGGER J-Link EDU is identical to the more expensive J-Link BASE model except for the terms of use.If...
https://www.adafruit.com/product/1369

SEGGER J-Link BASE - JTAG/SWD Debugger
The SEGGER J-Link BASE is identical to the cheaper J-Link EDU model except for the terms of...
https://www.adafruit.com/product/2209
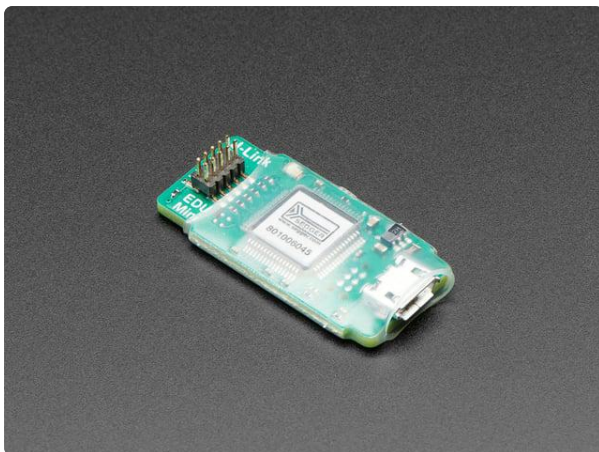
SEGGER J-Link EDU Mini - JTAG/SWD Debugger
Doing some serious development on any ARM-based platform, and tired of 'printf' plus an LED to debug? A proper JTAG/SWD HW debugger can make debugging more of a pleasure and...
https://www.adafruit.com/product/3571

You'll need an adapter and cable to convert the 2x10 JTAG cable to SWD

10-pin 2x5 Socket-Socket 1.27mm IDC (SWD) Cable - 150mm long
These little cables are handy when programming or debugging a tiny board that uses 10-pin 1.27mm (0.05") pitch SWD programming connectors. We see these connectors often on ARM...
https://www.adafruit.com/product/1675
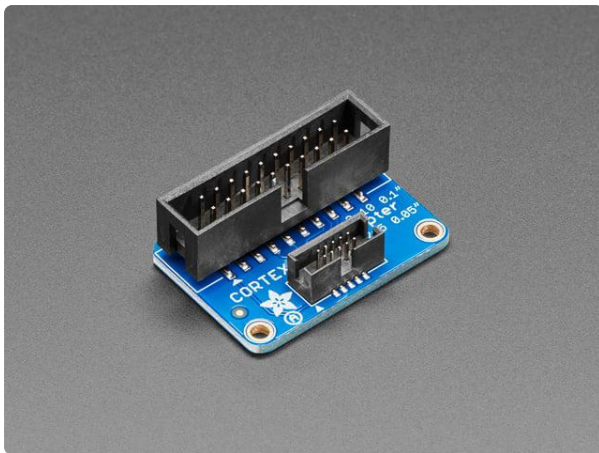
JTAG (2x10 2.54mm) to SWD (2x5 1.27mm) Cable Adapter Board
This adapter board is designed for adapting a 'classic' 2x10 (0.1"/2.54mm pitch) JTAG cable to a slimmer 2x5 (0.05"/1.27mm pitch) SWD Cable. It's helpful...
https://www.adafruit.com/product/2094

If you want to use the J-Link with Segger Ozone, check out this handy informative post on the forums ()

UF2 Bootloader Details

> This is an information page for advanced users who are curious how we get code from your computer into your Express board!

Adafruit SAMD21 (M0) and SAMD51 (M4) boards feature an improved bootloader that makes it easier than ever to flash different code onto the microcontroller. This bootloader makes it easy to switch between Microsoft MakeCode, CircuitPython and Arduino.

Instead of needing drivers or a separate program for flashing (say, `bossac`, `jlink` or `avrdude`), one can simply drag a file onto a removable drive.

The format of the file is a little special. Due to 'operating system woes' you cannot just drag a binary or hex file (trust us, we tried it, it isn't cross-platform compatible). Instead, the format of the file has extra information to help the bootloader know

where the data goes. The format is called UF2 (USB Flashing Format). Microsoft MakeCode generates UF2s for flashing and CircuitPython releases are also available as UF2. You can also create your own UF2s from binary files using uf2tool, available here. ()

The bootloader is also BOSSA compatible, so it can be used with the Arduino IDE which expects a BOSSA bootloader on ATSAMD-based boards

For more information about UF2, you can read a bunch more at the MakeCode blog (), then check out the UF2 file format specification. ()

Visit Adafruit's fork of the Microsoft UF2-samd bootloader GitHub repository () for source code and releases of pre-built bootloaders on CircuitPython.org ().

> The bootloader is not needed when changing your CircuitPython code. Its only needed when upgrading the CircuitPython core or changing between CircuitPython, Arduino and Microsoft MakeCode.

The CircuitPython Boot Sequence
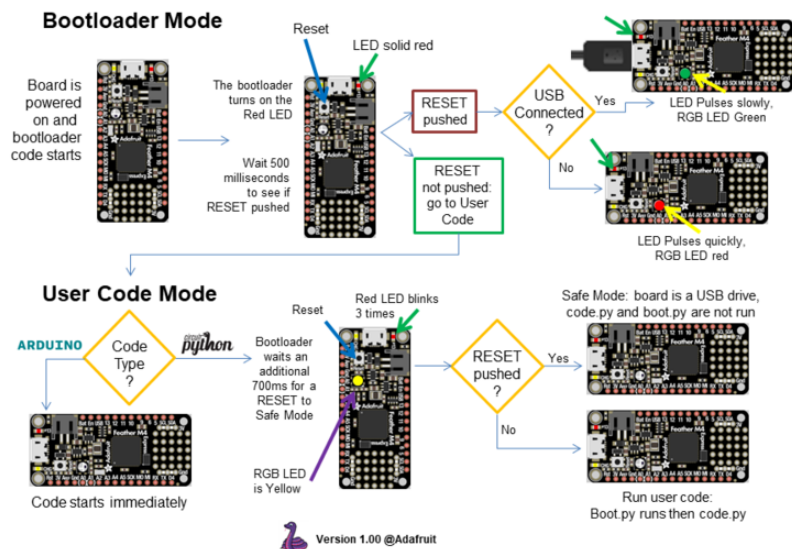
Entering Bootloader Mode

The first step to loading new code onto your board is triggering the bootloader. It is easily done by double tapping the reset button. Once the bootloader is active you will see the small red LED fade in and out and a new drive will appear on your computer with a name ending in BOOT. For example, feathers show up as FEATHERBOOT,

while the new CircuitPlayground shows up as CPLAYBOOT, Trinket M0 will show up as TRINKETBOOT, and Gemma M0 will show up as GEMMABOOT

Furthermore, when the bootloader is active, it will change the color of one or more onboard neopixels to indicate the connection status, red for disconnected and green for connected. If the board is plugged in but still showing that its disconnected, try a different USB cable. Some cables only provide power with no communication.

For example, here is a Feather M0 Express running a colorful Neopixel swirl. When the reset button is double clicked (about half second between each click) the NeoPixel will stay green to let you know the bootloader is active. When the reset button is clicked once, the 'user program' (NeoPixel color swirl) restarts.
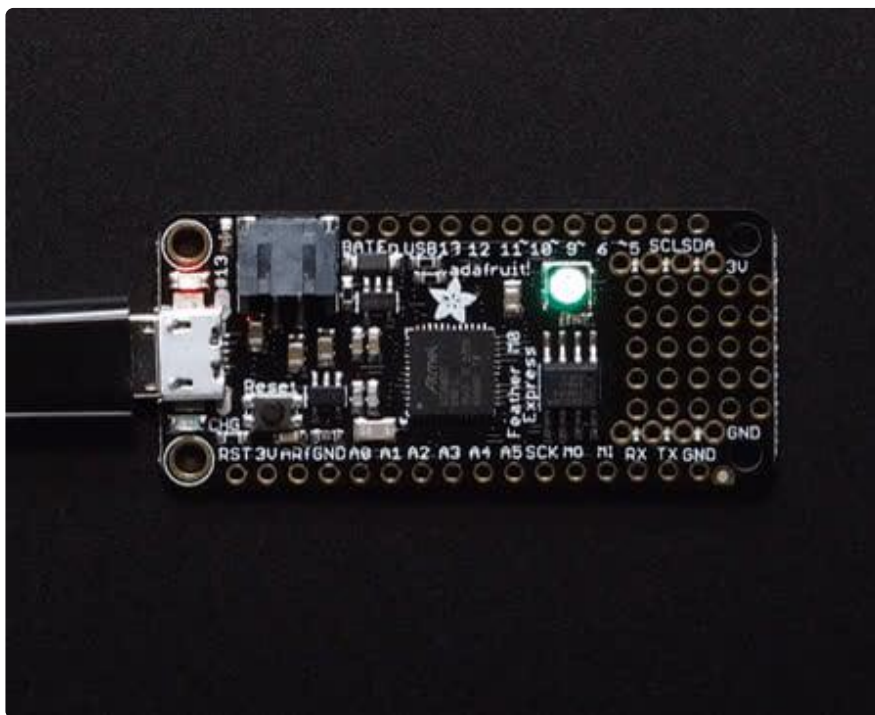
If the bootloader couldn't start, you will get a red NeoPixel LED.

That could mean that your USB cable is no good, it isn't connected to a computer, or maybe the drivers could not enumerate. Try a new USB cable first. Then try another port on your computer!
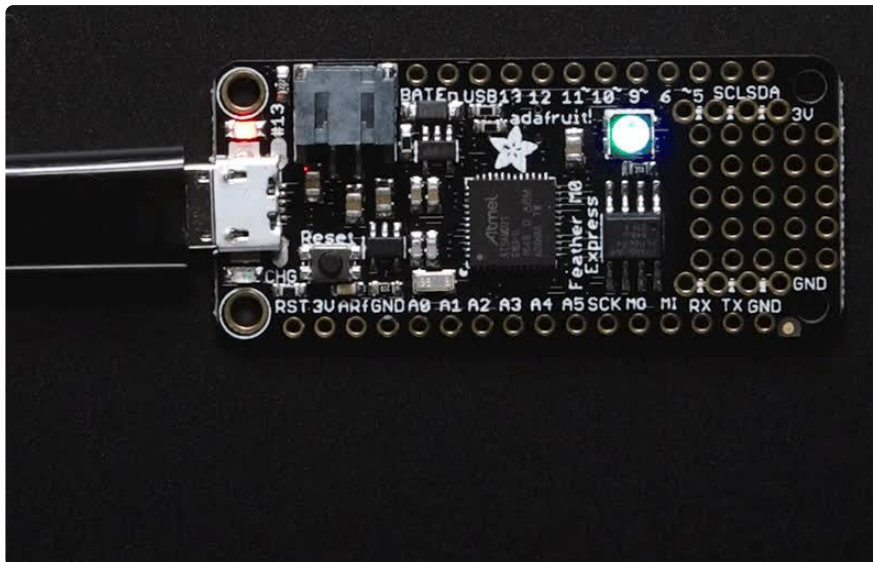
Once the bootloader is running, check your computer. You should see a USB Disk drive...
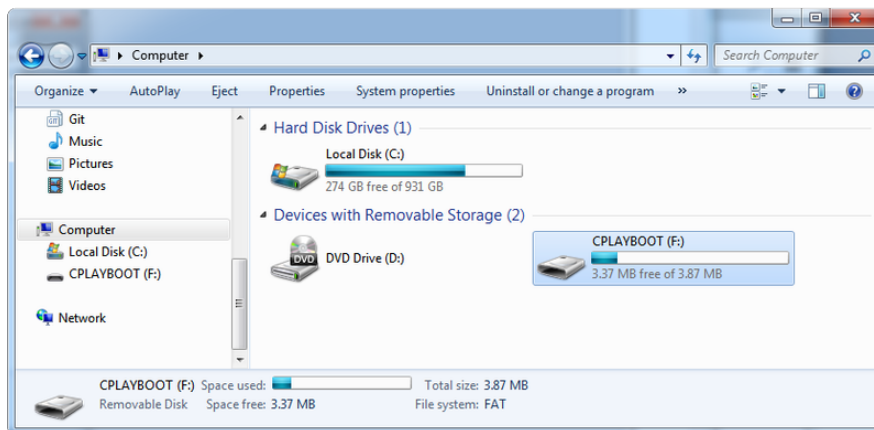
Once the bootloader is successfully connected you can open the drive and browse the virtual filesystem. This isn't the same filesystem as you use with CircuitPython or Arduino. It should have three files:

- CURRENT.UF2 - The current contents of the microcontroller flash.
- INDEX.HTM - Links to Microsoft MakeCode.
- INFO_UF2.TXT - Includes bootloader version info. Please include it on bug reports.
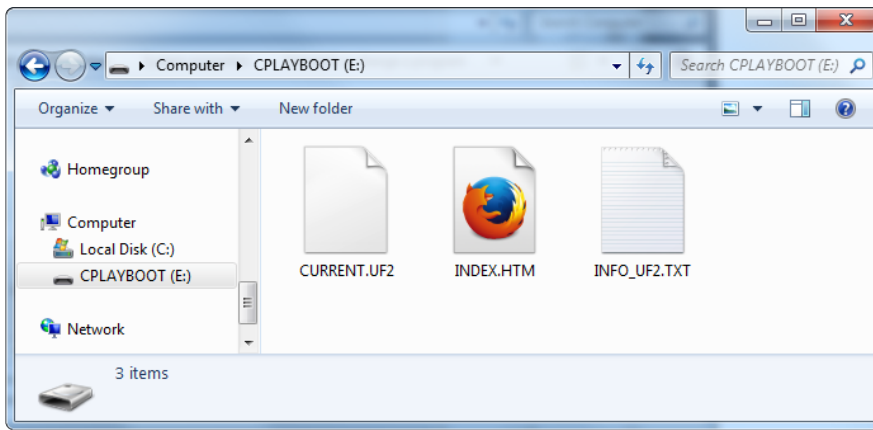
Using the Mass Storage Bootloader

To flash something new, simply drag any UF2 onto the drive. After the file is finished copying, the bootloader will automatically restart. This usually causes a warning about an unsafe eject of the drive. However, its not a problem. The bootloader knows when everything is copied successfully.
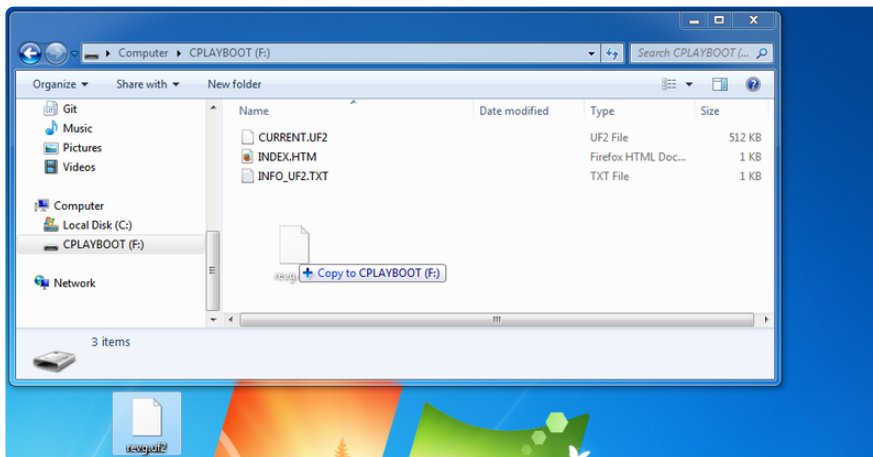
You may get an alert from the OS that the file is being copied without it's properties. You can just click Yes
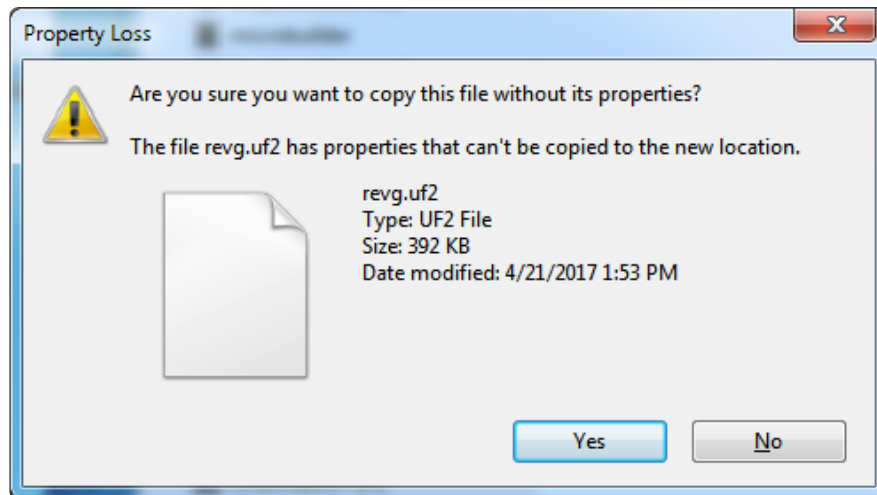
You may also get get a complaint that the drive was ejected without warning. Don't worry about this. The drive only ejects once the bootloader has verified and completed the process of writing the new code

Using the BOSSA Bootloader

As mentioned before, the bootloader is also compatible with BOSSA, which is the standard method of updating boards when in the Arduino IDE. It is a command-line tool that can be used in any operating system. We won't cover the full use of the boss ac tool, suffice to say it can do quite a bit! More information is available at ShumaTech ().

Windows 7 Drivers

If you are running Windows 7 (or, goodness, something earlier?) You will need a Serial Port driver file. Windows 10 users do not need this so skip this step.

You can download our full driver package here:

Download Latest Adafruit Driver Installer

Download and run the installer. We recommend just selecting all the serial port drivers available (no harm to do so) and installing them.
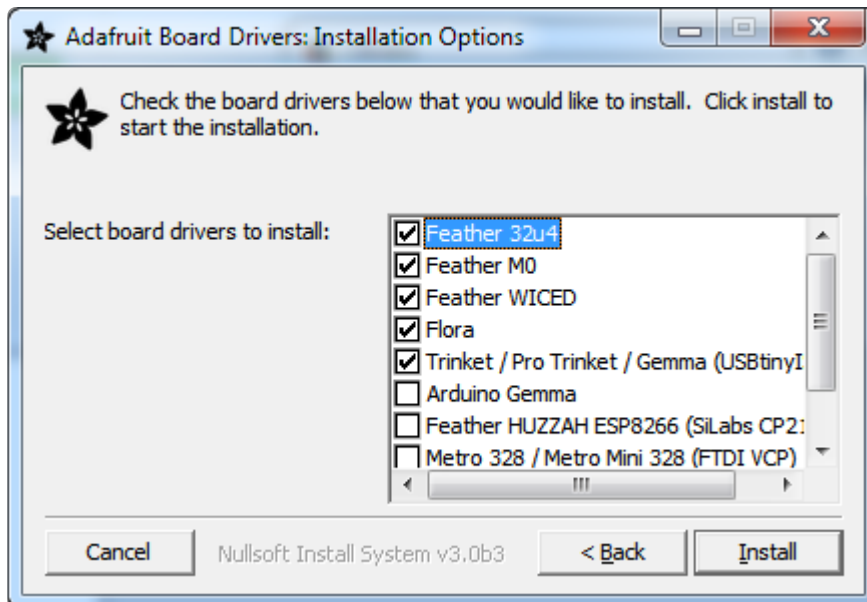
Verifying Serial Port in Device Manager

If you're running Windows, its a good idea to verify the device showed up. Open your Device Manager from the control panel and look under Ports (COM & LPT) for a device called Feather M0 or Circuit Playground or whatever!
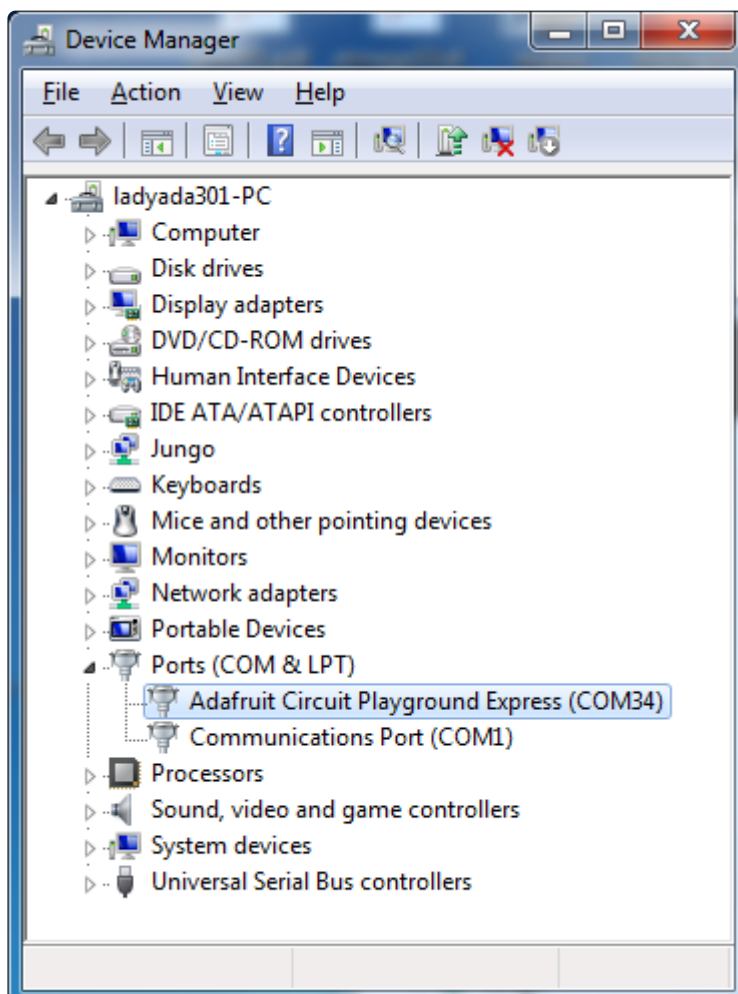
If you see something like this, it means you did not install the drivers. Go back and try again, then remove and re-plug the USB cable for your board
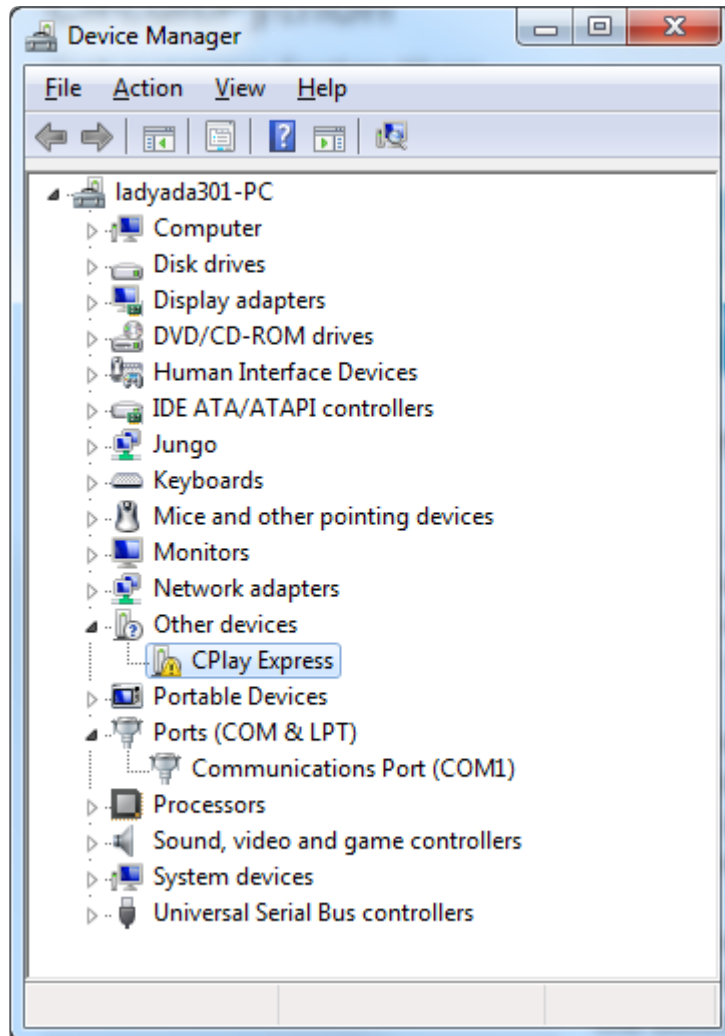
Running bossac on the command line

If you are using the Arduino IDE, this step is not required. But sometimes you want to read/write custom binary files, say for loading CircuitPython or your own code. We recommend using bossac v 1.7.0 (or greater), which has been tested. The Arduino branch is most recommended ().

You can download the latest builds here. () The `mingw32` version is for Windows, `apple-darwin` for Mac OSX and various `linux` options for Linux. Once downloaded, extract the files from the zip and open the command line to the directory with `bossac` .

> With bossac versions 1.9 or later, you must use the --offset parameter on the command line, and it must have the correct value for your board.

With bossac version 1.9 or later, you must give an `--offset` parameter on the command line to specify where to start writing the firmware in flash memory. This parameter was added in bossac 1.8.0 with a default of `0x2000`, but starting in 1.9, the default offset was changed to `0x0000`, which is not what you want in most cases. If you omit the argument for bossac 1.9 or later, you will probably see a "Verify Failed" error from bossac. Remember to change the option for `-p` or `--port` to match the port on your Mac.

Replace the filename below with the name of your downloaded `.bin`: it will vary based on your board!

Using bossac Versions 1.7.0, 1.8

There is no `--offset` parameter available. Use a command line like this:

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R adafruit-circuitpython-boardname-version.bin
```

For example,

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R adafruit-circuitpython-feather_m0_express-3.0.0.bin
```

Using bossac Versions 1.9 or Later

For M0 boards, which have an 8kB bootloader, you must specify `-offset=0x2000`, for example:

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R --offset=0x2000 adafruit-circuitpython-feather_m0_express-3.0.0.bin
```

For M4 boards, which have a 16kB bootloader, you must specify `-offset=0x4000`, for example:

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R --offset=0x4000 adafruit-circuitpython-feather_m4_express-3.0.0.bin
```

This will `e`rase the chip, `w`rite the given file, `v`erify the write and `R`eset the board. On Linux or MacOS you may need to run this command with `sudo ./bossac ...`, or add yourself to the dialout group first.

Updating the bootloader

The UF2 bootloader is relatively new and while we've done a ton of testing, it may contain bugs. Usually these bugs effect reliability rather than fully preventing the bootloader from working. If the bootloader is flaky then you can try updating the bootloader itself to potentially improve reliability.

If you're using MakeCode on a Mac, you need to make sure to upload the bootloader to avoid a serious problem with newer versions of MacOS. See instructions and more details here ().

In general, you shouldn't have to update the bootloader! If you do think you're having bootloader related issues, please post in the forums or discord.

Updating the bootloader is as easy as flashing CircuitPython, Arduino or MakeCode. Simply enter the bootloader as above and then drag the update bootloader uf2 file below. This uf2 contains a program which will unlock the bootloader section, update the bootloader, and re-lock it. It will overwrite your existing code such as CircuitPython or Arduino so make sure everything is backed up!

After the file is copied over, the bootloader will be updated and appear again. The INFO_UF2.TXT file should show the newer version number inside.

For example:

```
UF2 Bootloader v2.0.0-adafruit.5 SFHWRO
Model: Metro M0
Board-ID: SAMD21G18A-Metro-v0
```

Lastly, reload your code from Arduino or MakeCode or flash the latest CircuitPython core ().

Below are the latest updaters for various boards. The latest versions can always be found here (). Look for the `update-bootloader...` files, not the `bootloader...` files.

Circuit Playground Express V3.7.0 update-bootloader.uf2

Feather M0 Express v3.7.0 update-bootloader.uf2

Metro M0 Express v3.7.0 update-bootloader.uf2

Gemma M0 v3.7.0 update-bootloader.uf2

Trinket M0 v3.7.0 update-bootloader.uf2

Itsy Bitsy M0 v3.7.0 update-bootloader.uf2

Grand Central M4 v3.7.0 update-bootloader.uf2

Latest version of update-bootloader.uf2 for other boards. Make sure you pick the right one.

Getting Rid of Windows Pop-ups

If you do a lot of development on Windows with the UF2 bootloader, you may get annoyed by the constant "Hey you inserted a drive what do you want to do" pop-ups.

Go to the Control Panel. Click on the Hardware and Sound header
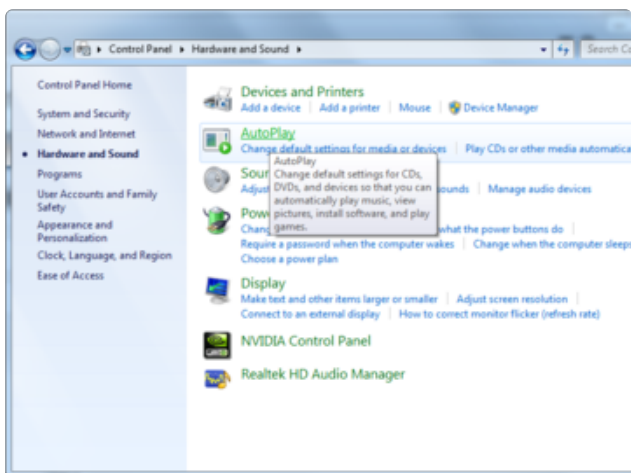
Click on the Autoplay header
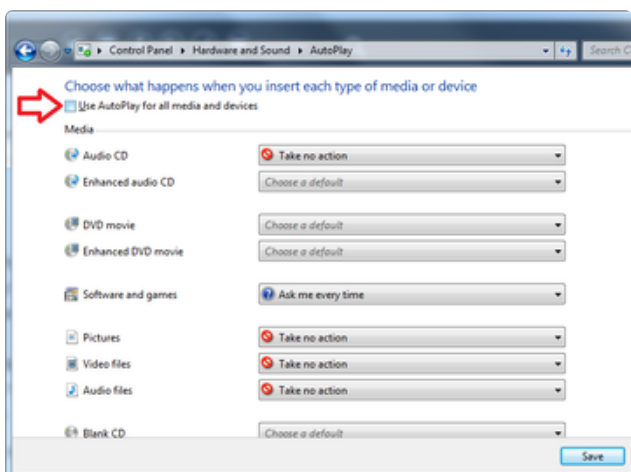
Uncheck the box at the top, labeled Use Autoplay for all devices

Making your own UF2

Making your own UF2 is easy! All you need is a .bin file of a program you wish to flash and the Python conversion script (). Make sure that your program was compiled to start at 0x2000 (8k) for M0 boards or 0x4000 (16kB) for M4 boards. The bootloader takes up the first 8kB (M0) or 16kB (M4). CircuitPython's linker script () is an example on how to do that.

Once you have a .bin file, you simply need to run the Python conversion script over it. Here is an example from the directory with uf2conv.py. This command will produce a firmware.uf2 file in the same directory as the source firmware.bin. The uf2 can then be flashed in the same way as above.

```
# For programs with 0x2000 offset (default)
uf2conv.py -c -o build-circuitplayground_express/firmware.uf2 build-
circuitplayground_express/firmware.bin

# For programs needing 0x4000 offset (M4 boards)
uf2conv.py -c -b 0x4000 -o build-metro_m4_express/firmware.uf2 build-
metro_M4_express/firmware.bin
```

Installing the bootloader on a fresh/bricked board

If you somehow damaged your bootloader or maybe you have a new board, you can use a JLink to re-install it.

Here's a Learn Guide explaining how to fix the bootloader on a variety of boards using Atmel Studio ()

Here's a short writeup by turbinenreiter on how to do it for the Feather M4 (but adaptable to other boards) ()

What is CircuitPython?

CircuitPython is a programming language designed to simplify experimenting and learning to program on low-cost microcontroller boards. It makes getting started easier than ever with no upfront desktop downloads needed. Once you get your board set up, open any text editor, and get started editing code. It's that simple.

CircuitPython is based on Python

Python is the fastest growing programming language. It's taught in schools and universities. It's a high-level programming language which means it's designed to be easier to read, write and maintain. It supports modules and packages which means it's easy to reuse your code for other projects. It has a built in interpreter which means there are no extra steps, like compiling, to get your code to work. And of course, Python is Open Source Software which means it's free for anyone to use, modify or improve upon.

CircuitPython adds hardware support to all of these amazing features. If you already have Python knowledge, you can easily apply that to using CircuitPython. If you have no previous experience, it's really simple to get started!

Why would I use CircuitPython?

CircuitPython is designed to run on microcontroller boards. A microcontroller board is a board with a microcontroller chip that's essentially an itty-bitty all-in-one computer. The board you're holding is a microcontroller board! CircuitPython is easy to use because all you need is that little board, a USB cable, and a computer with a USB connection. But that's only the beginning.

Other reasons to use CircuitPython include:

- You want to get up and running quickly. Create a file, edit your code, save the file, and it runs immediately. There is no compiling, no downloading and no uploading needed.
- You're new to programming. CircuitPython is designed with education in mind. It's easy to start learning how to program and you get immediate feedback from the board.

- Easily update your code. Since your code lives on the disk drive, you can edit it whenever you like, you can also keep multiple files around for easy experimentation.
- The serial console and REPL. These allow for live feedback from your code and interactive programming.
- File storage. The internal storage for CircuitPython makes it great for data-logging, playing audio clips, and otherwise interacting with files.
- Strong hardware support. CircuitPython has builtin support for microcontroller hardware features like digital I/O pins, hardware buses (UART, I2C, SPI), audio I/O, and other capabilities. There are also many libraries and drivers for sensors, breakout boards and other external components.
- It's Python! Python is the fastest-growing programming language. It's taught in schools and universities. CircuitPython is almost-completely compatible with Python. It simply adds hardware support.

This is just the beginning. CircuitPython continues to evolve, and is constantly being updated. Adafruit welcomes and encourages feedback from the community, and incorporate it into the development of CircuitPython. That's the core of the open source concept. This makes CircuitPython better for you and everyone who uses it!

Install CircuitPython

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 drive to iterate.

Set up CircuitPython Quick Start!

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

> Download the latest version of CircuitPython for this board via circuitpython.org

Further Information

For more detailed info on installing CircuitPython, check out Installing CircuitPython ().
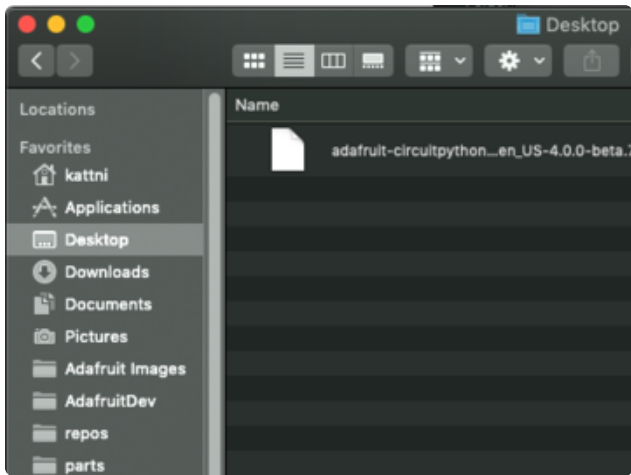
Click the link above and download the latest UF2 file.

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

Plug your Metro M4 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 next to the USB connector on your board, and you will see the NeoPixel RGB LED (indicated by the arrow) turn green. If it turns red, check the USB cable, try another USB port, etc.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!
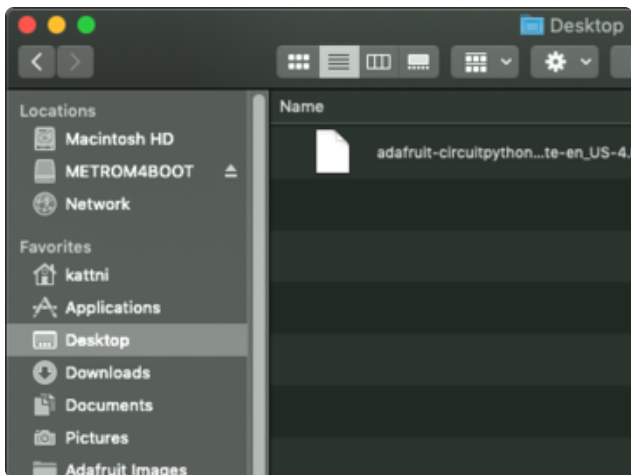
You will see a new disk drive appear called METROM4BOOT.
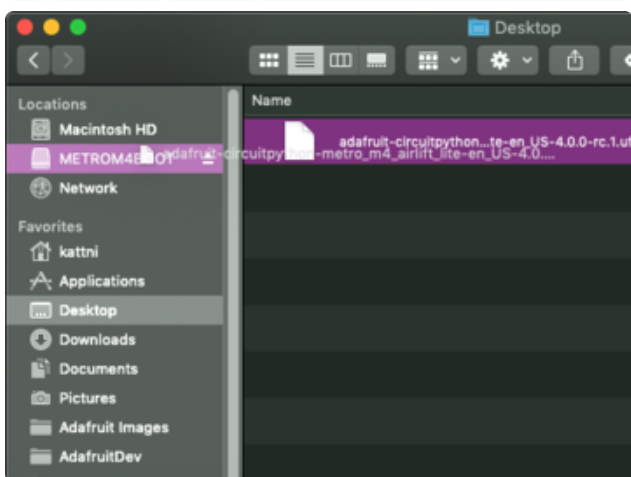
Drag the adafruit_circuitpython_etc.uf2 file to METROM4BOOT.
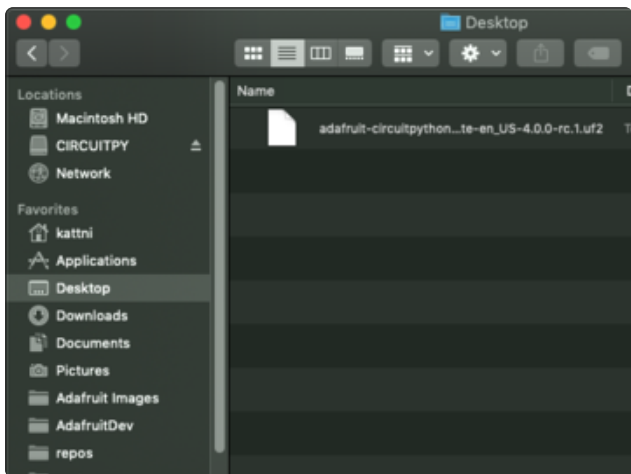
The LED will flash. Then, the METROM4BOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

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

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!

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.

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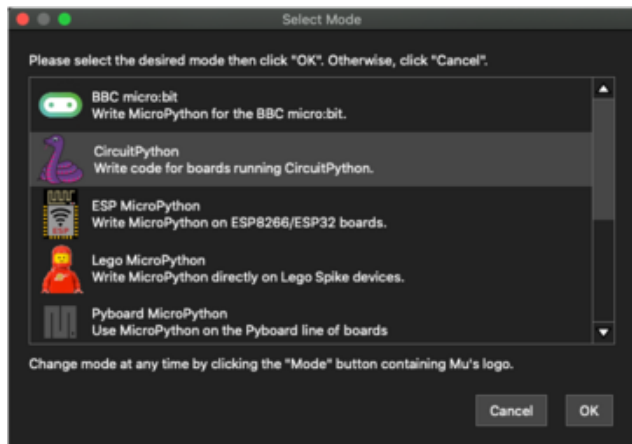
The first time you start Mu, you will be prompted to select your 'mode' - you can always change your mind later. For now please select CircuitPython!

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.
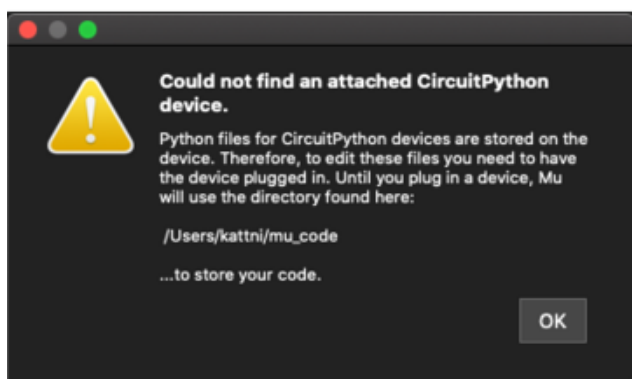
Mu attempts to auto-detect your board on startup, so if you do not have a CircuitPython board plugged in with a CIRCUITPY drive available, Mu will inform you where it will store any code you save until you plug in a board.

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.
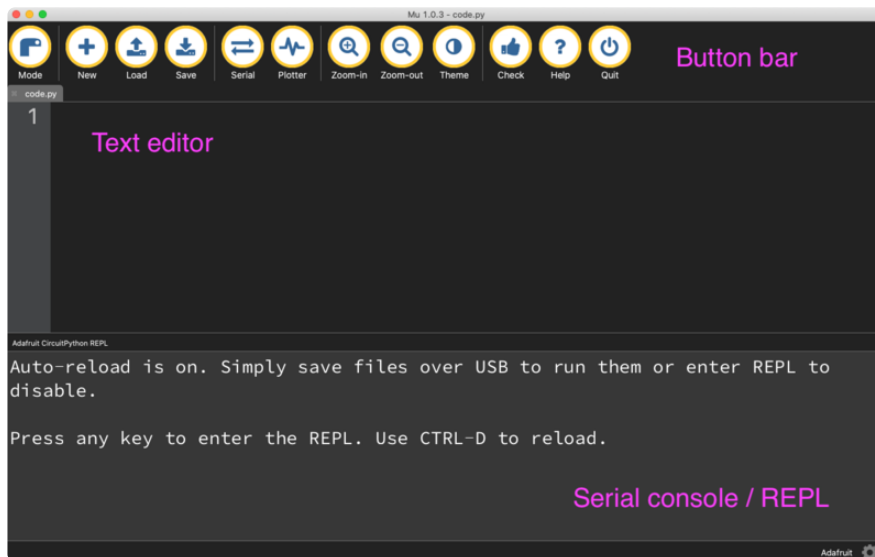
Now you're ready to code! Let's keep going...

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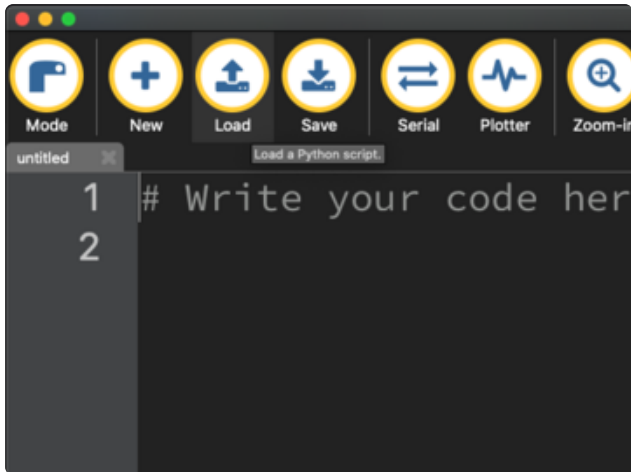
Installing CircuitPython generates a code.py file on your CIRCUITPY drive. To begin your own program, open your editor, and load the code.py file from the CIRCUITPY drive.

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:
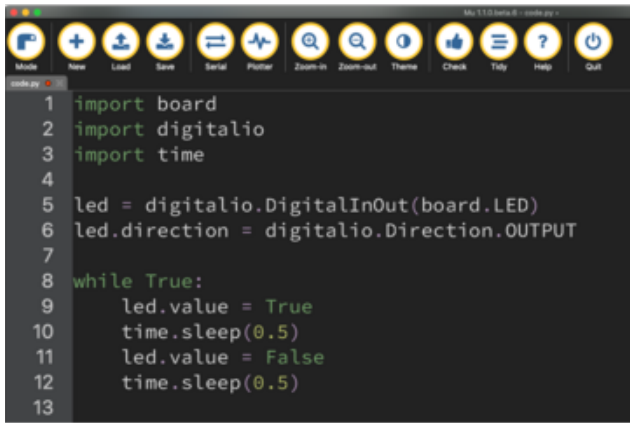
```
import board
import digitalio
import time

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!
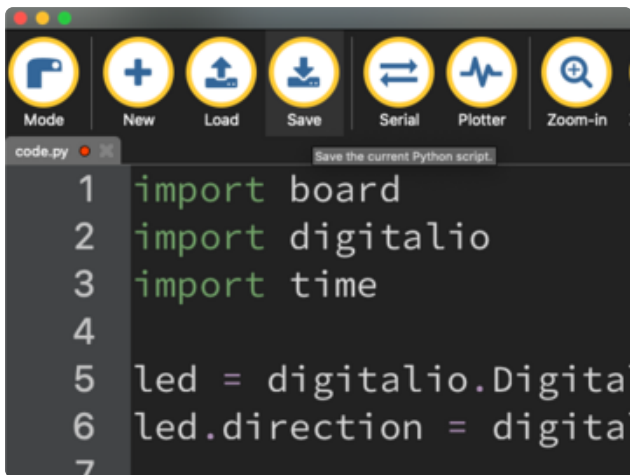
If you're using a KB2040, QT Py or a Trinkey, please download the NeoPixel blink example ().

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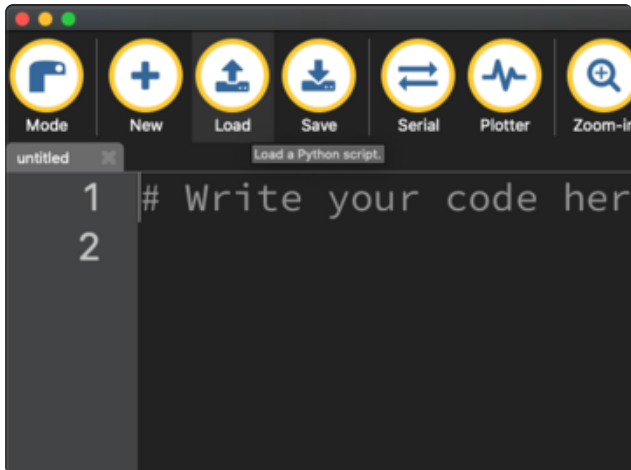
To edit code, open the code.py file on your CIRCUITPY drive into your editor.

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.
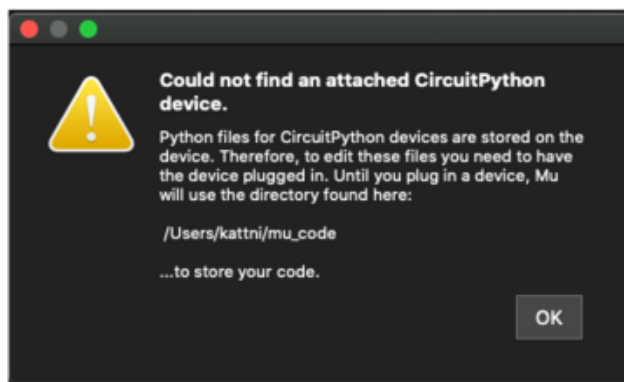
First, make sure your CircuitPython board is plugged in.

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.
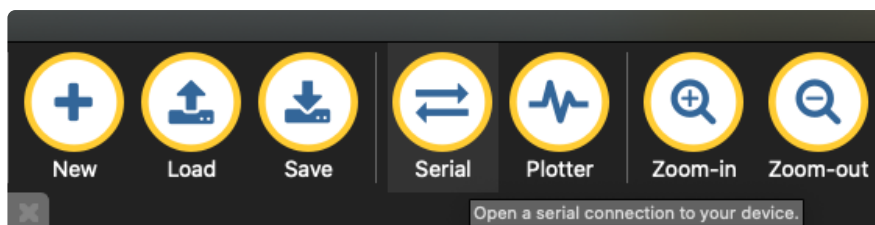
The Mu window will split in two, horizontally, and display the serial console at the bottom.

> 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 S erial 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.

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:
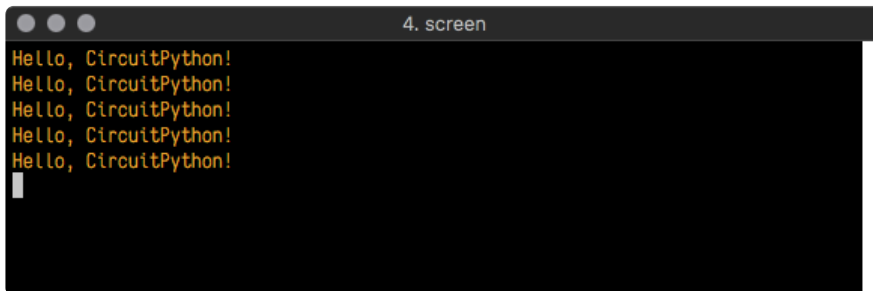
```
import board
import digitalio
import time

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.
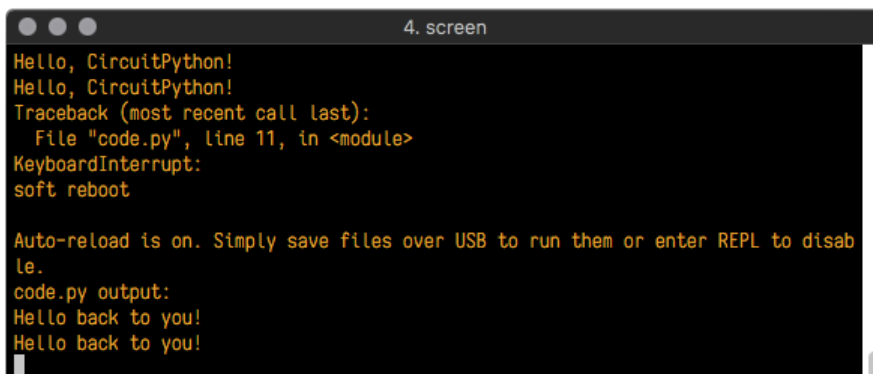
Excellent! Our print statement is showing up in our console! Try changing the printed text to something else.

```
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`
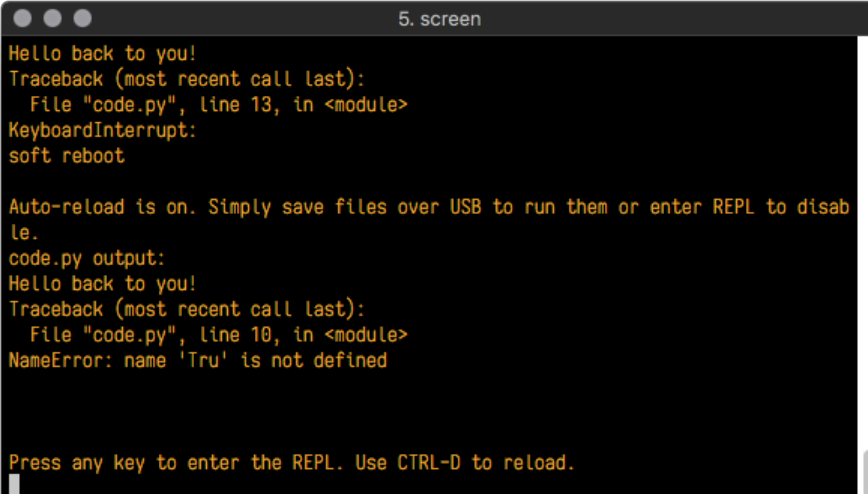
```
import board
import digitalio
import time

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.

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
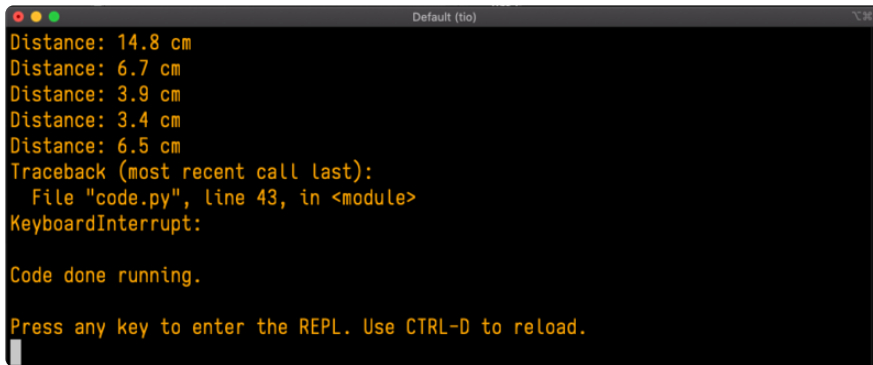
The other feature of the serial connection is the Read-Evaluate-Print-Loop, or REPL. The REPL allows you to enter individual lines of code and have them run immediately. It's really handy if you're running into trouble with a particular program and can't figure out why. It's interactive so it's great 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.
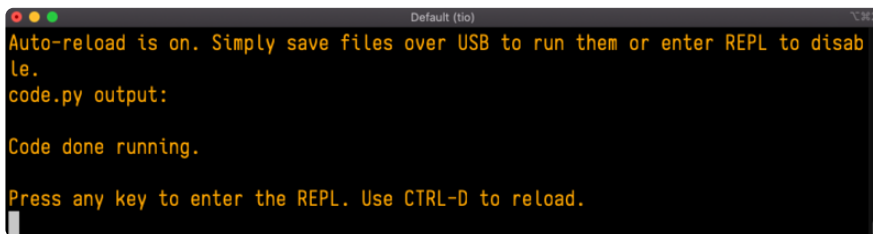
```
Distance: 14.8 cm
Distance: 6.7 cm
Distance: 3.9 cm
Distance: 3.4 cm
Distance: 6.5 cm
Traceback (most recent call last):
  File "code.py", line 43, in <module>
KeyboardInterrupt:

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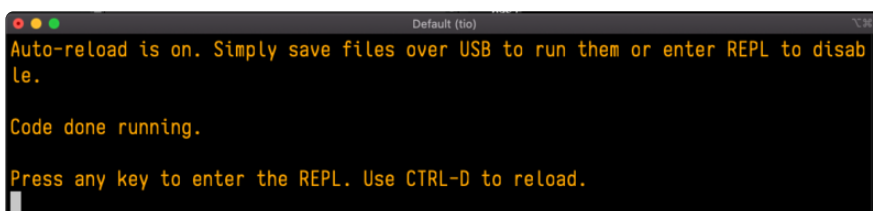
```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:

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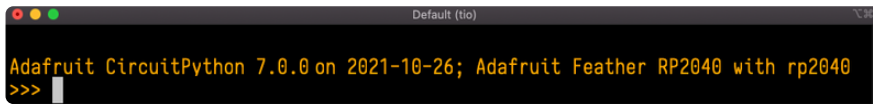
```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

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.

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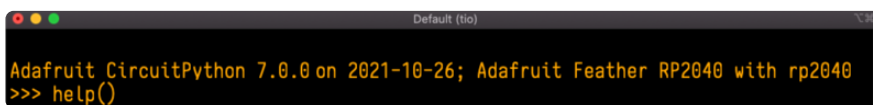
Interacting with the REPL
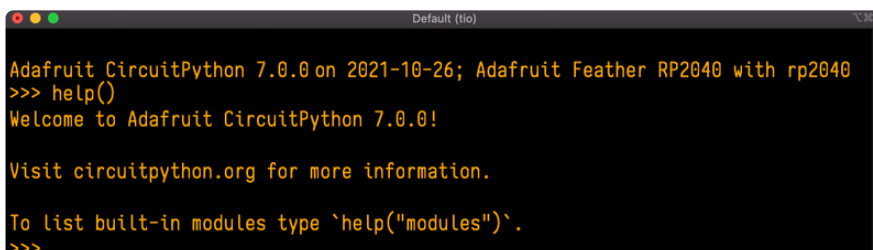
From this prompt you can run all sorts of commands and code. The first thing you'll do is run `help()`. This will tell you where to start exploring the REPL. To run code in the REPL, type it in next to the REPL prompt.

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

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

Then press enter. You should then see a message.

```
Adafruit CircuitPython 7.0.0 on 2021-10-26; Adafruit Feather RP2040 with rp2040
>>> help()
Welcome to Adafruit CircuitPython 7.0.0!

Visit circuitpython.org for more information.

To list built-in modules type `help("modules")`.
>>>
```

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.

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.
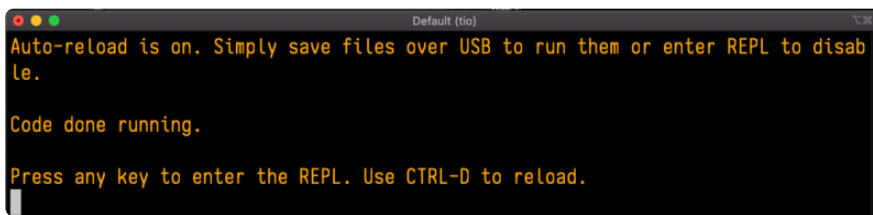
Try typing more into the REPL to see what happens!

> 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!

CircuitPython Libraries

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

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

Your board may ship with a lib folder already, it's in the base directory of the drive. If not, simply create the folder yourself. When you first install CircuitPython, an empty lib directory will be created for you.
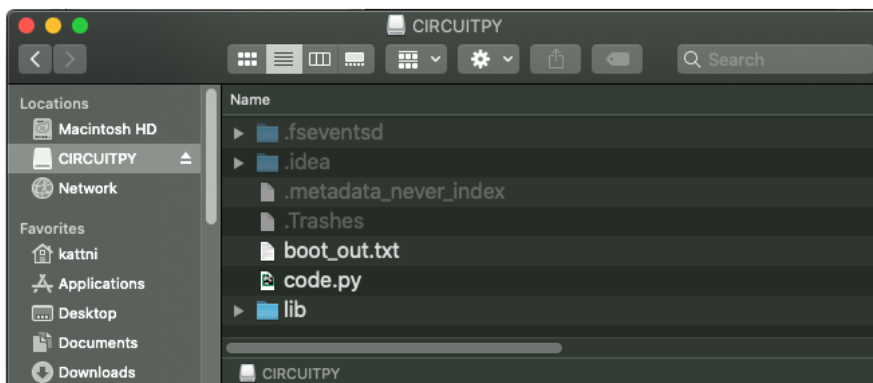
CircuitPython libraries work in the same way as regular Python modules so the Python docs () are an excellent reference for how it all should work. In Python terms, you can

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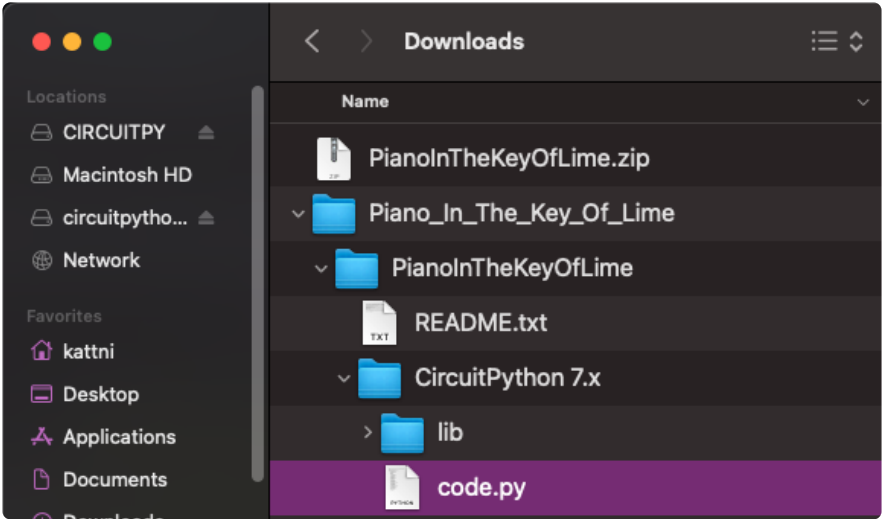
The Piano in the Key of Lime guide was chosen as an example. That guide is specific to Circuit Playground Express, and cannot be used on all boards. Do not expect to download that exact bundle and have it work on your non-CPX microcontroller.

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.

> ### Click for the latest CircuitPython Community Library Bundle release

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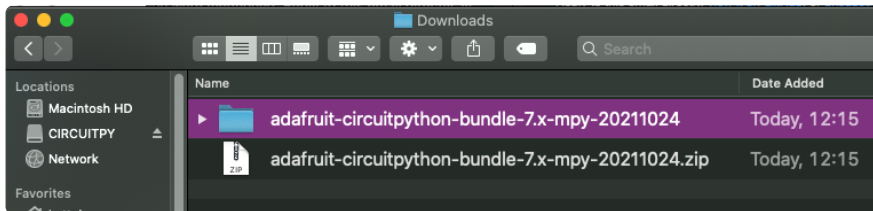
Open the bundle folder. Inside you'll find two information files, and two folders. One folder is the lib bundle, and the other folder is the examples bundle.
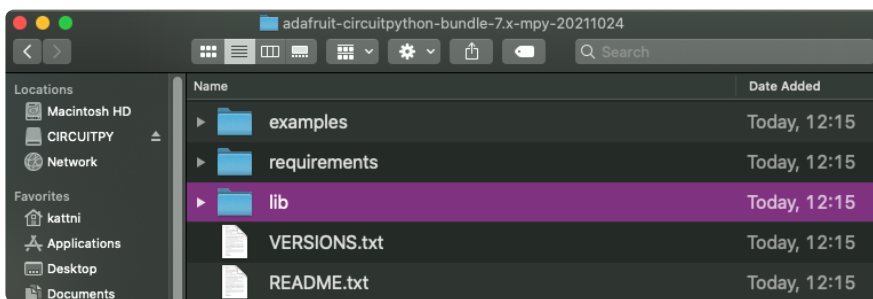
Now open the lib folder. When you open the folder, you'll see a large number of .mpy files, and folders.
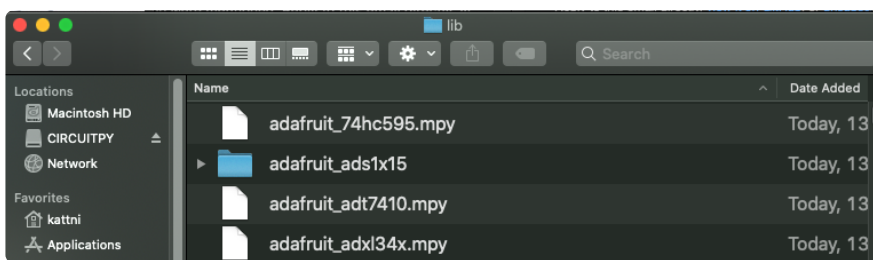
Example Files

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.
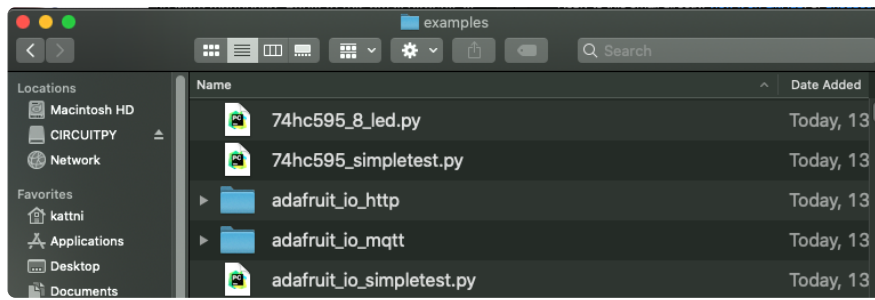
Copying Libraries to Your Board

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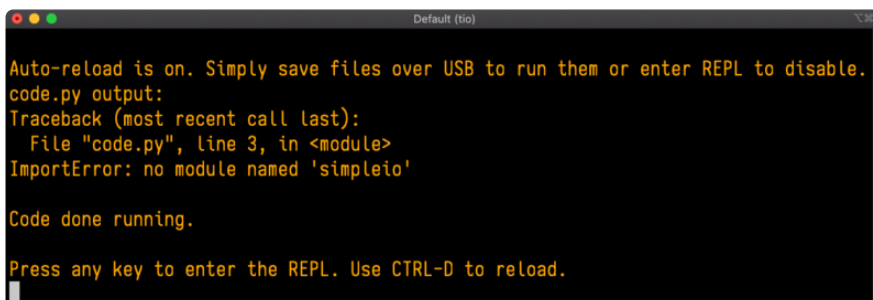
Let's use a modified version of the Blink example.

```
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

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.

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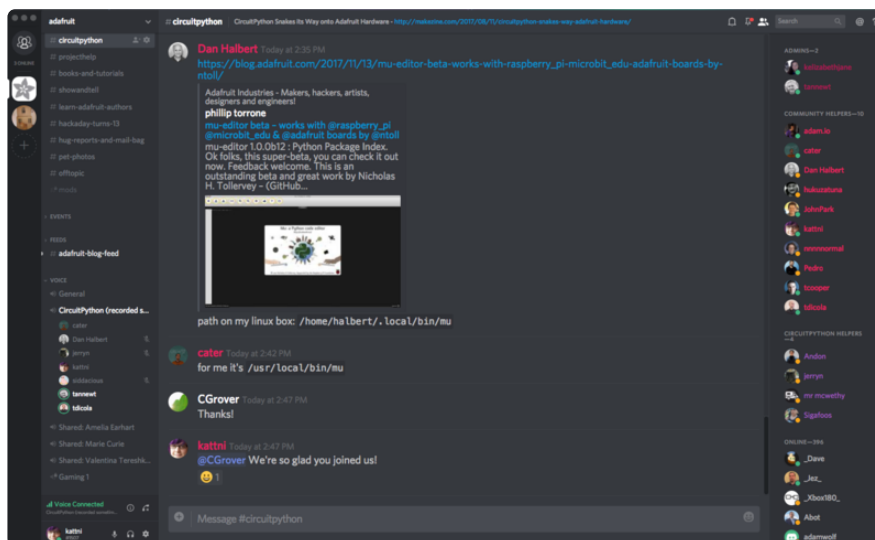
The Adafruit Discord server is the best place to start. Discord is where the community comes together to volunteer and provide live support of all kinds. From general discussion to detailed problem solving, and everything in between, Discord is a digital maker space with makers from around the world.

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 ().

Contributing

If you'd like to contribute to the CircuitPython project, the CircuitPython libraries are a great way to begin. This page is updated with daily status information from the CircuitPython libraries, including open pull requests, open issues and library infrastructure issues.

Do you write a language other than English? Another great way to contribute to the project is to contribute new localizations (translations) of CircuitPython, or update current localizations, using Weblate.

If this is your first time contributing, or you'd like to see our recommended contribution workflow, we have a guide on Contributing to CircuitPython with Git and Github. You can also find us in the #circuitpython channel on the Adafruit Discord.

Have an idea for a new driver or library? File an issue on the CircuitPython repo!

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.

Pull Requests   **Open Issues**   **Library Infrastructure Issues**   **CircuitPython Localization**

This is the current status of open pull requests and issues across all of the library repos.

Open Pull Requests

- Adafruit_CircuitPython_AdafruitIO
  - Call wifi.connect() after wifi.reset() (Open 113 days)
- Adafruit_CircuitPython_ADS1x15
  - Supress f-string recommendation in .pylintrc (Open 1 days)
- Adafruit_CircuitPython_ADT7410
  - Adding critical temp features (Open 168 days)

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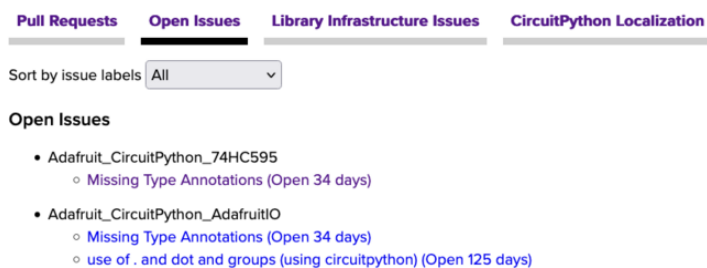
GitHub issues are filed for a number of reasons, including when there is a bug in the library or example code, or when someone wants to make a feature request. Issues are a great way to find an opportunity to contribute directly to the libraries by updating code or documentation. If you're interested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

Sort by issue labels | ✓ All
Good first issue
Documentation
Hacktoberfest
Open Issues None
Bug
- Adafruit_Circu Enhancement er
  ○ Missing Hacktoberfest er
Question
- Adafruit_Circu Support er
  ○ Missing Help wanted er
  ○ use of . Duplicate si
Needs retest
- Adafruit_Circu er
  ○ ad1115 to er
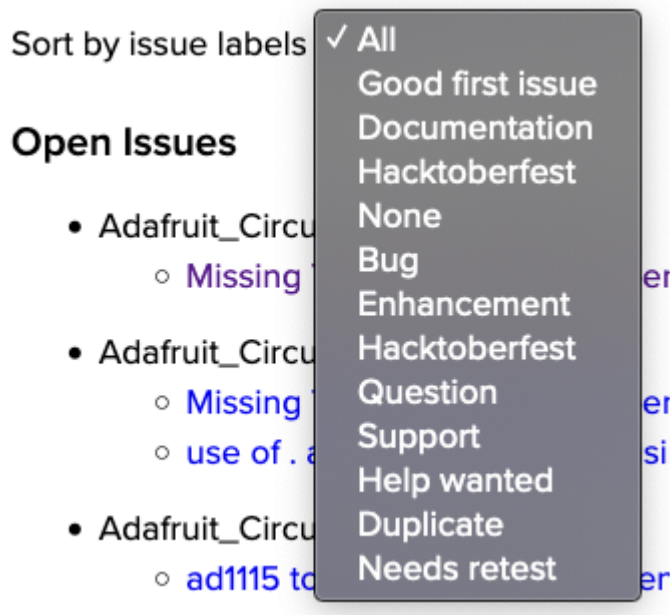
If you're new to everything, new to contributing to open source, or new to contributing to the CircuitPython project, you can choose "Good first issue". Issues with that label are well defined, with a finite scope, and are intended to be easy for someone new to figure out.

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 Discord () to answer questions.

Library Infrastructure Issues

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

Pull Requests **Open Issues** **Library Infrastructure Issues** **CircuitPython Localization**

Library Infrastructure Issues

The following are issues with the library infrastructure. Having a standard library structure greatly improves overall maintainability. Accordingly, we have a series of checks to ensure the standard is met. Most of these are changes that can be made via a pull request, however there are a few checks reported here that require changes to GitHub settings. If you are interested in addressing any of these issues, please feel free to contact us with any questions.

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.
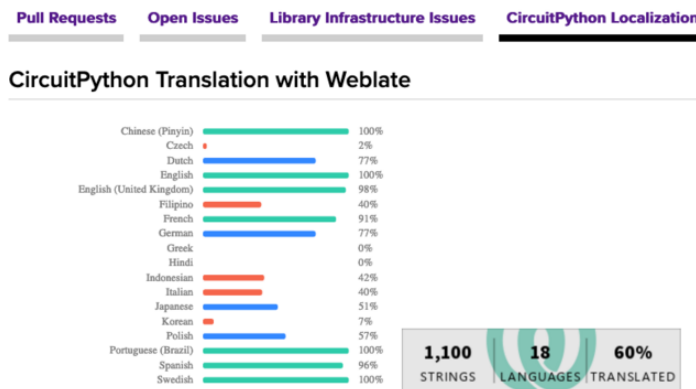
If you speak another language, you can help translate CircuitPython! The translations apply to informational and error messages that are within the CircuitPython core. It means that folks who do not speak English have the opportunity to have these messages shown to them in their own language when using 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
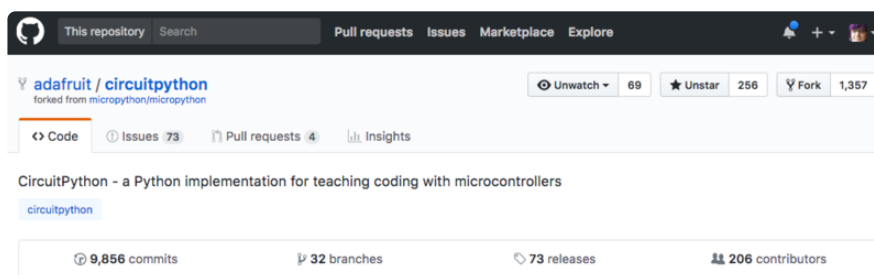
Whether you're just beginning or are life-long programmer who would like to contribute, there are ways for everyone to be a part of the CircuitPython project. The CircuitPython core is written in C. The libraries are written in Python. GitHub is the best source of ways to contribute to the CircuitPython 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 ().
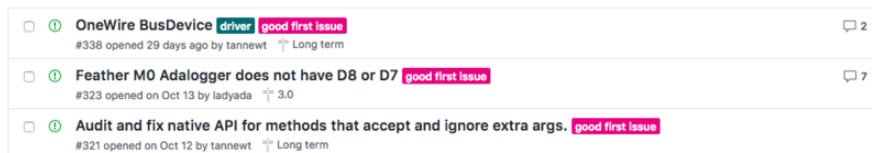
Already experienced and looking for a challenge? Checkout the rest of either issues list and you'll find plenty of ways to contribute. You'll find all sorts of things, from new driver requests, to library bugs, to core module updates. There's plenty of opportunities for everyone at any level!

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
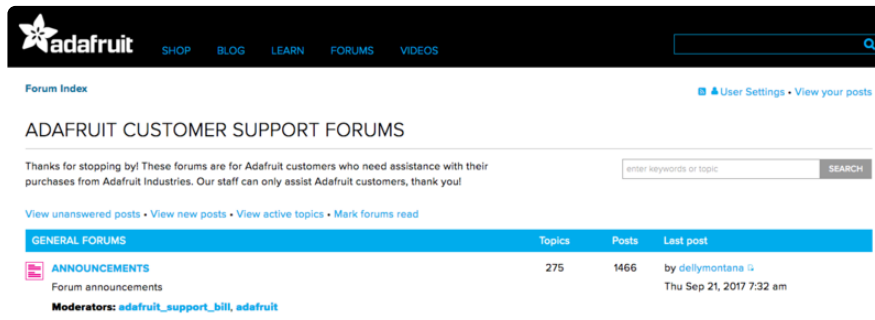
The Adafruit Forums () are the perfect place for support. Adafruit has wonderful paid support folks to answer any questions you may have. Whether your hardware is giving you issues or your code doesn't seem to be working, the forums are always there for you to ask. You need an Adafruit account to post 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.
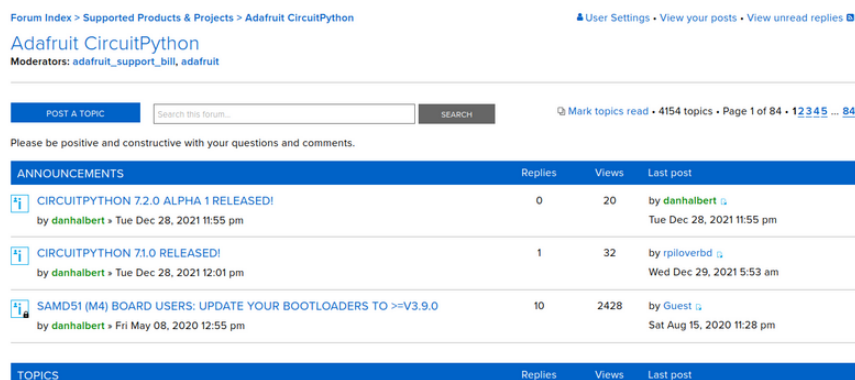
Be sure to include the steps you took to get to where you are. If it involves wiring, post a picture! If your code is giving you trouble, include your code in your post! These are great ways to make sure that there's enough information to help you with your issue.

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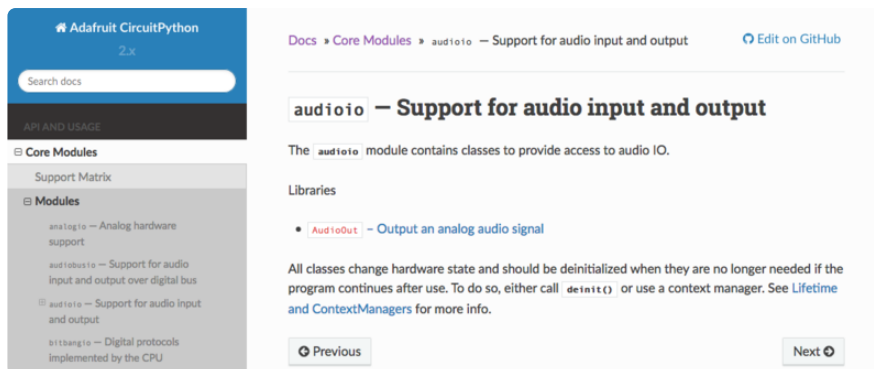
Read the Docs () is a an excellent resource for a more detailed look at the CircuitPython core and the CircuitPython libraries. This is where you'll find things like API documentation and example code. For an in depth look at viewing and understanding Read the Docs, check out the CircuitPython Documentation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

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.
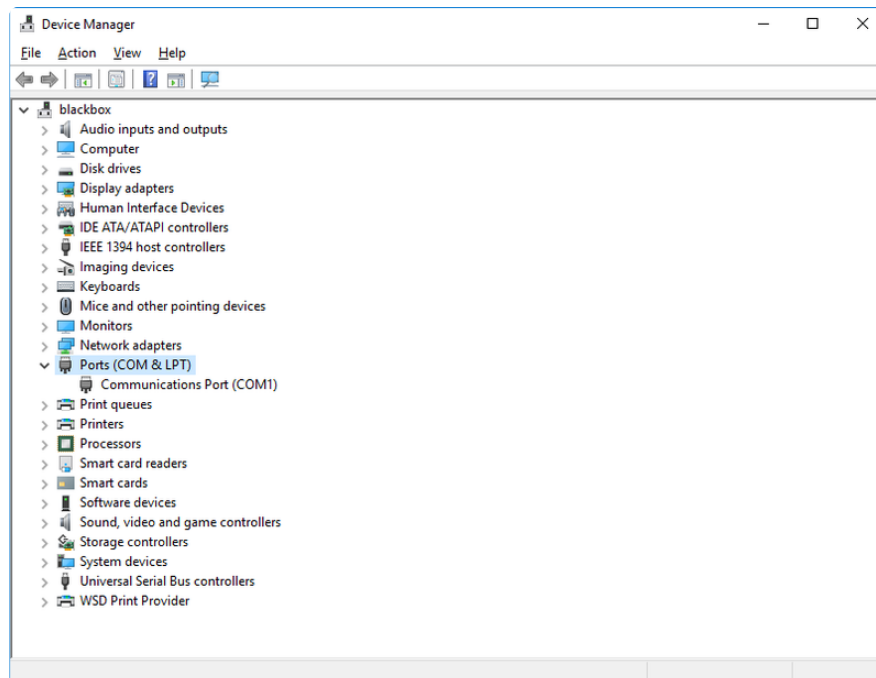
Now plug in your board. The Device Manager list will refresh and a new item will appear under Ports (COM & LPT). You'll find a different (COM#) after this item in the list.
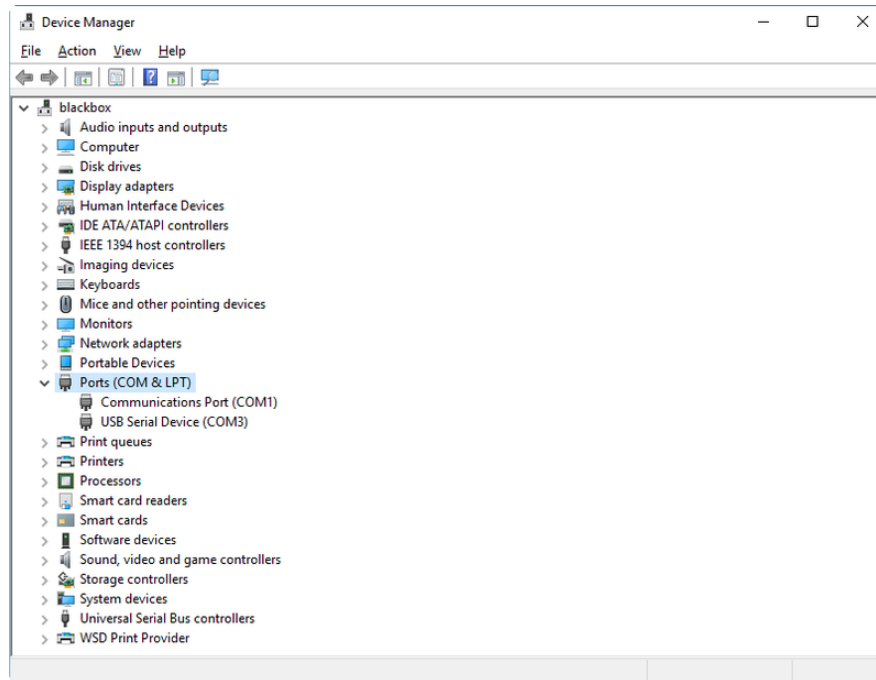
Sometimes the item will refer to the name of the board. Other times it may be called something like USB Serial Device, as seen in the image above. Either way, there is a new (COM#) following the name. This is the port your board is using.

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.
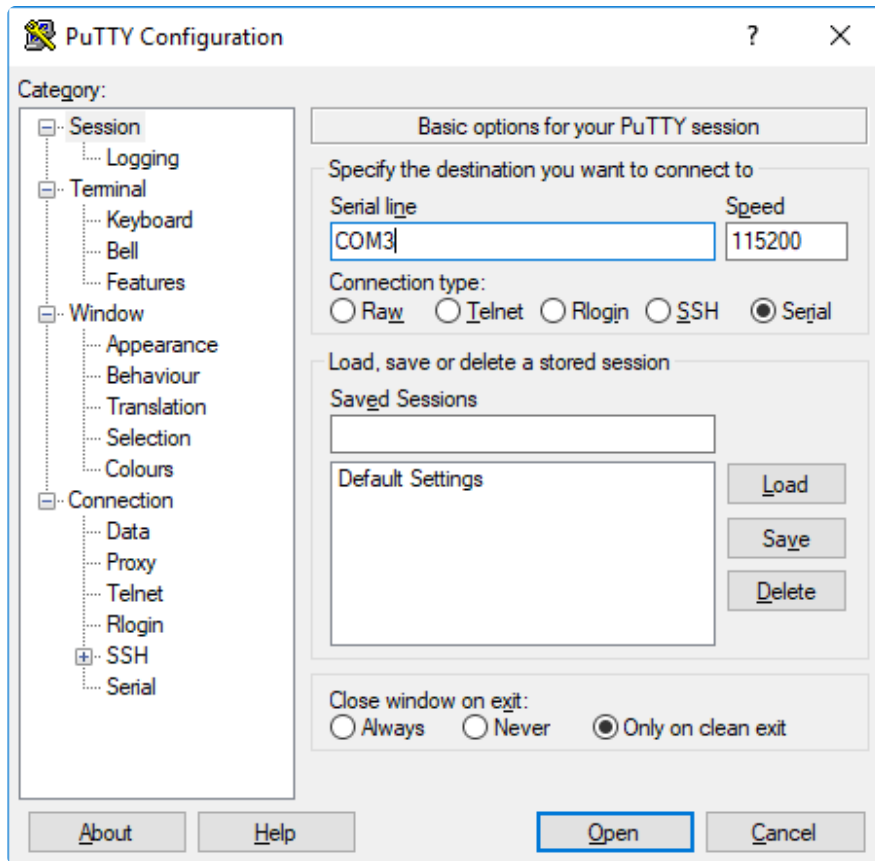
Once your settings are entered, you're ready to connect to the serial console. Click "Open" at the bottom of the window. A new window will open.
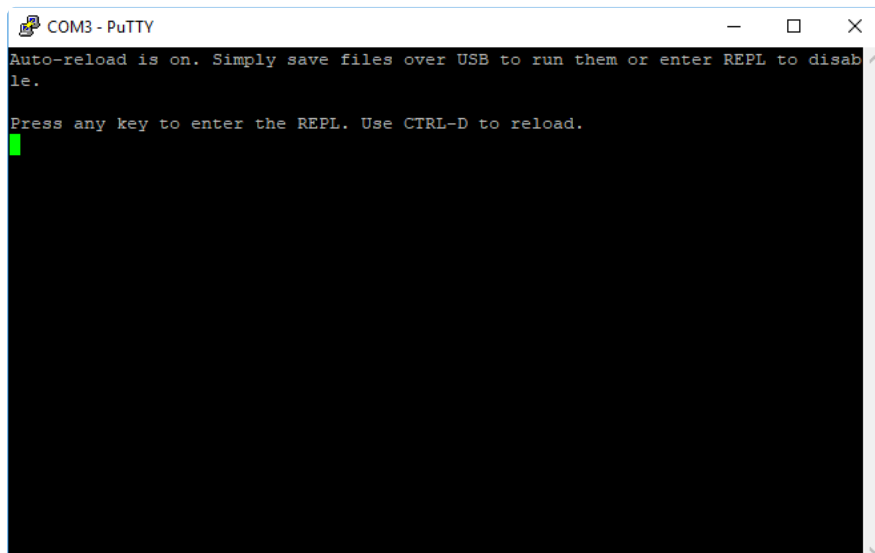
If no code is running, the window will either be blank or will look like the window above. Now you're ready to see the results of your code.

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

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.
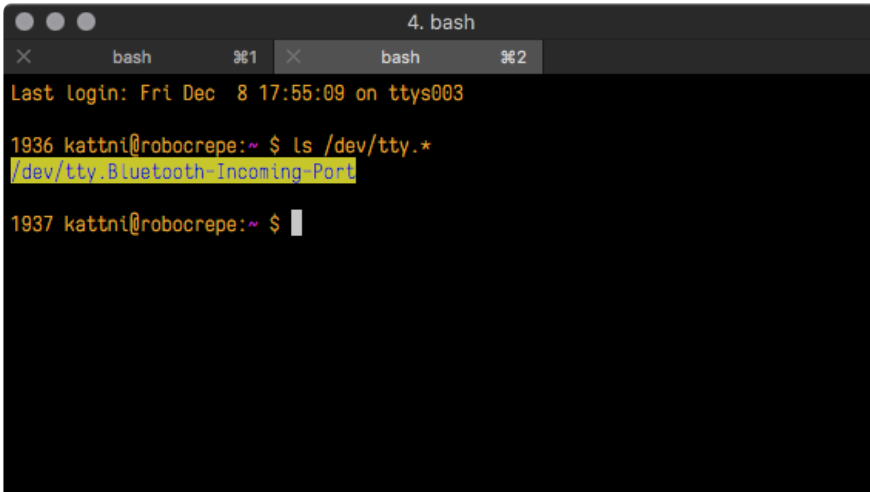
What's the Port?

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.
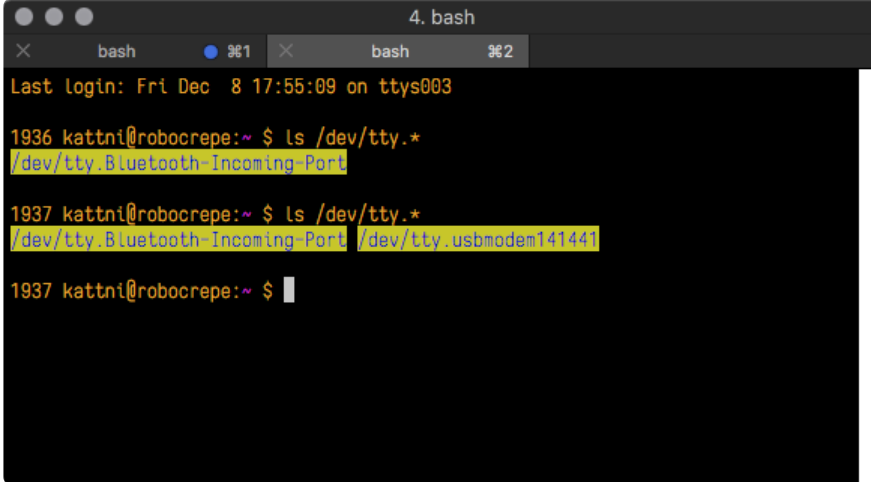
Now, plug your board. In Terminal, type:

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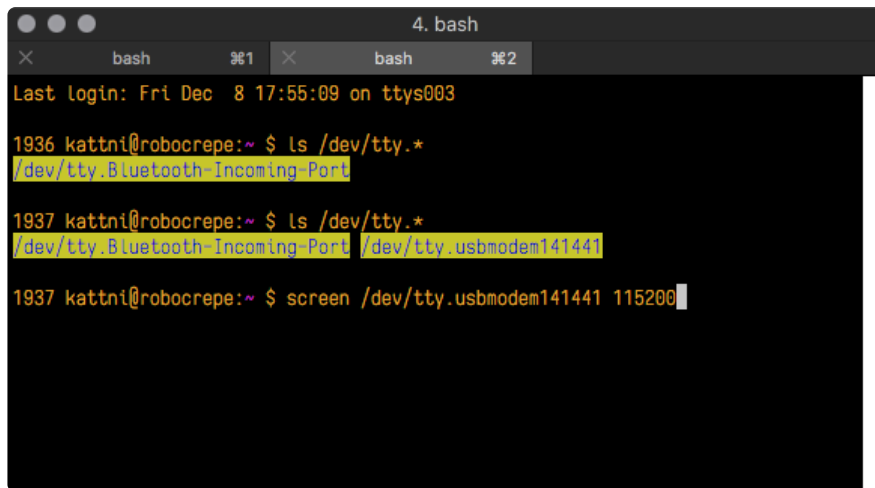
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

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!

"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 ()](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.
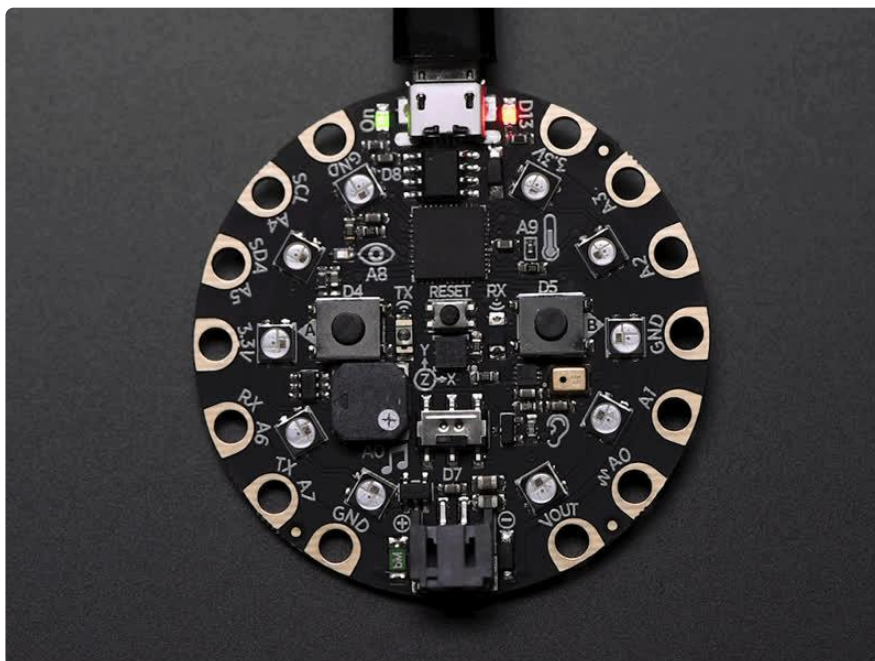
Then find the downloaded MakeCode .uf2 file and drag it to the CPLAYBOOT drive.
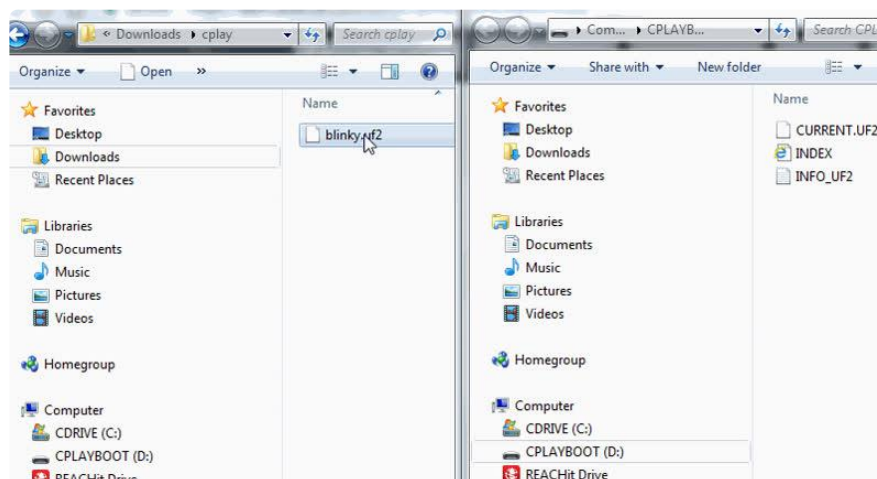
Your MakeCode is now running and CircuitPython has been removed. Going forward you only have to single click the reset button to get to CPLAYBOOT. This is an idiosyncrasy of MakeCode.

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.
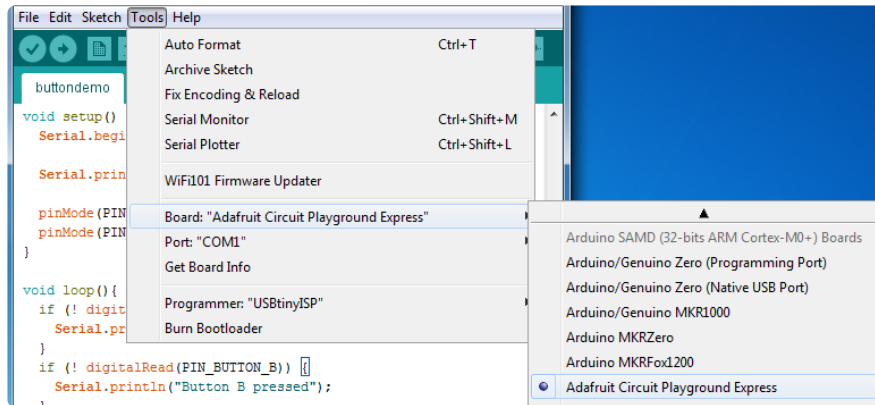
Select the correct matching Port:
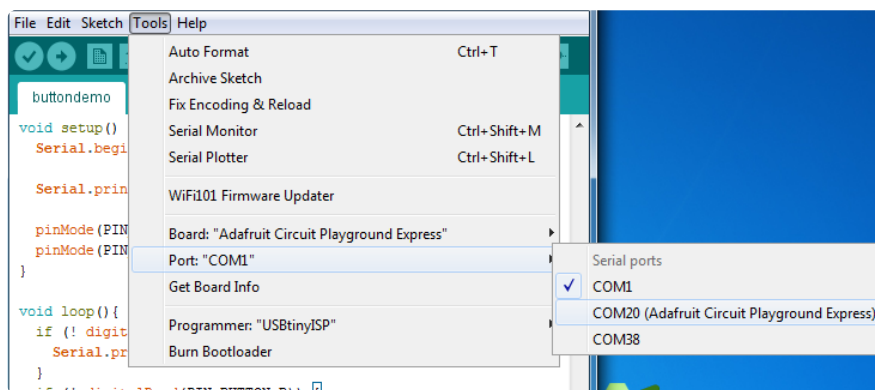
Create a new simple Blink sketch example:

```
// 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.

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 boardna meBOOT 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.
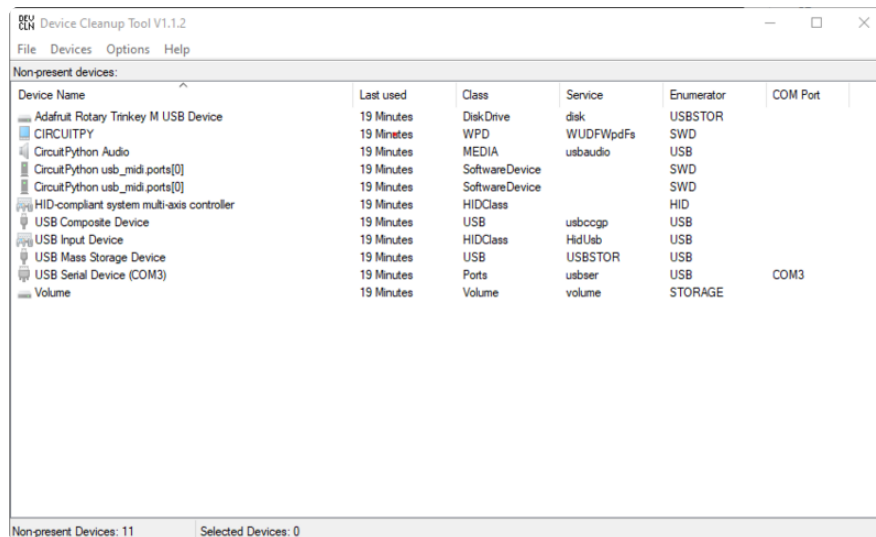
Select all the devices you want to remove, and then press Delete. It is usually safe just to select everything. Any device that is removed will get a fresh install when you plug it in. Using the Device Cleanup Tool also discards all the COM port assignments for the unplugged boards. If you have used 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.
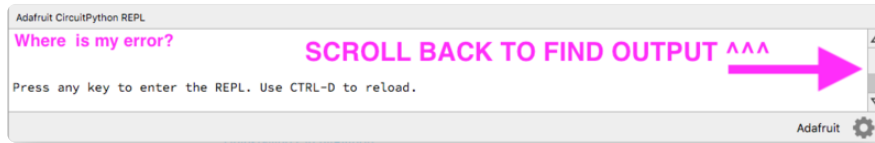
This applies to any kind of serial output whether it be error messages or print statements. So before you start trying to debug your problem on the hardware side, be sure to check that you haven't simply missed the serial messages due to serial output panel height.

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.disable_autoreload()
```

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
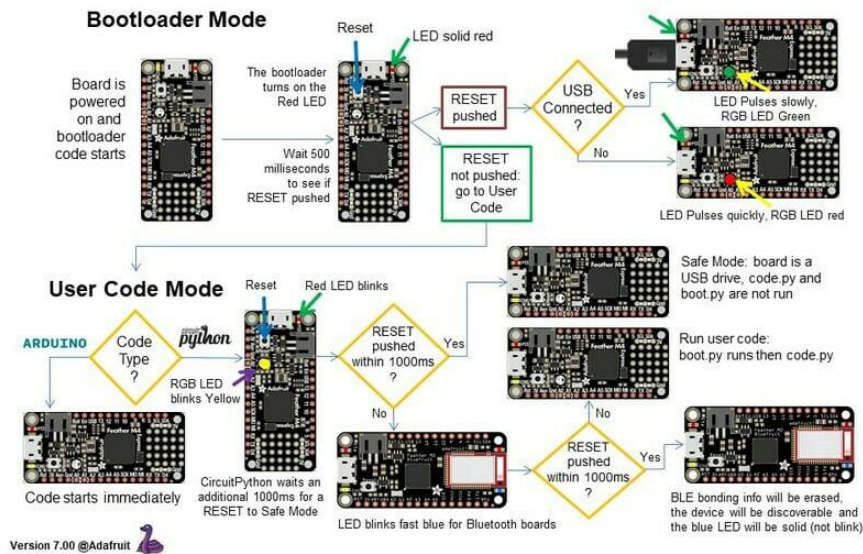
CircuitPython 6.3.0 and earlier

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
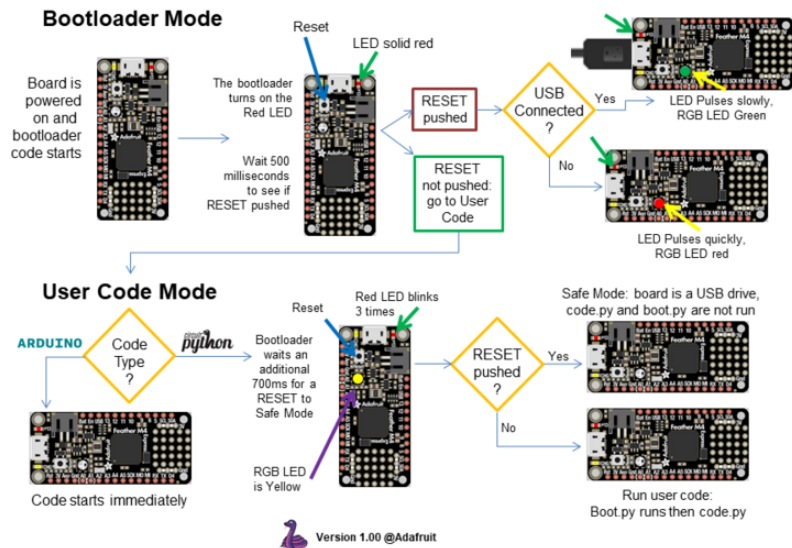
Serial console showing `ValueError: Incompatible .mpy file`

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
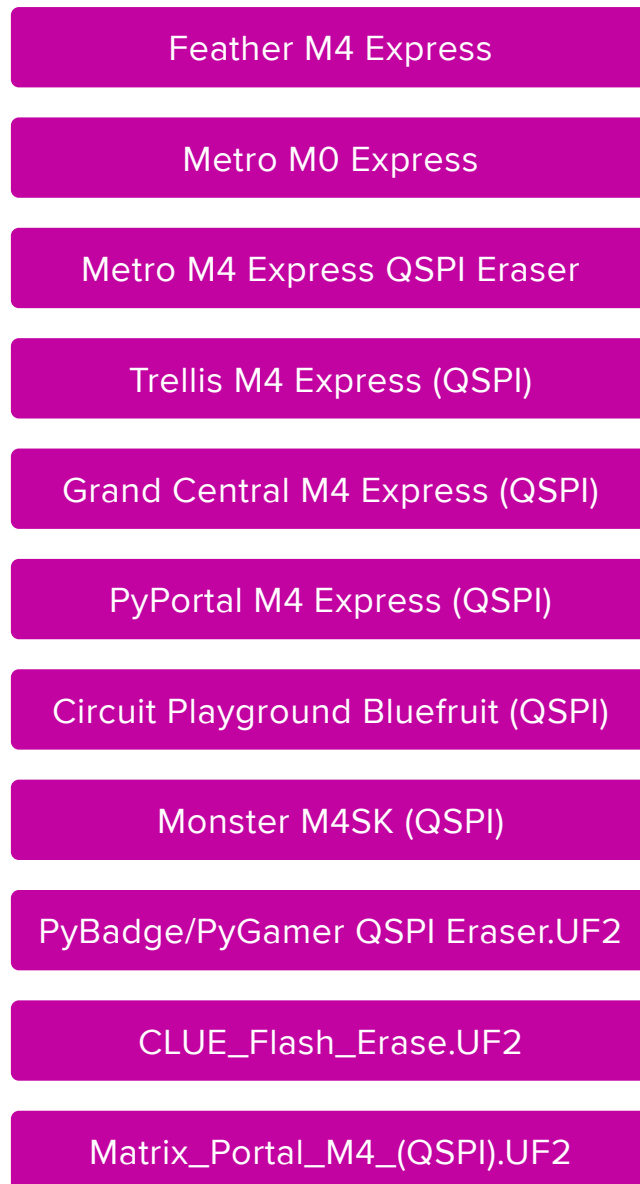
Feather M4 Express

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

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:

SAMD21 non-Express Boards

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.
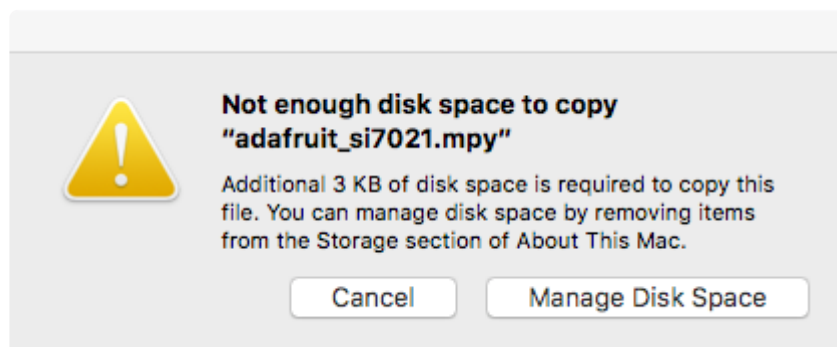
Delete something!

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 ()](#) 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.

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!

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.

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

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.

CircuitPython Setup

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

Adafruit CircuitPython Bundle

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

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

Download latest Library Bundle

</div>

Download the adafruit-circuitpython-bundle-version-mpy-*.zip bundle zip file, and unzip a folder of the same name. Inside you'll find a lib folder. The entire collection of libraries is too large to fit on the CIRCUITPY drive. Instead, add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

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

- adafruit_esp32spi - This is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- neopixel - for controlling the onboard neopixel
- adafruit_bus_device - low level support for I2C/SPI

Internet Connect!

Once you have CircuitPython setup and libraries installed we can get your board connected to the Internet. Note that access to enterprise level secured WiFi networks is not currently supported, only WiFi networks that require SSID and password.

To get connected, you will need to start by creating a secrets file.

What's a secrets file?

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a `secrets.py` file, that is in your CIRCUITPY drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your secrets.py file should look like this:

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

secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'github_token' : 'fawfj23rakjnfawiefa',
```

```
        'hackaday_token' : 'h4xx0rs3kret',
    }
```

Inside is a python dictionary named secrets with a line for each entry. Each entry has an entry name (say `'ssid'`) and then a colon to separate it from the entry key `'home ssid'` and finally a comma ,

At a minimum you'll need the `ssid` and `password` for your local WiFi setup. As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing github or the hackaday API. Other non-secret data like your timezone can also go here, just cause it's called secrets doesn't mean you can't have general customization data in there!

For the correct time zone string, look at http://worldtimeapi.org/timezones () and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your secrets.py - keep that out of GitHub, Discord or other project-sharing sites.

Connect to WiFi

OK now you have your secrets setup - you can connect to the Internet.

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

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

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

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

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

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

print("ESP32 SPI webclient test")

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


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

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

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

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

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

requests.set_socket(socket, esp)
```

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

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

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

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

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

print("Done!")
```

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

Don't forget you'll also need to create the `secrets.py` file as seen above, with your WiFi ssid and password.

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

In order, the example code...

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

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

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

Tells our `requests` library the type of socket we're using (socket type varies by connectivity type - we'll be using the `adafruit_esp32spi_socket` for this example). We'll also set the interface to an `esp` object. This is a little bit of a hack, but it lets us use `requests` like CPython does.

```
requests.set_socket(socket, esp)
```

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

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

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

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

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

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

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

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

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

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

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

Requests

We've written a requests-like () library for web interfacing named Adafruit_CircuitPython_Requests (). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

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

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

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


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

# adafruit_requests usage with an esp32spi_socket
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# Add a secrets.py to your filesystem that has a dictionary called secrets with
"ssid" and
# "password" keys with your WiFi credentials. DO NOT share that file or commit it
into Git or other
# source control.
# pylint: disable=no-name-in-module,wrong-import-order
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

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

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

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

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

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

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

# Initialize a requests object with a socket and esp32spi interface
socket.set_interface(esp)
requests.set_socket(socket, esp)

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_GET_URL = "https://httpbin.org/get"
JSON_POST_URL = "https://httpbin.org/post"

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

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

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

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

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

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

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

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

```
print("-" * 40)
response.close()
```

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

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

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

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

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

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

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)
```

HTTP GET with Requests

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

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

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

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

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

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

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

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

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

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

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

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

HTTP POST with Requests

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

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

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

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

```
    json_data = {"Date" : "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
```

```
response = requests.post(JSON_POST_URL, json=json_data)
print('-'*40)

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

Advanced Requests Usage

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

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

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

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

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


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

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

# Add a secrets.py to your filesystem that has a dictionary called secrets with
"ssid" and
# "password" keys with your WiFi credentials. DO NOT share that file or commit it
```

```
into Git or other
# source control.
# pylint: disable=no-name-in-module,wrong-import-order
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

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

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

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

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

# Initialize a requests object with a socket and esp32spi interface
socket.set_interface(esp)
requests.set_socket(socket, esp)

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

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

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

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

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

# Close, delete and collect the response data
response.close()
```

WiFi Manager

That simpletest example works but it's a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the

connection/status/requests loop for you - reconnecting if WiFi drops, resetting the ESP32 if it gets into a bad state, etc.

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

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

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

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


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

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

print("ESP32 SPI webclient test")

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

```python
        raise

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

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

    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
    esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
    """Use below for Most Boards"""
    status_light = neopixel.NeoPixel(
        board.NEOPIXEL, 1, brightness=0.2
    )  # Uncomment for Most Boards
    """Uncomment below for ItsyBitsy M4"""
    # status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1,
    brightness=0.2)
    # Uncomment below for an externally defined RGB LED
    # import adafruit_rgbled
    # from adafruit_esp32spi import PWMOut
    # RED_LED = PWMOut.PWMOut(esp, 26)
    # GREEN_LED = PWMOut.PWMOut(esp, 27)
    # BLUE_LED = PWMOut.PWMOut(esp, 25)
    # status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
    wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

    counter = 0

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

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

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

- `aio_username`
- `aio_key`

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

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

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

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

- If you do not know how to set up a feed, follow this page and come back when you've set up a feed named `test` . ()

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

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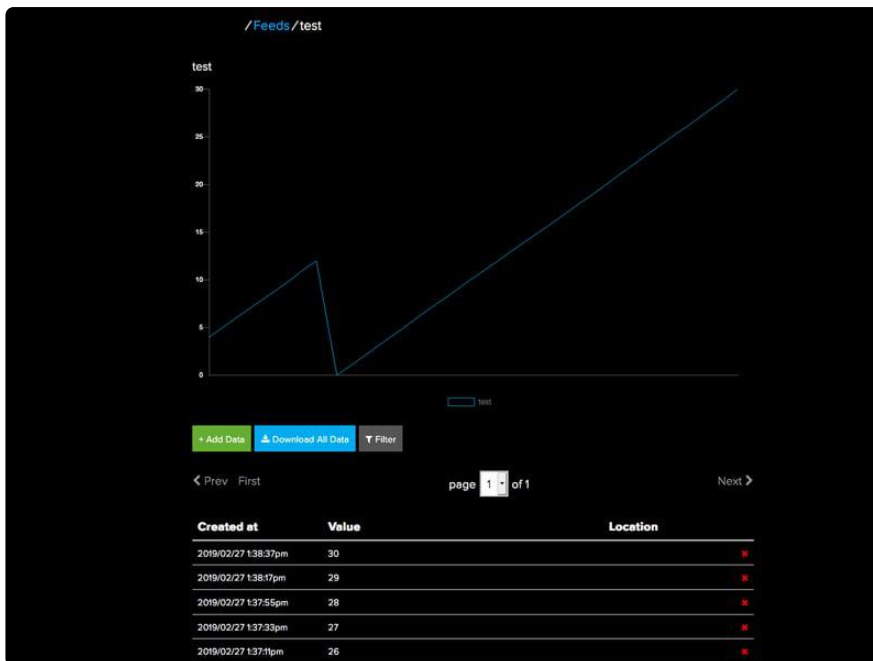
CircuitPython BLE

CircuitPython BLE UART Example

It's easy to use Adafruit AirLift ESP32 co-processor boards for Bluetooth Low Energy (BLE) with CircuitPython. When you reset the ESP32, you can put it in WiFi mode (the default), or in BLE mode; you cannot use both modes simultaneously.

Here's a simple example of using BLE to connect CircuitPython with the Bluefruit Connect app. Use CircuitPython 6.0.0 or later.

Note: Don't confuse the ESP32 with the ESP32-S2, which is a different module with a similar name. The ESP32-S2 does not support BLE.

> Currently the AirLift support for CircuitPython only provides BLE peripheral support. BLE central is under development. So you cannot connect to BLE devices like Heart Rate monitors, etc., but you can act as a BLE peripheral yourself.

On-Board Airlift Co-Processor - No Wiring Needed

If you have an Adafruit Metro M4 AirLift Lite, an Adafruit PyPortal (regular, Pynt, or Tita no), an Adafruit MatrixPortal, or other Adafruit board with an onboard ESP32 co-

processor, then everything is prewired for you, and the pins you need to use are predefined in CircuitPython.

Update the AirLift Firmware

You will need to update the AirLift's firmware to at least version 1.7.1. Previous versions of the AirLift firmware do not support BLE.

Follow the instructions in the guide below, and come back to this page when you've upgraded the AirLift's firmware:

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

Upgrade ESP32 AirLift Firmware

</div>

> Ensure the AirLift firmware is version 1.7.1 or higher for BLE to work.

Install CircuitPython Libraries

Make sure you are running the [latest version of Adafruit CircuitPython]() for your board; you'll need 6.0.0 or later.

Next you'll need to install the necessary libraries to use the hardware and BLE. Carefully follow the steps to find and install these libraries from [Adafruit's CircuitPython library bundle](). Our CircuitPython starter guide has [a great page on how to use the library bundle]().

Install these libraries from the bundle:

- adafruit_airlift
- adafruit_ble

Before continuing make sure your board's lib folder or root filesystem has the adafruit_airlift and adafruit_ble folders copied over.

Install the Adafruit Bluefruit LE Connect App

The Adafruit Bluefruit LE Connect iOS and Android apps allow you to connect to BLE peripherals that provide a over-the-air "UART" service. Follow the instructions in the [B luefruit LE Connect Guide ()](#) to download and install the app on your phone or tablet.

Copy and Adjust the Example Program

Copy the program below to the file code.py on CIRCUITPY on your board.

TAKE NOTE: Adjust the program as needed to suit the AirLift board you have. Comment and uncomment lines 12-39 below as necessary.

```
import board

from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService

from adafruit_airlift.esp32 import ESP32

# If you are using a Metro M4 Airlift Lite, PyPortal,
# or MatrixPortal, you can use the default pin settings.
# Leave this DEFAULT line uncommented.
esp32 = ESP32() # DEFAULT

# If you are using CircuitPython 6.0.0 or earlier,
# on PyPortal and PyPortal Titano only, use the pin settings
# below. Comment out the DEFAULT line above and uncomment
# the line below. For CircuitPython 6.1.0, the pin names
# have changed for these boards, and the DEFAULT line
# above is correct.
# esp32 = ESP32(tx=board.TX, rx=board.RX)

# If you are using an AirLift FeatherWing or AirLift Bitsy Add-On,
# use the pin settings below. Comment out the DEFAULT line above
# and uncomment the lines below.
# If you are using an AirLift Breakout, check that these
# choices match the wiring to your microcontroller board,
# or change them as appropriate.
# esp32 = ESP32(
#     reset=board.D12,
#     gpio0=board.D10,
#     busy=board.D11,
#     chip_select=board.D13,
#     tx=board.TX,
#     rx=board.RX,
# )

# If you are using an AirLift Shield,
# use the pin settings below. Comment out the DEFAULT line above
# and uncomment the lines below.
# esp32 = ESP32(
#     reset=board.D5,
#     gpio0=board.D6,
#     busy=board.D7,
#     chip_select=board.D10,
```

```
#     tx=board.TX,
#     rx=board.RX,
# )

adapter = esp32.start_bluetooth()

ble = BLERadio(adapter)
uart = UARTService()
advertisement = ProvideServicesAdvertisement(uart)

while True:
    ble.start_advertising(advertisement)
    print("waiting to connect")
    while not ble.connected:
        pass
    print("connected: trying to read input")
    while ble.connected:
        # Returns b'' if nothing was read.
        one_byte = uart.read(1)
        if one_byte:
            print(one_byte)
            uart.write(one_byte)
```

Talk to the AirLift via the Bluefruit LE Connect App

Start the Bluefruit LE Connect App on your phone or tablet. You should see a CIRCUITPY device available to connect to. Tap the Connect button (1):
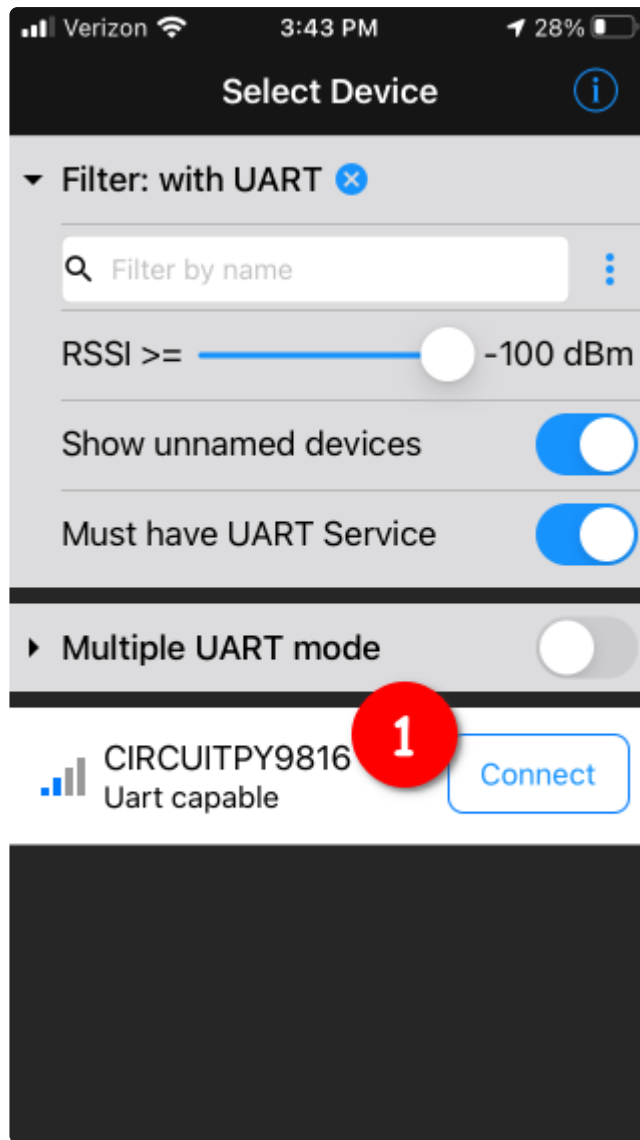
You'll then see a list of Bluefruit Connect functions ("modules"). Choose the UART module (2):
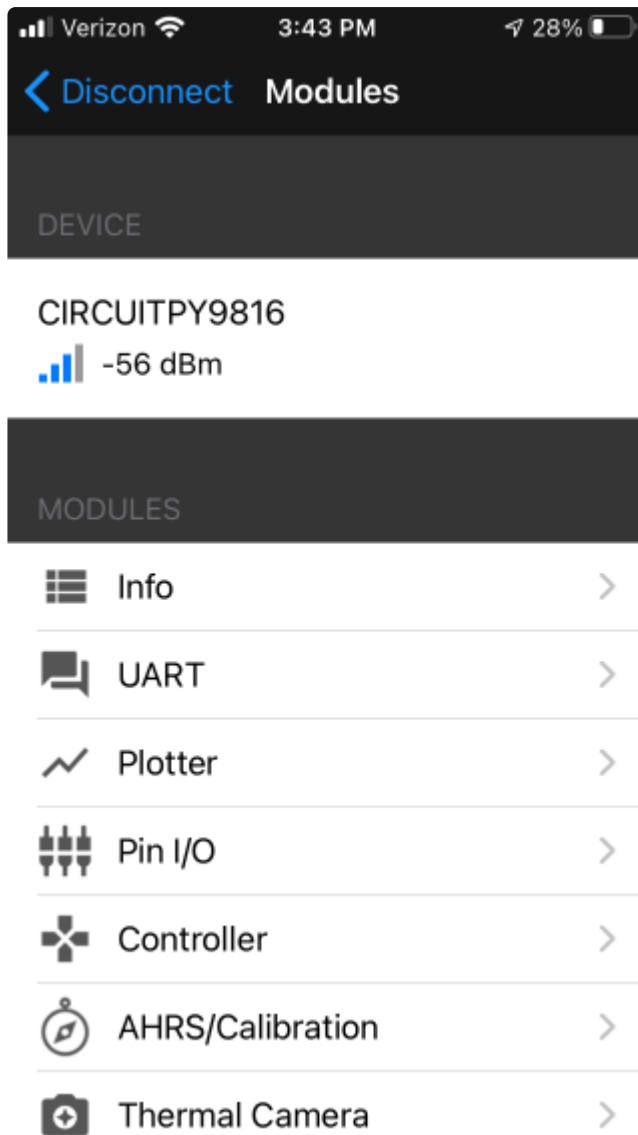
On the UART module page, you can type a string and press Send (3). You'll see that string entered, and then see it echoed back (echoing is in gray).

CircuitPython Essentials

You've gone through the Welcome to CircuitPython guide (). You've already gotten everything setup, and you've gotten CircuitPython running. Great! Now what? CircuitPython Essentials!

There are a number of core modules built into CircuitPython and commonly used libraries available. This guide will introduce you to these and show you an example of how to use each one.

Each section will present you with a piece of code designed to work with different boards, and explain how to use the code with each board. These examples work with any board designed for CircuitPython, including Circuit Playground Express, Trinket M0, Gemma M0, QT Py, ItsyBitsy M0 Express, ItsyBitsy M4 Express, Feather M0 Express, Feather M4 Express, Metro M4 Express, Metro M0 Express, Trellis M4 Express, and Grand Central M4 Express.

Some examples require external components, such as switches or sensors. You'll find wiring diagrams where applicable to show you how to wire up the necessary components to work with each example.

Let's get started learning the CircuitPython Essentials!

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. 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 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, 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 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 Kattni Rembor for Adafruit Industries
#
```
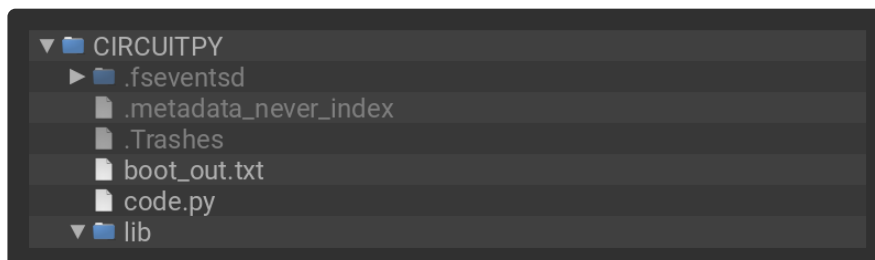
```python
# SPDX-License-Identifier: MIT

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

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("board.{}".format(alias))
        if len(pins) > 0:
            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:

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`. This means that you can access pin A0 with both `board.A0` and `board.D0`.

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 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")
```

```
>>> help("modules")
__main__            collections     neopixel_write   supervisor
_pixelbuf           digitalio       os               sys
adafruit_bus_device                 displayio        pulseio          terminalio
analogio            errno           pwmio            time
array               fontio          random           touchio
audiocore           gamepad         re               usb_hid
audioio             gc              rotaryio         usb_midi
board               math            rtc              vectorio
builtins            microcontroller storage
busio               micropython     struct
Plus any modules on the filesystem
```

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

CircuitPython Built-Ins

CircuitPython comes 'with the kitchen sink' - a lot of the things you know and love about classic Python 3 (sometimes called CPython) already work. There are a few things that don't but we'll try to keep this list updated as we add more capabilities!

> This is not an exhaustive list! It's simply some of the many features you can use.

Thing That Are Built In and Work

Flow Control

All the usual `if`, `elif`, `else`, `for`, `while` work just as expected.

Math

`import math` will give you a range of handy mathematical functions.

```
>>> dir(math)
['__name__', 'e', 'pi', 'sqrt', 'pow', 'exp', 'log', 'cos', 'sin',
'tan', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'copysign', 'fabs',
'floor', 'fmod', 'frexp', 'ldexp', 'modf', 'isfinite', 'isinf',
'isnan', 'trunc', 'radians', 'degrees']
```

CircuitPython supports 30-bit wide floating point values so you can use `int` and `float` whenever you expect.

Tuples, Lists, Arrays, and Dictionaries

You can organize data in `()`, `[]`, and `{}` including strings, objects, floats, etc.

Classes, Objects and Functions

We use objects and functions extensively in our libraries so check out one of our many examples like this [MCP9808 library ()](#) for class examples.

Lambdas

Yep! You can create function-functions with `lambda` just the way you like em:

```
>>> g = lambda x: x**2
>>> g(8)
64
```

Random Numbers

To obtain random numbers:

```
import random
```

`random.random()` will give a floating point number from `0` to `1.0`.

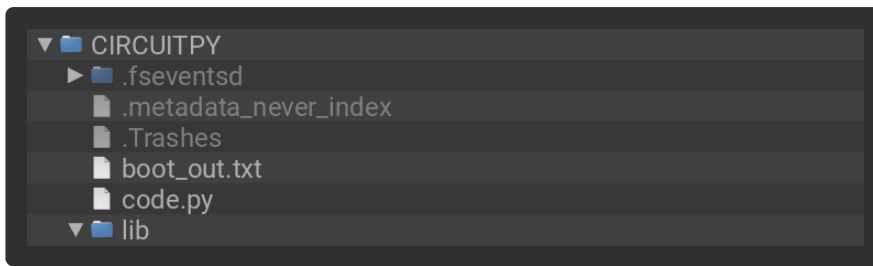`random.randint(min, max)` will give you an integer number between `min` and `max`.

CircuitPython Digital In & Out

The first part of interfacing with hardware is being able to manage digital inputs and outputs. With CircuitPython, it's super easy!

This example shows how to use both a digital input and output. You can use a switch input with pullup resistor (built in) to control a digital output - the built in red LED.

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/CircuitPython_Ditial_In_Out/ 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: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Digital In Out example"""
import time
import board
from digitalio import DigitalInOut, Direction, Pull

# LED setup.
led = DigitalInOut(board.LED)
# For QT Py M0. QT Py M0 does not have a D13 LED, so you can connect an external
LED instead.
# led = DigitalInOut(board.SCK)
led.direction = Direction.OUTPUT

# For Gemma M0, Trinket M0, Metro M0 Express, ItsyBitsy M0 Express, Itsy M4
Express, QT Py M0
switch = DigitalInOut(board.D2)
# switch = DigitalInOut(board.D5)  # For Feather M0 Express, Feather M4 Express
# switch = DigitalInOut(board.D7)  # For Circuit Playground Express
switch.direction = Direction.INPUT
switch.pull = Pull.UP

while True:
    # We could also do "led.value = not switch.value"!
    if switch.value:
        led.value = False
    else:
        led.value = True

    time.sleep(0.01)  # debounce delay
```

Note that we made the code a little less "Pythonic" than necessary. The `if/else` block could be replaced with a simple `led.value = not switch.value` but we wanted to make it super clear how to test the inputs. The interpreter will read the digital input when it evaluates `switch.value`.

For Gemma M0, Trinket M0, Metro M0 Express, Metro M4 Express, ItsyBitsy M0 Express, ItsyBitsy M4 Express, no changes to the initial example are needed.

> Note: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

For Feather M0 Express and Feather M4 Express, comment out `switch = DigitalInOut(board.D2)` (and/or `switch = DigitalInOut(board.D7)`

depending on what changes you already made), and uncomment `switch = DigitalInOut(board.D5)` .

For Circuit Playground Express, you'll need to comment out `switch = DigitalInOut(board.D2)` (and/or `switch = DigitalInOut(board.D5)` depending on what changes you already made), and uncomment `switch = DigitalInOut(board.D7)` .

> QT Py M0 does not have a little red LED built in. Therefore, you must connect an external LED for this example to work. See below for a wiring diagram illustrating how to connect an external LED to a QT Py M0.

For QT Py M0, you'll need to comment out `led = DigitalInOut(board.LED)` and uncomment `led = DigitalInOut(board.SCK)` . The switch code remains the same.

To find the pin or pad suggested in the code, see the list below. For the boards that require wiring, wire up a switch (also known as a tactile switch, button or push-button), following the diagram for guidance. Press or slide the switch, and the onboard red LED will turn on and off.

Note that on the M0/SAMD based CircuitPython boards, at least, you can also have internal pulldowns with Pull.DOWN and if you want to turn off the pullup/pulldown just assign switch.pull = None.

Find the pins!

The list below shows each board, explains the location of the Digital pin suggested for use as input, and the location of the D13 LED.

Circuit Playground Express
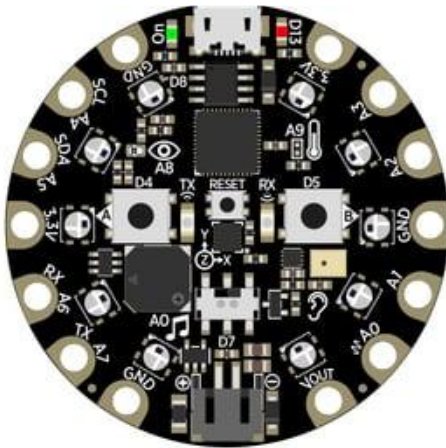
We're going to use the switch, which is pin D7, and is located between the battery connector and the reset switch on the board. The LED is labeled D13 and is located next to the USB micro port.
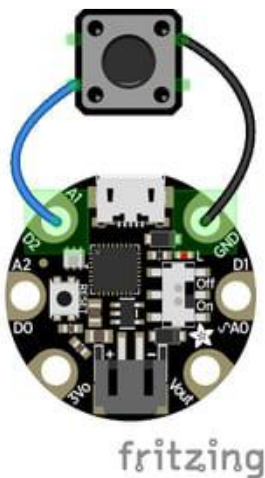
To use D7, comment out the current pin setup line, and uncomment the line labeled for Circuit Playground Express. See the details above!
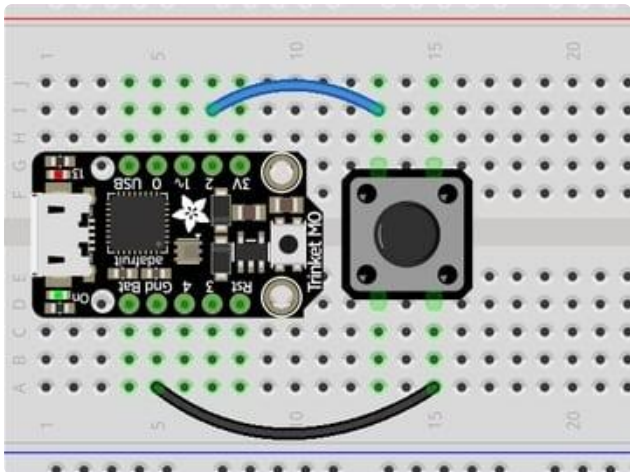
Trinket M0

D2 is connected to the blue wire, labeled "2", and located between "3V" and "1" on the board. The LED is labeled "13" and is located next to the USB micro port.
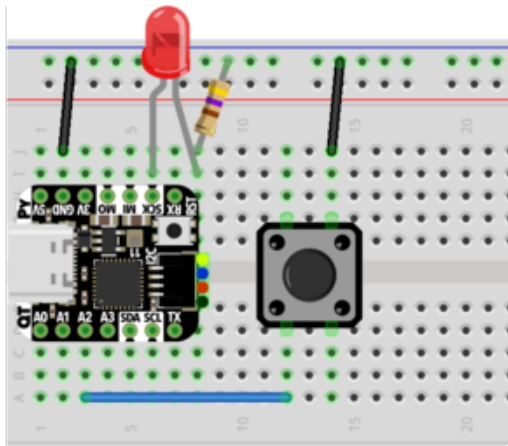
Gemma M0

D2 is an alligator-clip-friendly pad labeled both "D2" and "A1", shown connected to the blue wire, and is next to the USB micro port. The LED is located next to the "GND" label on the board, above the "On/Off" switch.

Use alligator clips to connect your switch to your Gemma M0!

QT Py M0

D2 is labeled A2, shown connected to the blue wire, and is near the USB port between A1 and A3.

There is no little red LED built-in to the QT Py M0. Therefore, you must connect an external LED for this example to work.

To wire up an external LED:

LED + to QT Py SCK
LED - to 470Ω resistor
470Ω resistor to QT Py GND
The button and the LED share the same GND pin.

To use the external LED, comment out the current LED setup line, and uncomment the line labeled for QT Py M0. See the details above!

Feather M0 Express and Feather M4 Express
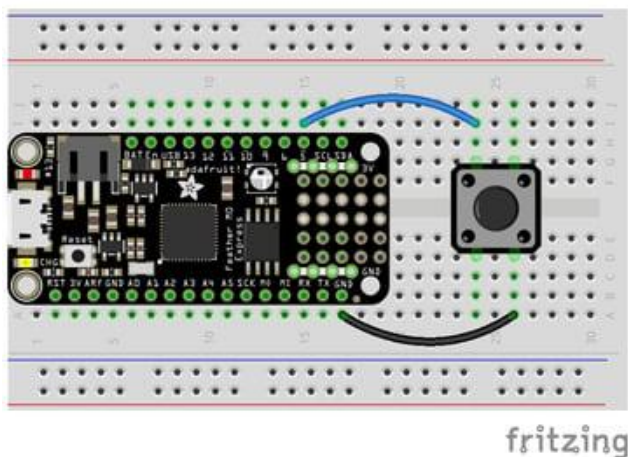
D5 is labeled "5" and connected to the blue wire on the board. The LED is labeled "#13" and is located next to the USB micro port.

To use D5, comment out the current pin setup line, and uncomment the line labeled for Feather M0 Express. See the details above!
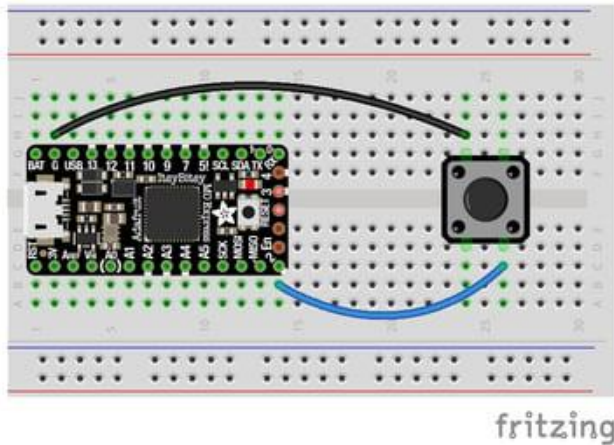
ItsyBitsy M0 Express and ItsyBitsy M4 Express

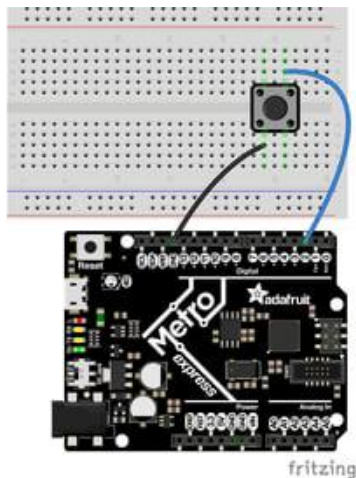D2 is labeled "2", located between the "MISO" and "EN" labels, and is connected to the blue wire on the board. The LED is located next to the reset button between the "3" and "4" labels on the board.

Metro M0 Express and Metro M4 Express

D2 is located near the top left corner, and is connected to the blue wire. The LED is labeled "L" and is located next to the USB micro port.

Read the Docs

For a more in-depth look at what `digitalio` can do, check out the `DigitalInOut` page in Read the Docs ().

CircuitPython Analog In

This example shows you how you can read the analog voltage on the A1 pin on your board.

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/CircuitPython_Analogin/ 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: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Analog In example"""
import time
import board
from analogio import AnalogIn

analog_in = AnalogIn(board.A1)


def get_voltage(pin):
    return (pin.value * 3.3) / 65536


while True:
    print((get_voltage(analog_in),))
    time.sleep(0.1)
```

Make sure you're running the latest CircuitPython! If you are not, you may run into an error: "AttributeError: 'module' object has no attribute 'A1'". If you receive this error, first make sure you're running the latest version of CircuitPython!

Creating the analog input

`analog1in = AnalogIn(board.A1)`

Creates an object and connects the object to A1 as an analog input.

`get_voltage` Helper

`get_voltage(pin)` is our little helper program. By default, analog readings will range from 0 (minimum) to 65535 (maximum). This helper will convert the 0-65535 reading from `pin.value` and convert it a 0-3.3V voltage reading.

Main Loop

The main loop is simple. It `prints` out the voltage as floating point values by calling `get_voltage` on our analog object. Connect to the serial console to see the results.
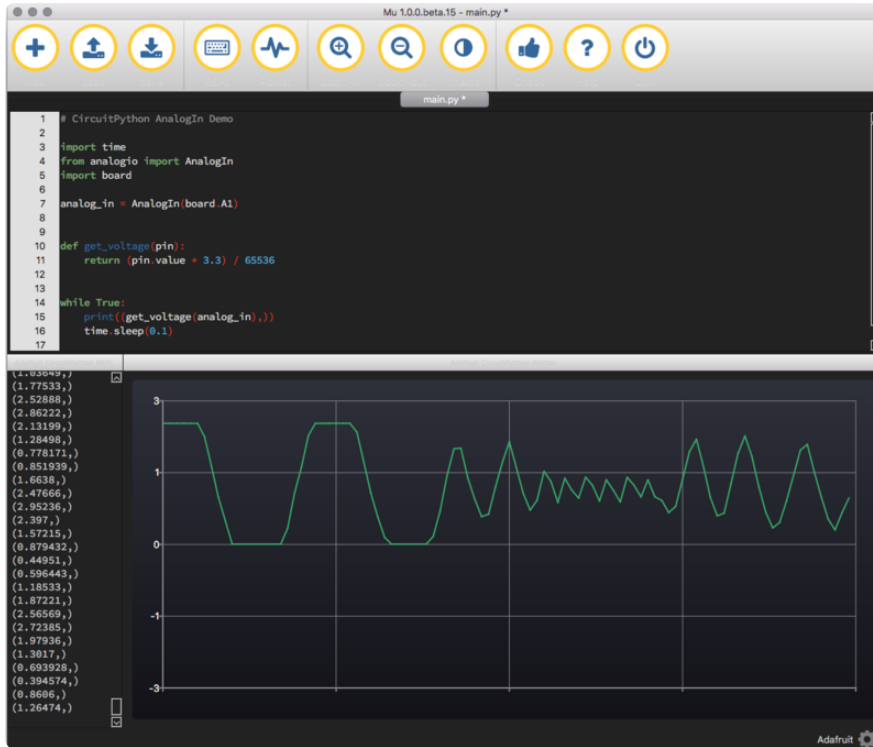
Changing It Up

By default the pins are floating so the voltages will vary. While connected to the serial console, try touching a wire from A1 to the GND pin or 3Vo pin to see the voltage change.

You can also add a potentiometer to control the voltage changes. From the potentiometer to the board, connect the left pin to ground, the middle pin to A1, and the right pin to 3V. If you're using Mu editor, you can see the changes as you rotate the potentiometer on the plotter like in the image above! (Click the Plotter icon at the top of the window to open the plotter.)

> When you turn the knob of the potentiometer, the wiper rotates left and right, increasing or decreasing the resistance. This, in turn, changes the analog voltage level that will be read by your board on A1.

Wire it up

The list below shows wiring diagrams to help find the correct pins and wire up the potentiometer, and provides more information about analog pins on your board!

Circuit Playground Express
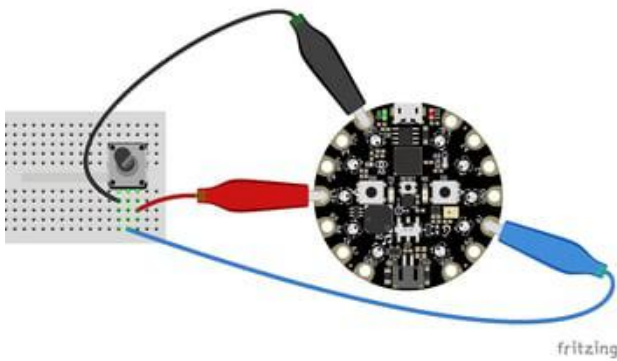
A1 is located on the right side of the board. There are multiple ground and 3V pads (pins).

Your board has 7 analog pins that can be used for this purpose. For the full list, see the pinout page () on the main guide.

Trinket M0
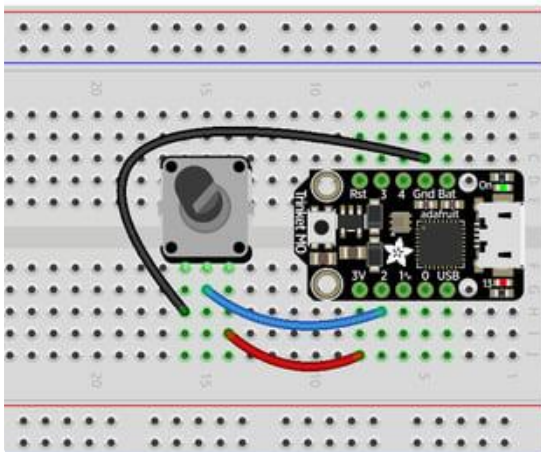
A1 is labeled as 2! It's located between "1~" and "3V" on the same side of the board as the little red LED. Ground is located on the opposite side of the board. 3V is located next to 2, on the same end of the board as the reset button.

You have 5 analog pins you can use. For the full list, see the pinouts page () on the main guide.

Gemma M0
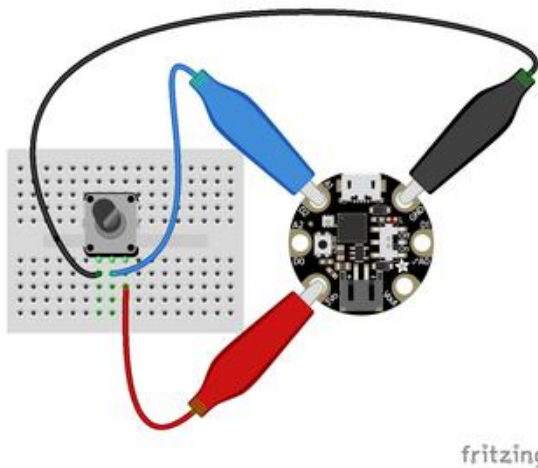
A1 is located near the top of the board of the board to the left side of the USB Micro port. Ground is on the other side of the USB port from A1. 3V is located to the left side of the battery connector on the bottom of the board.

fritzing

Your board has 3 analog pins. For the full list, see the pinout page () on the main guide.

QT Py M0
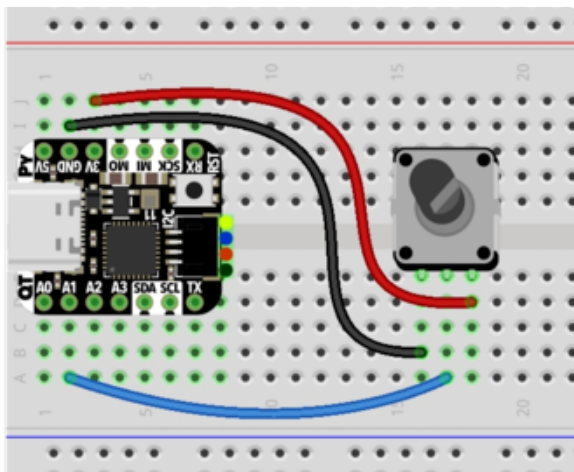
A1, shown connected to the blue wire, is near the USB port between A0 and A2. Ground is on the opposite side of the QT Py, near the USB port, between 3V and 5V. 3V is the next pin, between GND and MO.

Your board has 10 analog pins. For the full list, see the pinouts page () in the main guide.

Feather M0 Express and Feather M4 Express
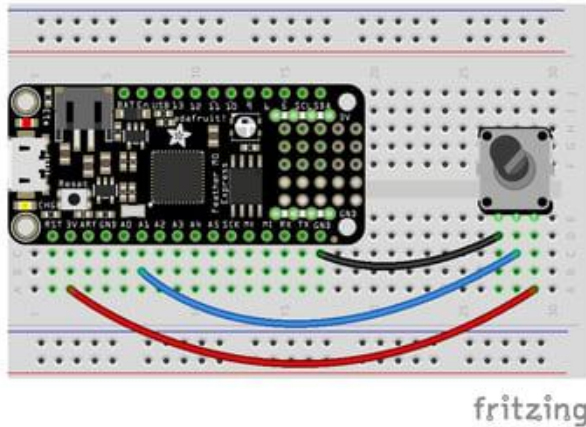
A1 is located along the edge opposite the battery connector. There are multiple ground pins. 3V is located along the same edge as A1, and is next to the reset button.

Your board has 6 analog pins you can use. For the full list, see the pinouts page () on the main guide.

ItsyBitsy M0 Express and ItsyBitsy M4 Express
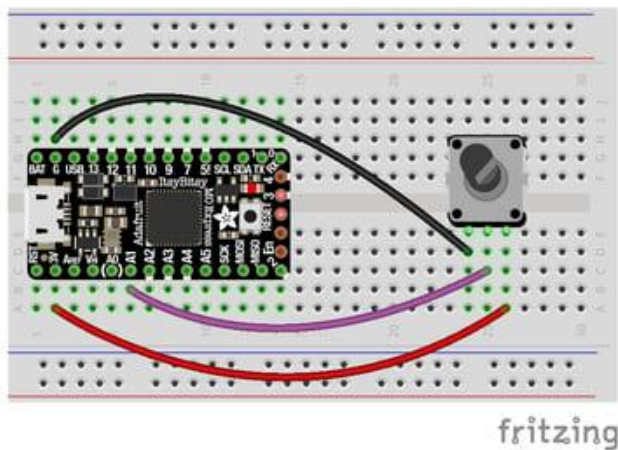
A1 is located in the middle of the board, near the "A" in "Adafruit". Ground is labled "G" and is located next to "BAT", near the USB Micro port. 3V is found on the opposite side of the USB port from Ground, next to RST.

You have 6 analog pins you can use. For a full list, see the pinouts page () on the main guide.
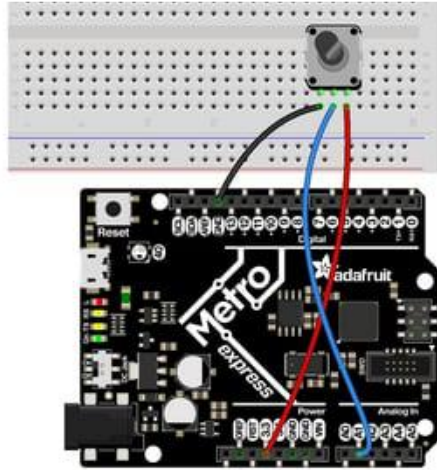
A1 is located on the same side of the board as the barrel jack. There are multiple ground pins available. 3V is labeled "3.3" and is located in the center of the board on the same side as the barrel jack (and as A1).

Your Metro M0 Express board has 6 analog pins you can use. For the full list, see the pinouts page () on the main guide.

Your Metro M4 Express board has 6 analog pins you can use. For the full list, see the pinouts page () on the main guide.

Reading Analog Pin Values

The `get_voltage()` helper used in the potentiometer example above reads the raw analog pin value and converts it to a voltage level. You can, however, directly read an analog pin value in your code by using `pin.value`. For example, to simply read the raw analog pin value from the potentiometer, you would run the following code:

```
import time
import board
from analogio import AnalogIn

analog_in = AnalogIn(board.A1)

while True:
    print(analog_in.value)
    time.sleep(0.1)
```

This works with any analog pin or input. Use the `<pin_name>.value` to read the raw value and utilise it in your code.

CircuitPython Analog Out

This example shows you how you can set the DAC (true analog output) on pin A0.

A0 is the only true analog output on the M0 boards. No other pins do true analog output!

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/CircuitPython_AnalogOut/ 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
  ▼ 🖿 lib
```
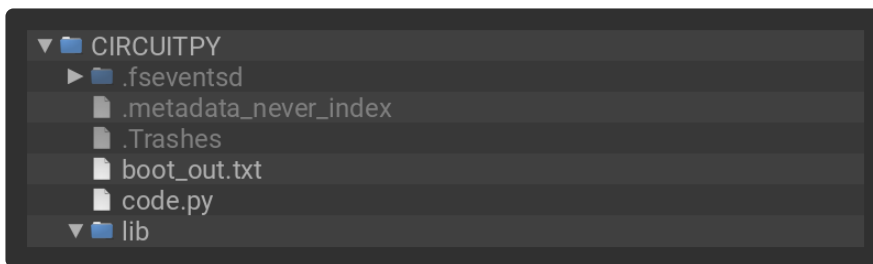
```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Analog Out example"""
import board
from analogio import AnalogOut

analog_out = AnalogOut(board.A0)

while True:
    # Count up from 0 to 65535, with 64 increment
    # which ends up corresponding to the DAC's 10-bit range
    for i in range(0, 65535, 64):
        analog_out.value = i
```

Creating an analog output

```
analog_out = AnalogOut(A0)
```

Creates an object `analog_out` and connects the object to A0, the only DAC pin available on both the M0 and the M4 boards. (The M4 has two, A0 and A1.)

Setting the analog output

The DAC on the SAMD21 is a 10-bit output, from 0-3.3V. So in theory you will have a resolution of 0.0032 Volts per bit. To allow CircuitPython to be general-purpose enough that it can be used with chips with anything from 8 to 16-bit DACs, the DAC takes a 16-bit value and divides it down internally.

For example, writing 0 will be the same as setting it to 0 - 0 Volts out.

Writing 5000 is the same as setting it to 5000 / 64 = 78, and 78 / 1024 * 3.3V = 0.25V output.

Writing 65535 is the same as 1023 which is the top range and you'll get 3.3V output

Main Loop

The main loop is fairly simple, it goes through the entire range of the DAC, from 0 to 65535, but increments 64 at a time so it ends up clicking up one bit for each of the 10-bits of range available.

CircuitPython is not terribly fast, so at the fastest update loop you'll get 4 Hz. The DAC isn't good for audio outputs as-is.

Express boards like the Circuit Playground Express, Metro M0 Express, ItsyBitsy M0 Express, ItsyBitsy M4 Express, Metro M4 Express, Feather M4 Express, or Feather M0 Express have more code space and can perform audio playback capabilities via the DAC. QT Py M0, Gemma M0 and Trinket M0 cannot!

Check out the Audio Out section of this guide () for examples!
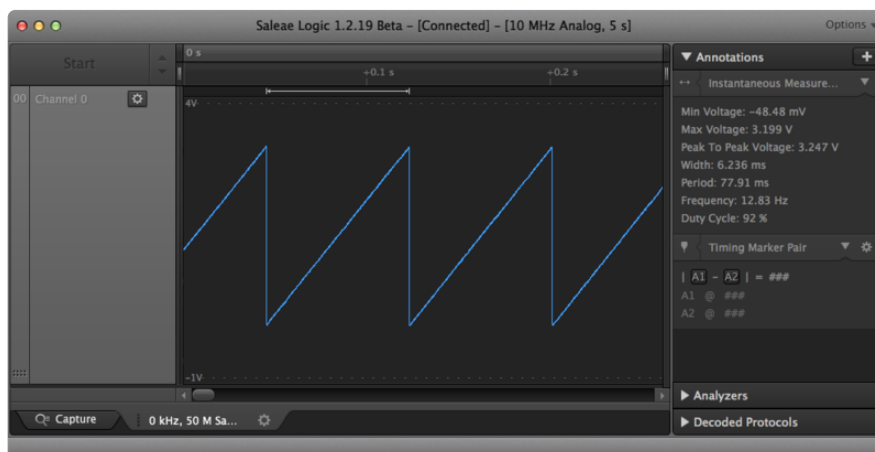
Find the pin

Use the diagrams below to find the A0 pin marked with a magenta arrow!
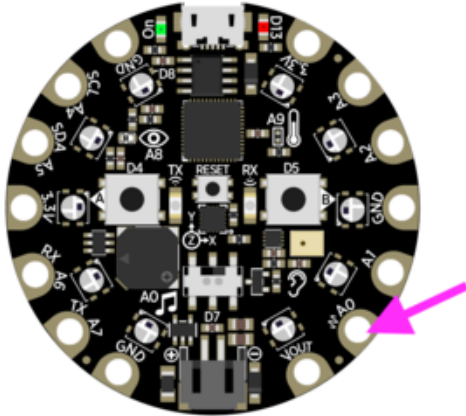
Circuit Playground Express

A0 is located between VOUT and A1 near the battery port.
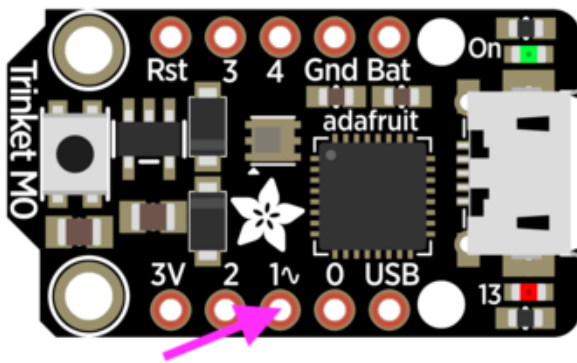
Trinket M0

A0 is labeled "1~" on Trinket! A0 is located between "0" and "2" towards the middle of the board on the same side as the red LED.
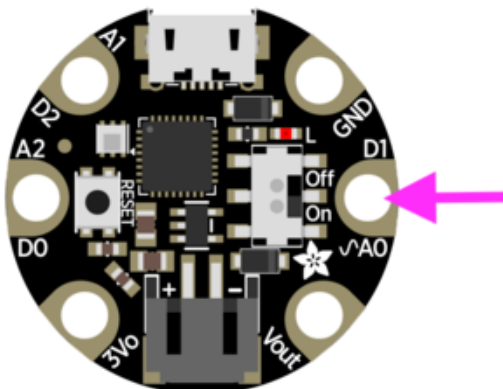
Gemma M0

A0 is located in the middle of the right side of the board next to the On/Off switch.
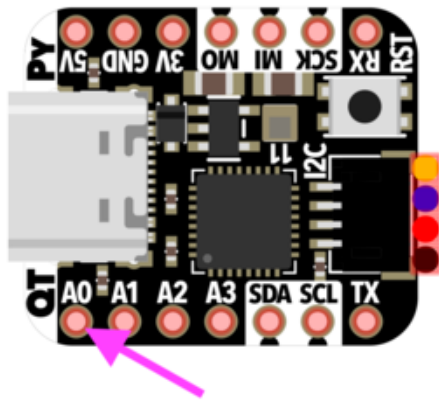
QT Py M0

A0 is located next to the USB port, by the "QT" label on the board silk.
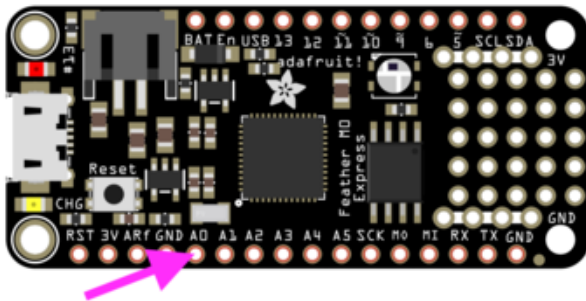
Feather M0 Express

A0 is located between GND and A1 on the opposite side of the board from the battery connector, towards the end with the Reset button.
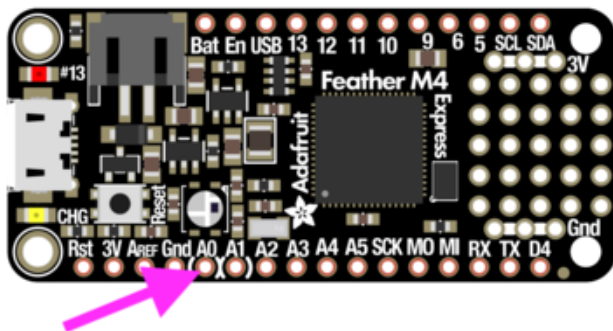
Feather M4 Express

A0 is located between GND and A1 on the opposite side of the board from the battery connector, towards the end with the Reset button, and the pin pad has left and right white parenthesis markings around it
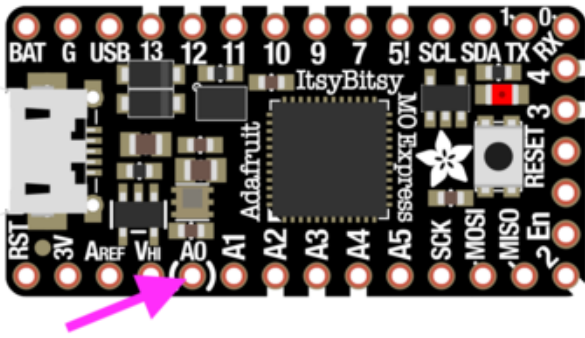
ItsyBitsy M0 Express

A0 is located between VHI and A1, near the "A" in "Adafruit", and the pin pad has left and right white parenthesis markings around it.
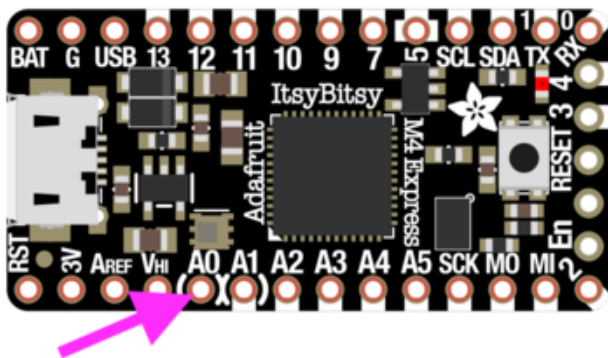
ItsyBitsy M4 Express

A0 is located between VHI and A1, and the pin pad has left and right white parenthesis markings around it.
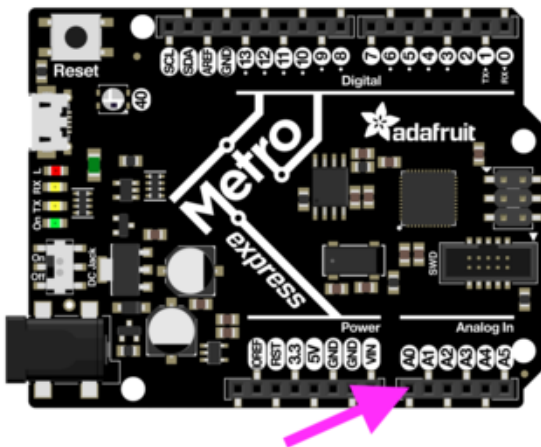
Metro M0 Express

A0 is between VIN and A1, and is located along the same side of the board as the barrel jack adapter towards the middle of the headers found on that side of the board.

Metro M4 Express
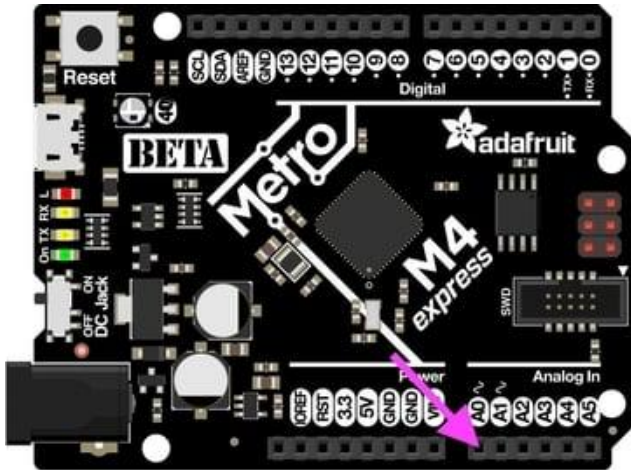
A0 is between VIN and A1, and is located along the same side of the board as the barrel jack adapter towards the middle of the headers found on that side of the board.

On the Metro M4 Express, there are TWO true analog outputs: A0 and A1.

CircuitPython PWM

Your board has `pwmio` support, which means you can PWM LEDs, control servos, beep piezos, and manage "pulse train" type devices like DHT22 and Infrared.

Nearly every pin has PWM support! For example, all ATSAMD21 board have an A0 pin which is 'true' analog out and does not have PWM support.

PWM with Fixed Frequency

This example will show you how to use PWM to fade the little red LED on your board.

> The QT Py M0 does not have a little red LED. Therefore, you must connect an external LED and edit this example for it to work. Follow the wiring diagram and steps below to run this example on QT Py M0.

The following illustrates how to connect an external LED to a QT Py M0.
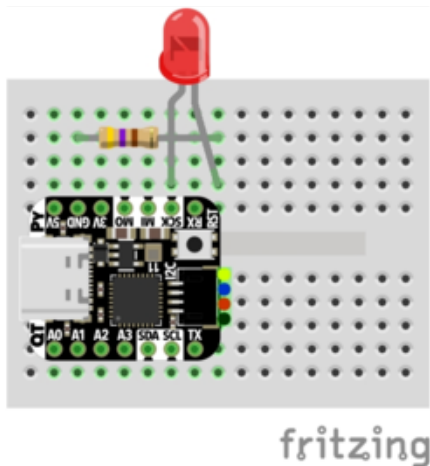
LED + to QT Py SCK

LED - to 470Ω resistor

470Ω resistor to QT Py GND

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/CircuitPython_PWM/ 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:
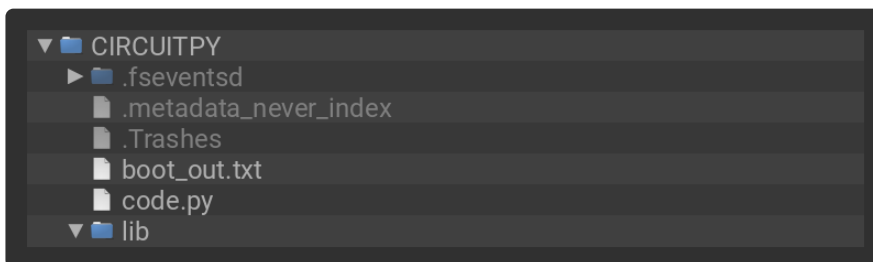

```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials: PWM with Fixed Frequency example."""
import time
import board
import pwmio

# LED setup for most CircuitPython boards:
led = pwmio.PWMOut(board.LED, frequency=5000, duty_cycle=0)
# LED setup for QT Py M0:
# led = pwmio.PWMOut(board.SCK, frequency=5000, duty_cycle=0)

while True:
    for i in range(100):
        # PWM LED up and down
        if i < 50:
            led.duty_cycle = int(i * 2 * 65535 / 100)  # Up
        else:
            led.duty_cycle = 65535 - int((i - 50) * 2 * 65535 / 100)  # Down
        time.sleep(0.01)
```

> Remember: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

To use with QT Py M0, you must comment out `led = pwmio.PWMOut(board.LED, frequency=5000, duty_cycle=0)` and uncomment `led = pwmio.PWMOut(board.SCK, frequency=5000, duty_cycle=0)`. Your setup lines should look like this for the example to work with QT Py M0:

```
# LED setup for most CircuitPython boards:
# led = pwmio.PWMOut(board.LED, frequency=5000, duty_cycle=0)
# LED setup for QT Py M0:
led = pwmio.PWMOut(board.SCK, frequency=5000, duty_cycle=0)
```

Create a PWM Output

`led = pwmio.PWMOut(board.LED, frequency=5000, duty_cycle=0)`

Since we're using the onboard LED, we'll call the object `led`, use `pwmio.PWMOut` to create the output and pass in the `D13` LED pin to use.

Main Loop

The main loop uses `range()` to cycle through the loop. When the range is below 50, it PWMs the LED brightness up, and when the range is above 50, it PWMs the brightness down. This is how it fades the LED brighter and dimmer!

The `time.sleep()` is needed to allow the PWM process to occur over a period of time. Otherwise it happens too quickly for you to see!

PWM Output with Variable Frequency

Fixed frequency outputs are great for pulsing LEDs or controlling servos. But if you want to make some beeps with a piezo, you'll need to vary the frequency.

The following example uses `pwmio` to make a series of tones on a piezo.

To use with any of the M0 boards, no changes to the following code are needed.

> Remember: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

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/CircuitPython_PWM_Piezo/ 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:
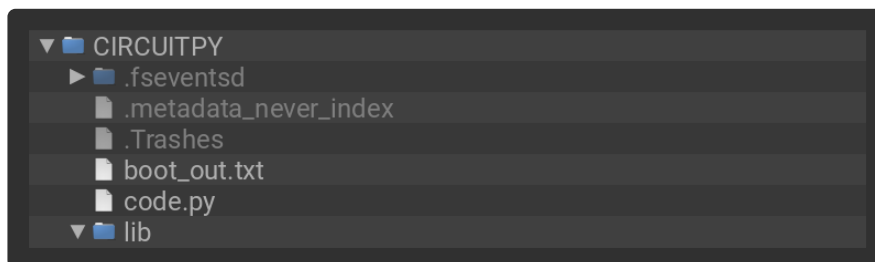
To use with the Metro M4 Express, ItsyBitsy M4 Express or the Feather M4 Express, you must comment out the `piezo = pwmio.PWMOut(board.A2, duty_cycle=0, frequency=440, variable_frequency=True)` line and uncomment the `piezo = pwmio.PWMOut(board.A1, duty_cycle=0, frequency=440, variable_frequency=True)` line. A2 is not a supported PWM pin on the M4 boards!

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials PWM with variable frequency piezo example"""
import time
import board
import pwmio

# For the M0 boards:
piezo = pwmio.PWMOut(board.A2, duty_cycle=0, frequency=440, variable_frequency=True)

# For the M4 boards:
# piezo = pwmio.PWMOut(board.A1, duty_cycle=0, frequency=440,
variable_frequency=True)

while True:
    for f in (262, 294, 330, 349, 392, 440, 494, 523):
        piezo.frequency = f
        piezo.duty_cycle = 65535 // 2  # On 50%
        time.sleep(0.25)  # On for 1/4 second
        piezo.duty_cycle = 0  # Off
        time.sleep(0.05)  # Pause between notes
    time.sleep(0.5)
```

The following example uses a nice little helper in the `simpleio` library that makes a tone for you on a piezo with a single command.

To use with any of the M0 boards, no changes to the following code are needed.

To use with the Metro M4 Express, ItsyBitsy M4 Express or the Feather M4 Express, you must comment out the `simpleio.tone(board.A2, f, 0.25)` line and uncomment the `simpleio.tone(board.A1, f, 0.25)` line. A2 is not a supported PWM pin on the M4 boards!

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 CircuitPython_Essentials/CircuitPython_PWM_Piezo_simpleio/ 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:


```python
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials PWM piezo simpleio example"""
import time
import board
import simpleio

while True:
    for f in (262, 294, 330, 349, 392, 440, 494, 523):
        # For the M0 boards:
        simpleio.tone(board.A2, f, 0.25)  # on for 1/4 second
        # For the M4 boards:
        # simpleio.tone(board.A1, f, 0.25)  # on for 1/4 second
```

```
        time.sleep(0.05)  # pause between notes
    time.sleep(0.5)
```

As you can see, it's much simpler!

Wire it up

Use the diagrams below to help you wire up your piezo. Attach one leg of the piezo to pin A2 on the M0 boards or A1 on the M4 boards, and the other leg to ground. It doesn't matter which leg is connected to which pin. They're interchangeable!

Circuit Playground Express

Use alligator clips to attach A2 and any one of the GND to different legs of the piezo.
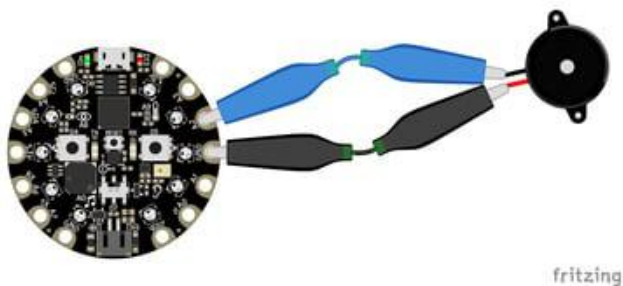
CPX has PWM on the following pins: A1, A2, A3, A6, RX, LIGHT, A8, TEMPERATURE, A9, BUTTON_B, D5, SLIDE_SWITCH, D7, D13, REMOTEIN, IR_RX, REMOTEOUT, IR_TX, IR_PROXIMITY, MICROPHONE_CLOCK, MICROPHONE_DATA, ACCELEROMETER_INTERRUPT, ACCELEROMETER_SDA, ACCELEROMETER_SCL, SPEAKER_ENABLE.

There is NO PWM on: A0, SPEAKER, A4, SCL, A5, SDA, A7, TX, BUTTON_A, D4, NEOPIXEL, D8, SCK, MOSI, MISO, FLASH_CS.

Trinket M0

Note: A2 on Trinket is also labeled Digital "0"!
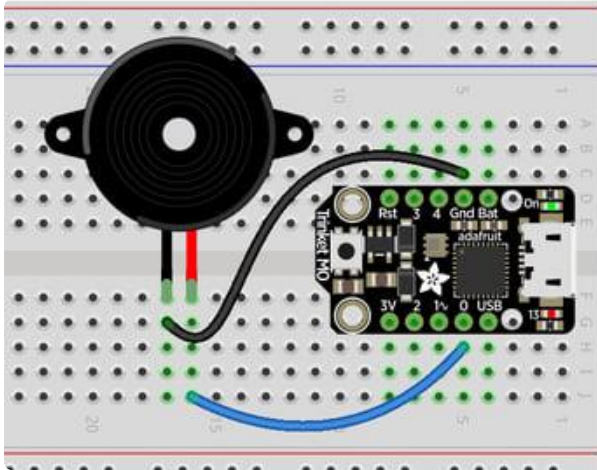
Use jumper wires to connect GND and D0 to different legs of the piezo.

Trinket has PWM available on the following pins: D0, A2, SDA, D2, A1, SCL, MISO, D4, A4, TX, MOSI, D3, A3, RX, SCK, D13, APA102_MOSI, APA102_SCK.

There is NO PWM on: A0, D1.

Gemma M0

Use alligator clips to attach A2 and GND to different legs on the piezo.
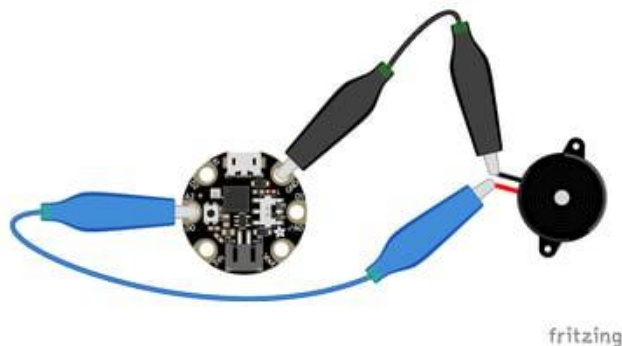
Gemma has PWM available on the following pins: A1, D2, RX, SCL, A2, D0, TX, SDA, L, D13, APA102_MOSI, APA102_SCK.

There is NO PWM on: A0, D1.

QT Py M0
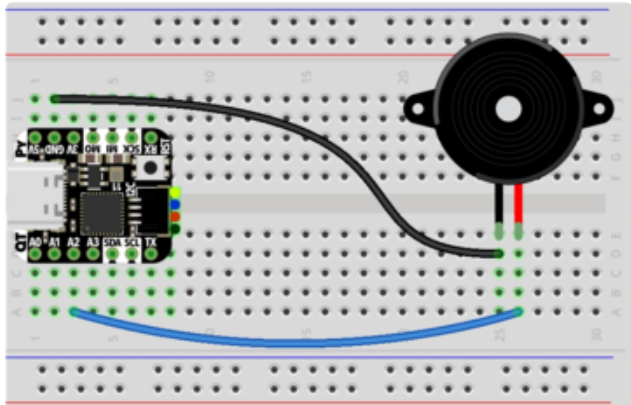
Use jumper wires to attach A2 and GND to different legs of the piezo.

The QT Py M0 has PWM on the following pins: A2, A3, A6, A7, A8, A9, A10, D2, D3, D4, D5, D6, D7, D8, D9, D10, SCK, MISO, MOSI, NEOPIXEL, RX, TX, SCL, SDA.

There is NO A0, A1, D0, D1, NEOPIXEL_POWER.

Feather M0 Express
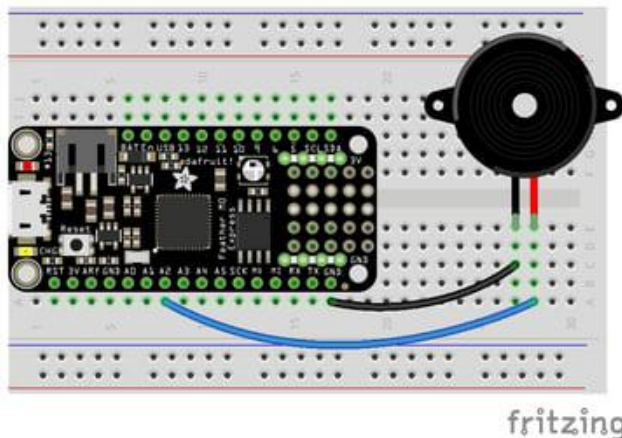
Use jumper wires to attach A2 and one of the two GND to different legs of the piezo.

Feather M0 Express has PWM on the following pins: A2, A3, A4, SCK, MOSI, MISO, D0, RX, D1, TX, SDA, SCL, D5, D6, D9, D10, D11, D12, D13, NEOPIXEL.

There is NO PWM on: A0, A1, A5.

Feather M4 Express

Use jumper wires to attach A1 and one of the two GND to different legs of the piezo.
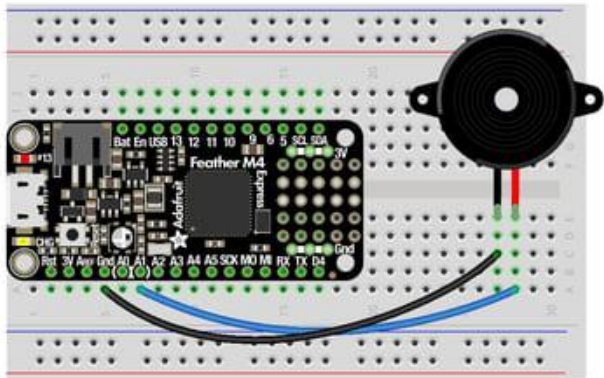
To use A1, comment out the current pin setup line, and uncomment the line labeled for the M4 boards. See the details above!

Feather M4 Express has PWM on the following pins: A1, A3, SCK, D0, RX, D1, TX, SDA, SCL, D4, D5, D6, D9, D10, D11, D12, D13.

There is NO PWM on: A0, A2, A4, A5, MOSI, MISO.

ItsyBitsy M0 Express

Use jumper wires to attach A2 and G to different legs of the piezo.
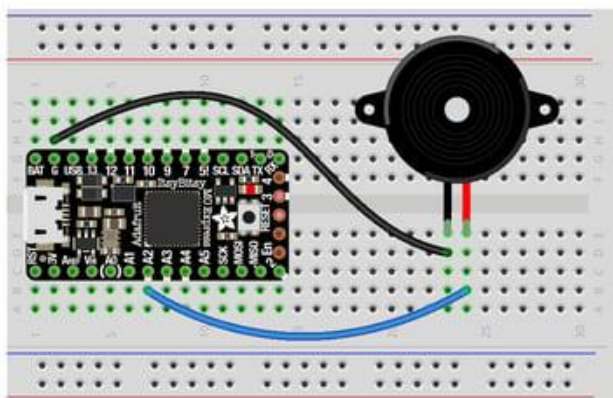
ItsyBitsy M0 Express has PWM on the following pins: D0, RX, D1, TX, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, L, A2, A3, A4, MOSI, MISO, SCK, SCL, SDA, APA102_MOSI, APA102_SCK.

There is NO PWM on: A0, A1, A5.

ItsyBitsy M4 Express

Use jumper wires to attach A1 and G to different legs of the piezo.
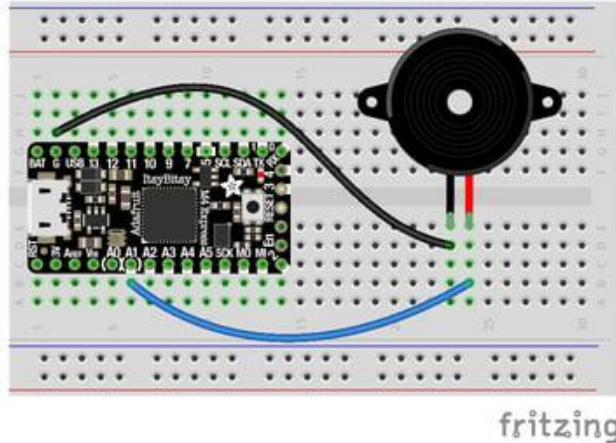
To use A1, comment out the current pin setup line, and uncomment the line labeled for the M4 boards. See the details above!

ItsyBitsy M4 Express has PWM on the following pins: A1, D0, RX, D1, TX, D2, D4, D5, D7, D9, D10, D11, D12, D13, SDA, SCL.

There is NO PWM on: A2, A3, A4, A5, D3, SCK, MOSI, MISO.

Metro M0 Express

Use jumper wires to connect A2 and any one of the GND to different legs on the piezo.
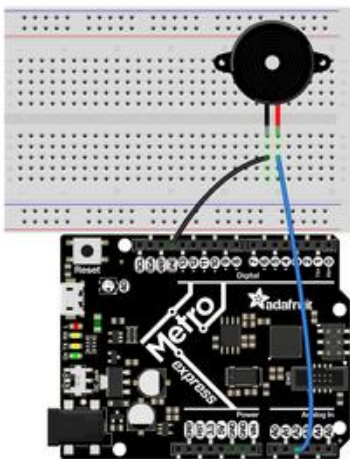
Metro M0 Express has PWM on the following pins: A2, A3, A4, D0, RX, D1, TX, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, SDA, SCL, NEOPIXEL, SCK, MOSI, MISO.

There is NO PWM on: A0, A1, A5, FLASH_CS.

Metro M4 Express

Use jumper wires to connect A1 and any one of the GND to different legs on the piezo.
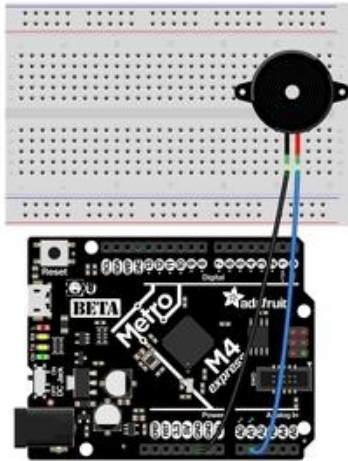
To use A1, comment out the current pin setup line, and uncomment the line labeled for the M4 boards. See the details above!

Metro M4 Express has PWM on: A1, A5, D0, RX, D1, TX, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, SDA, SCK, MOSI, MISO

There is No PWM on: A0, A2, A3, A4, SCL, AREF, NEOPIXEL, LED_RX, LED_TX.

Where's My PWM?

Want to check to see which pins have PWM yourself? We've written this handy script! It attempts to setup PWM on every pin available, and lets you know which ones work and which ones don't. Check it out!

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/PWM_Test_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: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials PWM pin identifying script"""
import board
import pwmio

for pin_name in dir(board):
    pin = getattr(board, pin_name)
    try:
        p = pwmio.PWMOut(pin)
        p.deinit()
        print("PWM on:", pin_name)  # Prints the valid, PWM-capable pins!
    except ValueError:  # This is the error returned when the pin is invalid.
        print("No PWM on:", pin_name)  # Prints the invalid pins.
    except RuntimeError:  # Timer conflict error.
        print("Timers in use:", pin_name)  # Prints the timer conflict pins.
    except TypeError:  # Error returned when checking a non-pin object in
dir(board).
        pass  # Passes over non-pin objects in dir(board).
```

CircuitPython Servo

In order to use servos, we take advantage of `pwmio` . Now, in theory, you could just use the raw `pwmio` calls to set the frequency to 50 Hz and then set the pulse widths. But we would rather make it a little more elegant and easy!

So, instead we will use `adafruit_motor` which manages servos for you quite nicely! `adafruit_motor` is a library so be sure to grab it from the library bundle if you have not yet ()! If you need help installing the library, check out the CircuitPython Libraries page ().

Servos come in two types:

- A standard hobby servo - the horn moves 180 degrees (90 degrees in each direction from zero degrees).
- A continuous servo - the horn moves in full rotation like a DC motor. Instead of an angle specified, you set a throttle value with 1.0 being full forward, 0.5 being half forward, 0 being stopped, and -1 being full reverse, with other values between.

Servo Wiring

> Servos will only work on PWM-capable pins! Check your board details to verify which pins have PWM outputs.

The connections for a servo are the same for standard servos and continuous rotation servos.

Connect the servo's brown or black ground wire to ground on the CircuitPython board.

Connect the servo's red power wire to 5V power, USB power is good for a servo or two. For more than that, you'll need an external battery pack. Do not use 3.3V for powering a servo!

Connect the servo's yellow or white signal wire to the control/data pin, in this case A1 or A2 but you can use any PWM-capable pin.
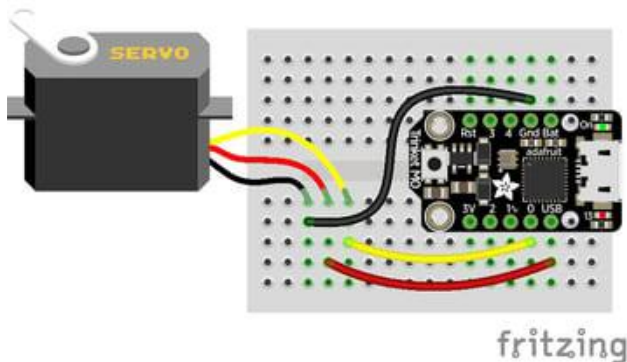
For example, to wire a servo to Trinket, connect the ground wire to GND, the power wire to USB, and the signal wire to 0.

Remember, A2 on Trinket is labeled "0".
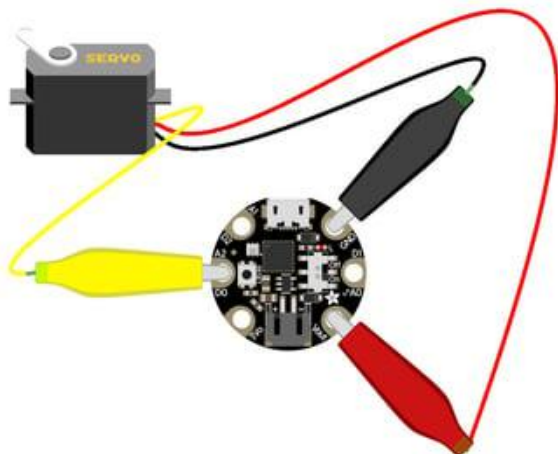
For Gemma, use jumper wire alligator clips to connect the ground wire to GND, the power wire to VOUT, and the signal wire to A2.
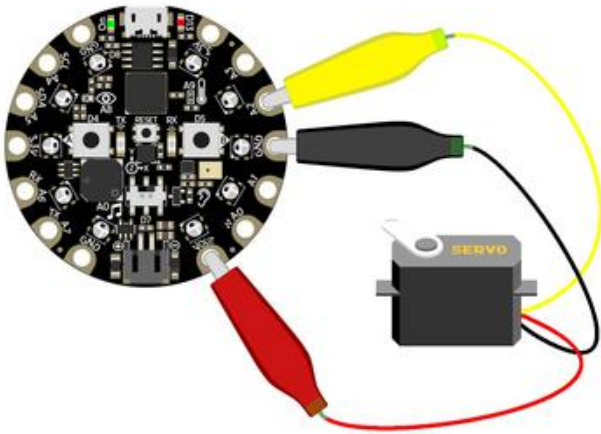
For Circuit Playground Express and Circuit Playground Bluefruit, use jumper wire alligator clips to connect the ground wire to GND, the power wire to VOUT, and the signal wire to A2.
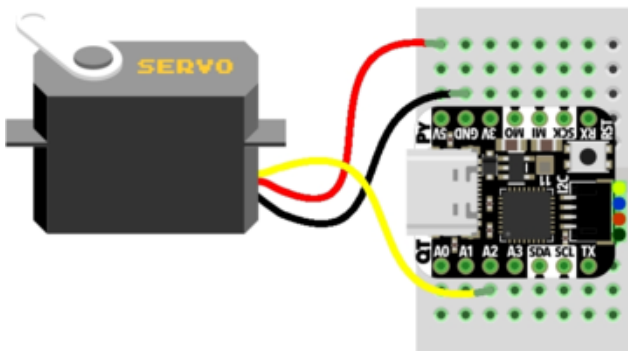
For QT Py M0, connect the ground wire to GND, the power wire to 5V, and the signal wire to A2.
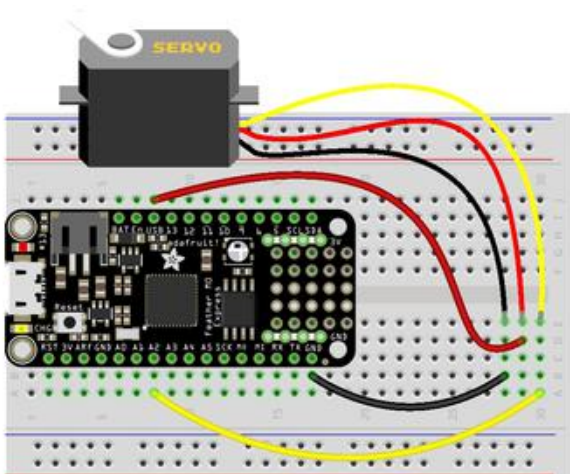
For boards like Feather M0 Express, ItsyBitsy M0 Express and Metro M0 Express, connect the ground wire to any GND, the power wire to USB or 5V, and the signal wire to A2.
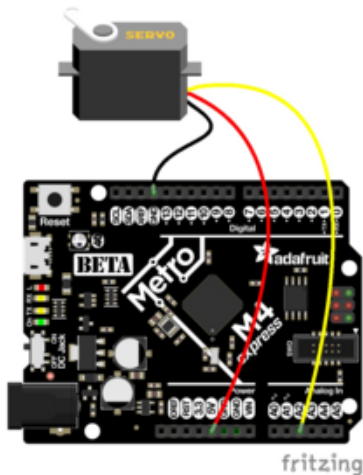
For the Metro M4 Express, ItsyBitsy M4 Express and the Feather M4 Express, connect the ground wire to any G or GND, the power wire to USB or 5V, and the signal wire to A2.

Standard Servo Code

Here's an example that will sweep a servo connected to pin A2 from 0 degrees to 180 degrees (-90 to 90 degrees) and back.

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 CircuitPython_Essentials/CircuitPython_Servo/ 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:
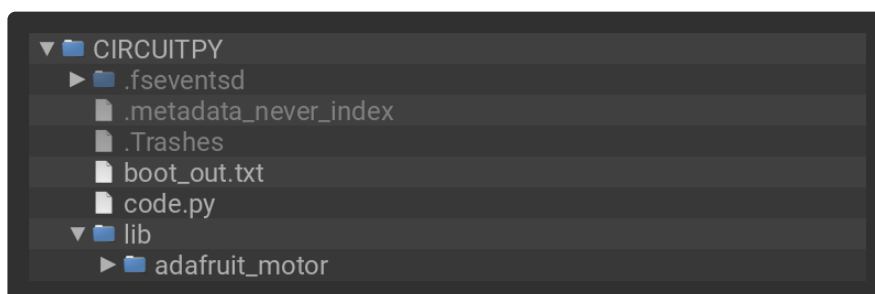

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Servo standard servo example"""
import time
import board
import pwmio
from adafruit_motor import servo
```

```
# create a PWMOut object on Pin A2.
pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)

# Create a servo object, my_servo.
my_servo = servo.Servo(pwm)

while True:
    for angle in range(0, 180, 5):  # 0 - 180 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
    for angle in range(180, 0, -5): # 180 - 0 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
```

Continuous Servo Code

There are two differences with Continuous Servos vs. Standard Servos:

1. The `servo` object is created like `my_servo = servo.ContinuousServo(pwm)` instead of `my_servo = servo.Servo(pwm)`
2. Instead of using `myservo.angle`, you use `my_servo.throttle` using a throttle value from 1.0 (full on) to 0.0 (stopped) to -1.0 (full reverse). Any number between would be a partial speed forward (positive) or reverse (negative). This is very similar to standard DC motor control with the adafruit_motor library.

This example runs full forward for 2 seconds, stops for 2 seconds, runs full reverse for 2 seconds, then stops for 4 seconds.

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 CircuitPython_Essentials/CircuitPython_Continuous_Servo/ 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
    ▼ ■ lib
        ► ■ adafruit_motor
```
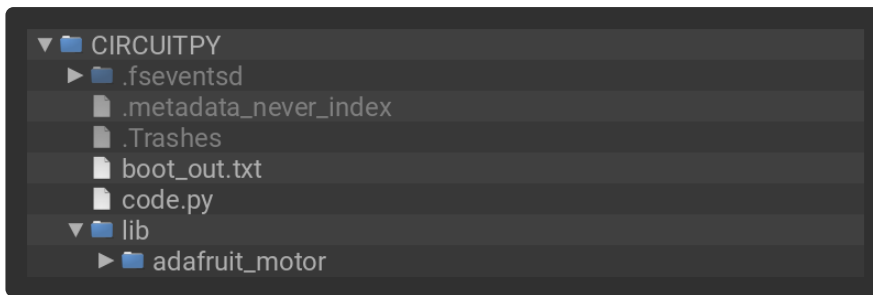
```python
# SPDX-FileCopyrightText: 2019 Anne Barela for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Servo continuous rotation servo example"""
import time
import board
import pwmio
from adafruit_motor import servo

# create a PWMOut object on Pin A2.
pwm = pwmio.PWMOut(board.A2, frequency=50)

# Create a servo object, my_servo.
my_servo = servo.ContinuousServo(pwm)

while True:
    print("forward")
    my_servo.throttle = 1.0
    time.sleep(2.0)
    print("stop")
    my_servo.throttle = 0.0
    time.sleep(2.0)
    print("reverse")
    my_servo.throttle = -1.0
    time.sleep(2.0)
    print("stop")
    my_servo.throttle = 0.0
    time.sleep(4.0)
```

Pretty simple!

Note that we assume that 0 degrees is 0.5ms and 180 degrees is a pulse width of 2.5ms. That's a bit wider than the official 1-2ms pulse widths. If you have a servo that has a different range you can initialize the `servo` object with a different `min_pulse` and `max_pulse`. For example:

```python
my_servo = servo.Servo(pwm, min_pulse = 500, max_pulse = 2500)
```

For more detailed information on using servos with CircuitPython, check out the [Circui tPython section of the servo guide ()]!

CircuitPython Internal RGB LED

Every board has a built in RGB LED. You can use CircuitPython to control the color and brightness of this LED. There are two different types of internal RGB LEDs: DotStar () and NeoPixel (). This section covers both and explains which boards have which LED.
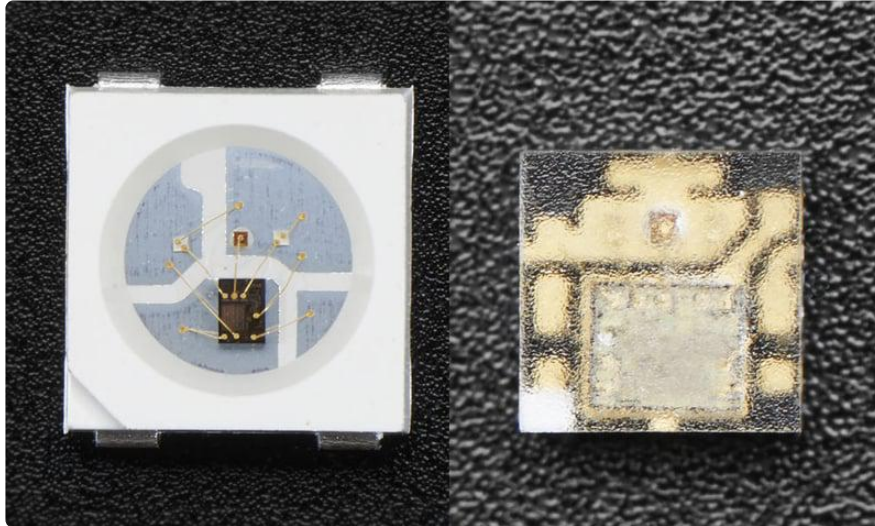
The first example will show you how to change the color and brightness of the internal RGB LED.

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 CircuitPython_Essentials/CircuitPython_Internal_RGB_LED_colors/ 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:
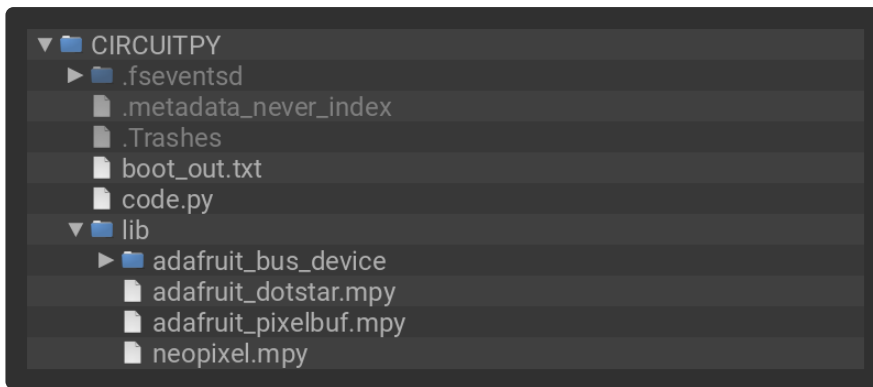
```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Internal RGB LED red, green, blue example"""
import time
import board

if hasattr(board, "APA102_SCK"):
    import adafruit_dotstar

    led = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
else:
    import neopixel

    led = neopixel.NeoPixel(board.NEOPIXEL, 1)

led.brightness = 0.3

while True:
    led[0] = (255, 0, 0)
    time.sleep(0.5)
    led[0] = (0, 255, 0)
    time.sleep(0.5)
    led[0] = (0, 0, 255)
    time.sleep(0.5)
```

Create the LED

First, we create the LED object and attach it to the correct pin or pins. In the case of a NeoPixel, there is only one pin necessary, and we have called it `NEOPIXEL` for easier use. In the case of a DotStar, however, there are two pins necessary, and so we use the pin names `APA102_MOSI` and `APA102_SCK` to get it set up. Since we're using the single onboard LED, the last thing we do is tell it that there's only `1` LED!

Trinket M0, Gemma M0, ItsyBitsy M0 Express, and ItsyBitsy M4 Express each have an onboard Dotstar LED, so no changes are needed to the initial version of the example.

> Remember: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

QT Py M0, Feather M0 Express, Feather M4 Express, Metro M0 Express, Metro M4 Express, and Circuit Playground Express each have an onboard NeoPixel LED, so you must comment out `import adafruit_dotstar` and `led = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)`, and uncomment `import neopixel` and `led = neopixel.NeoPixel(board.NEOPIXEL, 1)`.

Brightness

To set the brightness you simply use the `brightness` attribute. Brightness is set with a number between `0` and `1`, representative of a percent from 0% to 100%. So, `led.brightness = (0.3)` sets the LED brightness to 30%. The default brightness is `1` or 100%, and at it's maximum, the LED is blindingly bright! You can set it lower if you choose.

Main Loop

LED colors are set using a combination of red, green, and blue, in the form of an (R, G, B) tuple. Each member of the tuple is set to a number between 0 and 255 that determines the amount of each color present. Red, green and blue in different combinations can create all the colors in the rainbow! So, for example, to set the LED to red, the tuple would be (255, 0, 0), which has the maximum level of red, and no green or blue. Green would be (0, 255, 0), etc. For the colors between, you set a combination, such as cyan which is (0, 255, 255), with equal amounts of green and blue.

The main loop is quite simple. It sets the first LED to red using `(255, 0, 0)`, then green using `(0, 255, 0)`, and finally blue using `(0, 0, 255)`. Next, we give it a `time.sleep()` so it stays each color for a period of time. We chose `time.sleep(0.5)`, or half a second. Without the `time.sleep()` it'll flash really quickly and the colors will be difficult to see!

Note that we set `led[0]`. This means the first, and in the case of most of the boards, the only LED. In CircuitPython, counting starts at 0. So the first of any object, list, etc will be `0`!
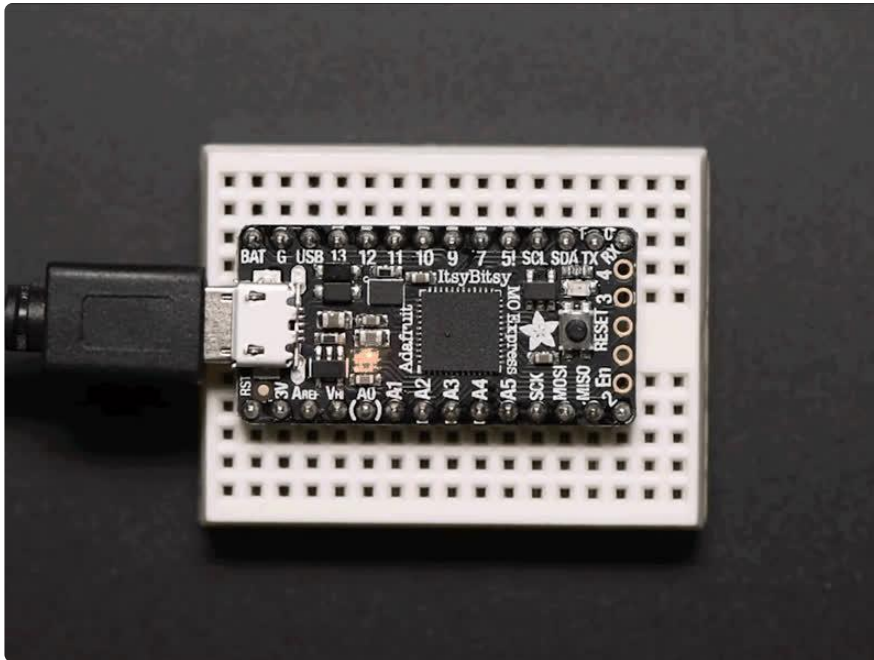
Try changing the numbers in the tuples to change your LED to any color of the rainbow. Or, you can add more lines with different color tuples to add more colors to the sequence. Always add the `time.sleep()`, but try changing the amount of time to create different cycle animations!

Making Rainbows (Because Who Doesn't Love 'Em!)
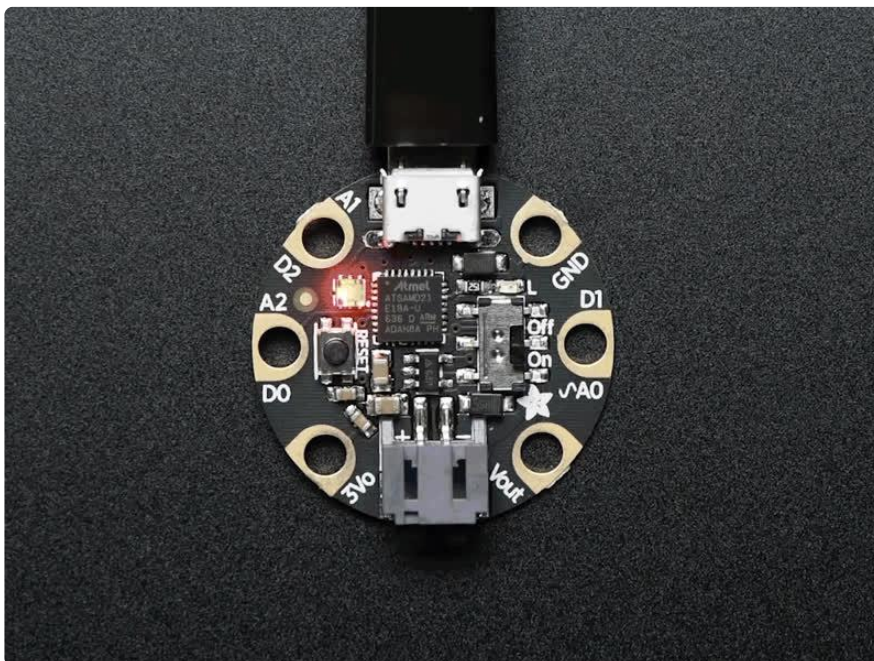
Coding a rainbow effect involves a little math and a helper function called `colorwheel`. For details about how wheel works, see [this explanation here ()](#)!

The last example shows how to do a rainbow animation on the internal RGB LED.

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 CircuitPython_Esse ntials/CircuitPython_Internal_RGB_LED_rainbow/ 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:
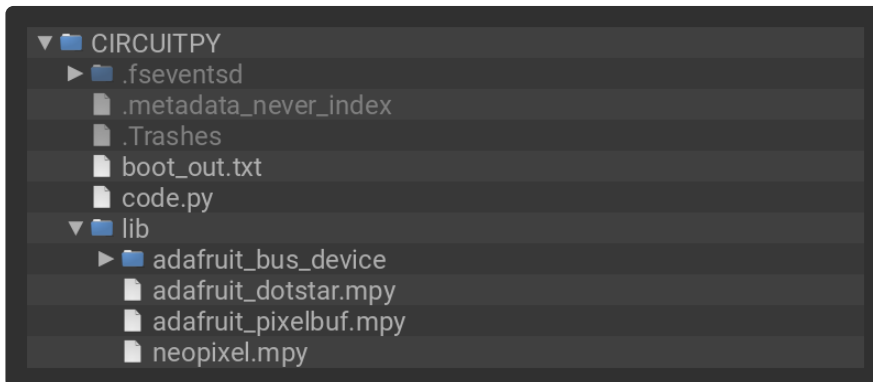

```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Internal RGB LED rainbow example"""
import time
import board
from rainbowio import colorwheel

if hasattr(board, "APA102_SCK"):
    import adafruit_dotstar

    led = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
else:
    import neopixel

    led = neopixel.NeoPixel(board.NEOPIXEL, 1)

led.brightness = 0.3

i = 0
while True:
    i = (i + 1) % 256  # run from 0 to 255
    led.fill(colorwheel(i))
    time.sleep(0.01)
```

We add the `colorwheel` function in after setup but before our main loop.

And right before our main loop, we assign the variable `i = 0`, so it's ready for use inside the loop.

The main loop contains some math that cycles `i` from `0` to `255` and around again repeatedly. We use this value to cycle `colorwheel()` through the rainbow!

The `time.sleep()` determines the speed at which the rainbow changes. Try a higher number for a slower rainbow or a lower number for a faster one!

Circuit Playground Express Rainbow

Note that here we use `led.fill` instead of `led[0]`. This means it turns on all the LEDs, which in the current code is only one. So why bother with `fill`? Well, you may have a Circuit Playground Express, which as you can see has TEN NeoPixel LEDs built in. The examples so far have only turned on the first one. If you'd like to do a rainbow on all ten LEDs, change the `1` in:

```
led = neopixel.NeoPixel(board.NEOPIXEL, 1)
```

to `10` so it reads:

```
led = neopixel.NeoPixel(board.NEOPIXEL, 10)
```
.

This tells the code to look for 10 LEDs instead of only 1. Now save the code and watch the rainbow go! You can make the same `1` to `10` change to the previous examples as well, and use `led.fill` to light up all the LEDs in the colors you chose! For more details, check out the [NeoPixel section of the CPX guide]() ()!

CircuitPython NeoPixel

NeoPixels are a revolutionary and ultra-popular way to add lights and color to your project. These stranded RGB lights have the controller inside the LED, so you just push the RGB data and the LEDs do all the work for you. They're a perfect match for CircuitPython!

You can drive 300 NeoPixel LEDs with brightness control (set `brightness=1.0` in object creation) and 1000 LEDs without. That's because to adjust the brightness we have to dynamically recreate the data-stream each write.
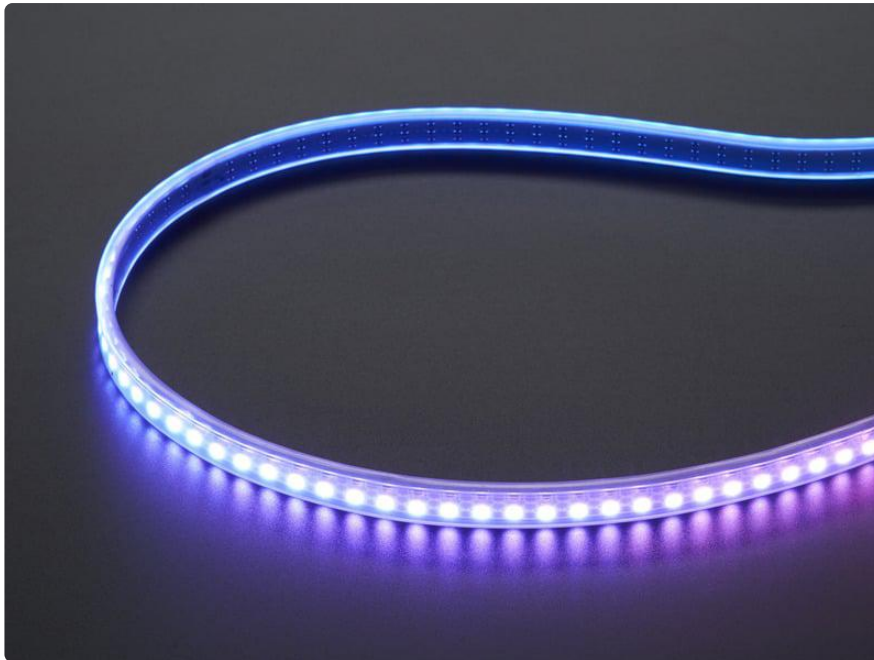
Wiring It Up

You'll need to solder up your NeoPixels first. Verify your connection is on the DATA INPUT or DIN side. Plugging into the DATA OUT or DOUT side is a common mistake! The connections are labeled and some formats have arrows to indicate the direction the data must flow.

For powering the pixels from the board, the 3.3V regulator output can handle about 500mA peak which is about 50 pixels with 'average' use. If you want really bright lights and a lot of pixels, we recommend powering direct from an external power source.

- On Gemma M0 and Circuit Playground Express this is the Vout pad - that pad has direct power from USB or the battery, depending on which is higher voltage.
- On Trinket M0, Feather M0 Express, Feather M4 Express, ItsyBitsy M0 Express and ItsyBitsy M4 Express the USB or BAT pins will give you direct power from the USB port or battery.
- On Metro M0 Express and Metro M4 Express, use the 5V pin regardless of whether it's powered via USB or the DC jack.
- On QT Py M0, use the 5V pin.

If the power to the NeoPixels is greater than 5.5V you may have some difficulty driving some strips, in which case you may need to lower the voltage to 4.5-5V or use a level shifter.

> Do not use the VIN pin directly on Metro M0 Express or Metro M4 Express! The voltage can reach 9V and this can destroy your NeoPixels!
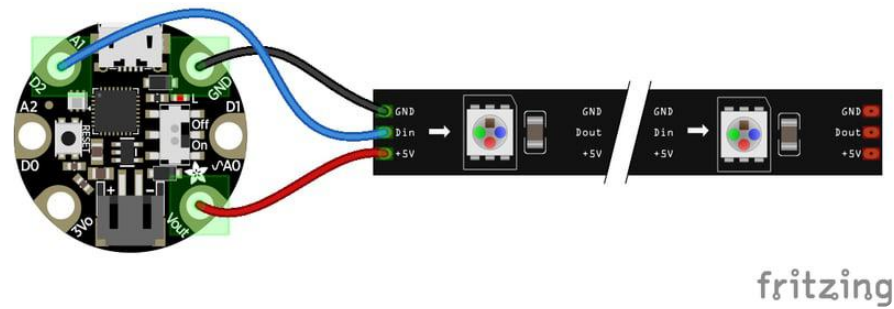
fritzing

> Note that the wire ordering on your NeoPixel strip or shape may not exactly match the diagram above. Check the markings to verify which pin is DIN, 5V and GND

The Code

This example includes multiple visual effects.

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 CircuitPython_Essentials/CircuitPython_NeoPixel/ 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: 2018 Kattni Rembor for Adafruit Industries
#
```
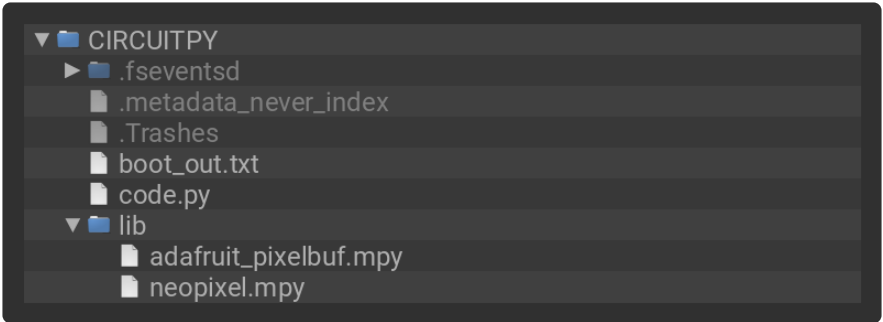
```python
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials NeoPixel example"""
import time
import board
from rainbowio import colorwheel
import neopixel

pixel_pin = board.A1
num_pixels = 8

pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.3, auto_write=False)


def color_chase(color, wait):
    for i in range(num_pixels):
        pixels[i] = color
        time.sleep(wait)
        pixels.show()
    time.sleep(0.5)


def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            rc_index = (i * 256 // num_pixels) + j
            pixels[i] = colorwheel(rc_index & 255)
        pixels.show()
        time.sleep(wait)


RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)

while True:
    pixels.fill(RED)
    pixels.show()
    # Increase or decrease to change the speed of the solid color change.
    time.sleep(1)
    pixels.fill(GREEN)
    pixels.show()
    time.sleep(1)
    pixels.fill(BLUE)
    pixels.show()
    time.sleep(1)

    color_chase(RED, 0.1)  # Increase the number to slow down the color chase
    color_chase(YELLOW, 0.1)
    color_chase(GREEN, 0.1)
    color_chase(CYAN, 0.1)
    color_chase(BLUE, 0.1)
    color_chase(PURPLE, 0.1)

    rainbow_cycle(0)  # Increase the number to slow down the rainbow
```

Create the LED

The first thing we'll do is create the LED object. The NeoPixel object has two required arguments and two optional arguments. You are required to set the pin you're using

to drive your NeoPixels and provide the number of pixels you intend to use. You can optionally set `brightness` and `auto_write`.

NeoPixels can be driven by any pin. We've chosen A1. To set the pin, assign the variable `pixel_pin` to the pin you'd like to use, in our case `board.A1`.

To provide the number of pixels, assign the variable `num_pixels` to the number of pixels you'd like to use. In this example, we're using a strip of `8`.

We've chosen to set `brightness=0.3`, or 30%.

By default, `auto_write=True`, meaning any changes you make to your pixels will be sent automatically. Since `True` is the default, if you use that setting, you don't need to include it in your LED object at all. We've chosen to set `auto_write=False`. If you set `auto_write=False`, you must include `pixels.show()` each time you'd like to send data to your pixels. This makes your code more complicated, but it can make your LED animations faster!

NeoPixel Helpers

Next we've included a few helper functions to create the super fun visual effects found in this code. First is `wheel()` which we just learned with the Internal RGB LED ( ). Then we have `color_chase()` which requires you to provide a `color` and the amount of time in seconds you'd like between each step of the chase. Next we have `rainbow_cycle()`, which requires you to provide the mount of time in seconds you'd like the animation to take. Last, we've included a list of variables for our colors. This makes it much easier if to reuse the colors anywhere in the code, as well as add more colors for use in multiple places. Assigning and using RGB colors is explained in this section of the CircuitPython Internal RGB LED page ().

Main Loop

Thanks to our helpers, our main loop is quite simple. We include the code to set every NeoPixel we're using to red, green and blue for 1 second each. Then we call `color_chase()`, one time for each `color` on our list with `0.1` second delay between setting each subsequent LED the same color during the chase. Last we call `rainbow_cycle(0)`, which means the animation is as fast as it can be. Increase both of those numbers to slow down each animation!

Note that the longer your strip of LEDs, the longer it will take for the animations to complete.

NeoPixel RGBW

NeoPixels are available in RGB, meaning there are three LEDs inside, red, green and blue. They're also available in RGBW, which includes four LEDs, red, green, blue and white. The code for RGBW NeoPixels is a little bit different than RGB.

If you run RGB code on RGBW NeoPixels, approximately 3/4 of the LEDs will light up and the LEDs will be the incorrect color even though they may appear to be changing. This is because NeoPixels require a piece of information for each available color (red, green, blue and possibly white).

Therefore, RGB LEDs require three pieces of information and RGBW LEDs require FOUR pieces of information to work. So when you create the LED object for RGBW LEDs, you'll include `pixel_order=(1, 0, 2, 3)`, which sets the pixel order and indicates four pieces of information involved.

Then, you must include an extra number in every color tuple you create. For example, red will be `(255, 0, 0, 0)`. This is how you send the fourth piece of information. Check out the example below to see how our NeoPixel code looks for using with RGBW LEDs!

The 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 CircuitPython_Essentials/CircuitPython_NeoPixel_RGBW/ 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
    ▼ ■ lib
        ■ adafruit_pixelbuf.mpy
        ■ neopixel.mpy
```
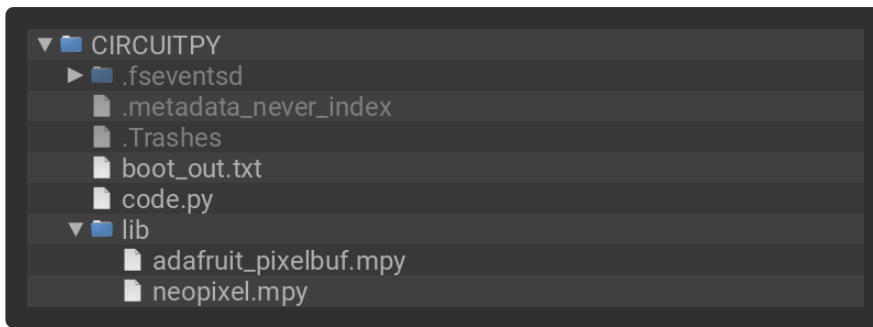
```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials NeoPixel RGBW example"""
import time
import board
import neopixel

pixel_pin = board.A1
num_pixels = 8

pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.3, auto_write=False,
                           pixel_order=(1, 0, 2, 3))


def colorwheel(pos):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition r - g - b - back to r.
    if pos < 0 or pos > 255:
        return (0, 0, 0, 0)
    if pos < 85:
        return (255 - pos * 3, pos * 3, 0, 0)
    if pos < 170:
        pos -= 85
        return (0, 255 - pos * 3, pos * 3, 0)
    pos -= 170
    return (pos * 3, 0, 255 - pos * 3, 0)


def color_chase(color, wait):
    for i in range(num_pixels):
        pixels[i] = color
        time.sleep(wait)
        pixels.show()
    time.sleep(0.5)


def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            rc_index = (i * 256 // num_pixels) + j
            pixels[i] = colorwheel(rc_index & 255)
        pixels.show()
        time.sleep(wait)


RED = (255, 0, 0, 0)
YELLOW = (255, 150, 0, 0)
GREEN = (0, 255, 0, 0)
CYAN = (0, 255, 255, 0)
BLUE = (0, 0, 255, 0)
PURPLE = (180, 0, 255, 0)

while True:
```

```
    pixels.fill(RED)
    pixels.show()
    # Increase or decrease to change the speed of the solid color change.
    time.sleep(1)
    pixels.fill(GREEN)
    pixels.show()
    time.sleep(1)
    pixels.fill(BLUE)
    pixels.show()
    time.sleep(1)

    color_chase(RED, 0.1)  # Increase the number to slow down the color chase
    color_chase(YELLOW, 0.1)
    color_chase(GREEN, 0.1)
    color_chase(CYAN, 0.1)
    color_chase(BLUE, 0.1)
    color_chase(PURPLE, 0.1)

    rainbow_cycle(0)  # Increase the number to slow down the rainbow
```

Read the Docs

For a more in depth look at what `neopixel` can do, check out NeoPixel on Read the Docs ().

CircuitPython DotStar

DotStars use two wires, unlike NeoPixel's one wire. They're very similar but you can write to DotStars much faster with hardware SPI and they have a faster PWM cycle so they are better for light painting.

Any pins can be used but if the two pins can form a hardware SPI port, the library will automatically switch over to hardware SPI. If you use hardware SPI then you'll get 4 MHz clock rate (that would mean updating a 64 pixel strand in about 500uS - that's 0.0005 seconds). If you use non-hardware SPI pins you'll drop down to about 3KHz, 1000 times as slow!

You can drive 300 DotStar LEDs with brightness control (set `brightness=1.0` in object creation) and 1000 LEDs without. That's because to adjust the brightness we have to dynamically recreate the data-stream each write.

You'll need the adafruit_dotstar.mpy library if you don't already have it in your /lib folder! You can get it from the CircuitPython Library Bundle (). If you need help installing the library, check out the CircuitPython Libraries page ().
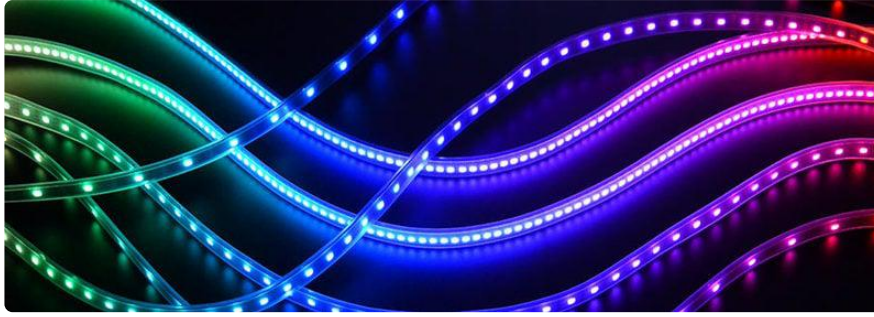
Wire It Up

You'll need to solder up your DotStars first. Verify your connection is on the DATA INPUT or DI and CLOCK INPUT or CI side. Plugging into the DATA OUT/DO or CLOCK OUT/CO side is a common mistake! The connections are labeled and some formats have arrows to indicate the direction the data must flow. Always verify your wiring with a visual inspection, as the order of the connections can differ from strip to strip!

For powering the pixels from the board, the 3.3V regulator output can handle about 500mA peak which is about 50 pixels with 'average' use. If you want really bright lights and a lot of pixels, we recommend powering direct from an external power source.

- On Gemma M0 and Circuit Playground Express this is the Vout pad - that pad has direct power from USB or the battery, depending on which is higher voltage.
- On Trinket M0, Feather M0 Express, Feather M4 Express, ItsyBitsy M0 Express and ItsyBitsy M4 Express the USB or BAT pins will give you direct power from the USB port or battery.
- On Metro M0 Express and Metro M4 Express, use the 5V pin regardless of whether it's powered via USB or the DC jack.
- On QT Py M0, use the 5V pin.

If the power to the DotStars is greater than 5.5V you may have some difficulty driving some strips, in which case you may need to lower the voltage to 4.5-5V or use a level shifter.

Do not use the VIN pin directly on Metro M0 Express or Metro M4 Express! The voltage can reach 9V and this can destroy your DotStars!

fritzing
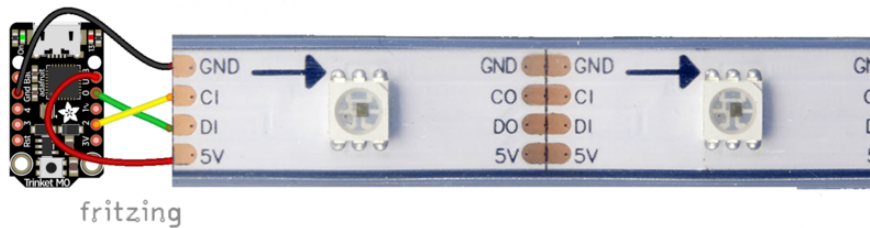
> Note that the wire ordering on your DotStar strip or shape may not exactly match the diagram above. Check the markings to verify which pin is DIN, CIN, 5V and GND

The Code

This example includes multiple visual effects.

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 CircuitPython_Essentials/CircuitPython_DotStar/ 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:
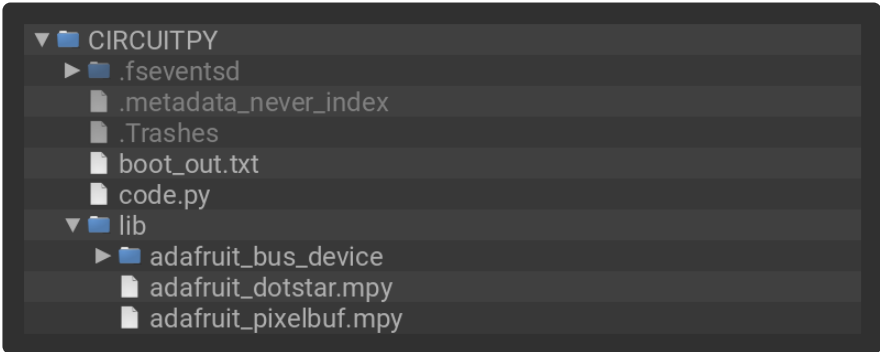

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials DotStar example"""
import time
```

```python
from rainbowio import colorwheel
import adafruit_dotstar
import board

num_pixels = 30
pixels = adafruit_dotstar.DotStar(board.A1, board.A2, num_pixels, brightness=0.1,
auto_write=False)


def color_fill(color, wait):
    pixels.fill(color)
    pixels.show()
    time.sleep(wait)


def slice_alternating(wait):
    pixels[::2] = [RED] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [ORANGE] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[::2] = [YELLOW] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [GREEN] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[::2] = [TEAL] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [CYAN] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[::2] = [BLUE] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [PURPLE] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[::2] = [MAGENTA] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)
    pixels[1::2] = [WHITE] * (num_pixels // 2)
    pixels.show()
    time.sleep(wait)


def slice_rainbow(wait):
    pixels[::6] = [RED] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[1::6] = [ORANGE] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[2::6] = [YELLOW] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[3::6] = [GREEN] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[4::6] = [BLUE] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
    pixels[5::6] = [PURPLE] * (num_pixels // 6)
    pixels.show()
    time.sleep(wait)
```

```python
def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            rc_index = (i * 256 // num_pixels) + j
            pixels[i] = colorwheel(rc_index & 255)
        pixels.show()
        time.sleep(wait)


RED = (255, 0, 0)
YELLOW = (255, 150, 0)
ORANGE = (255, 40, 0)
GREEN = (0, 255, 0)
TEAL = (0, 255, 120)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)
MAGENTA = (255, 0, 20)
WHITE = (255, 255, 255)

while True:
    # Change this number to change how long it stays on each solid color.
    color_fill(RED, 0.5)
    color_fill(YELLOW, 0.5)
    color_fill(ORANGE, 0.5)
    color_fill(GREEN, 0.5)
    color_fill(TEAL, 0.5)
    color_fill(CYAN, 0.5)
    color_fill(BLUE, 0.5)
    color_fill(PURPLE, 0.5)
    color_fill(MAGENTA, 0.5)
    color_fill(WHITE, 0.5)

    # Increase or decrease this to speed up or slow down the animation.
    slice_alternating(0.1)

    color_fill(WHITE, 0.5)

    # Increase or decrease this to speed up or slow down the animation.
    slice_rainbow(0.1)

    time.sleep(0.5)

    # Increase this number to slow down the rainbow animation.
    rainbow_cycle(0)
```

We've chosen pins A1 and A2, but these are not SPI pins on all boards. DotStars respond faster when using hardware SPI!

Create the LED

The first thing we'll do is create the LED object. The DotStar object has three required arguments and two optional arguments. You are required to set the pin you're using for data, set the pin you'll be using for clock, and provide the number of pixels you intend to use. You can optionally set `brightness` and `auto_write`.

DotStars can be driven by any two pins. We've chosen A1 for clock and A2 for data. To set the pins, include the pin names at the beginning of the object creation, in this case `board.A1` and `board.A2`.

To provide the number of pixels, assign the variable `num_pixels` to the number of pixels you'd like to use. In this example, we're using a strip of `72`.

We've chosen to set `brightness=0.1`, or 10%.

By default, `auto_write=True`, meaning any changes you make to your pixels will be sent automatically. Since `True` is the default, if you use that setting, you don't need to include it in your LED object at all. We've chosen to set `auto_write=False`. If you set `auto_write=False`, you must include `pixels.show()` each time you'd like to send data to your pixels. This makes your code more complicated, but it can make your LED animations faster!

DotStar Helpers

We've included a few helper functions to create the super fun visual effects found in this code.

First is `wheel()` which we just learned with the Internal RGB LED (). Then we have `color_fill()` which requires you to provide a `color` and the length of time you'd like it to be displayed. Next, are `slice_alternating()`, `slice_rainbow()`, and `rainbow_cycle()` which require you to provide the amount of time in seconds you'd between each step of the animation.

Last, we've included a list of variables for our colors. This makes it much easier if to reuse the colors anywhere in the code, as well as add more colors for use in multiple places. Assigning and using RGB colors is explained in this section of the CircuitPython Internal RGB LED page ().

The two slice helpers utilise a nifty feature of the DotStar library that allows us to use math to light up LEDs in repeating patterns. `slice_alternating()` first lights up the even number LEDs and then the odd number LEDs and repeats this back and forth. `slice_rainbow()` lights up every sixth LED with one of the six rainbow colors until the strip is filled. Both use our handy color variables. This slice code only works when the total number of LEDs is divisible by the slice size, in our case 2 and 6. DotStars come in strips of 30, 60, 72, and 144, all of which are divisible by 2 and 6. In the event that you cut them into different sized strips, the code in this example may not work without

modification. However, as long as you provide a total number of LEDs that is divisible by the slices, the code will work.

Main Loop

Our main loop begins by calling `color_fill()` once for each `color` on our list and sets each to hold for `0.5` seconds. You can change this number to change how fast each color is displayed. Next, we call `slice_alternating(0.1)`, which means there's a 0.1 second delay between each change in the animation. Then, we fill the strip white to create a clean backdrop for the rainbow to display. Then, we call `slice_rainbow(0.1)`, for a 0.1 second delay in the animation. Last we call `rainbow_cycle(0)`, which means it's as fast as it can possibly be. Increase or decrease either of these numbers to speed up or slow down the animations!

Note that the longer your strip of LEDs is, the longer it will take for the animations to complete.

> We have a ton more information on general purpose DotStar know-how at our DotStar UberGuide https://learn.adafruit.com/adafruit-dotstar-leds

Is it SPI?

We explained at the beginning of this section that the LEDs respond faster if you're using hardware SPI. On some of the boards, there are HW SPI pins directly available in the form of MOSI and SCK. However, hardware SPI is available on more than just those pins. But, how can you figure out which? Easy! We wrote a handy script.

We chose pins A1 and A2 for our example code. To see if these are hardware SPI on the board you're using, copy and paste the code into code.py using your favorite editor, and save the file. Then connect to the serial console to see the results.

To check if other pin combinations have hardware SPI, change the pin names on the line reading: `if is_hardware_SPI(board.A1, board.A2):` to the pins you want to use. Then, check the results in the serial console. Super simple!

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/SPI_Test_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:
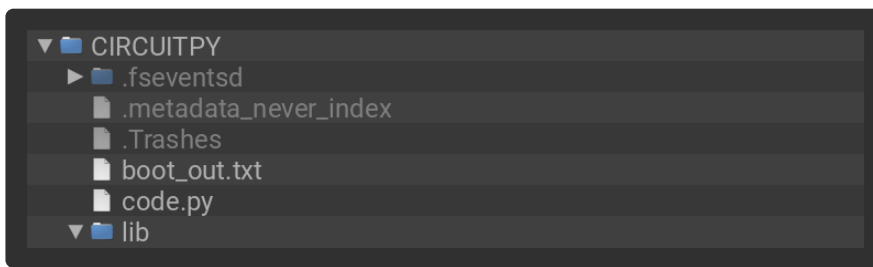

```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Hardware SPI pin verification script"""
import board
import busio


def is_hardware_spi(clock_pin, data_pin):
    try:
        p = busio.SPI(clock_pin, data_pin)
        p.deinit()
        return True
    except ValueError:
        return False


# Provide the two pins you intend to use.
if is_hardware_spi(board.A1, board.A2):
    print("This pin combination is hardware SPI!")
else:
    print("This pin combination isn't hardware SPI.")
```

Read the Docs

For a more in depth look at what `dotstar` can do, check out DotStar on Read the Docs ().


CircuitPython UART Serial

In addition to the USB-serial connection you use for the REPL, there is also a hardware UART you can use. This is handy to talk to UART devices like GPSs, some sensors, or other microcontrollers!

This quick-start example shows how you can create a UART device for communicating with hardware serial devices.

To use this example, you'll need something to generate the UART data. We've used a GPS! Note that the GPS will give you UART data without getting a fix on your location.

You can use this example right from your desk! You'll have data to read, it simply won't include your actual location.

> The QT Py M0 does not have a little red LED. Therefore, you must connect an external LED and edit this example for it to work. Follow the wiring diagram and steps below to run this example on QT Py M0.
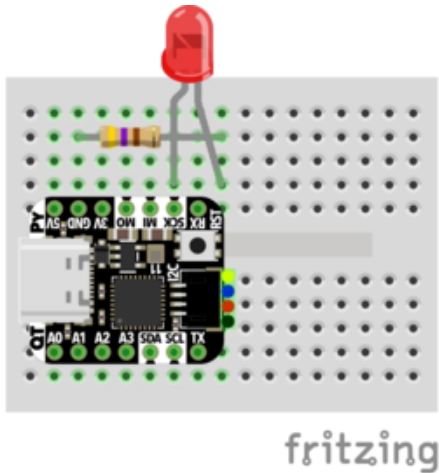
LED + to QT Py SCK
LED - to 470Ω resistor
470Ω resistor to QT Py GND

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/CircuitPython_UART/ 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:
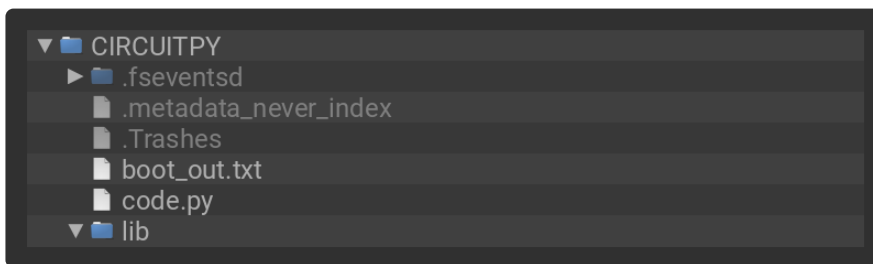

```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials UART Serial example"""
import board
import busio
import digitalio

# For most CircuitPython boards:
led = digitalio.DigitalInOut(board.LED)
# For QT Py M0:
# led = digitalio.DigitalInOut(board.SCK)
led.direction = digitalio.Direction.OUTPUT
```

```
uart = busio.UART(board.TX, board.RX, baudrate=9600)

while True:
    data = uart.read(32)  # read up to 32 bytes
    # print(data)  # this is a bytearray type

    if data is not None:
        led.value = True

        # convert bytearray to string
        data_string = ''.join([chr(b) for b in data])
        print(data_string, end="")

        led.value = False
```

> Note: To "comment out" a line, put a # and a space before it. To "uncomment" a line, remove the # + space from the beginning of the line.

For QT Py M0, you'll need to comment out `led = DigitalInOut(board.LED)` and uncomment `led = DigitalInOut(board.SCK)`. The UART code remains the same.

The Code

First we create the UART object. We provide the pins we'd like to use, `board.TX` and `board.RX`, and we set the `baudrate=9600`. While these pins are labeled on most of the boards, be aware that RX and TX are not labeled on Gemma, and are labeled on the bottom of Trinket. See the diagrams below for help with finding the correct pins on your board.

Once the object is created you read data in with `read(numbytes)` where you can specify the max number of bytes. It will return a byte array type object if anything was received already. Note it will always return immediately because there is an internal buffer! So read as much data as you can 'digest'.

If there is no data available, `read()` will return `None`, so check for that before continuing.
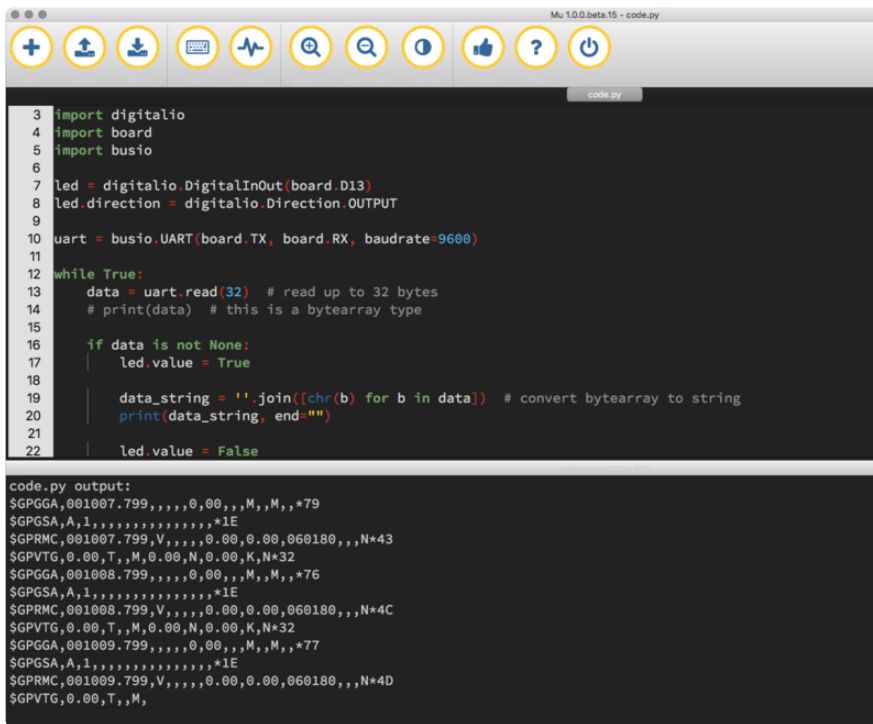
The data that is returned is in a byte array, if you want to convert it to a string, you can use this handy line of code which will run `chr()` on each byte:

```
datastr = ''.join([chr(b) for b in data]) # convert bytearray to string
```

Your results will look something like this:

```
3  import digitalio
4  import board
5  import busio
6
7  led = digitalio.DigitalInOut(board.D13)
8  led.direction = digitalio.Direction.OUTPUT
9
10 uart = busio.UART(board.TX, board.RX, baudrate=9600)
11
12 while True:
13     data = uart.read(32)  # read up to 32 bytes
14     # print(data)  # this is a bytearray type
15
16     if data is not None:
17         led.value = True
18
19         data_string = ''.join([chr(b) for b in data])  # convert bytearray to string
20         print(data_string, end="")
21
22         led.value = False
```

```
code.py output:
$GPGGA,001007.799,,,,,0,00,,,M,,M,,*79
$GPGSA,A,1,,,,,,,,,,,,,,,,*1E
$GPRMC,001007.799,V,,,,,0.00,0.00,060180,,,N*43
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,001008.799,,,,,0,00,,,M,,M,,*76
$GPGSA,A,1,,,,,,,,,,,,,,,,*1E
$GPRMC,001008.799,V,,,,,0.00,0.00,060180,,,N*4C
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,001009.799,,,,,0,00,,,M,,M,,*77
$GPGSA,A,1,,,,,,,,,,,,,,,,*1E
$GPRMC,001009.799,V,,,,,0.00,0.00,060180,,,N*4D
$GPVTG,0.00,T,,M,
```

> For more information about the data you're reading and the Ultimate GPS, check out the Ultimate GPS guide: https://learn.adafruit.com/adafruit-ultimate-gps

Wire It Up

You'll need a couple of things to connect the GPS to your board.

For Gemma M0 and Circuit Playground Express, you can use use alligator clips to connect to the Flora Ultimate GPS Module.

For Trinket M0, Feather M0 Express, Metro M0 Express and ItsyBitsy M0 Express, you'll need a breadboard and jumper wires to connect to the Ultimate GPS Breakout.

We've included diagrams show you how to connect the GPS to your board. In these diagrams, the wire colors match the same pins on each board.

- The black wire connects between the ground pins.
- The red wire connects between the power pins on the GPS and your board.
- The blue wire connects from TX on the GPS to RX on your board.
- The white wire connects from RX on the GPS to TX on your board.

Check out the list below for a diagram of your specific board!

Watch out! A common mixup with UART serial is that RX on one board connects to TX on the other! However, sometimes boards have RX labeled TX and vice versa. So, you'll want to start with RX connected to TX, but if that doesn't work, try the other way around!
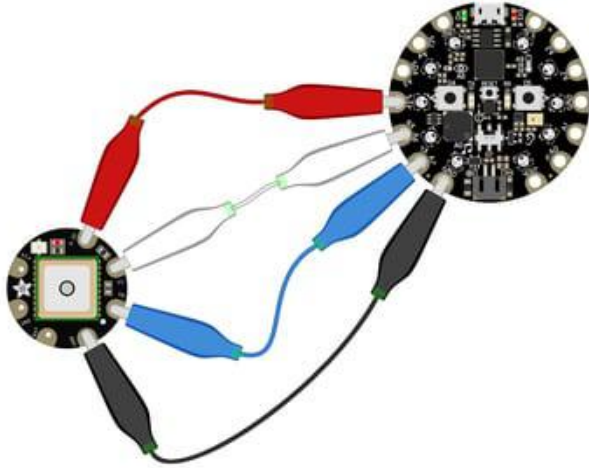

Circuit Playground Express and Circuit Playground Bluefruit

Connect 3.3v on your CPX to 3.3v on your GPS.
Connect GND on your CPX to GND on your GPS.
Connect RX/A6 on your CPX to TX on your GPS.
Connect TX/A7 on your CPX to RX on your GPS.
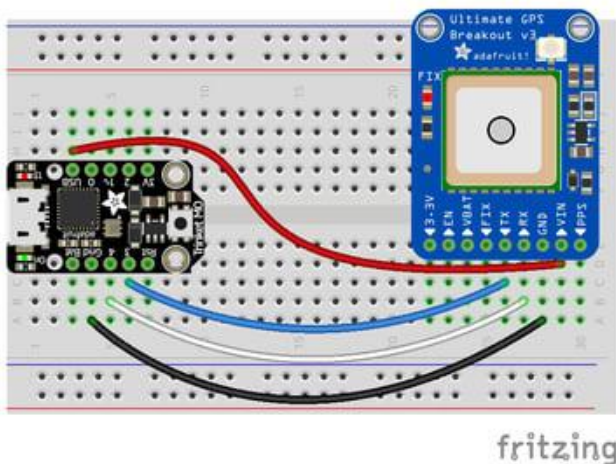
Trinket M0

Connect USB on the Trinket to VIN on the GPS.
Connect Gnd on the Trinket to GND on the GPS.
Connect D3 on the Trinket to TX on the GPS.
Connect D4 on the Trinket to RX on the GPS.
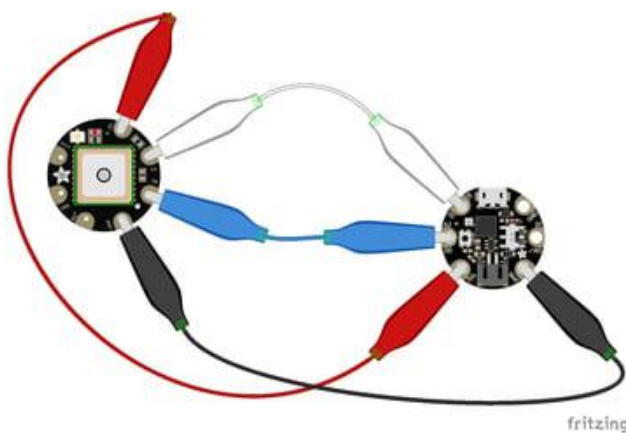
Gemma M0

Connect 3vo on the Gemma to 3.3v on the GPS.
Connect GND on the Gemma to GND on the GPS.
Connect A1/D2 on the Gemma to TX on the GPS.
Connect A2/D0 on the Gemma to RX on the GPS.
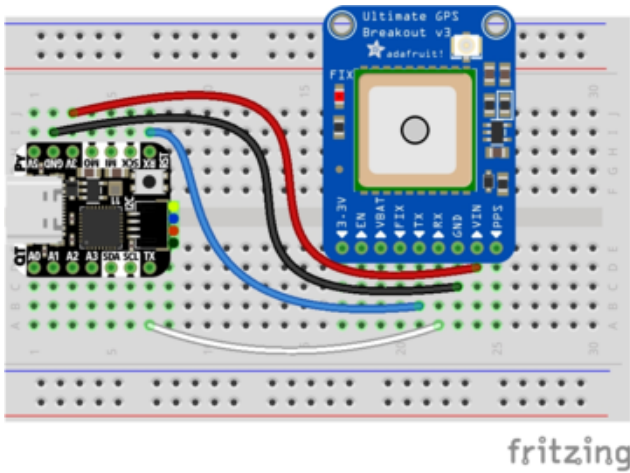
QT Py M0

Connect 3V on the QT Py to VIN on the GPS.
Connect GND on the QT Py to GND on the GPS.
Connect RX on the QT Py to TX on the GPS.
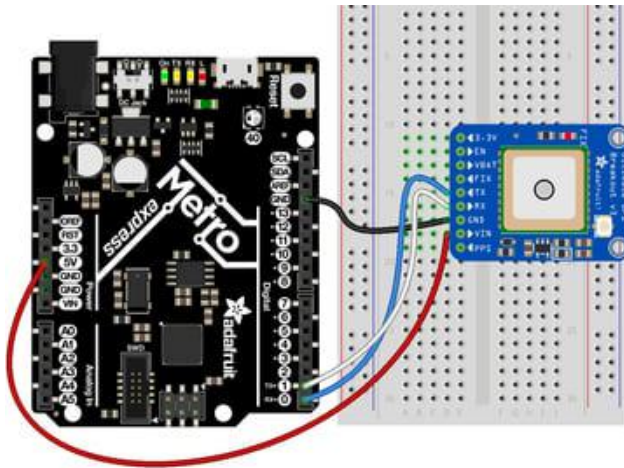Connect TX on the QT Py to RX on the GPS.
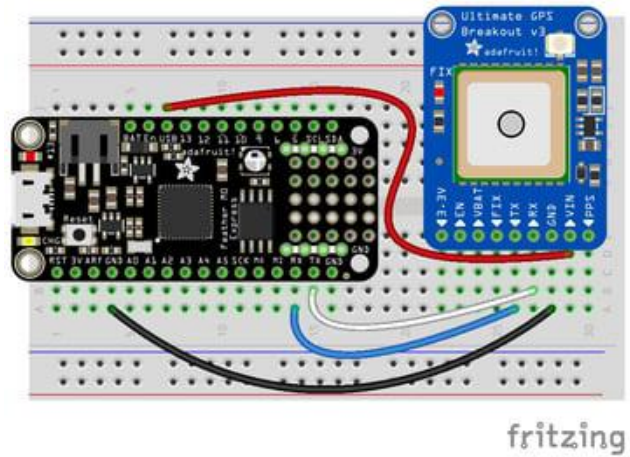
Feather M0 Express and Feather M4 Express

Connect USB on the Feather to VIN on the GPS.
Connect GND on the Feather to GND on the GPS.
Connect RX on the Feather to TX on the GPS.
Connect TX on the Feather to RX on the GPS.
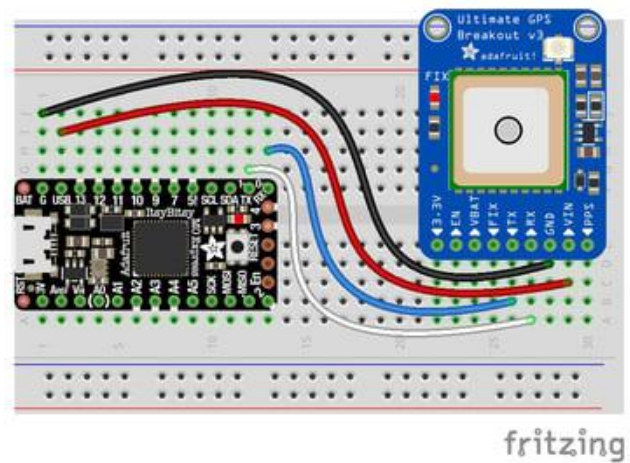
ItsyBitsy M0 Express and ItsyBitsy M4 Express

Connect USB on the ItsyBitsy to VIN on the GPS
Connect G on the ItsyBitsy to GND on the GPS.
Connect RX/0 on the ItsyBitsy to TX on the GPS.
Connect TX/1 on the ItsyBitsy to RX on the GPS.

Metro M0 Express and Metro M4 Express

Connect 5V on the Metro to VIN on the GPS.
Connect GND on the Metro to GND on the GPS.
Connect RX/D0 on the Metro to TX on the GPS.
Connect TX/D1 on the Metro to RX on the GPS.
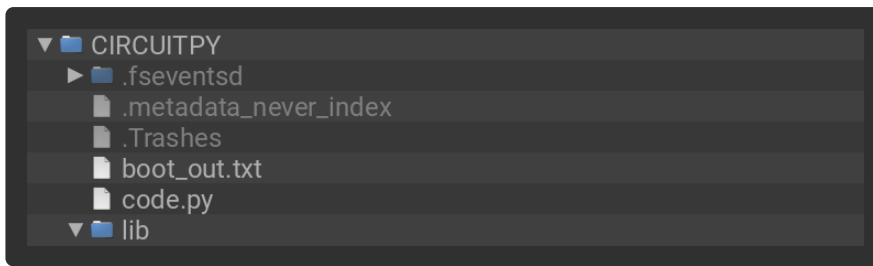
Where's my UART?

On the SAMD21, we have the flexibility of using a wide range of pins for UART. Compare this to some chips like the ESP8266 with fixed UART pins. The good news is you can use many but not all pins. Given the large number of SAMD boards we have, its impossible to guarantee anything other than the labeled 'TX' and 'RX'. So, if you want some other setup, or multiple UARTs, how will you find those pins? Easy! We've written a handy script.

These are the results from a Trinket M0, your output may vary and it might be very long. [For more details about UARTs and SERCOMs check out our detailed guide here ()](#)

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
RX pin: board.D2       TX pin: board.D0
RX pin: board.D4       TX pin: board.D0
RX pin: board.D3       TX pin: board.D0
RX pin: board.D13      TX pin: board.D0
RX pin: board.D0       TX pin: board.D4
RX pin: board.D2       TX pin: board.D4
RX pin: board.D3       TX pin: board.D4
RX pin: board.D0       TX pin: board.D13
RX pin: board.D2       TX pin: board.D13
RX pin: board.D3       TX pin: board.D13
```

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/UART_Test_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: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials UART possible pin-pair identifying script"""
import board
import busio
from microcontroller import Pin


def is_hardware_uart(tx, rx):
    try:
        p = busio.UART(tx, rx)
        p.deinit()
        return True
    except ValueError:
        return False


def get_unique_pins():
    exclude = ['NEOPIXEL', 'APA102_MOSI', 'APA102_SCK']
    pins = [pin for pin in [
        getattr(board, p) for p in dir(board) if p not in exclude]
            if isinstance(pin, Pin)]
    unique = []
    for p in pins:
        if p not in unique:
            unique.append(p)
    return unique


for tx_pin in get_unique_pins():
    for rx_pin in get_unique_pins():
        if rx_pin is tx_pin:
            continue
        if is_hardware_uart(tx_pin, rx_pin):
            print("RX pin:", rx_pin, "\t TX pin:", tx_pin)
```

Trinket M0: Create UART before I2C

On the Trinket M0 (only), if you are using both UART and I2C, you must create the
UART object first, e.g.:

```
>>> import board
>>> uart = board.UART()    # Uses pins 4 and 3 for TX and RX, baudrate 9600.
>>> i2c = board.I2C()      # Uses pins 2 and 0 for SCL and SDA.

# or alternatively,
```

Creating the I2C object first does not work:

```
>>> import board
>>> i2c = board.I2C()     # Uses pins 2 and 0 for SCL and SDA.
>>> uart = board.UART()   # Uses pins 4 and 3 for TX and RX, baudrate 9600.
Traceback (most recent call last):
File "", line 1, in
ValueError: Invalid pins
```

CircuitPython I2C

I2C is a 2-wire protocol for communicating with simple sensors and devices, meaning it uses two connections for transmitting and receiving data. There are many I2C devices available and they're really easy to use with CircuitPython. We have libraries available for many I2C devices in the library bundle (). (If you don't see the sensor you're looking for, keep checking back, more are being written all the time!)

In this section, we're going to do is learn how to scan the I2C bus for all connected devices. Then we're going to learn how to interact with an I2C device.

We'll be using the Adafruit TSL2591 (), a common, low-cost light sensor. While the exact code we're running is specific to the TSL2591 the overall process is the same for just about any I2C sensor or device.

These examples will use the TSL2591 lux sensor breakout. The first thing you'll want to do is get the sensor connected so your board has I2C to talk to.

Wire It Up

You'll need a couple of things to connect the TSL2591 to your board. The TSL2591 comes with STEMMA QT / QWIIC connectors on it, which makes it super simple to wire it up. No further soldering required!

For Gemma M0, Circuit Playground Express and Circuit Playground Bluefruit, you can use use the STEMMA QT to alligator clips cable () to connect to the TSL2591.

For Trinket M0, Feather M0 and M4 Express, Metro M0 and M4 Express and ItsyBitsy M0 and M4 Express, you'll need a breadboard and STEMMA QT to male jumper wires cable () to connect to the TSL2591.

For QT Py M0, you'll need a STEMMA QT cable () to connect to the TSL2591.

We've included diagrams show you how to connect the TSL2591 to your board. In these diagrams, the wire colors match the STEMMA QT cables and connect to the same pins on each board.

- The black wire connects from GND on the TSL2591 to ground on your board.
- The red wire connects from VIN on the TSL2591 to power on your board.
- The yellow wire connects from SCL on the TSL2591 to SCL on your board.
- The blue wire connects from SDA on the TSL2591 to SDA on your board.

Check out the list below for a diagram of your specific board!

Be aware that the Adafruit microcontroller boards do not have I2C pullup resistors built in! All of the Adafruit breakouts do, but if you're building your own board or using a non-Adafruit breakout, you must add 2.2K-10K ohm pullups on both SDA and SCL to the 3.3V.

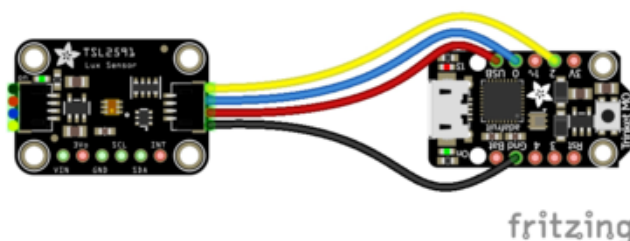Circuit Playground Express and Circuit Playground Bluefruit
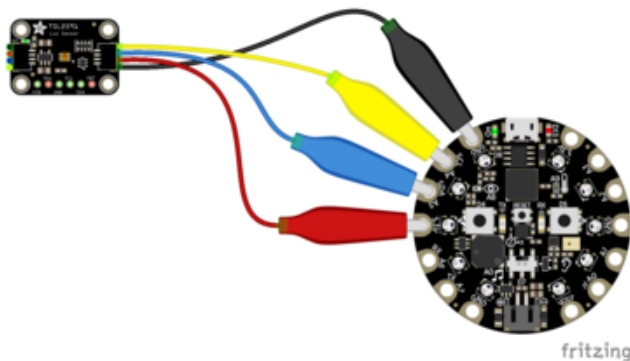
Connect 3.3v on your CPX to 3.3v on your TSL2591.
Connect GND on your CPX to GND on your TSL2591.
Connect SCL/A4 on your CPX to SCL on your TSL2591.
Connect SDL/A5 on your CPX to SDA on your TSL2591.
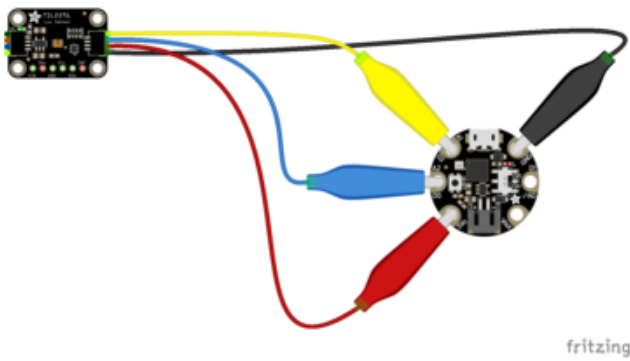
Trinket M0

Connect USB on the Trinket to VIN on the TSL2591.
Connect Gnd on the Trinket to GND on the TSL2591.
Connect D2 on the Trinket to SCL on the TSL2591.
Connect D0 on the Trinket to SDA on the TSL2591.

Gemma M0

Connect 3vo on the Gemma to 3V on the TSL2591.
Connect GND on the Gemma to GND on the TSL2591.
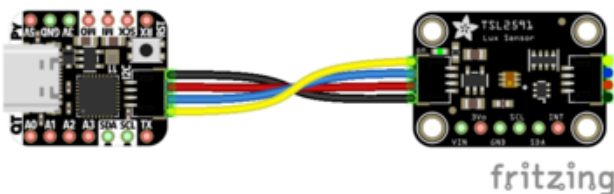Connect A1/D2 on the Gemma to SCL on the TSL2591.
Connect A2/D0 on the Gemma to SDA on the TSL2591.

QT Py M0

If using the STEMMA QT cable:

Connect the STEMMA QT cable from the connector on the QT Py to the connector on the TSL2591.
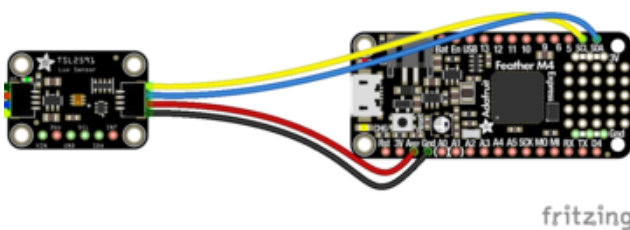Alternatively, if using a breadboard:

Connect 3V on the QT Py to VIN on the TSL2591.
Connect GND on the QT Py to GND on the TSL2591.
Connect SCL on the QT Py to SCL on the TSL2591.
Connect SDA on the QT Py to SDA on the TSL2591.
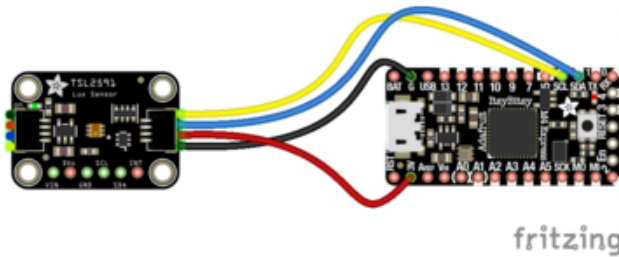
Feather M0 Express and Feather M4 Express

Connect USB on the Feather to VIN on the TSL2591.
Connect GND on the Feather to GND on the TSL2591.
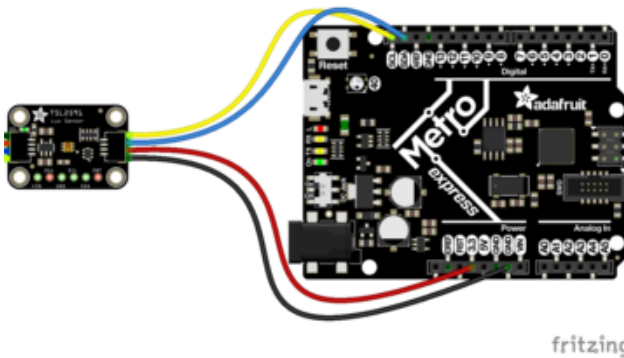Connect SCL on the Feather to SCL on the TSL2591.
Connect SDA on the Feather to SDA on the TSL2591.

ItsyBitsy M0 Express and ItsyBitsy M4 Express

Connect USB on the ItsyBitsy to VIN on the TSL2591
Connect G on the ItsyBitsy to GND on the TSL2591.
Connect SCL on the ItsyBitsy to SCL on the TSL2591.
Connect SDA on the ItsyBitsy to SDA on the TSL2591.

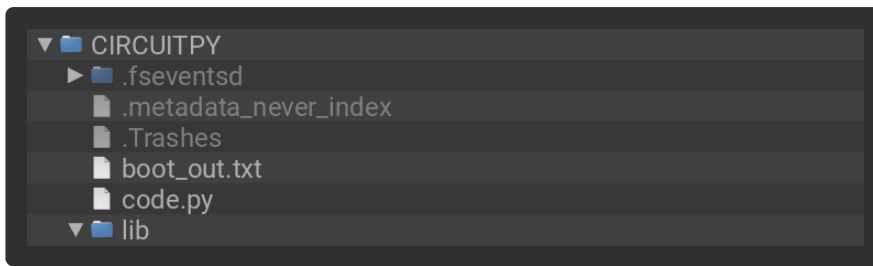Metro M0 Express and Metro M4 Express

Connect 5V on the Metro to VIN on the TSL2591.
Connect GND on the Metro to GND on the TSL2591.
Connect SCL on the Metro to SCL on the TSL2591.
Connect SDA on the Metro to SDA on the TSL2591.

Find Your Sensor

The first thing you'll want to do after getting the sensor wired up, is make sure it's wired correctly. We're going to do an I2C scan to see if the board is detected, and if it is, print out its I2C address.

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/CircuitPython_I2C_Scan/ 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: 2017 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython I2C Device Address Scan"""
# If you run this and it seems to hang, try manually unlocking
# your I2C bus from the REPL with
#   >>> import board
#   >>> board.I2C().unlock()

import time
import board

# To use default I2C bus (most boards)
i2c = board.I2C()

# To create I2C bus on specific pins
# import busio
# i2c = busio.I2C(board.SCL1, board.SDA1)  # QT Py RP2040 STEMMA connector
# i2c = busio.I2C(board.GP1, board.GP0)    # Pi Pico RP2040

while not i2c.try_lock():
    pass

try:
    while True:
        print(
            "I2C addresses found:",
            [hex(device_address) for device_address in i2c.scan()],
        )
        time.sleep(2)

finally:  # unlock the i2c bus when ctrl-c'ing out of the loop
    i2c.unlock()
```

First we create the `i2c` object, using `board.I2C()`. This convenience routine creates and saves a `busio.I2C` object using the default pins `board.SCL` and `board.SDA`. If the object has already been created, then the existing object is returned. No matter how many times you call `board.I2C()`, it will return the same object. This is called a singleton.

To be able to scan it, we need to lock the I2C down so the only thing accessing it is the code. So next we include a loop that waits until I2C is locked and then continues on to the scan function.

Last, we have the loop that runs the actual scan, `i2c_scan()`. Because I2C typically refers to addresses in hex form, we've included this bit of code that formats the

results into hex format: `[hex(device_address) for device_address in i2c.scan()]`.

Open the serial console to see the results! The code prints out an array of addresses. We've connected the TSL2591 which has a 7-bit I2C address of 0x29. The result for this sensor is `I2C addresses found: ['0x29']`. If no addresses are returned, refer back to the wiring diagrams to make sure you've wired up your sensor correctly.
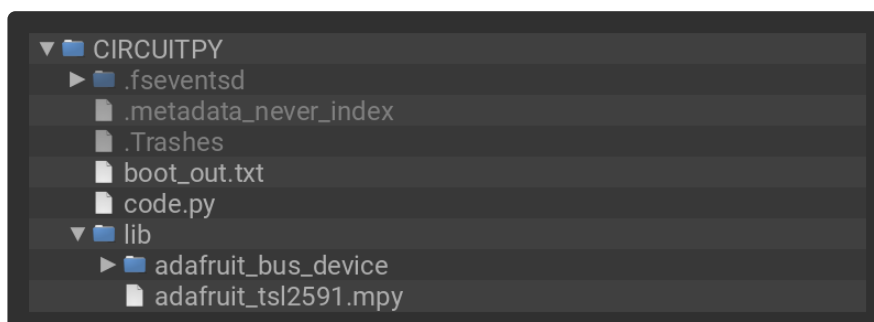
I2C Sensor Data

Now we know for certain that our sensor is connected and ready to go. Let's find out how to get the data from our sensor!

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 CircuitPython_Essentials/CircuitPython_I2C_TSL2591/ 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:


```python
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials I2C sensor example using TSL2591"""
import time
import board
import adafruit_tsl2591
```

```
i2c = board.I2C()

# Lock the I2C device before we try to scan
while not i2c.try_lock():
    pass
# Print the addresses found once
print("I2C addresses found:", [hex(device_address) for device_address in
i2c.scan()])

# Unlock I2C now that we're done scanning.
i2c.unlock()

# Create library object on our I2C port
tsl2591 = adafruit_tsl2591.TSL2591(i2c)

# Use the object to print the sensor readings
while True:
    print("Lux:", tsl2591.lux)
    time.sleep(0.5)
```

This code begins the same way as the scan code. We've included the scan code so you have verification that your sensor is wired up correctly and is detected. It prints the address once. After the scan, we unlock I2C with `i2c_unlock()` so we can use the sensor for data.

We create our sensor object using the sensor library. We call it `tsl2591` and provide it the `i2c` object.

Then we have a simple loop that prints out the lux reading using the sensor object we created. We add a `time.sleep(1.0)`, so it only prints once per second. Connect to the serial console to see the results. Try shining a light on it to see the results change!

Where's my I2C?

On the SAMD21, SAMD51 and nRF52840, we have the flexibility of using a wide range of pins for I2C. On the nRF52840, any pin can be used for I2C! Some chips, like the ESP8266, require using bitbangio, but can also use any pins for I2C. There's some other chips that may have fixed I2C pin.

The good news is you can use many but not all pins. Given the large number of SAMD boards we have, its impossible to guarantee anything other than the labeled 'SDA' and 'SCL'. So, if you want some other setup, or multiple I2C interfaces, how will you find those pins? Easy! We've written a handy script.
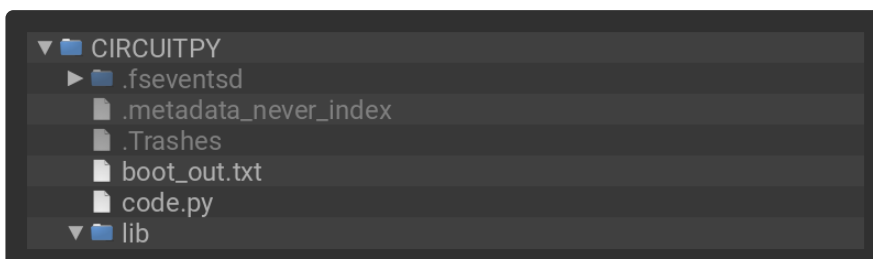
These are the results from an ItsyBitsy M0 Express. Your output may vary and it might be very long. For more details about I2C and SERCOMs, check out our detailed guide here ().

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
SCL pin: board.D3       SDA pin: board.D4
SCL pin: board.D3       SDA pin: board.A3
SCL pin: board.D3       SDA pin: board.MISO
SCL pin: board.D13      SDA pin: board.D11
SCL pin: board.D13      SDA pin: board.SDA
SCL pin: board.A2       SDA pin: board.A1
SCL pin: board.A2       SDA pin: board.MISO
SCL pin: board.A4       SDA pin: board.D4
SCL pin: board.A4       SDA pin: board.A3
SCL pin: board.SCL      SDA pin: board.D11
SCL pin: board.SCL      SDA pin: board.SDA


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

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/I2C_Test_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:

```
▼ 📁 CIRCUITPY
  ► 📁 .fseventsd
    📄 .metadata_never_index
    📄 .Trashes
    📄 boot_out.txt
    📄 code.py
  ▼ 📁 lib
```

```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials I2C possible pin-pair identifying script"""
import board
import busio
from microcontroller import Pin


def is_hardware_I2C(scl, sda):
    try:
        p = busio.I2C(scl, sda)
        p.deinit()
        return True
    except ValueError:
        return False
    except RuntimeError:
        return True


def get_unique_pins():
    exclude = ['NEOPIXEL', 'APA102_MOSI', 'APA102_SCK']
    pins = [pin for pin in [
        getattr(board, p) for p in dir(board) if p not in exclude]
            if isinstance(pin, Pin)]
    unique = []
    for p in pins:
        if p not in unique:
            unique.append(p)
    return unique


for scl_pin in get_unique_pins():
    for sda_pin in get_unique_pins():
        if scl_pin is sda_pin:
            continue
        if is_hardware_I2C(scl_pin, sda_pin):
            print("SCL pin:", scl_pin, "\t SDA pin:", sda_pin)
```

CircuitPython HID Keyboard and Mouse

One of the things we baked into CircuitPython is 'HID' (Human Interface Device) control - that means keyboard and mouse capabilities. This means your CircuitPython board can act like a keyboard device and press key commands, or a mouse and have it move the mouse pointer around and press buttons. This is really handy because even if you cannot adapt your software to work with hardware, there's almost always a keyboard interface - so if you want to have a capacitive touch interface for a game, say, then keyboard emulation can often get you going really fast!

This section walks you through the code to create a keyboard or mouse emulator. First we'll go through an example that uses pins on your board to emulate keyboard input. Then, we will show you how to wire up a joystick to act as a mouse, and cover the code needed to make that happen.

CircuitPython Keyboard Emulator

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 CircuitPython_Essentials/CircuitPython_HID_Keyboard/ 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:
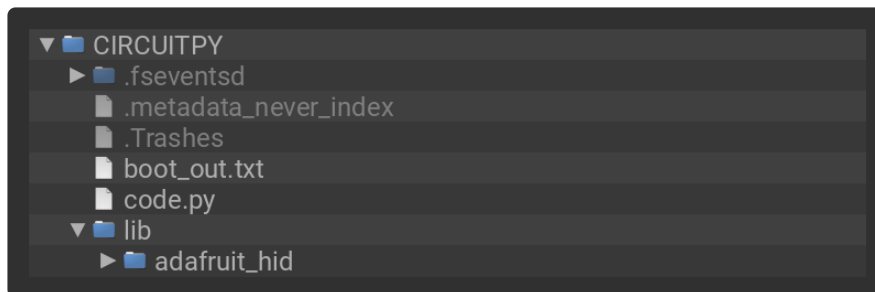

```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials HID Keyboard example"""
import time

import board
import digitalio
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS
from adafruit_hid.keycode import Keycode

# A simple neat keyboard demo in CircuitPython

# The pins we'll use, each will have an internal pullup
keypress_pins = [board.A1, board.A2]
# Our array of key objects
key_pin_array = []
# The Keycode sent for each button, will be paired with a control key
keys_pressed = [Keycode.A, "Hello World!\n"]
control_key = Keycode.SHIFT

# The keyboard object!
time.sleep(1)  # Sleep for a bit to avoid a race condition on some systems
keyboard = Keyboard(usb_hid.devices)
keyboard_layout = KeyboardLayoutUS(keyboard)  # We're in the US :)

# Make all pin objects inputs with pullups
for pin in keypress_pins:
    key_pin = digitalio.DigitalInOut(pin)
```

```python
        key_pin.direction = digitalio.Direction.INPUT
        key_pin.pull = digitalio.Pull.UP
        key_pin_array.append(key_pin)

# For most CircuitPython boards:
led = digitalio.DigitalInOut(board.LED)
# For QT Py M0:
# led = digitalio.DigitalInOut(board.SCK)
led.direction = digitalio.Direction.OUTPUT

print("Waiting for key pin...")

while True:
    # Check each pin
    for key_pin in key_pin_array:
        if not key_pin.value:  # Is it grounded?
            i = key_pin_array.index(key_pin)
            print("Pin #%d is grounded." % i)

            # Turn on the red LED
            led.value = True

            while not key_pin.value:
                pass  # Wait for it to be ungrounded!
            # "Type" the Keycode or string
            key = keys_pressed[i]  # Get the corresponding Keycode or string
            if isinstance(key, str):  # If it's a string...
                keyboard_layout.write(key)  # ...Print the string
            else:  # If it's not a string...
                keyboard.press(control_key, key)  # "Press"...
                keyboard.release_all()  # ..."Release"!

            # Turn off the red LED
            led.value = False

    time.sleep(0.01)
```
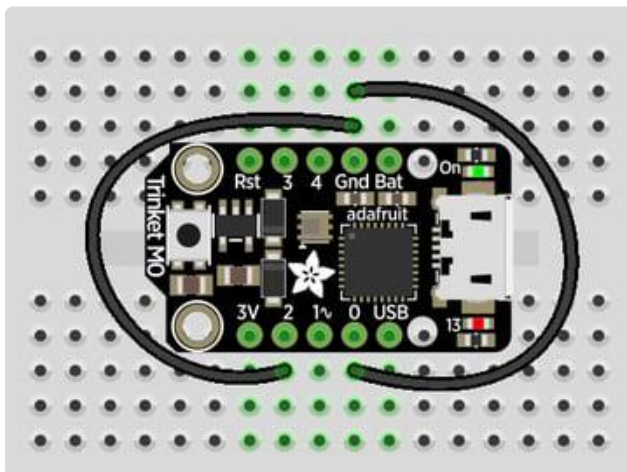
Connect pin A1 or A2 to ground, using a wire or alligator clip, then disconnect it to send the key press "A" or the string "Hello world!"

This wiring example shows A1 and A2 connected to ground.

Remember, on Trinket, A1 and A2 are labeled 2 and 0! On other boards, you will have A1 and A2 labeled as expected.

Create the Objects and Variables

First, we assign some variables for later use. We create three arrays assigned to variables: `keypress_pins` , `key_pin_array` , and `keys_pressed` . The first is the

pins we're going to use. The second is empty because we're going to fill it later. The third is what we would like our "keyboard" to output - in this case the letter "A" and the phrase, "Hello world!". We create our last variable assigned to `control_key` which allows us to later apply the shift key to our keypress. We'll be using two keypresses, but you can have up to six keypresses at once.

Next `keyboard` and `keyboard_layout` objects are created. We only have US right now (if you make other layouts please submit a GitHub pull request!). The `time.sleep(1)` avoids an error that can happen if the program gets run as soon as the board gets plugged in, before the host computer finishes connecting to the board.

Then we take the pins we chose above, and create the pin objects, set the direction and give them each a pullup. Then we apply the pin objects to `key_pin_array` so we can use them later.

Next we set up the little red LED to so we can use it as a status light.

The last thing we do before we start our loop is `print`, "Waiting for key pin..." so you know the code is ready and waiting!

The Main Loop

Inside the loop, we check each pin to see if the state has changed, i.e. you connected the pin to ground. Once it changes, it prints, "Pin # grounded." to let you know the ground state has been detected. Then we turn on the red LED. The code waits for the state to change again, i.e. it waits for you to unground the pin by disconnecting the wire attached to the pin from ground.

Then the code gets the corresponding keys pressed from our array. If you grounded and ungrounded A1, the code retrieves the keypress `a`, if you grounded and ungrounded A2, the code retrieves the string, `"Hello world!"`

If the code finds that it's retrieved a string, it prints the string, using the `keyboard_layout` to determine the keypresses. Otherwise, the code prints the keypress from the `control_key` and the keypress "a", which result in "A". Then it calls `keyboard.release_all()`. You always want to call this soon after a keypress or you'll end up with a stuck key which is really annoying!

Instead of using a wire to ground the pins, you can try wiring up buttons like we did in CircuitPython Digital In & Out (). Try altering the code to add more pins for more keypress options!

Non-US Keyboard Layouts

The code above uses KeyboardLayoutUS. If you would like to emulate a non-US keyboard, a number of other keyboard layout classes are available ().

CircuitPython Mouse Emulator

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 CircuitPython_Essentials/CircuitPython_HID_Mouse/ 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:
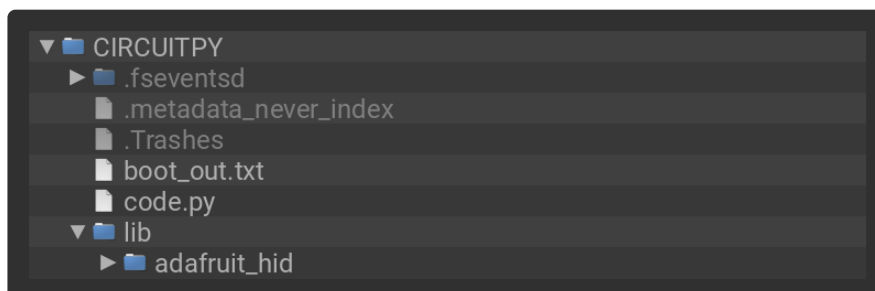

```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials HID Mouse example"""
import time
import analogio
import board
import digitalio
import usb_hid
from adafruit_hid.mouse import Mouse

mouse = Mouse(usb_hid.devices)

x_axis = analogio.AnalogIn(board.A0)
y_axis = analogio.AnalogIn(board.A1)
select = digitalio.DigitalInOut(board.A2)
select.direction = digitalio.Direction.INPUT
select.pull = digitalio.Pull.UP

pot_min = 0.00
pot_max = 3.29
```

```python
step = (pot_max - pot_min) / 20.0


def get_voltage(pin):
    return (pin.value * 3.3) / 65536


def steps(axis):
    """ Maps the potentiometer voltage range to 0-20 """
    return round((axis - pot_min) / step)


while True:
    x = get_voltage(x_axis)
    y = get_voltage(y_axis)

    if select.value is False:
        mouse.click(Mouse.LEFT_BUTTON)
        time.sleep(0.2)  # Debounce delay

    if steps(x) > 11.0:
        # print(steps(x))
        mouse.move(x=1)
    if steps(x) < 9.0:
        # print(steps(x))
        mouse.move(x=-1)

    if steps(x) > 19.0:
        # print(steps(x))
        mouse.move(x=8)
    if steps(x) < 1.0:
        # print(steps(x))
        mouse.move(x=-8)

    if steps(y) > 11.0:
        # print(steps(y))
        mouse.move(y=-1)
    if steps(y) < 9.0:
        # print(steps(y))
        mouse.move(y=1)

    if steps(y) > 19.0:
        # print(steps(y))
        mouse.move(y=-8)
    if steps(y) < 1.0:
        # print(steps(y))
        mouse.move(y=8)
```
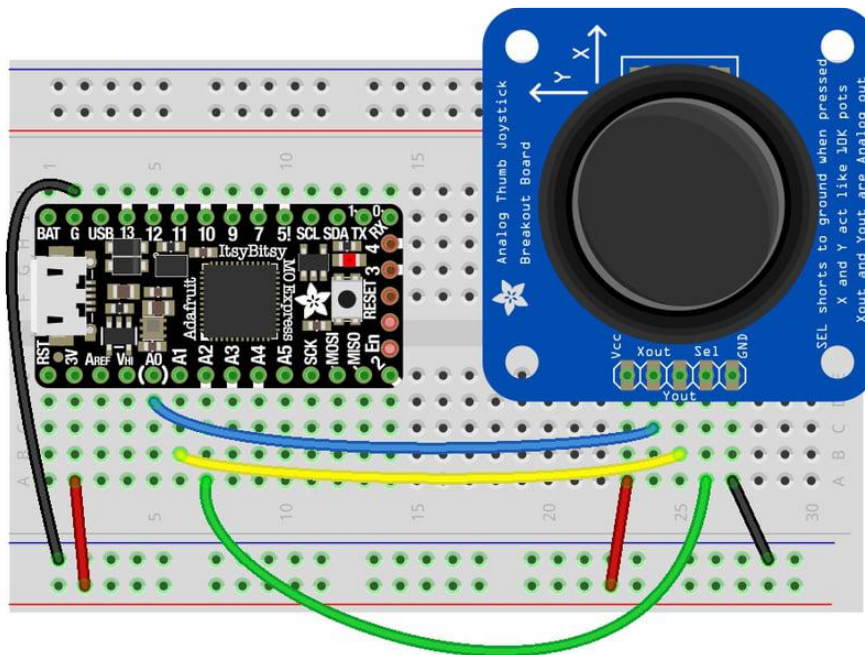
For this example, we've wired up a 2-axis thumb joystick with a select button. We use this to emulate the mouse movement and the mouse left-button click. To wire up this joytick:

- Connect VCC on the joystick to the 3V on your board. Connect ground to ground.
- Connect Xout on the joystick to pin A0 on your board.
- Connect Yout on the joystick to pin A1 on your board.
- Connect Sel on the joystick to pin A2 on your board.

Remember, Trinket's pins are labeled differently. Check the [Trinket Pinouts page ()](#) to verify your wiring.

To use this demo, simply move the joystick around. The mouse will move slowly if you move the joystick a little off center, and more quickly if you move it as far as it goes. Press down on the joystick to click the mouse. Awesome! Now let's take a look at the code.

Create the Objects and Variables

First we create the mouse object.

Next, we set `x_axis` and `y_axis` to pins `A0` and `A1`. Then we set `select` to `A2`, set it as input and give it a pullup.

The x and y axis on the joystick act like 2 potentiometers. We'll be using them just like we did in [CircuitPython Analog In ()](#). We set `pot_min` and `pot_max` to be the minimum and maximum voltage read from the potentiometers. We assign `step = (pot_max - pot_min) / 20.0` to use in a helper function.

CircuitPython HID Mouse Helpers

First we have the `get_voltage()` helper so we can get the correct readings from the potentiometers. Look familiar? We [learned about it in Analog In ()](#).

Second, we have `steps(axis)`. To use it, you provide it with the axis you're reading. This is where we're going to use the `step` variable we assigned earlier. The potentiometer range is 0-3.29. This is a small range. It's even smaller with the joystick because the joystick sits at the center of this range, 1.66, and the + and - of each axis is above and below this number. Since we need to have thresholds in our code, we're going to map that range of 0-3.29 to while numbers between 0-20.0 using this helper function. That way we can simplify our code and use larger ranges for our thresholds instead of trying to figure out tiny decimal number changes.

Main Loop

First we assign `x` and `y` to read the voltages from `x_axis` and `y_axis`.

Next, we check to see when the state of the select button is `False`. It defaults to `True` when it is not pressed, so if the state is `False`, the button has been pressed. When it's pressed, it sends the command to click the left mouse button. The `time.sleep(0.2)` prevents it from reading multiple clicks when you've only clicked once.

Then we use the `steps()` function to set our mouse movement. There are two sets of two `if` statements for each axis. Remember that `10` is the center step, as we've mapped the range `0-20`. The first set for each axis says if the joystick moves 1 step off center (left or right for the x axis and up or down for the y axis), to move the mouse the appropriate direction by 1 unit. The second set for each axis says if the joystick is moved to the lowest or highest step for each axis, to move the mouse the appropriate direction by 8 units. That way you have the option to move the mouse slowly or quickly!

To see what `step` the joystick is at when you're moving it, uncomment the `print` statements by removing the `#` from the lines that look like `# print(steps(x))`, and connecting to the serial console to see the output. Consider only uncommenting one set at a time, or you end up with a huge amount of information scrolling very quickly, which can be difficult to read!

For more detail check out the documentation at https://circuitpython.readthedocs.io/projects/hid/en/latest/

CircuitPython CPU Temp

There is a temperature sensor built into the CPU on your microcontroller board. It reads the internal CPU temperature, which varies depending on how long the board has been running or how intense your code is.

CircuitPython makes it really simple to read this data from the temperature sensor built into the microcontroller. Using the built-in `microcontroller` module, you can easily read the temperature.
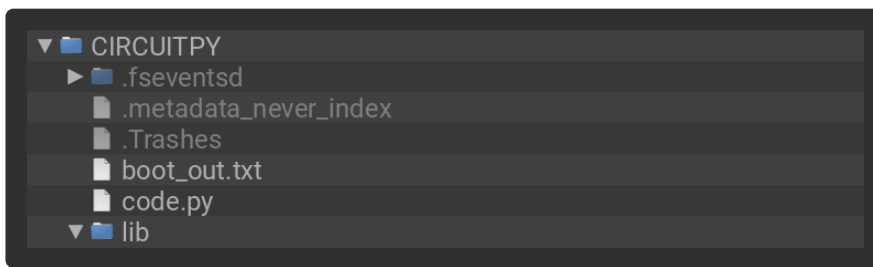
Microcontroller Location
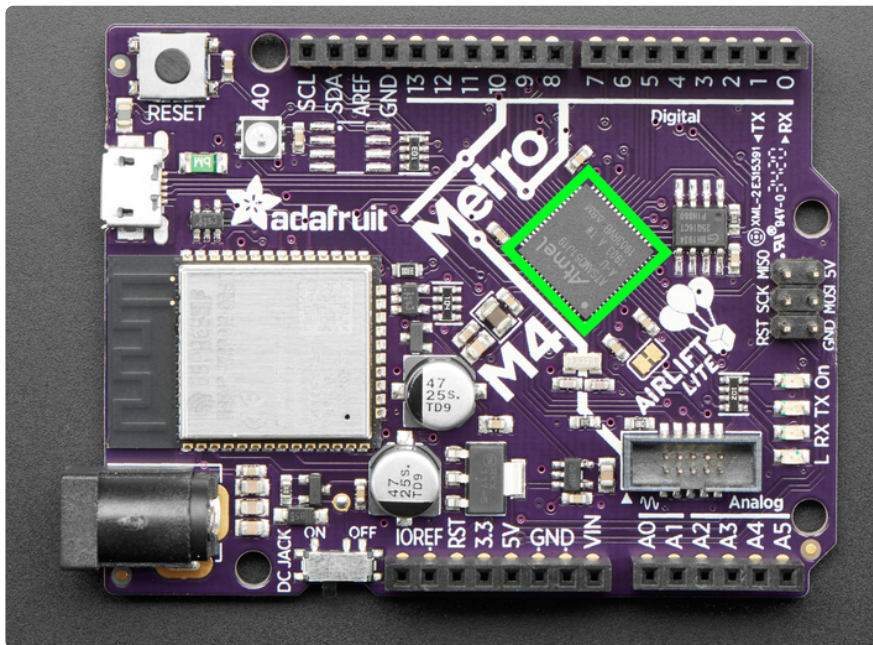
Reading the Microcontroller Temperature

The data is read using two lines of code. All necessary modules are built into CircuitPython, so you don't need to download any extra files to get started.

[Connect to the serial console]() (), and then update your code.py to the following.

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_Templates/cpu_temperature/ 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 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""CircuitPython CPU temperature example in Celsius"""
import time
import microcontroller

while True:
    print(microcontroller.cpu.temperature)
    time.sleep(0.15)
```
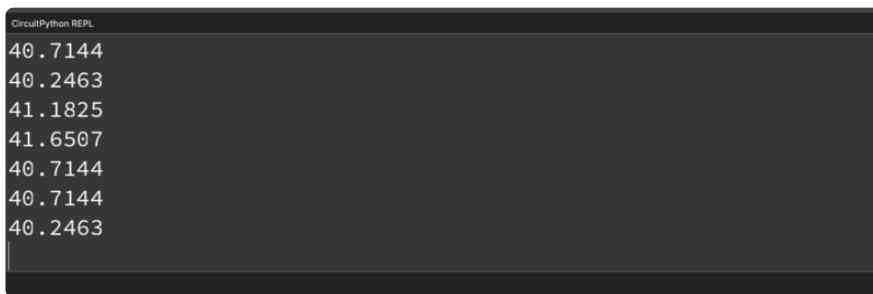

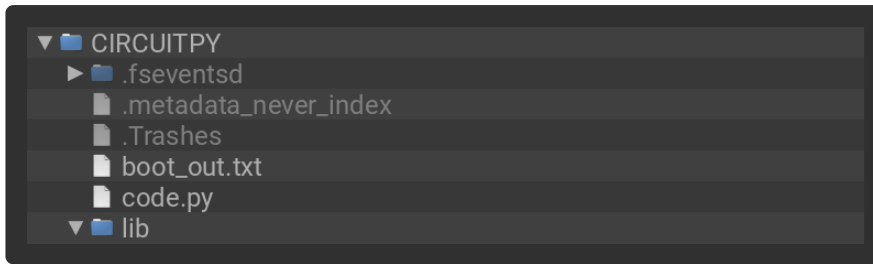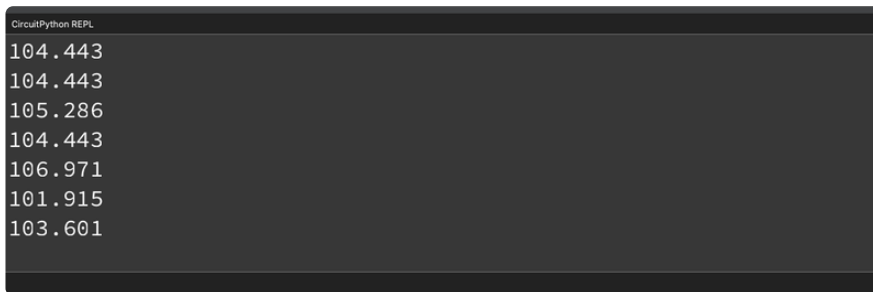The CPU temperature in Celsius is printed out to the serial console!

Try putting your finger on the microcontroller to see the temperature change.

The code is simple. First you import two modules: `time` and `microcontroller`. Then, inside the loop, you print the microcontroller CPU temperature, and the `time.sleep()` slows down the print enough to be readable. That's it!

You can easily print out the temperature in Fahrenheit by adding a little math to your code, using this simple formula: Celsius * (9/5) + 32.

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_Templates/cpu_temperature_f/ 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
  ▼ 📁 lib
```

```python
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""CircuitPython CPU temperature example in Fahrenheit"""
import time
import microcontroller

while True:
    print(microcontroller.cpu.temperature * (9 / 5) + 32)
    time.sleep(0.15)
```

```
CircuitPython REPL
104.443
104.443
105.286
104.443
106.971
101.915
103.601
```

The CPU temperature in Fahrenheit is printed out to the serial console!

That's all there is to reading the CPU temperature using CircuitPython!

CircuitPython Expectations

As we continue to develop CircuitPython and create new releases, we 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 we continue to develop CircuitPython and create new releases, we 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 we release new versions of CircuitPython, we 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 3.x or 2.x, where can I find compatible libraries?

We are no longer building or supporting the CircuitPython 2.x and 3.x 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, you can find the last available 2.x build here () and the last available 3.x build here ().

Switching Between CircuitPython and Arduino

Many of the CircuitPython boards also run Arduino. But how do you switch between the two? Switching between CircuitPython and Arduino is easy.

If you're currently running Arduino and would like to start using CircuitPython, follow the steps found in Welcome to CircuitPython: Installing CircuitPython ().

If you're currently running CircuitPython and would like to start using Arduino, plug in your board, and then load your Arduino sketch. If there are any issues, you can double tap the reset button to get into the bootloader and then try loading your sketch. Always backup any files you're using with CircuitPython that you want to save as they could be deleted.

That's it! It's super simple to switch between the two.

The Difference Between Express And Non-Express Boards

We often reference "Express" and "Non-Express" boards when discussing CircuitPython. What does this mean?

Express refers to the inclusion of an extra 2MB flash chip on the board that provides you with extra space for CircuitPython and your code. This means that we're able to include more functionality in CircuitPython and you're able to do more with your code on an Express board than you would on a non-Express board.

Express boards include Circuit Playground Express, ItsyBitsy M0 Express, Feather M0 Express, Metro M0 Express and Metro M4 Express.

Non-Express boards include Trinket M0, Gemma M0, QT Py, Feather M0 Basic, and other non-Express Feather M0 variants.

Non-Express Boards: Gemma, Trinket, and QT Py

CircuitPython runs nicely on the Gemma M0, Trinket M0, or QT Py M0 but there are some constraints

Small Disk Space

Since we use the internal flash for disk, and that's shared with runtime code, its limited! Only about 50KB of space.

No Audio or NVM

Part of giving up that FLASH for disk means we couldn't fit everything in. There is, at this time, no support for hardware audio playpack or NVM 'eeprom'. Modules `audioio` and `bitbangio` are not included. For that support, check out the Circuit Playground Express or other Express boards.

However, I2C, UART, capacitive touch, NeoPixel, DotStar, PWM, analog in and out, digital IO, logging storage, and HID do work! Check the CircuitPython Essentials for examples of all of these.

Differences Between CircuitPython and MicroPython

For the differences between CircuitPython and MicroPython, check out the [CircuitPython documentation ()](#).

Differences Between CircuitPython and Python

Python (also known as CPython) is the language that MicroPython and CircuitPython are based on. There are many similarities, but there are also many differences. This is a list of a few of the differences.

Python Libraries

Python is advertised as having "batteries included", meaning that many standard libraries are included. Unfortunately, for space reasons, many Python libraries are not available. So for instance while we wish you could `import numpy`, `numpy` isn't available (look for the `ulab` library for similar functions to `numpy` which works on many microcontroller boards). So you may have to port some code over yourself!

Integers in CircuitPython

On the non-Express boards, integers can only be up to 31 bits long. Integers of unlimited size are not supported. The largest positive integer that can be represented is 2^{30}-1, 1073741823, and the most negative integer possible is -2^{30}, -1073741824.

As of CircuitPython 3.0, Express boards have arbitrarily long integers as in Python.

Floating Point Numbers and Digits of Precision for Floats in CircuitPython

Floating point numbers are single precision in CircuitPython (not double precision as in Python). The largest floating point magnitude that can be represented is about +/-3.4e38. The smallest magnitude that can be represented with full accuracy is about +/-1.7e-38, though numbers as small as +/-5.6e-45 can be represented with reduced accuracy.

CircuitPython's floats have 8 bits of exponent and 22 bits of mantissa (not 24 like regular single precision floating point), which is about five or six decimal digits of precision.

Differences between MicroPython and Python

For a more detailed list of the differences between CircuitPython and Python, you can look at the MicroPython documentation. We keep up with MicroPython stable releases, so check out the core 'differences' they document here. ()

Arduino IDE Setup

The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using version 1.8 or higher for this guide

<div align="center">

Arduino IDE Download

</div>

After you have downloaded and installed the latest version of Arduino IDE, you will need to start the IDE and navigate to the Preferences menu. You can access it from the File menu in Windows or Linux, or the Arduino menu on OS X.
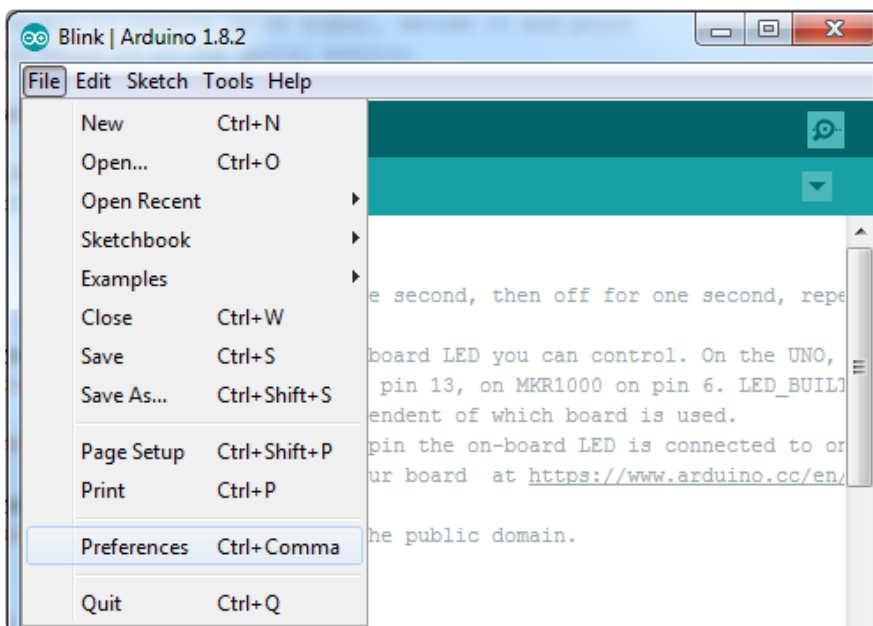
A dialog will pop up just like the one shown below.
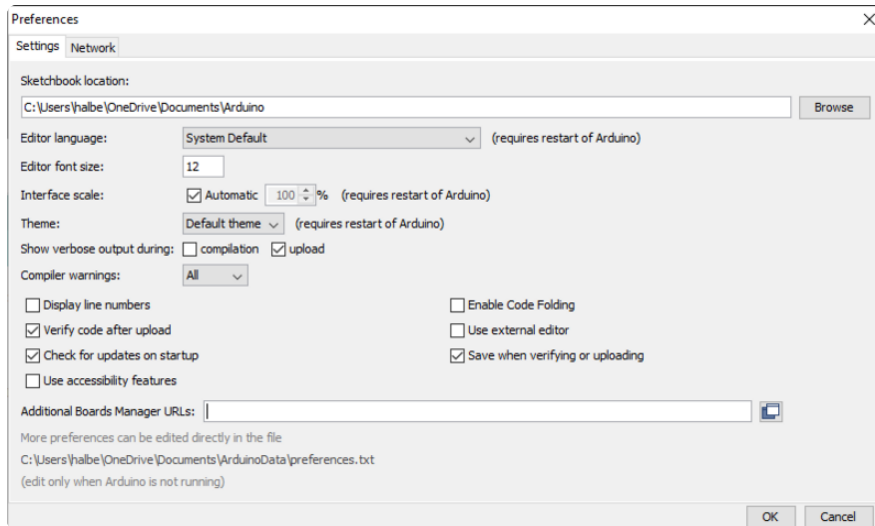
We will be adding a URL to the new Additional Boards Manager URLs option. The list of URLs is comma separated, and you will only have to add each URL once. New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

To find the most up to date list of URLs you can add, you can visit the list of third party board URLs on the Arduino IDE wiki (). We will only need to add one URL to the IDE in this example, but you can add multiple URLS by separating them with commas. Copy and paste the link below into the Additional Boards Manager URLs option in the Arduino IDE preferences.

```
https://adafruit.github.io/arduino-board-index/
package_adafruit_index.json
```
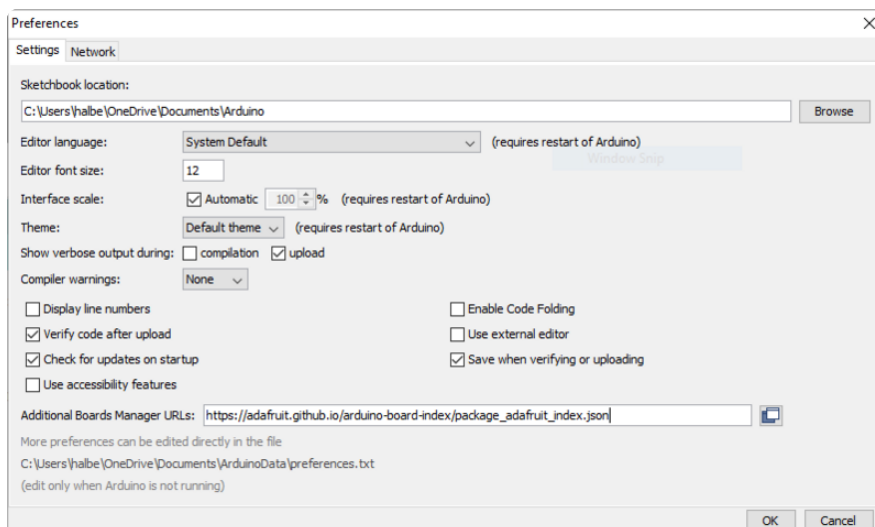
Here's a short description of each of the Adafruit supplied packages that will be available in the Board Manager when you add the URL:

- Adafruit AVR Boards - Includes support for Flora, Gemma, Feather 32u4, ItsyBitsy 32u4, Trinket, & Trinket Pro.
- Adafruit SAMD Boards - Includes support for Feather M0 and M4, Metro M0 and M4, ItsyBitsy M0 and M4, Circuit Playground Express, Gemma M0 and Trinket M0
- Arduino Leonardo & Micro MIDI-USB - This adds MIDI over USB support for the Flora, Feather 32u4, Micro and Leonardo using the arcore project ().

If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

Once done click OK to save the new preference settings. Next we will look at installing boards with the Board Manager.

Now continue to the next step to actually install the board support package!

Using with Arduino IDE

The Feather/Metro/Gemma/QTPy/Trinket M0 and M4 use an ATSAMD21 or ATSAMD51 chip, and you can pretty easily get it working with the Arduino IDE. Most libraries (including the popular ones like NeoPixels and display) will work with the M0 and M4, especially devices & sensors that use I2C or SPI.

Now that you have added the appropriate URLs to the Arduino IDE preferences in the previous page, you can open the Boards Manager by navigating to the Tools->Board menu.
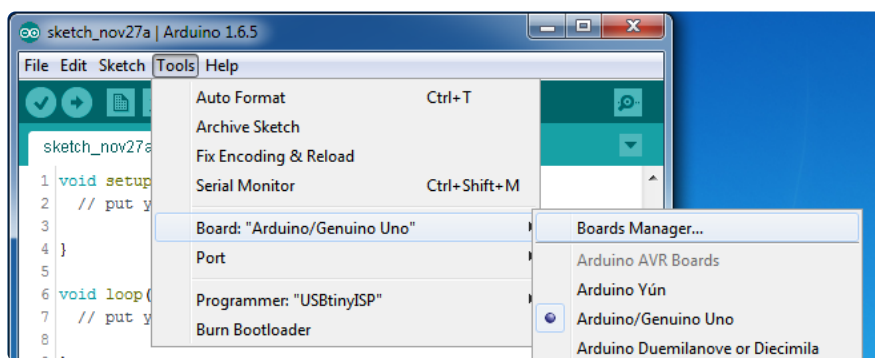
Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select All. You will then be able to select and install the boards supplied by the URLs added to the preferences.

Remember you need SETUP the Arduino IDE to support our board packages - see the previous page on how to add adafruit's URL to the preferences

Install SAMD Support

First up, install the latest Arduino SAMD Boards (version 1.6.11 or later)

You can type Arduino SAMD in the top search bar, then when you see the entry, click I nstall
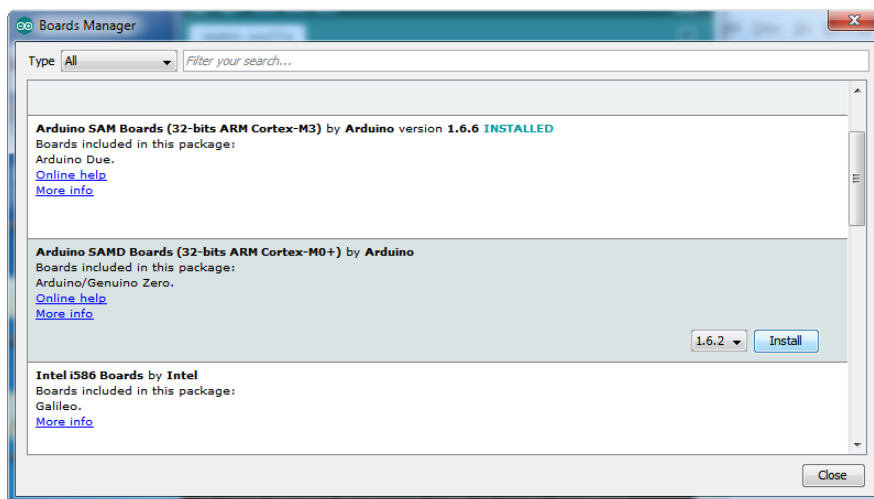
Install Adafruit SAMD

Next you can install the Adafruit SAMD package to add the board file definitions

Make sure you have Type All selected to the left of the Filter your search... box

You can type Adafruit SAMD in the top search bar, then when you see the entry, click I nstall
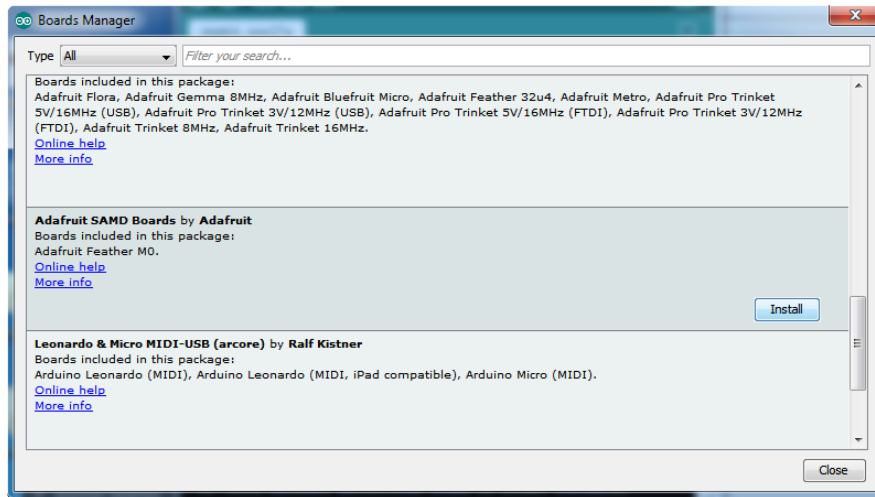
Even though in theory you don't need to - I recommend rebooting the IDE

Quit and reopen the Arduino IDE to ensure that all of the boards are properly installed. You should now be able to select and upload to the new boards listed in the Tools->Board menu.

Select the matching board, the current options are:

- Feather M0 (for use with any Feather M0 other than the Express)
- Feather M0 Express
- Metro M0 Express
- Circuit Playground Express
- Gemma M0
- Trinket M0
- QT Py M0
- ItsyBitsy M0
- Hallowing M0
- Crickit M0 (this is for direct programming of the Crickit, which is probably not what you want! For advanced hacking only)
- Metro M4 Express
- Grand Central M4 Express
- ItsyBitsy M4 Express
- Feather M4 Express
- Trellis M4 Express
- PyPortal M4
- PyPortal M4 Titano
- PyBadge M4 Express
- Metro M4 Airlift Lite
- PyGamer M4 Express
- MONSTER M4SK

- Hallowing M4
- MatrixPortal M4
- BLM Badge
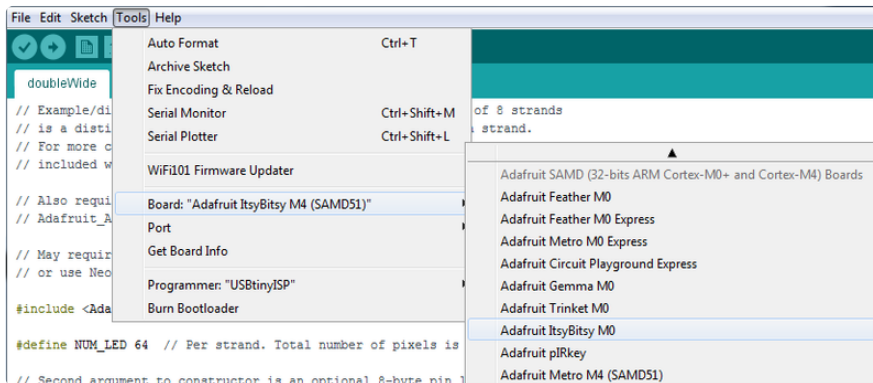
Install Drivers (Windows 7 & 8 Only)

When you plug in the board, you'll need to possibly install a driver

Click below to download our Driver Installer

<div align="center">

Download Latest Adafruit Drivers package
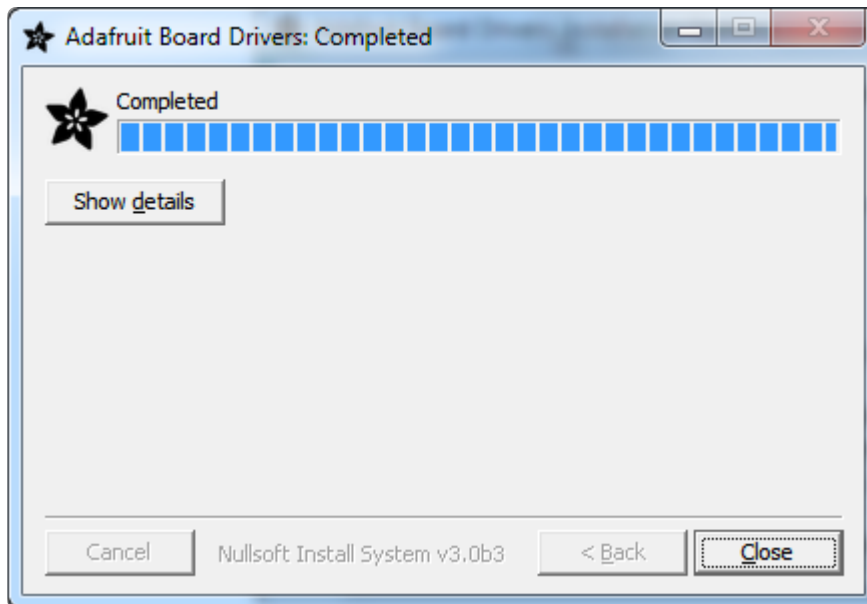
</div>
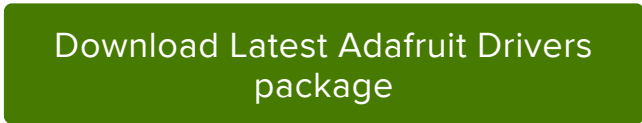
Download and run the installer
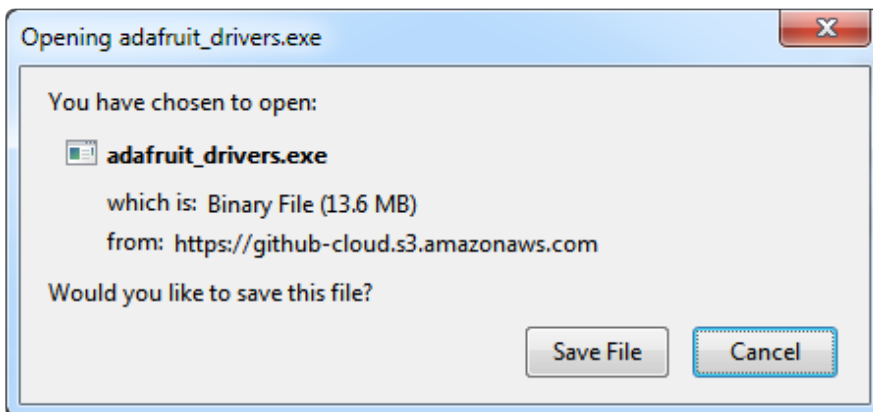
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license
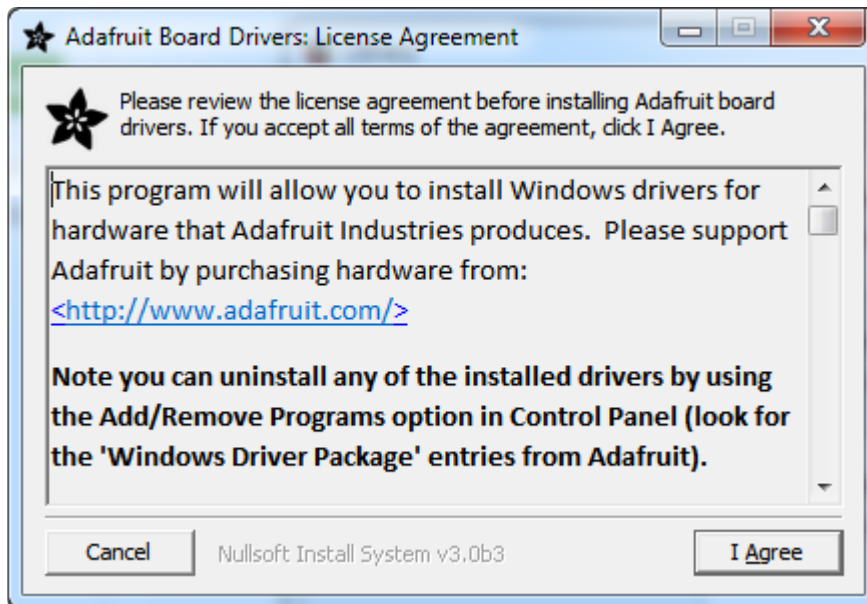
Select which drivers you want to install, the defaults will set you up with just about every Adafruit board!
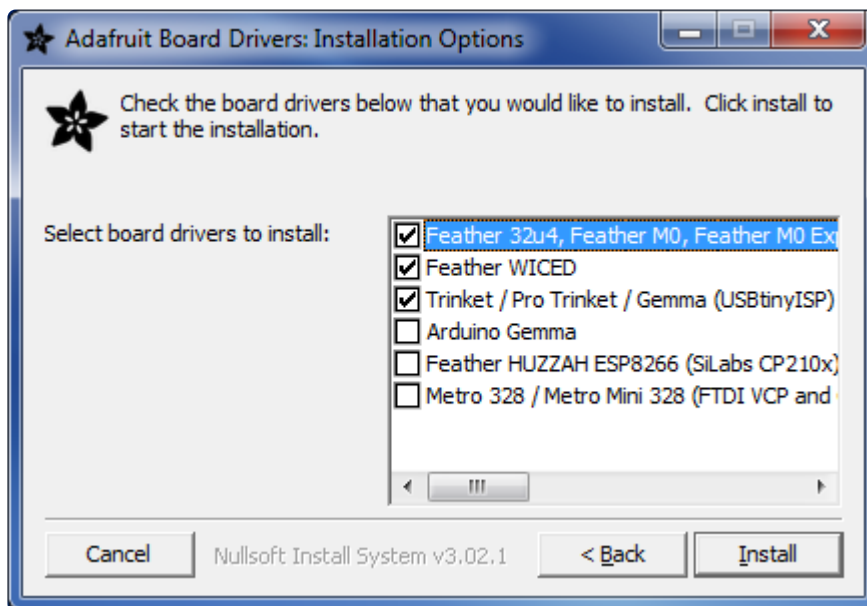
Click Install to do the installin'

Blink

Now you can upload your first blink sketch!

Plug in the M0 or M4 board, and wait for it to be recognized by the OS (just takes a few seconds). It will create a serial/COM port, you can now select it from the drop-down, it'll even be 'indicated' as Trinket/Gemma/Metro/Feather/ItsyBitsy/Trellis!

Please note, the QT Py and Trellis M4 Express are two of our very few boards that does not have an onboard pin 13 LED so you can follow this section to practice uploading but you wont see an LED blink!
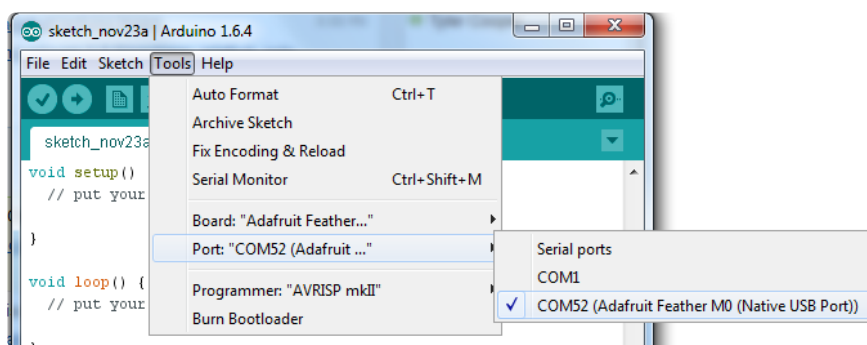
Now load up the Blink example

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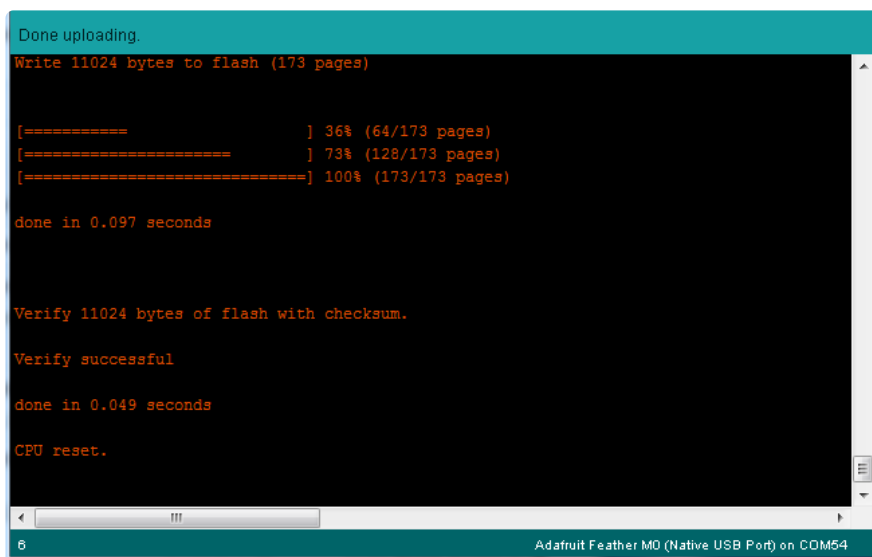
```
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

And click upload! That's it, you will be able to see the LED blink rate change as you adapt the delay() calls.

> If you are having issues, make sure you selected the matching Board in the menu that matches the hardware you have in your hand.

Successful Upload

If you have a successful upload, you'll get a bunch of red text that tells you that the device was found and it was programmed, verified & reset

After uploading, you may see a message saying "Disk Not Ejected Properly" about the ...BOOT drive. You can ignore that message: it's an artifact of how the bootloader and uploading work.

Compilation Issues

If you get an alert that looks like

Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-non-eabi-g++"

Make sure you have installed the Arduino SAMD boards package, you need both Arduino & Adafruit SAMD board packages
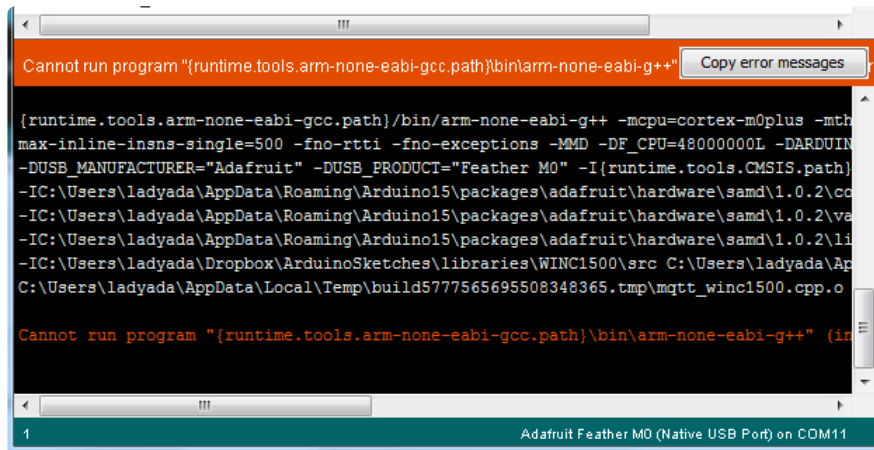

Manually bootloading

If you ever get in a 'weird' spot with the bootloader, or you have uploaded code that crashes and doesn't auto-reboot into the bootloader, click the RST button twice (like a double-click)to get back into the bootloader.

The red LED will pulse and/or RGB LED will be green, so you know that its in bootloader mode.

Once it is in bootloader mode, you can select the newly created COM/Serial port and re-try uploading.
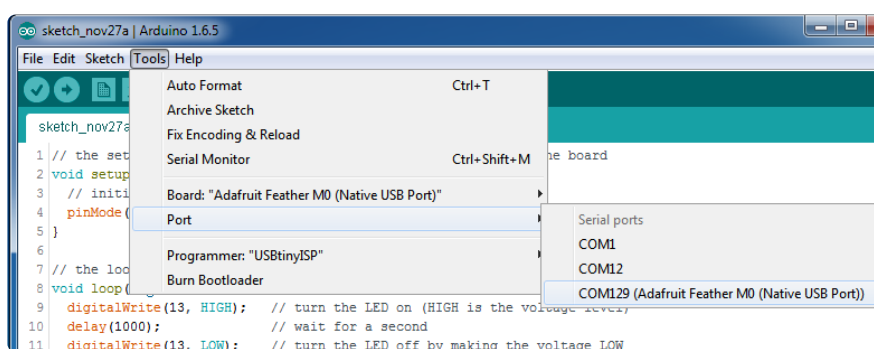
You may need to go back and reselect the 'normal' USB serial port next time you want to use the normal upload.

Ubuntu & Linux Issue Fix

Follow the steps for installing Adafruit's udev rules on this page. ()

Adapting Sketches to M0 & M4

The ATSAMD21 and 51 are very nice little chips, but fairly new as Arduino-compatible cores go. Most sketches & libraries will work but here's a collection of things we noticed.

The notes below cover a range of Adafruit M0 and M4 boards, but not every rule will apply to every board (e.g. Trinket and Gemma M0 do not have ARef, so you can skip the Analog References note!).

Analog References

If you'd like to use the ARef pin for a non-3.3V analog reference, the code to use is `analogReference(AR_EXTERNAL)` (it's AR_EXTERNAL not EXTERNAL)

Pin Outputs & Pullups

The old-style way of turning on a pin as an input with a pullup is to use

```
pinMode(pin, INPUT)
digitalWrite(pin, HIGH)
```

This is because the pullup-selection register on 8-bit AVR chips is the same as the output-selection register.

For M0 & M4 boards, you can't do this anymore! Instead, use:

```
pinMode(pin, INPUT_PULLUP)
```

Code written this way still has the benefit of being backwards compatible with AVR. You don't need separate versions for the different board types.

Serial vs SerialUSB

99.9% of your existing Arduino sketches use Serial.print to debug and give output. For the Official Arduino SAMD/M0 core, this goes to the Serial5 port, which isn't exposed on the Feather. The USB port for the Official Arduino M0 core is called SerialUSB instead.
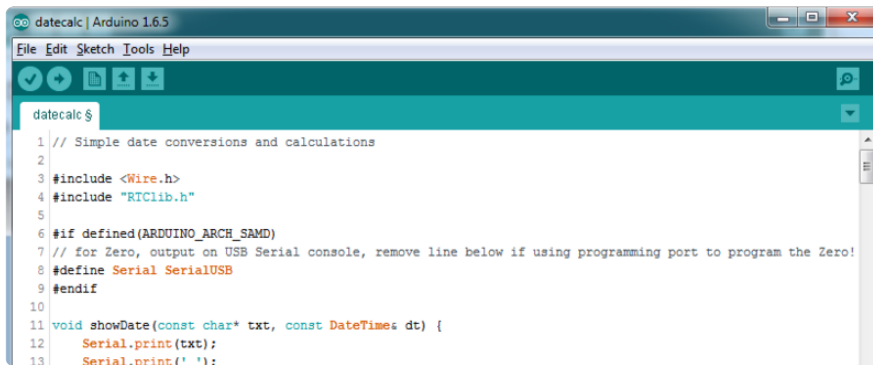
In the Adafruit M0/M4 Core, we fixed it so that Serial goes to USB so it will automatically work just fine.

However, on the off chance you are using the official Arduino SAMD core and not the Adafruit version (which really, we recommend you use our version because it's been tuned to our boards), and you want your Serial prints and reads to use the USB port, use SerialUSB instead of Serial in your sketch.

If you have existing sketches and code and you want them to work with the M0 without a huge find-replace, put

```
#if defined(ARDUINO_SAMD_ZERO) && defined(SERIAL_PORT_USBVIRTUAL)
  // Required for Serial on Zero based boards
  #define Serial SERIAL_PORT_USBVIRTUAL
#endif
```

right above the first function definition in your code. For example:

AnalogWrite / PWM on Feather/Metro M0

After looking through the SAMD21 datasheet, we've found that some of the options listed in the multiplexer table don't exist on the specific chip used in the Feather M0.

For all SAMD21 chips, there are two peripherals that can generate PWM signals: The Timer/Counter (TC) and Timer/Counter for Control Applications (TCC). Each SAMD21 has multiple copies of each, called 'instances'.

Each TC instance has one count register, one control register, and two output channels. Either channel can be enabled and disabled, and either channel can be inverted. The pins connected to a TC instance can output identical versions of the same PWM waveform, or complementary waveforms.

Each TCC instance has a single count register, but multiple compare registers and output channels. There are options for different kinds of waveform, interleaved switching, programmable dead time, and so on.

The biggest members of the SAMD21 family have five TC instances with two 'waveform output' (WO) channels, and three TCC instances with eight WO channels:

- TC[0-4],WO[0-1]
- TCC[0-2],WO[0-7]

And those are the ones shown in the datasheet's multiplexer tables.

The SAMD21G used in the Feather M0 only has three TC instances with two output channels, and three TCC instances with eight output channels:

- TC[3-5],WO[0-1]
- TCC[0-2],WO[0-7]

Tracing the signals to the pins broken out on the Feather M0, the following pins can't do PWM at all:

- Analog pin A5

The following pins can be configured for PWM without any signal conflicts as long as the SPI, I2C, and UART pins keep their protocol functions:

- Digital pins 5, 6, 9, 10, 11, 12, and 13
- Analog pins A3 and A4

If only the SPI pins keep their protocol functions, you can also do PWM on the following pins:

- TX and SDA (Digital pins 1 and 20)

analogWrite() PWM range

On AVR, if you set a pin's PWM with `analogWrite(pin, 255)` it will turn the pin fully HIGH. On the ARM cortex, it will set it to be 255/256 so there will be very slim but

still-existing pulses-to-0V. If you need the pin to be fully on, add test code that checks if you are trying to `analogWrite(pin, 255)` and, instead, does a `digitalWrite(pin, HIGH)`

analogWrite() DAC on A0

If you are trying to use `analogWrite()` to control the DAC output on A0, make sure you do not have a line that sets the pin to output. Remove: `pinMode(A0, OUTPUT)`.

Missing header files

There might be code that uses libraries that are not supported by the M0 core. For example if you have a line with

`#include <util/delay.h>`

you'll get an error that says

```
fatal error: util/delay.h: No such file or directory
  #include <util/delay.h>
                         ^
compilation terminated.
Error compiling.
```

In which case you can simply locate where the line is (the error will give you the file name and line number) and 'wrap it' with #ifdef's so it looks like:

```
#if !defined(ARDUINO_ARCH_SAM) && !defined(ARDUINO_ARCH_SAMD) && !
defined(ESP8266) && !defined(ARDUINO_ARCH_STM32F2)
  #include <util/delay.h>
#endif
```

The above will also make sure that header file isn't included for other architectures

If the #include is in the arduino sketch itself, you can try just removing the line.

Bootloader Launching

For most other AVRs, clicking reset while plugged into USB will launch the bootloader manually, the bootloader will time out after a few seconds. For the M0/M4, you'll need

to double click the button. You will see a pulsing red LED to let you know you're in bootloader mode. Once in that mode, it wont time out! Click reset again if you want to go back to launching code.

Aligned Memory Access

This is a little less likely to happen to you but it happened to me! If you're used to 8-bit platforms, you can do this nice thing where you can typecast variables around. e.g.

```
uint8_t mybuffer[4];
float f = (float)mybuffer;
```

You can't be guaranteed that this will work on a 32-bit platform because mybuffer might not be aligned to a 2 or 4-byte boundary. The ARM Cortex-M0 can only directly access data on 16-bit boundaries (every 2 or 4 bytes). Trying to access an odd-boundary byte (on a 1 or 3 byte location) will cause a Hard Fault and stop the MCU. Thankfully, there's an easy work around ... just use memcpy!

```
uint8_t mybuffer[4];
float f;
memcpy(&f, mybuffer, 4)
```

Floating Point Conversion

Like the AVR Arduinos, the M0 library does not have full support for converting floating point numbers to ASCII strings. Functions like sprintf will not convert floating point. Fortunately, the standard AVR-LIBC library includes the dtostrf function which can handle the conversion for you.

Unfortunately, the M0 run-time library does not have dtostrf. You may see some references to using #include <avr/dtostrf.h> to get dtostrf in your code. And while it will compile, it does not work.

Instead, check out this thread to find a working dtostrf function you can include in your code:

http://forum.arduino.cc/index.php?topic=368720.0 ()

How Much RAM Available?

The ATSAMD21G18 has 32K of RAM, but you still might need to track it for some reason. You can do so with this handy function:

```
extern "C" char *sbrk(int i);

int FreeRam () {
  char stack_dummy = 0;
  return &amp;stack_dummy - sbrk(0);
}
```

Thx to http://forum.arduino.cc/index.php?topic=365830.msg2542879#msg2542879 () for the tip!

Storing data in FLASH

If you're used to AVR, you've probably used PROGMEM to let the compiler know you'd like to put a variable or string in flash memory to save on RAM. On the ARM, its a little easier, simply add const before the variable name:

const char str[] = "My very long string";

That string is now in FLASH. You can manipulate the string just like RAM data, the compiler will automatically read from FLASH so you dont need special progmem-knowledgeable functions.

You can verify where data is stored by printing out the address: Serial.print("Address of str $"); Serial.println((int)&str, HEX);

If the address is $2000000 or larger, its in SRAM. If the address is between $0000 and $3FFFF Then it is in FLASH

Pretty-Printing out registers

There's a lot of registers on the SAMD21, and you often are going through ASF or another framework to get to them. So having a way to see exactly what's going on is handy. This library from drewfish will help a ton!

https://github.com/drewfish/arduino-ZeroRegs ()

M4 Performance Options

As of version 1.4.0 of the Adafruit SAMD Boards package in the Arduino Boards Manager, some options are available to wring extra performance out of M4-based devices. These are in the Tools menu.
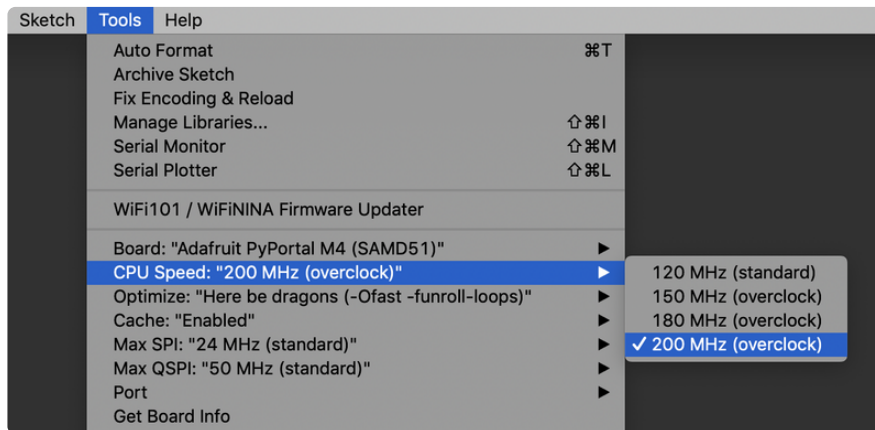
All of these performance tweaks involve a degree of uncertainty. There's no guarantee of improved performance in any given project, and some may even be detrimental, failing to work in part or in whole. If you encounter trouble, select the default performance settings and re-upload.

Here's what you get and some issues you might encounter...

CPU Speed (overclocking)

This option lets you adjust the microcontroller core clock...the speed at which it processes instructions...beyond the official datasheet specifications.

Manufacturers often rate speeds conservatively because such devices are marketed for harsh industrial environments...if a system crashes, someone could lose a limb or worse. But most creative tasks are less critical and operate in more comfortable settings, and we can push things a bit if we want more speed.

There is a small but nonzero chance of code locking up or failing to run entirely. If this happens, try dialing back the speed by one notch and re-upload, see if it's more stable.

Much more likely, some code or libraries may not play well with the nonstandard CPU speed. For example, currently the NeoPixel library assumes a 120 MHz CPU speed and won't issue the correct data at other settings (this will be worked on). Other

libraries may exhibit similar problems, usually anything that strictly depends on CPU timing…you might encounter problems with audio- or servo-related code depending how it's written. If you encounter such code or libraries, set the CPU speed to the default 120 MHz and re-upload.

Optimize

There's usually more than one way to solve a problem, some more resource-intensive than others. Since Arduino got its start on resource-limited AVR microcontrollers, the C++ compiler has always aimed for the smallest compiled program size. The "Optimize" menu gives some choices for the compiler to take different and often faster approaches, at the expense of slightly larger program size…with the huge flash memory capacity of M4 devices, that's rarely a problem now.

The "Small" setting will compile your code like it always has in the past, aiming for the smallest compiled program size.

The "Fast" setting invokes various speed optimizations. The resulting program should produce the same results, is slightly larger, and usually (but not always) noticably faster. It's worth a shot!

"Here be dragons" invokes some more intensive optimizations…code will be larger still, faster still, but there's a possibility these optimizations could cause unexpected behaviors. Some code may not work the same as before. Hence the name. Maybe you'll discover treasure here, or maybe you'll sail right off the edge of the world.

Most code and libraries will continue to function regardless of the optimizer settings. If you do encounter problems, dial it back one notch and re-upload.

Cache

This option allows a small collection of instructions and data to be accessed more quickly than from flash memory, boosting performance. It's enabled by default and should work fine with all code and libraries. But if you encounter some esoteric situation, the cache can be disabled, then recompile and upload.

Max SPI and Max QSPI

These should probably be left at their defaults. They're present mostly for our own experiments and can cause serious headaches.

Max SPI determines the clock source for the M4's SPI peripherals. Under normal circumstances this allows transfers up to 24 MHz, and should usually be left at that setting. But...if you're using write-only SPI devices (such as TFT or OLED displays), this option lets you drive them faster (we've successfully used 60 MHz with some TFT screens). The caveat is, if using any read/write devices (such as an SD card), this will not work at all...SPI reads absolutely max out at the default 24 MHz setting, and anything else will fail. Write = OK. Read = FAIL. This is true even if your code is using a lower bitrate setting...just having the different clock source prevents SPI reads.

Max QSPI does similarly for the extra flash storage on M4 "Express" boards. Very few Arduino sketches access this storage at all, let alone in a bandwidth-constrained context, so this will benefit next to nobody. Additionally, due to the way clock dividers are selected, this will only provide some benefit when certain "CPU Speed" settings are active. Our PyPortal Animated GIF Display () runs marginally better with it, if using the QSPI flash.

Enabling the Buck Converter on some M4 Boards

If you want to reduce power draw, some of our boards have an inductor so you can use the 1.8V buck converter instead of the built in linear regulator. If the board does have an inductor (see the schematic) you can add the line `SUPC->VREG.bit.SEL = 1;` to your code to switch to it. Note it will make ADC/DAC reads a bit noisier so we don't use it by default. You'll save ~4mA ().

WiFi Test

Library Install

We're using a variant of the Arduino WiFiNINA library, which is amazing and written by the Arduino team! We made a fork that you can install.

Click here to download the library:

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

Download Adafruit's version of WiFiNINA

</div>

Within the Arduino IDE, select Install library from ZIP...
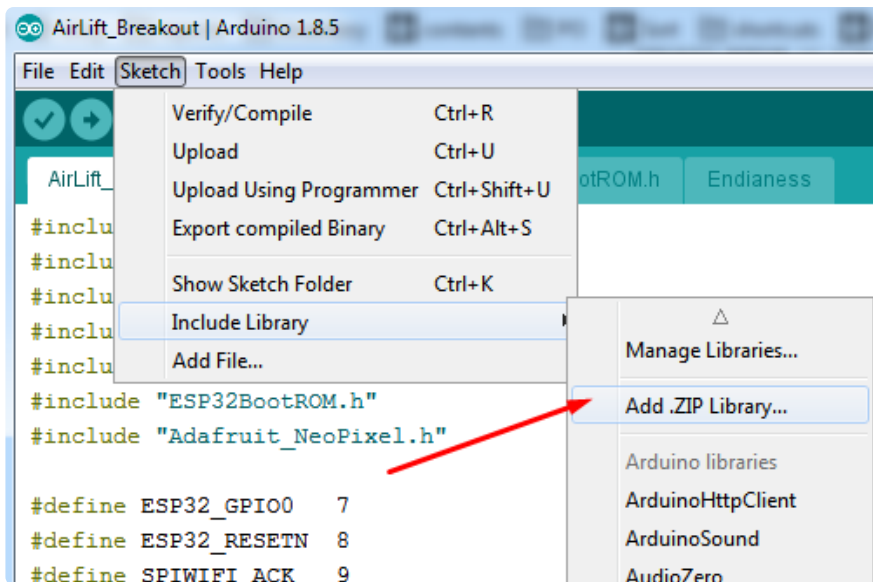
And select the zip you just downloaded.

First Test

OK now you have it wired and library installed, time to test it out!

Lets start by scanning the local networks. Load up the ScanNetworks example
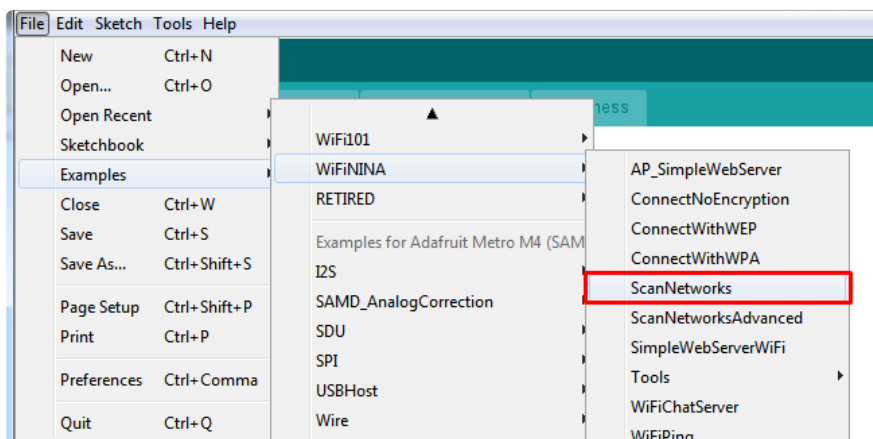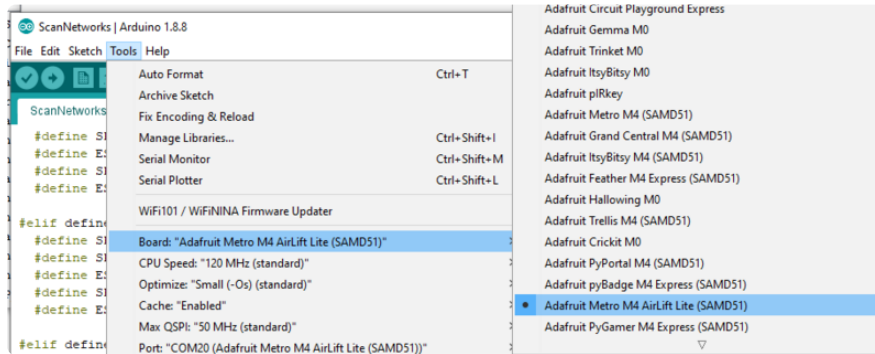

At the top you'll see a section where the GPIO pins are defined

```
// Configure the pins used for the ESP32 connection
#define SPIWIFI        SPI     // The SPI port
#define SPIWIFI_SS     10      // Chip select pin
#define SPIWIFI_ACK    7       // a.k.a BUSY or READY pin
#define ESP32_RESETN   5       // Reset pin
#define ESP32_GPIO0    -1      // Not connected
```

If you don't see this, you may have the wrong WiFiNINA library installed. Uninstall it and re-install the Adafruit one linked above.

You do not need to change any of these pins - they're pre-defined in the board definition for the Metro M4 WiFi but you still will want to have our version of the library installed because you can switch to the AirLift shield or breakout.

Make sure you select Metro M4 AirLift <u>not</u> Metro M4 Express!

Compile and upload to your board.

If you don't even get the MAC address printed out, check your wiring.

If you get the MAC address but cannot scan any networks, check your power supply. You need a solid 3-5VDC into Vin in order for the ESP32 not to brown out.

WiFi Connection Test

Now that you have your wiring checked, time to connect to the Internet!

Open up the WiFiWebClient example
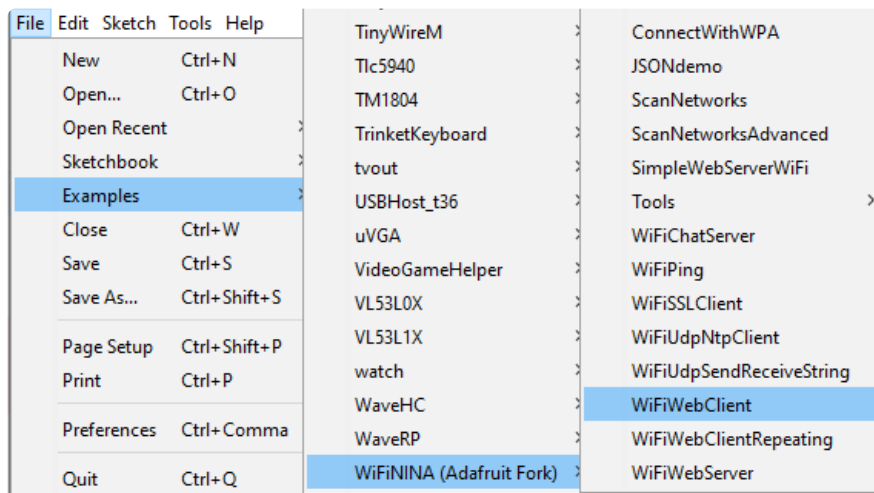
Open up the secondary tab, arduino_secrets.h. This is where you will store private data like the SSID/password to your network.

You must change these string values before updating to your board!

After you've set it correctly, upload and check the serial monitor. You should see the following. If not, go back, check wiring, power and your SSID/password
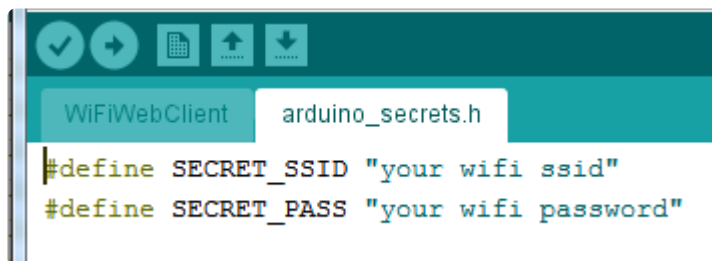
You must change these string values before updating to your board!

After you've set it correctly, upload and check the serial monitor. You should see the following. If not, go back, check wiring, power and your SSID/password

```
Found firmware 1.3.0
Attempting to connect to SSID: Adafruit
Connected to wifi
SSID: Adafruit
IP Address: 10.0.1.179
signal strength (RSSI):-44 dBm

Starting connection to server...
connected to server
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 10 Apr 2019 20:55:51 GMT
Content-Type: text/html
Content-Length: 73
Last-Modified: Thu, 16 Feb 2017 17:42:29 GMT
Connection: close
ETag: "58a5e485-49"
Accept-Ranges: bytes

This is a test of the CC3000 module!
If you can read this, its working :)
disconnecting from server.
```
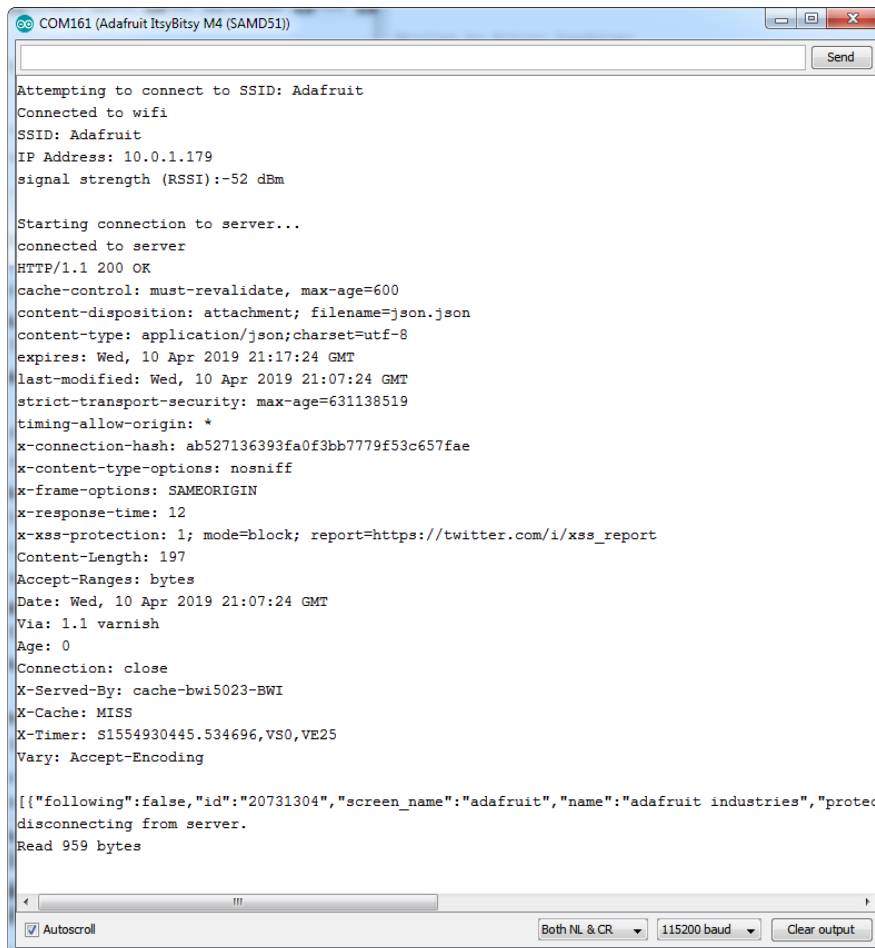
Secure Connection Example

Many servers today do not allow non-SSL connectivity. Lucky for you the ESP32 has a great TLS/SSL stack so you can have that all taken care of for you. Here's an example of a secure WiFi connection:
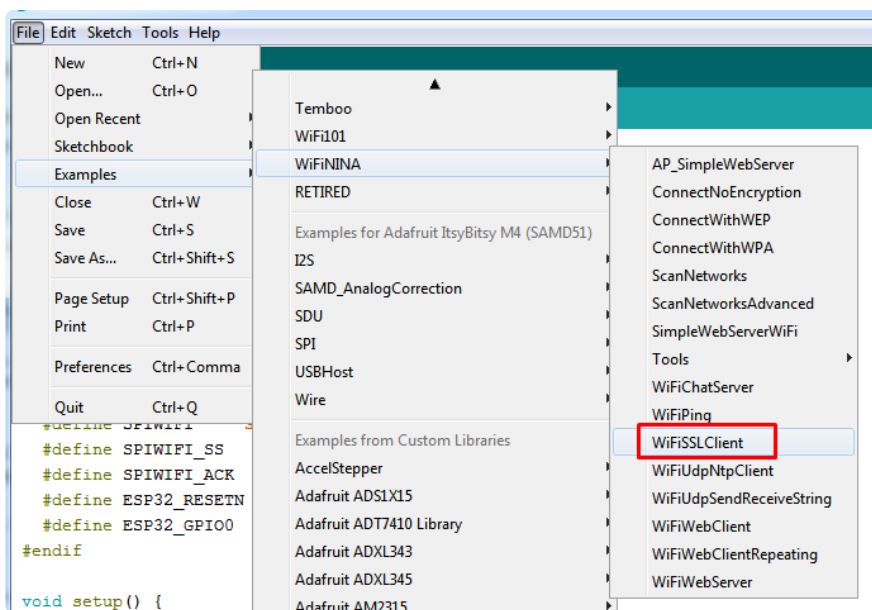
Note we use `WiFiSSLClient client;` instead of `WiFiClient client;` to require an SSL connection!

JSON Parsing Demo

This example is a little more advanced - many sites will have API's that give you JSON data. We'll use ArduinoJSON () to convert that to a format we can use and then display that data on the serial port (which can then be re-directed to a display of some sort)

First up, use the Library manager to install ArduinoJSON ().

Then load the example JSONdemo
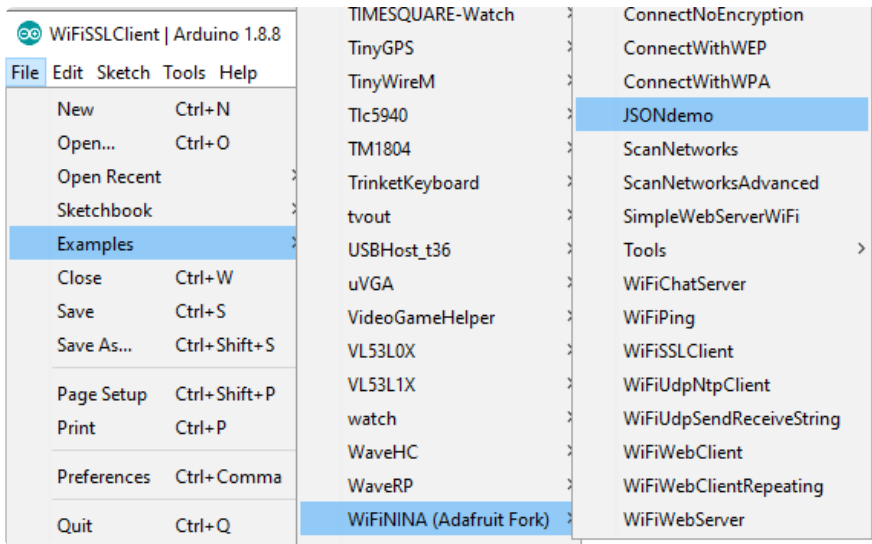
By default it will connect to to the Twitter banner image API, parse the username and followers and display them.

```
Attempting to connect to SSID: Adafruit
Connected to wifi
SSID: Adafruit
IP Address: 10.0.1.179
signal strength (RSSI):-51 dBm

Starting connection to server...
connected to server
Response:
Twitter username: adafruit
Twitter followers: 159265
```

Updating ESP32 Firmware

There may come a time when you want to update the firmware on the ESP32 itself. This isn't something we expect you'll do often if at all, but its good to know how if you need to.

We have a guide here which details the process of updating the ESP32 firmware on Airlift All-in-One boards (including the PyPortal, MatrixPortal, and Metro M4 AirLift) here... ()
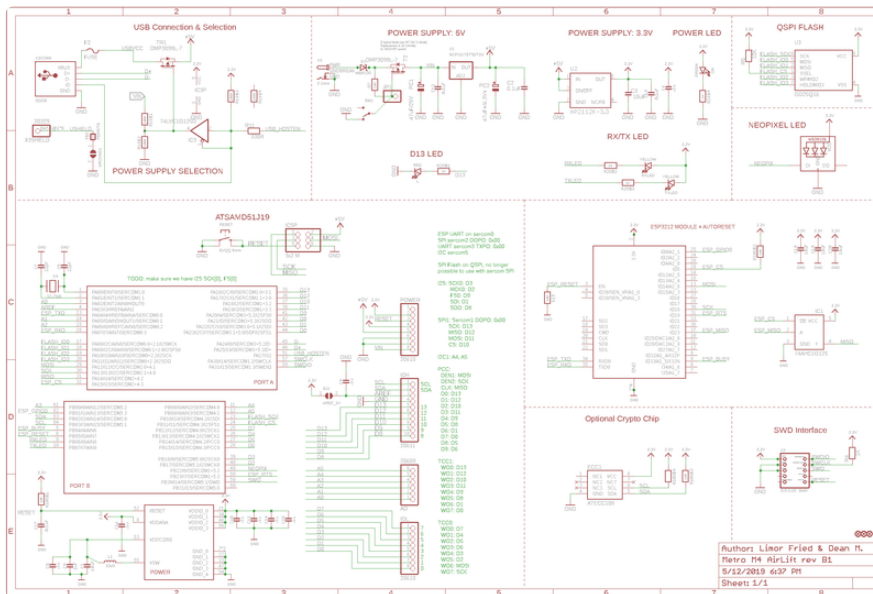
Downloads

Files

- [ATSAMD51J19 Product page w/datasheets](#) () (the main chip on the Metro M4 Express AirLift Lite)
- [Fritzing Object in the Adafruit Fritzing Library](#) ()
- [EagleCAD files on GitHub](#) ()
- [Default files shipped with Metro M4 Express AirLift](#) ()
- [PDF for Metro M4 Express AirLift Board Diagram on GitHub](#) ()

> ### SVG for Metro M4 Express AirLift Board Diagram

Schematic

Fab Print