



Introducing Adafruit PyGamer

Created by Kattni Rembor



<https://learn.adafruit.com/adafruit-pygamer>

Last updated on 2023-04-08 10:47:06 PM EDT

Table of Contents

Overview	7
Which Board?	11
<ul style="list-style-type: none">• Similarities• Differences between PyGamer and PyBadge/PyBadge LC• Differences between PyBadge & PyBadge LC• Comparison Table	
Update the PyGamer Bootloader	14
<ul style="list-style-type: none">• Updating Your PyGamer Bootloader• Oh no, I updated MacOS already and I can't see the boot drive!	
Pinouts	17
<ul style="list-style-type: none">• Microcontroller and Flash• Power• Display	
Build the PyGamer Case	24
<ul style="list-style-type: none">• Prep• Paper Protection• Speaker• Battery• Button Caps• Case Layers• Spacers• Backing• Fasteners	
Load a MakeCode Game on PyGamer/PyBadge	33
<ul style="list-style-type: none">• Board Definition• Change Board screen• Bootloader Mode• Drag and Drop• Play!	
CircuitPython	37
<ul style="list-style-type: none">• Set up CircuitPython Quick Start!• Further Information	
Installing the Mu Editor	40
<ul style="list-style-type: none">• Download and Install Mu• Starting Up Mu• Using Mu	
Creating and Editing Code	43
<ul style="list-style-type: none">• Creating Code• Editing Code• Back to Editing Code...• Naming Your Program File	

Connecting to the Serial Console	48
<ul style="list-style-type: none">• Are you using Mu?• Serial Console Issues or Delays on Linux• Setting Permissions on Linux• Using Something Else?	
Interacting with the Serial Console	51
The REPL	54
<ul style="list-style-type: none">• Entering the REPL• Interacting with the REPL• Returning to the Serial Console	
CircuitPython Libraries	59
<ul style="list-style-type: none">• The Adafruit Learn Guide Project Bundle• The Adafruit CircuitPython Library Bundle• Downloading the Adafruit CircuitPython Library Bundle• The CircuitPython Community Library Bundle• Downloading the CircuitPython Community Library Bundle• Understanding the Bundle• Example Files• Copying Libraries to Your Board• Understanding Which Libraries to Install• Example: ImportError Due to Missing Library• Library Install on Non-Express Boards• Updating CircuitPython Libraries and Examples• CircUp CLI Tool	
CircuitPython Pins and Modules	70
<ul style="list-style-type: none">• CircuitPython Pins• import board• I2C, SPI, and UART• What Are All the Available Names?• Microcontroller Pin Names• CircuitPython Built-In Modules	
Advanced Serial Console on Windows	76
<ul style="list-style-type: none">• Windows 7 and 8.1• What's the COM?• Install Putty	
Advanced Serial Console on Mac	80
<ul style="list-style-type: none">• What's the Port?• Connect with screen	
Welcome to the Community!	82
<ul style="list-style-type: none">• Adafruit Discord• CircuitPython.org• Adafruit GitHub• Adafruit Forums• Read the Docs	
Frequently Asked Questions	91
<ul style="list-style-type: none">• Using Older Versions• Python Arithmetic	

- [Wireless Connectivity](#)
- [Asyncio and Interrupts](#)
- [Status RGB LED](#)
- [Memory Issues](#)
- [Unsupported Hardware](#)

Troubleshooting

96

- [Always Run the Latest Version of CircuitPython and Libraries](#)
- [I have to continue using CircuitPython 5.x or earlier. Where can I find compatible libraries?](#)
- [Bootloader \(boardnameBOOT\) Drive Not Present](#)
- [Windows Explorer Locks Up When Accessing boardnameBOOT Drive](#)
- [Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied](#)
- [CIRCUITPY Drive Does Not Appear or Disappears Quickly](#)
- [Device Errors or Problems on Windows](#)
- [Serial Console in Mu Not Displaying Anything](#)
- [code.py Restarts Constantly](#)
- [CircuitPython RGB Status Light](#)
- [CircuitPython 7.0.0 and Later](#)
- [CircuitPython 6.3.0 and earlier](#)
- [Serial console showing ValueError: Incompatible .mpy file](#)
- [CIRCUITPY Drive Issues](#)
- [Safe Mode](#)
- [To erase CIRCUITPY: storage.erase_filesystem\(\)](#)
- [Erase CIRCUITPY Without Access to the REPL](#)
- [For the specific boards listed below:](#)
- [For SAMD21 non-Express boards that have a UF2 bootloader:](#)
- [For SAMD21 non-Express boards that do not have a UF2 bootloader:](#)
- [Running Out of File Space on SAMD21 Non-Express Boards](#)
- [Delete something!](#)
- [Use tabs](#)
- [On MacOS?](#)
- [Prevent & Remove MacOS Hidden Files](#)
- [Copy Files on MacOS Without Creating Hidden Files](#)
- [Other MacOS Space-Saving Tips](#)
- [Device Locked Up or Boot Looping](#)

"Uninstalling" CircuitPython

114

- [Backup Your Code](#)
- [Moving Circuit Playground Express to MakeCode](#)
- [Moving to Arduino](#)

Arduino IDE Setup

117

Using with Arduino IDE

119

- [Install SAMD Support](#)
- [Install Adafruit SAMD](#)
- [Install Drivers \(Windows 7 & 8 Only\)](#)
- [Blink](#)
- [Successful Upload](#)
- [Compilation Issues](#)
- [Manually bootloading](#)
- [Ubuntu & Linux Issue Fix](#)

Arcada Libraries

127

- [Install Libraries](#)
- [Adafruit Arcada](#)

- If you aren't running Arduino IDE 1.8.10 or later, you'll need to install all of the following!
- [Adafruit NeoPixel](#)
- [Adafruit FreeTouch](#)
- [Adafruit Touchscreen](#)
- [Adafruit SPIFlash](#)
- [Adafruit Zero DMA](#)
- [Adafruit GFX](#)
- [Adafruit ST7735](#)
- [Adafruit ILI9341](#)
- [Adafruit LIS3DH](#)
- [Adafruit Sensor](#)
- [Adafruit ImageReader](#)
- [ArduinoJson](#)
- [Adafruit ZeroTimer](#)
- [Adafruit TinyUSB](#)
- [Adafruit WavePlayer](#)
- [SdFat \(Adafruit Fork\)](#)
- [Audio - Adafruit Fork](#)

Arduino Test 133

Graphics Demos 134

Arcada Library 135

- [Initialization](#)
- [Joystick & Buttons](#)
- [Backlight, Speaker and Sensors](#)
- [Alert Boxes](#)

Arcada Library Docs 137

Adapting Sketches to M0 & M4 137

- [Analog References](#)
- [Pin Outputs & Pullups](#)
- [Serial vs SerialUSB](#)
- [AnalogWrite / PWM on Feather/Metro M0](#)
- [analogWrite\(\) PWM range](#)
- [analogWrite\(\) DAC on A0](#)
- [Missing header files](#)
- [Bootloader Launching](#)
- [Aligned Memory Access](#)
- [Floating Point Conversion](#)
- [How Much RAM Available?](#)
- [Storing data in FLASH](#)
- [Pretty-Printing out registers](#)
- [M4 Performance Options](#)
- [CPU Speed \(overclocking\)](#)
- [Optimize](#)
- [Cache](#)
- [Max SPI and Max QSPI](#)
- [Enabling the Buck Converter on some M4 Boards](#)

Downloads 146

- [Files:](#)
- [Fab Print](#)
- [Schematic](#)

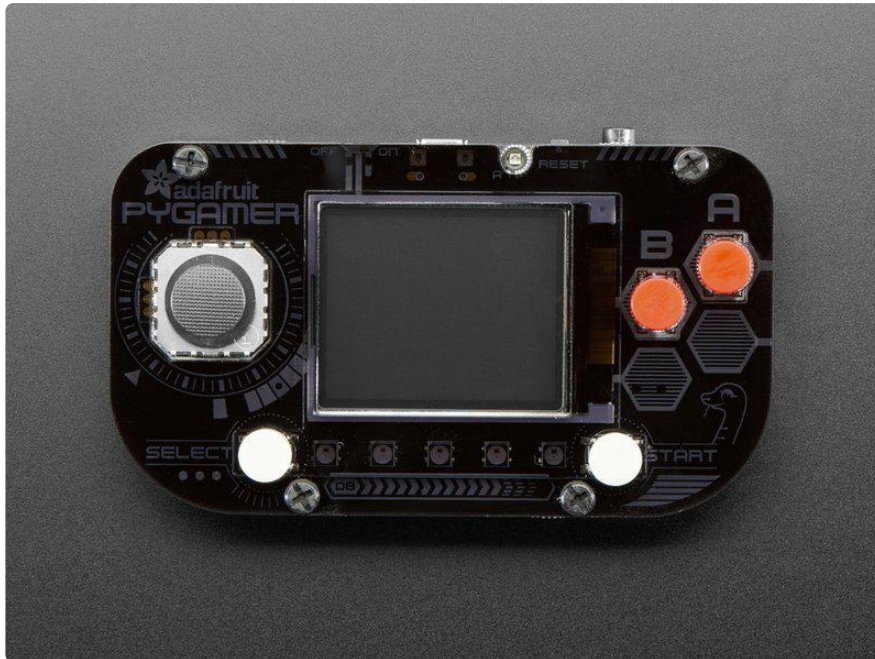
- [Laser Cut Acrylic Case](#)
- [3D Model](#)

Troubleshooting

149

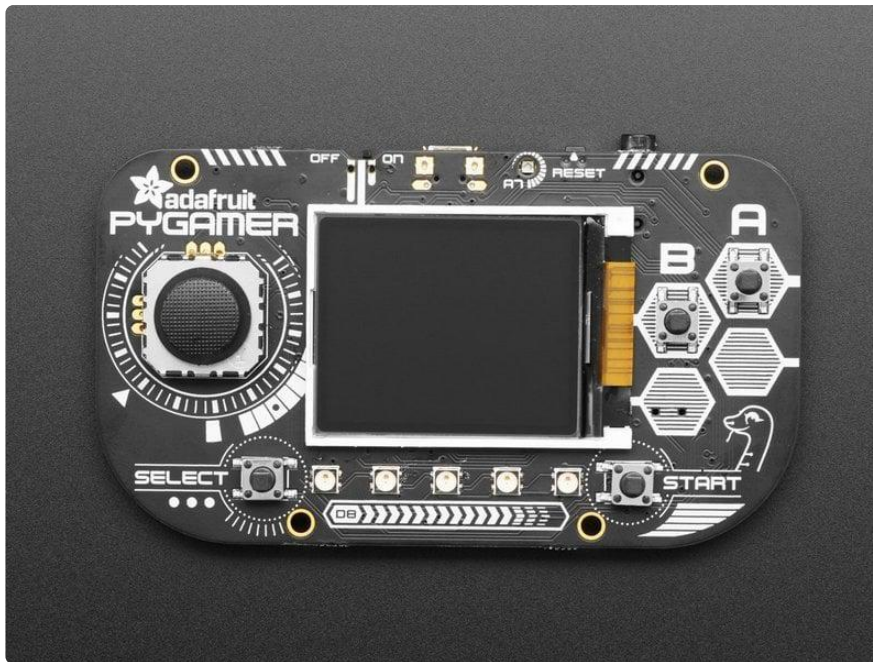
- [Digi-Key x Adafruit Order Shipper Game](#)
- [Screen Adhesive Fix](#)

Overview



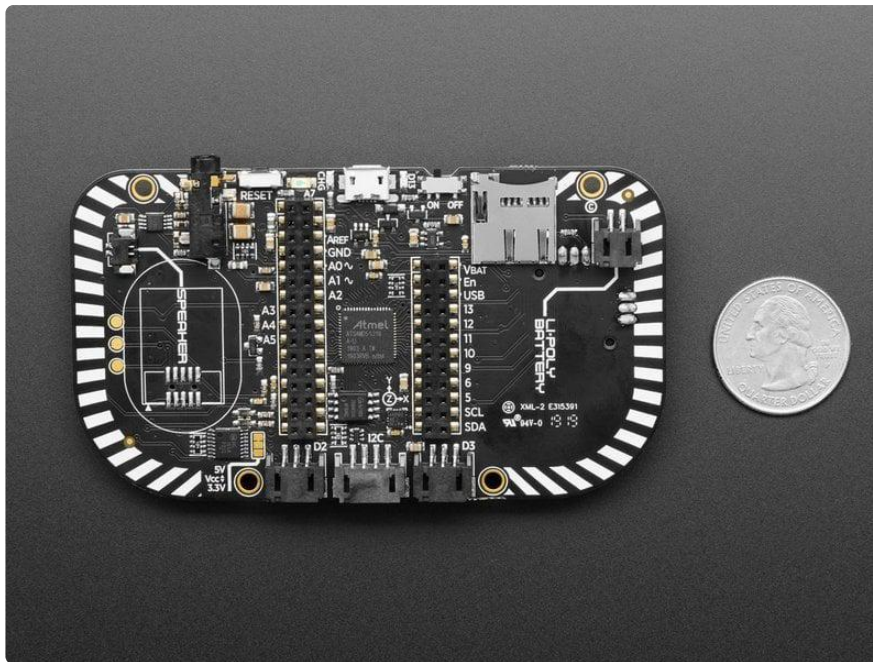
What fits in your pocket, is fully Open Source, and can run CircuitPython, MakeCode Arcade or Arduino games you write yourself? That's right, it's the Adafruit PyGamer! We wanted to make an entry-level gaming handheld for DIY gaming, and maybe a little retro-emulation. It's not the fastest and best of everything but it is an all-in-one dev board with a lot of possibilities!

The PyGamer is powered by our favorite chip, the ATSAM51, with 512KB of flash and 192KB of RAM. We add 8 MB of QSPI flash for file storage, handy for images, fonts, sounds, or game assets.



On the front you get a 1.8" 160x128 color TFT display with dimmable backlight - we have fast DMA support for drawing so updates are incredibly fast. A dual-potentiometer analog stick gives you great control, with easy diagonal movement - or really any direction you like. There's also 4 square-top buttons, [which fit our square top button caps \(\)](#). The buttons are arranged to mimic a gaming handheld, with 2 menu-select buttons and 2 fire-action buttons. There's also 5 NeoPixel LEDs to dazzle or track activity.

On the back we have a full Feather-compatible header socket set, so you can plug in any FeatherWing to expand the capabilities of the PyGamer. There's also 3 STEMMA 2.0mm PH connectors - [two 3-pin with ADC/PWM capability \(\)](#) and [one 4-pin that connects to I2C \(\)](#) - you can use this for Grove sensors as well as [Stemma QT with an adapter cable \(\)](#).



For built in sensors, there's a light sensor that points out the front, and a 3-axis accelerometer that can detect taps and free-fall. To make bleeps and bloops, plug in any set of stereo headphones. [For projects where you need more volume, you can plug in one of our 8 ohm speakers \(\)](#). The PyGamer will auto-switch to speakers when they're plugged in.

[You can power the PyGamer from any of our LiPoly batteries \(\), but we like this 350mAh one \(\)](#) which will fit into the acrylic case. An on-off switch will save battery power when not in use. Or power from the Micro USB port - it will also charge up the battery if one is attached.

Now, how to program it? Well you've got a lot of options!

- MakeCode Arcade is the easiest to start for making games, [you can drag-and-drop blocks and load games over the disk-drive bootloader \(\)](#)
- [CircuitPython \(\)](#) lets you draw graphics, play wave files and print out text in any fonts - all in Python! There's tons of sensor support as well.
- Arduino is low level, powerful, but a little more challenging. [You can use Adafruit Arcada \(\)](#) to interface with the hardware and it will abstract some of the nitty-gritty details like reading buttons for you.



Here's a list of everything you get

- ATSAM51J19 @ 120MHz with 3.3V logic/power - 512KB of FLASH + 192KB of RAM
- 8 MB of QSPI Flash for storing images, sounds, animations, whatever!
- Micro SD Card Slot for storing even more stuff when the QSPI flash isn't enough
- 1.8" 160x128 Color TFT Display connected to its own SPI port
- 1 x Analog Thumbstick with X and Y analog inputs
- 4 x Game/Control Buttons with square tops
- 5 x NeoPixels for dazzle, or game score-keeping
- Triple-axis accelerometer (motion sensor)
- Light sensor, reverse-mount so that it points out the front
- Stereo headphone jack
- Mono Class-D speaker driver for 4-8 ohm speakers, up to 2 Watts
- LiPoly battery port with built in recharging capability
- USB port for battery charging, programming and debugging
- Two female header strips with Feather-compatible pinout so you can plug any FeatherWings in
- JST ports for NeoPixels, sensor input, and I2C (you can fit I2C Grove connectors in here)
- Reset button
- On-Off switch



Which Board?

We have a few very similar boards in the PyGamer/PyBadge family - you may wonder which one you want! Well here we will outline it for you.

Basically, PyGamer has the most stuff and is more expensive. PyBadge is in the middle, has a lot of stuff, and PyBadge LC is low cost, very minimal!

Similarities

- All the boards use the SAMD51 microcontroller, which runs at 120MHz (but we often overclock to 200MHz)
- All the boards have onboard QSPI storage of some size
- All the boards have a display
- All the boards have an on/off switch and reset button
- All the boards can run Arduino, CircuitPython and Microsoft MakeCode Arcade
- All have some way to control direction (X & Y) as well as 4 buttons
- All have at least one NeoPixel
- All have a light sensor

Differences between PyGamer and PyBadge/PyBadge LC

- PyGamer has an Analog Joystick instead of PyBadge's 4x D-Pad buttons, which makes it easier to use for gaming
- PyGamer has a stereo headphone jack - the PyBadges do not
- PyGamer has a connector for a loud speaker but does not have a simple buzzer built in (we expect you to use the nicer loud speaker or headphone jack)
- PyGamer has an SD card slot - the PyBadges do not
- PyGamer has an 8 MB QSPI Flash chip - the PyBadges have the smaller 2MB

Differences between PyBadge & PyBadge LC

- PyBadge LC does not have a connector for a loudspeaker, both have a simple buzzer.
- PyBadge LC does not have a Feather Header set on the back
- PyBadge LC does not have STEMMA connectors for quick hardware addition
- PyBadge LC does not have an accelerometer

Comparison Table

Feature	PyGamer	PyBadge	PyBadge LC
Processor	SAMD51J19	SAMD51J19	SAMD51J19
FLASH/RAM	512KB / 192KB	512KB / 192KB	512KB / 192KB
QSPI FLASH	8MB	2MB	2MB
On/Off & Reset Switches	Yes	Yes	Yes

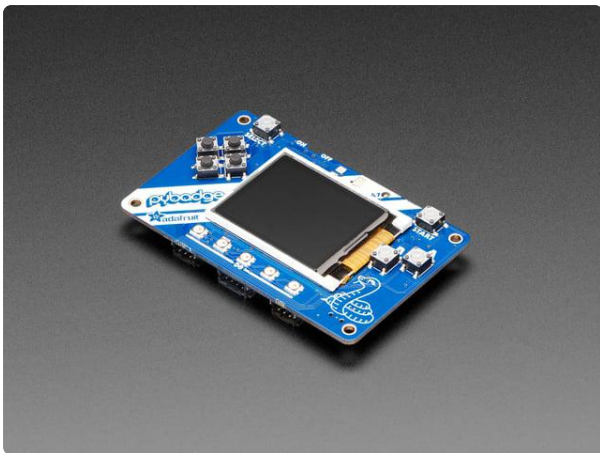
LiPoly Battery Charging	Yes	Yes	Yes
X & Y Controls	Analog Thumbstick	4 Button D-Pad	4 Button D-Pad
A / B / Select / Start	Yes	Yes	Yes
Built-in buzzer/beeper	No	Yes	Yes
Connection for Speaker	Yes	Yes	No
Stereo Headphone Jack	Yes	No	No
Display	160x128 Color TFT	160x128 Color TFT	160x128 Color TFT
NeoPixels	5	5	1
Micro-SD Card	Yes	No	No
Light Sensor	Yes	Yes	Yes
Accelerometer	Yes	Yes	No
FeatherWing Headers	Yes	Yes	No
STEMMA Connectors	Yes	Yes	No



[Adafruit PyGamer for MakeCode Arcade, CircuitPython or Arduino](https://www.adafruit.com/product/4242)

What fits in your pocket, is fully Open Source, and can run CircuitPython, MakeCode Arcade or Arduino games you write yourself? That's right, it's the Adafruit...

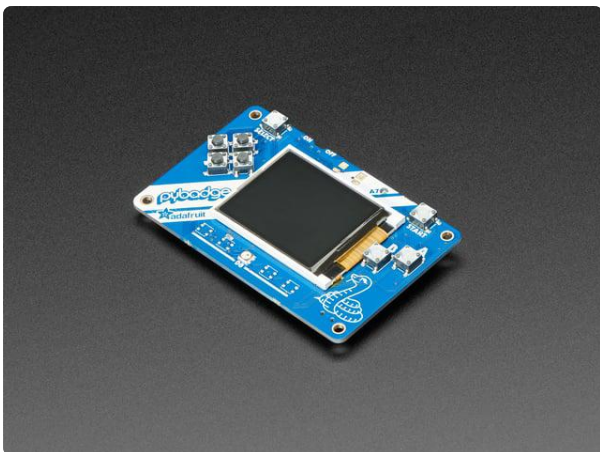
<https://www.adafruit.com/product/4242>



[Adafruit PyBadge for MakeCode Arcade, CircuitPython, or Arduino](https://www.adafruit.com/product/4200)

What's the size of a credit card and can run CircuitPython, MakeCode Arcade or Arduino? That's right, its the Adafruit PyBadge! We wanted to see how much we...

<https://www.adafruit.com/product/4200>



[Adafruit PyBadge LC - MakeCode Arcade, CircuitPython, or Arduino](https://www.adafruit.com/product/3939)

What's the size of a credit card and can run CircuitPython, MakeCode Arcade or Arduino even when you're on a budget? That's right, it's the Adafruit...

<https://www.adafruit.com/product/3939>

Update the PyGamer Bootloader

Update the PyGamer Bootloader to prevent a problem with MacOS 10.14.4 and to fix button problems, and to prevent occasional damage to the loaded program.

Your PyGamer may need its bootloader updated for several reasons.

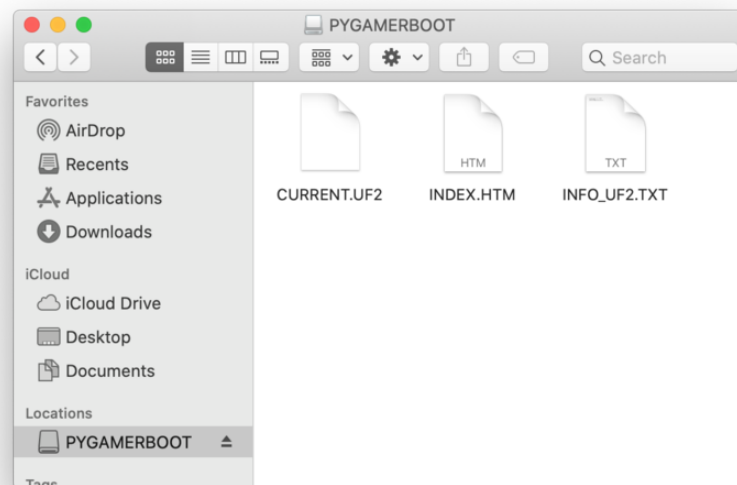
Bootloaders earlier than v3.9.0 do not protect against a rare problem in which part of internal flash is erased on startup.

Starting with MacOS 10.14.4, Apple changed how USB devices are recognized on certain Macs. This caused a timing problem with boards that were loaded with a MakeCode program, preventing the **PYGAMERBOOT** drive from appearing. Also the A and B buttons will be reversed in MakeCode if your bootloader is too old.

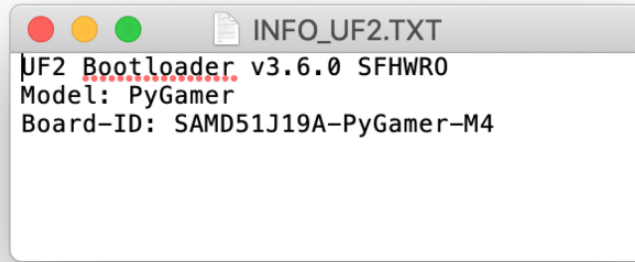
Updating Your PyGamer Bootloader

To see if you need to update your bootloader, get the UF2 boot drive to appear on your board. If you're running MakeCode, click the reset button once. If you're running CircuitPython or an Arduino program, double-click the reset button.

When you see **PYGAMERBOOT**, click the **PYGAMERBOOT** drive in the Finder and then double-click the **INFO_UF2.TXT** file to see what's inside.



The bootloader version is listed in **INFO_UF2.TXT**. In this example, the version is v3.6 .0.

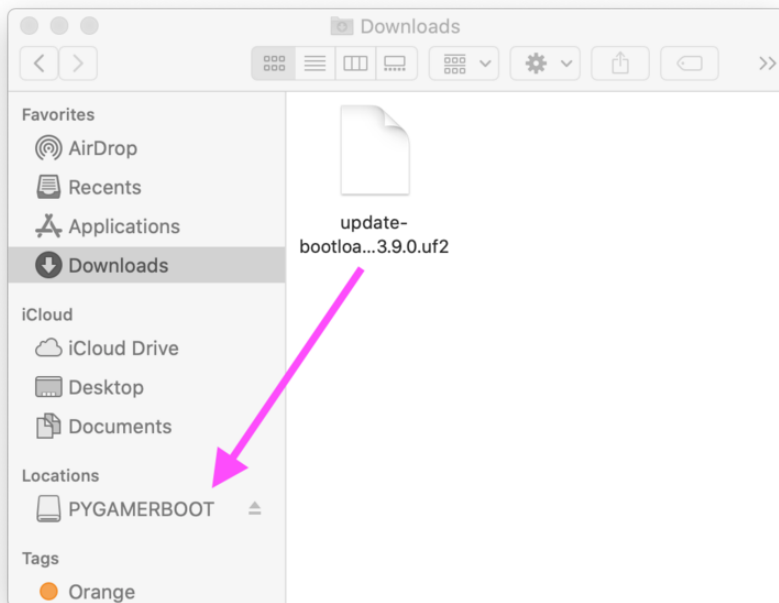


If the bootloader version you see is smaller than "v3.9.0", you need to update. For instance, the bootloader above needs to be upgraded.

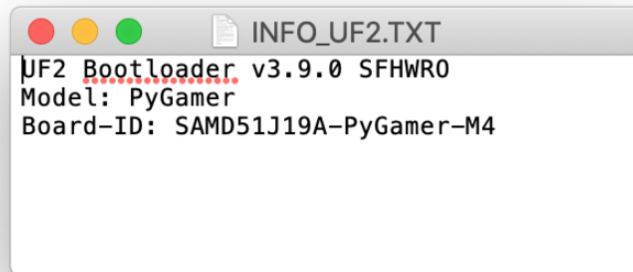
Download the latest version of the PyGamer bootloader updater from the [circuitpython.org Downloads](https://circuitpython.org/downloads) page.

Latest PyGamer downloads

The bootloader updater will be named `update-bootloader-arcade_pygamer-v3.9.0.uf2` or some later version. Drag that file from your `Downloads` folder onto the bootloader drive: `ARCADE-D5`, `GAMERBOOT`, or `PYGAMERBOOT`.



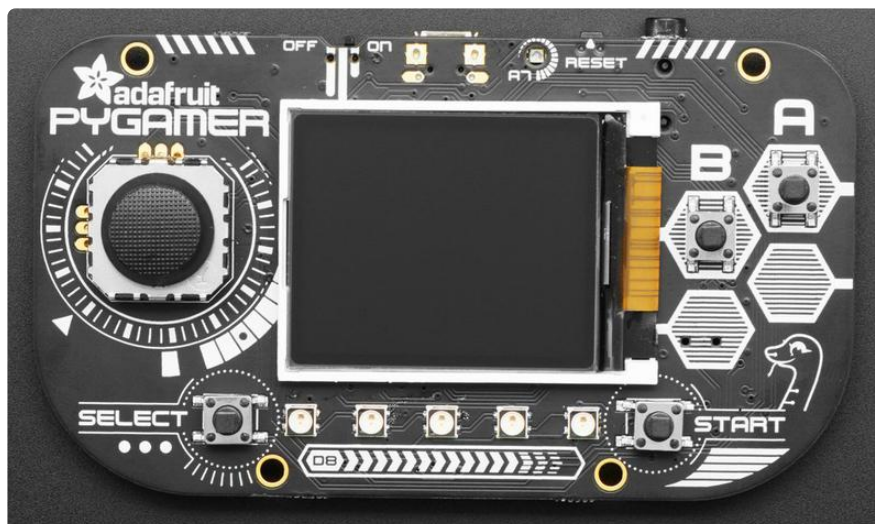
After you drag the updater onto the boot drive, the red LED on the board will flicker and then blink slowly about five times. A few seconds later, **PYGAMERBOOT** will appear in the Finder. After that, you can click on **PYGAMERBOOT** and double-click **INFO_UF2.TXT** again to confirm you've updated the bootloader.



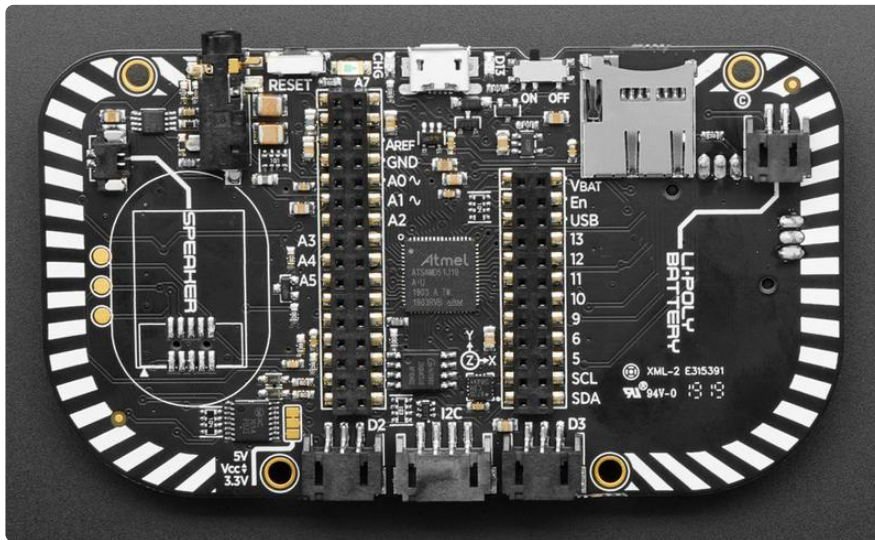
Oh no, I updated MacOS already and I can't see the boot drive!

If your Mac has already been updated to MacOS 10.14.4 and now you can't see a boot drive in the Finder, you need to find another computer that will work. Not all upgraded Macs will fail to show the boot drive: older ones can work. Or find a Mac that hasn't been upgraded yet. Any Windows 10 or Linux computer should work for upgrading your bootloader. Windows 7 computers [will need drivers installed \(\)](#), but then can work.

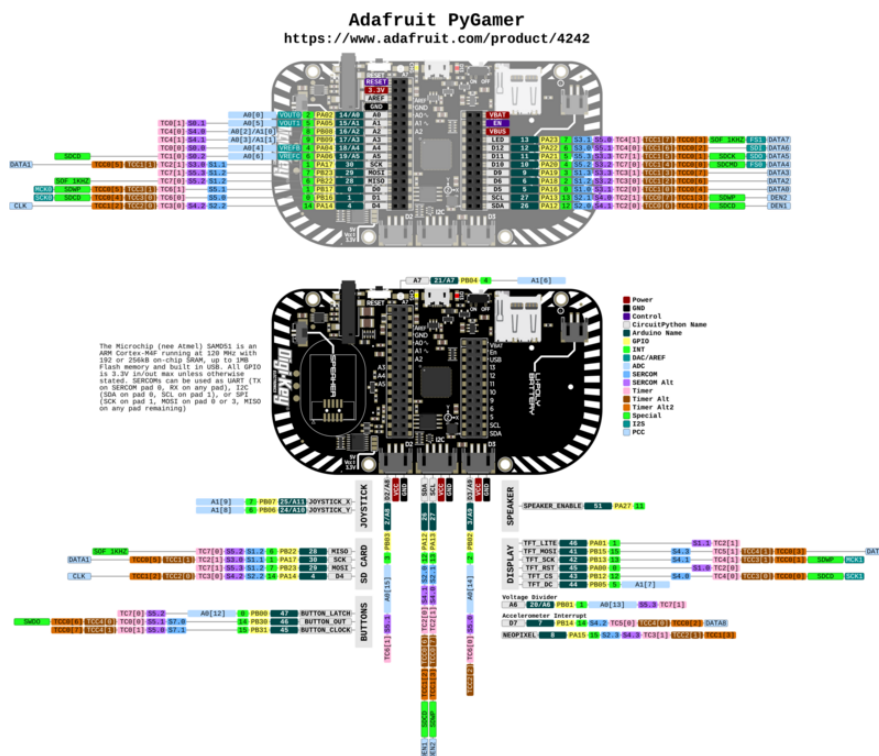
Pinouts



The PyGamer is a DIY handheld gaming development platform, and it's loaded with all sorts of goodies. The front features a display, buttons, a joystick, RGB LEDs, and a light sensor.

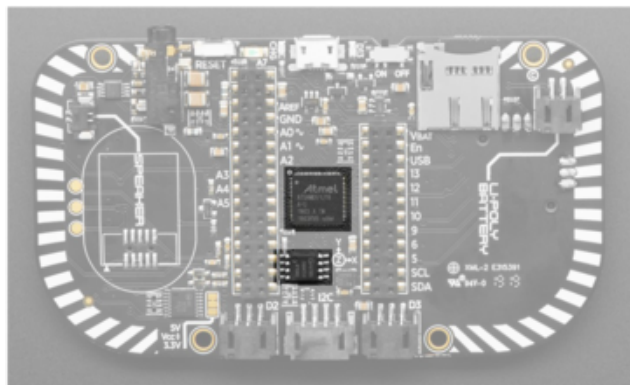


The back features a speaker connector, headphone jack, SD card slot, battery connector, peripheral connectors, an accelerometer and more. Let's take a look!



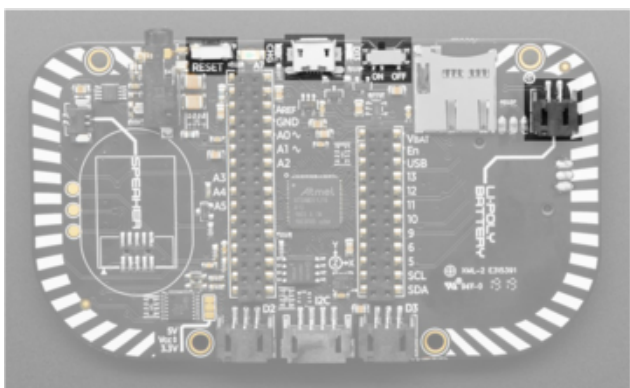
[Click here to view a PDF version of the pinout diagram \(\)](#)

Microcontroller and Flash



The main processor chip is the ATSAM51J19 Cortex M4 running at 120MHz with 3.3v logic/power. It has 512KB of Flash and 192KB of RAM. We also include 8 MB of QSPI Flash for storing images, sounds, animations, whatever.

Power

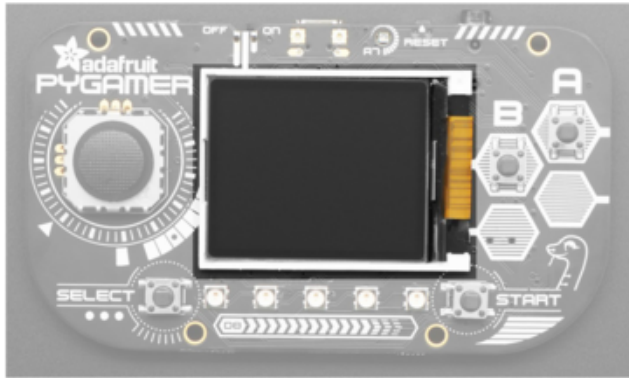


The On/Off switch is located on the top. There is one USB port on the board. On the left side, towards the bottom, is a USB Micro port, which is used for powering and programming the board.

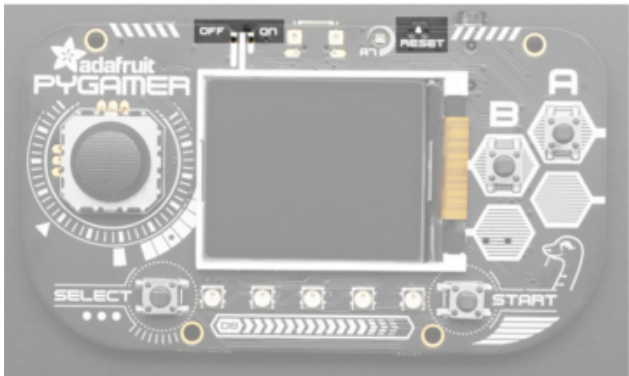
There's two ways to power your PyGamer. The best way is to plug in a 3.7/4.2V Lipoly battery into the JST 2-PH port. You can then recharge the battery over the Micro USB jack. You can also just run the board directly from Micro USB, it will automatically 'switch over' to USB power when that's plugged in.

Pin A6 is connected to a voltage divider which gives half the current battery voltage. You can read the battery voltage by using the Arcada library function `readBatterySensor()` (it multiplies by two to give the actual voltage), by using `analogRead(A6)` in Arduino, or by using `analogio.AnalogIn(board.A6)` in CircuitPython.

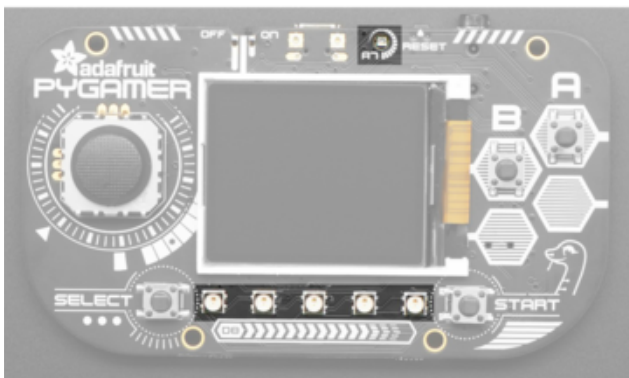
Display



1.8" TFT display - The front features a 160x128 pixel color display.

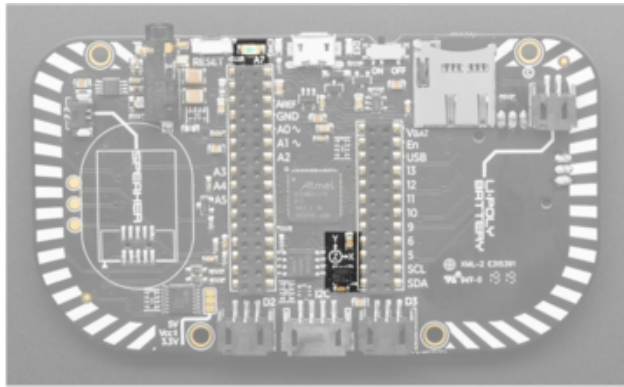


On/Off switch - Powers off the board. Even if plugged in, the board won't work if switched to off. You can charge the battery when the board is off, but the USB device will not be active or any other electronics. The reset button is located on the top to the right of center. Click it once to re-start your firmware. Click twice to enter bootloader mode.

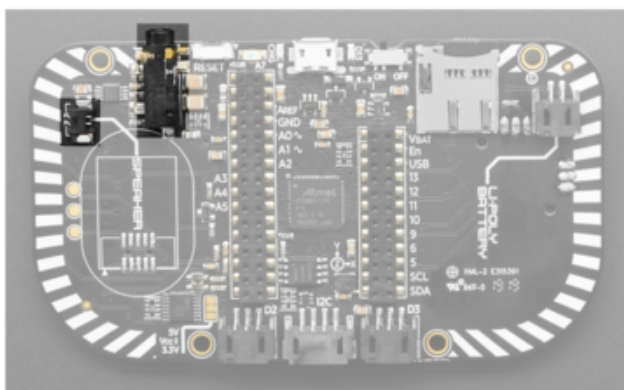


Light sensor (A7) - There is an ambient light sensor on the top, which points through to the front. The light sensor is an analog input, connected to `board.LIGHT` (CircuitPython) or `A7` (Arduino) you can read it as any analog value ranging from 0 (dark) to 1023 (in Arduino) or 65535 (CircuitPython) when bright.

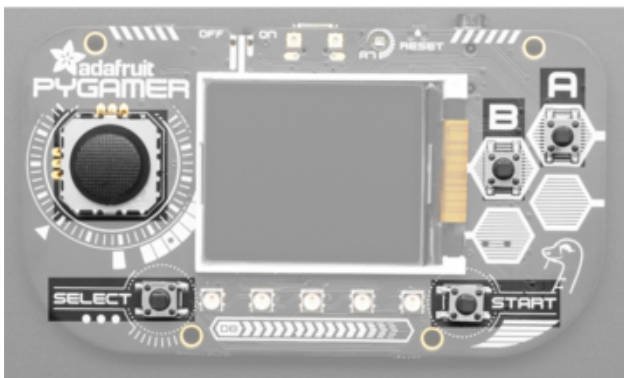
NeoPixels (D8) - There are also 5 individually addressable RGB NeoPixel LEDs located on the front of the board along the bottom middle. They are connected to `board.NEOPIXEL` (CircuitPython) or `8` (Arduino)



Accelerometer - There is an accelerometer located near the middle, above the I2C connector. It is an LIS3DH.



There is a speaker connector on the left side of the back, which is a [Molex PicoBlade](#) (). You can attach one of the speakers available in the Adafruit shop. There is also a stereo headphone jack on the left top.



For gaming interface, there are 4 buttons and an analog thumbstick

There is a thumb joystick on the left side of the board. The thumb joystick is a dual potentiometer, one pot for X axis and one for Y axis. You can read the two analog values to determine the position of the joystick. For example, the reading will be at 0V (ground) when the X axis is all the way to the left and 3.3V (analog-max) when the stick is all the way to the right. Arduino **A11** is joystick X, and Y is **A10**.

```
uint16_t joyy = analogRead(A10);  
uint16_t joyx = analogRead(A11);
```

In CircuitPython they are just named `JOYSTICK_X` and `JOYSTICK_Y`

```
joystick_x = analogio.AnalogIn(board.JOYSTICK_X)
joystick_y = analogio.AnalogIn(board.JOYSTICK_Y)
```

For information on reading the values, check out the [Reading Analog Pin Values section of the CircuitPython Essentials Analog In page \(\)](#).

There are four buttons: A, B, select and start. These four pads are not connected to GPIO pins. Instead, they are connected to a latch via 3 digital pins that will read up to 8 inputs and send the data over one bit at a time. If using Arduino, this psuedo-code snippet will read the 8 bits for you. A 0 bit indicates no press. A 1 bit indicates a press.

```
#define BUTTON_CLOCK 48
#define BUTTON_DATA 49
#define BUTTON_LATCH 50

uint8_t read_buttons() {
    uint8_t result = 0;

    pinMode(BUTTON_CLOCK, OUTPUT);
    digitalWrite(BUTTON_CLOCK, HIGH);
    pinMode(BUTTON_LATCH, OUTPUT);
    digitalWrite(BUTTON_LATCH, HIGH);
    pinMode(BUTTON_DATA, INPUT);

    digitalWrite(BUTTON_LATCH, LOW);
    digitalWrite(BUTTON_LATCH, HIGH);

    for(int i = 0; i < 8; i++) {
        result &&= 1;
        //Serial.print(digitalRead(BUTTON_DATA)); Serial.print(", ");
        result |= digitalRead(BUTTON_DATA);
        digitalWrite(BUTTON_CLOCK, HIGH);
        digitalWrite(BUTTON_CLOCK, LOW);
    }
    Serial.println();
    return result;
}
```

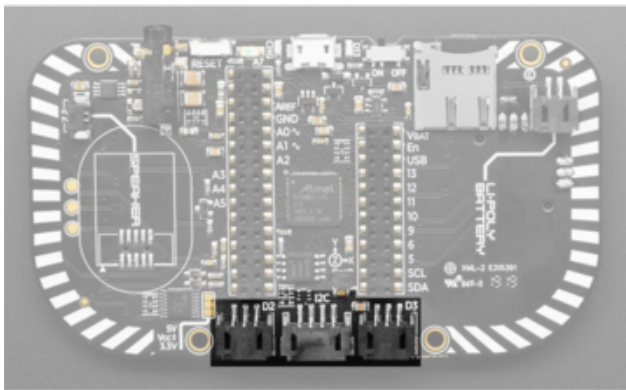
[But we would recommend using Arcada instead, where a lot of this abstraction is handled for you \(\)](#)

In CircuitPython, use the `keypad.ShiftRegisterKeys` library to read the button presses for you:

```
k = keypad.ShiftRegisterKeys(
    clock=board.BUTTON_CLOCK,
    data=board.BUTTON_0OUT,
    latch=board.BUTTON_LATCH,
    key_count=8,
    value_when_pressed=True,
)
while True:
```

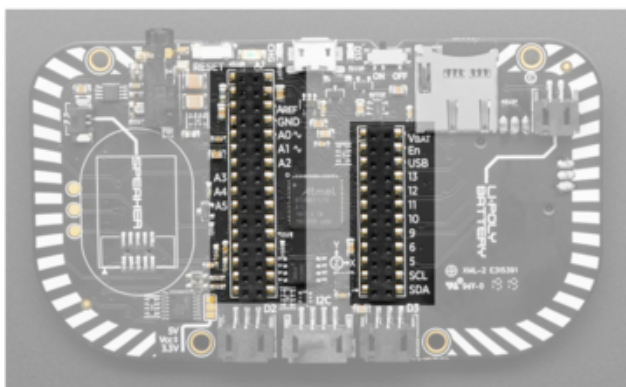
```
event = k.events.get()
if event:
    print(event)
```

[More info on keypad.ShiftRegisterKeys is here \(\)](#). There is also a [guide covering it's usage here. \(\)](#)

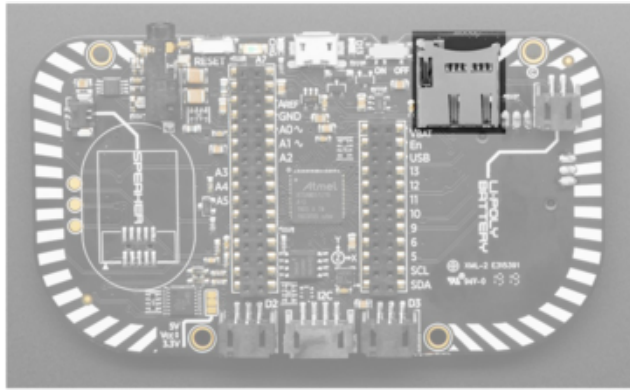


There is a 4-pin JST PH 2.0mm I2C connector in the center on the bottom, that is STEMMA and Grove compatible as well as [Stemma QT with an adapter cable \(\)](#). You can access VCC power and ground as well as a level-shifted **SDA** & **SCL** connection. In CircuitPython, you can use the STEMMA connector with **board.SCL** and **board.SDA**, or **board.STEMMA_I2C()**. You can change VCC from 5V (default) to 3V by cutting/soldering the solder jumper to the right of the D3 connector.

On the bottom are two connectors labeled **D2** and **D3**. These are 3-pin JST PH 2.0mm digital or analog connectors for sensors or NeoPixels. These pins can be analog inputs or digital I/O. D2 is also known as **A8**, D3 is also known as **A9** for analog reads.

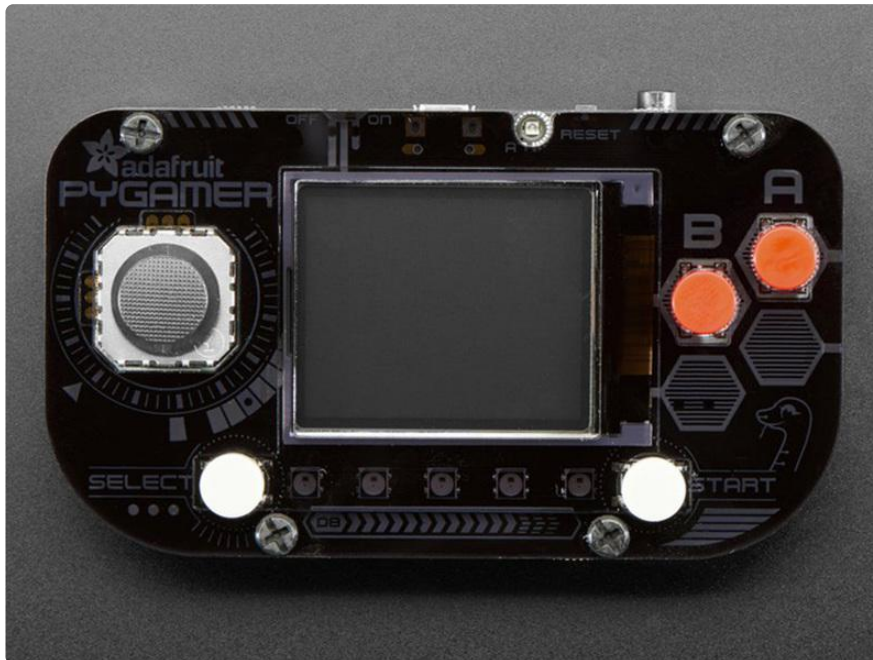


You can easily attach FeatherWings to the back of your PyGamer using the convenient "Feather Headers" on the back. Located in the middle, they break out all the same pins you have access to on a Feather board, allowing use of any of our wide range of FeatherWings. Easily add all kinds of functionality to your PyGamer! The GPIO is all 3.3V logic.

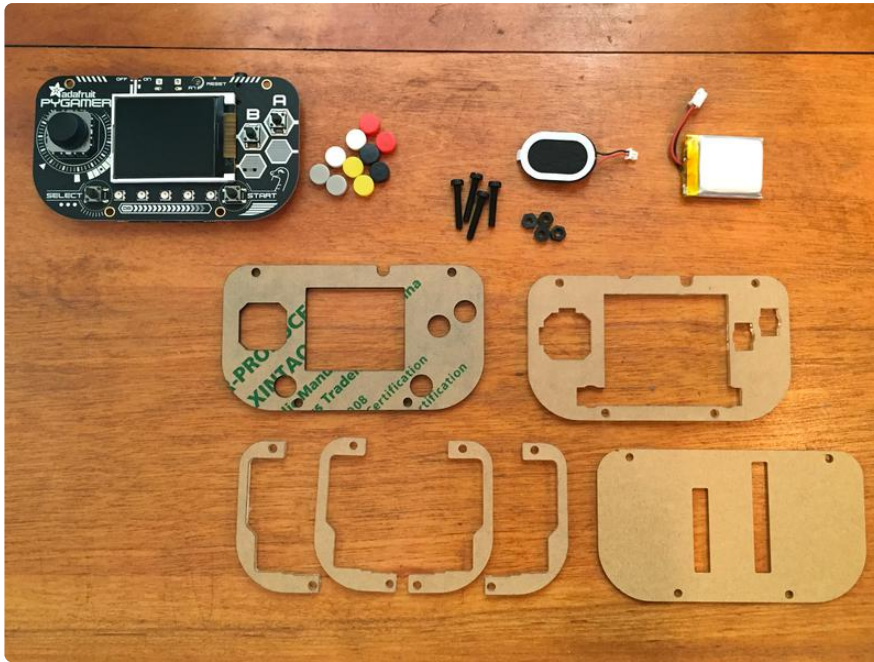


On the back, there is a micro SD card slot. Insert a micro SD card for even more storage for more games, images, sounds, etc.

Build the PyGamer Case

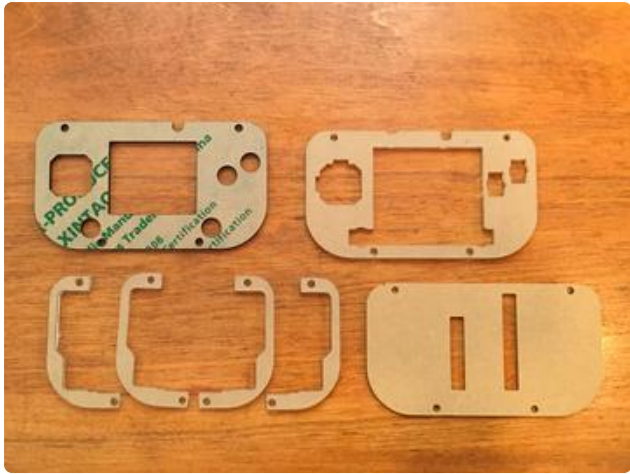


Here's how to assemble the laser cut acrylic case for the PyGamer. The kit comes with seven pieces of acrylic, and four screws and nuts. You've got ten button caps to pick from (you'll pick four), and you'll also want to connect the speaker and battery for the full portable experience.



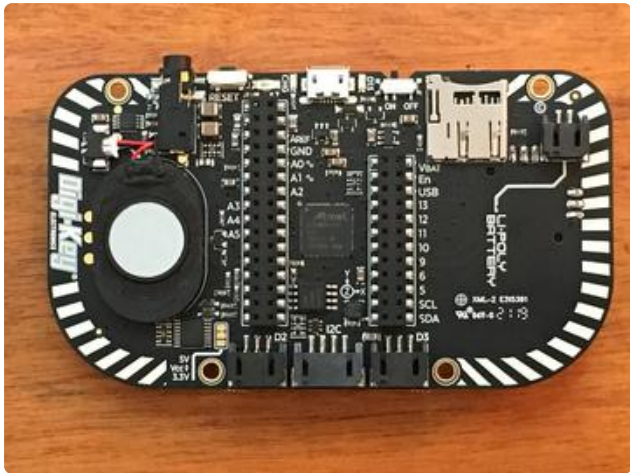
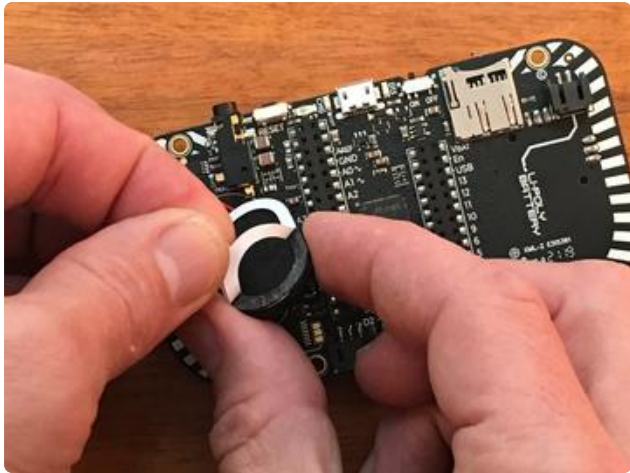
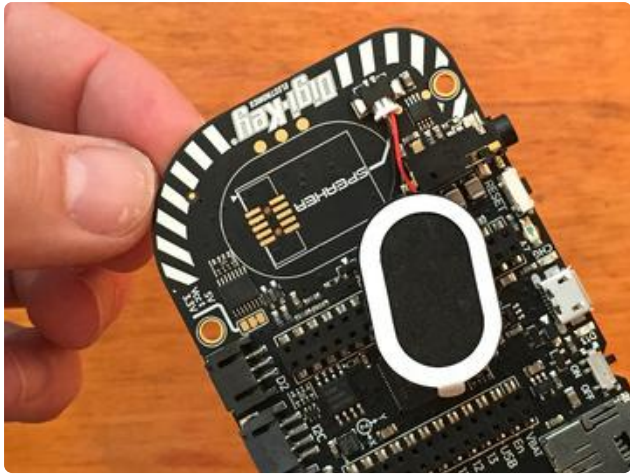
Prep

If you haven't already, remove the clear plastic screen protector film from the PyGamer display.



Paper Protection

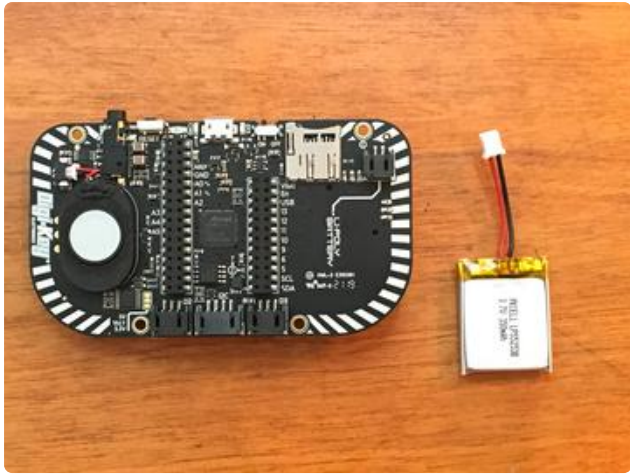
Remove the protective paper backing from both sides of all the acrylic pieces.



Speaker

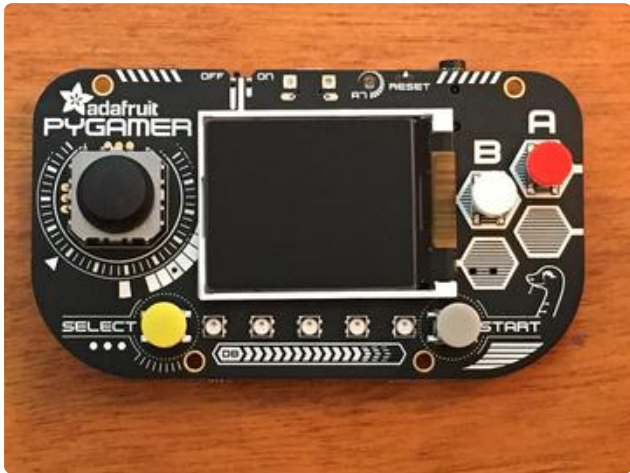
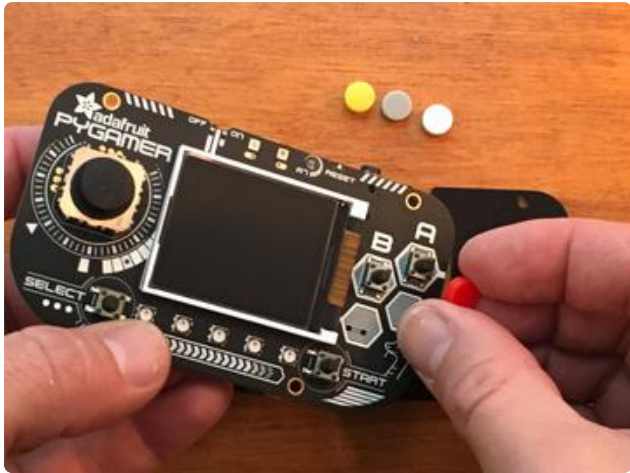
Plug the speaker into the speaker port on the PyGamer.

Then, remove the white oval plastic ring to expose the adhesive and press the speaker to the PyGamer where the silkscreen oval outline indicates.



Battery

Plug the battery into the on-board connector. Very carefully, bend the wires so that the battery fits the spot shown.



Button Caps

Pick four of the button caps and click them into place on the square shafts of the buttons. Which color combo will you choose?!

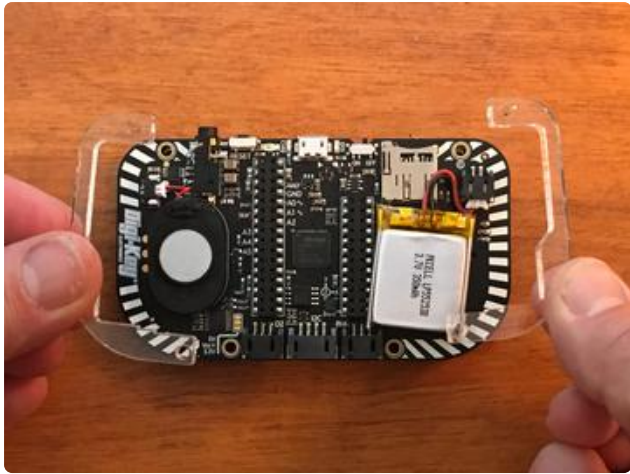


Case Layers

The case assembly is pretty simple. Place the clear top side piece on as shown.

Next, place the smoked gray piece on.





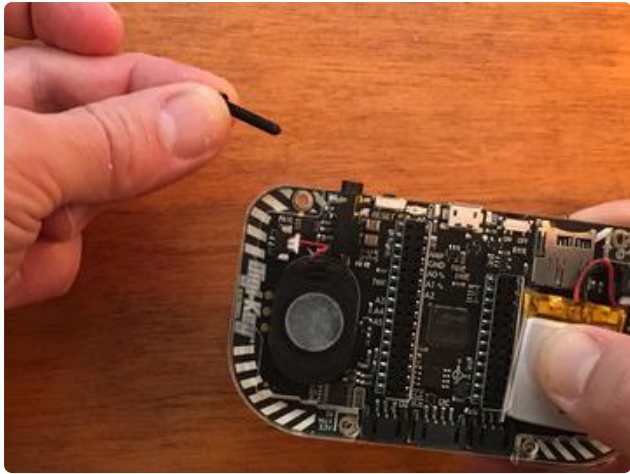
Spacers

Flip the board over, then place the four spacer pieces onto the back as shown.



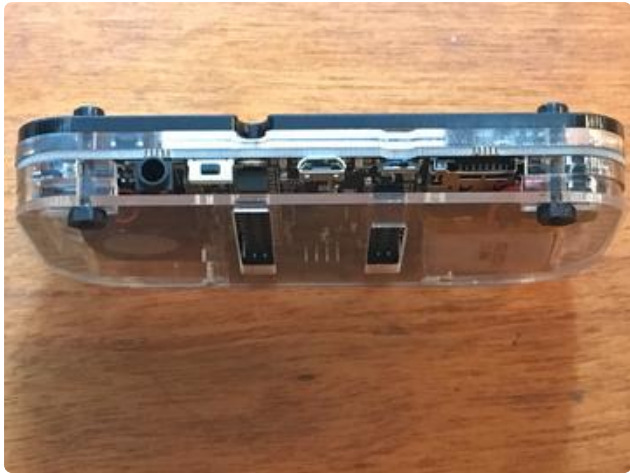
Backing

The last piece to go on is the thin bottom layer with the Feather header cutouts.

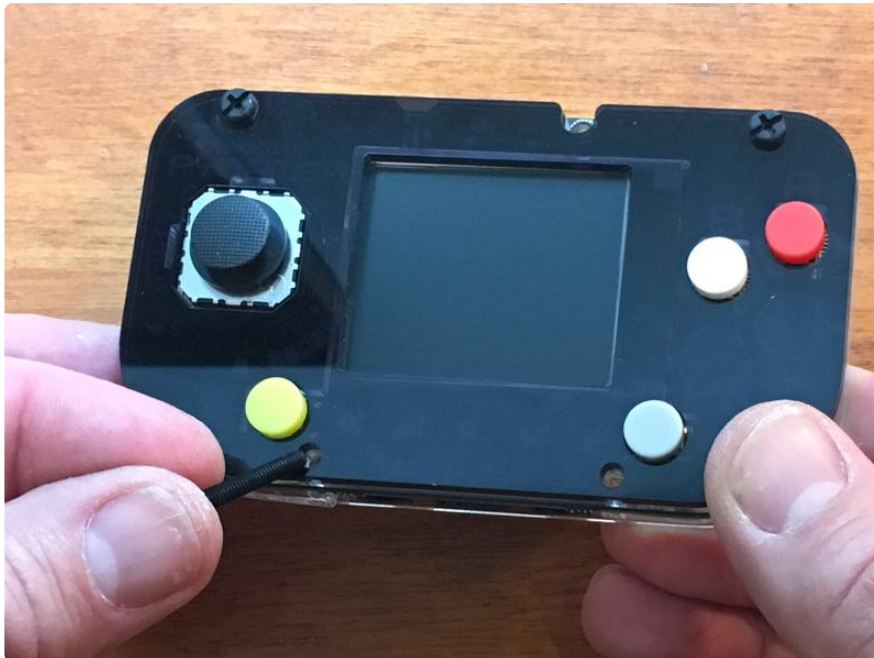


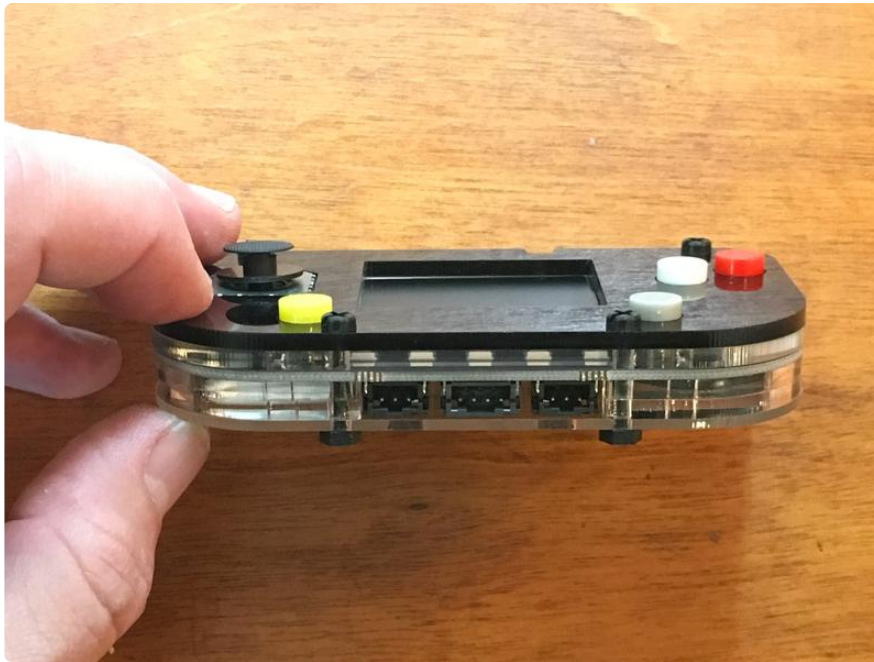
Fasteners

Push the four screws through from front to back, being sure they go through the holes in all layers and the PyGamer.



Screw on the nuts to secure things. Hand tight is fine -- you don't want to crack anything by using excessive force.



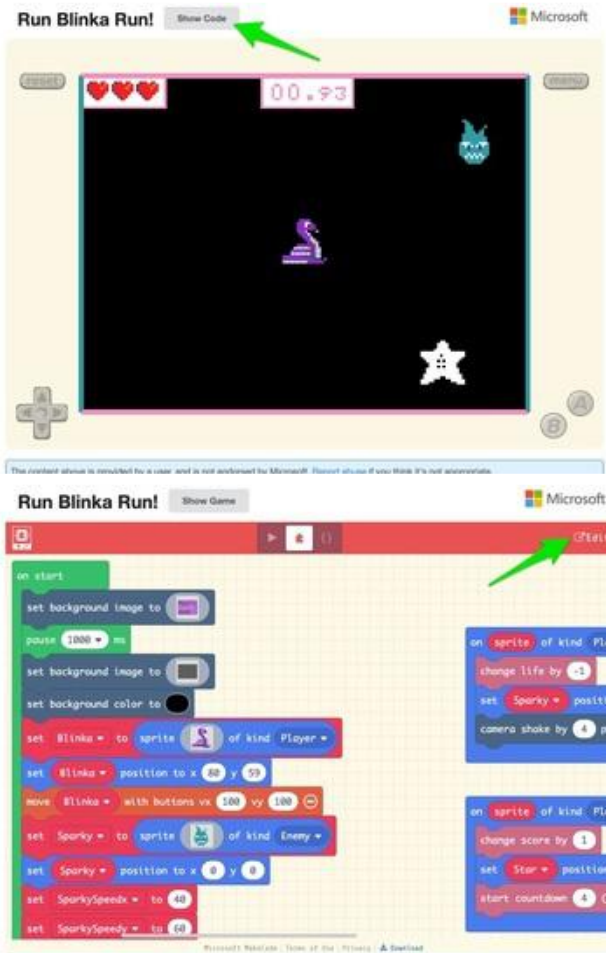


That's all there is to it -- you're ready to play with your PyGamer in its excellent, stylish case!

Load a MakeCode Game on PyGamer/ PyBadge

Let's load a game! For example, here's a link to Run, Blinka, Run! To open the game in the MakeCode Arcade editor, first, click the share link below. This will allow you to play the game in the browser right away.

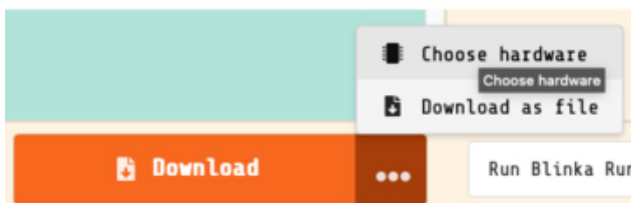
Makecode Arcade Game: Run, Blinka, Run!



Then, click on the Show Code button in the upper left corner. This shows the code for the game, and by clicking the Edit button in the upper right corner, it'll open into the editor where you can upload it to your PyGamer/PyBadge.

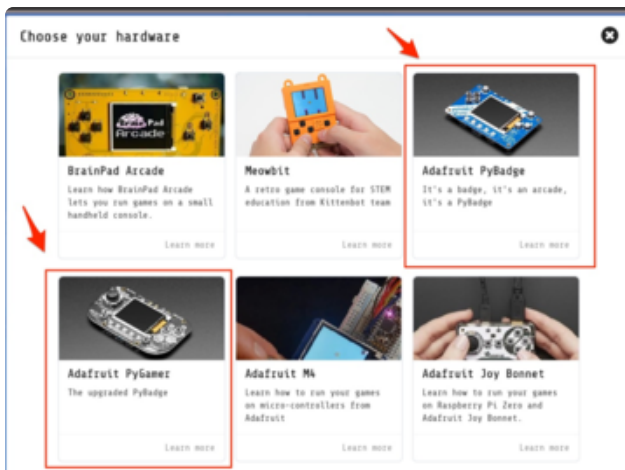
Once you have a game working on the MakeCode Arcade web editor, it's time to download it and flash it onto your board.

Please only use the Google Chrome browser with MakeCode! It has WebUSB support and seems to work best



Board Definition

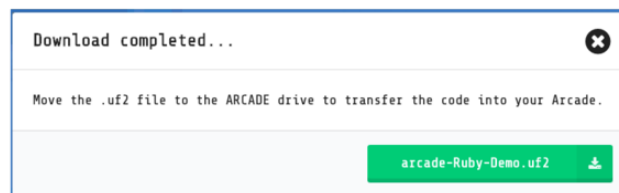
In order to load a game made in MakeCode Arcade onto the PyBadge, first choose the proper board definition inside of MakeCode. Click the ellipsis (...) next to DOWNLOAD and then the Choose Hardware item.



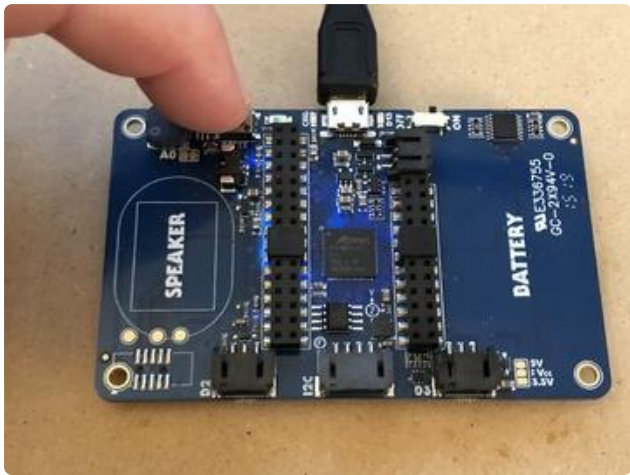
Change Board screen

Click on the image of your board, either the PyBadge/PyBadge LC or the PyGamer

This will cause the game .uf2 file for your particular board to be saved to your hard drive. You only need to do this the first time you use a new board. Thereafter you can simply click the Download button on the MakeCode Arcade editor page.



A HUUUUUUGE number of people have problems because they pick a 'charge only' USB cable rather than a "Data/Sync" cable. Make 100% sure you have a good quality syncing cable. Srsly, I can't even express how many times people have nearly given up due to a flakey USB cable!

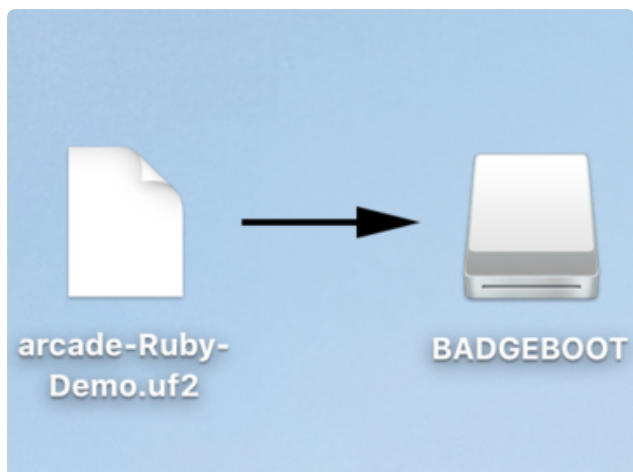


Bootloader Mode

Now, we'll put the board into bootloader mode so we can drag on the saved .uf2 file. On the back side of the board you'll see a reset button at the top. Make sure the board is plugged into your computer via USB with a USB micro B to A data cable. Also, be sure the board is turned on.

Then, press the reset button. This will initiate bootloader mode.

When the board is in bootloader mode you'll see a screen similar to this one show up.

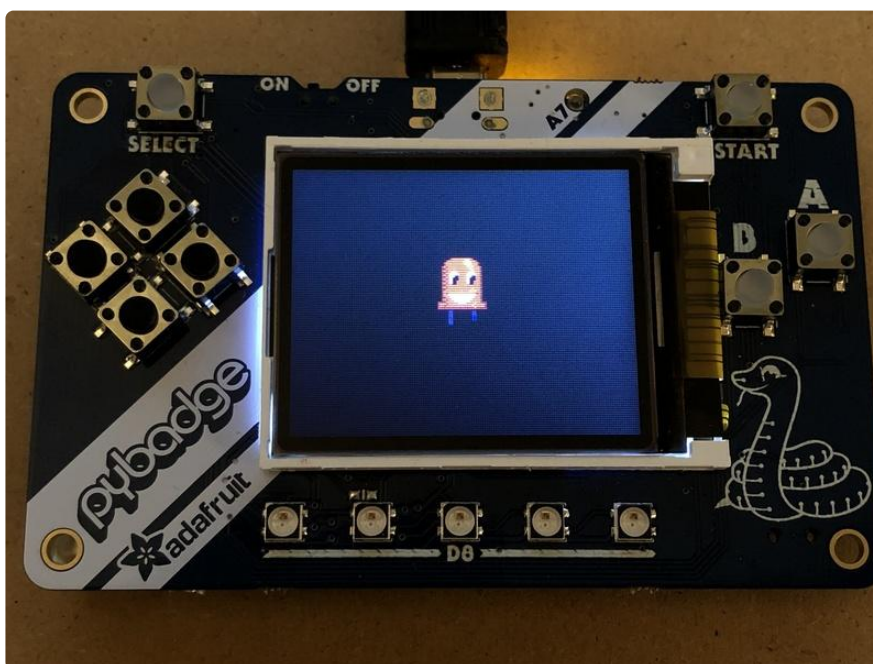


Drag and Drop

Now that the board is in bootloader mode, you should see a BADGEBOOT drive show up on your computer as a USB flash drive. Simply drag the arcade game .uf2 file onto the drive.

Play!

That's all there is to it! Once the file is copied over the board will restart and launch the game!



Keep an eye on [Adafruit.com](https://adafruit.com) for additional game related content.

CircuitPython

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

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

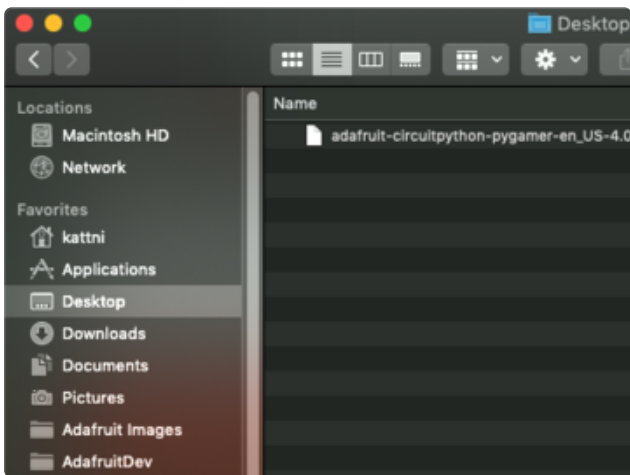
Set up CircuitPython Quick Start!

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

Download the latest version of
CircuitPython for PyGamer via
circuitpython.org

Further Information

For more detailed info on installing CircuitPython, check out [Installing CircuitPython \(\)](#).

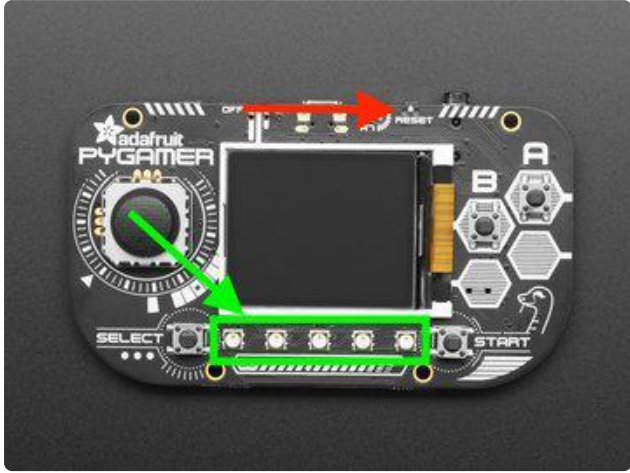


Click the link above and download the latest UF2 file.

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

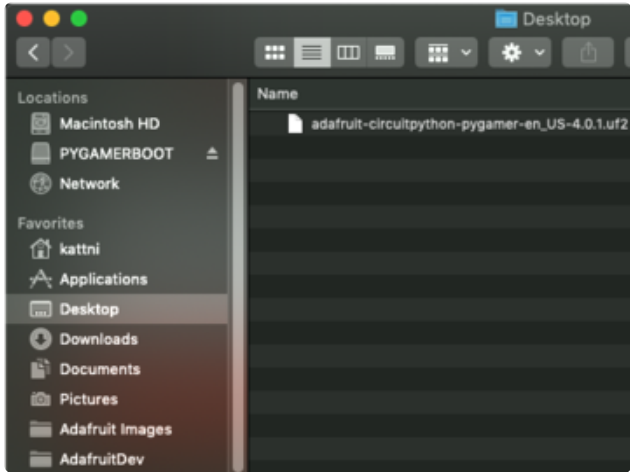
Plug your PyGamer into your computer using a known-good USB cable.

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.

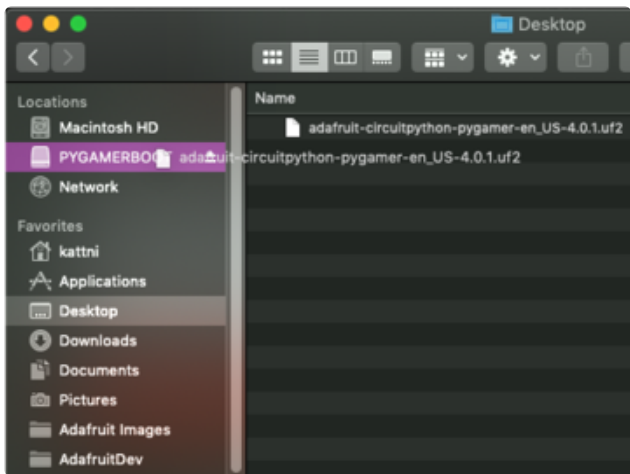


Double-click the Reset button on the top of your board (indicated by the red arrow in the first image). You will see an image on the display instructing you to drag a UF2 file to your board, and the row of NeoPixel RGB LEDs on the front will turn green (indicated by the green arrow and square in the image). If they turn red, check the USB cable, try another USB port, etc.

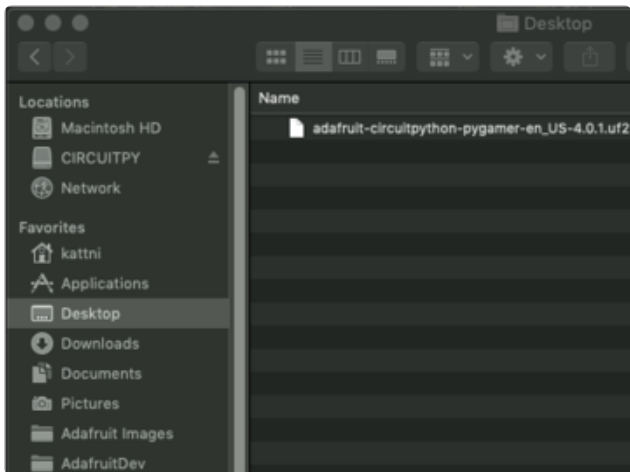
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



You will see a new disk drive appear called PYGAMERBOOT.



Drag the adafruit_circuitpython_etc.uf2 file to PYGAMERBOOT.



The LEDs will flash. Then, the PYGAMERBOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

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

Installing the Mu Editor

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

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

Download and Install Mu



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

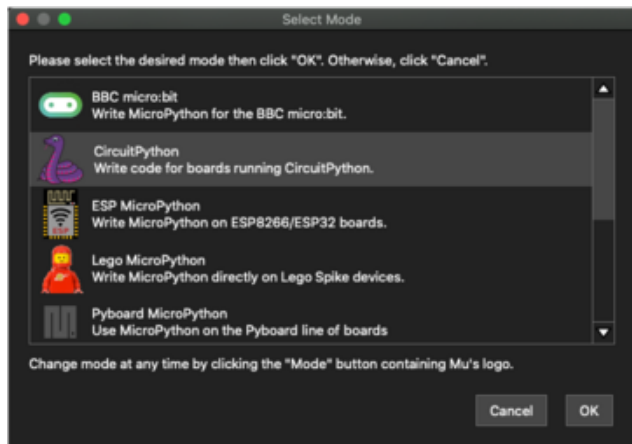
Click the Download link for downloads and installation instructions.

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

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

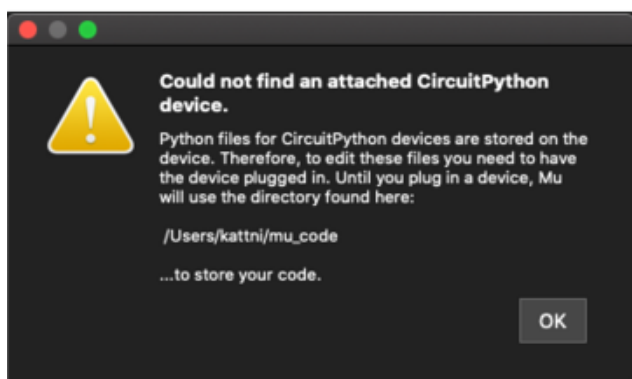
Ubuntu users: Mu currently (checked May 4, 2022) does not install properly on Ubuntu 22.04. See <https://github.com/mu-editor/mu/issues> to track this issue. See <https://learn.adafruit.com/welcome-to-circuitpython/recommended-editors> and <https://learn.adafruit.com/welcome-to-circuitpython/pycharm-and-circuitpython> for other editors to use.

Starting Up Mu



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

The current mode is displayed in the lower right corner of the window, next to the "gear" icon. If the mode says "Microbit" or something else, click the Mode button in the upper left, and then choose "CircuitPython" in the dialog box that appears.

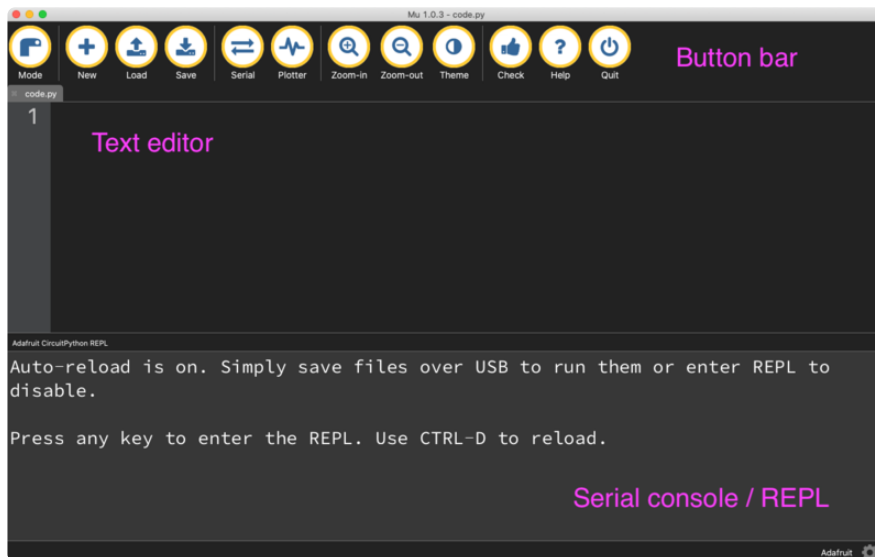


Mu attempts to auto-detect your board on startup, so if you do not have a CircuitPython board plugged in with a CIRCUITPY drive available, Mu will inform you where it will store any code you save until you plug in a board.

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

Using Mu

You can now explore Mu! The three main sections of the window are labeled below; the button bar, the text editor, and the serial console / REPL.



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

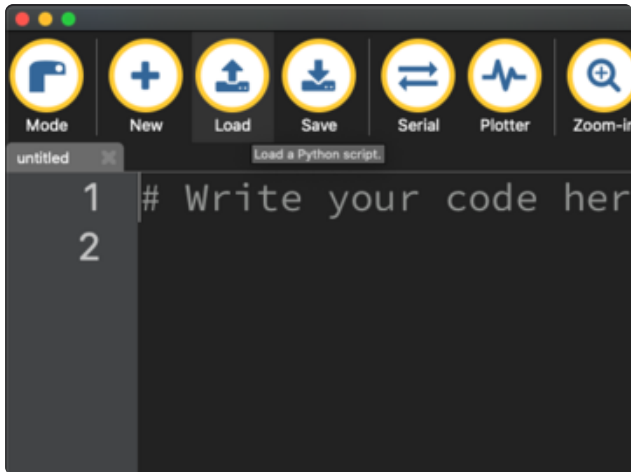
Creating and Editing Code

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

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

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

Creating Code



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

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

Copy and paste the following code into your editor:

```
import board
import digitalio
import time

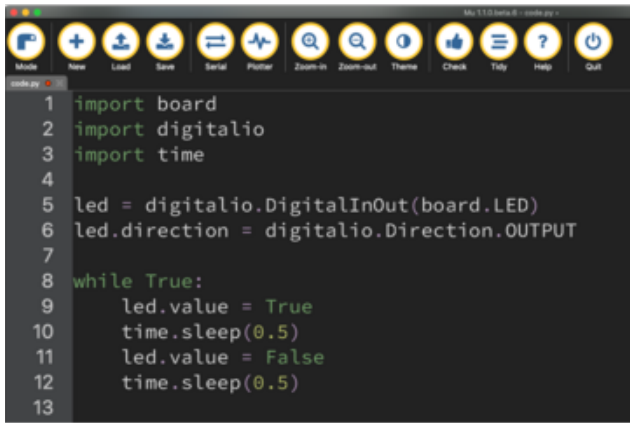
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

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

The KB2040, QT Py and the Trinkeys do not have a built-in little red LED! There is an addressable RGB NeoPixel LED. The above example will NOT work on the KB2040, QT Py or the Trinkeys!

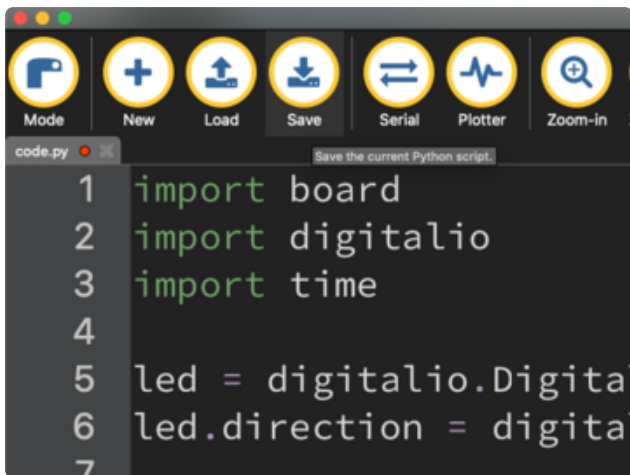
If you're using a KB2040, QT Py or a Trinkey, please download the [NeoPixel blink example \(\)](#).

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



```
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.LED)
6 led.direction = digitalio.Direction.OUTPUT
7
8 while True:
9     led.value = True
10    time.sleep(0.5)
11    led.value = False
12    time.sleep(0.5)
13
```

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



```
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.Digital
6 led.direction = digita
7
```

Save the code.py file on your CIRCUITPY drive.

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

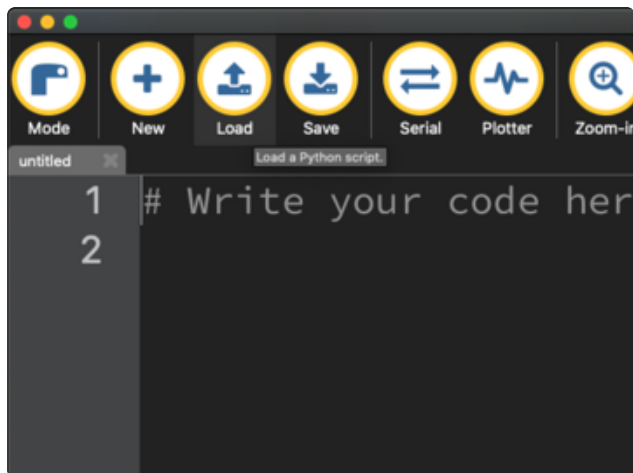
Congratulations, you've just run your first CircuitPython program!

On most boards you'll find a tiny red LED.

On the ItsyBitsy nRF52840, you'll find a tiny blue LED.

On QT Py M0, QT Py RP2040, and the Trinkey series, you will find only an RGB NeoPixel LED.

Editing Code



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

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

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

There's one warning before you continue...

Don't click reset or unplug your board!

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

There are a couple of ways to avoid filesystem corruption.

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

Check out the [Recommended Editors page \(\)](#) for details on different editing options.

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

2. Eject or Sync the Drive After Writing

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

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

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

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

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

Back to Editing Code...

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

```
import board
import digitalio
import time

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

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

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

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

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

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

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

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

Naming Your Program File

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

Connecting to the Serial Console

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

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

This line in your `code.py` would result in:

```
Hello, world!
```

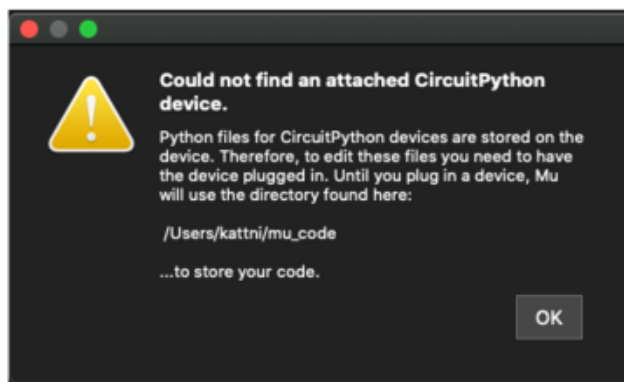
However, these print statements need somewhere to display. That's where the serial console comes in!

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

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

Are you using Mu?

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

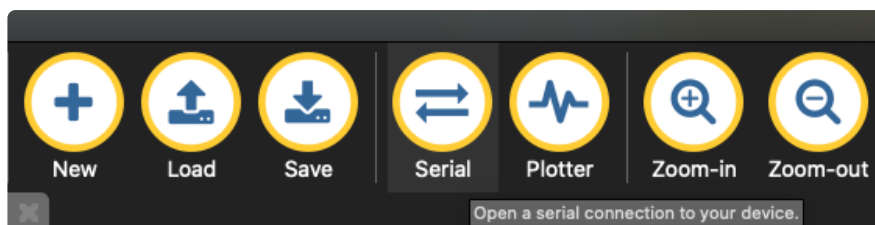


First, make sure your CircuitPython board is plugged in.

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

If you are using Windows 7, make sure you installed the drivers ([link](#)).

Once you've opened Mu with your board plugged in, look for the Serial button in the button bar and click it.



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

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

Code done running.

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

If nothing appears in the serial console, it may mean your code is done running or has no print statements in it. Click into the serial console part of Mu, and press CTRL+D to reload.

Serial Console Issues or Delays on Linux

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

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

```
sudo apt purge modemmanager
```

Setting Permissions on Linux

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



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

```
sudo adduser $USER dialout
```

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

Using Something Else?

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

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

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

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

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



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

Code done running.

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

Interacting with the Serial Console

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

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

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

```
import board
import digitalio
import time

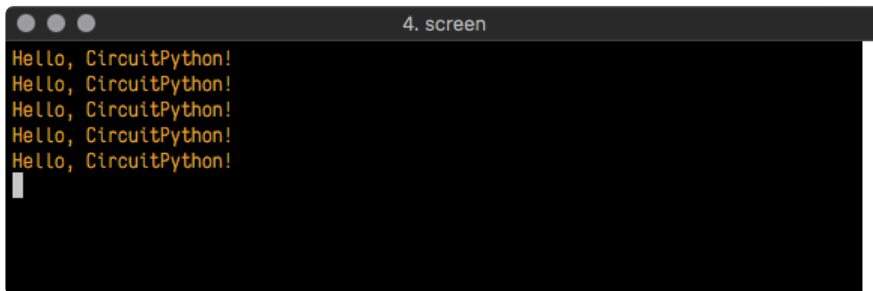
led = digitalio.DigitalInOut(board.LED)
```

```
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.



```
4. screen
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
```

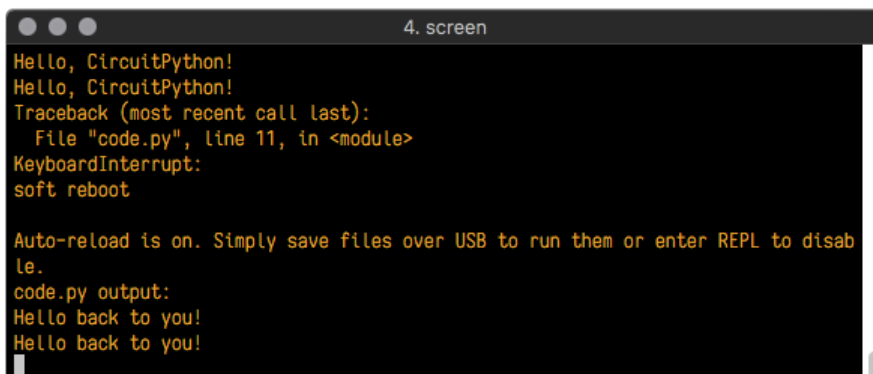
Excellent! Our print statement is showing up in our console! Try changing the printed text to something else.

```
import board
import digitalio
import time

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

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

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



```
4. screen
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 disable.
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 you can see how it is used.

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

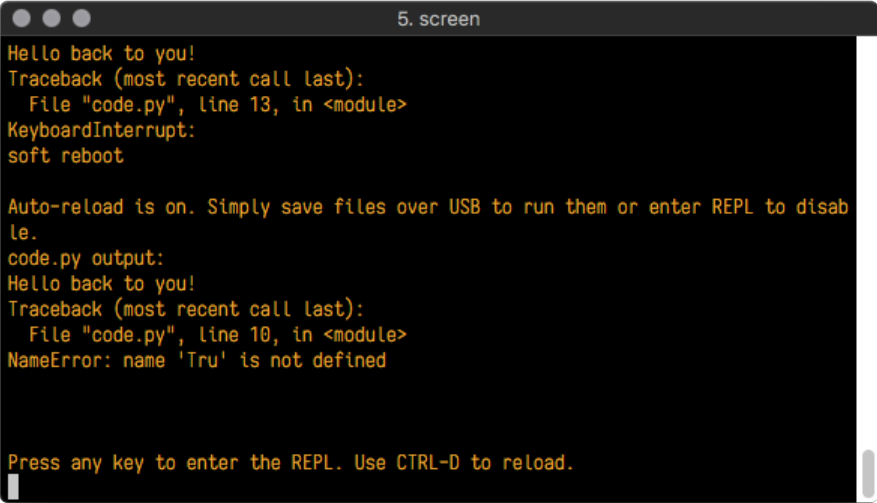
```
import board
import digitalio
import time

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

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

Save your file. You will notice that your red LED will stop blinking, and you may have a colored status LED blinking at you. This is because the code is no longer correct and can no longer run properly. You need to fix it!

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



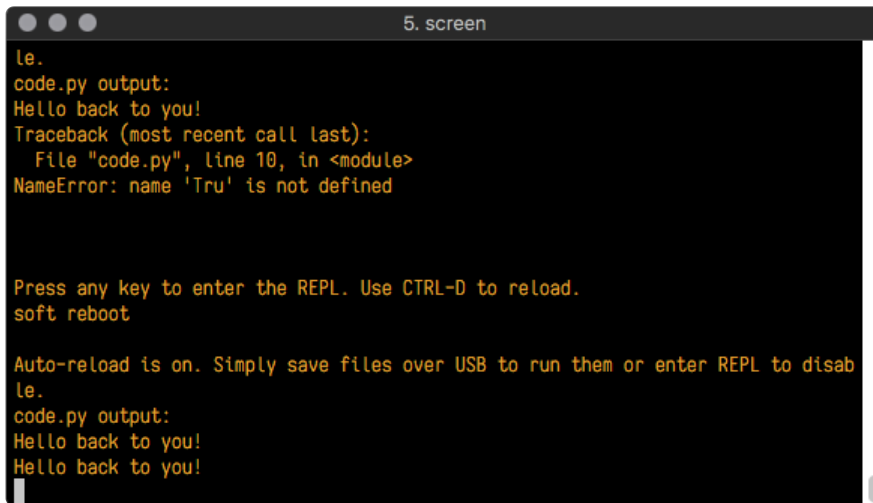
```
5. screen
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 13, in <module>
KeyboardInterrupt:
soft reboot

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

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

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

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

The REPL

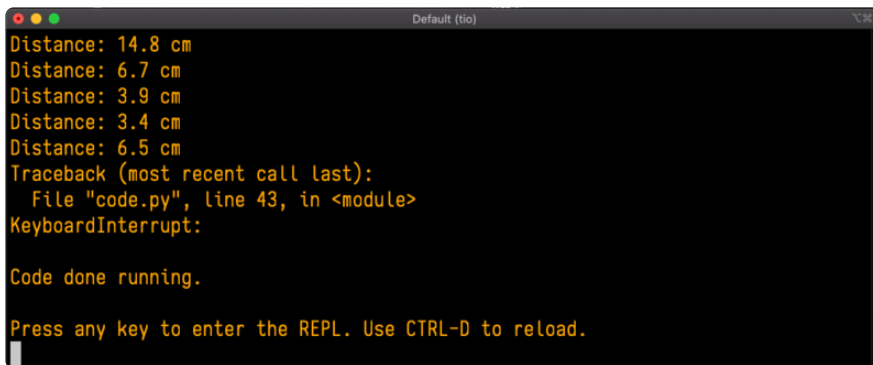
The other feature of the serial connection is the Read-Evaluate-Print-Loop, or REPL. The REPL allows you to enter individual lines of code and have them run immediately. It's really handy if you're running into trouble with a particular program and can't figure out why. It's interactive so it's great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

Entering the REPL

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

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

The **Traceback (most recent call last):** is telling you the last thing your board was doing before you pressed Ctrl + C and interrupted it. The **KeyboardInterrupt** is you pressing CTRL+C. This information can be handy when troubleshooting, but for now, don't worry about it. Just note that it is expected behavior.

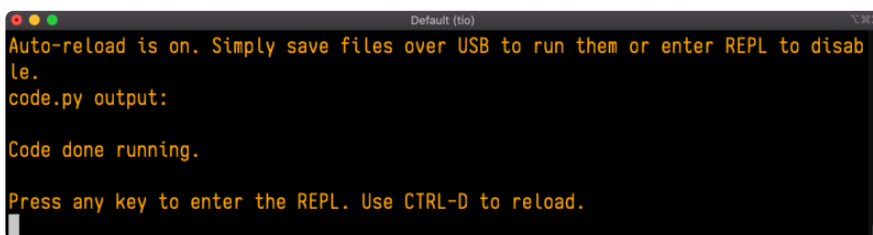


```
Distance: 14.8 cm
Distance: 6.7 cm
Distance: 3.9 cm
Distance: 3.4 cm
Distance: 6.5 cm
Traceback (most recent call last):
  File "code.py", line 43, in <module>
KeyboardInterrupt:

Code done running.

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

If your code.py file is empty or does not contain a loop, it will show an empty output and **Code done running.** There is no information about what your board was doing before you interrupted it because there is no code running.

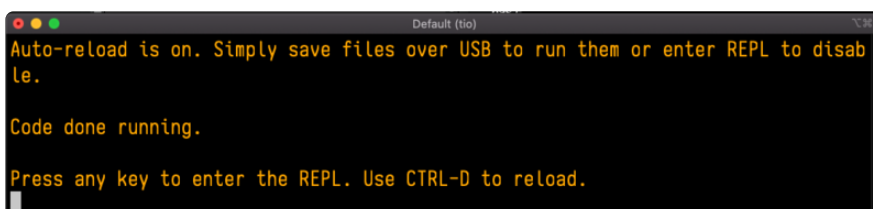


```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:

Code done running.

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

If you have no code.py on your CIRCUITPY drive, you will enter the REPL immediately after pressing CTRL+C. Again, there is no information about what your board was doing before you interrupted it because there is no code running.

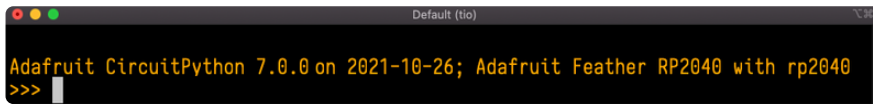


```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Code done running.

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

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



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

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

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



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

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

This is followed by the CircuitPython prompt.

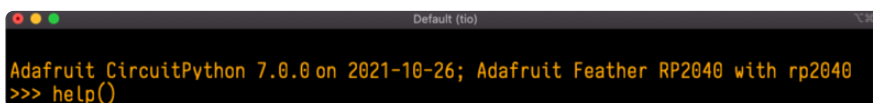


```
>>>
```

Interacting with the REPL

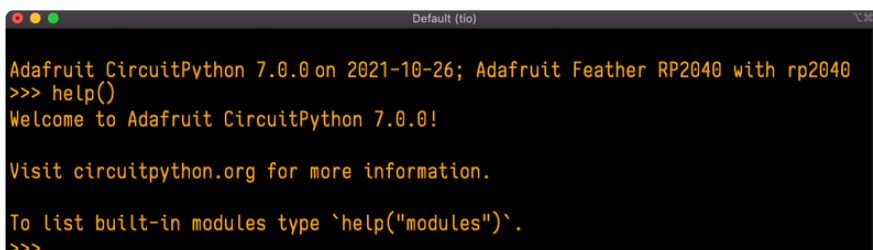
From this prompt you can run all sorts of commands and code. The first thing you'll do is run `help()`. This will tell you where to start exploring the REPL. To run code in the REPL, type it in next to the REPL prompt.

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



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

Then press enter. You should then see a message.



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

Visit circuitpython.org for more information.

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


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

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

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

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

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

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

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

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

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

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

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

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

Then press enter.

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

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

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

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

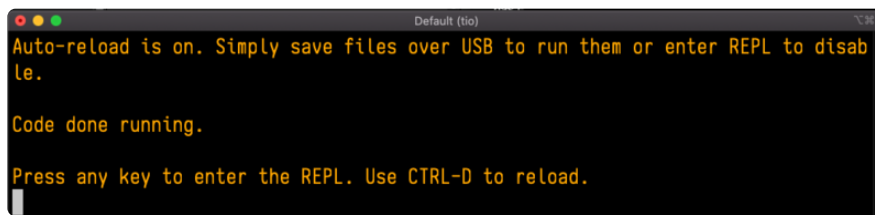
Try typing more into the REPL to see what happens!

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

Returning to the Serial Console

When you're ready to leave the REPL and return to the serial console, simply press CT RL+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!



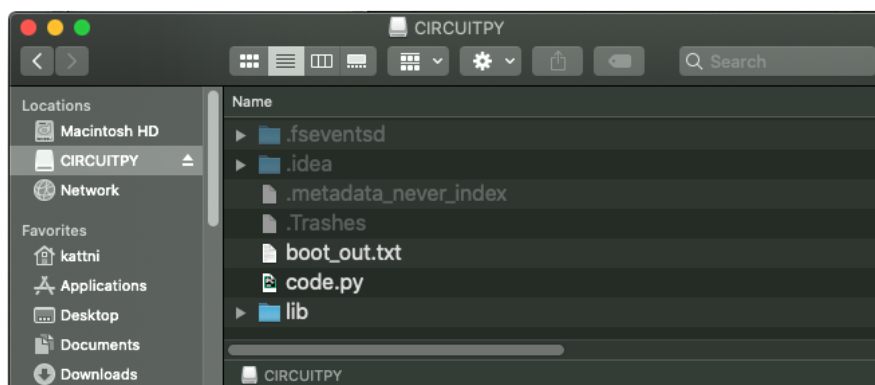
```
Default (tio)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
Code done running.
Press any key to enter the REPL. Use CTRL-D to reload.
```

CircuitPython Libraries

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

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

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



CircuitPython libraries work in the same way as regular Python modules so the [Python docs \(\)](#) are an excellent reference for how it all should work. In Python terms, you can

place our library files in the lib directory because it's part of the Python path by default.

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

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

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

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

The Adafruit Learn Guide Project Bundle

The quickest and easiest way to get going with a project from the Adafruit Learn System is by utilising the Project Bundle. Most guides now have a Download Project Bundle button available at the top of the full code example embed. This button downloads all the necessary files, including images, etc., to get the guide project up and running. Simply click, open the resulting zip, copy over the right files, and you're good to go!

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

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

🏠 > Circuit Playground Express: Piano in the Key of Lime > Piano in the Key of Lime



Piano in the Key of Lime

Now we'll take everything we learned and put it together!

Be sure to save your current code.py if you've changed anything you'd like to keep. Download the following file. Rename it to code.py and save it to your Circuit Playground Express.

[Download Project Bundle](#) [Copy Code](#)

```
# SPDX-FileCopyrightText: 2017 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

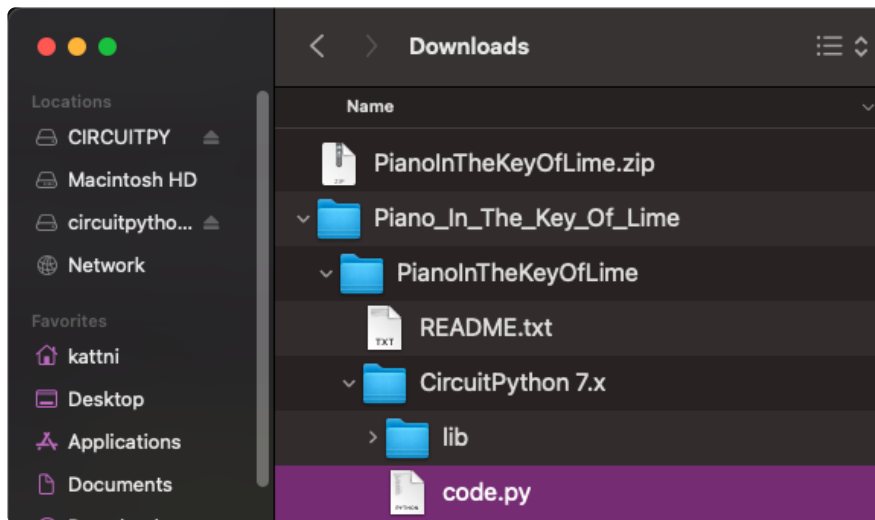
from adafruit_circuitplayground import cp

while True:
    if cp.switch:
        print("Slide switch off!")
        cp.pixels.fill((0, 0, 0))
```

Circuit Playground Express: Piano in the Key of Lime
By [Kattni Rembor](#)
Create a full scale tone piano using CircuitPython, capacitive touch and some cute little fruits.

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

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



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

When you open the zip, you'll find some nested directories. Navigate through them until you find what you need. You'll eventually find a directory for your CircuitPython

version (in this case, 7.x). In the version directory, you'll find the file and directory you need: code.py and lib/. Once you find the content you need, you can copy it all over to your CIRCUITPY drive, replacing any files already on the drive with the files from the freshly downloaded zip.

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

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

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

The Adafruit CircuitPython Library Bundle

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

Downloading the Adafruit CircuitPython Library Bundle

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

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

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

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

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

The CircuitPython Community Library Bundle

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

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

Downloading the CircuitPython Community Library Bundle

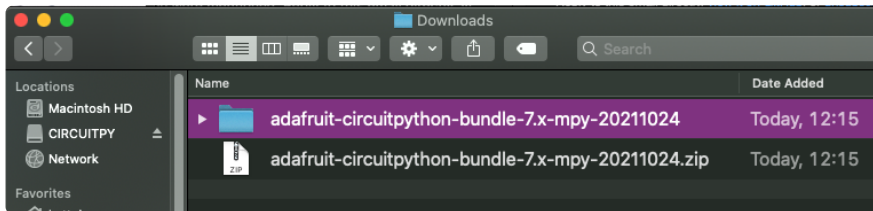
You can download the latest CircuitPython Community Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

[Click for the latest CircuitPython Community Library Bundle release](#)

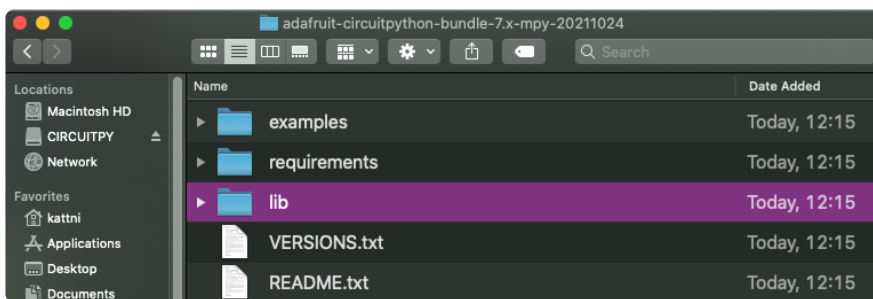
The link takes you to the latest release of the CircuitPython Community Library Bundle on GitHub. There are multiple versions of the bundle available. Download the bundle version that matches your CircuitPython firmware version. If you don't know the version, check the version info in `boot_out.txt`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

Understanding the Bundle

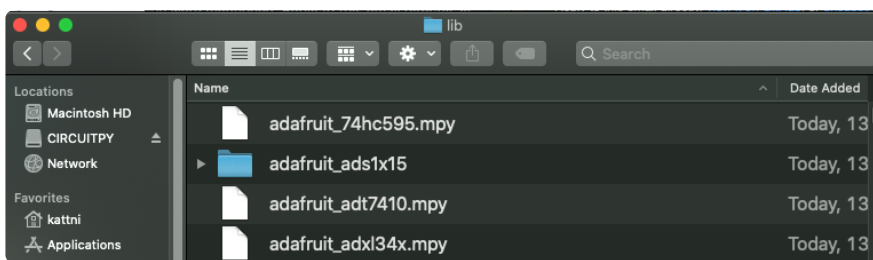
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



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



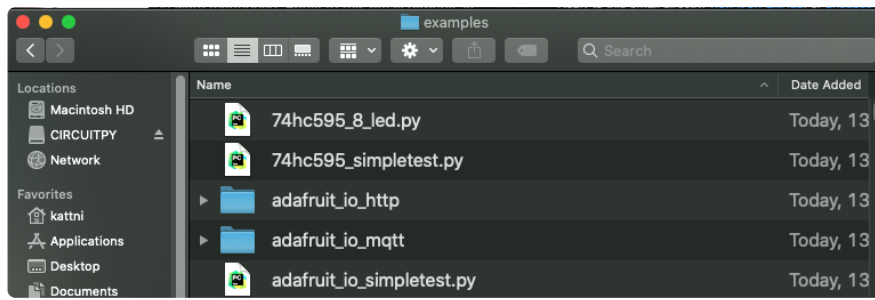
Now open the lib folder. When you open the folder, you'll see a large number of .mpy files, and folders.



Example Files

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

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



Copying Libraries to Your Board

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

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

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

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

Understanding Which Libraries to Install

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

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

- `import library_or_module`

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

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

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

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

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

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

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

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

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

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

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

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

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

The fifth one is `usb_hid`, and it is in the modules list, so it is built in. Often all of the built-in modules come first in the import list, but sometimes they don't! Don't assume that everything after the first library is also a library, and verify each import with the modules list to be sure. Otherwise, you'll search the bundle and come up empty!

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

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

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

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

Example: `ImportError` Due to Missing Library

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

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

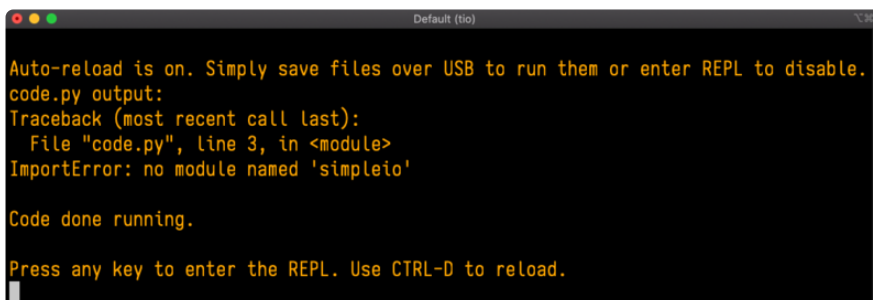
Let's use a modified version of the Blink example.

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.LED)

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

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

A screenshot of a terminal window titled "Default (tio)". The terminal output shows a successful execution of a Python script followed by an error. The error message is: "Traceback (most recent call last): File 'code.py', line 3, in <module> ImportError: no module named 'simpleio'". The terminal also displays "Code done running." and "Press any key to enter the REPL. Use CTRL-D to reload." at the bottom.

```
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 3, in <module>
ImportError: no module named 'simpleio'

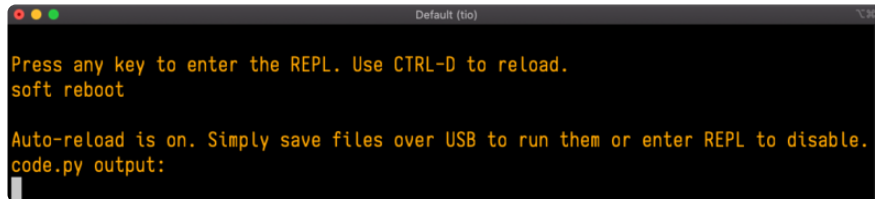
Code done running.

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

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

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

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



```
Default (tio)
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 disable.
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 an M0 non-Express board such as Trinket M0, Gemma M0, QT Py M0, or one of the M0 Trinkeys, you'll want to follow the same steps in the example above to install libraries as you need them. Remember, you don't need to wait for an `ImportError` if you know what library you added to your code. Open the library bundle you downloaded, find the library you need, and drag it to the lib folder on your CIRCUITPY drive.

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

Updating CircuitPython Libraries and Examples

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

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

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

CircUp CLI Tool

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

CircuitPython Pins and Modules

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

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

CircuitPython Pins

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

import board

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

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

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

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

```
>>> 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? \(\)](#) section below.

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

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

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

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

I2C, SPI, and UART

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

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

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

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

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

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

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

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


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

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

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

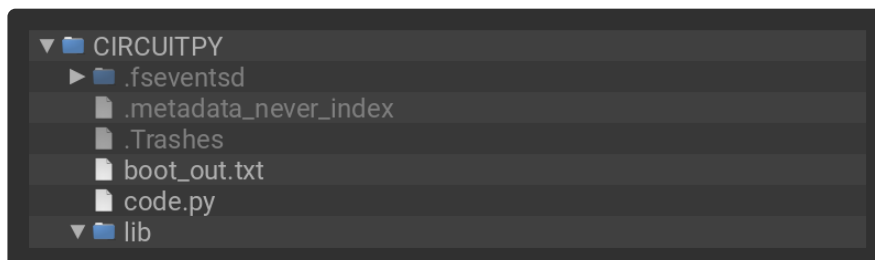
What Are All the Available Names?

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

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

In the example below, click the Download Project Bundle button below to download the necessary libraries and the `code.py` file in a zip file. Extract the contents of the zip file, open the directory `CircuitPython_Essentials/Pin_Map_Script/` and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

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



```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries  
#
```

```
# SPDX-License-Identifier: MIT

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

board_pins = []
for pin in dir(microcontroller.pin):
    if isinstance(getattr(microcontroller.pin, pin), microcontroller.Pin):
        pins = []
        for alias in dir(board):
            if getattr(board, alias) is getattr(microcontroller.pin, pin):
                pins.append("board.{}".format(alias))
        if len(pins) > 0:
            board_pins.append(" ".join(pins))
for pins in sorted(board_pins):
    print(pins)
```

Here is the result when this script is run on QT Py:

```
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 \(\)](#) and the Python-like modules included [here \(\)](#). However, not every module is available for every board due to size constraints or hardware limitations. How do you find out what modules are available for your board?

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

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

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

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

Windows 7 and 8.1

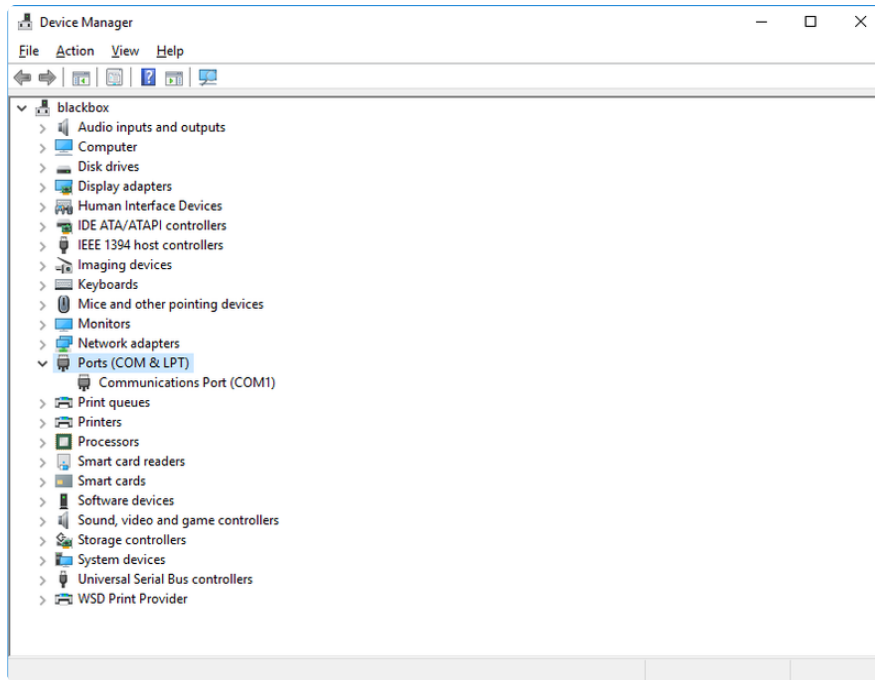
If you're using Windows 7 (or 8 or 8.1), you'll need to install drivers. See the [Windows 7 and 8.1 Drivers page \(\)](#) for details. You will not need to install drivers on Mac, Linux or Windows 10.

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

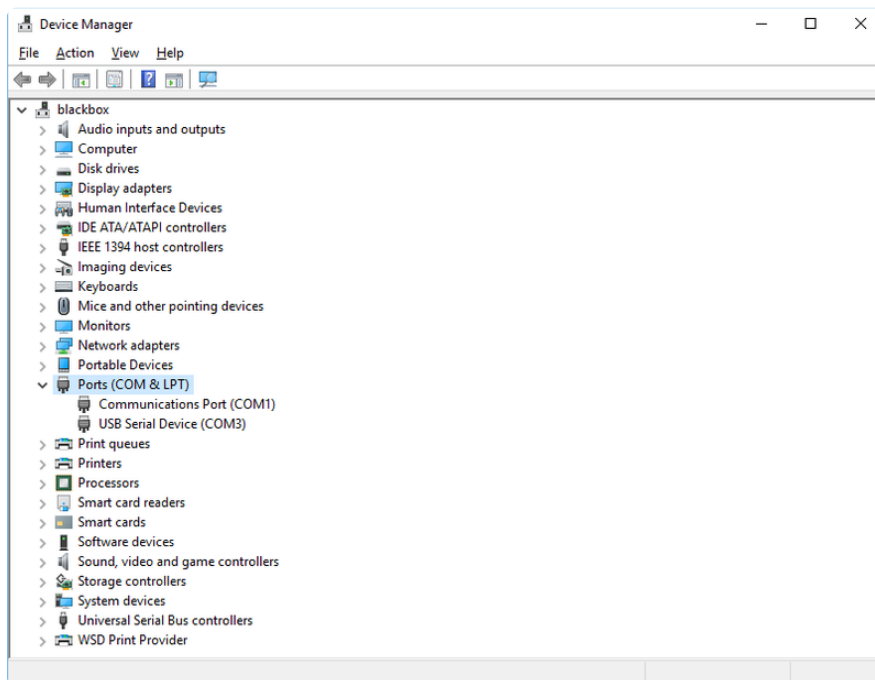
What's the COM?

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

You'll use Windows Device Manager to determine which port the board is using. The easiest way to determine which port the board is using is to first check without the board plugged in. Open Device Manager. Click on Ports (COM & LPT). You should find something already in that list with (COM#) after it where # is a number.



Now plug in your board. The Device Manager list will refresh and a new item will appear under Ports (COM & LPT). You'll find a different (COM#) after this item in the list.



Sometimes the item will refer to the name of the board. Other times it may be called something like USB Serial Device, as seen in the image above. Either way, there is a new (COM#) following the name. This is the port your board is using.

Install Putty

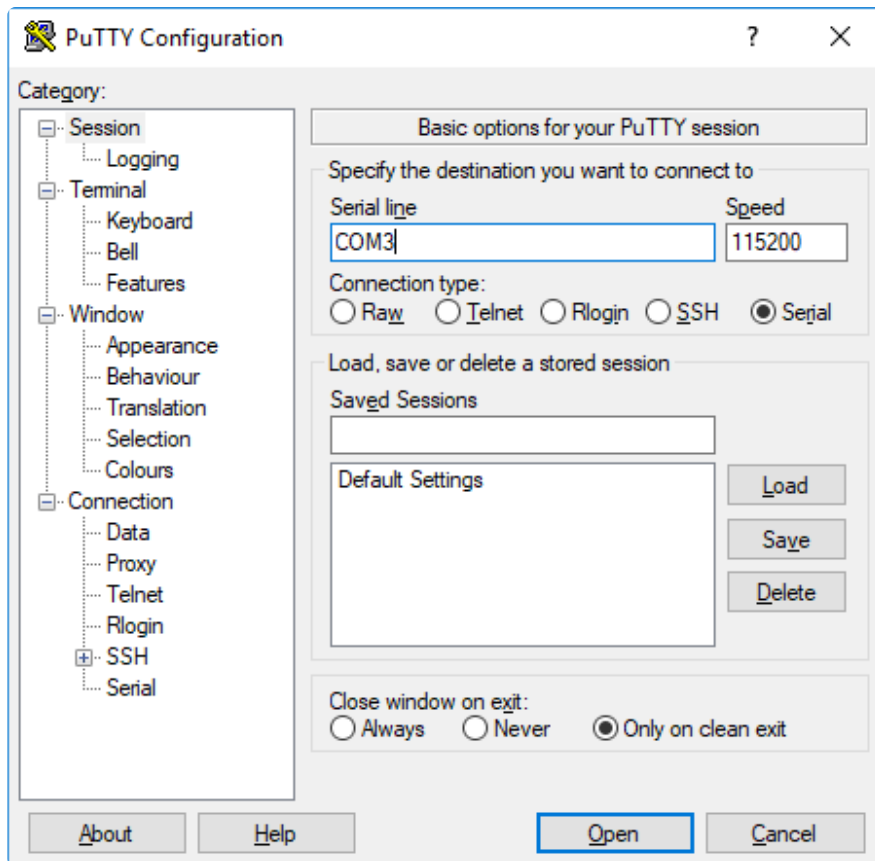
If you're using Windows, you'll need to download a terminal program. You're going to use PuTTY.

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

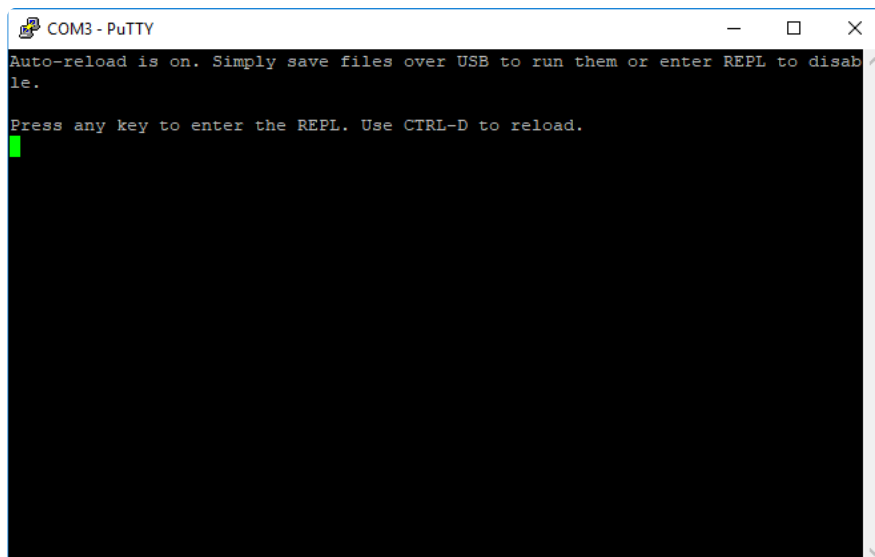
Now you need to open PuTTY.

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

If you want to save those settings for later, use the options under Load, save or delete a stored session. Enter a name in the box under Saved Sessions, and click the Save button on the right.



Once your settings are entered, you're ready to connect to the serial console. Click "Open" at the bottom of the window. A new window will open.



If no code is running, the window will either be blank or will look like the window above. Now you're ready to see the results of your code.

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

Advanced Serial Console on Mac

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

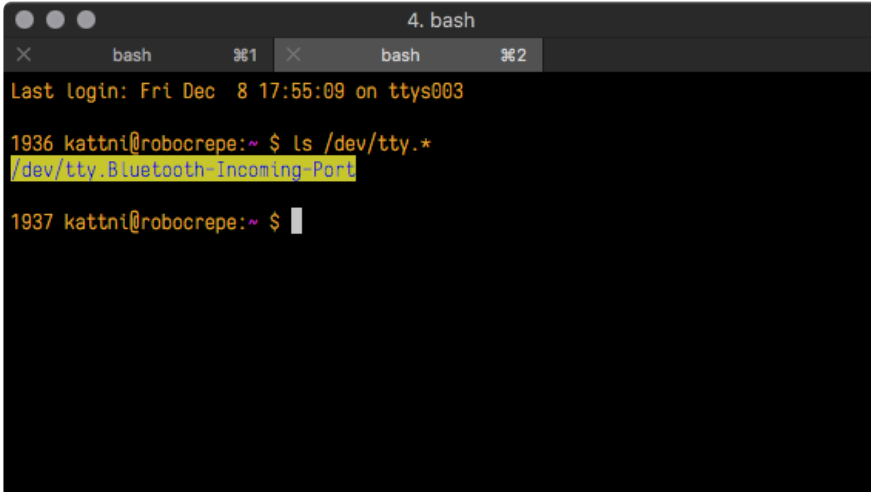
What's the Port?

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

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

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

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

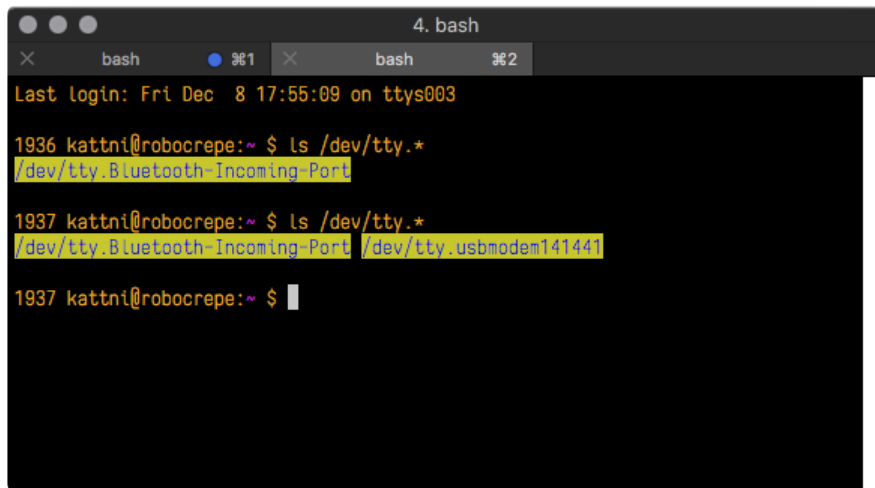


```
4. bash
bash  %1  bash  %2
Last login: Fri Dec  8 17:55:09 on ttys003
1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Blueetooth-Incoming-Port
1937 kattni@robocrepe:~ $
```

Now, plug your board. In Terminal, type:

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

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



```
4. bash
bash  #1 bash  #2
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:~ $
```

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

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

Connect with screen

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

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

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

```
4. bash
bash %1 bash %2
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
```

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!

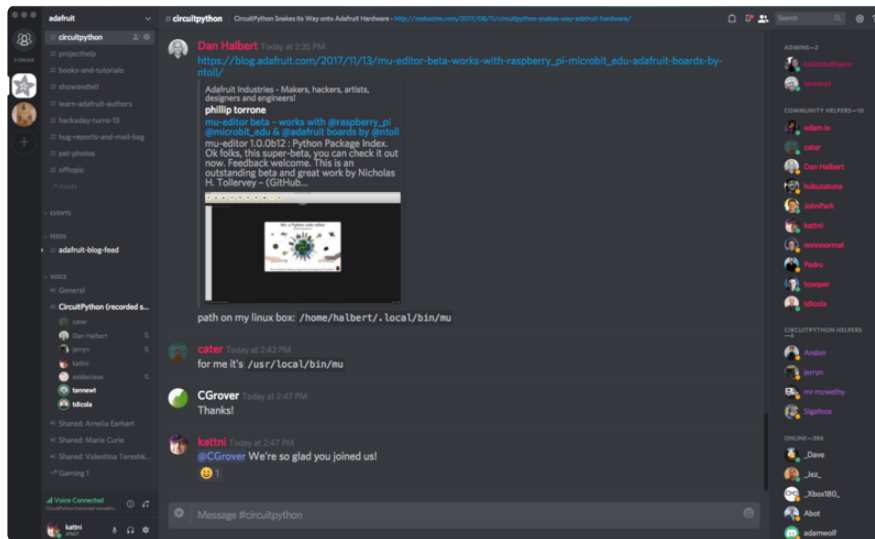
Welcome to the Community!



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

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

Adafruit Discord



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

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

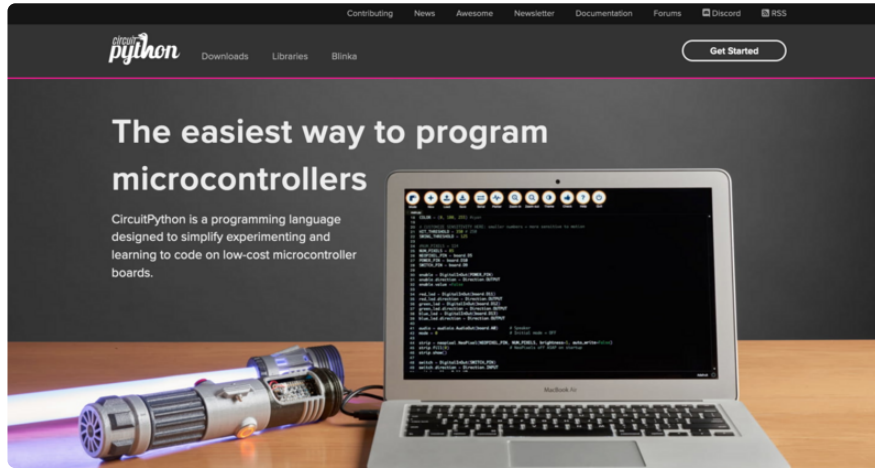
The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. Your contributions are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

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

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

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

CircuitPython.org



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

Contributing

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

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

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

Have an idea for a new driver or library? [File an issue on the CircuitPython repo!](#)

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

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

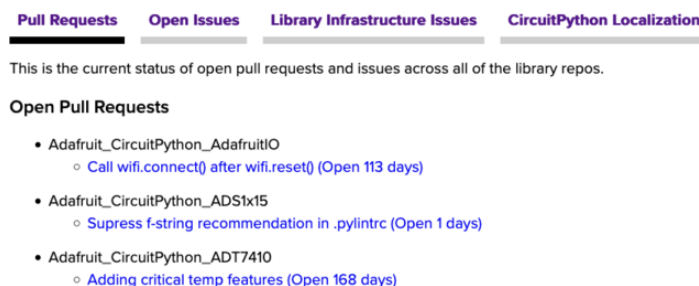
Current Status for Tue, Nov 02, 2021

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

Now, a look at the different options.

Pull Requests

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



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

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

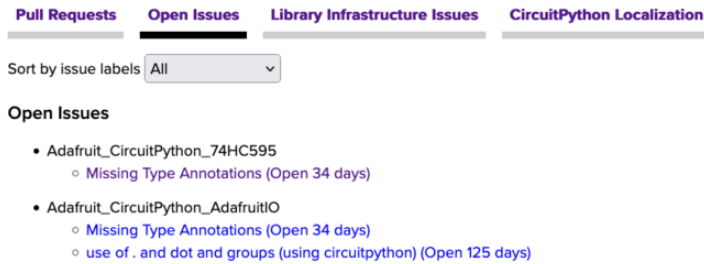
Open Pull Requests

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

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

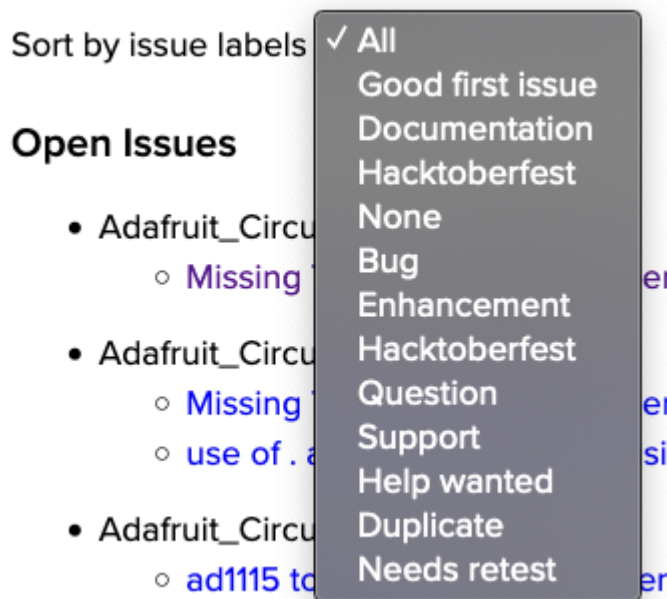
Open Issues

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



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

If you're not sure where to start, you can search the issues by label. Labels are applied to issues to make the goal easier to identify at a first glance, or to indicate the difficulty level of the issue. Click on the dropdown next to "Sort by issue labels" to see the list of available labels, and click on one to choose it.



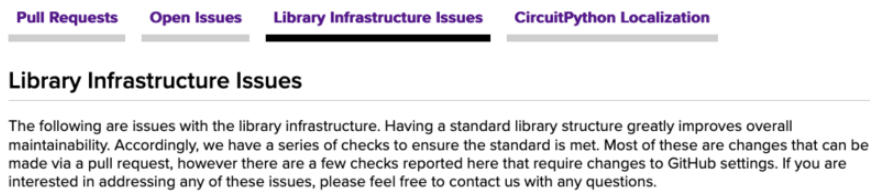
If you're new to everything, new to contributing to open source, or new to contributing to the CircuitPython project, you can choose "Good first issue". Issues with that label are well defined, with a finite scope, and are intended to be easy for someone new to figure out.

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

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

Library Infrastructure Issues

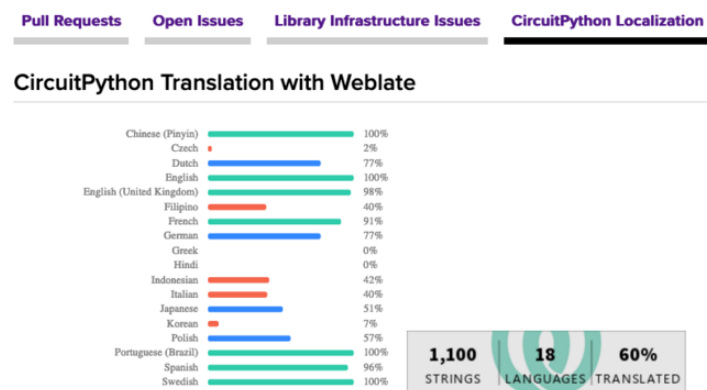
The third tab you'll find is a list of library infrastructure issues.



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

CircuitPython Localization

The fourth tab you'll find is the CircuitPython Localization tab.

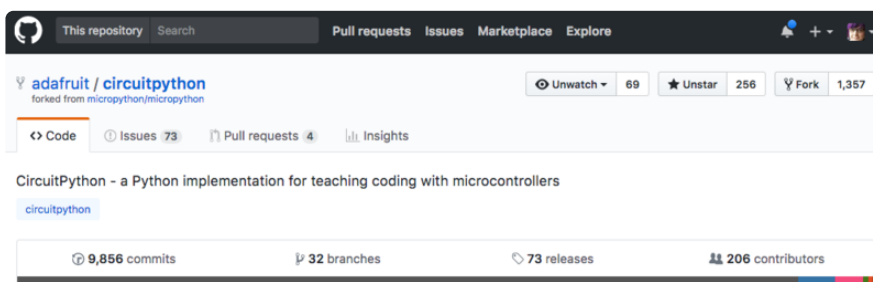


If you speak another language, you can help translate CircuitPython! The translations apply to informational and error messages that are within the CircuitPython core. It means that folks who do not speak English have the opportunity to have these messages shown to them in their own language when using CircuitPython. This is

incredibly important to provide the best experience possible for all users. CircuitPython uses Weblate to translate, which makes it much simpler to contribute translations. You will still need to know some CircuitPython-specific practices and a few basics about coding strings, but as with any CircuitPython contributions, folks are there to help.

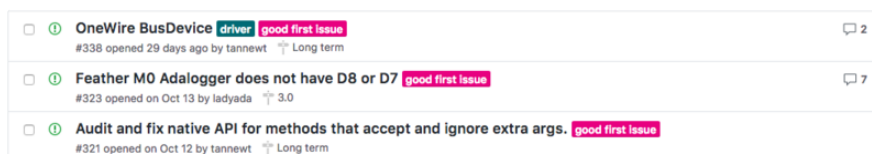
Regardless of your skill level, or how you want to contribute to the CircuitPython project, there is an opportunity available. The [Contributing page \(\)](#) is an excellent place to start!

Adafruit GitHub



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

If you're new to GitHub or programming in general, there are great opportunities for you. For the CircuitPython core, head over to the CircuitPython repository on GitHub, click on "[Issues \(\)](#)", and you'll find a list that includes issues labeled "[good first issue \(\)](#)". For the libraries, head over to the [Contributing page Issues list \(\)](#), and use the drop down menu to search for "[good first issue \(\)](#)". These issues are things that have been identified as something that someone with any level of experience can help with. These issues include options like updating documentation, providing feedback, and fixing simple bugs. If you need help getting started with GitHub, there is an excellent guide on [Contributing to CircuitPython with Git and GitHub \(\)](#).



Already experienced and looking for a challenge? Checkout the rest of either issues list and you'll find plenty of ways to contribute. You'll find all sorts of things, from new

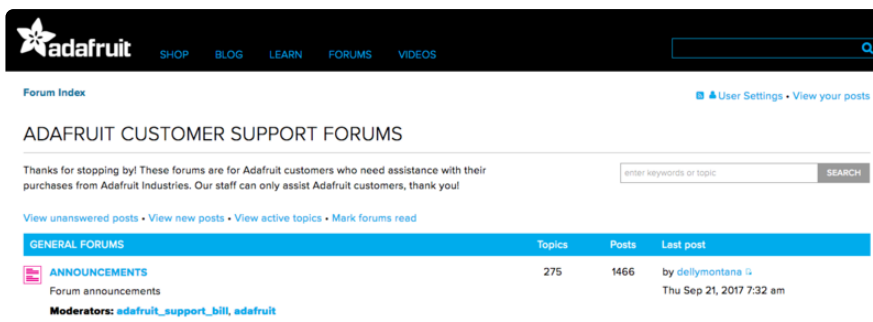
driver requests, to library bugs, to core module updates. There's plenty of opportunities for everyone at any level!

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

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

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

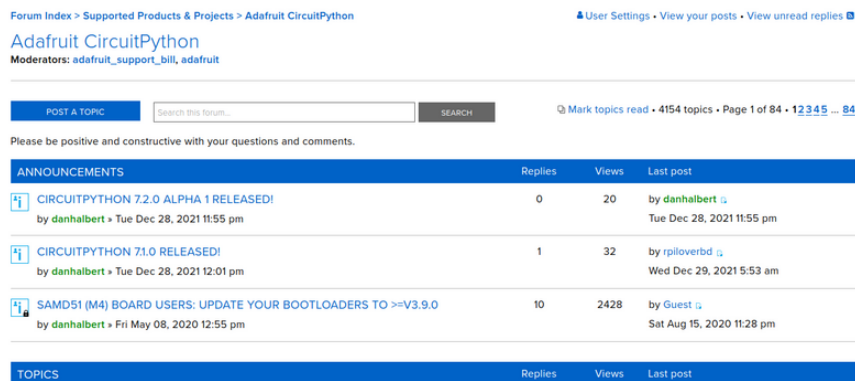
Adafruit Forums



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

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

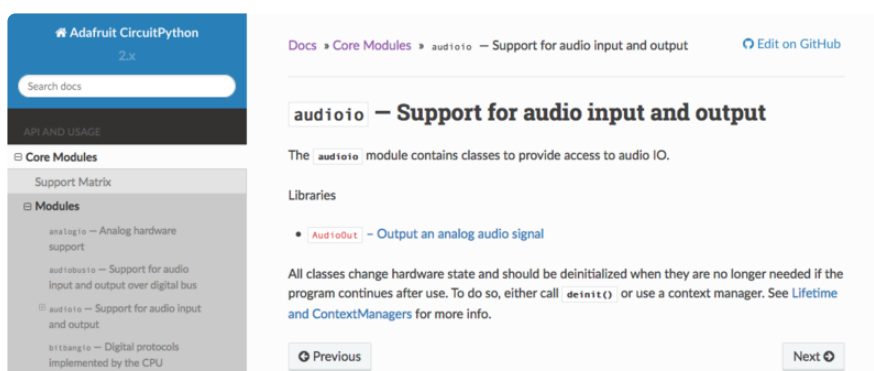
There are forum categories that cover all kinds of topics, including everything Adafruit. The [Adafruit CircuitPython \(\)](#) category under "Supported Products & Projects" is the best place to post your CircuitPython questions.



Be sure to include the steps you took to get to where you are. If it involves wiring, post a picture! If your code is giving you trouble, include your code in your post! These are great ways to make sure that there's enough information to help you with your issue.

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

Read the Docs



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

Here is blinky:

```
import time
import digitalio
import board

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

Frequently Asked Questions

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

What are some common acronyms to know?

CP or CPy = [CircuitPython \(\)](#)

CPC = [Circuit Playground Classic \(\)](#) (does not run CircuitPython)

CPX = [Circuit Playground Express \(\)](#)

CPB = [Circuit Playground Bluefruit \(\)](#)

Using Older Versions

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

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

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

- [2.x bundle \(\)](#)
- [3.x bundle \(\)](#)
- [4.x bundle \(\)](#)

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

Python Arithmetic

Does CircuitPython support floating-point numbers?

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

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

Does CircuitPython support long integers, like regular Python?

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

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

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

Wireless Connectivity

How do I connect to the Internet with CircuitPython?

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

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

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

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

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

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

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

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

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

Are there other ways to communicate by radio with CircuitPython?

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

Asyncio and Interrupts

Is there asyncio support in CircuitPython?

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

Does CircuitPython support interrupts?

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

Status RGB LED

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

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

Memory Issues

What is a MemoryError?

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

What do I do when I encounter a MemoryError?

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

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

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

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

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

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

How can I create my own .mpy files?

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

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

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

How do I check how much memory I have free?

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

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

Unsupported Hardware

Is ESP8266 or ESP32 supported in CircuitPython? Why not?

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

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

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

Does Feather M0 support WINC1500?

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

Can AVR's such as ATmega328 or ATmega2560 run CircuitPython?

No.

Troubleshooting

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

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

Always Run the Latest Version of CircuitPython and Libraries

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

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

As new versions of CircuitPython are released, Adafruit will stop providing the previous bundles as automatically created downloads on the Adafruit CircuitPython Library Bundle repo. If you must continue to use an earlier version, you can still

download the appropriate version of `mpy-cross` from the particular release of CircuitPython on the CircuitPython repo and create your own compatible .mpy library files. However, it is best to update to the latest for both CircuitPython and the library bundle.

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

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

Bootloader (boardnameBOOT) Drive Not Present

You may have a different board.

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

MakeCode

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

MacOS

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

Windows 10

Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 with the driver package installed? You don't need to install this

package on Windows 10 for most Adafruit boards. The old version (v1.5) can interfere with recognizing your device. Go to Settings -> Apps and uninstall all the "Adafruit" driver programs.

Windows 7 or 8.1

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

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

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

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

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

Windows Explorer Locks Up When Accessing boardnameBOOT Drive

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

- AIDA64: to fix, stop the program. This problem has been reported to AIDA64. They acquired hardware to test, and released a beta version that fixes the problem. This may have been incorporated into the latest release. Please let us know in the forums if you test this.
- Hard Disk Sentinel

- Kaspersky anti-virus: To fix, you may need to disable Kaspersky completely. Disabling some aspects of Kaspersky does not always solve the problem. This problem has been reported to Kaspersky.
- ESET NOD32 anti-virus: There have been problems with at least version 9.0.386.0, solved by uninstallation.

Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied

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

CIRCUITPY Drive Does Not Appear or Disappears Quickly

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

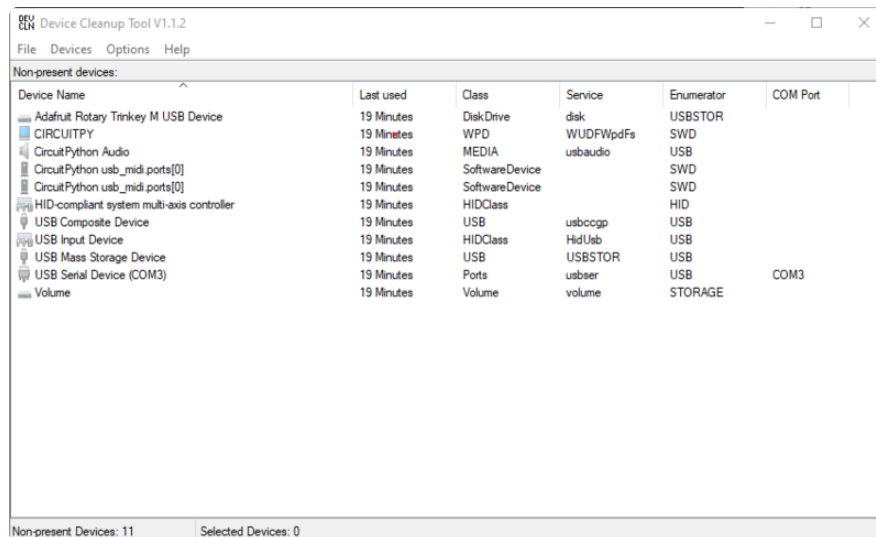
Norton anti-virus can interfere with CIRCUITPY. A user has reported this problem on Windows 7. The user turned off both Smart Firewall and Auto Protect, and CIRCUITPY then appeared.

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

Device Errors or Problems on Windows

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

If not, try cleaning up your USB devices. Use [Uwe Sieber's Device Cleanup Tool \(\)](#) (on that page, scroll down to "Device Cleanup Tool"). Download and unzip the tool. Unplug all the boards and other USB devices you want to clean up. Run the tool as Administrator. You will see a listing like this, probably with many more devices. It is listing all the USB devices that are not currently attached.



Select all the devices you want to remove, and then press Delete. It is usually safe just to select everything. Any device that is removed will get a fresh install when you plug it in. Using the Device Cleanup Tool also discards all the COM port assignments for the unplugged boards. If you have used many Arduino and CircuitPython boards, you have probably seen higher and higher COM port numbers used, seemingly without end. This will fix that problem.

Serial Console in Mu Not Displaying Anything

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

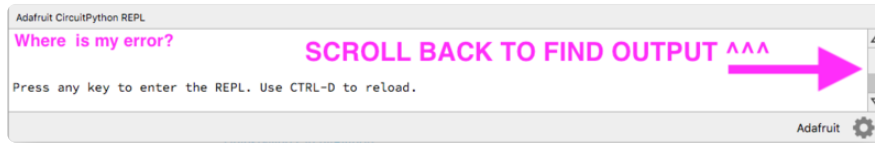
Depending on the size of your screen or Mu window, when you open the serial console, the serial console panel may be very small. This can be a problem. A basic CircuitPython error takes 10 lines to display!

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

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

More complex errors take even more lines!

Therefore, if your serial console panel is five lines tall or less, you may only see blank lines or blank lines followed by **Press any key to enter the REPL. Use CTRL-D to reload.** . If this is the case, you need to either mouse over the top of the panel to utilise the option to resize the serial panel, or use the scrollbar on the right side to scroll up and find your message.



This applies to any kind of serial output whether it be error messages or print statements. So before you start trying to debug your problem on the hardware side, be sure to check that you haven't simply missed the serial messages due to serial output panel height.

code.py Restarts Constantly

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

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

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

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

```
import supervisor
supervisor.disable_autoreload()
```

CircuitPython RGB Status Light

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

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

CircuitPython 7.0.0 and Later

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

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

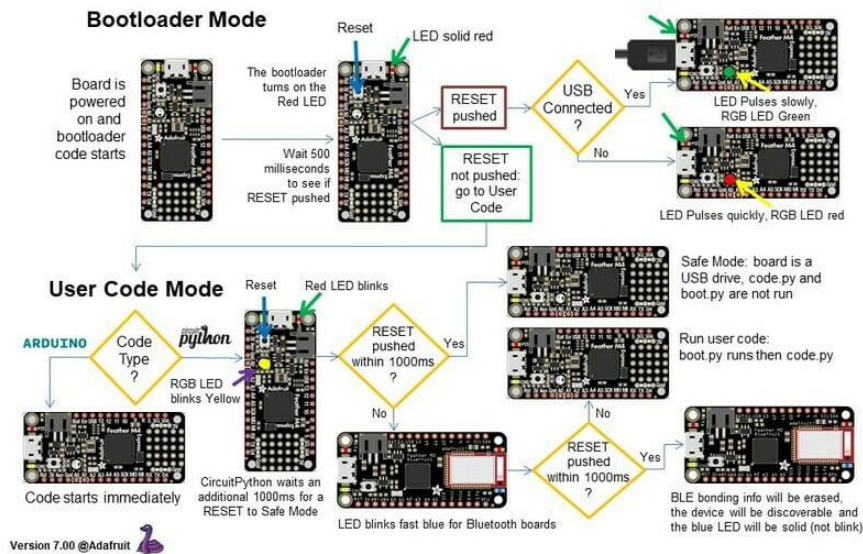
Once started, CircuitPython will blink a pattern every 5 seconds when no user code is running to indicate why the code stopped:

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

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

The CircuitPython Boot Sequence

Version 7.0 and later



CircuitPython 6.3.0 and earlier

Here's what the colors and blinking mean:

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

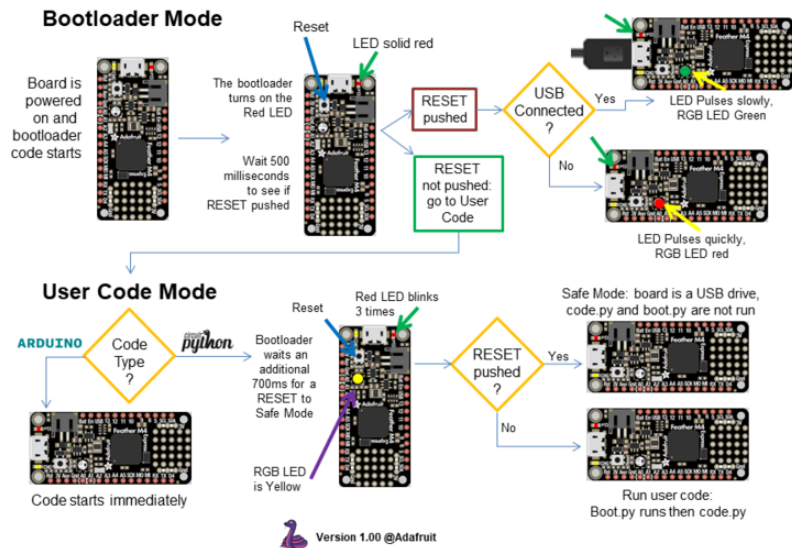
Colors with multiple flashes following indicate a Python exception and then indicate the line number of the error. The color of the first flash indicates the type of error:

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

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

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

The CircuitPython Boot Sequence



Serial console showing **ValueError: Incompatible .mpy file**

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

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

CIRCUITPY Drive Issues

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

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

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

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

Safe Mode

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

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

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

Entering Safe Mode in CircuitPython 7.x and Later

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

Entering Safe Mode in CircuitPython 6.x

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

In Safe Mode

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

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

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

```
Auto-reload is off.  
Running in safe mode! Not running saved code.  
  
CircuitPython is in safe mode because you pressed the reset button during boot.  
Press again to exit safe mode.  
  
Press any key to enter the REPL. Use CTRL-D to reload.
```

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

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

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

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

To erase CIRCUITPY: `storage.erase_filesystem()`

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

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

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

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

Erase CIRCUITPY Without Access to the REPL

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

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

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

For the specific boards listed below:

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

1. Download the correct erase file:

Circuit Playground Express

Feather M0 Express



2. Double-click the reset button on the board to bring up the boardnameBOOT drive.
3. Drag the erase .uf2 file to the boardnameBOOT drive.
4. The status LED will turn yellow or blue, indicating the erase has started.
5. After approximately 15 seconds, the status LED will light up green. On the NeoTrellis M4 this is the first NeoPixel on the grid
6. Double-click the reset button on the board to bring up the boardnameBOOT drive.
7. [Drag the appropriate latest release of CircuitPython \(\)](#) .uf2 file to the boardnameBOOT drive.

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

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

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

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

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

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

1. Download the erase file:

SAMD21 non-Express Boards

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

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

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

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

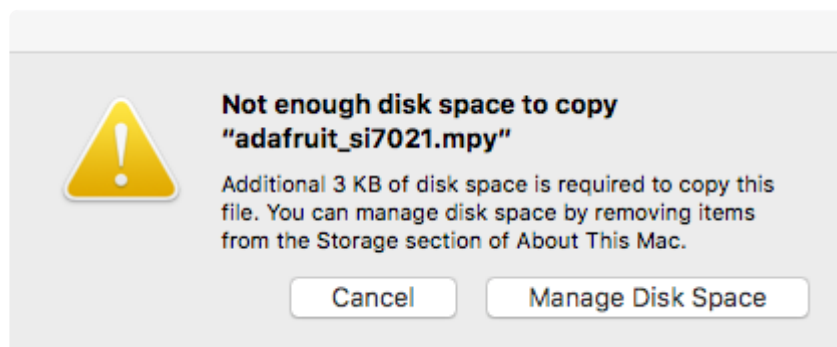
Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that do not have a UF2 bootloader include the Feather M0 Basic Proto, Feather Adalogger, or the Arduino Zero.

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

Running Out of File Space on SAMD21 Non-Express Boards

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

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



Delete something!

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

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

Use tabs

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

On MacOS?

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

Prevent & Remove MacOS Hidden Files

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

```
ls -l /Volumes
```

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

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

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

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

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

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

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

Copy Files on MacOS Without Creating Hidden Files

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

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

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

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

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

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

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

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


Other MacOS Space-Saving Tips

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

```
Default (-bash)
Last login: Thu Oct 28 17:19:15 on ttys008

7039 kattni@robocrepe:~ $ cd /Volumes/

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

7041 kattni@robocrepe:Volumes $
```

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

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

7042 kattni@robocrepe:Volumes $
```

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

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

7043 kattni@robocrepe:Volumes $
```

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

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

7044 kattni@robocrepe:Volumes $
```

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

Device Locked Up or Boot Looping

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

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

For most devices:

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

For ESP32-S2 based devices:

Press and release the reset button, then press and release the boot button about 3/4 of a second later.

Refer to the diagrams above for boot sequence details.

"Uninstalling" CircuitPython

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

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

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

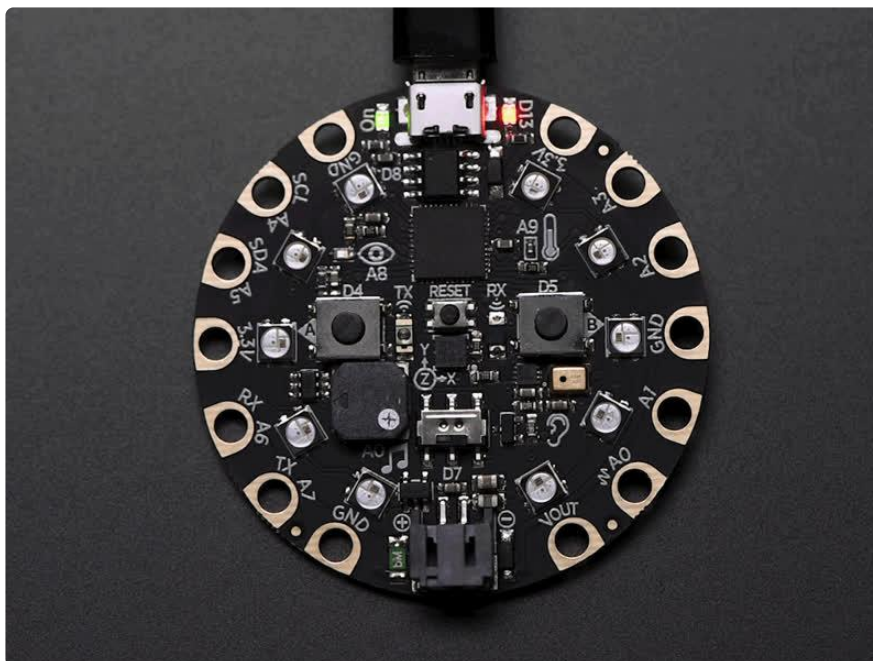
Backup Your Code

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

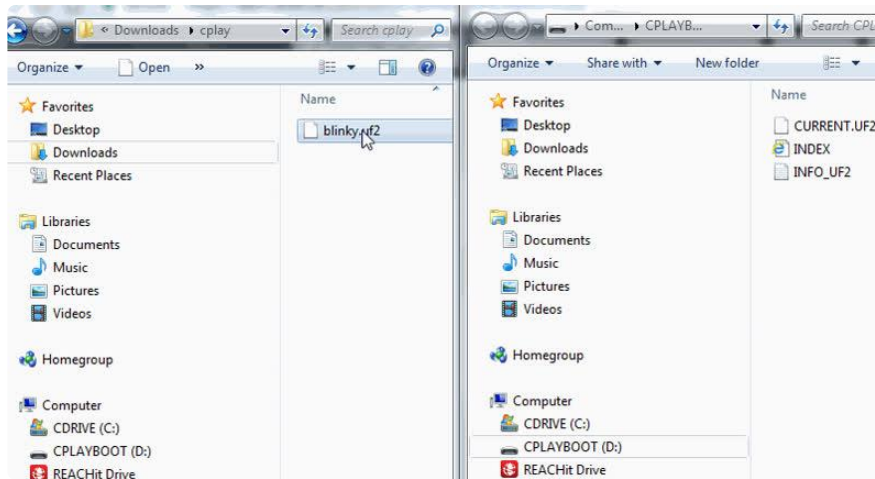
Moving Circuit Playground Express to MakeCode

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

Now double-click your CircuitPython board until you see the onboard LED(s) turn green and the ...BOOT directory shows up.



Then find the downloaded MakeCode .uf2 file and drag it to the CPLAYBOOT drive.



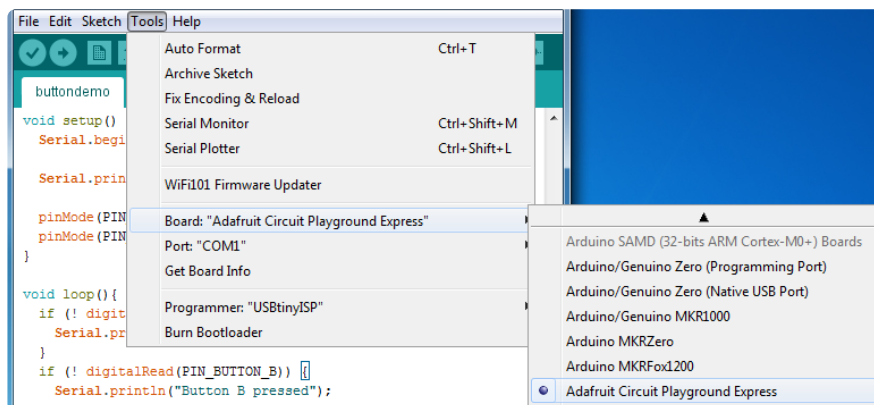
Your MakeCode is now running and CircuitPython has been removed. Going forward you only have to single click the reset button to get to CPLAYBOOT. This is an idiosyncrasy of MakeCode.

Moving to Arduino

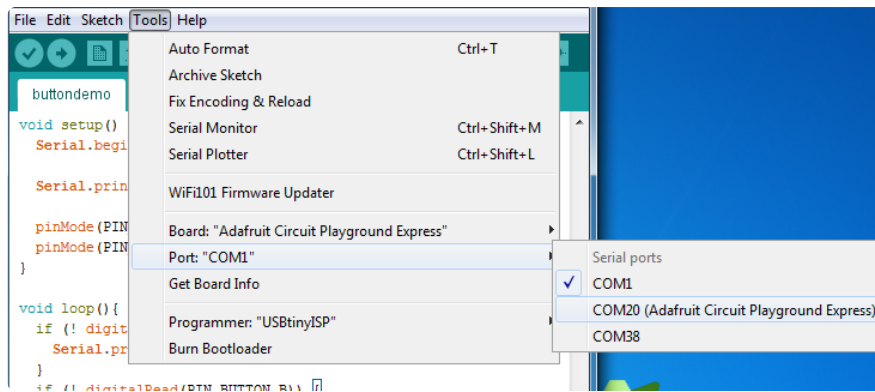
If you want to use Arduino instead, you just use the Arduino IDE to load an Arduino program. Here's an example of uploading a simple "Blink" Arduino program, but you don't have to use this particular program.

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

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



Select the correct matching Port:



Create a new simple Blink sketch example:

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

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

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

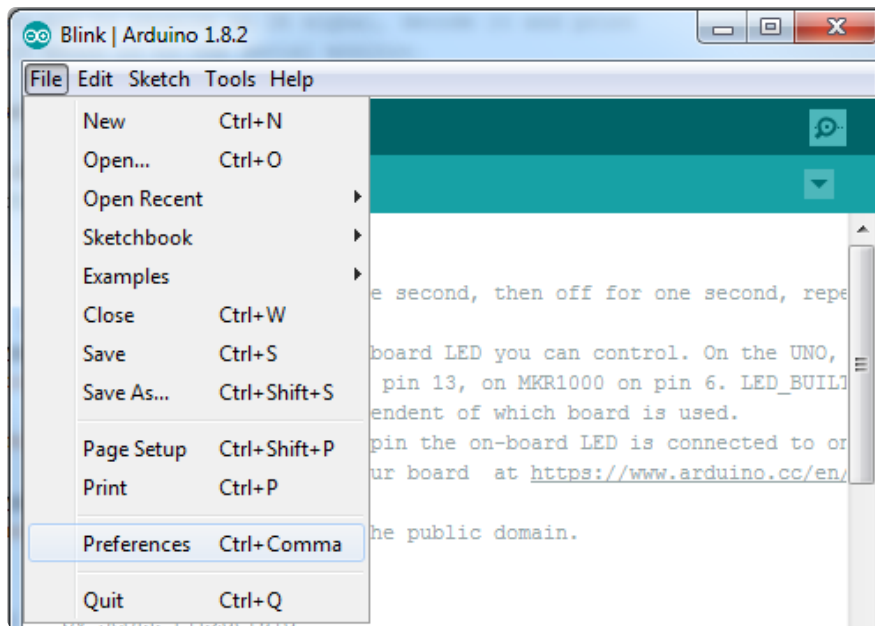
Once Blink is uploaded you should no longer need to double-click to enter bootloader mode. Arduino will automatically reset when you upload.

Arduino IDE Setup

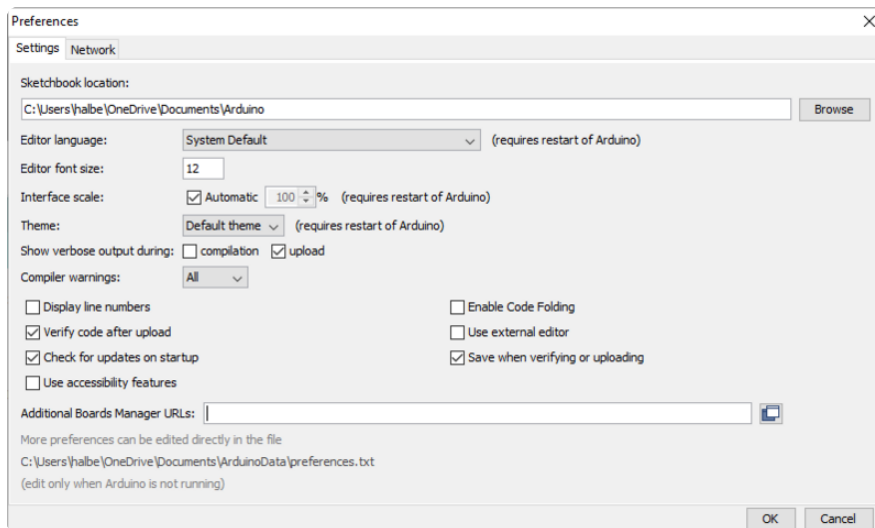
The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using version 1.8 or higher for this guide

[Arduino IDE Download](#)

After you have downloaded and installed the latest version of Arduino IDE, you will need to start the IDE and navigate to the Preferences menu. You can access it from the File menu in Windows or Linux, or the Arduino menu on OS X.



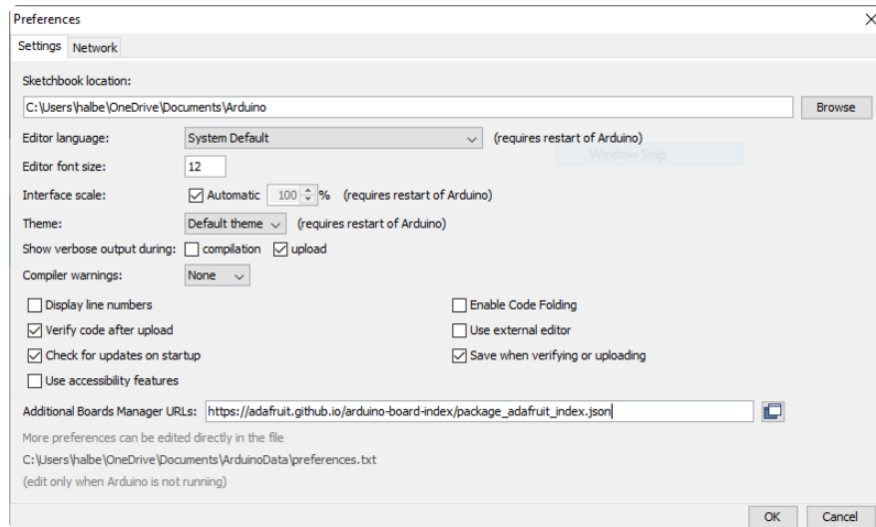
A dialog will pop up just like the one shown below.



We will be adding a URL to the new Additional Boards Manager URLs option. The list of URLs is comma separated, and you will only have to add each URL once. New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

To find the most up to date list of URLs you can add, you can visit the list of [third party board URLs on the Arduino IDE wiki](#) (). We will only need to add one URL to the IDE in this example, but you can add multiple URLs by separating them with commas. Copy and paste the link below into the Additional Boards Manager URLs option in the Arduino IDE preferences.

https://adafruit.github.io/arduino-board-index/package_adafruit_index.json



Here's a short description of each of the Adafruit supplied packages that will be available in the Board Manager when you add the URL:

- Adafruit AVR Boards - Includes support for Flora, Gemma, Feather 32u4, ItsyBitsy 32u4, Trinket, & Trinket Pro.
- Adafruit SAMD Boards - Includes support for Feather M0 and M4, Metro M0 and M4, ItsyBitsy M0 and M4, Circuit Playground Express, Gemma M0 and Trinket M0
- Arduino Leonardo & Micro MIDI-USB - This adds MIDI over USB support for the Flora, Feather 32u4, Micro and Leonardo using the [arcore project](#) ().

If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

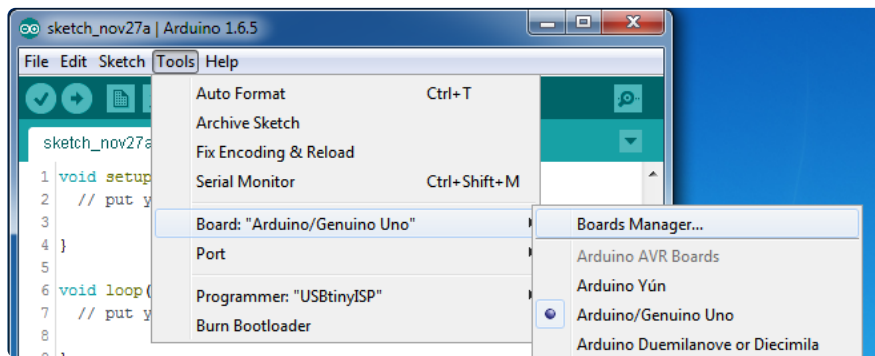
Once done click OK to save the new preference settings. Next we will look at installing boards with the Board Manager.

Now continue to the next step to actually install the board support package!

Using with Arduino IDE

The Feather/Metro/Gemma/QTPy/Trinket M0 and M4 use an ATSAM21 or ATSAM51 chip, and you can pretty easily get it working with the Arduino IDE. Most libraries (including the popular ones like NeoPixels and display) will work with the M0 and M4, especially devices & sensors that use I2C or SPI.

Now that you have added the appropriate URLs to the Arduino IDE preferences in the previous page, you can open the Boards Manager by navigating to the Tools->Board menu.



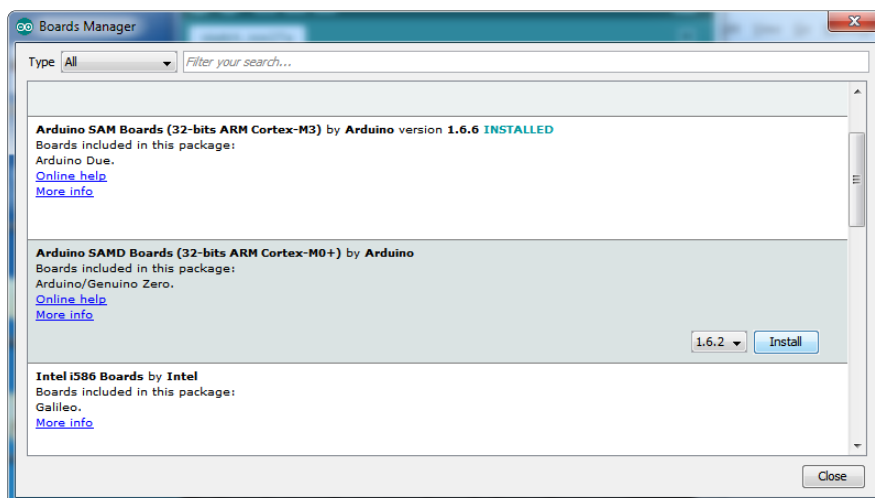
Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select All. You will then be able to select and install the boards supplied by the URLs added to the preferences.

Remember you need SETUP the Arduino IDE to support our board packages - see the previous page on how to add adafruit's URL to the preferences

Install SAMD Support

First up, install the latest Arduino SAMD Boards (version 1.6.11 or later)

You can type Arduino SAMD in the top search bar, then when you see the entry, click Install

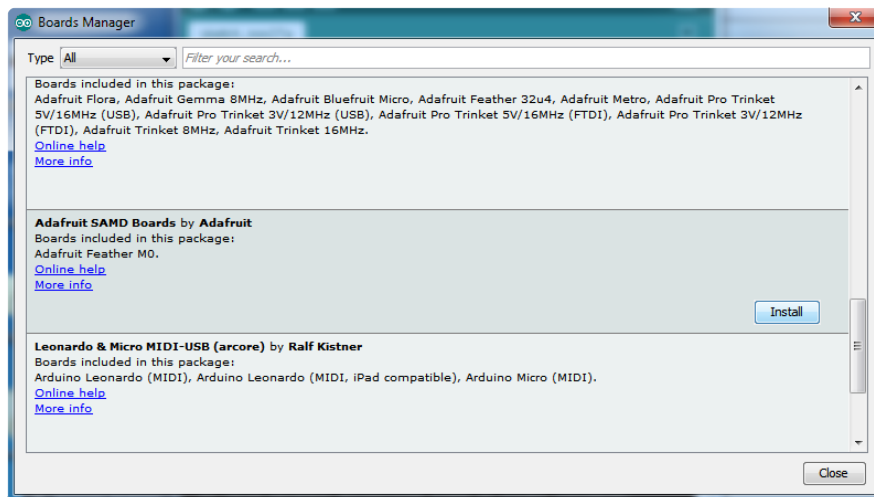


Install Adafruit SAMD

Next you can install the Adafruit SAMD package to add the board file definitions

Make sure you have Type All selected to the left of the Filter your search... box

You can type Adafruit SAMD in the top search bar, then when you see the entry, click Install



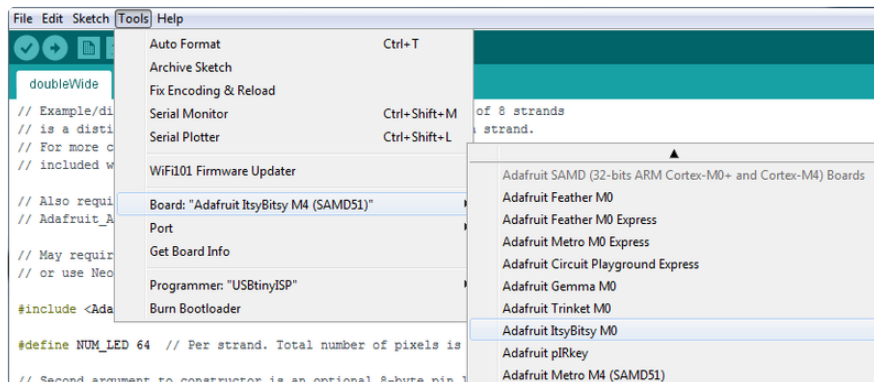
Even though in theory you don't need to - I recommend rebooting the IDE

Quit and reopen the Arduino IDE to ensure that all of the boards are properly installed. You should now be able to select and upload to the new boards listed in the Tools->Board menu.

Select the matching board, the current options are:

- Feather M0 (for use with any Feather M0 other than the Express)
- Feather M0 Express
- Metro M0 Express
- Circuit Playground Express
- Gemma M0
- Trinket M0
- QT Py M0
- ItsyBitsy M0
- Hallowing M0
- Crickit M0 (this is for direct programming of the Crickit, which is probably not what you want! For advanced hacking only)
- Metro M4 Express

- Grand Central M4 Express
- ItsyBitsy M4 Express
- Feather M4 Express
- Trellis M4 Express
- PyPortal M4
- PyPortal M4 Titano
- PyBadge M4 Express
- Metro M4 Airlift Lite
- PyGamer M4 Express
- MONSTER M4SK
- Hallowing M4
- MatrixPortal M4
- BLM Badge



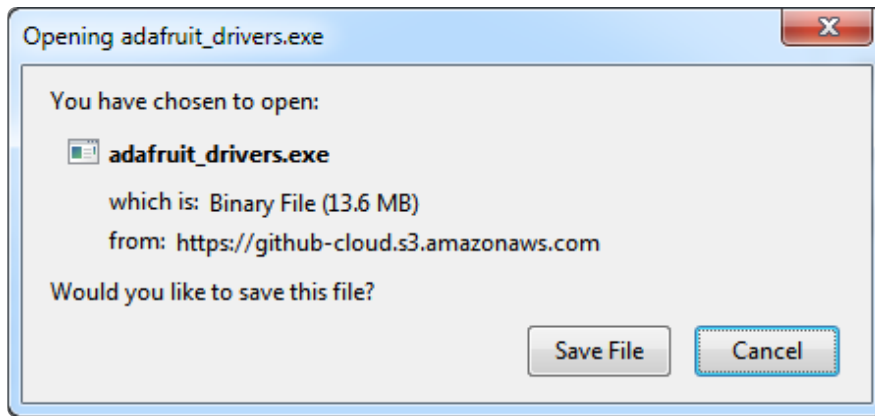
Install Drivers (Windows 7 & 8 Only)

When you plug in the board, you'll need to possibly install a driver

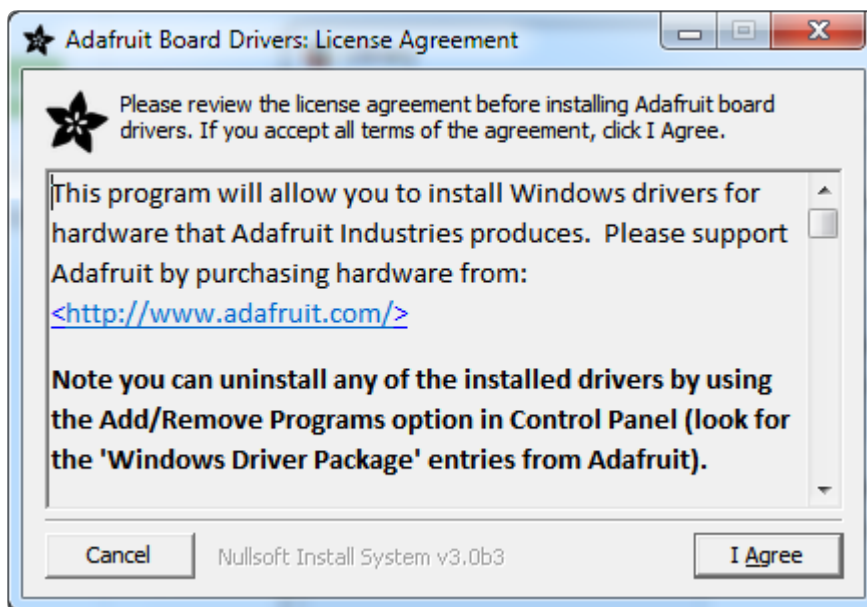
Click below to download our Driver Installer

[Download Latest Adafruit Drivers package](#)

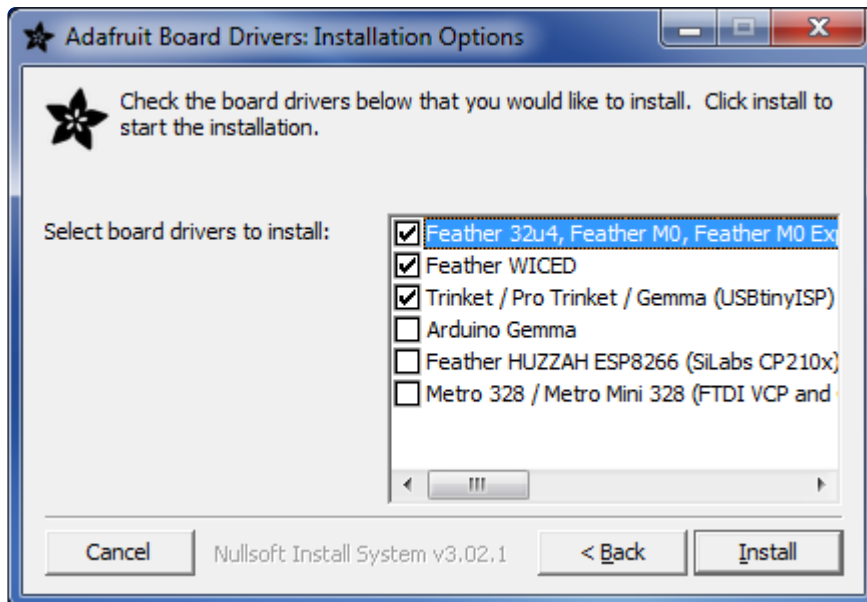
Download and run the installer



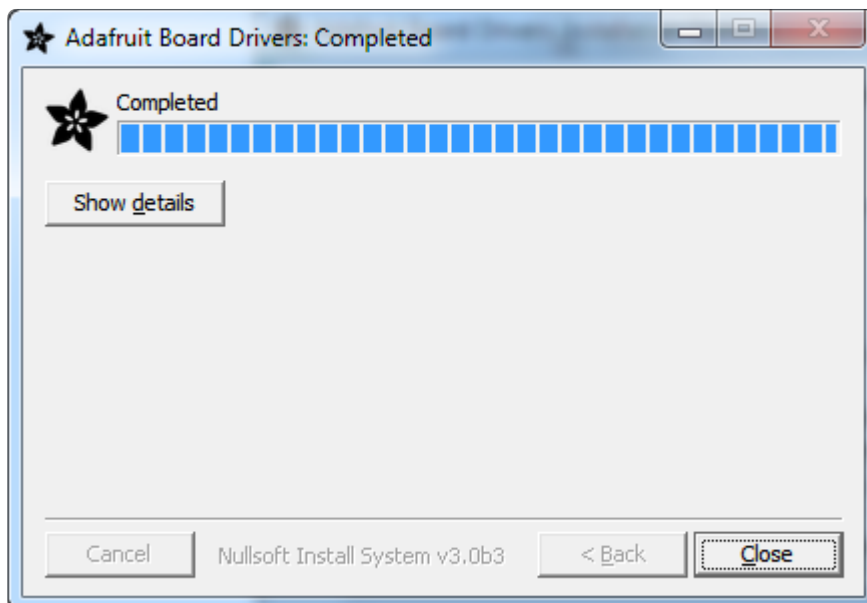
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



Select which drivers you want to install, the defaults will set you up with just about every Adafruit board!



Click Install to do the installin'

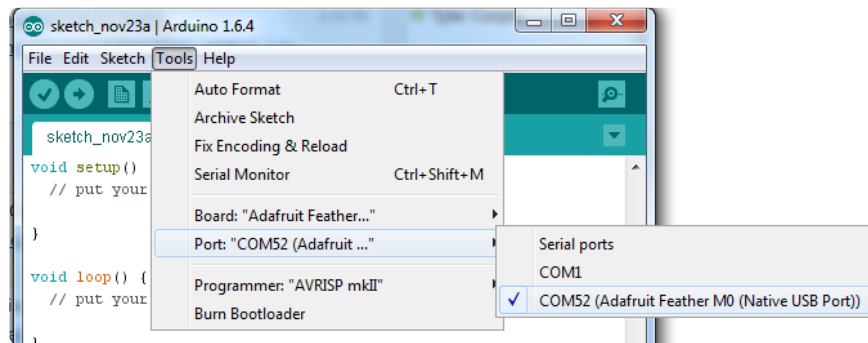


Blink

Now you can upload your first blink sketch!

Plug in the M0 or M4 board, and wait for it to be recognized by the OS (just takes a few seconds). It will create a serial/COM port, you can now select it from the drop-down, it'll even be 'indicated' as Trinket/Gemma/Metro/Feather/ItsyBitsy/Trellis!

Please note, the QT Py and Trellis M4 Express are two of our very few boards that does not have an onboard pin 13 LED so you can follow this section to practice uploading but you wont see an LED blink!



Now load up the Blink example

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

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

And click upload! That's it, you will be able to see the LED blink rate change as you adapt the delay() calls.

If you are having issues, make sure you selected the matching Board in the menu that matches the hardware you have in your hand.

Successful Upload

If you have a successful upload, you'll get a bunch of red text that tells you that the device was found and it was programmed, verified & reset

```
Done uploading.
Write 11024 bytes to flash (173 pages)

[=====          ] 36% (64/173 pages)
[=====          ] 73% (128/173 pages)
[=====          ] 100% (173/173 pages)

done in 0.097 seconds

Verify 11024 bytes of flash with checksum.

Verify successful

done in 0.049 seconds

CPU reset.
```

6 Adafruit Feather M0 (Native USB Port) on COM54

After uploading, you may see a message saying "Disk Not Ejected Properly" about the ...BOOT drive. You can ignore that message: it's an artifact of how the bootloader and uploading work.

Compilation Issues

If you get an alert that looks like

Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-non-eabi-g++"

Make sure you have installed the Arduino SAMD boards package, you need both Arduino & Adafruit SAMD board packages

```
Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-non-eabi-g++" Copy error messages

{runtime.tools.arm-none-eabi-gcc.path}/bin/arm-none-eabi-g++ -mcpu=cortex-m0plus -mth
max-inline-insns-single=500 -fno-rtti -fno-exceptions -MMD -DF_CPU=48000000L -DARDUIN
-DUSB_MANUFACTURER="Adafruit" -DUSB_PRODUCT="Feather M0" -I{runtime.tools.CMSIS.path}
-IC:\Users\ladyada\AppData\Roaming\Arduino15\packages\adafruit\hardware\samd\1.0.2\co
-IC:\Users\ladyada\AppData\Roaming\Arduino15\packages\adafruit\hardware\samd\1.0.2\va
-IC:\Users\ladyada\AppData\Roaming\Arduino15\packages\adafruit\hardware\samd\1.0.2\li
-IC:\Users\ladyada\Dropbox\ArduinoSketches\libraries\WINC1500\src C:\Users\ladyada\Ap
C:\Users\ladyada\AppData\Local\Temp\build5777565695508348365.tmp\mqtt_winc1500.cpp.o

Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-non-eabi-g++" (in
```

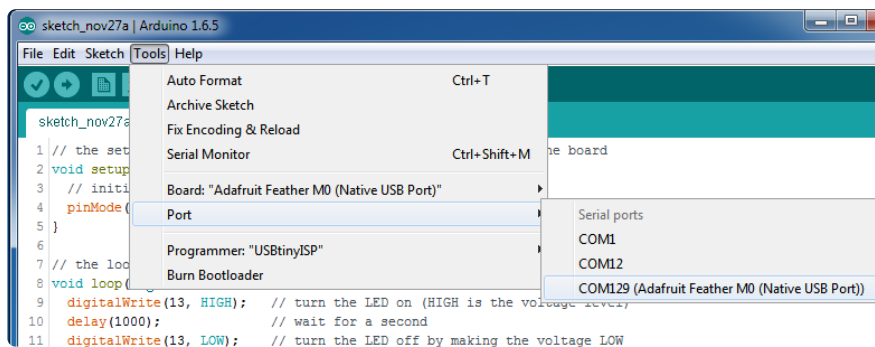
1 Adafruit Feather M0 (Native USB Port) on COM11

Manually bootloading

If you ever get in a 'weird' spot with the bootloader, or you have uploaded code that crashes and doesn't auto-reboot into the bootloader, click the RST button twice (like a double-click) to get back into the bootloader.

The red LED will pulse and/or RGB LED will be green, so you know that its in bootloader mode.

Once it is in bootloader mode, you can select the newly created COM/Serial port and re-try uploading.



You may need to go back and reselect the 'normal' USB serial port next time you want to use the normal upload.

Ubuntu & Linux Issue Fix

[Follow the steps for installing Adafruit's udev rules on this page. \(\)](#)

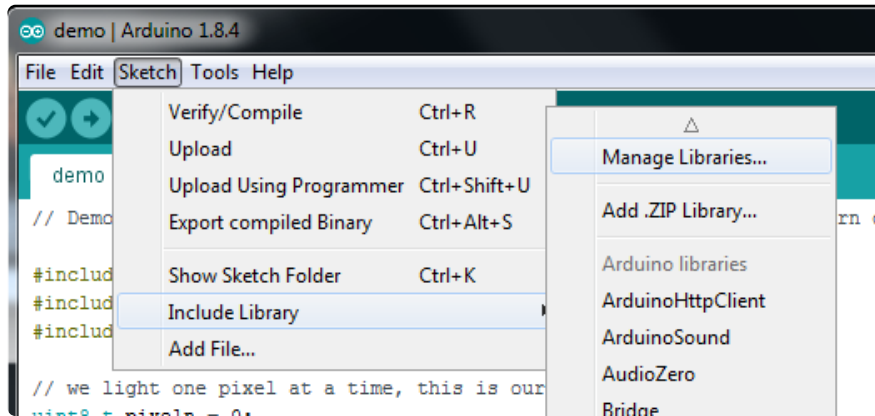
Arcada Libraries

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

There's a lot of libraries!

Install Libraries

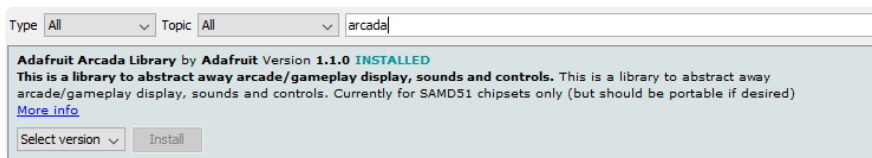
Open up the library manager...



And install the following libraries:

Adafruit Arcada

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

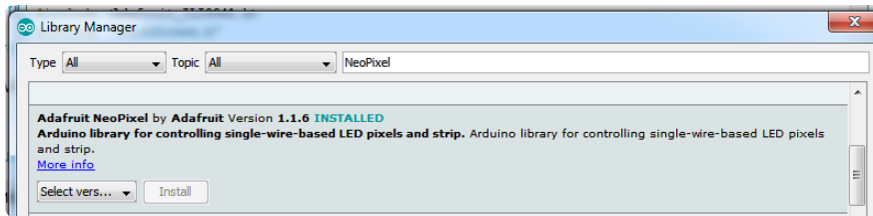


If you use Arduino 1.8.10 or later, the IDE will automatically install all the libraries you need to run all the Arcada demos when you install Arcada. We strongly recommend using the latest IDE so you don't miss one of the libraries!

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

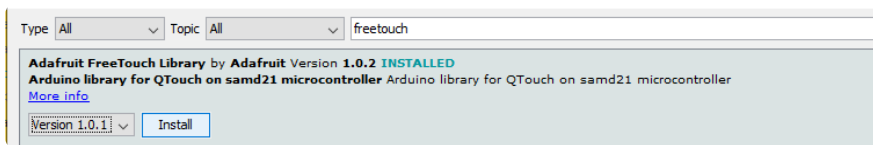
Adafruit NeoPixel

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



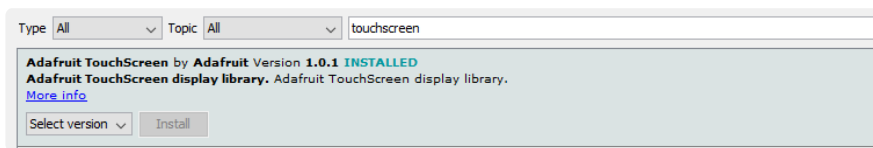
Adafruit FreeTouch

This is the open source version of QTouch for SAMD21 boards



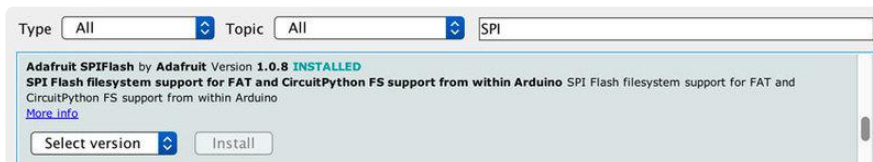
Adafruit Touchscreen

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



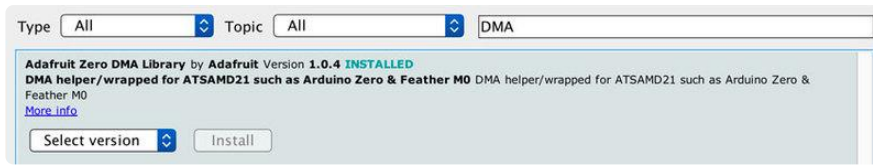
Adafruit SPIFlash

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



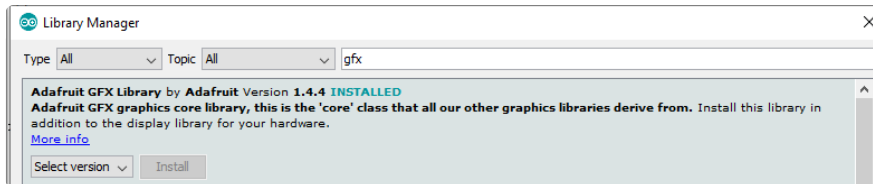
Adafruit Zero DMA

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



Adafruit GFX

This is the graphics library used to draw to the screen



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

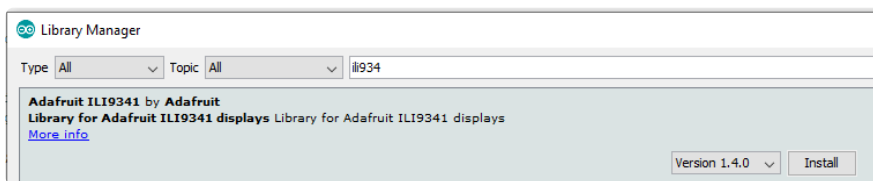
Adafruit ST7735

The display on the PyBadge/PyGamer & other Arcada boards



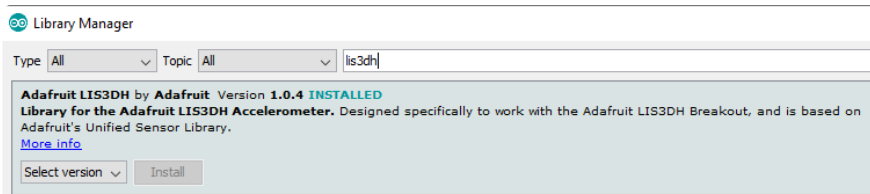
Adafruit ILI9341

The display on the PyPortal & other Arcada boards



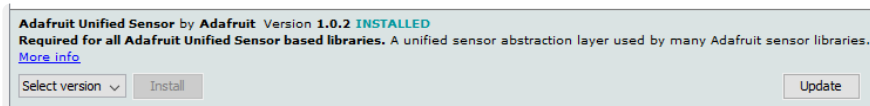
Adafruit LIS3DH

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



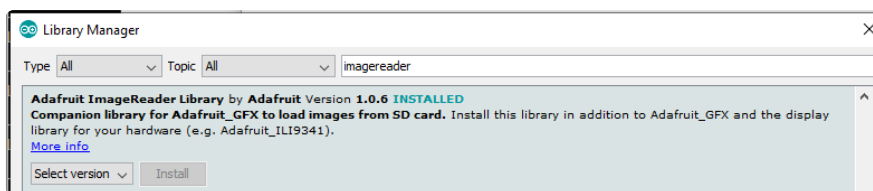
Adafruit Sensor

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



Adafruit ImageReader

For reading bitmaps from SPI Flash or SD and displaying



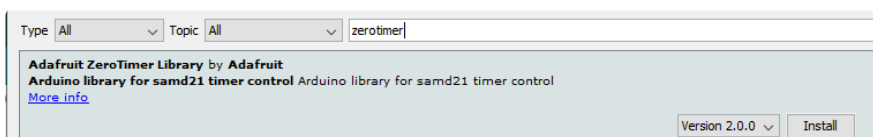
ArduinoJson

We use this library to read and write configuration files



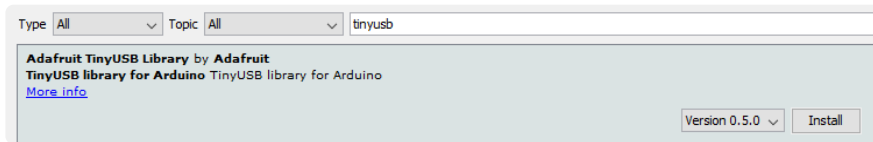
Adafruit ZeroTimer

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



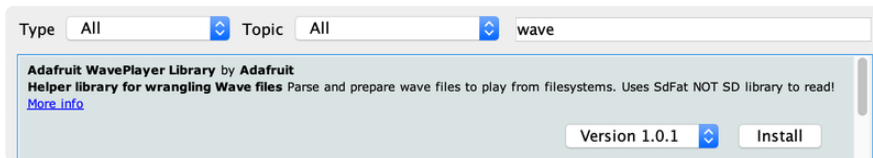
Adafruit TinyUSB

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



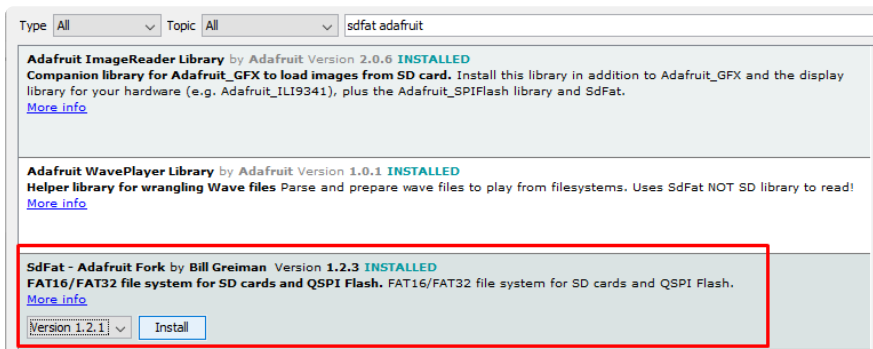
Adafruit WavePlayer

Helps us play .WAV sound files.



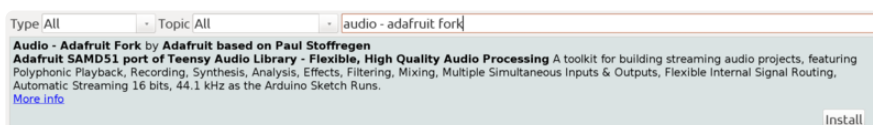
SdFat (Adafruit Fork)

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



Audio - Adafruit Fork

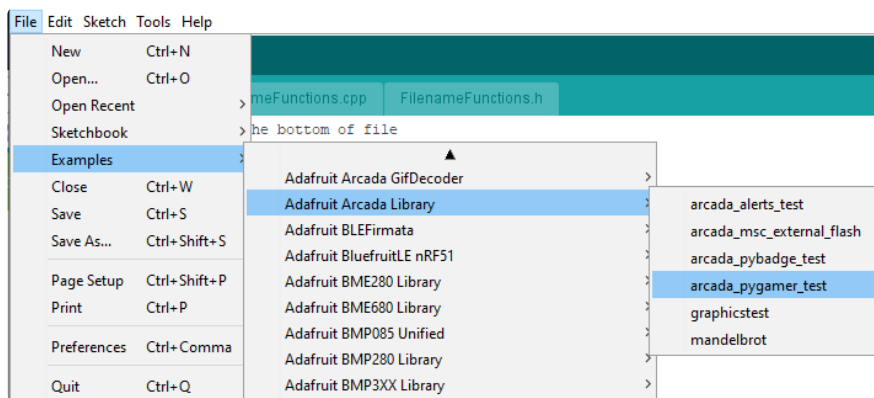
Our fork of the Audio library provides a toolkit for building streaming audio projects.



Arduino Test

Once you've got the IDE installed and libraries in place you can run our 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 light sensor or read the buttons.

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



The test code

1. Checks the QSPI flash chip initialised correctly, and displays the manufacturer/ device ID if so
2. Checks if the QPI flash has a filesystem on it (if not, try loading CircuitPython which will create a filesystem). It will print the # of files found in the root directory
3. Tests if an accelerometer was found & print out the X, Y, Z gravitational tuple.
4. Display the light sensor value, which ranges from 0 (dark) to 1023 (bright)
5. Display the detected battery voltage, from ~3.3V to 4.2V (charged). If no battery, this will float around 4.1V and is normal (there's no way to detect a battery is connected)
6. **D3/A8** and **D4/A9** measure the analog voltages on the 3 pin JST connectors. They'll be floating until some voltage is applied to them, so ~0.4V is normal
7. Draw a 'virtual' joystick for the thumbstick and 4 buttons on the front of the Gamer, when buttons are pressed

If the PyGamer accelerometer is shaken or tapped, it will play a 'coin' sound from the speaker or headphones if they are plugged in

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

PYGAMER_TEST.UF2



Graphics Demos

PyBadge & PyGamer uses a 1.8" 160x128 to display graphics, messages, games what have you!

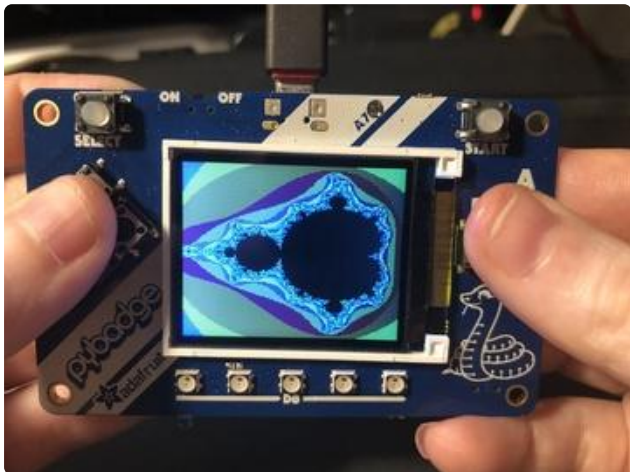
We use a wrapper library called Arcada to let you draw to the display, read buttons and sensors, manage the audio, etc. It also handles things like allocating a framebuffer and then drawing it on command, either as a blocking function (waits until drawing is complete to return) or non-blocking (returns immediately, DMA will draw in the background)

Arcada is a direct subclass of Adafruit_GFX, so if you want to use it to draw shapes and text, check out the GFX guide here first!

[Adafruit GFX Guide](#)



If you want to try out all the shapes and drawing capabilities, check out the `Adafruit_Arcada->graphicstest` example in the library.



The `Adafruit_Arcada->mandelbrot` example is a good demo to show how we allocate a full display buffer, do all our calculations, then draw it all at once.

Call

```
arcada.createFramebuffer(ARCADA_TFT_WIDTH,  
ARCADA_TFT_HEIGHT) to allocate a  
framebuffer in the arcada object, then  
request the pointer with framebuffer =  
arcada.getFramebuffer();
```

Fill it up with data and call

```
arcada.blitFramebuffer(x, y,  
blocking); when its ready to draw all at  
once.
```

Arcada Library

This is a quickstart explanation of what Adafruit Arcada library provides, see the detailed Doxygen documents for arguments & return values

Initialization

- `arcadaBegin()` must be called first, it will set pin directions, turn off NeoPixels, and check for connected hardware
- `filesystemBeginMSD()` will initialize the storage method (SD or SPI flash) and check if a proper filesystem exists. On SD cards that's a FAT filesystem (so make sure its formatted). On SPI Flash we use CircuitPython's FAT filesystem, the best

way to format is to load CircuitPython on once. If you're using TinyUSB as your USB stack, this will also make the disk drive appear on a computer

- `displayBegin()` initializes the display, you will need to turn on the backlight after this is done - we don't do it for you!

Joystick & Buttons

- `readJoystickX` and `readJoystickY` read the analog joystick (if there is one) and returns -512 to 511 with 0 being 'center' (approximately)
- `readButtons` returns a 32 bit mask for each button pressed at the moment of the function call - right now only the bottom 8 bits are used. Check `Adafruit_Arcada_Def.h` for the button mask names. Analog joysticks are checked against a threshold and 'emulate' a button press

Some boards, like the MONSTER M4SK and HalloWings, do not have a proper joystick - instead we will return the capacitive touch pads or buttons as if there was a joystick. For example, the M4SK's three buttons will return 'up', 'A' and 'down' respectively.

- After `readButtons` is called, `justPressedButtons` will tell you buttons were pressed as of the `readButtons` call
- Ditto for `justReleasedButtons`

Backlight, Speaker and Sensors

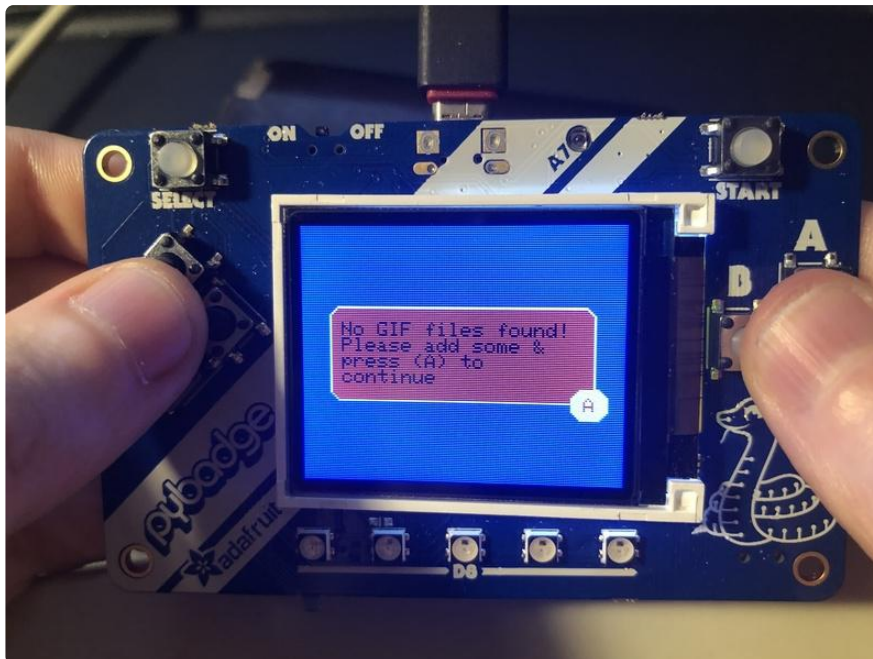
- Enable/disable speaker amplifier (if there is one) with `enableSpeaker` - this doesn't affect headphones if there are any
- `readBatterySensor` returns the battery voltage detected. You cannot detect whether a battery is being charged, only the voltage.
- `readLightSensor` will return 0 for dark, 1023 for bright surrounding light.
- `setBacklight` can set the backlight from 0 (off) to 255 (all the way on)

Alert Boxes

These info boxes and alert display on the screen to let the user know something they need to do, get ready for, or went wrong. You can have the alert wait for a button

press or have it return immediately (then you can delay or wait for something else to occur)

- `alertBox` is the generic, you can set the message, box and text color, as well as button press
- `infoBox` is an `alertBox` where the default button is A and the box color is white, text color is black
- `warnBox` is an `alertBox` where the default button is A and the box color is yellow, text color is black
- `errorBox` is an `alertBox` where the default button is A and the box color is red, text color is white
- `haltBox` is an `alertBox` where the box color is red, text color is white. It will sit in a busy loop and never return



Arcada Library Docs

[Arcada Library Docs \(\)](#)

Adapting Sketches to M0 & M4

The ATSAM21 and 51 are very nice little chips, but fairly new as Arduino-compatible cores go. Most sketches & libraries will work but here's a collection of things we noticed.

The notes below cover a range of Adafruit M0 and M4 boards, but not every rule will apply to every board (e.g. Trinket and Gemma M0 do not have ARef, so you can skip the Analog References note!).

Analog References

If you'd like to use the ARef pin for a non-3.3V analog reference, the code to use is `analogReference(AR_EXTERNAL)` (it's AR_EXTERNAL not EXTERNAL)

Pin Outputs & Pullups

The old-style way of turning on a pin as an input with a pullup is to use

```
pinMode(pin, INPUT)
digitalWrite(pin, HIGH)
```

This is because the pullup-selection register on 8-bit AVR chips is the same as the output-selection register.

For M0 & M4 boards, you can't do this anymore! Instead, use:

```
pinMode(pin, INPUT_PULLUP)
```

Code written this way still has the benefit of being backwards compatible with AVR. You don't need separate versions for the different board types.

Serial vs SerialUSB

99.9% of your existing Arduino sketches use Serial.print to debug and give output. For the Official Arduino SAMD/M0 core, this goes to the Serial5 port, which isn't exposed on the Feather. The USB port for the Official Arduino M0 core is called SerialUSB instead.

In the Adafruit M0/M4 Core, we fixed it so that Serial goes to USB so it will automatically work just fine.

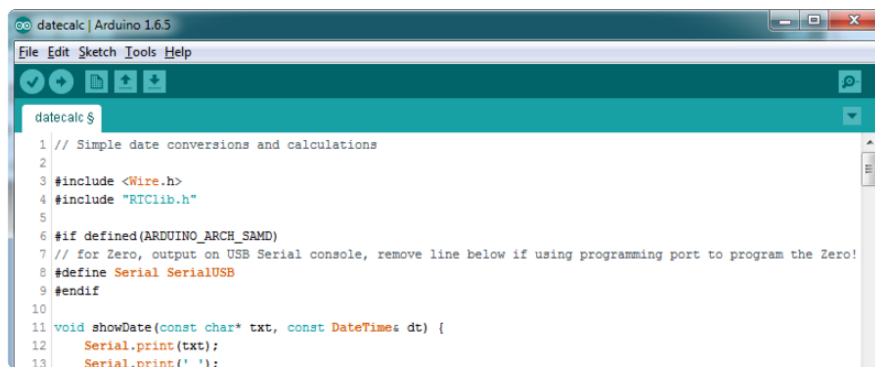
However, on the off chance you are using the official Arduino SAMD core and not the Adafruit version (which really, we recommend you use our version because it's been

tuned to our boards), and you want your Serial prints and reads to use the USB port, use SerialUSB instead of Serial in your sketch.

If you have existing sketches and code and you want them to work with the M0 without a huge find-replace, put

```
#if defined(ARDUINO_SAMD_ZERO) && defined(SERIAL_PORT_USBVIRTUAL)
// Required for Serial on Zero based boards
#define Serial SERIAL_PORT_USBVIRTUAL
#endif
```

right above the first function definition in your code. For example:



```
datecalc | Arduino 1.6.5
File Edit Sketch Tools Help
datecalc $
1 // Simple date conversions and calculations
2
3 #include <Wire.h>
4 #include "RTCLib.h"
5
6 #if defined(ARDUINO_ARCH_SAMD)
7 // for Zero, output on USB Serial console, remove line below if using programming port to program the Zero!
8 #define Serial SerialUSB
9 #endif
10
11 void showDate(const char* txt, const DateTime: dt) {
12     Serial.print(txt);
13     Serial.print(' ');
```

AnalogWrite / PWM on Feather/Metro M0

After looking through the SAMD21 datasheet, we've found that some of the options listed in the multiplexer table don't exist on the specific chip used in the Feather M0.

For all SAMD21 chips, there are two peripherals that can generate PWM signals: The Timer/Counter (TC) and Timer/Counter for Control Applications (TCC). Each SAMD21 has multiple copies of each, called 'instances'.

Each TC instance has one count register, one control register, and two output channels. Either channel can be enabled and disabled, and either channel can be inverted. The pins connected to a TC instance can output identical versions of the same PWM waveform, or complementary waveforms.

Each TCC instance has a single count register, but multiple compare registers and output channels. There are options for different kinds of waveform, interleaved switching, programmable dead time, and so on.

The biggest members of the SAMD21 family have five TC instances with two 'waveform output' (WO) channels, and three TCC instances with eight WO channels:

- TC[0-4],WO[0-1]
- TCC[0-2],WO[0-7]

And those are the ones shown in the datasheet's multiplexer tables.

The SAMD21G used in the Feather M0 only has three TC instances with two output channels, and three TCC instances with eight output channels:

- TC[3-5],WO[0-1]
- TCC[0-2],WO[0-7]

Tracing the signals to the pins broken out on the Feather M0, the following pins can't do PWM at all:

- Analog pin A5

The following pins can be configured for PWM without any signal conflicts as long as the SPI, I2C, and UART pins keep their protocol functions:

- Digital pins 5, 6, 9, 10, 11, 12, and 13
- Analog pins A3 and A4

If only the SPI pins keep their protocol functions, you can also do PWM on the following pins:

- TX and SDA (Digital pins 1 and 20)

analogWrite() PWM range

On AVR, if you set a pin's PWM with `analogWrite(pin, 255)` it will turn the pin fully HIGH. On the ARM cortex, it will set it to be 255/256 so there will be very slim but still-existing pulses-to-0V. If you need the pin to be fully on, add test code that checks if you are trying to `analogWrite(pin, 255)` and, instead, does a `digitalWrite(pin, HIGH)`

analogWrite() DAC on A0

If you are trying to use `analogWrite()` to control the DAC output on A0, make sure you do not have a line that sets the pin to output. Remove: `pinMode(A0, OUTPUT)`.

Missing header files

There might be code that uses libraries that are not supported by the M0 core. For example if you have a line with

```
#include <util/delay.h>
```

you'll get an error that says

```
fatal error: util/delay.h: No such file or directory
```

```
#include <util/delay.h>
```

```
^
```

```
compilation terminated.
```

```
Error compiling.
```

In which case you can simply locate where the line is (the error will give you the file name and line number) and 'wrap it' with `#ifdef`'s so it looks like:

```
#if !defined(ARDUINO_ARCH_SAM) && !defined(ARDUINO_ARCH_SAMD) && !  
defined(ESP8266) && !defined(ARDUINO_ARCH_STM32F2)  
  #include <util/delay.h>  
#endif
```

The above will also make sure that header file isn't included for other architectures

If the `#include` is in the arduino sketch itself, you can try just removing the line.

Bootloader Launching

For most other AVR's, clicking reset while plugged into USB will launch the bootloader manually, the bootloader will time out after a few seconds. For the M0/M4, you'll need to double click the button. You will see a pulsing red LED to let you know you're in bootloader mode. Once in that mode, it won't time out! Click reset again if you want to go back to launching code.

Aligned Memory Access

This is a little less likely to happen to you but it happened to me! If you're used to 8-bit platforms, you can do this nice thing where you can typecast variables around. e.g.

```
uint8_t mybuffer[4];  
float f = (float)mybuffer;
```

You can't be guaranteed that this will work on a 32-bit platform because mybuffer might not be aligned to a 2 or 4-byte boundary. The ARM Cortex-M0 can only directly access data on 16-bit boundaries (every 2 or 4 bytes). Trying to access an odd-boundary byte (on a 1 or 3 byte location) will cause a Hard Fault and stop the MCU. Thankfully, there's an easy work around ... just use memcpy!

```
uint8_t mybuffer[4];  
float f;  
memcpy(&f, mybuffer, 4)
```

Floating Point Conversion

Like the AVR Arduinos, the M0 library does not have full support for converting floating point numbers to ASCII strings. Functions like sprintf will not convert floating point. Fortunately, the standard AVR-LIBC library includes the dtostrf function which can handle the conversion for you.

Unfortunately, the M0 run-time library does not have dtostrf. You may see some references to using #include <avr/dtostrf.h> to get dtostrf in your code. And while it will compile, it does not work.

Instead, check out this thread to find a working dtostrf function you can include in your code:

[http://forum.arduino.cc/index.php?topic=368720.0 \(\)](http://forum.arduino.cc/index.php?topic=368720.0)

How Much RAM Available?

The ATSAM21G18 has 32K of RAM, but you still might need to track it for some reason. You can do so with this handy function:

```
extern "C" char *sbrk(int i);

int FreeRam () {
    char stack_dummy = 0;
    return &stack_dummy - sbrk(0);
}
```

Thx to <http://forum.arduino.cc/index.php?topic=365830.msg2542879#msg2542879> () for the tip!

Storing data in FLASH

If you're used to AVR, you've probably used PROGMEM to let the compiler know you'd like to put a variable or string in flash memory to save on RAM. On the ARM, it's a little easier, simply add const before the variable name:

```
const char str[] = "My very long string";
```

That string is now in FLASH. You can manipulate the string just like RAM data, the compiler will automatically read from FLASH so you don't need special progmem-knowledgeable functions.

You can verify where data is stored by printing out the address:
Serial.print("Address of str \$"); Serial.println((int)&str, HEX);

If the address is \$2000000 or larger, it's in SRAM. If the address is between \$0000 and \$3FFFF Then it is in FLASH

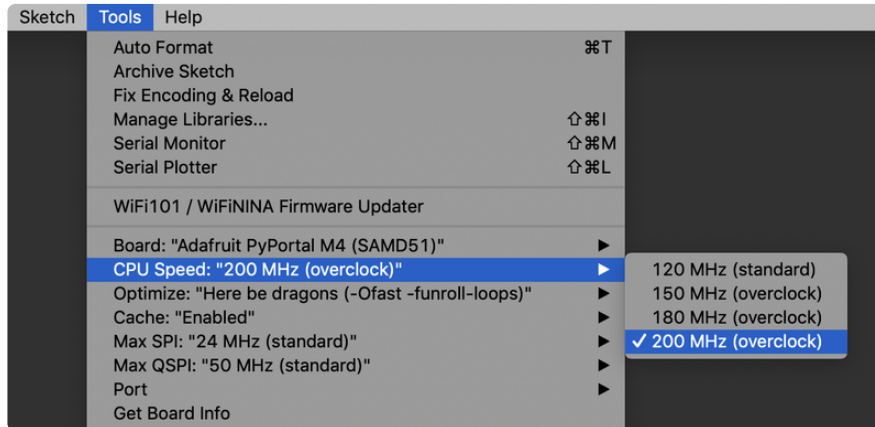
Pretty-Printing out registers

There's a lot of registers on the SAMD21, and you often are going through ASF or another framework to get to them. So having a way to see exactly what's going on is handy. This library from drewfish will help a ton!

<https://github.com/drewfish/arduino-ZeroRegs> ()

M4 Performance Options

As of version 1.4.0 of the Adafruit SAMD Boards package in the Arduino Boards Manager, some options are available to wring extra performance out of M4-based devices. These are in the Tools menu.



All of these performance tweaks involve a degree of uncertainty. There's no guarantee of improved performance in any given project, and some may even be detrimental, failing to work in part or in whole. If you encounter trouble, select the default performance settings and re-upload.

Here's what you get and some issues you might encounter...

CPU Speed (overclocking)

This option lets you adjust the microcontroller core clock...the speed at which it processes instructions...beyond the official datasheet specifications.

Manufacturers often rate speeds conservatively because such devices are marketed for harsh industrial environments...if a system crashes, someone could lose a limb or worse. But most creative tasks are less critical and operate in more comfortable settings, and we can push things a bit if we want more speed.

There is a small but nonzero chance of code locking up or failing to run entirely. If this happens, try dialing back the speed by one notch and re-upload, see if it's more stable.

Much more likely, some code or libraries may not play well with the nonstandard CPU speed. For example, currently the NeoPixel library assumes a 120 MHz CPU speed and won't issue the correct data at other settings (this will be worked on). Other

libraries may exhibit similar problems, usually anything that strictly depends on CPU timing...you might encounter problems with audio- or servo-related code depending how it's written. If you encounter such code or libraries, set the CPU speed to the default 120 MHz and re-upload.

Optimize

There's usually more than one way to solve a problem, some more resource-intensive than others. Since Arduino got its start on resource-limited AVR microcontrollers, the C++ compiler has always aimed for the smallest compiled program size. The "Optimize" menu gives some choices for the compiler to take different and often faster approaches, at the expense of slightly larger program size...with the huge flash memory capacity of M4 devices, that's rarely a problem now.

The "Small" setting will compile your code like it always has in the past, aiming for the smallest compiled program size.

The "Fast" setting invokes various speed optimizations. The resulting program should produce the same results, is slightly larger, and usually (but not always) noticeably faster. It's worth a shot!

"Here be dragons" invokes some more intensive optimizations...code will be larger still, faster still, but there's a possibility these optimizations could cause unexpected behaviors. Some code may not work the same as before. Hence the name. Maybe you'll discover treasure here, or maybe you'll sail right off the edge of the world.

Most code and libraries will continue to function regardless of the optimizer settings. If you do encounter problems, dial it back one notch and re-upload.

Cache

This option allows a small collection of instructions and data to be accessed more quickly than from flash memory, boosting performance. It's enabled by default and should work fine with all code and libraries. But if you encounter some esoteric situation, the cache can be disabled, then recompile and upload.

Max SPI and Max QSPI

These should probably be left at their defaults. They're present mostly for our own experiments and can cause serious headaches.

Max SPI determines the clock source for the M4's SPI peripherals. Under normal circumstances this allows transfers up to 24 MHz, and should usually be left at that setting. But...if you're using write-only SPI devices (such as TFT or OLED displays), this option lets you drive them faster (we've successfully used 60 MHz with some TFT screens). The caveat is, if using any read/write devices (such as an SD card), this will not work at all...SPI reads absolutely max out at the default 24 MHz setting, and anything else will fail. Write = OK. Read = FAIL. This is true even if your code is using a lower bitrate setting...just having the different clock source prevents SPI reads.

Max QSPI does similarly for the extra flash storage on M4 "Express" boards. Very few Arduino sketches access this storage at all, let alone in a bandwidth-constrained context, so this will benefit next to nobody. Additionally, due to the way clock dividers are selected, this will only provide some benefit when certain "CPU Speed" settings are active. Our [PyPortal Animated GIF Display \(\)](#) runs marginally better with it, if using the QSPI flash.

Enabling the Buck Converter on some M4 Boards

If you want to reduce power draw, some of our boards have an inductor so you can use the 1.8V buck converter instead of the built in linear regulator. If the board does have an inductor (see the schematic) you can add the line `SUPC->VREG.bit.SEL = 1;` to your code to switch to it. Note it will make ADC/DAC reads a bit noisier so we don't use it by default. [You'll save ~4mA \(\)](#).

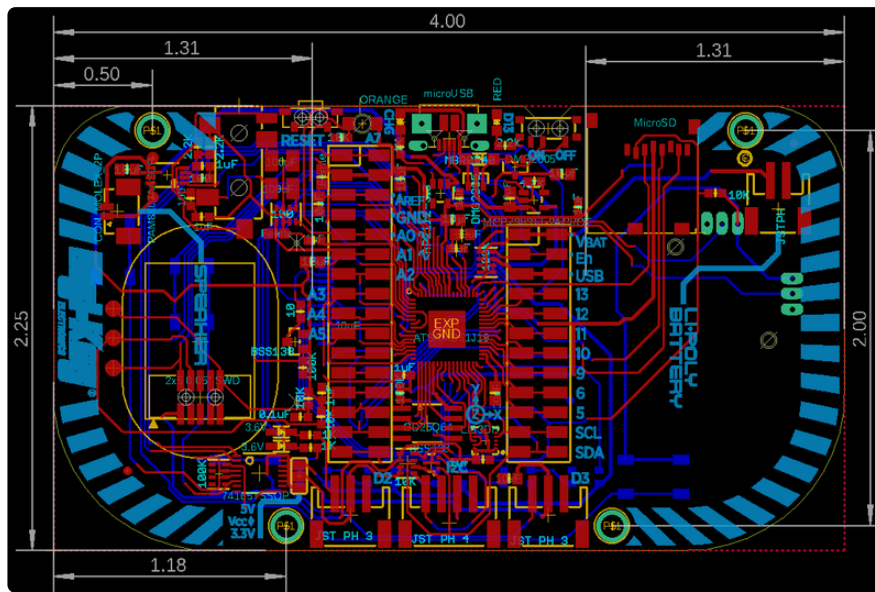
Downloads

Files:

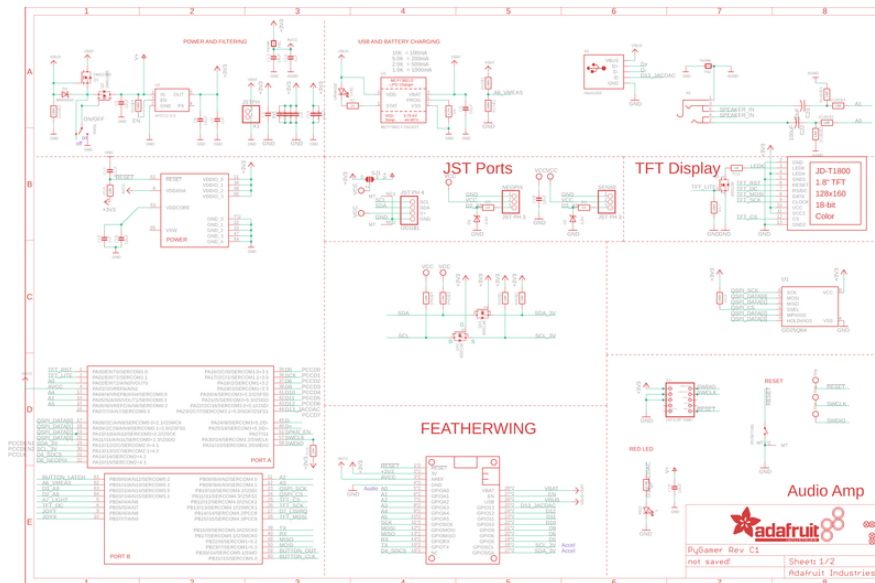
- [ATSAMD51J19 Product page w/datasheets \(\)](#)
- [LIS3DH Datasheet \(\)](#)
- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in the Adafruit Fritzing Library \(\)](#)
- [3D Model on GitHub \(\)](#)
- [PDF for PyGamer Board Diagram on GitHub \(\)](#)

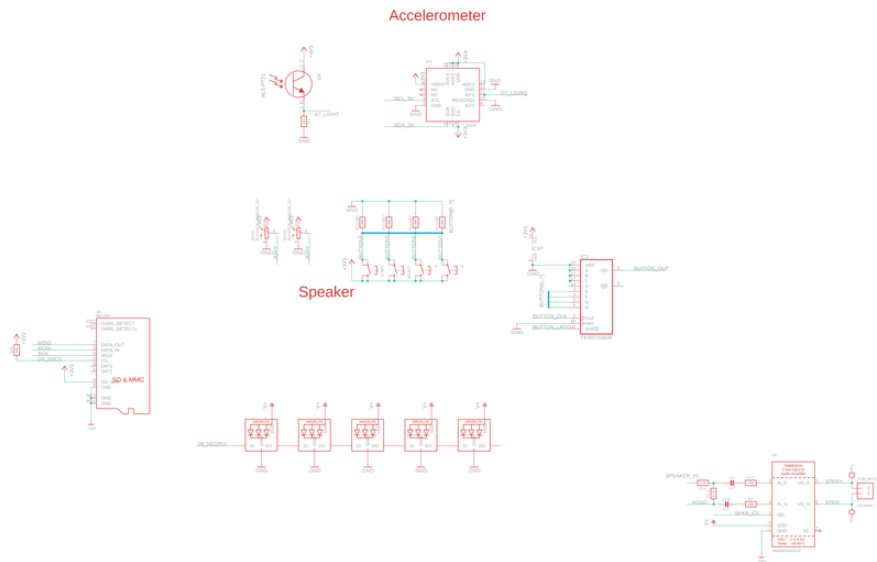
[SVG for PyGamer Board Diagram](#)

Fab Print



Schematic





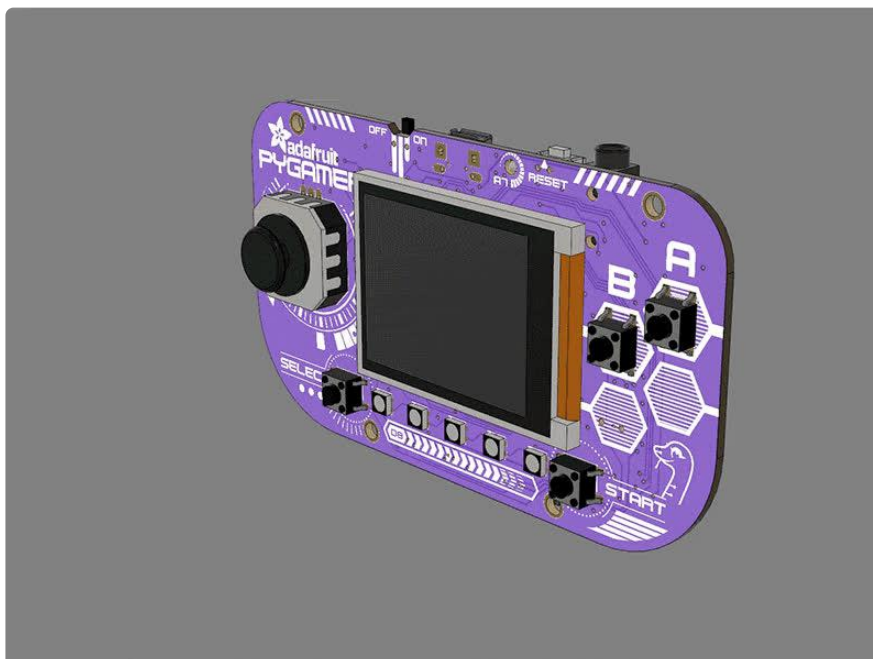
Laser Cut Acrylic Case

PyGamer-Clear-1mm.ai

PyGamer-Smoke-3mm.ai

PyGamer-Clear-3mm.ai

3D Model



Troubleshooting

Digi-Key x Adafruit Order Shipper Game

If you need to re-load the Digi-Key Shipping game, here's the .uf2 file below. Download this to your computer.

pygamer_Digi-
Key_shipping_game.UF2

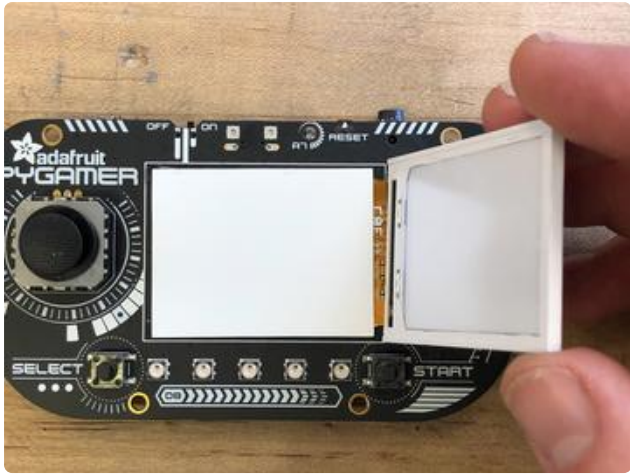
Plug in your PyGamer to your computer with a known good USB data cable (not a "charge-only" cable which should be killed and burned with fire) then turn on the PyGamer and press the reset button to bring up the bootloader. (Also, try both with and without a USB hub if you have one, sometimes they help, sometimes not.)

Drag the .uf2 file onto the PYGAMERBOOT drive that shows up on your computer.

Screen Adhesive Fix

The screen can become un-adhered pretty easily -- these screens were originally designed to be secured with a bezel. Your PyGamer case does a good job of holding it in place, but if you have it out of a case, it can become unstuck.

Luckily, it's easy to fix! Just use two strips of double stick adhesive tape to hold the screen down to the backing plastic.



The small bits of adhesive may have given up the fight. Go ahead and lift the screen away from the backing, being careful to go gentle on the ribbon connector!



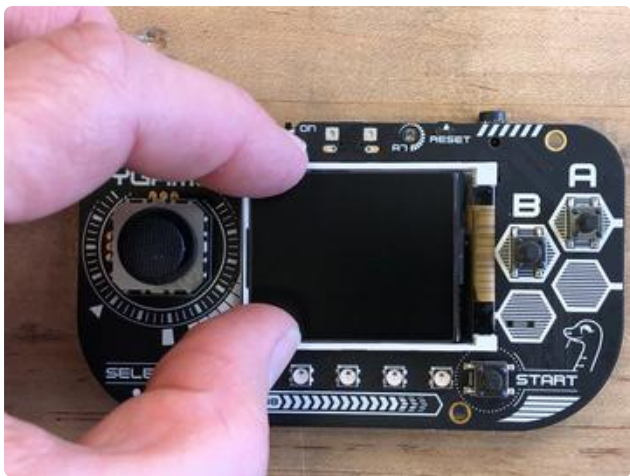
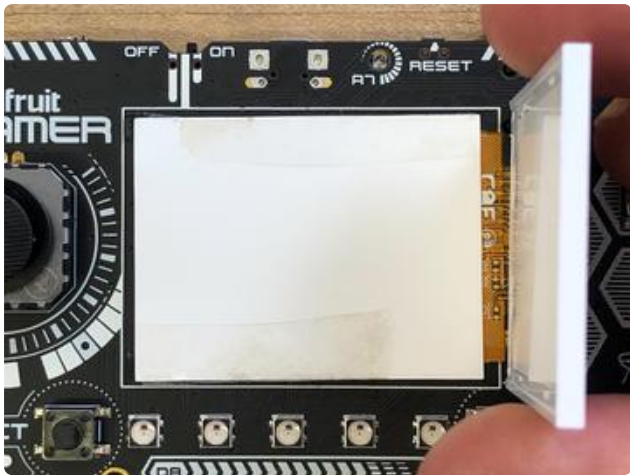
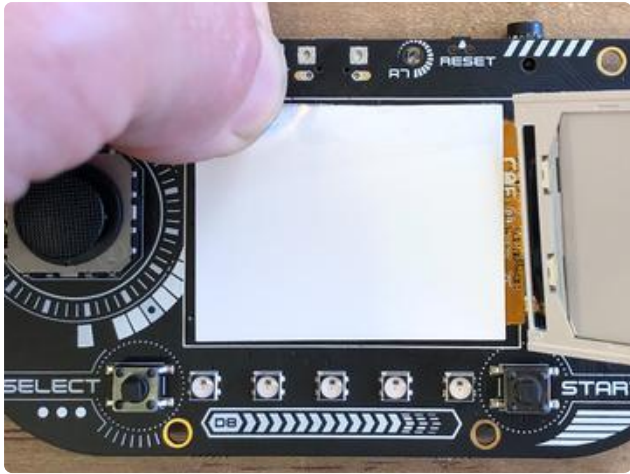
Get some good double stick tape, as shown.

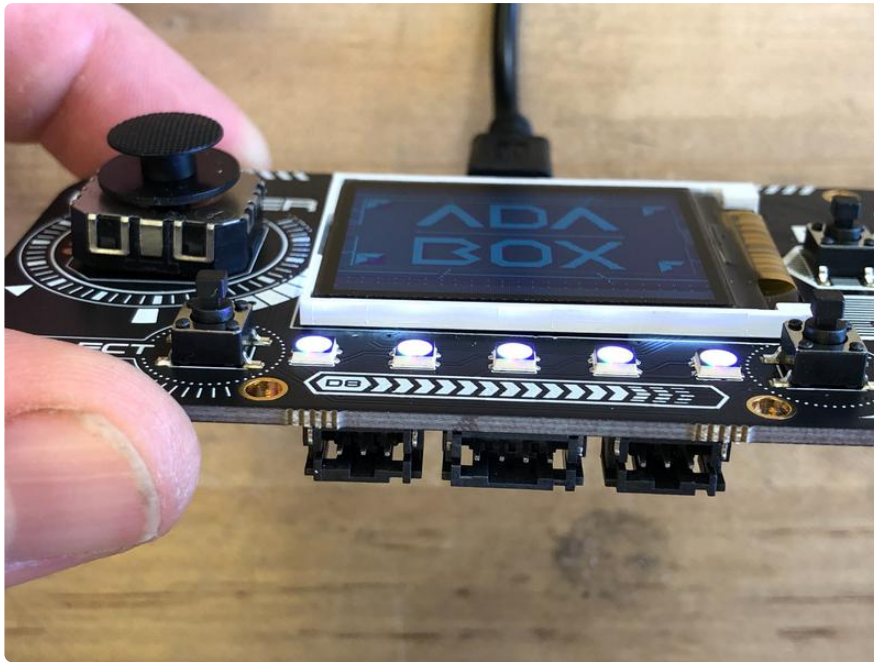
Cut off two short lengths of the tape, you can even use one piece and cut it in half length-wise to make narrow strips.

Place a piece of tape across the bottom and top edges of the backing.



Press the screen down firmly and hold for 30 seconds to ensure good adhesion.





Your PyGamer should be better than ever now, and ready to play!

