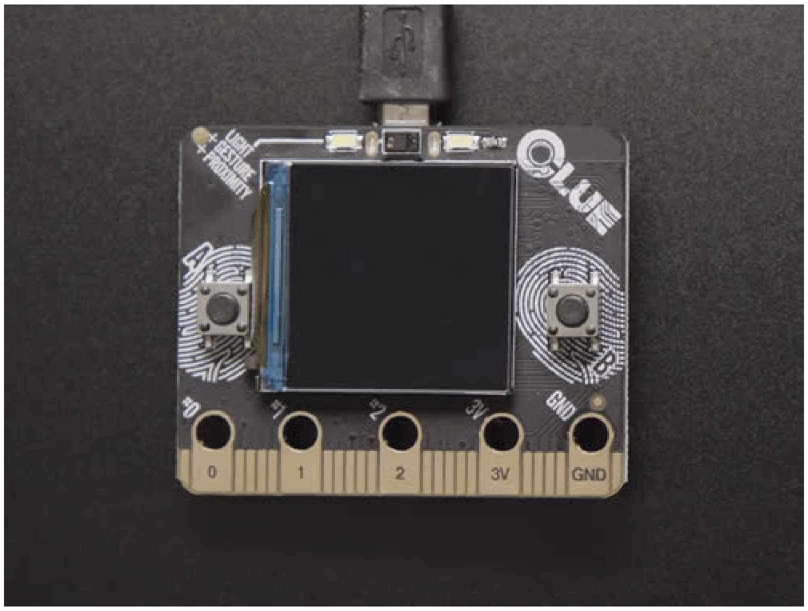
Introducing Adafruit CLUE

Created by Kattni Rembor



Last updated on 2021-08-19 10:44:43 AM EDT

Guide Contents

Guide Contents 2
Overview 7
Pinouts 10
Microcontroller and QSPI 11
Display 12
Sensors 12
USB and Battery 14
Buttons 15
STEMMA QT 15
LEDs 15
GPIO and Power Pads 16
Edge Connector 17
- Debug Pads 18

Powering Your CLUE 19
micro:bit Power 19
CLUE Power 19
HELP! Accel/Gyro Not Working? 21
Arduino Support Setup 22
1. BSP Installation 22
- Recommended: Installing the BSP via the Board Manager 22

2. LINUX ONLY: adafruit-nrfutil Tool Installation 23
3. Update the bootloader (nRF52832 ONLY) 24
Advanced Option: Manually Install the BSP via 'git' 24
- Adafruit nRF52 BSP via git (for core development and PRs only) 24

Arduino Board Testing 26
1. Select the Board Target 26
2. Select the USB CDC Serial Port 26
- Download & Install CP2104 Driver (nRF52832) 26
- Download & Install Adafruit Driver (nRF52840 Windows) 27

3. Update the bootloader (nRF52832 Feather Only) 27
4. Run a Test Sketch 28
- If Arduino failed to upload sketch to the Feather 28
  - If you get this error: 28
- On Linux I'm getting 'arm-none-eabi-g++: no such file or directory', even though 'arm-none-eabi-g++' exists in the path specified. What should I do? 29

Arcada Libraries 31
Install Libraries 31
- Adafruit Arcada 31
- If you aren't running Arduino IDE 1.8.10 or later, you'll need to install all of the following! 31
- Adafruit NeoPixel 32

Adafruit FreeTouch 32
Adafruit Touchscreen 32
- Adafruit SPIFlash 32
- Adafruit Zero DMA 32
- Adafruit GFX 33
- Adafruit ST7735 33
- Adafruit ILI9341 33

- Adafruit LIS3DH 33
- Adafruit Sensor 34

Adafruit ImageReader 34

ArduinoJson 34

Adafruit ZeroTimer 34

Adafruit TinyUSB 35

Adafruit WavePlayer 35

- SdFat (Adafruit Fork) 35

Audio - Adafruit Fork 35

Sensor Libraries 36

- Adafruit Sensor Lab 36
- If you aren't running Arduino IDE 1.8.10 or later, you'll need to install all of the following! 36
- Adafruit Unified Sensor 36
- Adafruit ADXL343 36
- Adafruit APDS9660 37
- Adafruit BMP280 37
- Adafruit BME280 37
- Adafruit DPS310 37
- Adafruit LIS2MDL 37
- Adafruit LIS3MDL 37
- Adafruit LSM6DS 38
- Adafruit MSA301 38
- Adafruit SHT31 38
- Adafruit AHRS & Adafruit Sensor Calibration 38

Arduino Test 39

Animated GIF Player 41

Arduino Bluefruit nRF52 API 42

Arduino BLE Examples 43

CircuitPython on CLUE 44

- Set up CircuitPython Quick Start! 44

CLUE CircuitPython Libraries 46

Installing CircuitPython Libraries on your CLUE 46

Getting Started with BLE and CircuitPython 48

Guides 48

Installing Mu Editor 50

Download and Install Mu 50

Using Mu 50

Creating and Editing Code 52

- Creating Code 52
- Editing Code 54
  - Your code changes are run as soon as the file is done saving. 55
- 1. Use an editor that writes out the file completely when you save it. 55
- 2. Eject or Sync the Drive After Writing 56
- Oh No I Did Something Wrong and Now The CIRCUITPY Drive Doesn't Show Up!!! 56

Back to Editing Code... 57

Exploring Your First CircuitPython Program 58

- Imports & Libraries 58
- Setting Up The LED 58
- Loop-de-loops 59
- What Happens When My Code Finishes Running? 59
- What if I don't have the loop? 60

More Changes 61
Naming Your Program File 61
Connecting to the Serial Console 62
Are you using Mu? 62
- Setting Permissions on Linux 63

Using Something Else? 64
Interacting with the Serial Console 65
The REPL 69
Returning to the serial console 72
CircuitPython Libraries 73
- Installing the CircuitPython Library Bundle 74
- Example Files 75

Copying Libraries to Your Board 76
- Example: ImportError Due to Missing Library 76
- Library Install on Non-Express Boards 78
- Updating CircuitPython Libraries/Examples 78

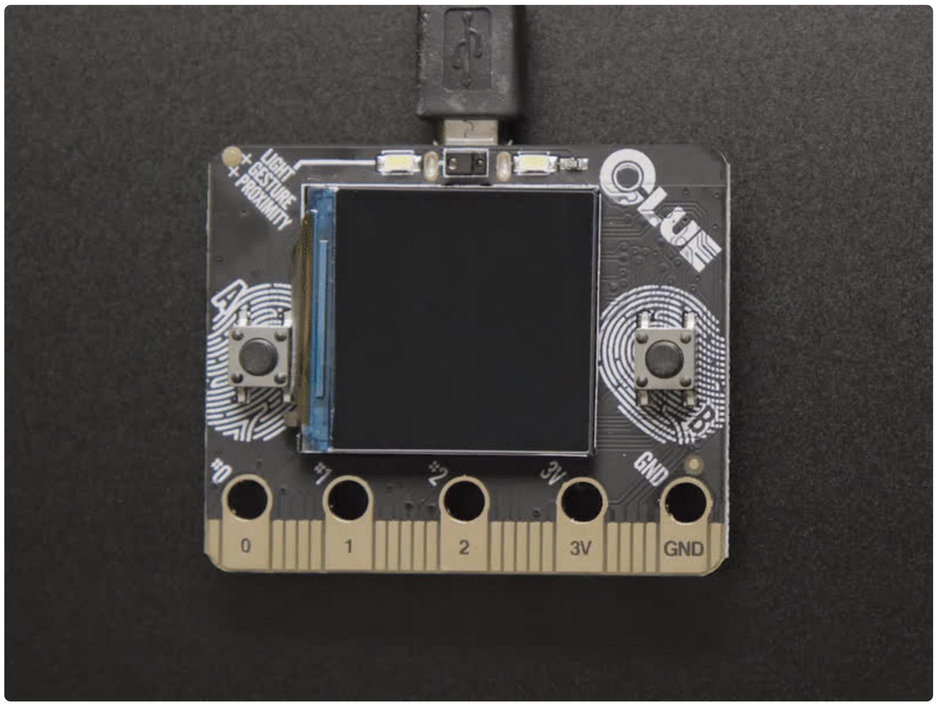
CircuitPython Pins and Modules 79
CircuitPython Pins 79
- import board 79
- I2C, SPI, and UART 80
- What Are All the Available Names? 81
- Microcontroller Pin Names 82

CircuitPython Built-In Modules 83
Advanced Serial Console on Mac and Linux 84
What's the Port? 84
Connect with screen 86
Permissions on Linux 87
Advanced Serial Console on Windows 90
Windows 7 Driver 90
What's the COM? 90
Install Putty 91
Welcome to the Community! 93
- Adafruit Discord 93
- Adafruit Forums 94
- Adafruit Github 95
- ReadTheDocs 96

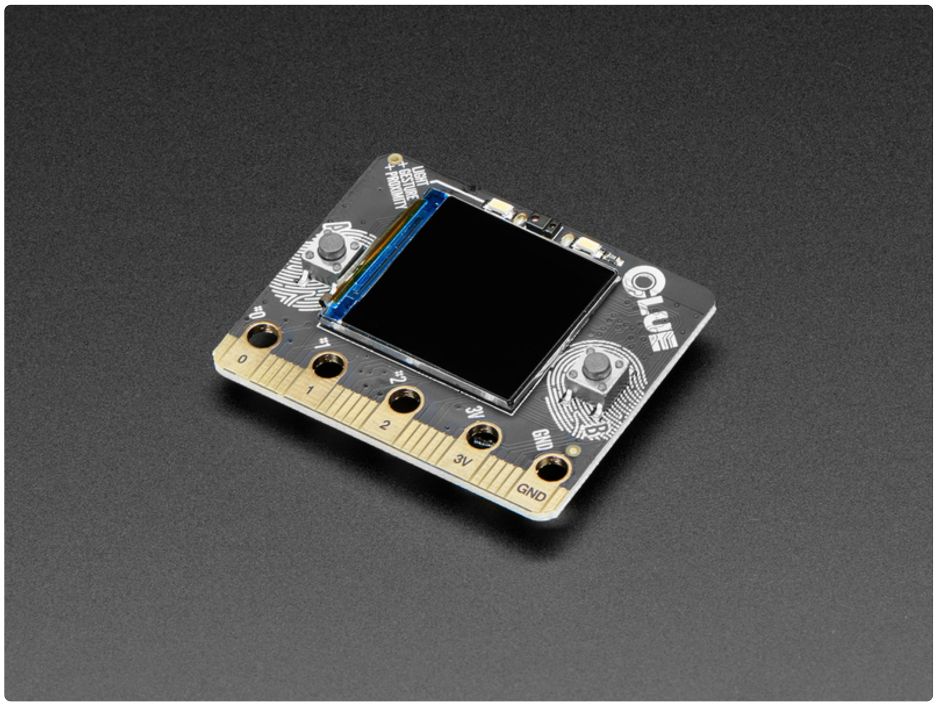
Frequently Asked Questions 98
- I have to continue using an older version of CircuitPython; where can I find compatible libraries? 98
- Is ESP8266 or ESP32 supported in CircuitPython? Why not? 98
- How do I connect to the Internet with CircuitPython? 99
- Is there asyncio support in CircuitPython? 100
- My RGB NeoPixel/DotStar LED is blinking funny colors - what does it mean? 101
- What is a MemoryError? 102
- What do I do when I encounter a MemoryError? 102
- Can the order of my import statements affect memory? 103
- How can I create my own .mpy files? 103
- How do I check how much memory I have free? 103
- Does CircuitPython support interrupts? 103
- Does Feather M0 support WINC1500? 104
- Can AVRs such as ATmega328 or ATmega2560 run CircuitPython? 104

Commonly Used Acronyms 104
Troubleshooting 105
Always Run the Latest Version of CircuitPython and Libraries 105
I have to continue using CircuitPython 5.x, 4.x, 3.x or 2.x, where can I find compatible libraries? 105
CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present 105
You may have a different board. 106
MakeCode 106
MacOS 106
Windows 10 106
Windows 7 or 8.1 106
Windows Explorer Locks Up When Accessing boardnameBOOT Drive 107
Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied 108
CIRCUITPY Drive Does Not Appear 108
Windows 7 and 8.1 Problems 108
Serial Console in Mu Not Displaying Anything 108
CircuitPython RGB Status Light 109
ValueError: Incompatible .mpy file. 110
CIRCUITPY Drive Issues 110
Easiest Way: Use storage.erase_filesystem() 111
Old Way: For the Circuit Playground Express, Feather M0 Express, and Metro M0 Express: 111
Old Way: For Non-Express Boards with a UF2 bootloader (Gemma M0, Trinket M0): 113
Old Way: For non-Express Boards without a UF2 bootloader (Feather M0 Basic Proto, Feather Adalogger, Arduino Zero): 113
Running Out of File Space on Non-Express Boards 113
Delete something! 113
Use tabs 114
MacOS loves to add extra files. 114
Prevent & Remove MacOS Hidden Files 114
Copy Files on MacOS Without Creating Hidden Files 115
Other MacOS Space-Saving Tips 116
Device locked up or boot looping 117
Uninstalling CircuitPython 118
Backup Your Code 118
Moving Circuit Playground Express to MakeCode 118
Moving to Arduino 119
CircuitPython Essentials 122
Clue Library Documentation 123
CLUE CircuitPython Demos 124
CLUE Spirit Level 125
CLUE Temperature and Humidity Monitor 128
CLUE Height Calculator 132
CLUE Slideshow 135
Blinka Bitmaps 135
CircuitPython Libraries 136
CLUE Slideshow Example 136
Downloads 138
Schematic 138
Fab Print 139

Overview

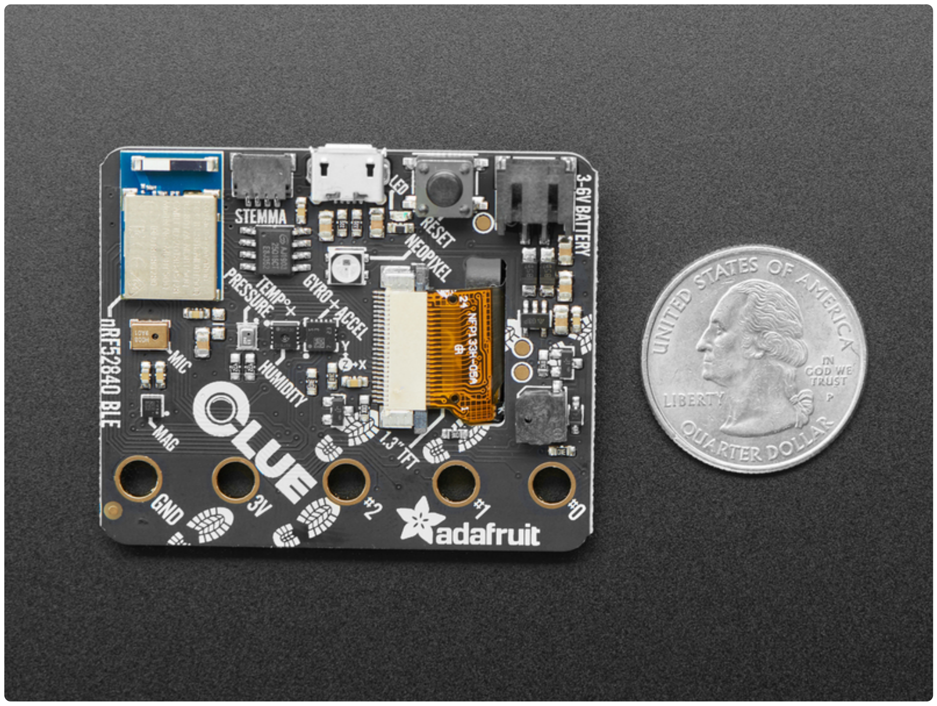


Do you feel like you just don't have a CLUE? Well, we can help with that - get a CLUE here at Adafruit by picking up this sensor-packed development board. We wanted to build some projects that have a small screen and a lot of sensors. To make it compatible with existing projects, we made it the same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



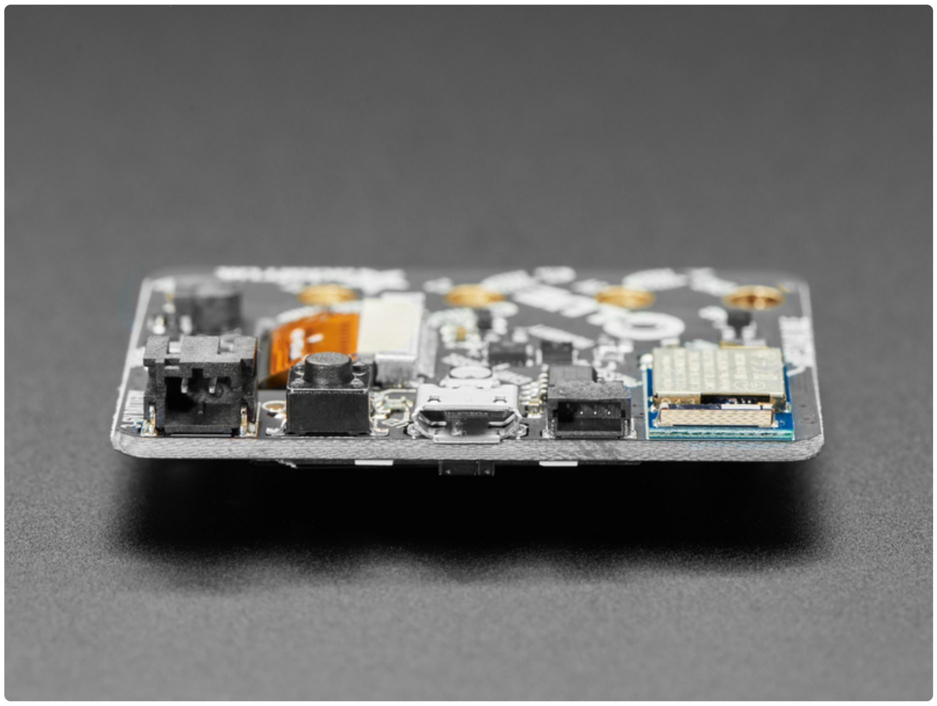
While the CLUE looks a bit like a 'bit it has totally redesigned-from-scratch technology:

- **Nordic nRF52840 Bluetooth LE processor** - 1 MB of Flash, 256KB RAM, 64 MHz Cortex M4 processor
- **1.3″ 240×240 Color IPS TFT display** for high resolution text and graphics
- **Power it from any 3-6V battery source** (internal regulator and protection diodes)
- **Two A / B user buttons** and one reset button
- Tons of sensors!
    - ST Micro series 9-DoF motion - LSM6DS33 Accel/Gyro (https://adafru.it/IfN) + LIS3MDL (https://adafru.it/Iqa) magnetometer (https://adafru.it/IHC)
    - APDS9960 Proximity, Light, Color, and Gesture Sensor (https://adafru.it/IHD)
    - PDM Microphone sound sensor (https://adafru.it/FB0)
    - SHT Humidity (https://adafru.it/IHE)
    - BMP280 temperature and barometric pressure/altitude (https://adafru.it/ufr)
- **RGB NeoPixel** indicator LED
- **2 MB internal flash storage** for datalogging, images, fonts or CircuitPython code
- **Buzzer/speaker** for playing tones and beeps
- **Two bright white LEDs** in front for illumination / color sensing.
- **Qwiic / STEMMA QT connector** for adding more sensors, motor controllers, or displays over I2C. You can plug in GROVE I2C sensors by using an adapter cable (https://adafru.it/IDk).
- **Programmable with Arduino IDE or CircuitPython**

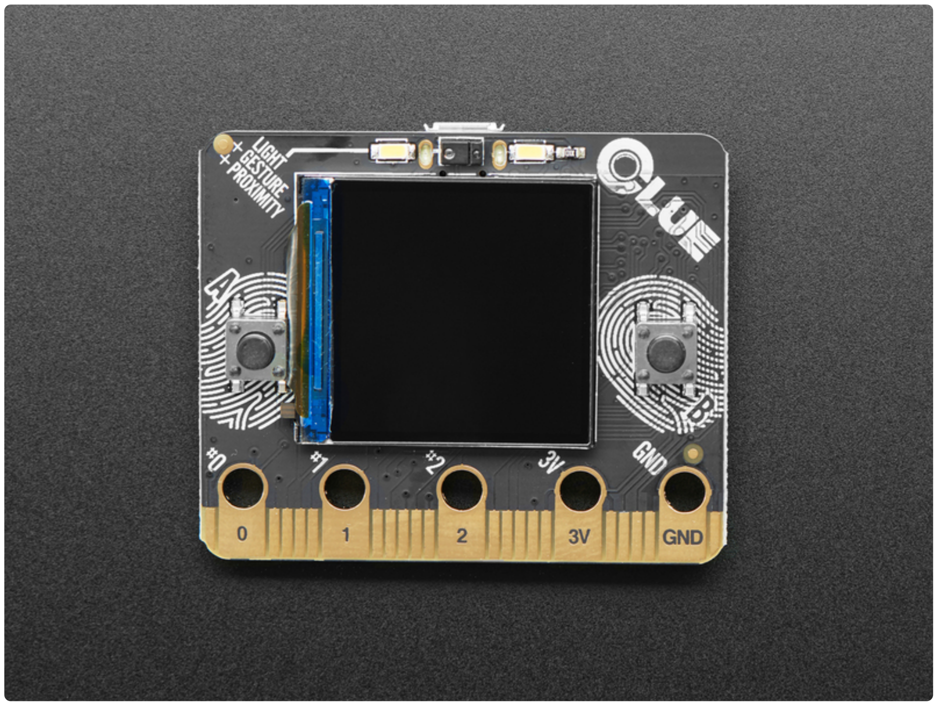


Please note that at this time there is **no MakeCode or Scratch support** for the nRF52840 chipset (of course, we'd love to see MakeCode but there is no ETA when it may be added). While the CLUE is the same outline and we did our best to make the edge-connector pins match up, most cases for the 'bit wont fit the CLUE, and code may not be immediately compatible without adjustment, especially since only

Arduino and CircuitPython are supported at this time.



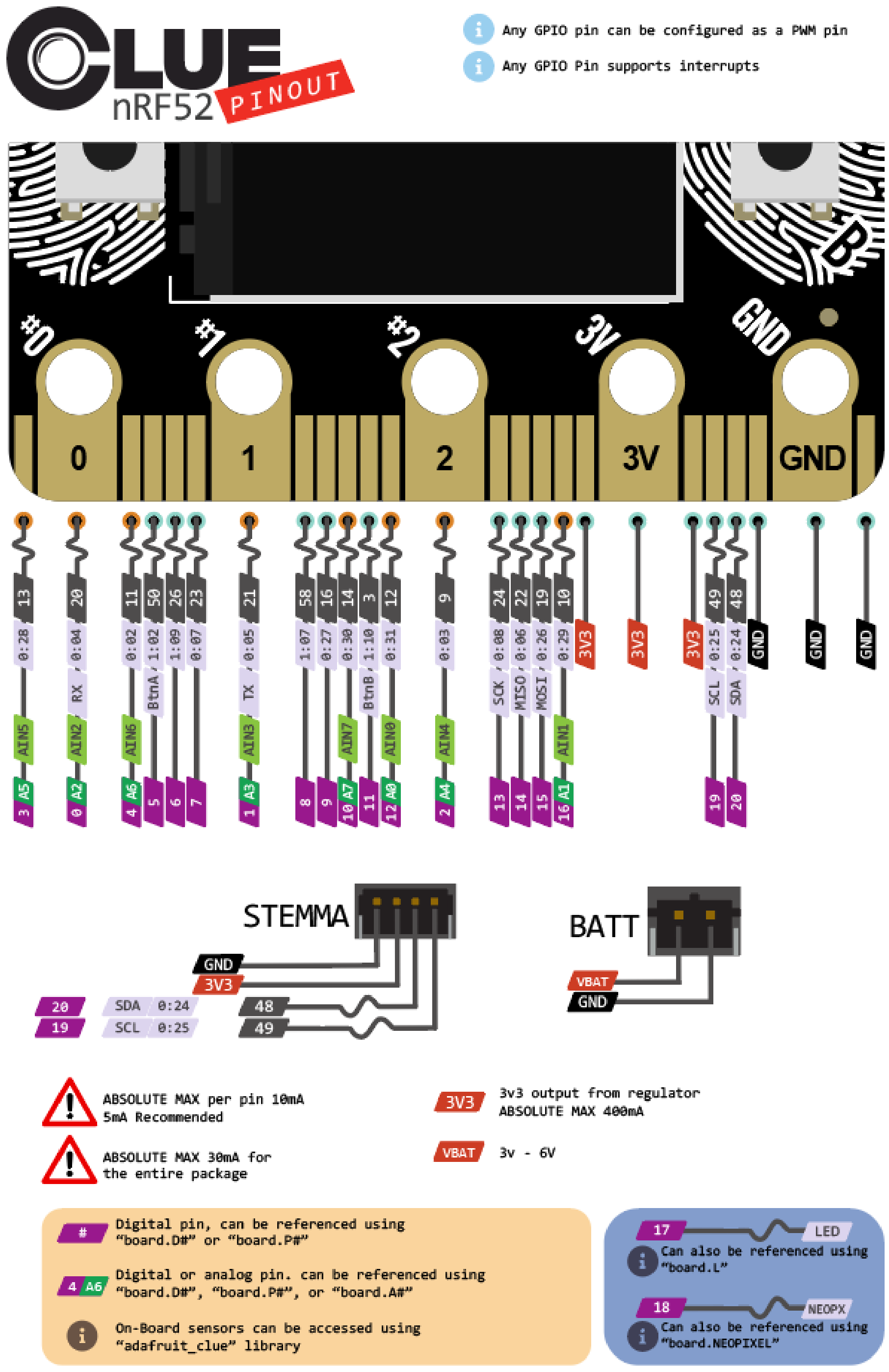
The CLUE is designed for projects that use a ton of sensors - and they're all built in! So you can start exploring your world, measuring, logging and learning. You can transmit data over Bluetooth to a computer or mobile device for data plotting and logging, or save it to the built in storage.



Pinouts

There's all kinds of features packed into CLUE. Let's take a look!

Thanks to **Andrew Tribble**, here's a lovely pinout diagram of the edge-connector:



Any GPIO pin can be configured as a PWM pin

Any GPIO Pin supports interrupts

ABSOLUTE MAX per pin 10mA
5mA Recommended

ABSOLUTE MAX 30mA for the entire package

3V3 — 3v3 output from regulator
ABSOLUTE MAX 400mA

VBAT — 3v - 6V

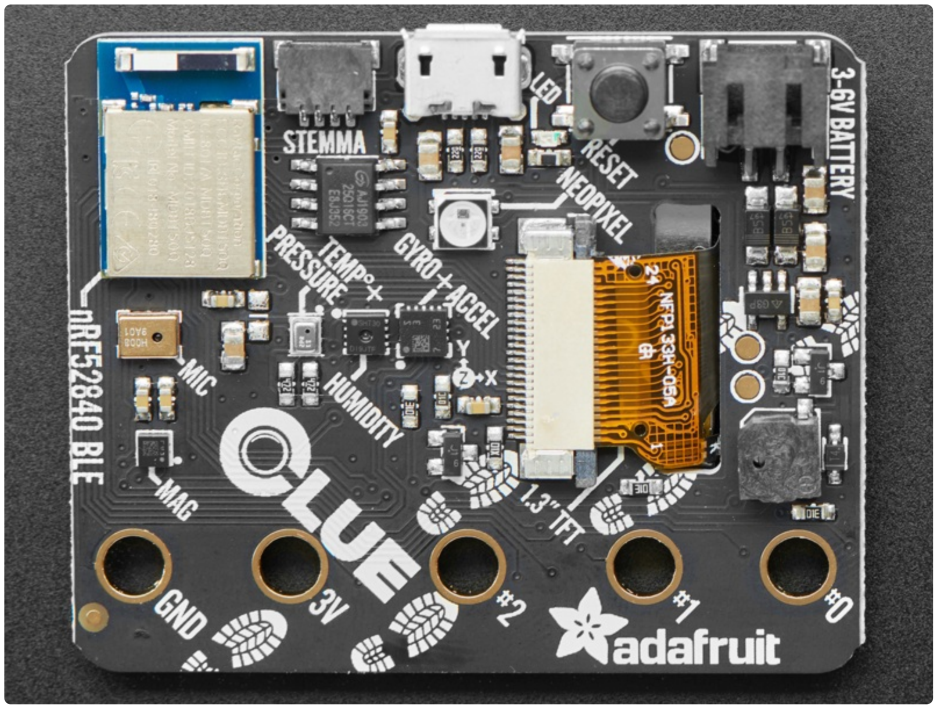
# — Digital pin, can be referenced using "board.D#" or "board.P#"

4 A6 — Digital or analog pin. can be referenced using "board.D#", "board.P#", or "board.A#"

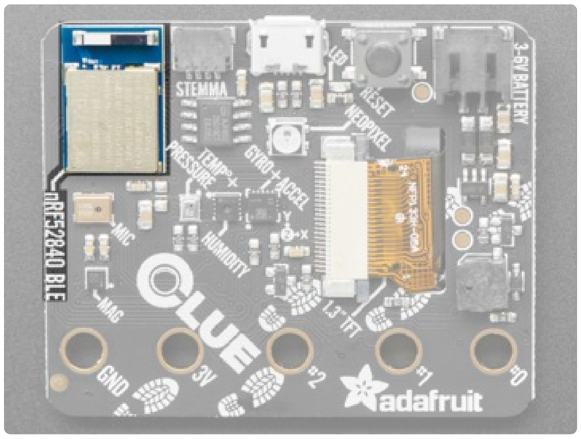
On-Board sensors can be accessed using "adafruit_clue" library

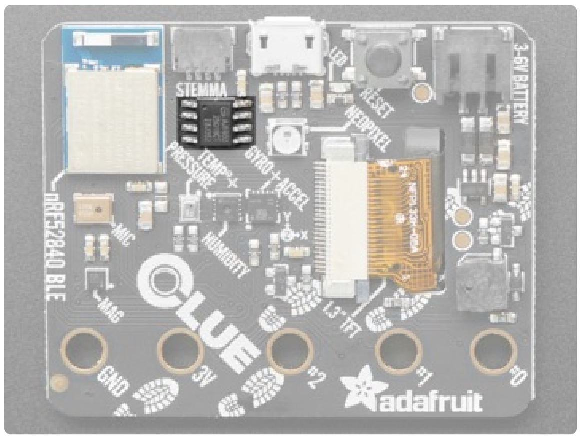
17 LED — Can also be referenced using "board.L"

18 NEOPX — Can also be referenced using "board.NEOPIXEL"



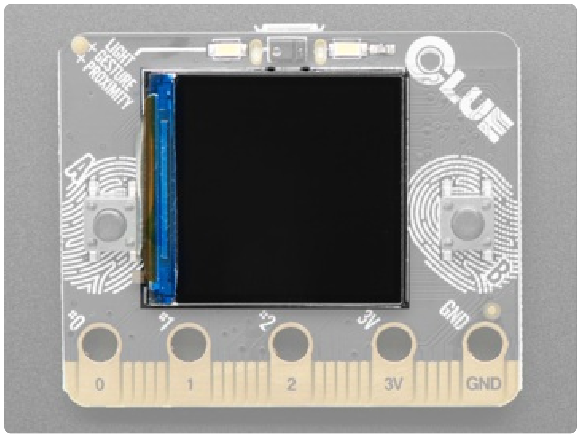
Microcontroller and QSPI

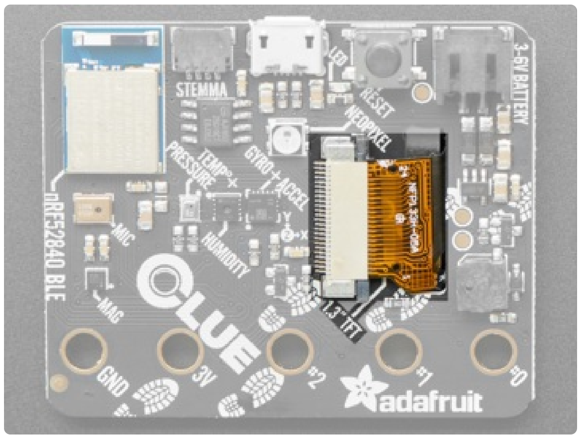




- **Nordic nRF52840 Bluetooth LE processor** - 1 MB of Flash, 256KB RAM, 64 MHz Cortex M4 processor.
- **QSPI flash** - 2MB of internal flash storage for datalogging, images, fonts or CircuitPython code.

Display

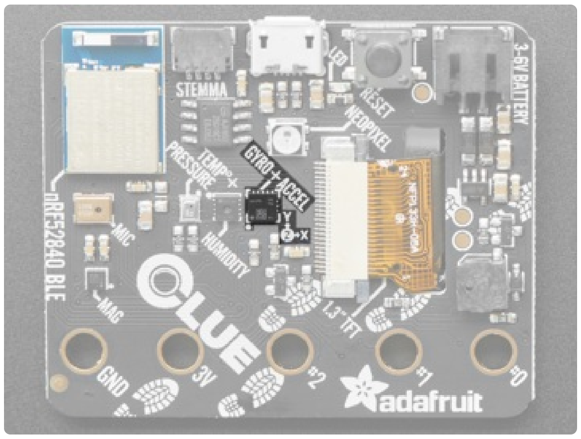




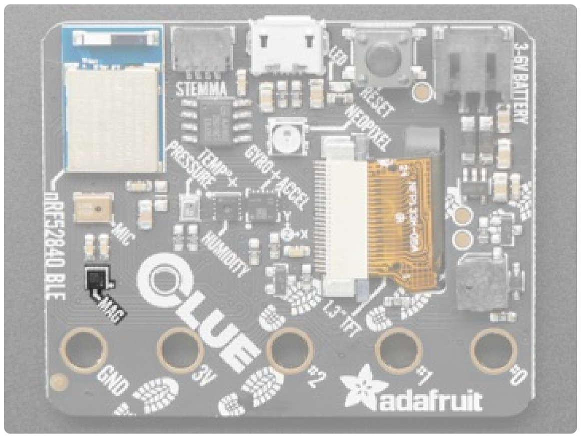
- **1.3″ 240×240 Color IPS TFT display** - Display high resolution text and graphics. The cable goes through a slot in the board to the back to the display connector.

The front of the TFT has a controller chip embedded in the connector cable (you can see it as a thin rectangle to the left of the display). This chip is light sensitive, so if you use a xenon strobe you may disable the display. If you need to use the CLUE In a strobe/very-high-brightness setup, cover up the chip with a strip of black electrical tape

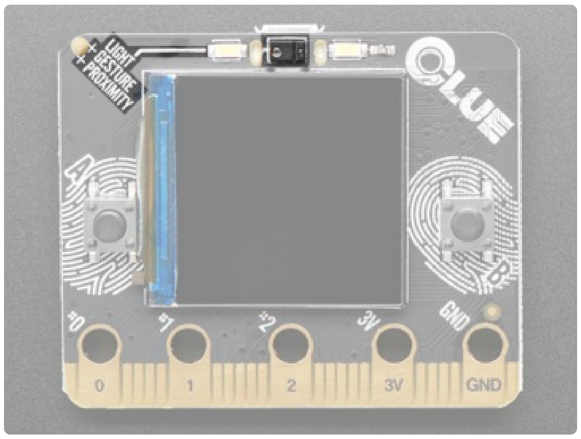
Sensors



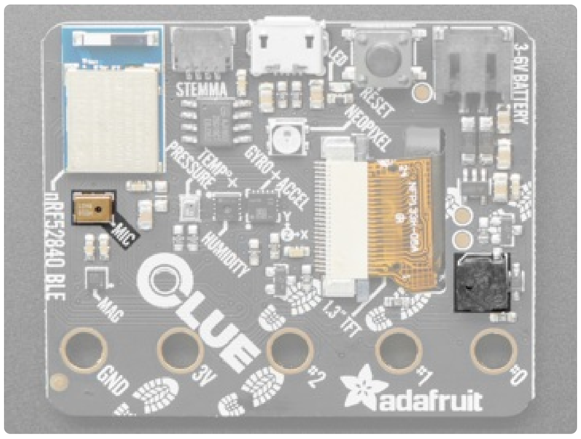
- **Gyro + Accel: LSM6DS33** - This sensor is a 6-DoF IMU accelerometer + gyroscope. The 3-axis accelerometer, can tell you which direction is down towards the Earth (by measuring gravity) or how fast the CLUE is accelerating in 3D space. The 3-axis gyroscope that can measure spin and twist. Pair with a triple-axis magnetometer to create a 9-DoF inertial measurement unit that can detect its orientation in real-space thanks to Earth's stable magnetic field. Sensor is I2C on standard pins.



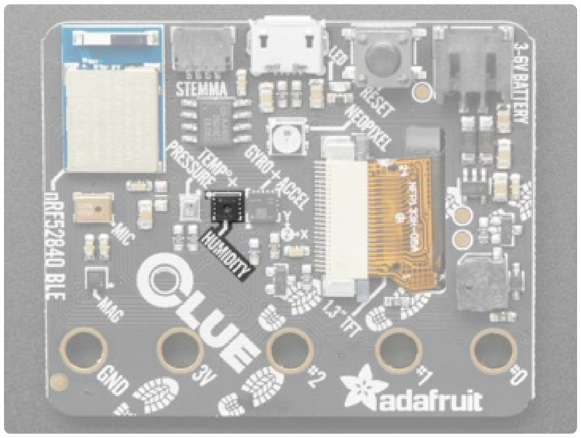
- **Magnetometer: LIS3MDL** - Sense the magnetic fields that surround us with this handy triple-axis magnetometer (compass) module. Magnetometers can sense where the strongest magnetic force is coming from, generally used to detect magnetic north, but can also be used for measuring magnetic fields. This sensor tends to be paired with a 6-DoF (degree of freedom) accelerometer/gyroscope to create a 9-DoF inertial measurement unit that can detect its orientation in real-space thanks to Earth's stable magnetic field. Sensor is I2C on standard pins.



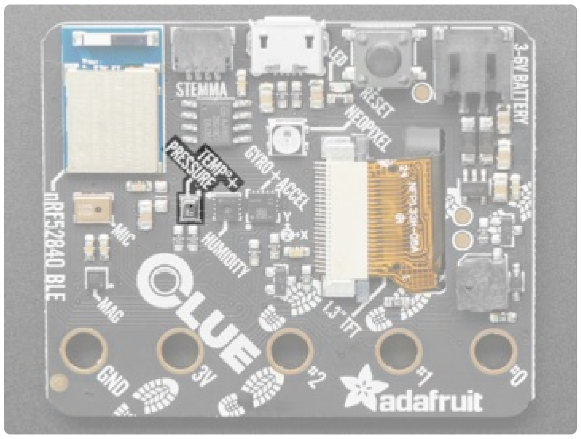
- **Light** + **Gesture** + **Proximity: APDS9960** - Detect simple gestures (left to right, right to left, up to down, down to up are currently supported), return the amount of red, blue, green, and clear light, or return how close an object is to the front of the sensor. This sensor has an integrated IR LED and driver, along with four directional photodiodes that sense reflected IR energy from the LED. Since there are four IR sensors, you can measure the changes in light reflectance at each of the cardinal locations over time and turn those changes into gestures. Sensor is I2C on standard pins.



- **PDM Microphone sound sensor: MP34DT01-M** - PDM sound sensor. In CircuitPython, `board.MICROPHONE_DATA` is PDM data, and `board.MICROPHONE_CLOCK` is PDM clock. In Arduino, `D35` is PDM data, and `D36` is PDM clock.
- **Speaker/buzzer** - This tiny buzzer is good for playing back beeps and tones. It's not suitable for playing back audio files. Addressable in CircuitPython as `board.SPEAKER`, and in Arduino as `D46`.



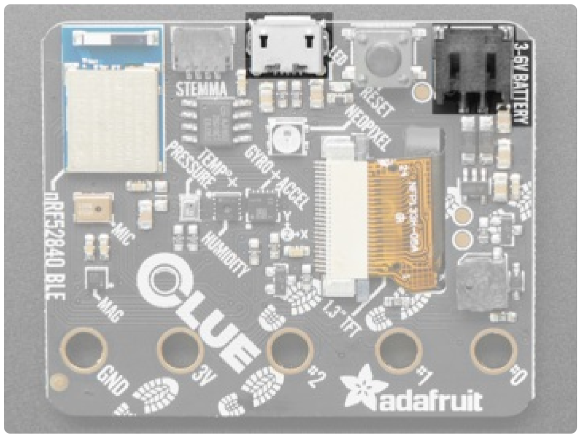
- **Humidity: SHT30** - This sensor has an excellent ±2% relative humidity and ±0.5°C accuracy for most uses. Sensor is I2C on standard pins.



- **Temp + Pressure: BMP280** - This sensor is a precision sensing solution for measuring barometric pressure with ±1 hPa absolute accuraccy, and temperature with ±1.0°C accuracy. Because pressure changes with altitude, and the pressure measurements are so good, you can also use it as an altimeter with ±1 meter accuracy. It has a a low altitude noise of 0.25m and a fast conversion time. Sensor is I2C on standard pins.

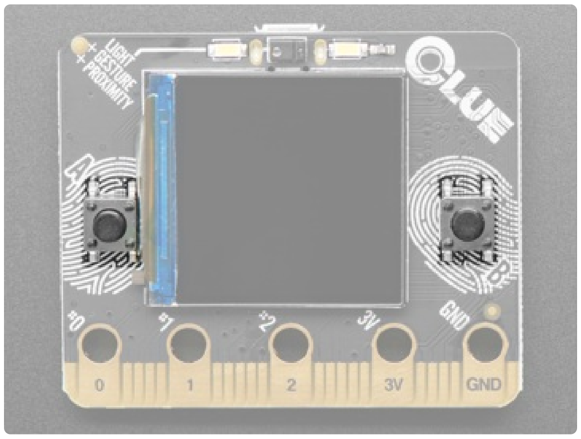
USB and Battery

Like the micro:bit, the CLUE does not have built in LiPoly battery charging. This is for your safety so you can use Alkaline or NiMH batteries without damaging them! You can use LiPoly batteries but you will need an external charger

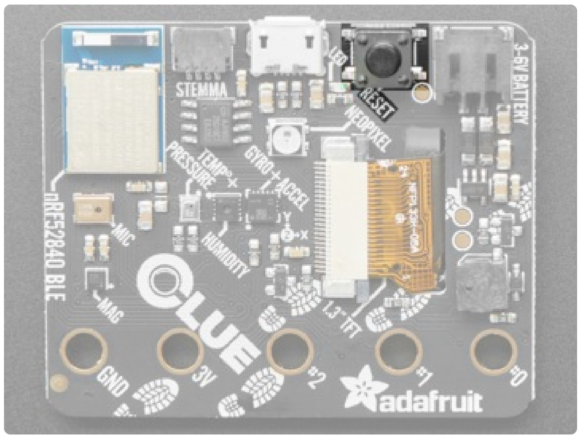


- **USB Micro** - This USB port is used for programming and/or powering the CLUE. It is a standard USB Micro connector.
- **Battery** - 2-pin JST PH connector for a battery. Power the CLUE from any 3V-6V power source, as it has internal regulator and protection diodes.

Buttons

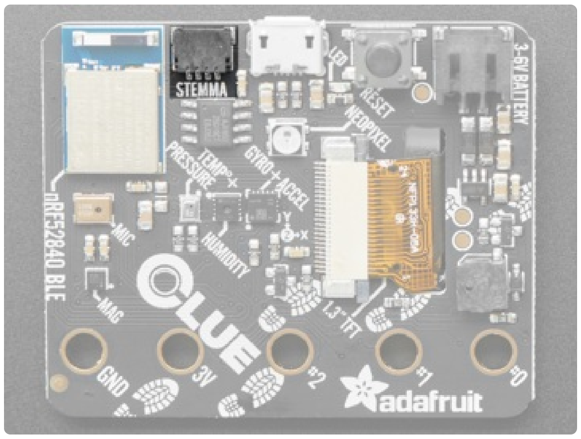


- **A and B buttons** - The CLUE has two user-programmable buttons on the front, labeled A and B. Use them as inputs to control your code. These are unconnected when not pressed, and connected to GND when pressed, so they read LOW. Set the pins to use an internal pull-UP when reading these pins so they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



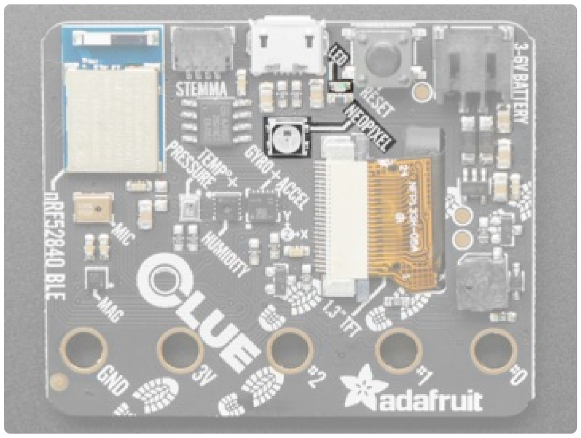
- **Reset button** - This button resets the board. Press once to reset. Quickly press twice to enter the bootloader.

STEMMA QT

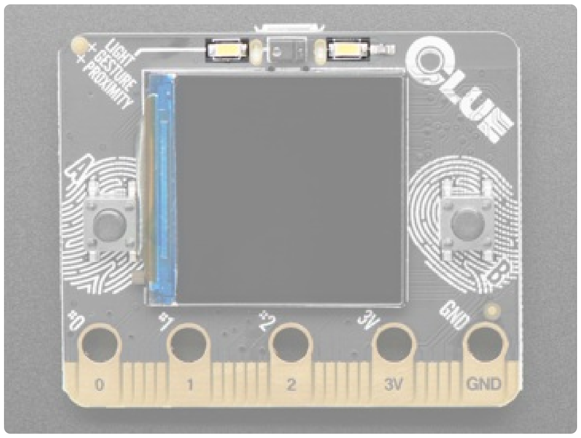


- **Qwiic / STEMMA QT connector** - Use to add more sensors, motor controllers, or displays over I2C. You can plug in GROVE I2C sensors by using an adapter cable (https://adafru.it/IDk).

LEDs

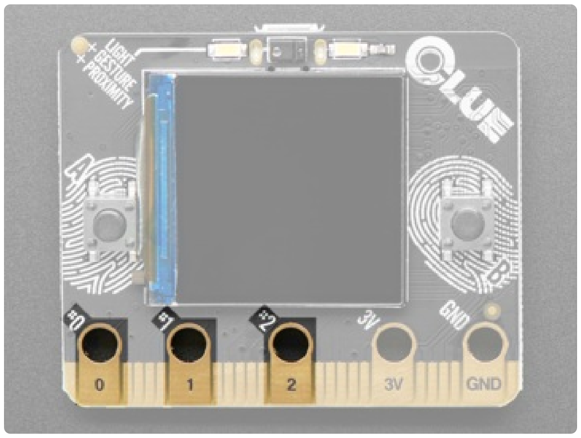


- **NeoPixel** - The addressable RGB NeoPixel LED is used as a status LED by the bootloader and CircuitPython, but is also controllable using code. Control it using `board.NEOPIXEL` in CircuitPython and `D18` in Arduino.
- **Status LED** - This little red LED works as a status LED in the bootloader. Otherwise, it is controllable using code by addressing `board.D17` in CircuitPython, and `D17` in Arduino.

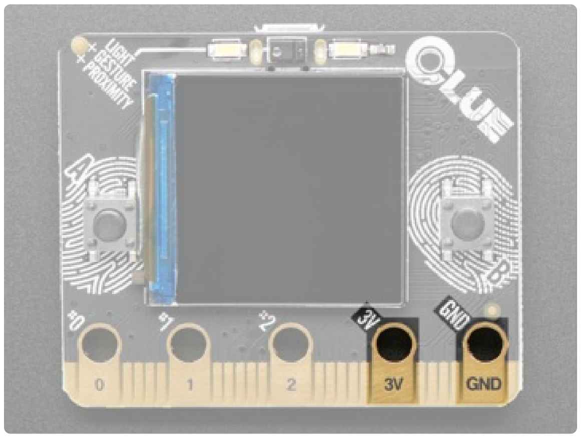


- **Bright white LEDs** - On the front of the board are two bright white LEDs for illumination and color sensing. Control them in CircuitPython using `board.WHITE_LEDS`, and in Arduino using `D43`.

GPIO and Power Pads

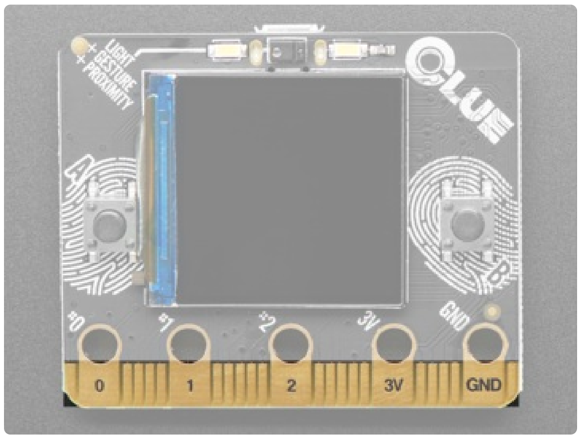


- **Pads 0, 1 and 2** - These pads are used for connecting external sensors etc, typically using alligator clips. They also work as inputs using capacitive touch. In CircuitPython they are `board.D0`, `board.D1`, and `board.D2`. In Arduino, they are `D0`, `D1` and `D2`.



- **3V and GND** - These are the power and ground pads used when connecting external sensors etc. typically using alligator clips.

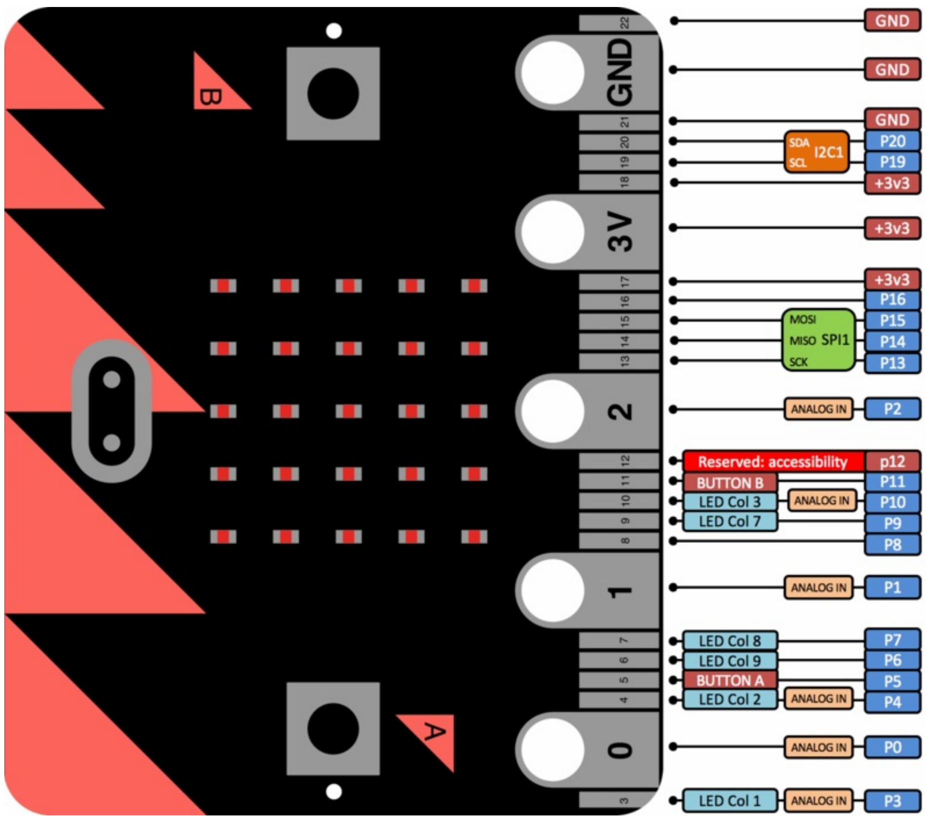
Edge Connector



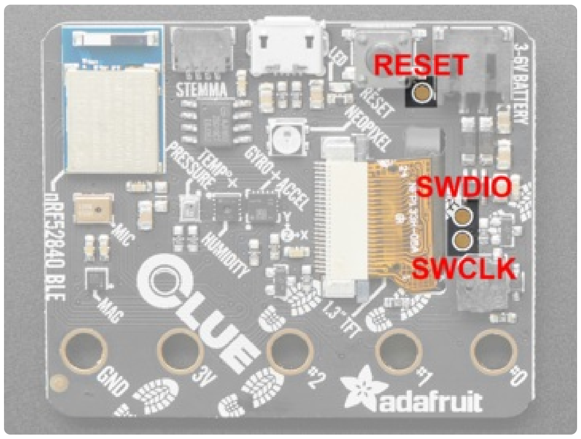
- **Micro:Bit compatible edge connector** - This is the Micro:Bit compatible edge connector, used to break out all of the other features of this microcontroller. Compatible with other Micro:Bit-compatible hardware. While the CLUE is the same outline and we did our best to make the edge-connector pins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

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

- **The I2C pins are on on the same P19/P20** (we like to use D19/D20 naming)
- **The SPI pins are on on the same P13-P15** (we like to use D13-D15 naming)
- There are **analog** pins on **P0** (Arduino `A2`), **P1** (Arduino `A3`), **P2** (Arduino `A4`), **P3** (Arduino `A5`), **P4** (Arduino `A6`), **P10** (Arduino `A7`) just like the micro:bit
- There are *additional* analog pins on D12 (Arduino `A0`) and P16 (Arduino `A1`)
- **Button A and B are on the same P5 and P11 pins**
- Since we don't have an LED matrix, you can use **P3, P4, P6, P7, P9, P10, P11** without worrying about conflicting with an LED grid

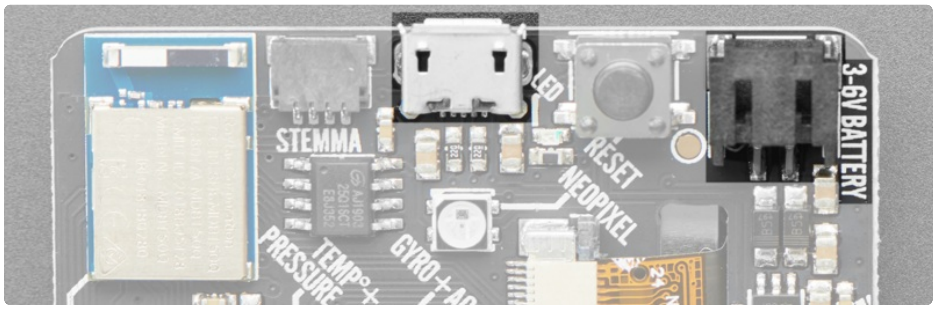


Debug Pads



On the bottom of the board are three pads, one near the reset button, and two to the right of the display cable. The pad near the reset button is reset. Of the two pads near the display cable, the top is SWDIO and on the bottom is SWCLK. On the off chance you want to reprogram your CLUE or debug it using a debug/programmer, you will need to solder/connect to these pads.

Powering Your CLUE



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

micro:bit Power

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

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

Can't use:

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

CLUE Power

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

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

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

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

HELP! Accel/Gyro Not Working?

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

To fix, visit

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

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

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

Arduino Support Setup

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

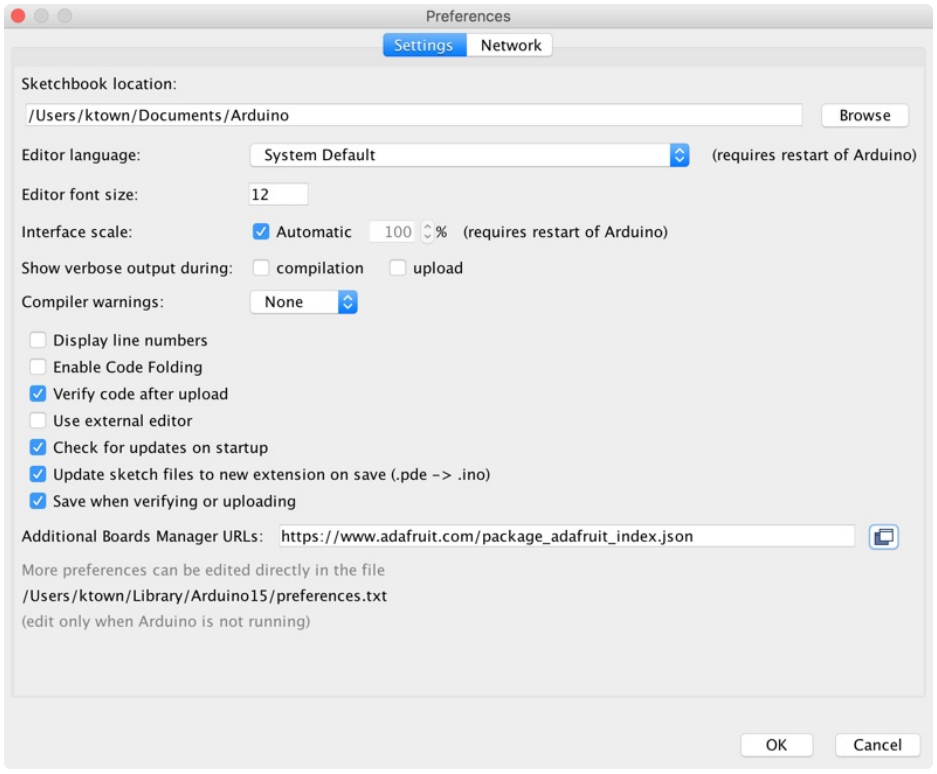
nRF52 support requires at least Arduino IDE version 1.8.6! Please make sure you have an up to date version before proceeding with this guide!

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

1. BSP Installation

Recommended: Installing the BSP via the Board Manager

- Download and install the Arduino IDE (https://adafru.it/fvm) (At least **v1.8**)
- Start the Arduino IDE
- Go into Preferences
- Add `https://www.adafruit.com/package_adafruit_index.json` as an '**Additional Board Manager URL**' (see image below)



Sketchbook location: /Users/ktown/Documents/Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: [✓] Automatic 100 % (requires restart of Arduino)

Show verbose output during: [ ] compilation [ ] upload

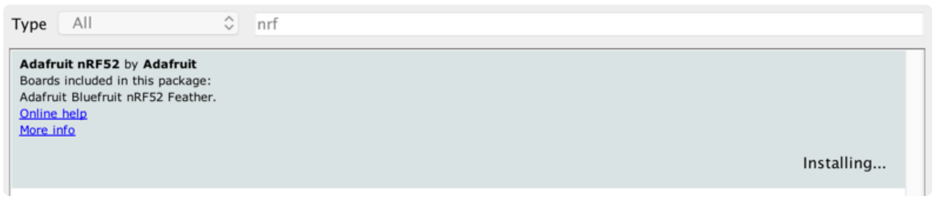
Compiler warnings: None

[ ] Display line numbers
[ ] Enable Code Folding
[✓] Verify code after upload
[ ] Use external editor
[✓] Check for updates on startup
[✓] Update sketch files to new extension on save (.pde -> .ino)
[✓] Save when verifying or uploading

Additional Boards Manager URLs: https://www.adafruit.com/package_adafruit_index.json

More preferences can be edited directly in the file
/Users/ktown/Library/Arduino15/preferences.txt
(edit only when Arduino is not running)

- Restart the Arduino IDE
- Open the **Boards Manager** option from the **Tools -> Board** menu and install '**Adafruit nRF52 by Adafruit**' (see image below)



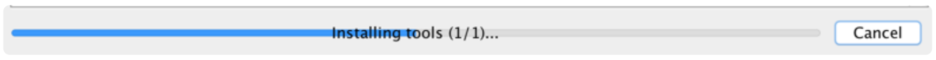
Type All nrf

Adafruit nRF52 by **Adafruit**
Boards included in this package:
Adafruit Bluefruit nRF52 Feather.
Online help
More info

Installing...

It will take up to a few minutes to finish installing the cross-compiling toolchain and tools associated with this BSP.

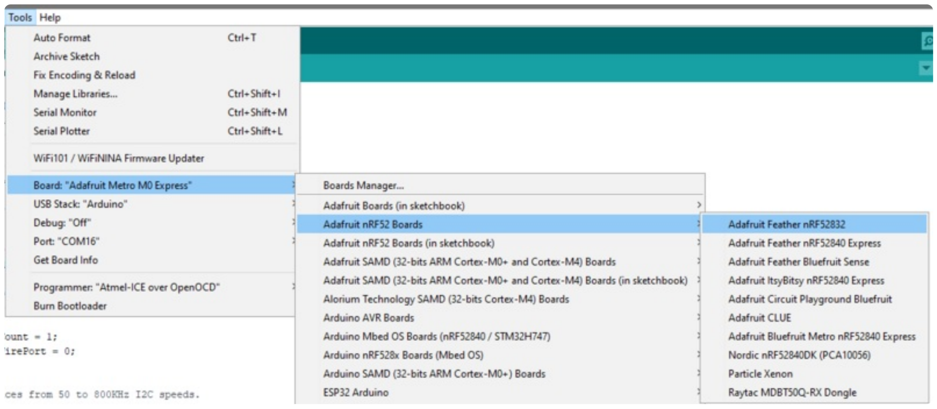
The delay during the installation stage shown in the image below is normal , please be patient and let the installation terminate normally:



Once the BSP is installed, select

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

from the **Tools -> Board** menu, which will update your system config to use the right compiler and settings for the nRF52:


Tools
Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Adafruit Metro M0 Express"
USB Stack: "Arduino"
Debug: "Off"
Port: "COM16"
Get Board Info
Programmer: "Atmel-ICE over OpenOCD"
Burn Bootloader
Boards Manager...
Adafruit Boards (in sketchbook)
Adafruit nRF52 Boards
Adafruit nRF52 Boards (in sketchbook)
Adafruit SAMD (32-bits ARM Cortex-M0+ and Cortex-M4) Boards
Adafruit SAMD (32-bits ARM Cortex-M0+ and Cortex-M4) Boards (in sketchbook)
Alorium Technology SAMD (32-bits Cortex-M4) Boards
Arduino AVR Boards
Arduino Mbed OS Boards (nRF52840 / STM32H747)
Arduino nRF528x Boards (Mbed OS)
Arduino SAMD (32-bits ARM Cortex-M0+) Boards
ESP32 Arduino
Adafruit Feather nRF52832
Adafruit Feather nRF52840 Express
Adafruit Feather Bluefruit Sense
Adafruit ItsyBitsy nRF52840 Express
Adafruit Circuit Playground Bluefruit
Adafruit CLUE
Adafruit Bluefruit Metro nRF52840 Express
Nordic nRF52840DK (PCA10056)
Particle Xenon
Raytac MDBT50Q-RX Dongle

```
ount = 1;
irePort = 0;

ces from 50 to 800KHz I2C speeds.
```

2. LINUX ONLY: adafruit-nrfutil Tool Installation

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

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

Install python3 if it is not installed in your system already

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

Then run the following command to install the tool from PyPi

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

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

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

3. Update the bootloader (nRF52832 ONLY)

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

Follow this link for instructions on how to do that

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

https://adafru.it/Dsx

https://adafru.it/Dsx

Advanced Option: Manually Install the BSP via 'git'

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

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

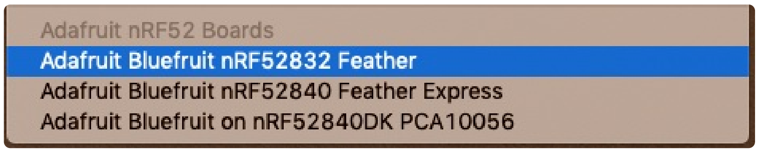
1. Install BSP via Board Manager as above to install compiler & tools.
2. Delete the core folder **nrf52** installed by Board Manager in Adruino15, depending on your OS. It could be
   macOS: `~/Library/Arduino15/packages/adafruit/hardware/nrf52`
   Linux: `~/.arduino15/packages/adafruit/hardware/nrf52`
   Windows: `%APPDATA%\Local\Arduino15\packages\adafruit\hardware\nrf52`
3. Go to the sketchbook folder on your command line, which should be one of the following:
   macOS: `~/Documents/Arduino`
   Linux: `~/Arduino`
   Windows: `~/Documents/Arduino`
4. Create a folder named `hardware/Adafruit`, if it does not exist, and change directories into it.
5. Clone the Adafruit_nRF52_Arduino (https://adafru.it/vaF) repo in the folder described in step 2:
   `git clone --recurse-submodules git@github.com:adafruit/Adafruit_nRF52_Arduino.git`
6. This should result in a final folder name like
   `~/Documents/Arduino/hardware/Adafruit/Adafruit_nRF52_Arduino` (macOS).
7. Restart the Arduino IDE

Arduino Board Testing

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

1. Select the Board Target

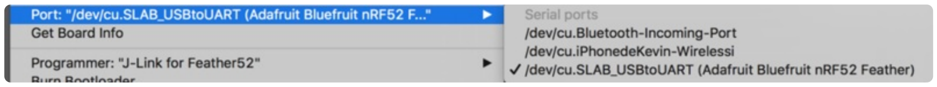
- Go to the **Tools** menu
- Select **Tools** > **Board** > **Adafruit Bluefruit nRF52 Feather** for **nRF52832-based boards**
- Select **Tools** > **Board** > **Adafruit Bluefruit nRF52840 Feather Express** for **nRF52840-based boards**
- Select **Tools** > **Board** > **Adafruit CLUE** for the **Adafruit CLUE**


Adafruit nRF52 Boards
Adafruit Bluefruit nRF52832 Feather
Adafruit Bluefruit nRF52840 Feather Express
Adafruit Bluefruit on nRF52840DK PCA10056

2. Select the USB CDC Serial Port

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

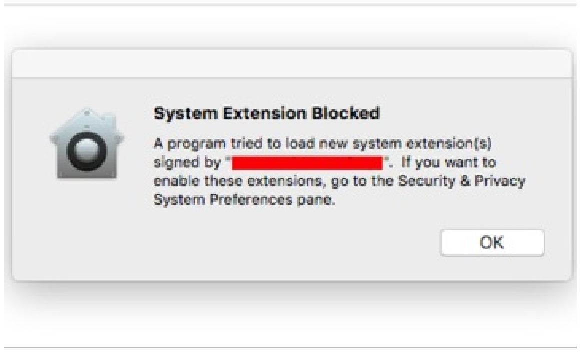
- Go to **Tools** > **Port** and select the appropriate device


Port: "/dev/cu.SLAB_USBtoUART (Adafruit Bluefruit nRF52 F..."
Get Board Info
Programmer: "J-Link for Feather52"
Burn Bootloader
Serial ports
/dev/cu.Bluetooth-Incoming-Port
/dev/cu.iPhonedeKevin-Wirelessi
✓ /dev/cu.SLAB_USBtoUART (Adafruit Bluefruit nRF52 Feather)

Download & Install CP2104 Driver (nRF52832)

For Feather nRF52832 If you don't see the SiLabs device listed, you may need to install the [SiLabs CP2104 driver](https://adafru.it/yfA) (https://adafru.it/yfA) on your system.

On MacOS If you see this dialog message while installing driver

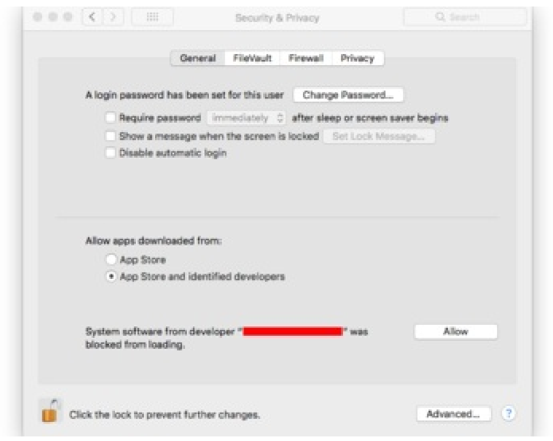


System Extension Blocked

A program tried to load new system extension(s) signed by "[redacted]". If you want to enable these extensions, go to the Security & Privacy System Preferences pane.

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

And cannot find the serial port of CP2104, it is highly possible that driver is blocked.



Security & Privacy

A login password has been set for this user

- [ ] Require password [immediately] after sleep or screen saver begins
- [ ] Show a message when the screen is locked

- [ ] Disable automatic login

Allow apps downloaded from:

App Store [✓ App Store and identified developers]

System software from developer "[redacted]" was blocked from loading.

Click the lock to prevent further changes.

To enable it go to **System Preferences -> Security & Privacy** and click allow if you see Silab in the developer name.

Download & Install Adafruit Driver (nRF52840 Windows)

For Feather nRF52840, If you are using Windows, you will need to follows [Windows Driver Installation](https://adafru.it/D0H) (https://adafru.it/D0H) to download and install driver.

3. Update the bootloader (nRF52832 Feather Only)

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

Follow this link for instructions on how to do that

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

https://adafru.it/Dsx

https://adafru.it/Dsx

4. Run a Test Sketch

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

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

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

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

If Arduino failed to upload sketch to the Feather

If you get this error:

```
Timed out waiting for acknowledgement from device.

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

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

The Bootloader (https://adafru.it/Dsx) guide

It only has to be done once to update your Feather

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

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

```
1.
   sudo dpkg --add-architecture i386

2.
   sudo apt-get update

3.
   sudo apt-get install libc6:i386
```

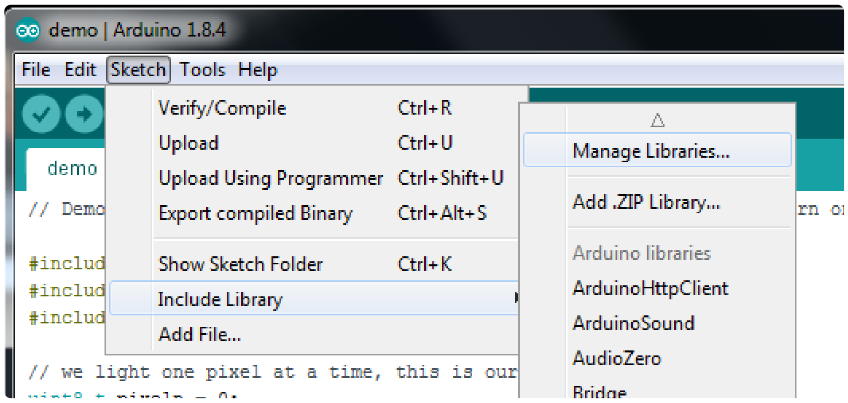
Arcada Libraries

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

There's a lot of libraries!

Install Libraries

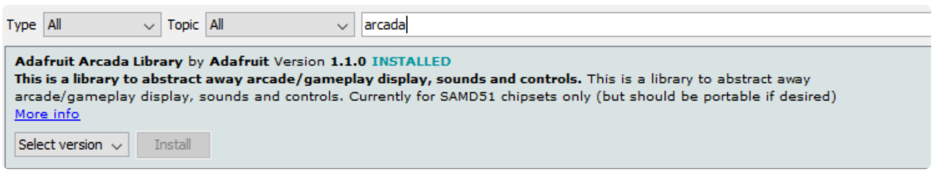
Open up the library manager...



And install the following libraries:

Adafruit Arcada

This library generalizes the hardware for you so you can read the joystick, draw to the display, read files, etc. without having to worry about the underlying methods



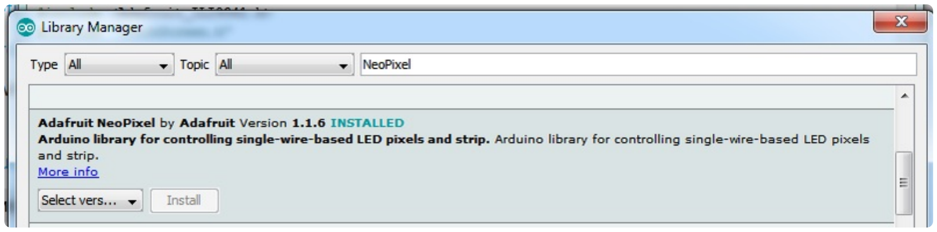
Adafruit Arcada Library by **Adafruit** Version **1.1.0** INSTALLED
This is a library to abstract away arcade/gameplay display, sounds and controls. This is a library to abstract away arcade/gameplay display, sounds and controls. Currently for SAMD51 chipsets only (but should be portable if desired)
More info

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

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

Adafruit NeoPixel

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


Library Manager

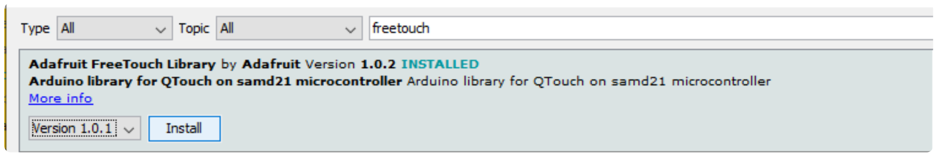
Type All Topic All NeoPixel

Adafruit NeoPixel by Adafruit Version 1.1.6 INSTALLED
Arduino library for controlling single-wire-based LED pixels and strip. Arduino library for controlling single-wire-based LED pixels and strip.
More info

Select vers... Install

Adafruit FreeTouch

This is the open source version of QTouch for SAMD21 boards

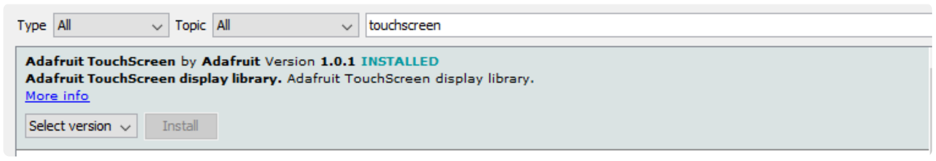

Type All Topic All freetouch

Adafruit FreeTouch Library by Adafruit Version 1.0.2 INSTALLED
Arduino library for QTouch on samd21 microcontroller Arduino library for QTouch on samd21 microcontroller
More info

Version 1.0.1 Install

Adafruit Touchscreen

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

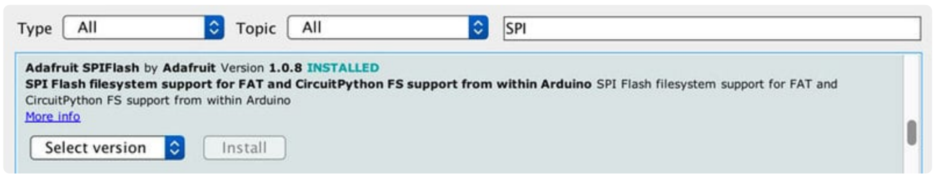

Type All Topic All touchscreen

Adafruit TouchScreen by Adafruit Version 1.0.1 INSTALLED
Adafruit TouchScreen display library. Adafruit TouchScreen display library.
More info

Select version Install

Adafruit SPIFlash

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

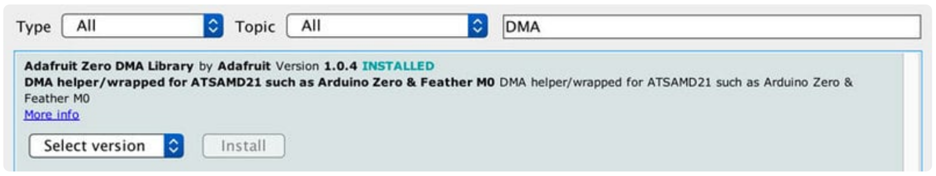

Type All Topic All SPI

Adafruit SPIFlash by Adafruit Version 1.0.8 INSTALLED
SPI Flash filesystem support for FAT and CircuitPython FS support from within Arduino SPI Flash filesystem support for FAT and CircuitPython FS support from within Arduino
More info

Select version Install

Adafruit Zero DMA

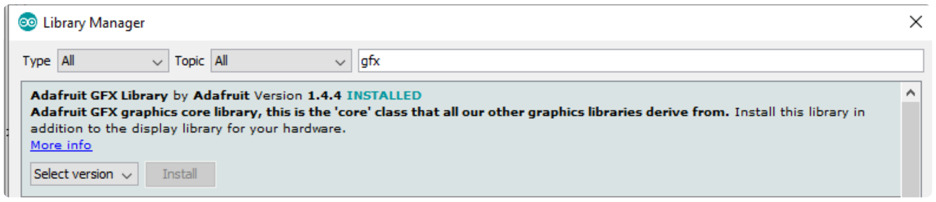
This is used by the Graphics Library if you choose to use DMA


Type All Topic All DMA

Adafruit Zero DMA Library by **Adafruit** Version **1.0.4** **INSTALLED**
DMA helper/wrapped for ATSAMD21 such as Arduino Zero & Feather M0 DMA helper/wrapped for ATSAMD21 such as Arduino Zero & Feather M0
More info

Adafruit GFX

This is the graphics library used to draw to the screen


Library Manager

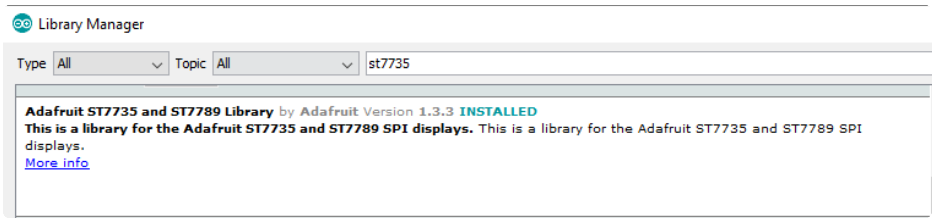
Type All Topic All gfx

Adafruit GFX Library by **Adafruit** Version **1.4.4** **INSTALLED**
Adafruit GFX graphics core library, this is the 'core' class that all our other graphics libraries derive from. Install this library in addition to the display library for your hardware.
More info

If using an older (pre-1.8.10) Arduino IDE, locate and install **Adafruit_BusIO** (newer versions do this one automatically).

Adafruit ST7735

The display on the PyBadge/PyGamer & other Arcada boards

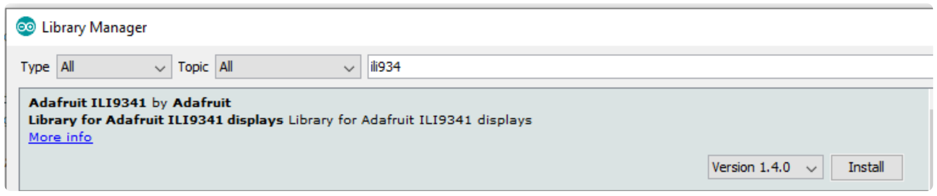

Library Manager

Type All Topic All st7735

Adafruit ST7735 and ST7789 Library by **Adafruit** Version **1.3.3** **INSTALLED**
This is a library for the Adafruit ST7735 and ST7789 SPI displays. This is a library for the Adafruit ST7735 and ST7789 SPI displays.
More info

Adafruit ILI9341

The display on the PyPortal & other Arcada boards

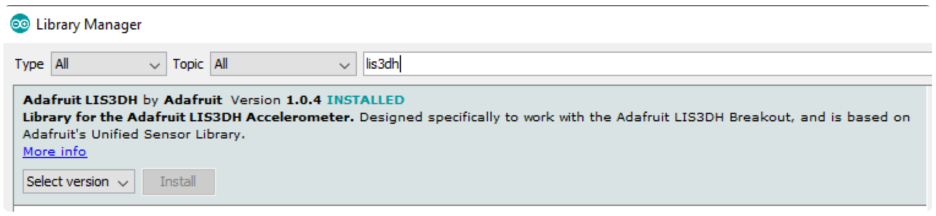

Library Manager

Type All Topic All ili934

Adafruit ILI9341 by **Adafruit**
Library for Adafruit ILI9341 displays Library for Adafruit ILI9341 displays
More info

Adafruit LIS3DH

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

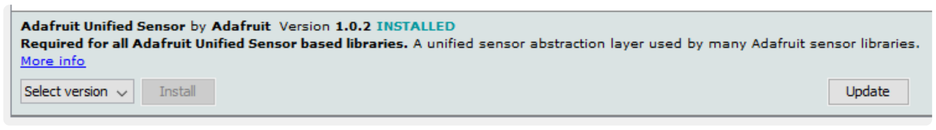

Library Manager

Type All Topic All lis3dh

Adafruit LIS3DH by **Adafruit** Version **1.0.4** INSTALLED
Library for the Adafruit LIS3DH Accelerometer. Designed specifically to work with the Adafruit LIS3DH Breakout, and is based on Adafruit's Unified Sensor Library.
More info

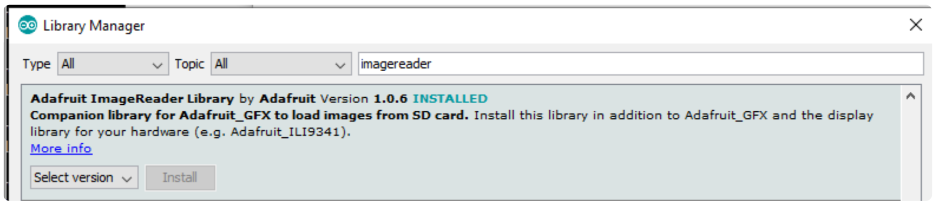
Adafruit Sensor

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


Adafruit Unified Sensor by **Adafruit** Version **1.0.2** INSTALLED
Required for all Adafruit Unified Sensor based libraries. A unified sensor abstraction layer used by many Adafruit sensor libraries.
More info

Adafruit ImageReader

For reading bitmaps from SPI Flash or SD and displaying

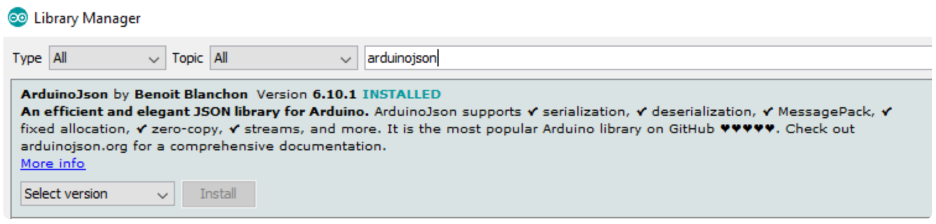

Library Manager

Type All Topic All imagereader

Adafruit ImageReader Library by **Adafruit** Version **1.0.6** INSTALLED
Companion library for Adafruit_GFX to load images from SD card. Install this library in addition to Adafruit_GFX and the display library for your hardware (e.g. Adafruit_ILI9341).
More info

ArduinoJson

We use this library to read and write configuration files

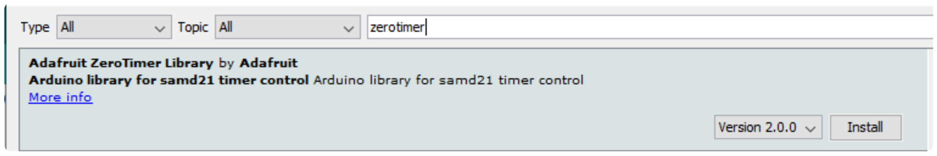

Library Manager

Type All Topic All arduinojson

ArduinoJson by **Benoit Blanchon** Version **6.10.1** INSTALLED
An efficient and elegant JSON library for Arduino. ArduinoJson supports ✔ serialization, ✔ deserialization, ✔ MessagePack, ✔ fixed allocation, ✔ zero-copy, ✔ streams, and more. It is the most popular Arduino library on GitHub ♥♥♥♥♥. Check out arduinojson.org for a comprehensive documentation.
More info

Adafruit ZeroTimer

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

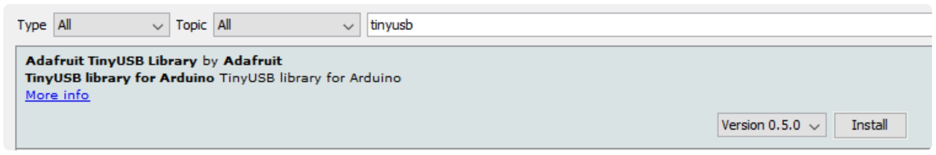

Type All Topic All zerotimer

Adafruit ZeroTimer Library by **Adafruit**
Arduino library for samd21 timer control Arduino library for samd21 timer control
More info

Version 2.0.0 Install

Adafruit TinyUSB

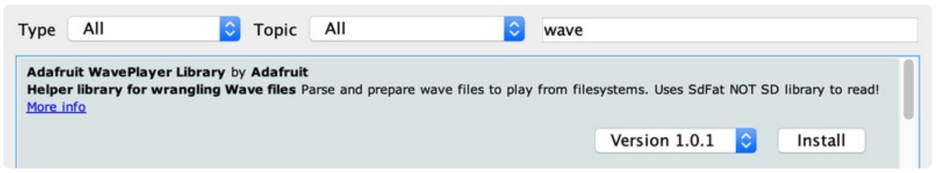
This lets us do cool stuff with USB like show up as a Keyboard or Disk Drive


Type All Topic All tinyusb

Adafruit TinyUSB Library by **Adafruit**
TinyUSB library for Arduino TinyUSB library for Arduino
More info

Adafruit WavePlayer

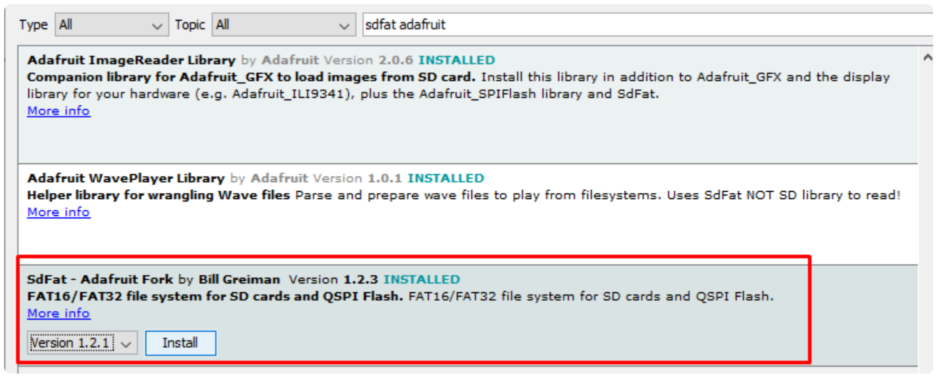
Helps us play .WAV sound files.


Type All Topic All wave

Adafruit WavePlayer Library by **Adafruit**
Helper library for wrangling Wave files Parse and prepare wave files to play from filesystems. Uses SdFat NOT SD library to read!
More info

SdFat (Adafruit Fork)

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


Type All Topic All sdfat adafruit

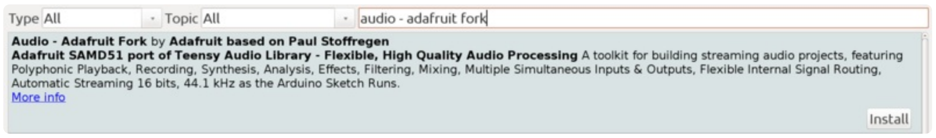
Adafruit ImageReader Library by **Adafruit** Version **2.0.6** **INSTALLED**
Companion library for Adafruit_GFX to load images from SD card. Install this library in addition to Adafruit_GFX and the display library for your hardware (e.g. Adafruit_ILI9341), plus the Adafruit_SPIFlash library and SdFat.
More info

Adafruit WavePlayer Library by **Adafruit** Version **1.0.1** **INSTALLED**
Helper library for wrangling Wave files Parse and prepare wave files to play from filesystems. Uses SdFat NOT SD library to read!
More info

SdFat - Adafruit Fork by **Bill Greiman** Version **1.2.3** **INSTALLED**
FAT16/FAT32 file system for SD cards and QSPI Flash. FAT16/FAT32 file system for SD cards and QSPI Flash.
More info

Audio - Adafruit Fork

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

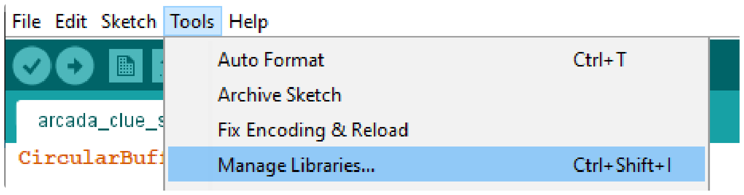

Type All Topic All audio - adafruit fork

Audio - Adafruit Fork by **Adafruit based on Paul Stoffregen**
Adafruit SAMD51 port of Teensy Audio Library - Flexible, High Quality Audio Processing A toolkit for building streaming audio projects, featuring Polyphonic Playback, Recording, Synthesis, Analysis, Effects, Filtering, Mixing, Multiple Simultaneous Inputs & Outputs, Flexible Internal Signal Routing, Automatic Streaming 16 bits, 44.1 kHz as the Arduino Sketch Runs.
More info

Sensor Libraries

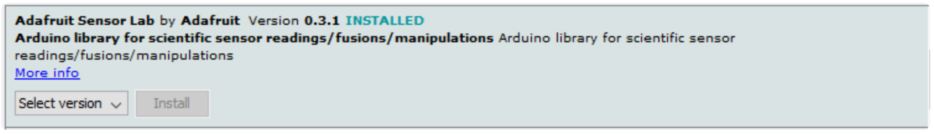
To read and manage all the sensors on your CLUE board, we will need libraries to control each and every one of them.

Use the library manager to install them.



Adafruit Sensor Lab

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



Adafruit Sensor Lab by **Adafruit** Version **0.3.1** INSTALLED
Arduino library for scientific sensor readings/fusions/manipulations Arduino library for scientific sensor readings/fusions/manipulations
More info

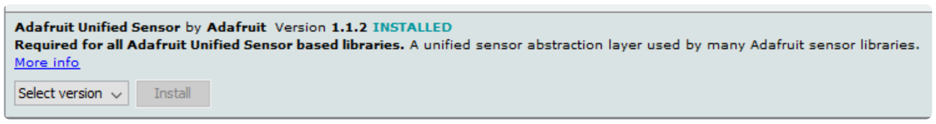
Select version Install

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

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

Search for and install the following:

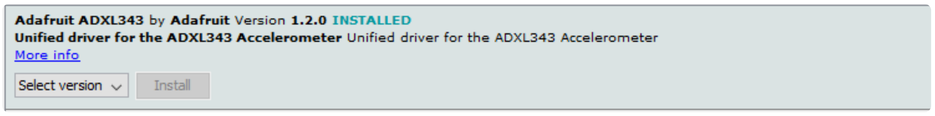
Adafruit Unified Sensor



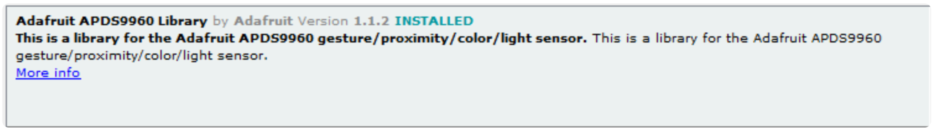
Adafruit Unified Sensor by **Adafruit** Version **1.1.2** INSTALLED
Required for all Adafruit Unified Sensor based libraries. A unified sensor abstraction layer used by many Adafruit sensor libraries.
More info

Select version Install

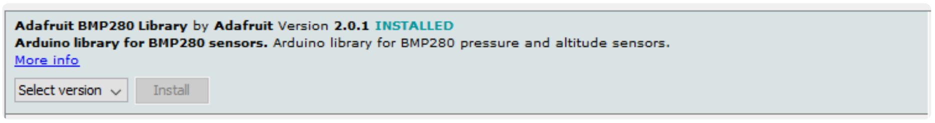
Adafruit ADXL343


Adafruit ADXL343 by **Adafruit** Version **1.2.0** INSTALLED
Unified driver for the ADXL343 Accelerometer Unified driver for the ADXL343 Accelerometer
More info

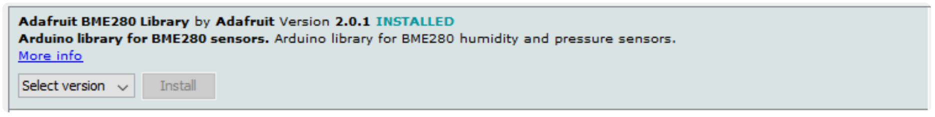
Adafruit APDS9660


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

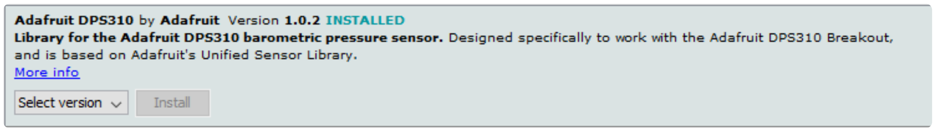
Adafruit BMP280


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

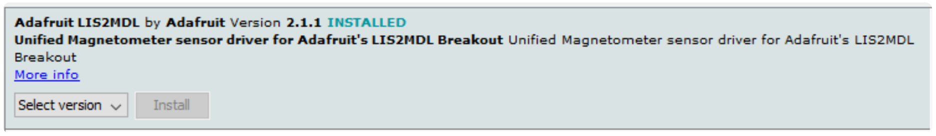
Adafruit BME280


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

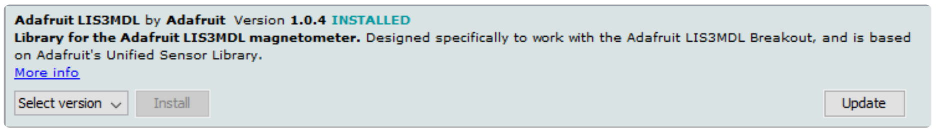
Adafruit DPS310


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

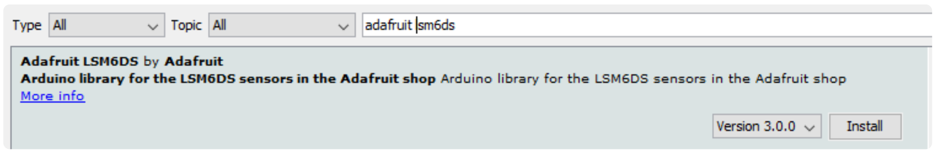
Adafruit LIS2MDL


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

Adafruit LIS3MDL


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

Adafruit LSM6DS

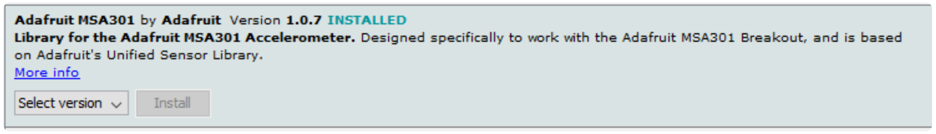


Type All Topic All adafruit lsm6ds

Adafruit LSM6DS by **Adafruit**
Arduino library for the LSM6DS sensors in the Adafruit shop Arduino library for the LSM6DS sensors in the Adafruit shop
More info

Version 3.0.0 Install

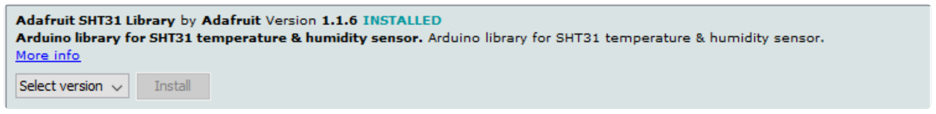
Adafruit MSA301



Adafruit MSA301 by **Adafruit** Version **1.0.7** INSTALLED
Library for the Adafruit MSA301 Accelerometer. Designed specifically to work with the Adafruit MSA301 Breakout, and is based on Adafruit's Unified Sensor Library.
More info

Select version Install

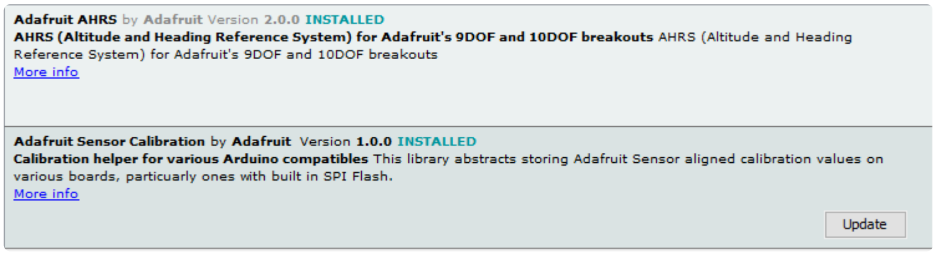
Adafruit SHT31



Adafruit SHT31 Library by **Adafruit** Version **1.1.6** INSTALLED
Arduino library for SHT31 temperature & humidity sensor. Arduino library for SHT31 temperature & humidity sensor.
More info

Select version Install

Adafruit AHRS & Adafruit Sensor Calibration



Adafruit AHRS by **Adafruit** Version **2.0.0** INSTALLED
AHRS (Altitude and Heading Reference System) for Adafruit's 9DOF and 10DOF breakouts AHRS (Altitude and Heading Reference System) for Adafruit's 9DOF and 10DOF breakouts
More info

Adafruit Sensor Calibration by **Adafruit** Version **1.0.0** INSTALLED
Calibration helper for various Arduino compatibles This library abstracts storing Adafruit Sensor aligned calibration values on various boards, particuarly ones with built in SPI Flash.
More info

Update

Arduino Test

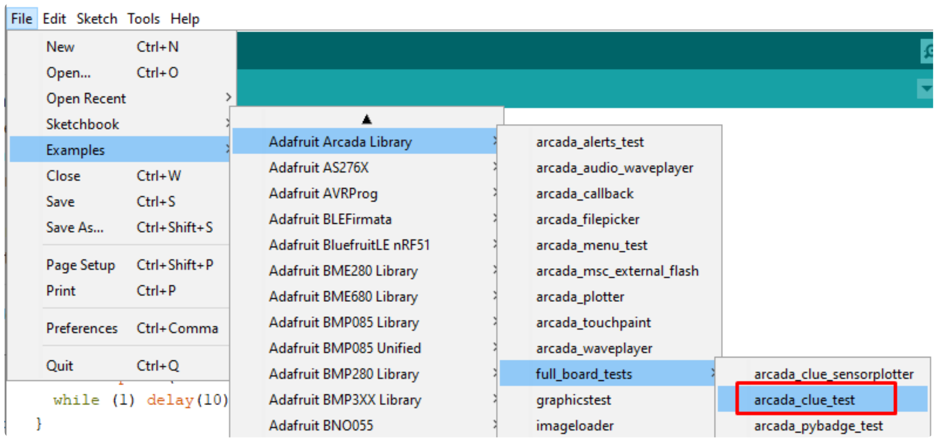
Once you've got the IDE installed and both **Arcada** and **SensorLab** libraries in place you can compile and run the test sketch. This will check all the hardware, and display it on the screen, its sort of a universal test because every part is checked. It's also a great reference if you want to know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

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

https://adafru.it/Mcx

https://adafru.it/Mcx

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



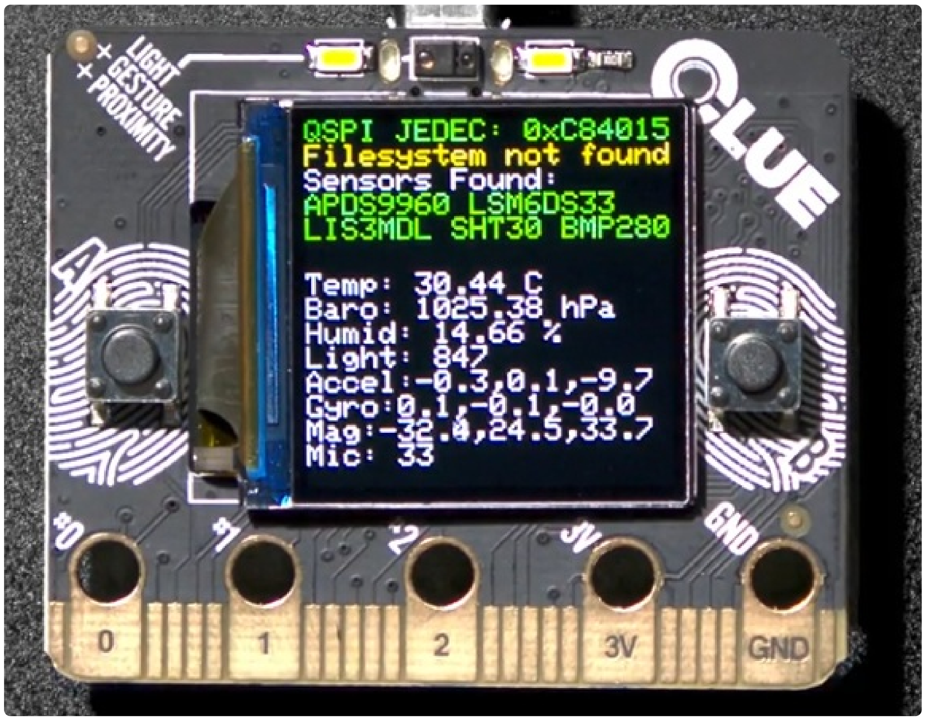
```
  while (1) delay(10);
}
```

The test code

1. Checks the QSPI flash chip initialised correctly, and displays the manufacturer/device ID if so
2. Checks if the QPI flash has a filesystem on it (if not, try loading CircuitPython which will create a filesystem)
3. Tests if all the sensors are found. You should see **APDS9960 LSM6DS33 LISS3MDL SHT30** and **BMP280** all in green text. If any are in red text, that means there was difficulty detecing the sensor. Try disconnecting it from power completely, waiting a few seconds, and plugging it back in.
4. Print the ambient temperature from the BMP280
5. Print the barometric pressure from the BMP280
6. Print the humidity from the SHT30
7. Print the light level from the APDS9960
8. Print the accelerometer output from the LSM6DS33
9. Print the gyroscope output from the LSM6DS33
10. Print the magnetometer output from the LIS3MDL

11. Print the audio level detected by the microphone (you can try blowing on the mic to see the number increase)
12. When the left button is held down, you will hear a beep to test the piezo
13. When the right button is held down, the front white LEDs will light up
14. The NeoPixel on the back will display colors in the rainbow
15. The red LED will pulse in and out.

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



```
QSPI JEDEC: 0xC84015
Filesystem not found
Sensors Found:
APDS9960 LSM6DS33
LIS3MDL SHT30 BMP280

Temp: 30.44 C
Baro: 1025.38 hPa
Humid: 14.66 %
Light: 847
Accel:-0.3,0.1,-9.7
Gyro:0.1,-0.1,-0.0
Mag:-32.4,24.5,33.7
Mic: 33
```

Animated GIF Player

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

We have a full guide on the animated GIF code here (https://adafru.it/EkO), its an Arduino sketch (CircuitPython does not yet have the ability to play animated GIFs).

Here's the CLUE quickstart:

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

https://adafru.it/Jac

https://adafru.it/Jac

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

https://adafru.it/Jad

https://adafru.it/Jad

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

Arduino Bluefruit nRF52 API

Arduino Bluefruit nRF52 API (https://adafru.it/Ila)

Arduino BLE Examples

Arduino BLE Examples (https://adafru.it/wsF)

CircuitPython on CLUE

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

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

Set up CircuitPython Quick Start!

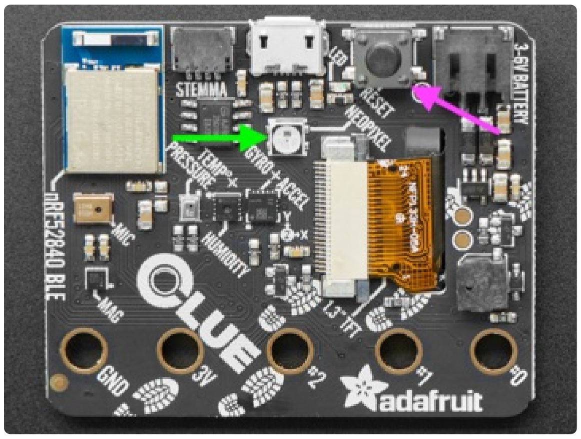
Follow this quick step-by-step for super-fast Python power :)

https://adafru.it/IHF

https://adafru.it/IHF

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

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

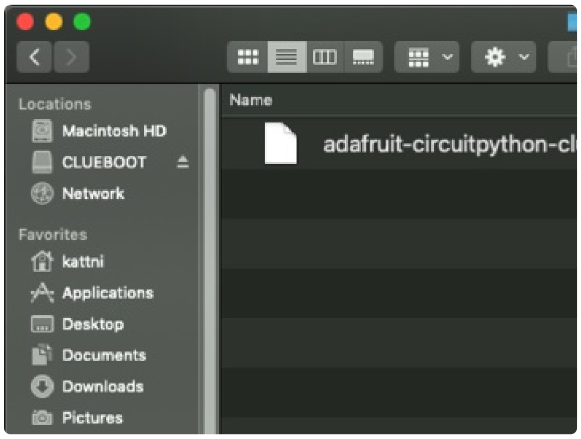


Plug your CLUE into your computer using a known-good USB cable.

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

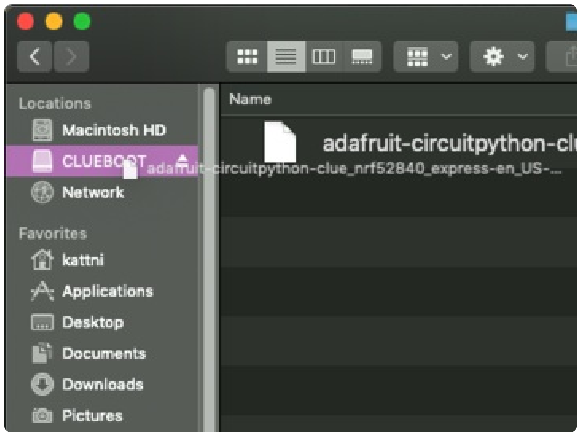
Double-click the **Reset** button on the top (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

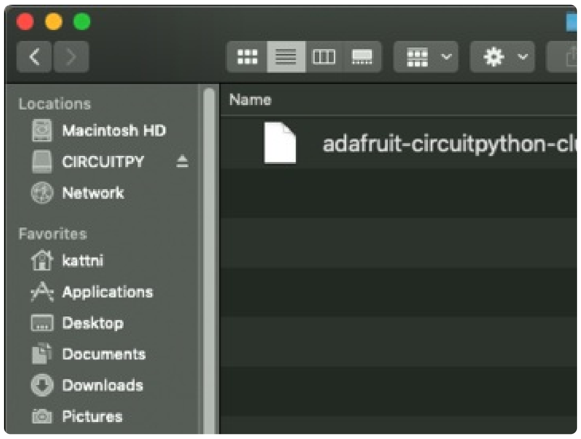
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



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

Drag the **adafruit-circuitpython-clue-etc.uf2** file to **CLUEBOOT.**





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

If this is the first time you're installing CircuitPython or you're doing a completely fresh install after erasing the filesystem, you will have two files - **boot_out.txt**, and **code.py**, and one folder - **lib** on your **CIRCUITPY** drive.

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

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

CLUE CircuitPython Libraries

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

Follow these steps to get the necessary libraries installed.

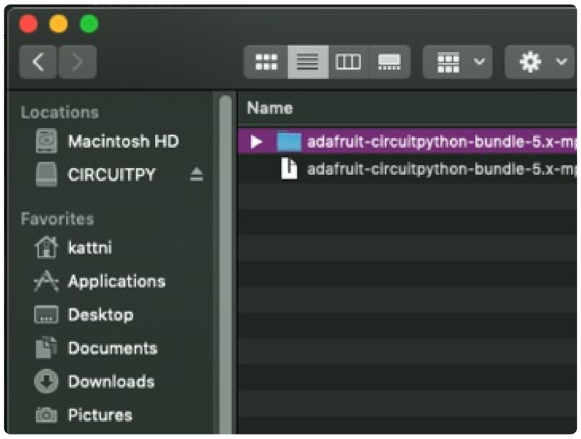
Installing CircuitPython Libraries on your CLUE

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

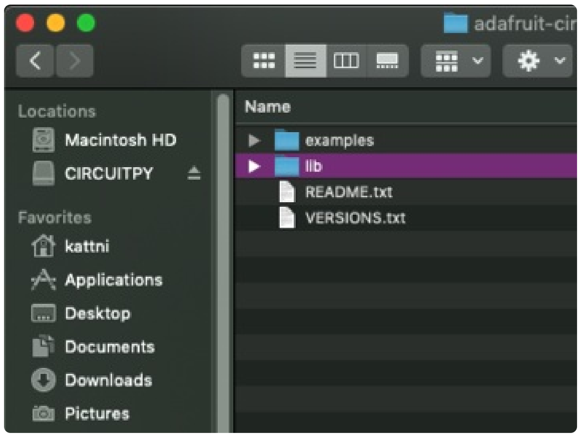
Then, download the CircuitPython library bundle that matches your version of CircuitPython from CircuitPython.org.

https://adafru.it/ENC

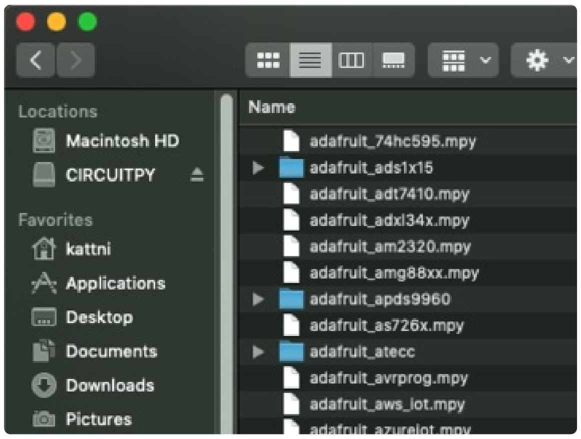
https://adafru.it/ENC



The bundle downloads as a .zip file. Extract the file. Open the resulting folder.

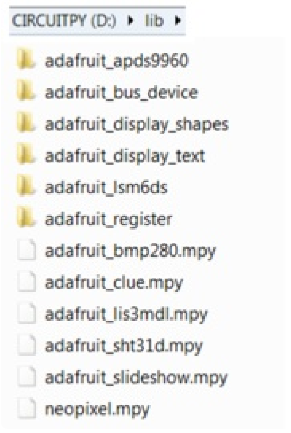


Open the **lib** folder found within.


Locations
Macintosh HD
CIRCUITPY
Favorites
kattni
Applications
Desktop
Documents
Downloads
Pictures

Name
adafruit_74hc595.mpy
adafruit_ads1x15
adafruit_adt7410.mpy
adafruit_adxl34x.mpy
adafruit_am2320.mpy
adafruit_amg88xx.mpy
adafruit_apds9960
adafruit_as726x.mpy
adafruit_atecc
adafruit_avrprog.mpy
adafruit_aws_iot.mpy
adafruit_azureiot.mpy

Once inside, you'll find a lengthy list of folders and .mpy files. To install a CircuitPython library, you drag the file or folder from the **bundle lib folder** to **the lib folder on your CIRCUITPY drive**.


CIRCUITPY (D:) ▸ lib ▸
adafruit_apds9960
adafruit_bus_device
adafruit_display_shapes
adafruit_display_text
adafruit_lsm6ds
adafruit_register
adafruit_bmp280.mpy
adafruit_clue.mpy
adafruit_lis3mdl.mpy
adafruit_sht31d.mpy
adafruit_slideshow.mpy
neopixel.mpy

Copy the following folders and files **from the bundle lib folder** to **the lib folder on your CIRCUITPY drive**:

- **adafruit_apds9960**
- **adafruit_bmp280.mpy**
- **adafruit_bus_device**
- **adafruit_clue.mpy**
- **adafruit_display_shapes**
- **adafruit_display_text**
- **adafruit_lis3mdl.mpy**
- **adafruit_lsm6ds**
- **adafruit_register**
- **adafruit_sht31d.mpy**
- **adafruit_slideshow.mpy**
- **neopixel.mpy**

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

Getting Started with BLE and CircuitPython

Guides

- Getting Started with CircuitPython and Bluetooth Low Energy (https://adafru.it/FxH) - Get started with CircuitPython, the Adafruit nRF52840 and the Bluefruit LE Connect app.
- BLE Light Switch with Feather nRF52840 and Crickit (https://adafru.it/Ile) - Control a robot finger from across the room to flip on and off the lights!
- Color Remote with Circuit Playground Bluefruit (https://adafru.it/Ije) - Mix NeoPixels wirelessly with a Bluetooth LE remote control!
- MagicLight Bulb Color Mixer with Circuit Playground Bluefruit (https://adafru.it/Ilf) - Mix colors on a MagicLight Bulb wirelessly with a Bluetooth LE remote control.
- Bluetooth Turtle Bot with CircuitPython and Crickit (https://adafru.it/Hcx) - Build your own Bluetooth controlled turtle rover!
- Wooden NeoPixel Xmas Tree (https://adafru.it/IlA) - Cut a Christmas tree of wood and mount some NeoPixels in the tree to create a festive yuletide light display.
- Bluefruit TFT Gizmo ANCS Notifier for iOS (https://adafru.it/IlB) - Circuit Playground Bluefruit displays your iOS notification icons so you know when there's fresh activity!
- Bluefruit Playground Hide and Seek (https://adafru.it/HjC) - Use Circuit Playground Bluefruit devices to create a colorful signal strength-based proximity detector!
- Snow Globe with Circuit Playground Bluefruit (https://adafru.it/HgA) - Make your own festive (or creatively odd!) snow globe with custom lighting effects and Bluetooth control.
- Bluetooth Controlled NeoPixel Lightbox (https://adafru.it/IlC) - Great for tracing and writing, this lightbox lets you adjust color and brightness with your phone.
- Circuit Playground Bluefruit NeoPixel Animation and Color Remote Control (https://adafru.it/HE0) - Control NeoPixel colors and animation remotely over Bluetooth with the Circuit Playground Bluefruit!
- Circuit Playground Bluetooth Cauldron (https://adafru.it/IlD) - Build a Bluetooth Controlled Light Up Cauldron.
- NeoPixel Badge Lanyard with Bluetooth LE (https://adafru.it/IlE) - Light up your convention badge and control colors with your phone!
- CircuitPython BLE Controlled NeoPixel Hat (https://adafru.it/IlF) - Wireless control NeoPixels on your wearables!
- Bluefruit nRF52 Feather Learning Guide (https://adafru.it/Chj) - Get started now with our most powerful Bluefruit board yet!
- CircusPython: Jump through Hoops with CircuitPython Bluetooth LE (https://adafru.it/Ima) - Blinka jumps through a ring of fire, controlled via Bluetooth LE and the Bluefruit LE Connect app!
- A CircuitPython BLE Remote Control On/Off Switch (https://adafru.it/Imb) - Make a remote control on/off switch for a computer with CircuitPython and BLE.
- NeoPixel Infinity Cube (https://adafru.it/Imc) - Build a 3D printed, Bluetooth controlled Mirrored Acrylic and NeoPixel Infinity cube.
- CircuitPython BLE Crickit Rover (https://adafru.it/Imd) - Purple Robot with Feather nRF52840 and Crickit plus NeoPixel underlighting!
- Circuit Playground Bluefruit Pumpkin with Lights and Sounds (https://adafru.it/HcB) - Add the Circuit

Playground Bluefruit and STEMMA speaker to an inexpensive plastic pumpkin.

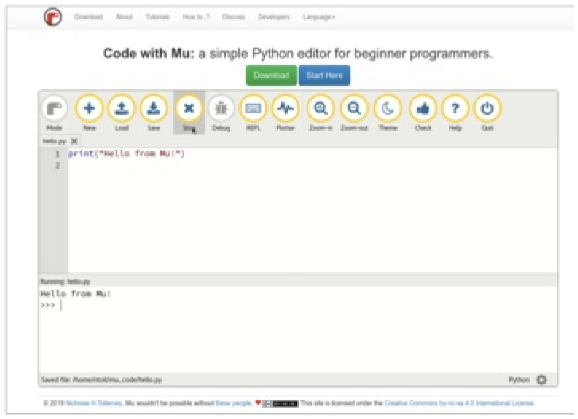
- No-Solder LED Disco Tie with Bluetooth (https://adafru.it/Ime) - Build an LED tie controlled by Bluetooth LE.
- Bluetooth Remote Control for the Lego Droid Developer Kit (https://adafru.it/Imf) - Reinvigorating the Lego Star Wars Droid Developer Kit with an Adafruit powered remote control using Bluetooth LE.

Installing Mu Editor

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

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

Download and Install Mu

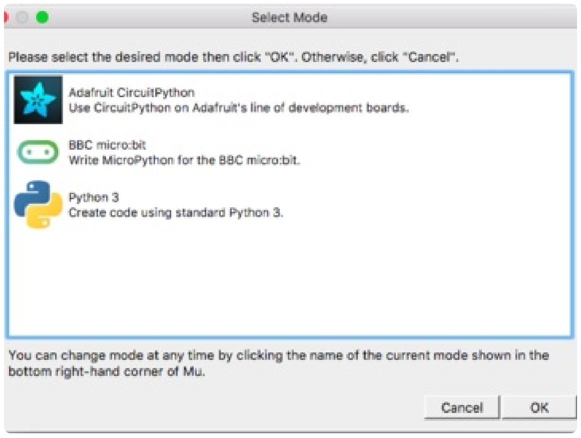

Code with Mu: a simple Python editor for beginner programmers.

```
print("Hello from Mu!")
```

```
Hello from Mu!
>>> |
```

Download Mu from https://codewith.mu (https://adafru.it/Be6). Click the **Download** or **Start Here** links there for downloads and installation instructions. The website has a wealth of other information, including extensive tutorials and and how-to's.

Using Mu


Select Mode

Please select the desired mode then click "OK". Otherwise, click "Cancel".

Adafruit CircuitPython
Use CircuitPython on Adafruit's line of development boards.

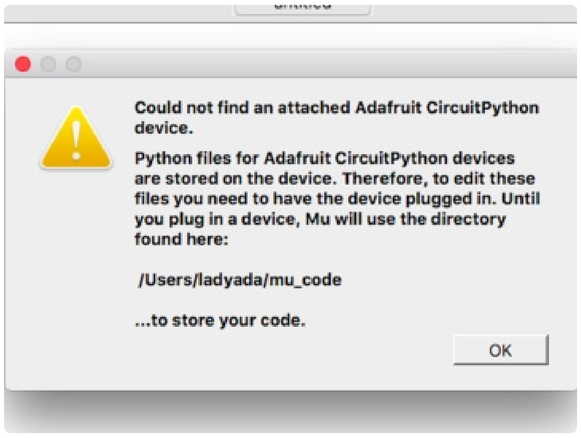
BBC micro:bit
Write MicroPython for the BBC micro:bit.

Python 3
Create code using standard Python 3.

You can change mode at any time by clicking the name of the current mode shown in the bottom right-hand corner of 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.



Could not find an attached Adafruit CircuitPython device.

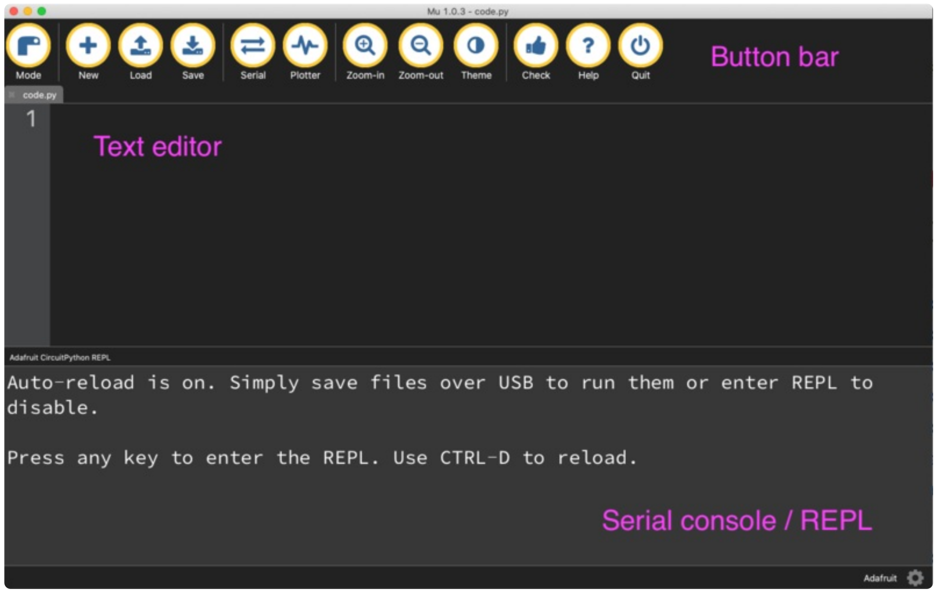
Python files for Adafruit CircuitPython devices are stored on the device. Therefore, to edit these files you need to have the device plugged in. Until you plug in a device, Mu will use the directory found here:

/Users/ladyada/mu_code

...to store your code.

Mu attempts to auto-detect your board, so please plug in your CircuitPython device and make sure it shows up as a **CIRCUITPY** drive before starting 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.



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

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

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

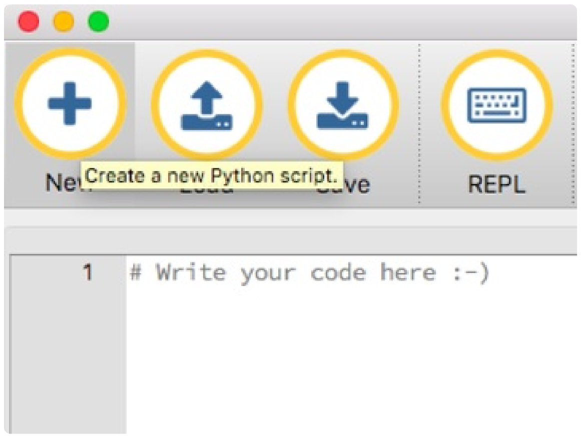
Creating and Editing Code

One of the best things about CircuitPython is how simple it is to get code up and running. In this section, we're going to cover 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. **We strongly recommend 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 basic text editors built into every operating system such as Notepad on Windows, TextEdit on Mac, and gedit on Linux. However, many of these editors don't write back changes immediately to files that you edit. That can cause problems when using CircuitPython. See the Editing Code (https://adafru.it/id3) section below. If you want to skip that section for now, 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



Open your editor, and create a new file. If you are using Mu, click the **New** button in the top left

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 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 QT Py or the Trinkeys!

If you're using QT Py or a Trinkey, please download the [NeoPixel blink example](https://adafru.it/SB2) (https://adafru.it/SB2).

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.

If you are using Adafruit CLUE, you will need to edit the code to use board.D17 as shown below!

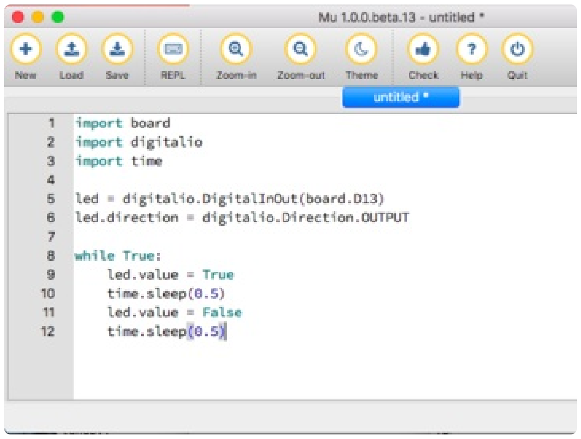
For Adafruit CLUE, you'll need to use `board.D17` instead of `board.LED`. The rest of the code remains the same. Make the following change to the `led =` line:

```
led = digitalio.DigitalInOut(board.D17)
```

If you are using Adafruit ItsyBitsy nRF52840, you will need to edit the code to use board.BLUE_LED as shown below!

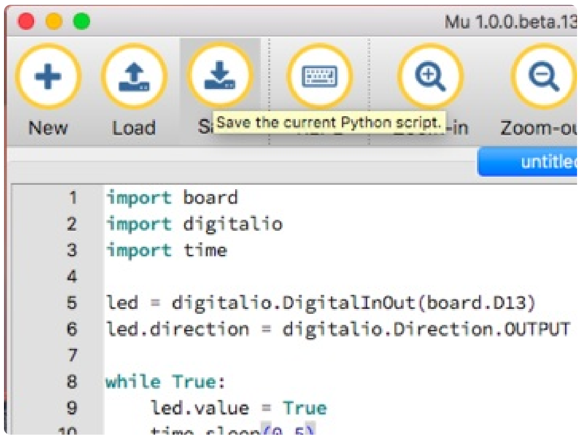
For Adafruit ItsyBitsy nRF52840, you'll need to use `board.BLUE_LED` instead of `board.LED`. The rest of the code remains the same. Make the following change to the `led =` line:

```
led = digitalio.DigitalInOut(board.BLUE_LED)
```

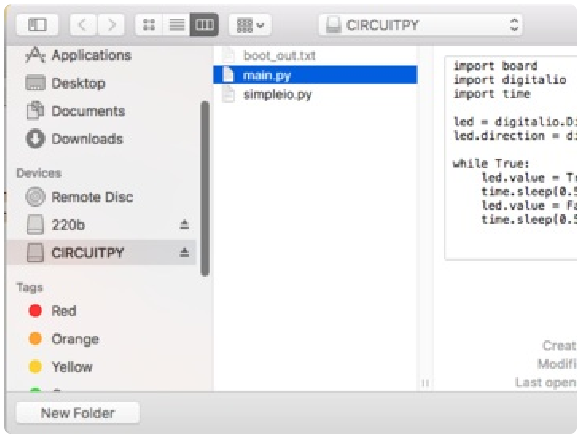


```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

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

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



```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
```



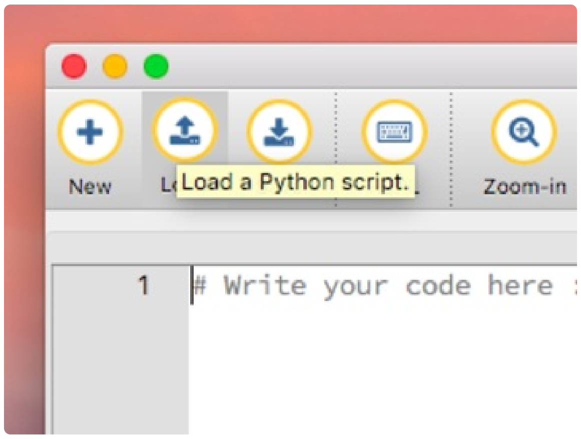
Save this file as **code.py** on your CIRCUITPY drive.

On each board (except the ItsyBitsy nRF52840) you'll find a tiny red LED. On the ItsyBitsy nRF52840, you'll find a tiny blue LED.

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

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

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 just one warning we have to give you before we continue...

Don't Click Reset or Unplug!

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.

However, you must wait until the file is done being saved before unplugging or resetting your board! On Windows using some editors this can sometimes take up to 90 seconds, on Linux it can take 30 seconds to complete because the text editor does not save the file completely. Mac OS does not seem to have this delay, which is nice!

This is really important to be aware of. 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 few ways to avoid this:

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

Recommended editors:

- **mu** (https://adafru.it/Be6) is an editor that safely writes all changes (it's also our recommended editor!)
- **emacs** (https://adafru.it/xNA) is also an editor that will fully write files on save (https://adafru.it/Be7)
- Sublime Text **(https://adafru.it/xNB)** safely writes all changes
- Visual Studio Code **(https://adafru.it/Be9)** appears to safely write all changes
- **gedit** on Linux appears to safely write all changes
- IDLE (https://adafru.it/IWB), in Python 3.8.1 or later, was fixed (https://adafru.it/IWD) to write all changes immediately
- thonny (https://adafru.it/Qb6) fully writes files on save

Recommended *only* with particular settings or with add-ons:

- **vim** (https://adafru.it/ek9) / **vi** safely writes all changes. But set up **vim** to not write swapfiles (https://adafru.it/ELO) (.swp files: temporary records of your edits) to CIRCUITPY. Run vim with `vim -n`, set the `no swapfile` option, or set the `directory` option to write swapfiles elsewhere. Otherwise the swapfile writes trigger restarts of your program.
- The PyCharm IDE **(https://adafru.it/xNC)** is safe if "Safe Write" is turned on in Settings->System Settings->Synchronization (true by default).
- If you are using **Atom** (https://adafru.it/fMG), install the fsync-on-save package (https://adafru.it/E9m) so that it will always write out all changes to files on `CIRCUITPY`.

- **SlickEdit** (https://adafru.it/DdP) works only if you add a macro to flush the disk (https://adafru.it/ven).

We *don't* recommend these editors:

- **notepad** (the default Windows editor) and **Notepad++** can be slow to write, so we recommend the editors above! If you are using notepad, be sure to eject the drive (see below)
- **IDLE** in Python 3.8.0 or earlier does not force out changes immediately
- **nano** (on Linux) does not force out changes
- **geany** (on Linux) does not force out changes
- **Anything else** - we haven't tested other editors so please use a recommended one!

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 (https://adafru.it/Den) 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. We'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? Let's find out!

Exploring Your First CircuitPython Program

First, we'll take a look at the code we're editing.

Here is the original code again:

```
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)
```

Imports & Libraries

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. The files built into CircuitPython are called **modules**, and the files you load separately are called **libraries**. Modules are built into CircuitPython. Libraries are stored on your CIRCUITPY drive in a folder called **lib**.

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

The `import` statements tells the board that you're going to use a particular library in your code. In this example, we imported three modules: `board`, `digitalio`, and `time`. All three of these modules are built into CircuitPython, so no separate library files are needed. That's one of the things that makes this an excellent first example. You don't need any thing extra to make it work! `board` gives you access to the *hardware on your board*, `digitalio` lets you *access that hardware as inputs/outputs* and `time` let's you pass time by 'sleeping'

Setting Up The LED

The next two lines setup the code to use the LED.

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

Your board knows the red LED as `LED`. So, we initialise that pin, and we set it to output. We set `led` to equal the rest of that information so we don't have to type it all out again later in our code.

Loop-de-loops

The third section starts with a `while` statement. `while True:` essentially means, "forever do the following:". `while True:` creates a loop. Code will loop "while" the condition is "true" (vs. false), and as `True` is never False, the code will loop forever. All code that is indented under `while True:` is "inside" the loop.

Inside our loop, we have four items:

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

First, we have `led.value = True`. This line tells the LED to turn on. On the next line, we have `time.sleep(0.5)`. This line is telling CircuitPython to pause running code for 0.5 seconds. Since this is between turning the led on and off, the led will be on for 0.5 seconds.

The next two lines are similar. `led.value = False` tells the LED to turn off, and `time.sleep(0.5)` tells CircuitPython to pause for another 0.5 seconds. This occurs between turning the led off and back on so the LED will be off for 0.5 seconds too.

Then the loop will begin again, and continue to do so as long as the code is running!

So, when you changed the first `0.5` to `0.1`, you decreased the amount of time that the code leaves the LED on. So it blinks on really quickly before turning off!

Great job! You've edited code in a CircuitPython program!

What Happens When My Code Finishes Running?

When your code finishes running, CircuitPython resets your microcontroller board to prepare it for the next run of code. That means any set up you did earlier no longer applies, and the pin states are reset.

For example, try reducing the above example to `led.value = True`. The LED will flash almost too quickly to see, and turn off. This is because the code finishes running and resets the pin state, and the LED is no longer receiving a signal.

To that end, most CircuitPython programs involve some kind of loop, infinite or otherwise

What if I don't have the loop?

If you don't have the loop, the code will run to the end and exit. This can lead to some unexpected behavior in simple programs like this since the "exit" also resets the state of the hardware. This is a different behavior than running commands via REPL. So if you are writing a simple program that doesn't seem to work, you may need to add a loop to the end so the program doesn't exit.

The simplest loop would be:

```
while True:
    pass
```

And remember - you can press to exit the loop.

See also the Behavior section in the docs (https://adafru.it/Bvz).

More Changes

We 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 we suggest using **code.py** as 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 looks like this:

`print("Hello, world!")`

This line 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 print those too.

The serial console requires a terminal program. A terminal is a program that gives you a text-based interface to perform various tasks.

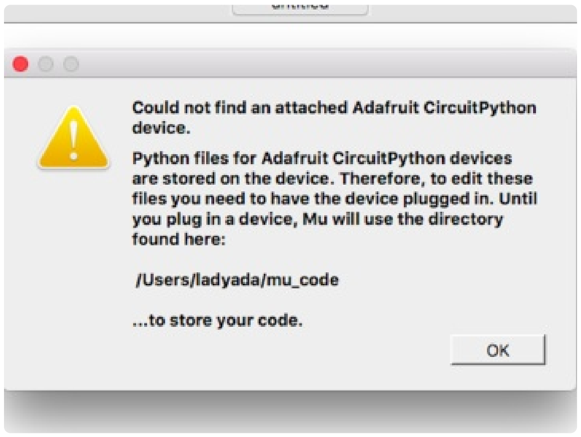
If you're on Linux, and are seeing multi-second delays connecting to the serial console, or are seeing "AT" and other gibberish when you connect, then the modemmanager service might be interfering. Just remove it; it doesn't have much use unless you're still using dial-up modems. To remove, type this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

```
sudo apt purge modemmanager
```

Are you using Mu?

If so, good news! The serial console **is built into Mu** and will **autodetect your board** making using the REPL *really really easy*.

Please note that Mu does yet not work with nRF52 or ESP8266-based CircuitPython boards, skip down to the next section for details on using a terminal program.



Could not find an attached Adafruit CircuitPython device.

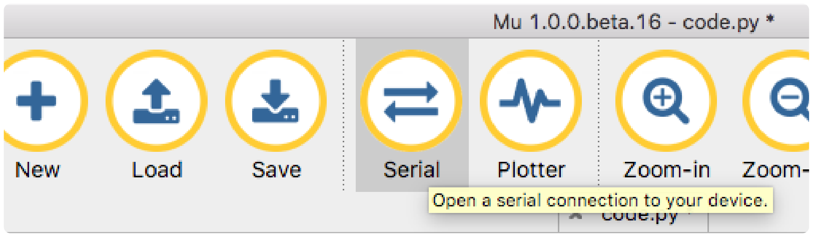
Python files for Adafruit CircuitPython devices are stored on the device. Therefore, to edit these files you need to have the device plugged in. Until you plug in a device, Mu will use the directory found here:

/Users/ladyada/mu_code

...to store your code.

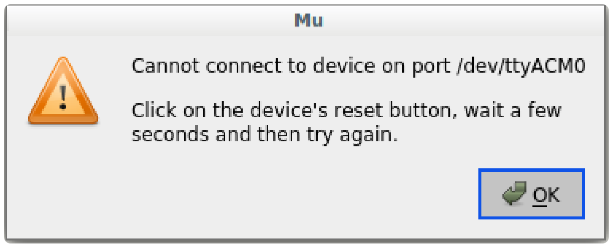
First, make sure your CircuitPython board is plugged in. If you are using Windows 7, make sure you installed the drivers (https://adafru.it/Amd).

Once in Mu, look for the **Serial** button in the menu and click it.


Mu 1.0.0.beta.16 - code.py *
New
Load
Save
Serial
Plotter
Zoom-in
Zoom-
Open a serial connection to your device.
code.py

Setting Permissions on Linux

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



Mu

Cannot connect to device on port /dev/ttyACM0

Click on the device's reset button, wait a few seconds and then try again.

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 Advanced Serial Console on Mac and

Linux (https://adafru.it/AAI) for details on how to add yourself to the right group.

Using Something Else?

If you're not using Mu to edit, are using ESP8266 or nRF52 CircuitPython, or if for some reason you are not a fan of the built in serial console, you can run the serial console as a separate program.

Windows requires you to download a terminal program, check out this page for more details (https://adafru.it/AAH)

Mac and Linux both have one built in, though other options are available for download, check this page for more details (https://adafru.it/AAI)

Interacting with the Serial Console

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

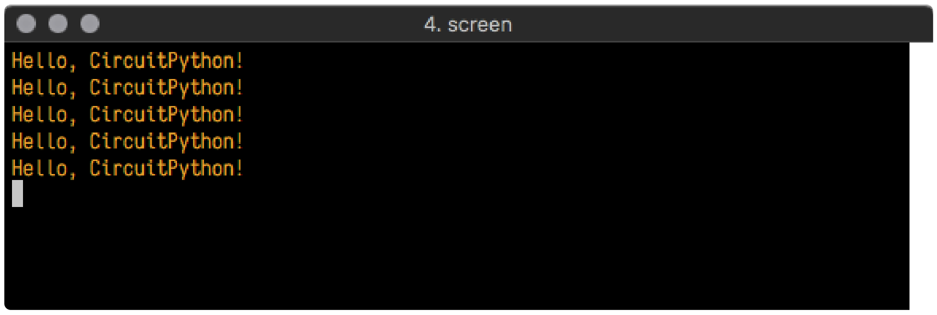
The code you wrote earlier has no output to the serial console. So, we're going to edit it to create some output.

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)
```

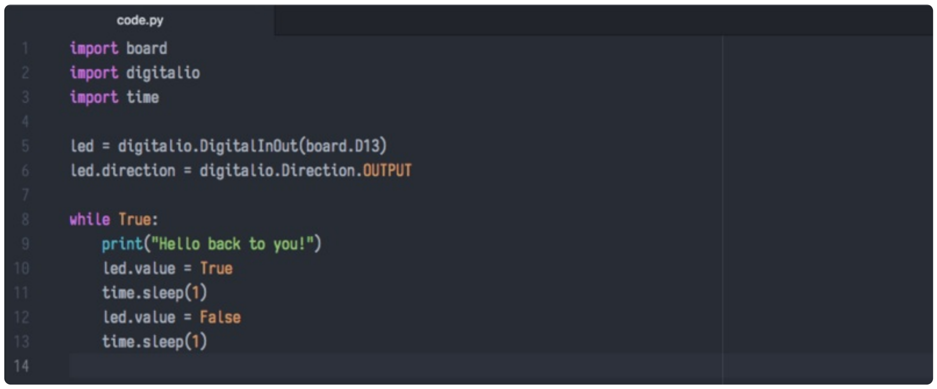
Save your file.

Now, let's go take a look at the window with our connection to the serial console.



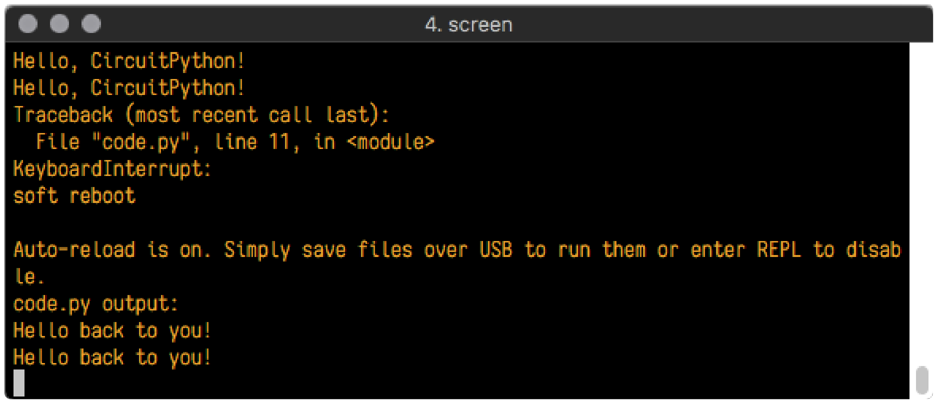
```
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
```

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



```python
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

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

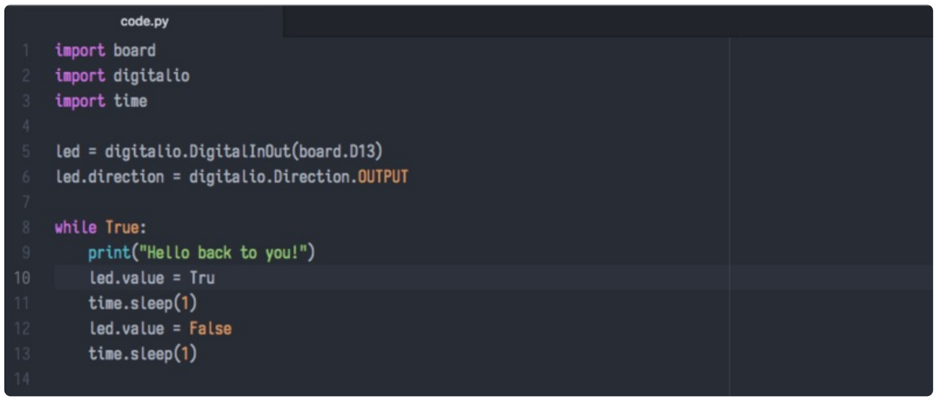
Keep your serial console window where you can see it. Save your file. You'll see what the serial console displays when the board reboots. Then you'll see your new change!



```
Hello, CircuitPython!
Hello, CircuitPython!
Traceback (most recent call last):
  File "code.py", line 11, in <module>
KeyboardInterrupt:
soft reboot

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

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 we can see how it is used.

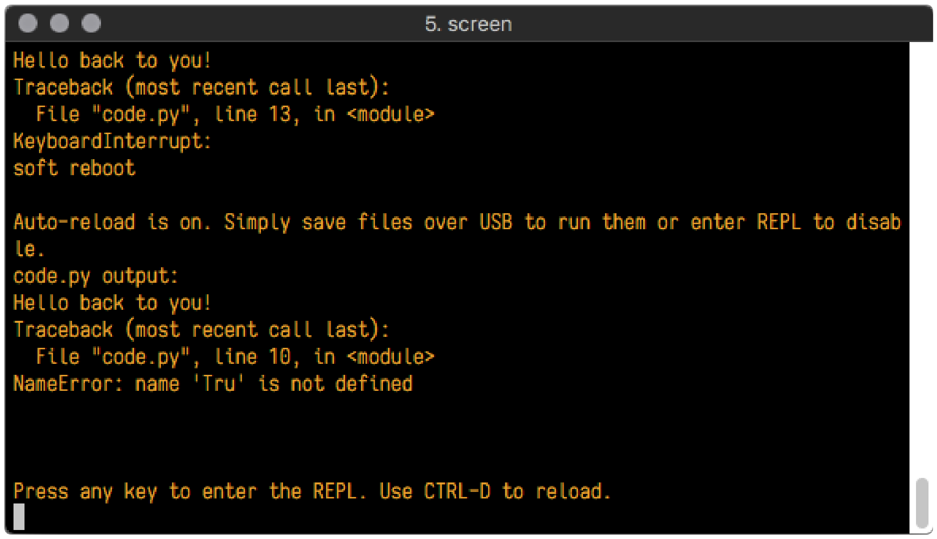
Delete the `e` at the end of `True` from the line `led.value = True` so that it says `led.value = Tru`



```python
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.D13)
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. We need to fix it!

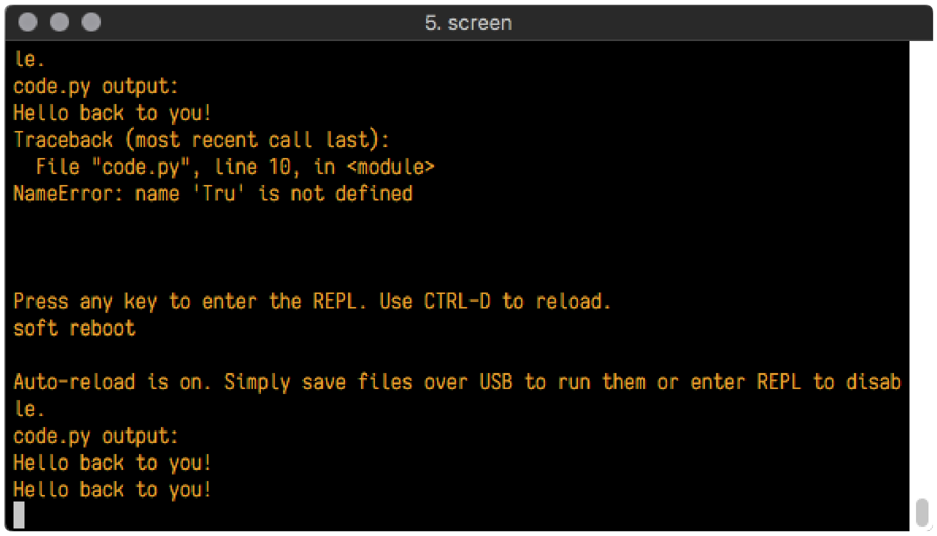
Usually when you run into errors, it's not because you introduced them on purpose. You may have 200 lines of code, and have no idea where your error could be hiding. This is where the serial console can help. Let's take a look!



```
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 13, in <module>
KeyboardInterrupt:
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.
code.py output:
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 10, in <module>
NameError: name 'Tru' is not defined



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

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

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



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



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

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

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

The serial console will display any output generated by your code. Some sensors, such as a humidity sensor or a thermistor, receive data and you can use print statements to display that information. You can also use print statements for troubleshooting. 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!

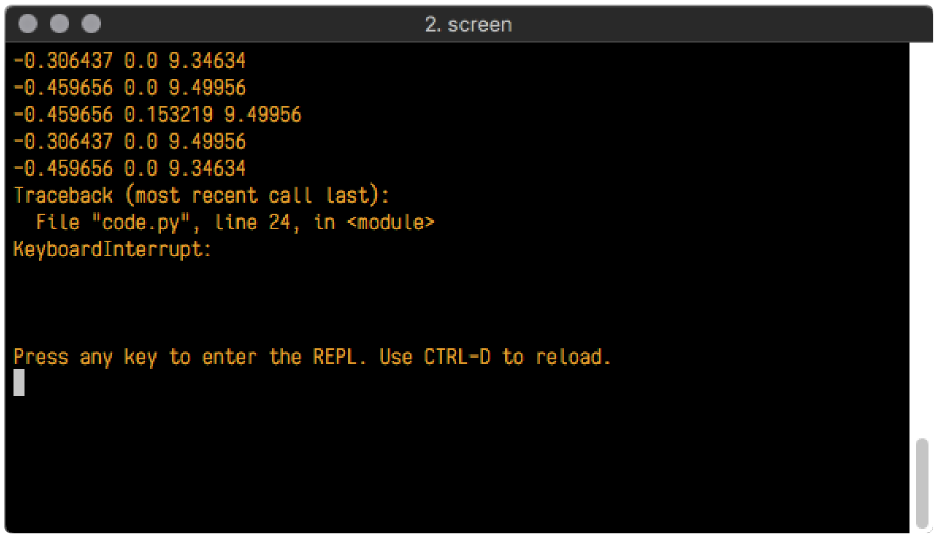
The REPL

The other feature of the serial connection is the **R**ead-**E**valuate-**P**rint-**L**oop, 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.

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**.

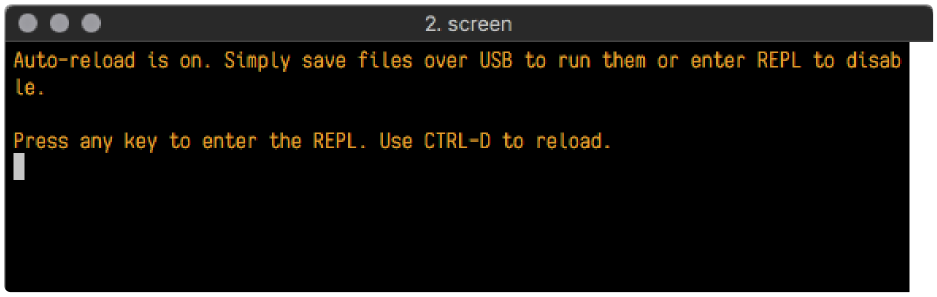
If there is code running, it will stop and you'll see `Press any key to enter the REPL. Use CTRL-D to reload.` Follow those instructions, and press any key on your keyboard.

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.



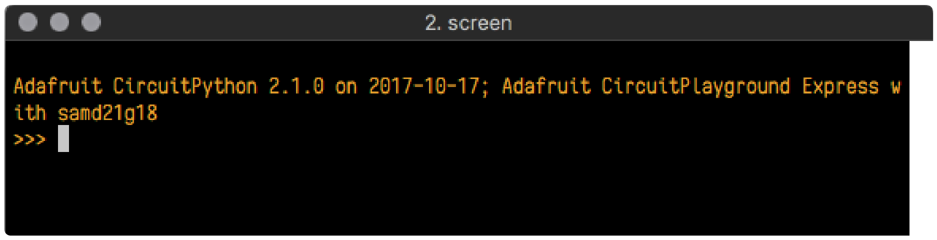
```
-0.306437 0.0 9.34634
-0.459656 0.0 9.49956
-0.459656 0.153219 9.49956
-0.306437 0.0 9.49956
-0.459656 0.0 9.34634
Traceback (most recent call last):
  File "code.py", line 24, in <module>
KeyboardInterrupt:



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

If there is no code running, you will enter the REPL immediately after pressing Ctrl + C. 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 disab
le.

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

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



```
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit CircuitPlayground Express w
ith samd21g18
>>> 
```

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 2.1.0 on 2017-10-17; Adafruit CircuitPlayground Express with samd21g18
```

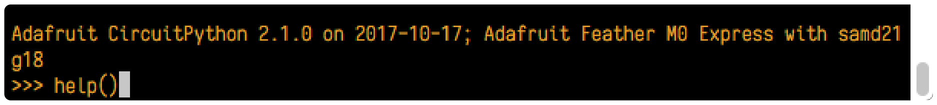
This line tells you the version of CircuitPython you're using and when it was released. Next, it gives you the type of board you're using and the type of microcontroller the board uses. Each part of this may be different for your board depending on the versions you're working with.

This is followed by the CircuitPython prompt.

```
>>>
```

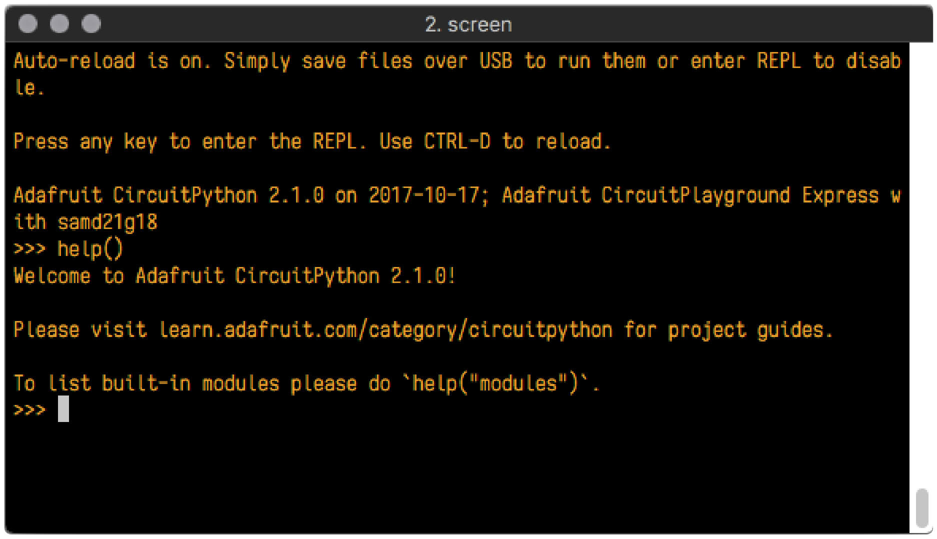
From this prompt you can run all sorts of commands and code. The first thing we'll do is run `help()`. This will tell us where to start exploring the REPL. To run code in the REPL, type it in next to the REPL prompt.

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



```
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Feather M0 Express with samd21
g18
>>> help()
```

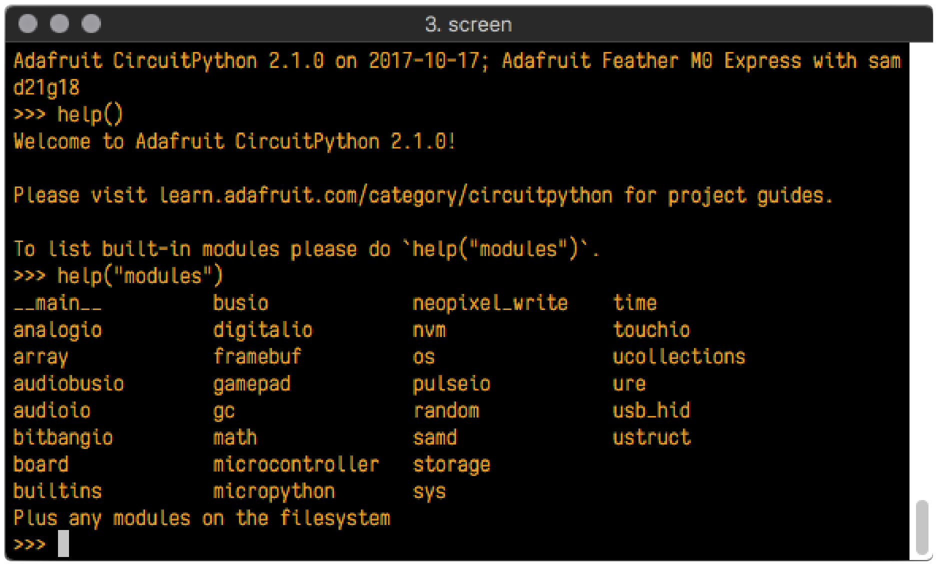
Then press enter. You should then see a message.



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

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

Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit CircuitPlayground Express w
ith samd21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

Please visit learn.adafruit.com/category/circuitpython for project guides.

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

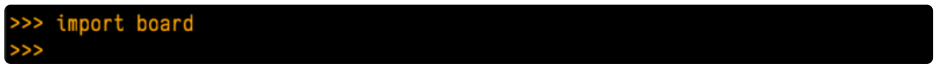
First part of the message is another reference to the version of CircuitPython you're using. Second, a URL for the CircuitPython related project guides. Then... wait. What's this? `To list built-in modules, please do `help("modules")`.` Remember the libraries you learned about while going through creating 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.

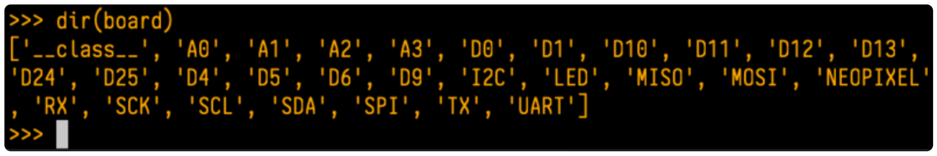

```
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Feather M0 Express with sam
d21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

Please visit learn.adafruit.com/category/circuitpython for project guides.

To list built-in modules please do `help("modules")`.
>>> help("modules")
__main__          busio             neopixel_write    time
analogio          digitalio         nvm               touchio
array             framebuf          os                ucollections
audiobusio        gamepad           pulseio           ure
audioio           gc                random            usb_hid
bitbangio         math              samd              ustruct
board             microcontroller   storage
builtins          micropython       sys
Plus any modules on the filesystem
>>>
```

This is a list of all the core libraries built into CircuitPython. We discussed how 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__', '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']
>>>
```

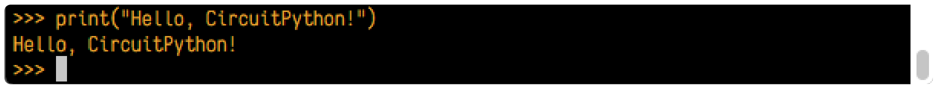
This is a list of all of the pins on your board that are available for you to use in your code. Each board's list will differ slightly depending on the number of pins available. Do you see `LED`? That's the pin you used to blink the red LED!

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." We're going to say hello to something else. Type into the REPL:

`print("Hello, CircuitPython!")`

Then press enter.



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

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

You can write single lines of code that run stand-alone. You can also write entire programs into the REPL to test them. As we said though, 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 libraries are available and explore those libraries.

Try typing more into the REPL to see what happens!

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.

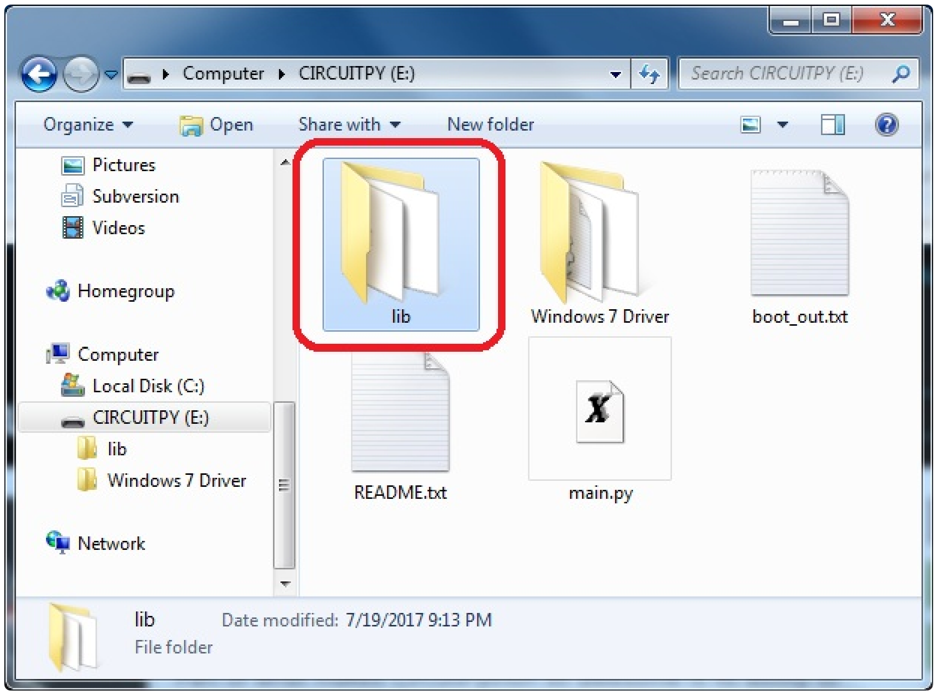
You can return to the REPL at any time!

CircuitPython Libraries

As 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.

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 awesome is its ability to store code separately from the firmware itself. Storing code separately from the firmware makes it easier to update both the code you write and the libraries you depend.

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



CircuitPython libraries work in the same way as regular Python modules so the Python docs (https://adafru.it/rar) are a great reference for how it all should work. In Python terms, we can place our library files in the **lib** directory because its 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, we provide a bundle full of our libraries.

Our bundle and releases 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.

Installing the CircuitPython Library Bundle

We're constantly updating and improving our libraries, so we don't (at this time) ship our CircuitPython boards with the full library bundle. Instead, you can find example code in the guides for your board that depends on external libraries. Some of these libraries may be available from us at Adafruit, some may be written by community members!

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

You can grab the latest Adafruit CircuitPython Bundle release by clicking the button below.

Note: Match up the bundle version with the version of CircuitPython you are running - 3.x library for running any version of CircuitPython 3, 4.x for running any version of CircuitPython 4, etc. If you mix libraries with major CircuitPython versions, you will most likely get errors due to changes in library interfaces possible during major version changes.

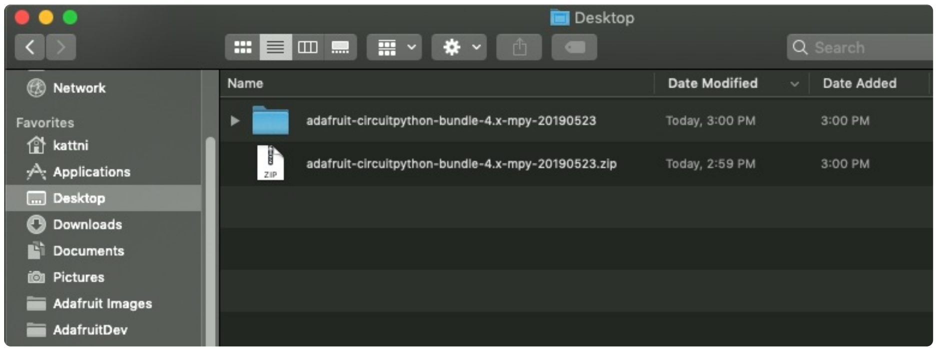
https://adafru.it/ENC

https://adafru.it/ENC

If you need another version, you can also visit the bundle release page (https://adafru.it/Ayy) which will let you select exactly what version you're looking for, as well as information about changes.

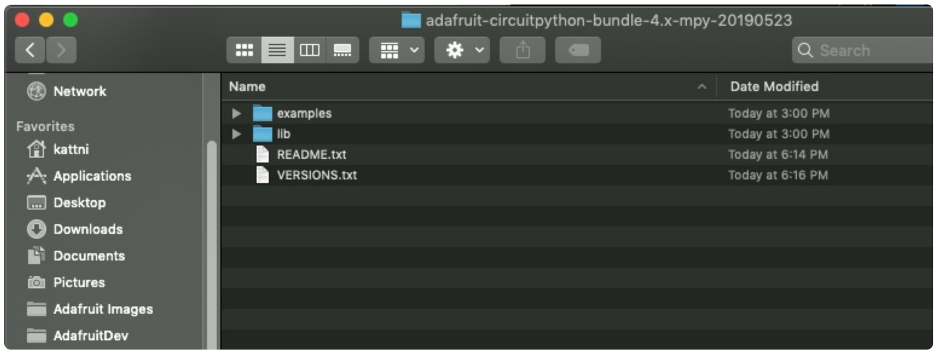
Either way, download the version that matches your CircuitPython firmware version. If you don't know the version, look at the initial prompt in the CircuitPython REPL, which reports the version. For example, if you're running v4.0.1, download the 4.x library bundle. There's also a **py** bundle which contains the uncompressed python files, you probably *don't* want that unless you are doing advanced work on libraries.

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.



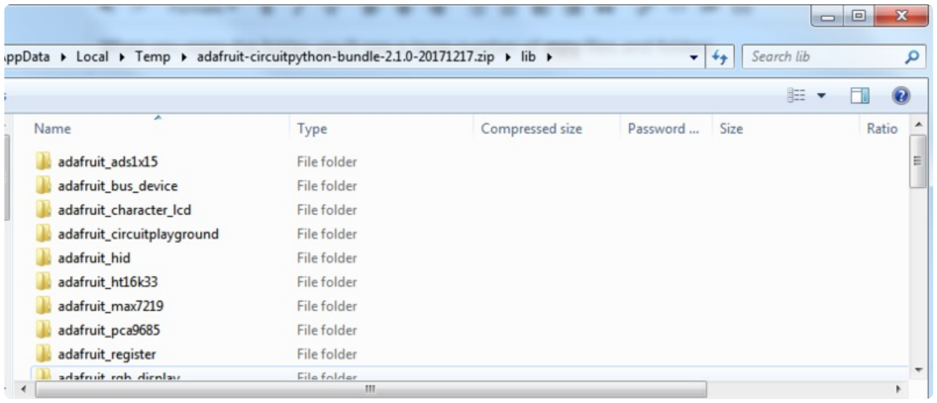
Name	Date Modified	Date Added
adafruit-circuitpython-bundle-4.x-mpy-20190523	Today, 3:00 PM	3:00 PM
adafruit-circuitpython-bundle-4.x-mpy-20190523.zip	Today, 2:59 PM	3:00 PM

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.



Name	Date Modified
examples	Today at 3:00 PM
lib	Today at 3:00 PM
README.txt	Today at 6:14 PM
VERSIONS.txt	Today at 6:16 PM

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

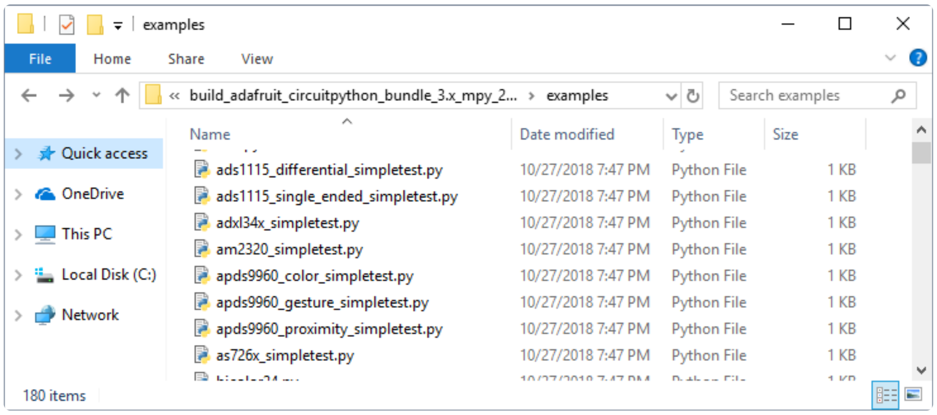

ppData ▸ Local ▸ Temp ▸ adafruit-circuitpython-bundle-2.1.0-20171217.zip ▸ lib ▸
Search lib

Name	Type	Compressed size	Password ...	Size	Ratio
adafruit_ads1x15	File folder				
adafruit_bus_device	File folder				
adafruit_character_lcd	File folder				
adafruit_circuitplayground	File folder				
adafruit_hid	File folder				
adafruit_ht16k33	File folder				
adafruit_max7219	File folder				
adafruit_pca9685	File folder				
adafruit_register	File folder				
adafruit_rgb_display	File folder				

Example Files

All example files from each library are now included in the bundles, 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.



Name	Date modified	Type	Size
ads1115_differential_simpletest.py	10/27/2018 7:47 PM	Python File	1 KB
ads1115_single_ended_simpletest.py	10/27/2018 7:47 PM	Python File	1 KB
adxl34x_simpletest.py	10/27/2018 7:47 PM	Python File	1 KB
am2320_simpletest.py	10/27/2018 7:47 PM	Python File	1 KB
apds9960_color_simpletest.py	10/27/2018 7:47 PM	Python File	1 KB
apds9960_gesture_simpletest.py	10/27/2018 7:47 PM	Python File	1 KB
apds9960_proximity_simpletest.py	10/27/2018 7:47 PM	Python File	1 KB
as726x_simpletest.py	10/27/2018 7:47 PM	Python File	1 KB

Copying Libraries to Your Board

First you'll want to create a **lib** folder on your **CIRCUITPY** drive. Open the drive, right click, choose the option to create a new folder, and call it **lib**. 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**.

This also applies to example files. They are only supplied as raw **.py** files, so they may need to be converted to **.mpy** using the **mpy-cross** utility if you encounter `MemoryErrors`. This is discussed in the CircuitPython Essentials Guide (https://adafru.it/CTw). Usage is the same as described above in the Express Boards section. Note: If you do not place examples in a separate folder, you would remove the examples from the `import` statement.

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

Example: `ImportError` Due to Missing Library

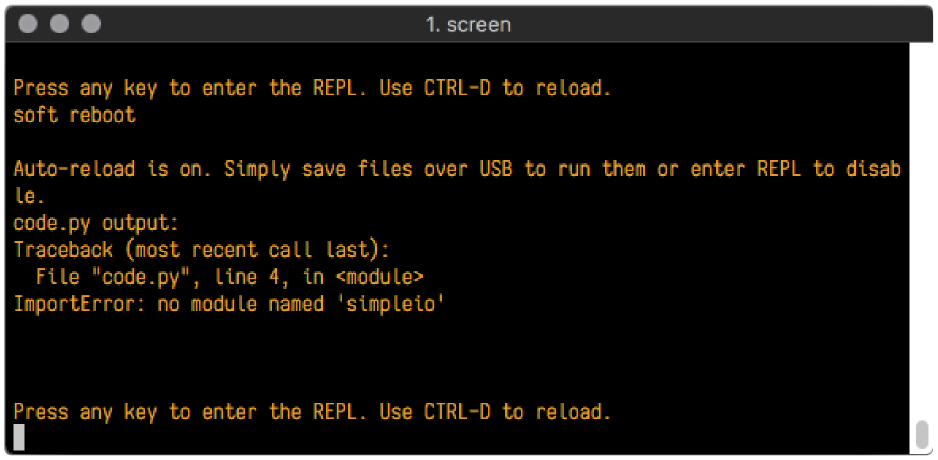
If you choose to load libraries as you need them, you may write up code that tries to use a library you haven't yet loaded. We're going to demonstrate what happens when you try to utilise a library that you don't have loaded on your board, and cover the steps required to resolve the issue.

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 blinky example.

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.D13)

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.

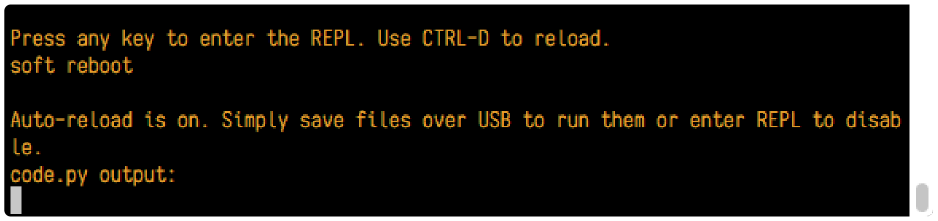


```
Press any key to enter the REPL. Use CTRL-D to reload.
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.
code.py output:
Traceback (most recent call last):
  File "code.py", line 4, in <module>
ImportError: no module named 'simpleio'



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

We have an `ImportError`. It says there is `no module named 'simpleio'`. That's the one we just included in our 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 we're looking for! Follow the steps above to load an individual library file.

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



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

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

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 a Trinket M0 or Gemma M0, you'll want to follow the same steps in the example above to install libraries as you need them. You don't always need to wait for an `ImportError` as you probably know what library you added to your code. Simply open the **lib** folder you downloaded, find the library you need, and drag it to the **lib** folder on your **CIRCUITPY** drive.

You may end up running out of space on your Trinket M0 or Gemma M0 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 them in the Troubleshooting page in the Learn guides for your board.

Updating CircuitPython Libraries/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.

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.

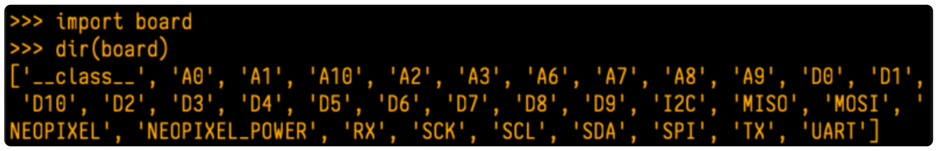
`import board`

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.



```
>>> import board
>>> dir(board)
['__class__', 'A0', 'A1', 'A10', 'A2', 'A3', 'A6', 'A7', 'A8', 'A9', 'D0', 'D1',
 'D10', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9', 'I2C', 'MISO', 'MOSI', '
NEOPIXEL', 'NEOPIXEL_POWER', 'RX', 'SCK', 'SCL', 'SDA', 'SPI', 'TX', 'UART']
```

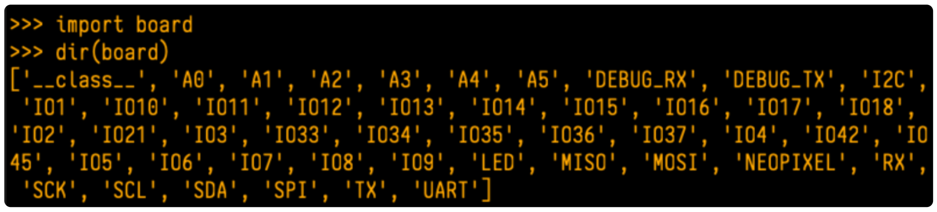
The following pins have labels on the physical QT Py board: A0, A1, A2, A3, SDA, SCL, TX, RX, SCK, MISO, and MOSI. You see that there are many more entries available in `board` than the labels on the QT Py.

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?](https://adafru.it/QkA) (https://adafru.it/QkA) 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())
```

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.

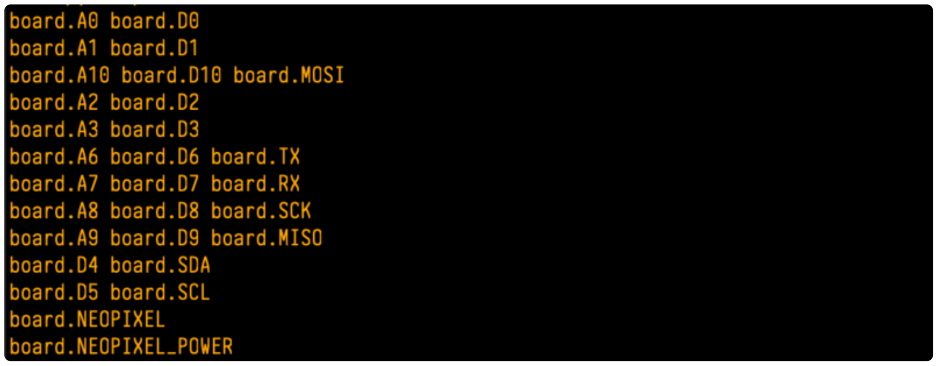
What Are All the Available Names?

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, connect to the serial console. Then, save the following as **code.py** on your **CIRCUITPY** drive.

```
"""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:



```
board.A0 board.D0
board.A1 board.D1
board.A10 board.D10 board.MOSI
board.A2 board.D2
board.A3 board.D3
board.A6 board.D6 board.TX
board.A7 board.D7 board.RX
board.A8 board.D8 board.SCK
board.A9 board.D9 board.MISO
board.D4 board.SDA
board.D5 board.SCL
board.NEOPIXEL
board.NEOPIXEL_POWER
```

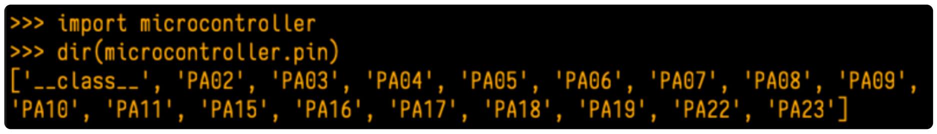
Each line represents a single pin. Find the line containing the pin name that's labeled on the physical board, and you'll find the other names available for that pin. For example, the first pin on the board is labeled **A0**. The first line in the output is `board.A0 board.D0`. This means that you can 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']
```

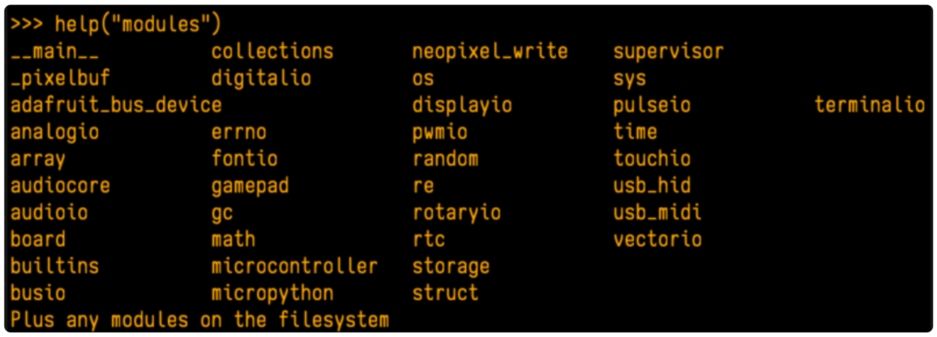
CircuitPython Built-In Modules

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 (https://adafru.it/QkB) and the Python-like modules included here (https://adafru.it/QkC). 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 (https://adafru.it/N2a), 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.

Advanced Serial Console on Mac and Linux

Connecting to the serial console on Mac and Linux uses essentially the same process. Neither operating system needs drivers installed. On MacOSX, **Terminal comes** installed. On Linux, there are a variety such as gnome-terminal (called Terminal) or Konsole on KDE.

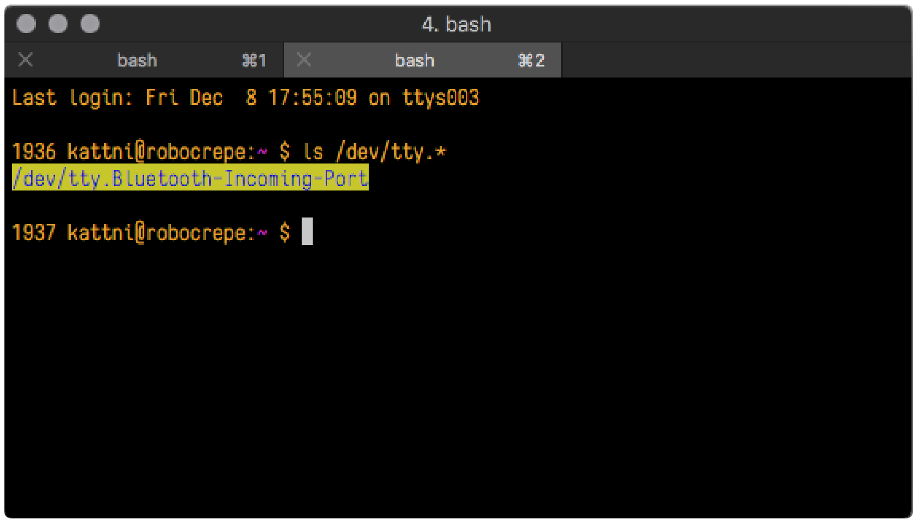
What's the Port?

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.

We're going to use Terminal to determine what 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. On Mac, 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, we'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.

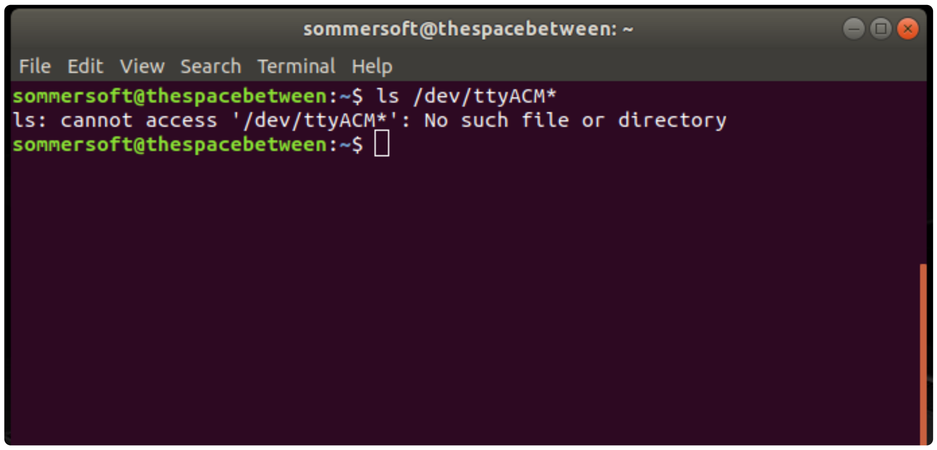


```
Last login: Fri Dec  8 17:55:09 on ttys003

1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port

1937 kattni@robocrepe:~ $ 
```

For Linux, the procedure is the same, however, the name is slightly different. If you're using Linux, you'll type:

`ls /dev/ttyACM*`

The concept is the same with Linux. We are asking to see the listings in the `/dev/` folder, starting with `ttyACM` and ending with anything. This will show you the current serial connections. In the example below, the error is indicating that are no current serial connections starting with `ttyACM`.

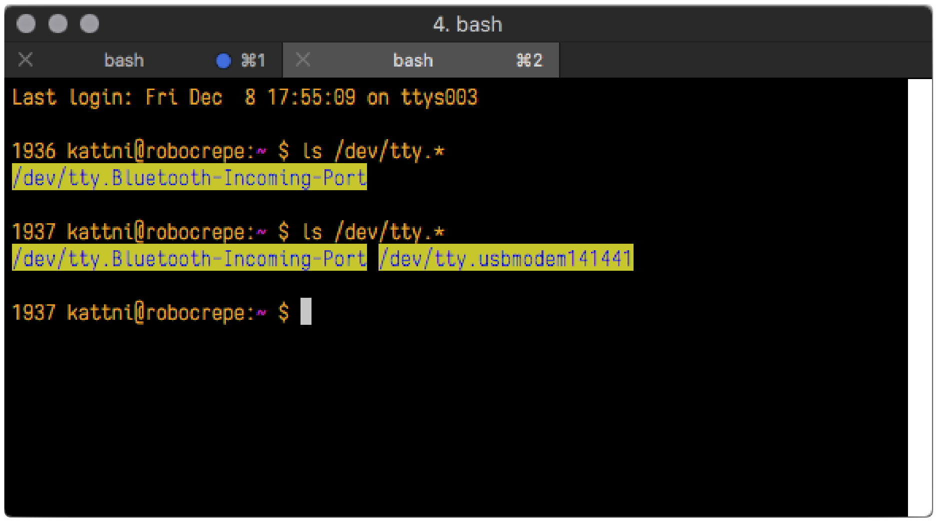

sommersoft@thespacebetween: ~
File Edit View Search Terminal Help

```
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
sommersoft@thespacebetween:~$
```

Now, plug your board. Using Mac, type:

`ls /dev/tty.*`

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



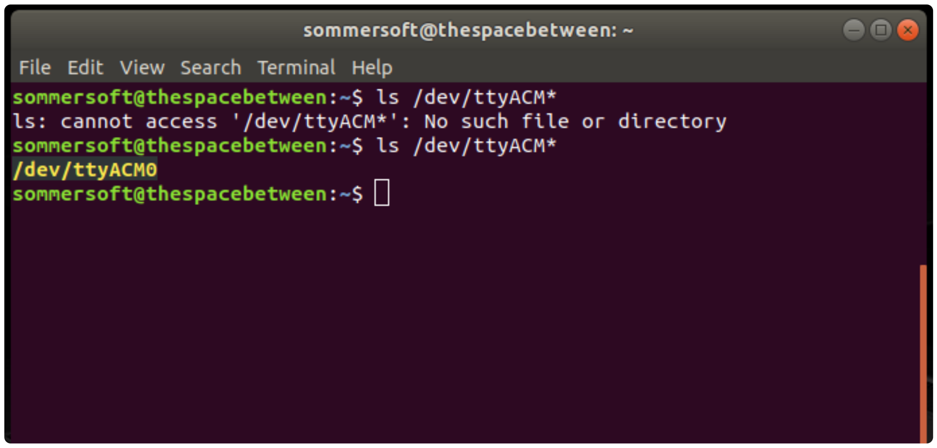
```
Last login: Fri Dec  8 17:55:09 on ttys003

1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port

1937 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441

1937 kattni@robocrepe:~ $
```

Using Mac, 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, type:

`ls /dev/ttyACM*`

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



```
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
/dev/ttyACM0
sommersoft@thespacebetween:~$ 
```

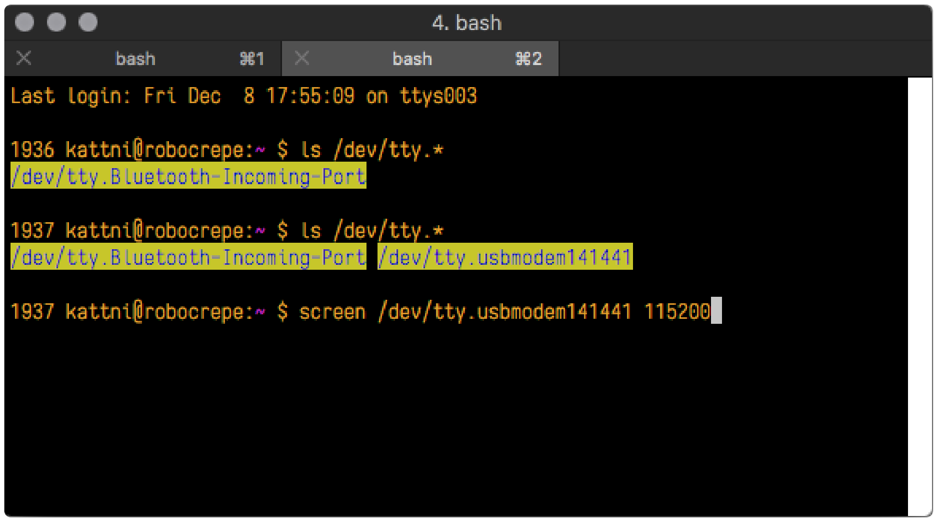
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

Connect with screen

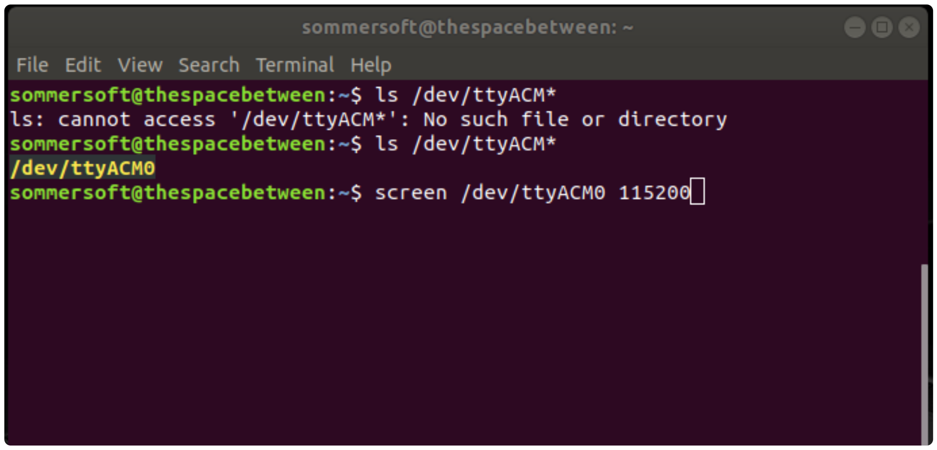
Now that you know the name your board is using, you're ready connect to the serial console. We're going to use a command called `screen`. The `screen` command is included with MacOS. Linux users may need to install it using their package manager. To connect to the serial console, use Terminal. Type the 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.



```
Last login: Fri Dec  8 17:55:09 on ttys003

1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port

1937 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441

1937 kattni@robocrepe:~ $ screen /dev/tty.usbmodem141441 115200
```


sommersoft@thespacebetween: ~
File Edit View Search Terminal Help

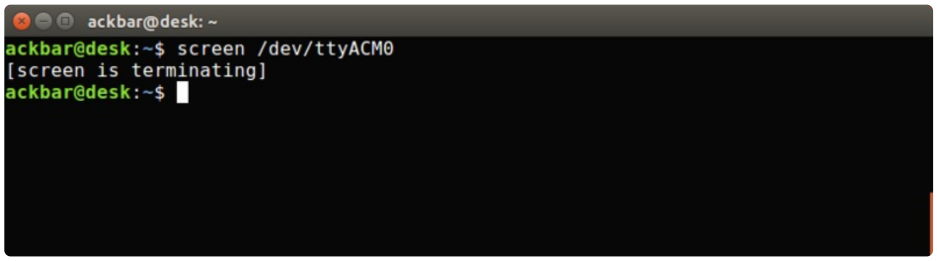
```
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
/dev/ttyACM0
sommersoft@thespacebetween:~$ screen /dev/ttyACM0 115200
```

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!

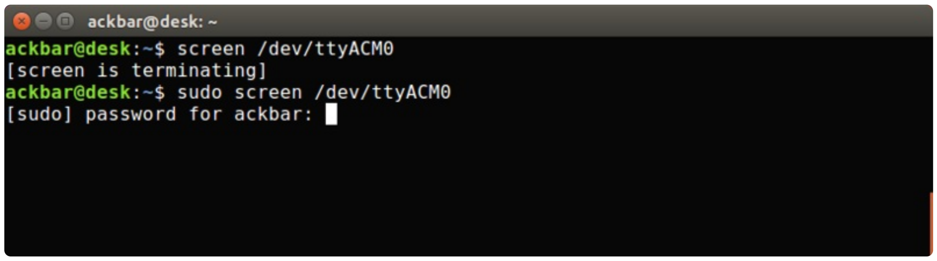
Permissions on Linux

If you try to run `screen` and it doesn't work, then you may be running into an issue with permissions. Linux keeps track of users and groups and what they are allowed to do and not do, like access the hardware associated with the serial connection for running `screen`. So if you see something like this:



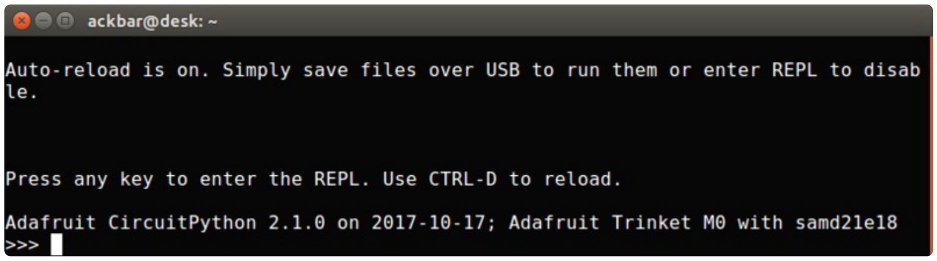
```
ackbar@desk:~$ screen /dev/ttyACM0
[screen is terminating]
ackbar@desk:~$
```

then you may need to grant yourself access. There are generally two ways you can do this. The first is to just run `screen` using the `sudo` command, which temporarily gives you elevated privileges.



```
ackbar@desk:~$ screen /dev/ttyACM0
[screen is terminating]
ackbar@desk:~$ sudo screen /dev/ttyACM0
[sudo] password for ackbar:
```

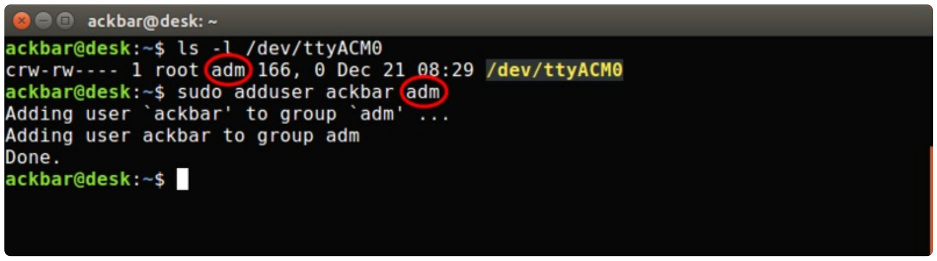
Once you enter your password, you should be in:



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


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

Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Trinket M0 with samd21e18
>>>
```

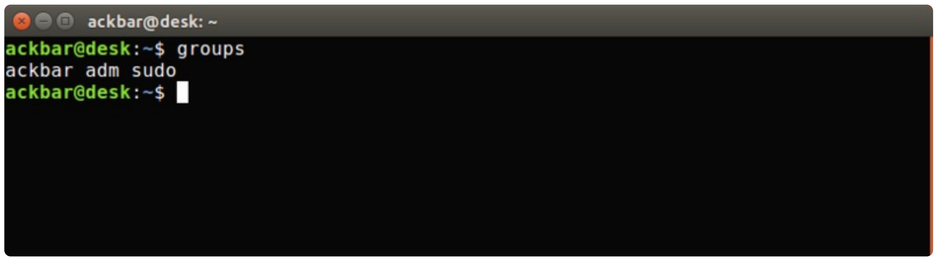
The second way is to add yourself to the group associated with the hardware. To figure out what that group is, use the command `ls -l` as shown below. The group name is circled in red.

Then use the command `adduser` to add yourself to that group. You need elevated privileges to do this, so you'll need to use `sudo`. In the example below, the group is **adm** and the user is **ackbar**.



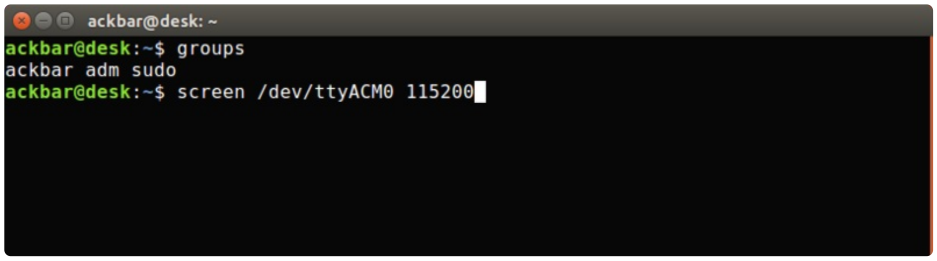
```
ackbar@desk:~$ ls -l /dev/ttyACM0
crw-rw---- 1 root adm 166, 0 Dec 21 08:29 /dev/ttyACM0
ackbar@desk:~$ sudo adduser ackbar adm
Adding user `ackbar' to group `adm' ...
Adding user ackbar to group adm
Done.
ackbar@desk:~$
```

After you add yourself to the group, you'll need to logout and log back in, or in some cases, reboot your machine. After you log in again, verify that you have been added to the group using the command `groups`. If you are still not in the group, reboot and check again.



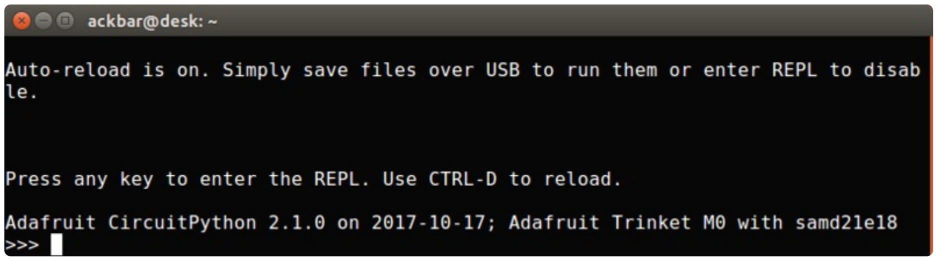
```
ackbar@desk:~$ groups
ackbar adm sudo
ackbar@desk:~$
```

And now you should be able to run `screen` without using `sudo`.



```
ackbar@desk:~$ groups
ackbar adm sudo
ackbar@desk:~$ screen /dev/ttyACM0 115200
```

And you're in:



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


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

Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Trinket M0 with samd21e18
>>> 
```

The examples above use `screen`, but you can also use other programs, such as `putty` or `picocom`, if you prefer.

Advanced Serial Console on Windows

Windows 7 Driver

If you're using Windows 7, use the link below to download the driver package. You will not need to install drivers on Mac, Linux or Windows 10.

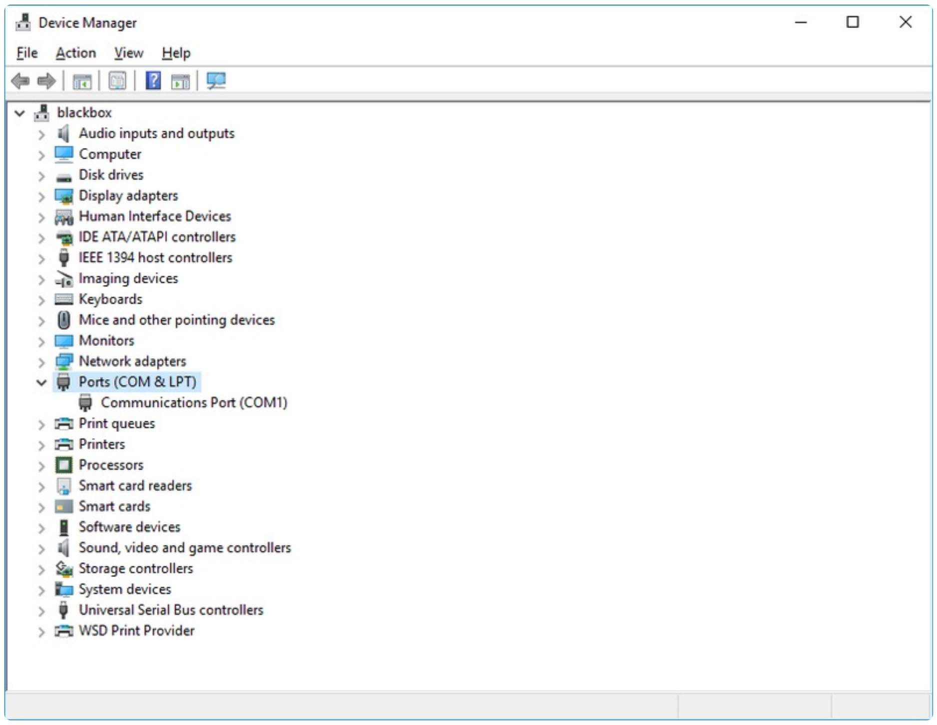
https://adafru.it/AB0

https://adafru.it/AB0

What's the COM?

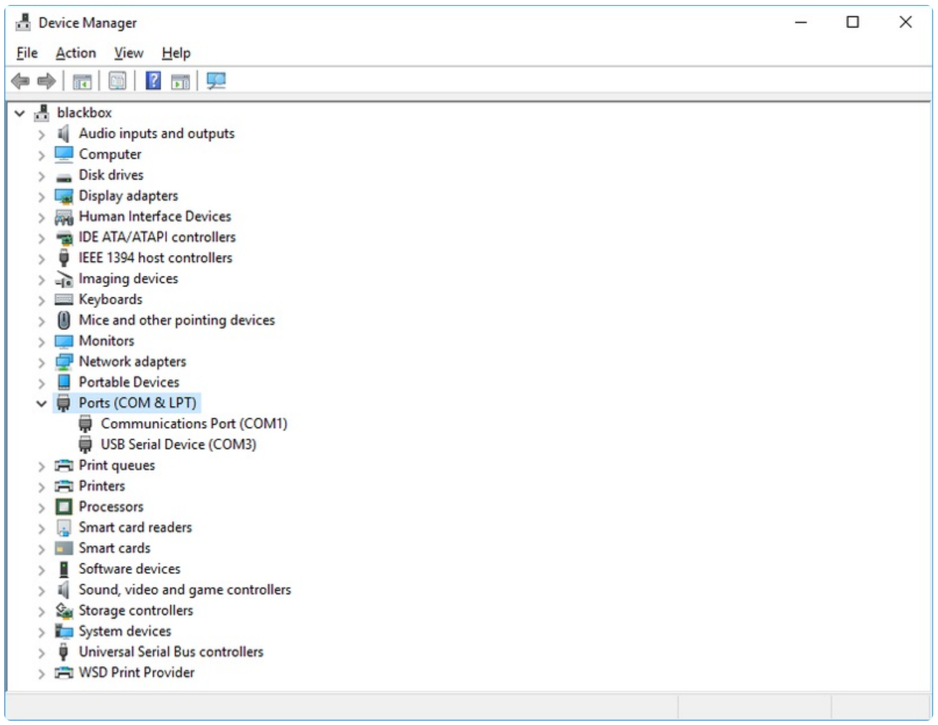
First, you'll want to find out which serial port your board is using. When you plug your board in to USB on your computer, it connects to a serial port. The port is like a door through which your board can communicate with your computer using USB.

We'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.



blackbox
- Audio inputs and outputs
- Computer
- Disk drives
- Display adapters
- Human Interface Devices
- IDE ATA/ATAPI controllers
- IEEE 1394 host controllers
- Imaging devices
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- Ports (COM & LPT)
  - Communications Port (COM1)
- Print queues
- Printers
- Processors
- Smart card readers
- Smart cards
- Software devices
- Sound, video and game controllers
- Storage controllers
- System devices
- Universal Serial Bus controllers
- WSD Print Provider

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.



blackbox
- Audio inputs and outputs
- Computer
- Disk drives
- Display adapters
- Human Interface Devices
- IDE ATA/ATAPI controllers
- IEEE 1394 host controllers
- Imaging devices
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- Portable Devices
- Ports (COM & LPT)
  - Communications Port (COM1)
  - USB Serial Device (COM3)
- Print queues
- Printers
- Processors
- Smart card readers
- Smart cards
- Software devices
- Sound, video and game controllers
- Storage controllers
- System devices
- Universal Serial Bus controllers
- WSD Print Provider

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

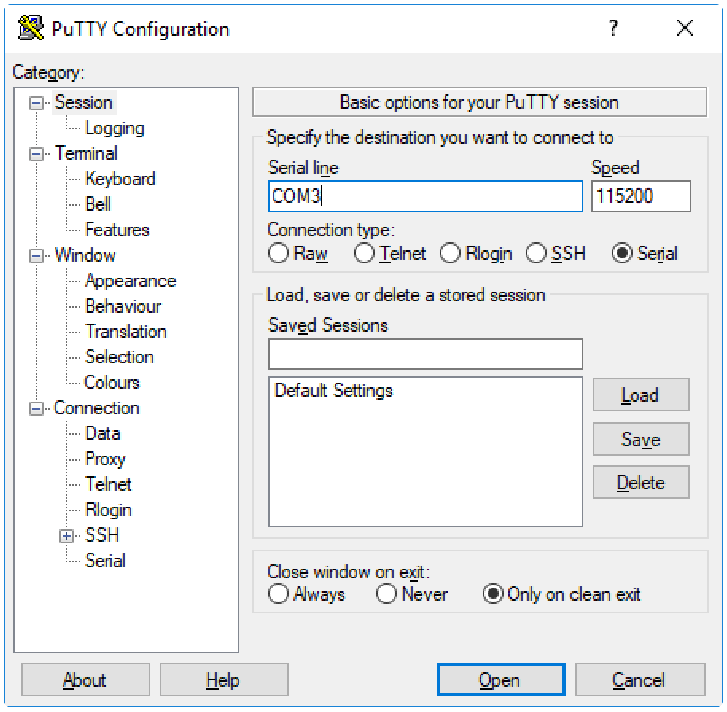
If you're using Windows, you'll need to download a terminal program. We're going to use PuTTY.

The first thing to do is download the latest version of PuTTY (https://adafru.it/Bf1). 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.

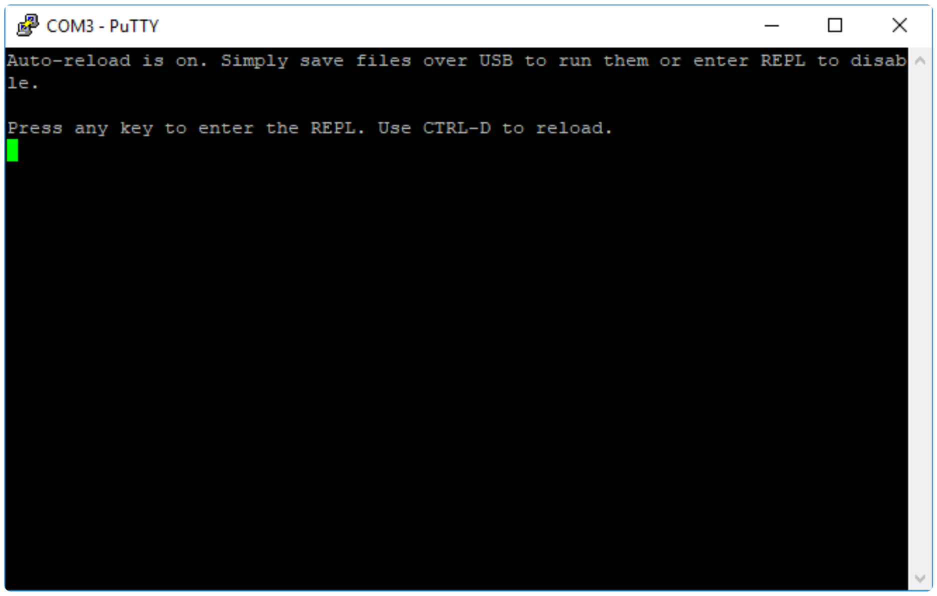
- Under **Connection type:** choose the button next to **Serial**.
- In the box under **Serial line**, enter the serial port you found that your board is using.
- In the box under **Speed**, enter 115200. This called the baud rate, which is the speed in bits per second that data is sent over the serial 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.



Specify the destination you want to connect to
Serial line: COM3
Speed: 115200
Connection type: Raw Telnet Rlogin SSH [✓ Serial]
Close window on exit: Always Never [✓ Only on clean exit]

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.

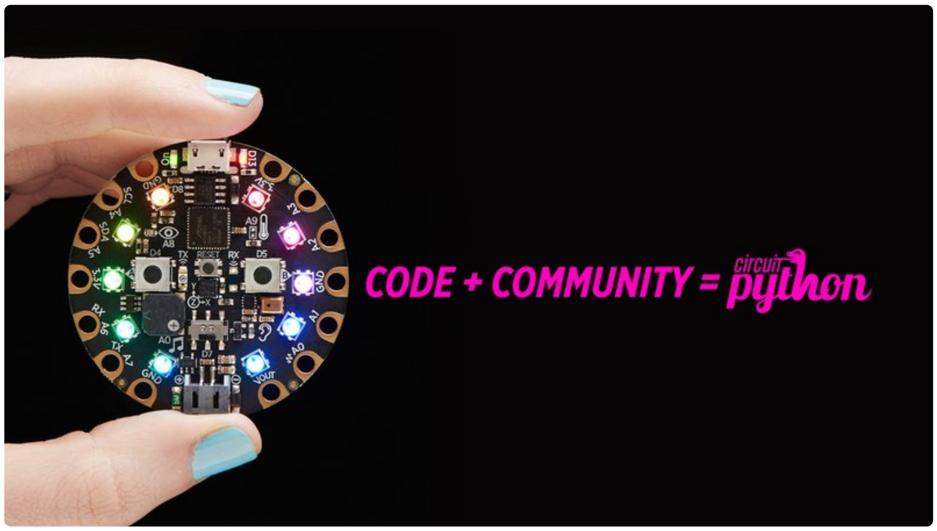


```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.

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

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!

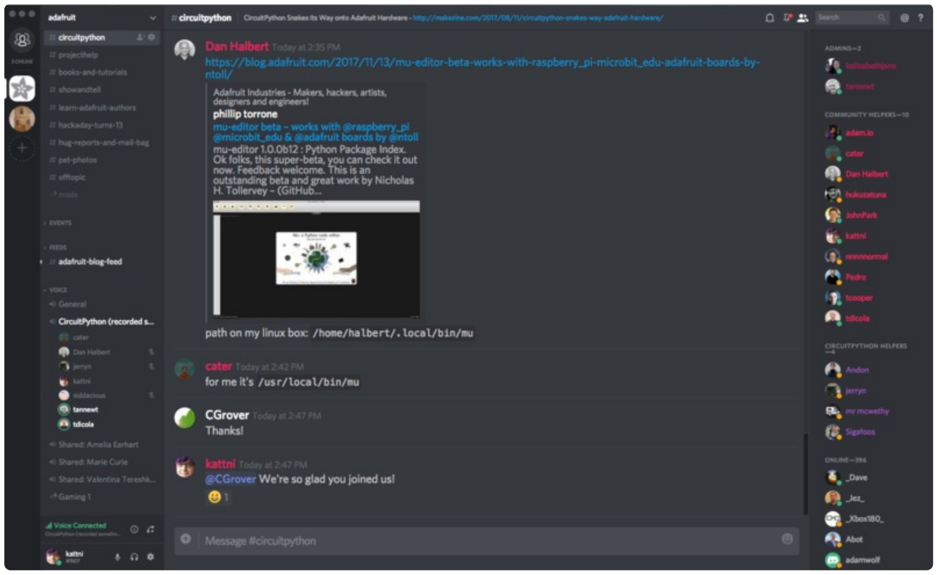
Welcome to the Community!



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. It doesn't matter whether this is your first microcontroller board or you're a computer engineer, you have something important to offer the Adafruit CircuitPython community. We're going to highlight some of the many ways you can be a part of it!

Adafruit Discord


#circuitpython CircuitPython Snakes Its Way onto Adafruit Hardware - http://makezine.com/2017/08/11/circuitpython-snakes-way-adafruit-hardware/

Dan Halbert Today at 2:35 PM
https://blog.adafruit.com/2017/11/13/mu-editor-beta-works-with-raspberry_pi-microbit_edu-adafruit-boards-by-ntoll/

Adafruit Industries - Makers, hackers, artists, designers and engineers!
phillip torrone
mu-editor beta - works with @raspberry_pi @microbit_edu & @adafruit boards by @ntoll
mu-editor 1.0.0b12 : Python Package Index. Ok folks, this super-beta, you can check it out now. Feedback welcome. This is an outstanding beta and great work by Nicholas H. Tollervey - (GitHub...

path on my linux box: `/home/halbert/.local/bin/mu`

cater Today at 2:42 PM
for me it's `/usr/local/bin/mu`

CGrover Today at 2:47 PM
Thanks!

kattni Today at 2:47 PM
@CGrover We're so glad you joined us!

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 #showandtell 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.

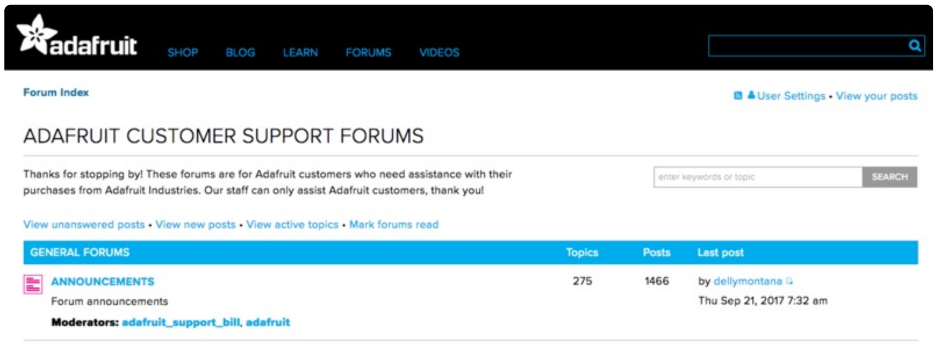
The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. We'd love to hear what you have to say! The #circuitpython 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. We're looking forward to meeting you!

Adafruit Forums



ADAFRUIT CUSTOMER SUPPORT FORUMS

Thanks for stopping by! These forums are for Adafruit customers who need assistance with their purchases from Adafruit Industries. Our staff can only assist Adafruit customers, thank you!

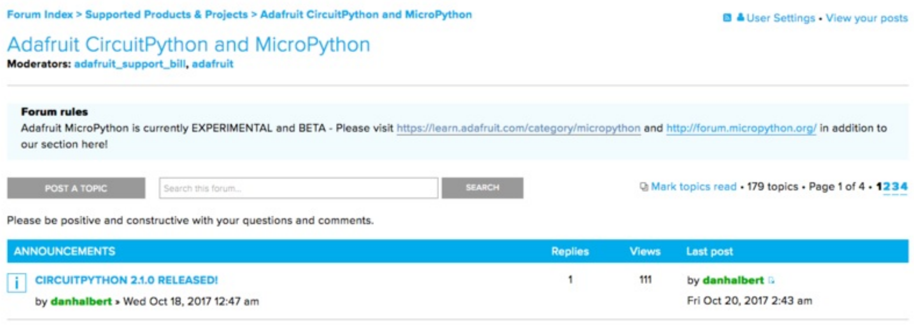
View unanswered posts • View new posts • View active topics • Mark forums read

GENERAL FORUMS	Topics	Posts	Last post
ANNOUNCEMENTS Forum announcements **Moderators:** adafruit_support_bill, adafruit	275	1466	by dellymontana Thu Sep 21, 2017 7:32 am

The Adafruit Forums (https://adafru.it/jlf) 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 and MicroPython (https://adafru.it/xXA) category under "Supported Products & Projects" is the best place to post your CircuitPython questions.


Forum Index > Supported Products & Projects > Adafruit CircuitPython and MicroPython

User Settings • View your posts

Adafruit CircuitPython and MicroPython

Moderators: adafruit_support_bill, adafruit

Forum rules

Adafruit MicroPython is currently EXPERIMENTAL and BETA - Please visit https://learn.adafruit.com/category/micropython and http://forum.micropython.org/ in addition to our section here!

Mark topics read • 179 topics • Page 1 of 4 • 1 2 3 4

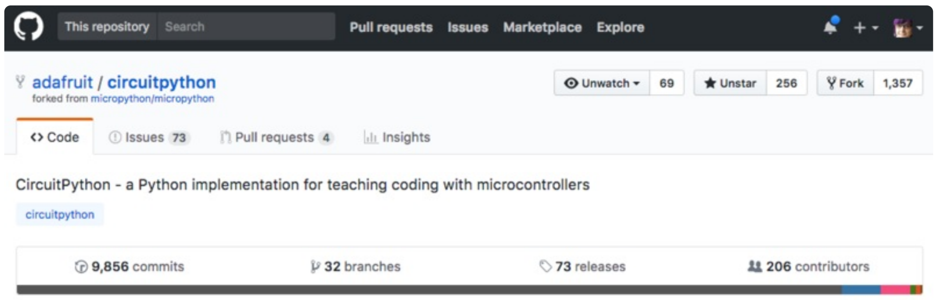
Please be positive and constructive with your questions and comments.

ANNOUNCEMENTS	Replies	Views	Last post
CIRCUITPYTHON 2.1.0 RELEASED! by danhalbert » Wed Oct 18, 2017 12:47 am	1	111	by danhalbert Fri Oct 20, 2017 2:43 am

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!

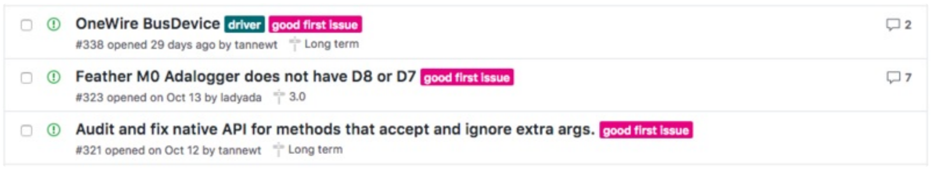
Adafruit Github



CircuitPython - a Python implementation for teaching coding with microcontrollers

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 building CircuitPython. GitHub is the best source of ways to contribute to CircuitPython (https://adafru.it/tB7) itself. If you need an account, visit https://github.com/ (https://adafru.it/d6C)and sign up.

If you're new to GitHub or programming in general, there are great opportunities for you. Head over to adafruit/circuitpython (https://adafru.it/tB7) on GitHub, click on "Issues (https://adafru.it/Bee)", and you'll find a list that includes issues labeled "good first issue (https://adafru.it/Bef)". These are things we've 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.


OneWire BusDevice driver good first issue
#338 opened 29 days ago by tannewt Long term
2

Feather M0 Adalogger does not have D8 or D7 good first issue
#323 opened on Oct 13 by ladyada 3.0
7

Audit and fix native API for methods that accept and ignore extra args. good first issue
#321 opened on Oct 12 by tannewt Long term

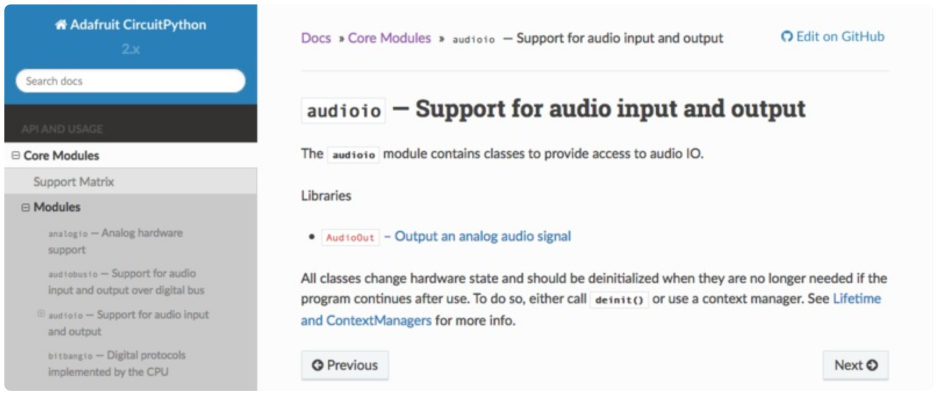
Already experienced and looking for a challenge? Checkout the rest of the issues list and you'll find plenty of ways to contribute. You'll find everything from new driver requests to core module updates. There's plenty of opportunities for everyone at any level!

When working with CircuitPython, you may find problems. If you find a bug, that's great! We love bugs! Posting a detailed issue to GitHub is an invaluable way to contribute to improving CircuitPython. 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 current and beta releases is a very important part of contributing CircuitPython. We can't possibly find all the problems ourselves! We 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!

ReadTheDocs


Adafruit CircuitPython
2.x
Search docs
API AND USAGE
Core Modules
Support Matrix
Modules
analogio – Analog hardware support
audiobusio – Support for audio input and output over digital bus
audioio – Support for audio input and output
bitbangio – Digital protocols implemented by the CPU
Docs » Core Modules » audioio – Support for audio input and output
Edit on GitHub
Previous
Next

audioio – Support for audio input and output

The `audioio` module contains classes to provide access to audio IO.

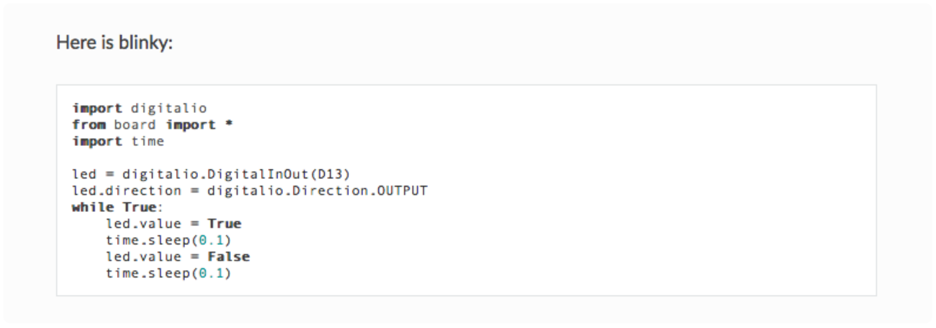
Libraries

- `AudioOut` – Output an analog audio signal

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See Lifetime and ContextManagers for more info.

ReadTheDocs (https://adafru.it/Beg) is a an excellent resource for a more in depth look at CircuitPython. This is where you'll find things like API documentation and details about core modules. There is also a Design Guide that includes contribution guidelines for CircuitPython.

RTD gives you access to a low level look at CircuitPython. There are details about each of the core modules (https://adafru.it/Beh). Each module lists the available libraries. Each module library page lists the available parameters and an explanation for each. In many cases, you'll find quick code examples to help you understand how the modules and parameters work, however it won't have detailed explanations like the Learn Guides. If you want help understanding what's going on behind the scenes in any CircuitPython code you're writing, ReadTheDocs is there to help!



Here is blinky:

```
import digitalio
from board import *
import time

led = digitalio.DigitalInOut(D13)
led.direction = digitalio.Direction.OUTPUT
while True:
    led.value = True
    time.sleep(0.1)
    led.value = False
    time.sleep(0.1)
```

Frequently Asked Questions

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

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.

I have to continue using an older version of CircuitPython; where can I find compatible libraries?

We are no longer building or supporting library bundles for older versions of CircuitPython. We highly encourage you to update CircuitPython to the latest version (https://adafru.it/Em8) and use the current version of the libraries (https://adafru.it/ENC). However, if for some reason you cannot update, here are points to the last available library bundles for previous versions:

- 2.x (https://adafru.it/FJA)
- 3.x (https://adafru.it/FJB)
- 4.x (https://adafru.it/QDL)
- 5.x (https://adafru.it/QDJ)

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!

https://learn.adafruit.com/welcome-to-circuitpython/circuitpython-for-esp8266 (https://adafru.it/CiG)

We do not support ESP32 because it does not have native USB. We do support ESP32-S2, which does.

How do I connect to the Internet with CircuitPython?

If you'd like to add WiFi support, check out our guide on ESP32/ESP8266 as a co-processor. (https://adafru.it/Dwa)

Is there asyncio support in CircuitPython?

We do not have asyncio support in CircuitPython at this time. However, `async` and `await` are turned on in many builds, and we are looking at how to use event loops and other constructs effectively and easily.

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! (https://adafru.it/Den)

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 (REPL).

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](https://adafru.it/uap) (https://adafru.it/uap) 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 [https://adafruit-circuit-python.s3.amazonaws.com/index.html?prefix=bin/mpy-cross/](https://adafru.it/QDK) (https://adafru.it/QDK). 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?

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

Will give you the number of bytes available for use.

Does CircuitPython support interrupts?

No. CircuitPython does not currently support interrupts. We do not have an estimated time for when they will be included.

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.

Commonly Used Acronyms

CP or CPy = CircuitPython (https://adafru.it/cpy-welcome)
CPC = Circuit Playground Classic (https://adafru.it/ncE)
CPX = Circuit Playground Express (https://adafru.it/wpF)

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 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. (https://adafru.it/Em8).**

You need to download the CircuitPython Library Bundle that matches your version of CircuitPython. **Please update CircuitPython and then download the latest bundle (https://adafru.it/ENC).**

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 5.x, 4.x, 3.x or 2.x, where can I find compatible libraries?

We are no longer building or supporting the CircuitPython 2.x, 3.x, 4.x or 5.x library bundles. We highly encourage you to update CircuitPython to the latest version (https://adafru.it/Em8) and use the current version of the libraries (https://adafru.it/ENC). However, if for some reason you cannot update, you can find the last available 2.x build here (https://adafru.it/FJA), the last available 3.x build here (https://adafru.it/FJB), the last available 4.x build here (https://adafru.it/QDL), and the last available 5.x build here (https://adafru.it/QDJ).

CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present

You may have a different board.

Only Adafruit Express boards and the Trinket M0 and Gemma M0 boards ship with the [UF2 bootloader](https://adafru.it/zbX) (https://adafru.it/zbX)installed. Feather M0 Basic, Feather M0 Adalogger, and similar boards use a regular Arduino-compatible bootloader, which does not show a `boardnameBOOT` drive.

MakeCode

If you are running a [MakeCode](https://adafru.it/zbY) (https://adafru.it/zbY) 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](https://adafru.it/sTc) (https://adafru.it/sTc) for how to fix the problem.

Windows 10

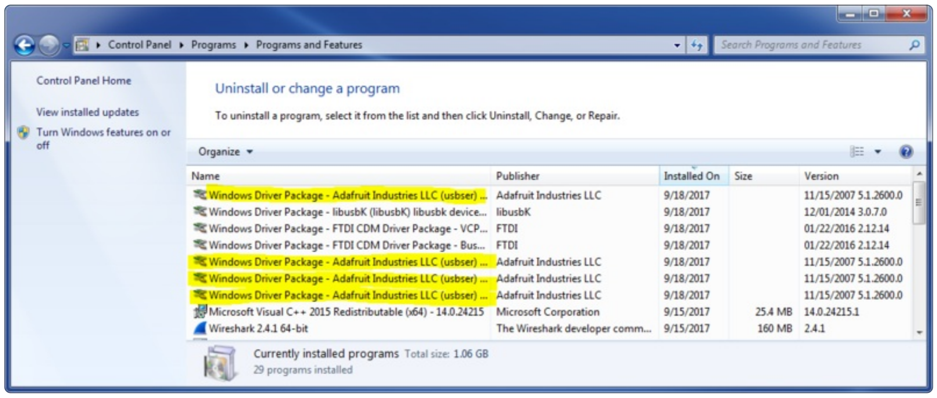
Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 with the driver package installed? You don't need to install this package on Windows 10 for most Adafruit boards. The old version (v1.5) can interfere with recognizing your device. Go to **Settings** -> **Apps** and uninstall all the "Adafruit" driver programs.

Windows 7 or 8.1

Version 2.5.0.0 or later of the Adafruit Windows Drivers will fix the missing `boardnameBOOT` drive problem on Windows 7 and 8.1. To resolve this, first uninstall the old versions of the drivers:

- Unplug any boards. In **Uninstall or Change a Program (Control Panel->Programs->Uninstall a program)**, uninstall everything named "Windows Driver Package - Adafruit Industries LLC ...".

We [recommend](https://adafru.it/Amd) (https://adafru.it/Amd) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you: see the link.


Control Panel ▸ Programs ▸ Programs and Features
Search Programs and Features
Control Panel Home
View installed updates
Turn Windows features on or off
Organize

Uninstall or change a program

To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.

Name	Publisher	Installed On	Size	Version
Windows Driver Package - Adafruit Industries LLC (usbser) ...	Adafruit Industries LLC	9/18/2017		11/15/2007 5.1.2600.0
Windows Driver Package - libusbK (libusbK) libusbk device...	libusbK	9/18/2017		12/01/2014 3.0.7.0
Windows Driver Package - FTDI CDM Driver Package - VCP...	FTDI	9/18/2017		01/22/2016 2.12.14
Windows Driver Package - FTDI CDM Driver Package - Bus...	FTDI	9/18/2017		01/22/2016 2.12.14
Windows Driver Package - Adafruit Industries LLC (usbser) ...	Adafruit Industries LLC	9/18/2017		11/15/2007 5.1.2600.0
Windows Driver Package - Adafruit Industries LLC (usbser) ...	Adafruit Industries LLC	9/18/2017		11/15/2007 5.1.2600.0
Windows Driver Package - Adafruit Industries LLC (usbser) ...	Adafruit Industries LLC	9/18/2017		11/15/2007 5.1.2600.0
Microsoft Visual C++ 2015 Redistributable (x64) - 14.0.24215	Microsoft Corporation	9/15/2017	25.4 MB	14.0.24215.1
Wireshark 2.4.1 64-bit	The Wireshark developer comm...	9/15/2017	160 MB	2.4.1

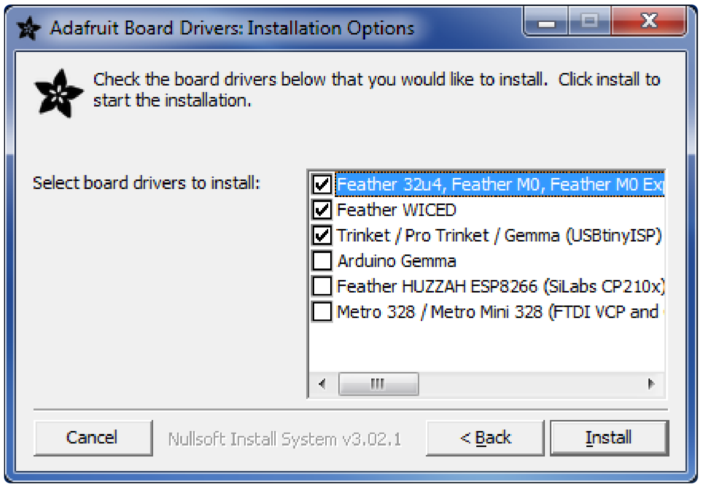
Currently installed programs Total size: 1.06 GB
29 programs installed

- Now install the new 2.5.0.0 (or higher) Adafruit Windows Drivers Package:

https://adafru.it/AB0

https://adafru.it/AB0

- When running the installer, you'll be shown a list of drivers to choose from. You can check and uncheck the boxes to choose which drivers to install.


Adafruit Board Drivers: Installation Options

Check the board drivers below that you would like to install. Click install to start the installation.

Select board drivers to install:

- [✓] Feather 32u4, Feather M0, Feather M0 Ex
- [✓] Feather WICED
- [✓] Trinket / Pro Trinket / Gemma (USBtinyISP)
- [ ] Arduino Gemma
- [ ] Feather HUZZAH ESP8266 (SiLabs CP210x)
- [ ] Metro 328 / Metro Mini 328 (FTDI VCP and

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 *boardname*BOOT drive.

Let us know in the Adafruit support forums (https://adafru.it/jIf) 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 we know of 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**: We have seen 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

Kaspersky anti-virus can block the appearance of the `CIRCUITPY` drive. We haven't yet figured out a settings change 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.

Windows 7 and 8.1 Problems

Windows 7 and 8.1 can become confused about USB device installations. We recommend (https://adafru.it/Amd) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you: see the link. If not, try cleaning up your USB devices with your board unplugged. Use Uwe Sieber's Device Cleanup Tool (https://adafru.it/RWd), which you must run as Administrator.

Serial Console in Mu Not Displaying Anything

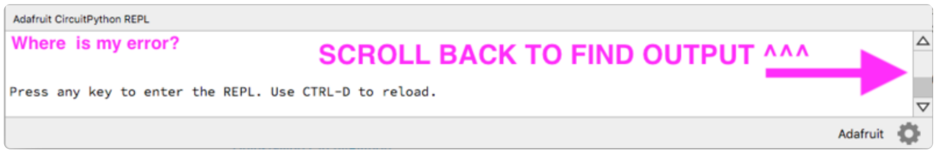
There are times when the serial console will accurately not display anything, such as, when no code is currently running, or when code with no serial output is already running before you open the console. However, if you find yourself in a situation where you feel it should be displaying something like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



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

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.

CircuitPython RGB Status Light

Nearly all Adafruit 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. They do NOT indicate any status while running CircuitPython.

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.

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 2.x and 3.x, as well as between 1.x and 2.x.

So, for instance, if you upgraded to CircuitPython 3.x from 2.x you'll need to download a newer version of the library that triggered the error on `import`. They are all available in the Adafruit bundle (https://adafru.it/y8E).

Make sure to download a version with 2.0.0 or higher in the filename if you're using CircuitPython version 2.2.4, and the version with 3.0.0 or higher in the filename if you're using CircuitPython version 3.0.

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.

First check - have you used Arduino to program your board? If so, CircuitPython is no longer able to provide the USB services. Reset the board so you get a *`boardnameBOOT`* drive rather than a `CIRCUITPY` drive, copy the latest version of CircuitPython (`.uf2`) back to the board, then Reset. This may restore `CIRCUITPY` functionality.

If still broken - 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.

In this situation, 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.

Easiest Way: Use storage.erase_filesystem()

Starting with version 2.3.0, CircuitPython includes a built-in function to erase and reformat the filesystem. If you have an older version of CircuitPython on your board, you can update to the newest version (https://adafru.it/Amd) to do this.

1. Connect to the CircuitPython REPL (https://adafru.it/Bec) using Mu or a terminal program.
2. Type:

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

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

Old Way: For the Circuit Playground Express, Feather M0 Express, and Metro M0 Express:

If you can't get to the REPL, or you're running a version of CircuitPython before 2.3.0, and you don't want to upgrade, you can do this.

1. Download the correct erase file:

https://adafru.it/AdI

https://adafru.it/AdI

https://adafru.it/AdJ

https://adafru.it/AdJ

https://adafru.it/EVK

https://adafru.it/EVK

https://adafru.it/AdK

https://adafru.it/AdK

https://adafru.it/EoM

https://adafru.it/EoM

https://adafru.it/DjD

https://adafru.it/DjD

https://adafru.it/DBA

https://adafru.it/DBA

https://adafru.it/Eca

https://adafru.it/Eca

https://adafru.it/Gnc

https://adafru.it/Gnc

https://adafru.it/GAN

https://adafru.it/GAN

https://adafru.it/GAO

https://adafru.it/GAO

https://adafru.it/Jat

https://adafru.it/Jat

https://adafru.it/Q5B

https://adafru.it/Q5B

2. Double-click the reset button on the board to bring up the *boardname*BOOT drive.
3. Drag the erase .uf2 file to the *boardname*BOOT drive.
4. The onboard NeoPixel will turn yellow or blue, indicating the erase has started.
5. After approximately 15 seconds, the mainboard NeoPixel 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 *boardname*BOOT drive.
7. Drag the appropriate latest release of CircuitPython (https://adafru.it/Amd) .uf2 file to the *boardname*BOOT 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 (https://adafru.it/Amd). You'll also need to install your libraries and code!

Old Way: For Non-Express Boards with a UF2 bootloader (Gemma M0, Trinket M0):

If you can't get to the REPL, or you're running a version of CircuitPython before 2.3.0, and you don't want to upgrade, you can do this.

1. Download the erase file:

https://adafru.it/AdL

https://adafru.it/AdL

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 (https://adafru.it/Amd) `.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 (https://adafru.it/Amd) You'll also need to install your libraries and code!

Old Way: For non-Express Boards without a UF2 bootloader (Feather M0 Basic Proto, Feather Adalogger, Arduino Zero):

If you are running a version of CircuitPython before 2.3.0, and you don't want to upgrade, or you can't get to the REPL, you can do this.

Just follow these directions to reload CircuitPython using `bossac` (https://adafru.it/Bed), which will erase and re-create `CIRCUITPY`.

Running Out of File Space on Non-Express 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 couple ways to free up space.

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. Its ~12KiB or so.

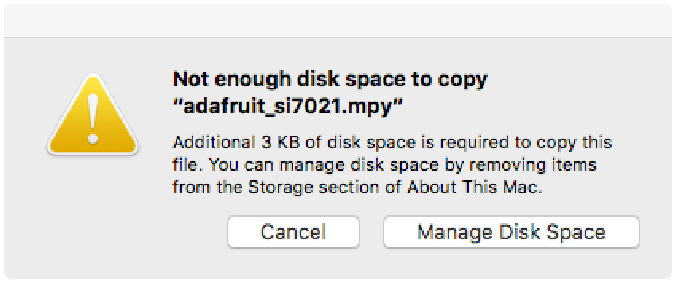
Delete something!

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.

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, we recommend that 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 we're counting bytes.

MacOS loves to add extra files.



Not enough disk space to copy "adafruit_si7021.mpy"

Additional 3 KB of disk space is required to copy this file. You can manage disk space by removing items from the Storage section of About This Mac.

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 (https://adafru.it/u1c) 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 `foo.mpy` file to the board use a command like:

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

(Replace `foo.mpy` with the name of the file you want to copy.) Or to copy a folder and all of its child files/folders use a command like:

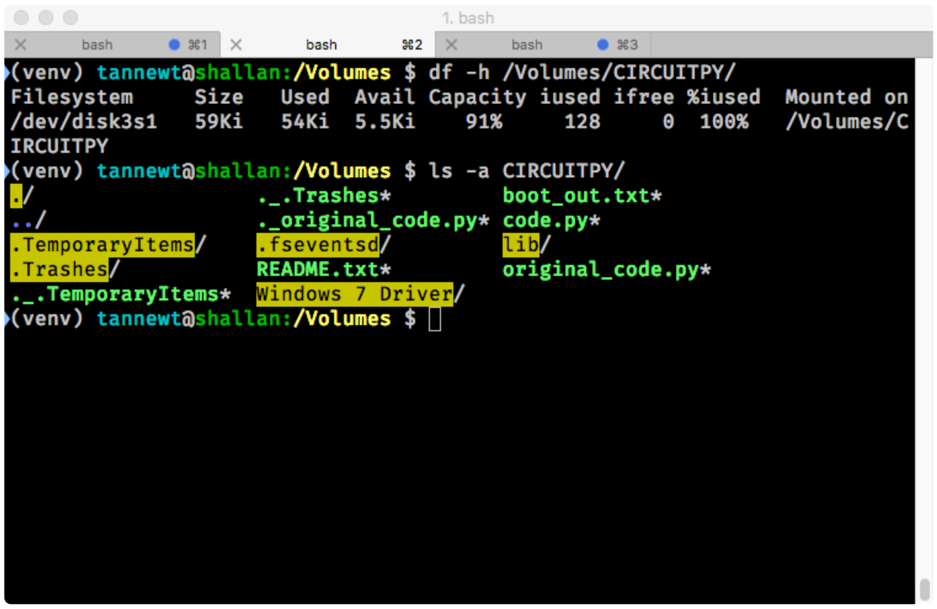
```
cp -rX folder_to_copy /Volumes/CIRCUITPY
```

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 foo.mpy /Volumes/CIRCUITPY/lib
# This is safer, and will complain if a lib folder does not exist.
cp -X foo.mpy /Volumes/CIRCUITPY/lib/
```

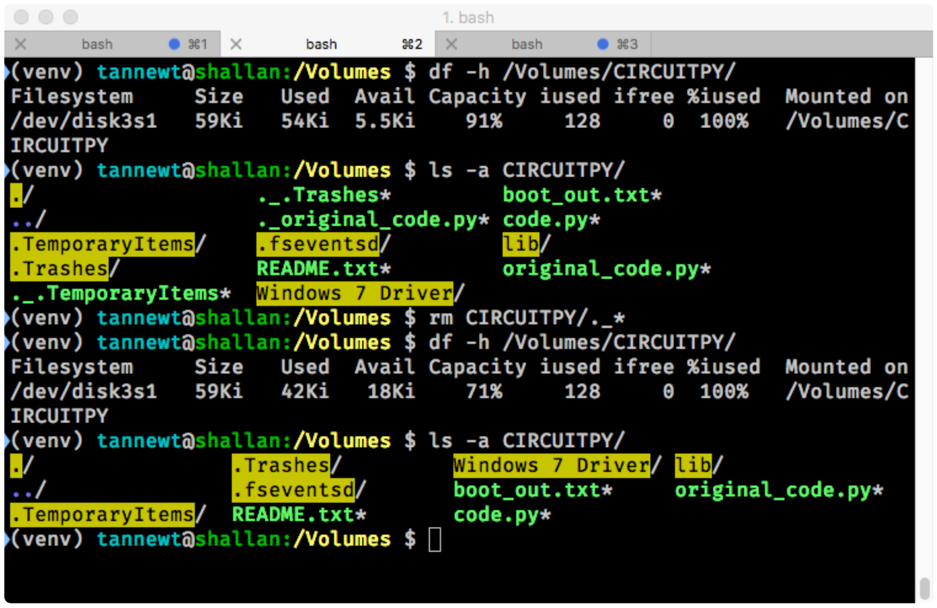
Other MacOS Space-Saving Tips

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 list the amount of space used on the `CIRCUITPY` drive with the **df** command:



```
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem     Size   Used  Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1   59Ki   54Ki  5.5Ki    91%     128     0  100%   /Volumes/C
IRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./                  ._.Trashes*          boot_out.txt*
../                 ._original_code.py* code.py*
.TemporaryItems/    .fseventsd/          lib/
.Trashes/           README.txt*          original_code.py*
._.TemporaryItems*  Windows 7 Driver/
(venv) tannewt@shallan:/Volumes $
```

Lets remove the `._` files first.



```
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem     Size   Used  Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1   59Ki   54Ki  5.5Ki    91%     128     0  100%   /Volumes/C
IRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./                  ._.Trashes*          boot_out.txt*
../                 ._original_code.py* code.py*
.TemporaryItems/    .fseventsd/          lib/
.Trashes/           README.txt*          original_code.py*
._.TemporaryItems*  Windows 7 Driver/
(venv) tannewt@shallan:/Volumes $ rm CIRCUITPY/._*
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem     Size   Used  Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1   59Ki   42Ki   18Ki    71%     128     0  100%   /Volumes/C
IRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./                .Trashes/          Windows 7 Driver/ lib/
../               .fseventsd/        boot_out.txt*     original_code.py*
.TemporaryItems/  README.txt*        code.py*
(venv) tannewt@shallan:/Volumes $
```

Whoa! We have 13Ki more than before! This space can now be used for libraries and code!

Device locked up or boot looping

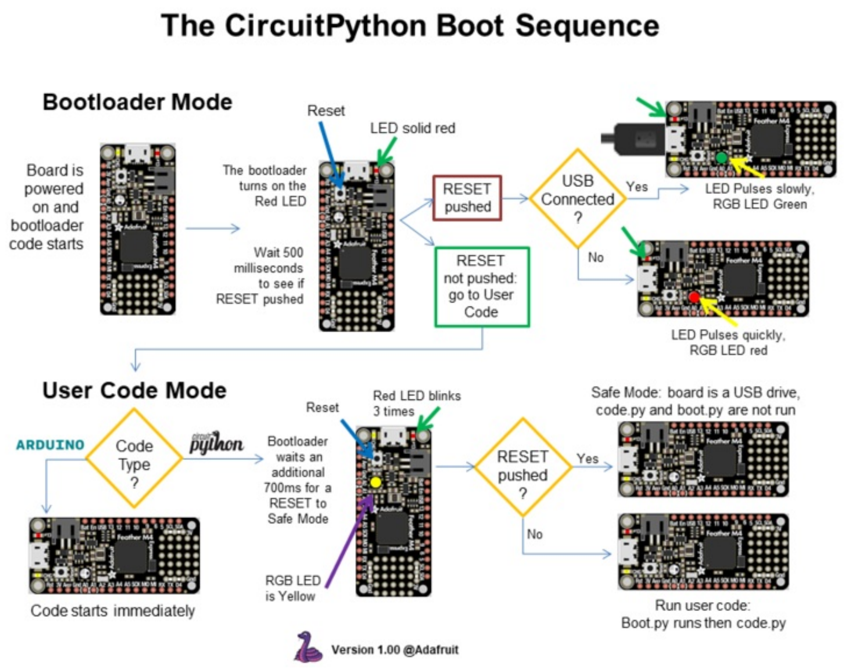
In rare cases, it may happen that something in your **code.py** or **boot.py** files causes the device to get locked up, or even go into a boot loop. These are not your everyday Python exceptions, typically it's the result of a deeper problem within CircuitPython. In this situation, it can be difficult 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 is yellow, press the reset button again.

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 following diagram for boot sequence details:


The CircuitPython Boot Sequence
Bootloader Mode
Board is powered on and bootloader code starts
Reset
LED solid red
The bootloader turns on the Red LED
Wait 500 milliseconds to see if RESET pushed
RESET pushed
RESET not pushed: go to User Code
USB Connected ?
Yes
No
LED Pulses slowly, RGB LED Green
LED Pulses quickly, RGB LED red
User Code Mode
ARDUINO
Code Type ?
python
Code starts immediately
Reset
Red LED blinks 3 times
Bootloader waits an additional 700ms for a RESET to Safe Mode
RGB LED is Yellow
RESET pushed ?
Yes
No
Safe Mode: board is a USB drive, code.py and boot.py are not run
Run user code: Boot.py runs then code.py
Version 1.00 @Adafruit

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/re-install CircuitPython *whenever you want!* Heck, you can change your mind every day!

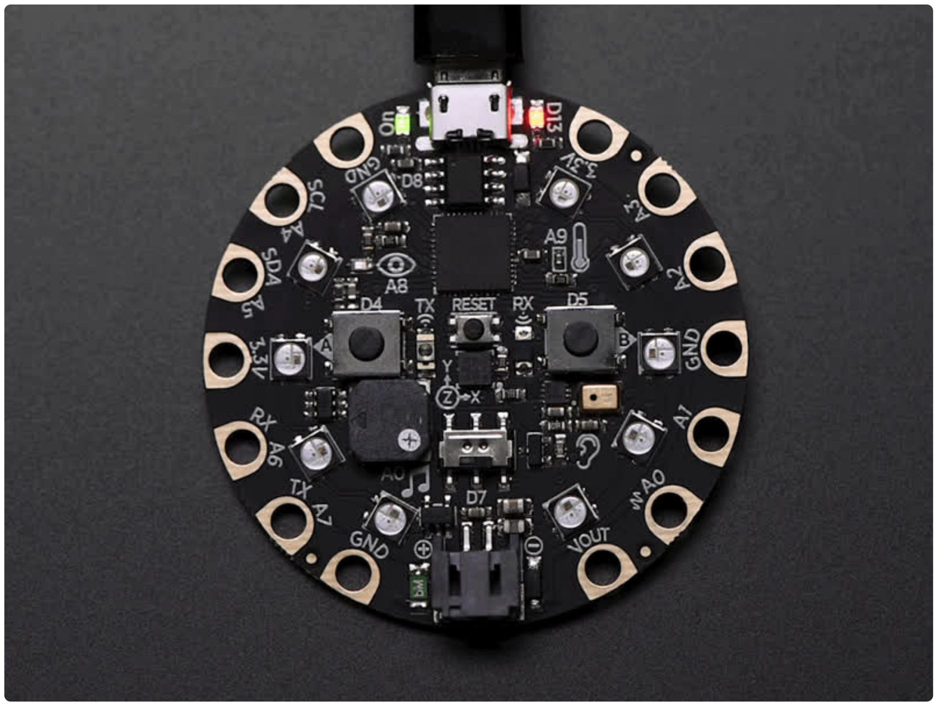
Backup Your Code

Before uninstalling CircuitPython, don't forget to make a backup of the code you have on the little disk drive. That means your **main.py** or **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.

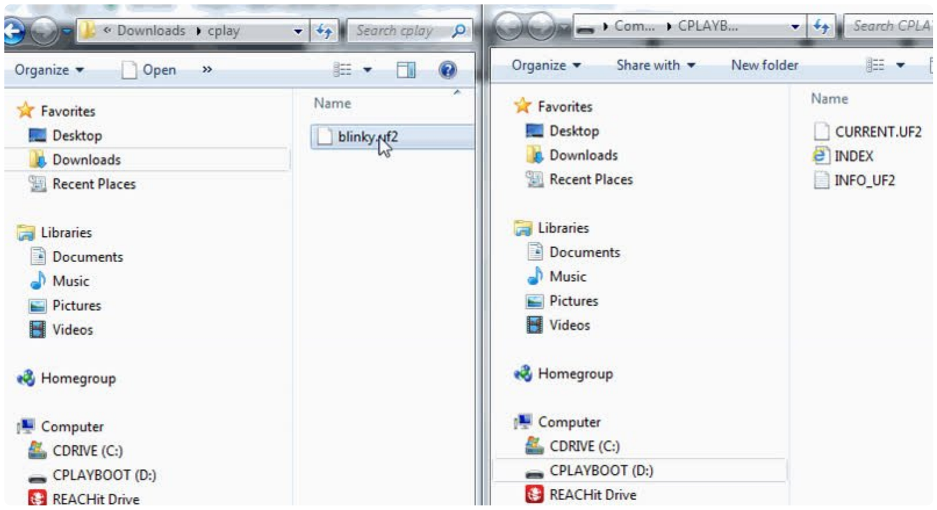
Moving Circuit Playground Express to MakeCode

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 (https://adafru.it/wpC) 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 **...BOOT** drive.



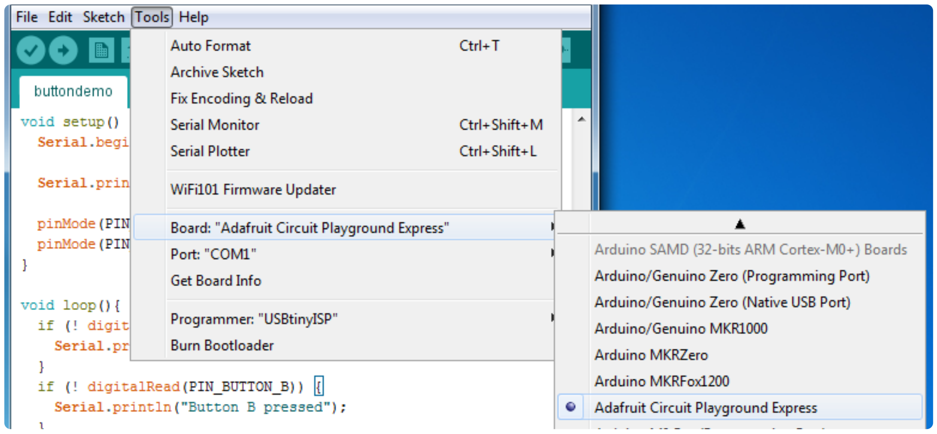
Your MakeCode is now running and CircuitPython has been removed. Going forward you only have to **single click** the reset button

Moving to Arduino

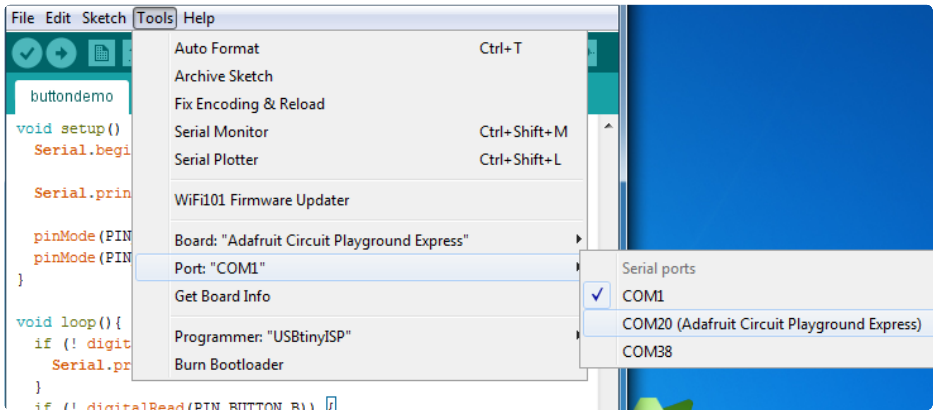
If you want to change your firmware to Arduino, it's also pretty easy.

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

Within Arduino IDE, select the matching board, say Circuit Playground Express


File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Adafruit Circuit Playground Express"
Port: "COM1"
Get Board Info
Programmer: "USBtinyISP"
Burn Bootloader
Arduino SAMD (32-bits ARM Cortex-M0+) Boards
Arduino/Genuino Zero (Programming Port)
Arduino/Genuino Zero (Native USB Port)
Arduino/Genuino MKR1000
Arduino MKRZero
Arduino MKRFox1200
Adafruit Circuit Playground Express
buttondemo

Select the correct matching Port:



Create a new simple Blink sketch example:

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

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

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

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

CircuitPython Essentials

CircuitPython Essentials (https://adafru.it/Bfr)

Clue Library Documentation

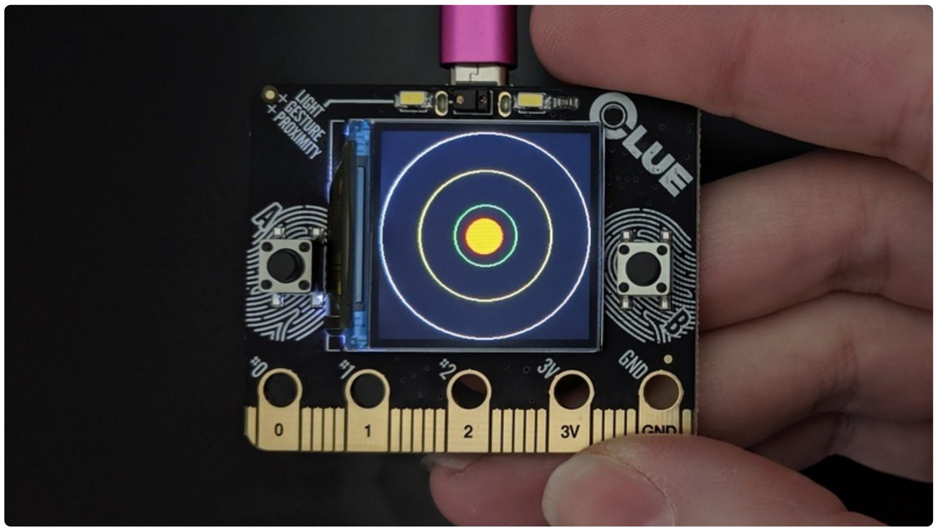
Clue Library Documentation (https://adafru.it/Ka-)

CLUE CircuitPython Demos

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

(For the high-level CLUE library, check out the docs here (https://adafru.it/Ka-).)

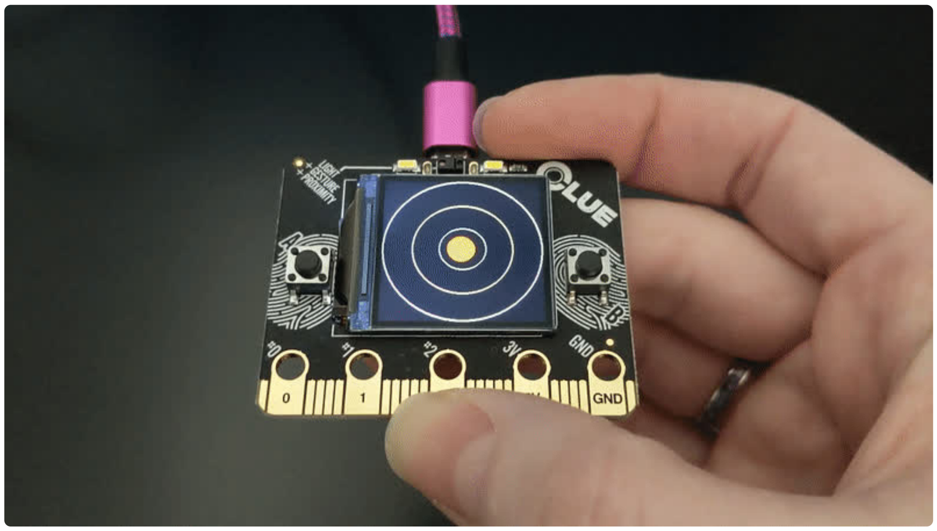
CLUE Spirit Level



The Adafruit CLUE comes with a built in accelerometer for measuring acceleration. For more information on how that works, check out Wikipedia (https://adafru.it/J2d).

Remember that you must have the necessary libraries installed. Verify that your **lib** folder matches the list found on the CLUE CircuitPython Libraries page (https://adafru.it/Jb9) before continuing.

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



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

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

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

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

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

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

Let's take a look at the code.

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

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

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

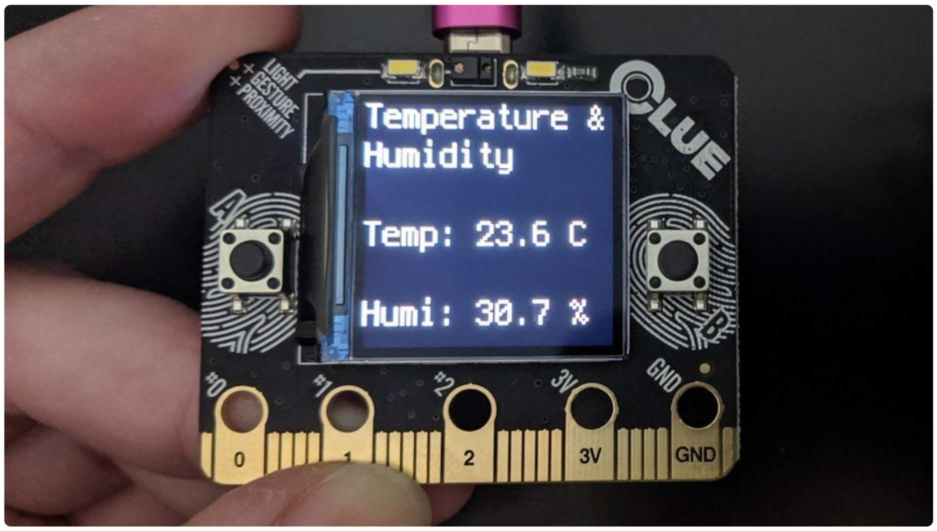
Then we create the `bubble_group` to hold the bubble. With the circles, as they are static, we could simply append them into a group. The bubble needs to move, so it must be in its own group that we will add to the same group as the circles once created. Next we create the level bubble. Instead of a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

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

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

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

CLUE Temperature and Humidity Monitor



The Adafruit CLUE has a temperature and humidity sensor that reads the temperature in degrees Celsius and the relative humidity in percent. To learn more about relative humidity, check Wikipedia (https://adafru.it/CxM).

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

Remember that you must have the necessary libraries installed. Verify that your **lib** folder matches the list found on the CLUE CircuitPython Libraries page (https://adafru.it/Jb9) before continuing.

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

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

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

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

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

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

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

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

while True:
    alarm = False

    temperature = clue.temperature
    humidity = clue.humidity

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

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

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

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

Let's take a look at the code.

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

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

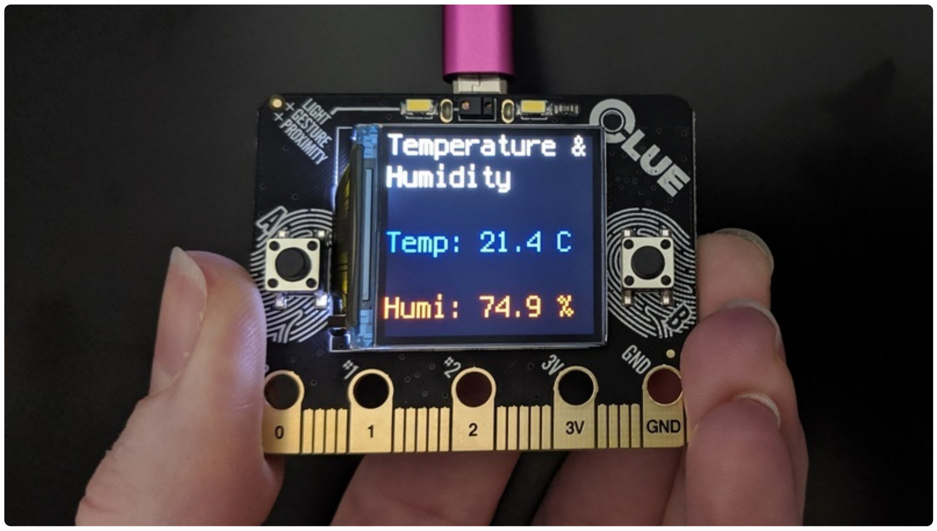
Then we add the first two lines of text which are the title.

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

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

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

That's what goes into making a temperature and humidity monitor using CircuitPython and CLUE!



Note: The temperature will read higher than ambient temperature because of the backlight on the CLUE display.

CLUE Height Calculator

The Adafruit CLUE has a barometric pressure sensor that uses the pressure readings to calculate altitude. For more information on how this works, check out Wikipedia (https://adafru.it/J2e).

With CircuitPython and a little math, it is possible to use the altitude reading to calculate the height of an object using your CLUE! Start with the CLUE at the bottom of the object, and press button A to set the initial reading, and then lift your CLUE up to the top of the object to get your reading.

Remember that you must have the necessary libraries installed. Verify that your **lib** folder matches the list found on the CLUE CircuitPython Libraries page (https://adafru.it/Jb9) before continuing.

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

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

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

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

initial_height = clue.altitude

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

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

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

Let's take a look at the code.

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

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

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

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

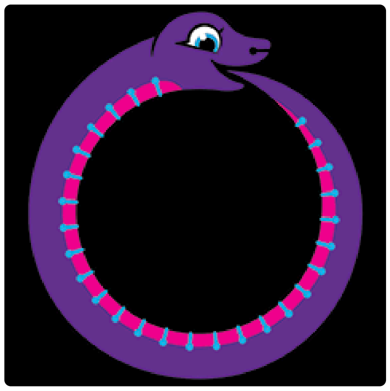
That's all there is to creating a height calculator with CircuitPython and CLUE!

CLUE Slideshow

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

Blinka Bitmaps

First, you'll need to load some compatible bitmap files onto your **CIRCUITPY** drive. We've included three compatible bitmaps below to get you started. For information on how to create your own compatible bitmaps, check out the Customization section of the Notifcation Icons page in this guide (https://adafru.it/Jfo).

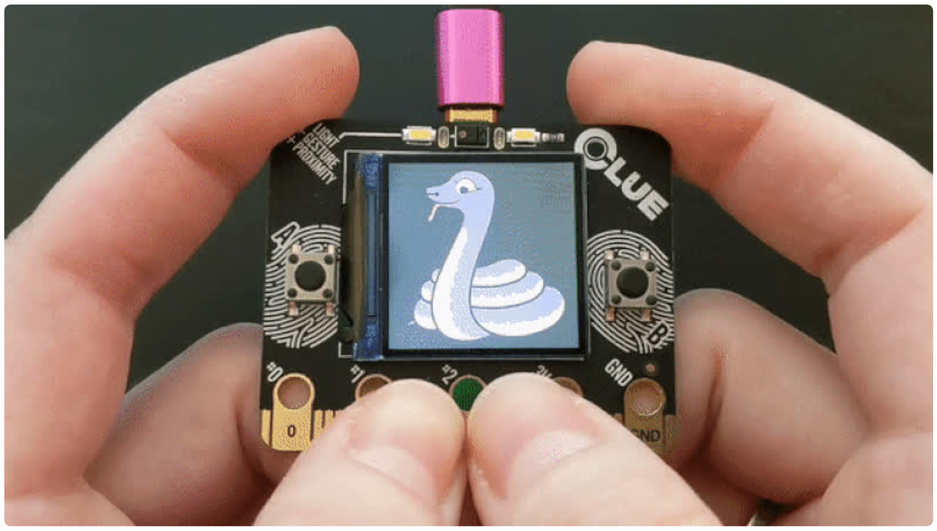


CircuitPython Libraries

Remember that you must have the necessary libraries installed. Verify that your **lib** folder matches the list found on the CLUE CircuitPython Libraries page (https://adafru.it/Jb9) before continuing.

CLUE Slideshow Example

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



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

Requires the Adafruit CircuitPython Slideshow library!"""

from adafruit_slideshow import SlideShow, PlayBackDirection
from adafruit_clue import clue

slideshow = SlideShow(clue.display, auto_advance=False)

while True:
    if clue.button_b:
        slideshow.direction = PlayBackDirection.FORWARD
        slideshow.advance()
    if clue.button_a:
        slideshow.direction = PlayBackDirection.BACKWARD
        slideshow.advance()
```

Let's take a look at the code.

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

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

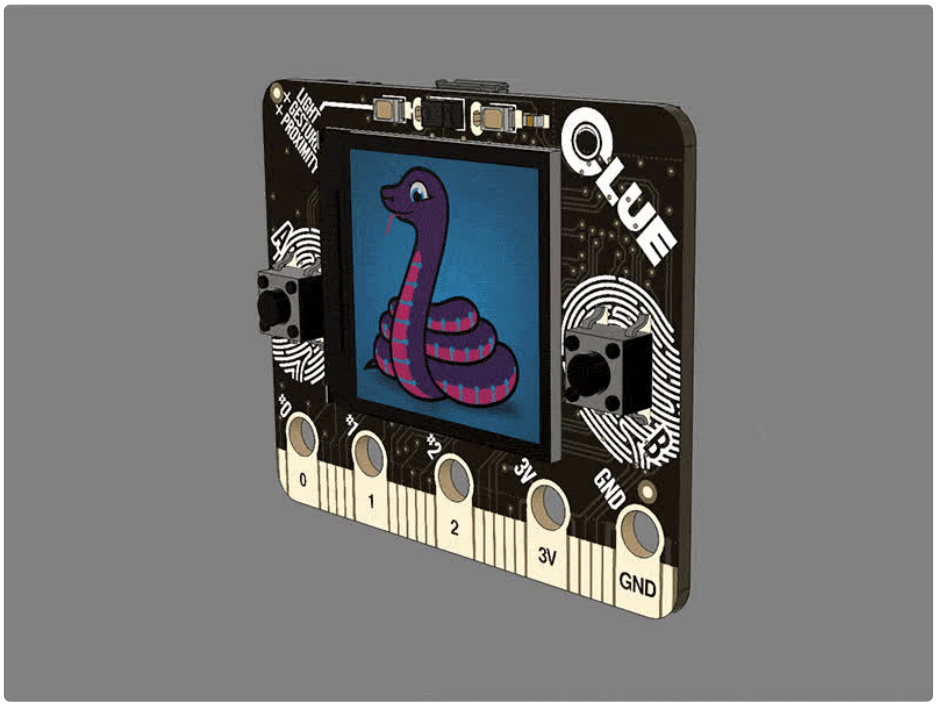
Inside the loop, we check for when each button is pressed. When button B is pressed, we set the playback direction to FORWARD and advance one image. When button A is pressed, we set the playback direction to BACKWARD, and advance one image.

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

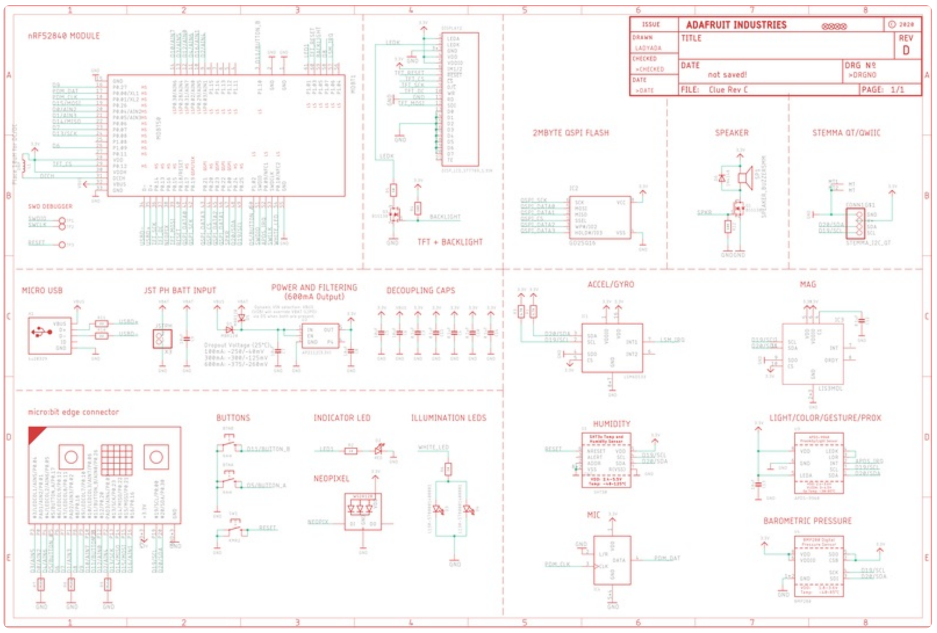
Downloads

Files:

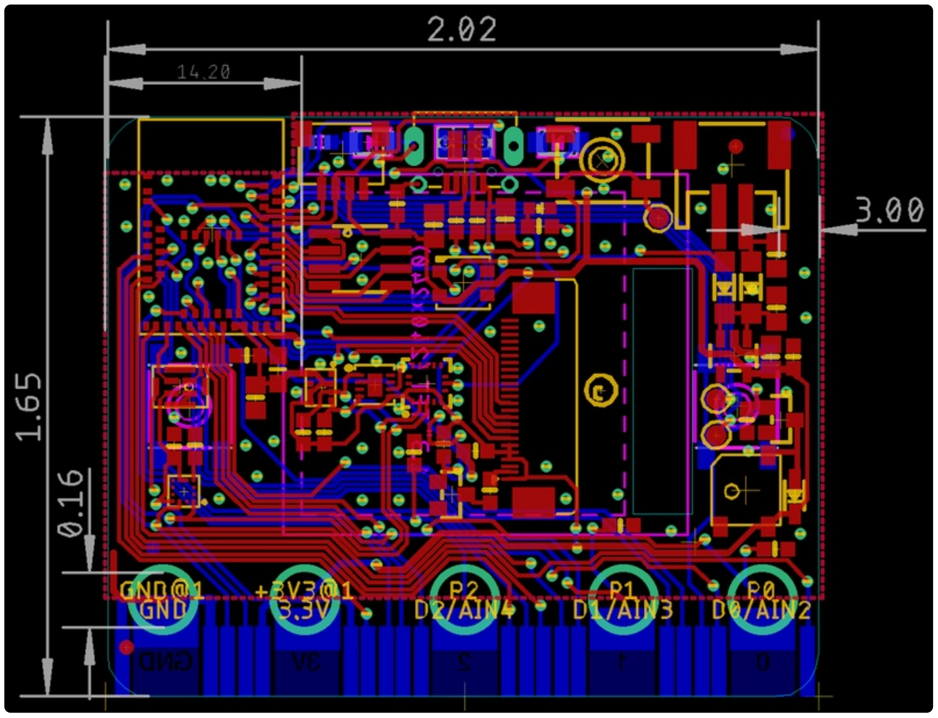
- General nRF52840 Product Specification (https://adafru.it/Dvt)
- CLUE nRF52840 module: Raytac MDBT50Q details (https://adafru.it/Dvu)
- PDM mic: MP34DT01-M datasheet (https://adafru.it/CiZ)
- Humidity sensor: SHT3x-DIS dataheet (https://adafru.it/k6a)
- Temperature and Barometric Pressure sensor: BMP280 datasheet (https://adafru.it/fIO)
- Proximity, Light, Gesture, Color sensor: APDS9960 datasheet (https://adafru.it/z0c)
- Gyroscope and Accelerometer: LSM6DS33 Datasheet (https://adafru.it/Iqf)
- Magnetometer: LIS3MDL Datasheet (https://adafru.it/IbR)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/ISc)
- EagleCAD files on GitHub (https://adafru.it/ISd)
- 3D Models on GitHub (https://adafru.it/ISe)



Schematic



Fab Print


2.02
14.20
3.00
1.65
0.16
GND@1
GND
+3V3@1
3.3V
P2
D2/AIN4
P1
D1/AIN3
P0
D0/AIN2