adafruit learning system

Introducing Adafruit PyGamer

Created by Kattni Rembor

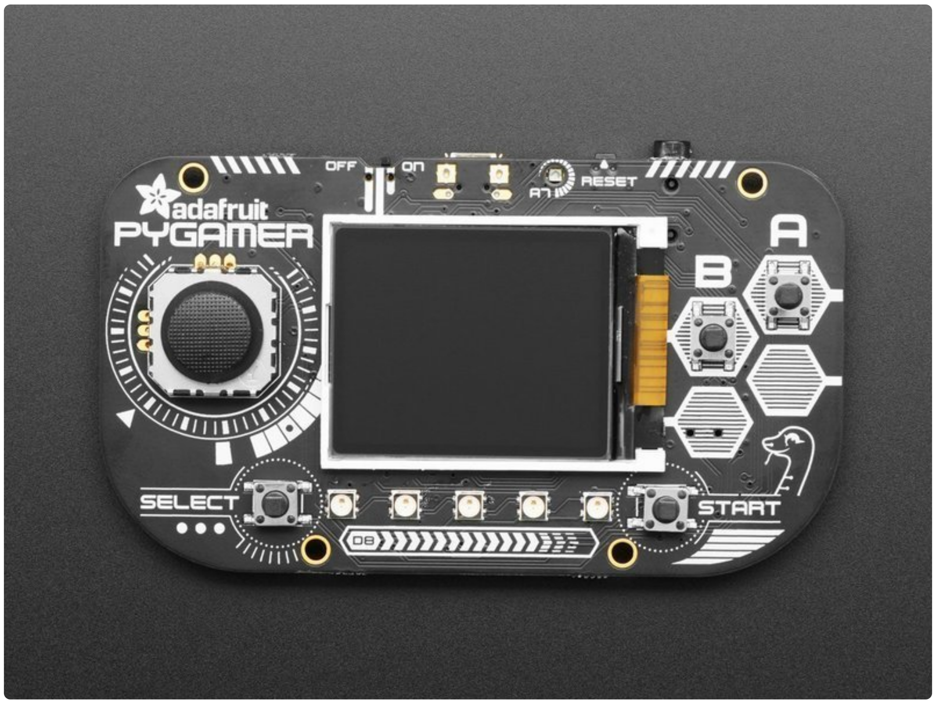
Last updated on 2021-05-19 05:48:19 PM EDT

Guide Contents

Overview

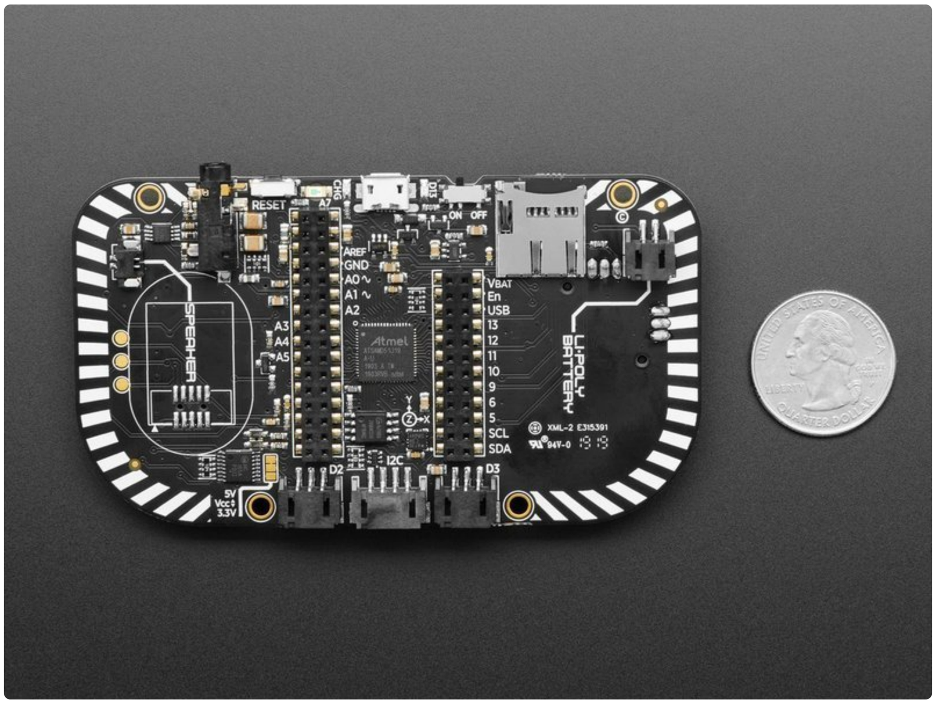
What fits in your pocket, is fully Open Source, and can run CircuitPython, MakeCode Arcade or Arduino games you write yourself? That's right, it's the **Adafruit PyGamer!** We wanted to make an entry-level gaming handheld for DIY gaming, and maybe a little retro-emulation. It's not the fastest and best 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 ATSAMD51, 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 (https://adafru.it/ET7). 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 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.



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 (https://adafru.it/CEv). The PyGamer will auto-switch to speakers when they're plugged in.

You can power the PyGamer from any of our LiPoly batteries (https://adafru.it/waX), but we like this 350mAh one (https://adafru.it/kBj) 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 (https://adafru.it/DCY)
- CircuitPython **(https://adafru.it/FxM)** 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 (https://adafru.it/EF5) 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

- **ATSAMD51J19** @ 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 will 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

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...

$39.95

In Stock

Add to Cart

Adafruit PyBadge for MakeCode Arcade, CircuitPython, or Arduino

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...

$34.95

In Stock

Add to Cart

Adafruit PyBadge LC - MakeCode Arcade, CircuitPython, or Arduino

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...

$24.95

In Stock

Add to Cart

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.

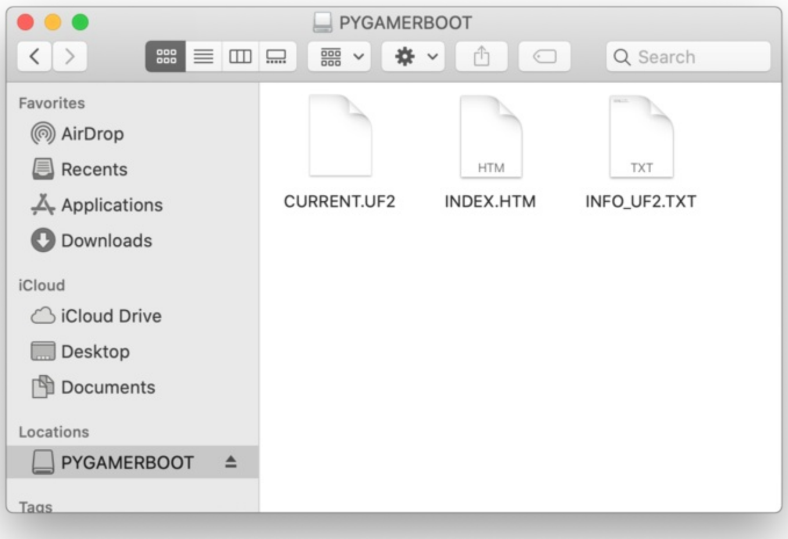
Bootloaders earlier than **v3.9.0** do not protect against a rare problem in which part of internal flash is erased on startup.

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.

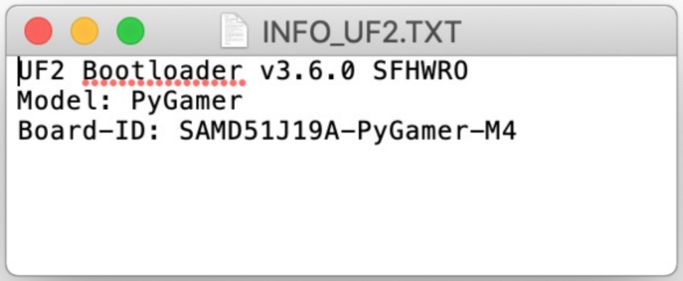
Updating Your PyGamer Bootloader

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**.


```
UF2 Bootloader v3.6.0 SFHWRO
Model: PyGamer
Board-ID: SAMD51J19A-PyGamer-M4
```

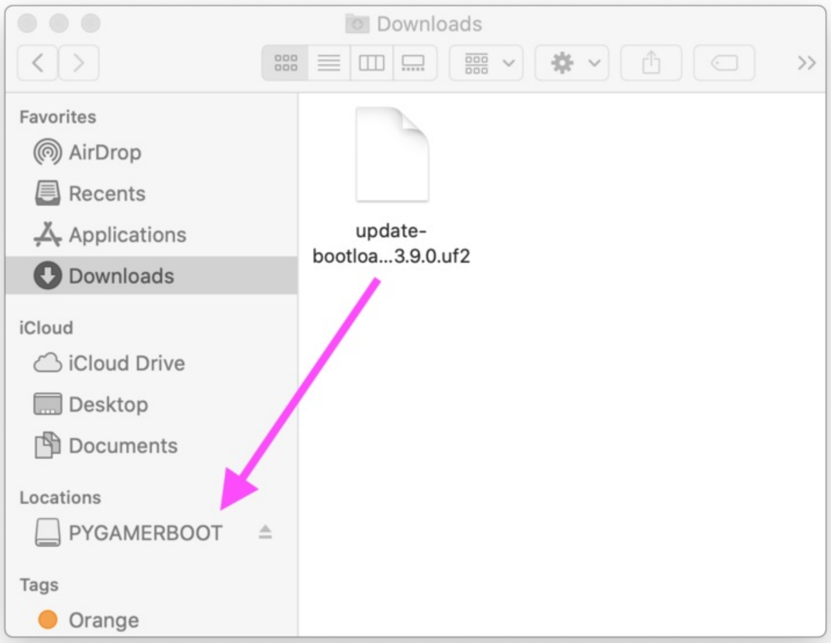
If the bootloader version you see is smaller than "**v3.9.0**", you need to update. For instance, the bootloader above needs to be upgraded.

Download the latest version of the PyGamer bootloader updater from the circuitpython.org Downloads page.

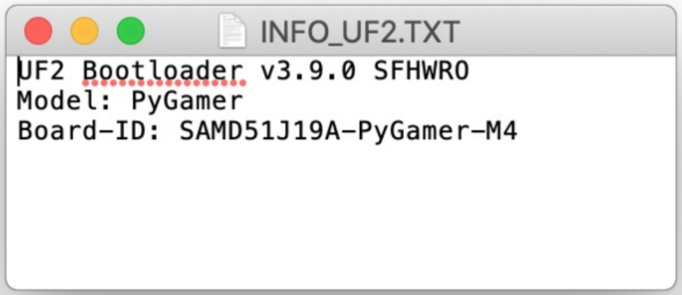
https://adafru.it/FxM

https://adafru.it/FxM

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.

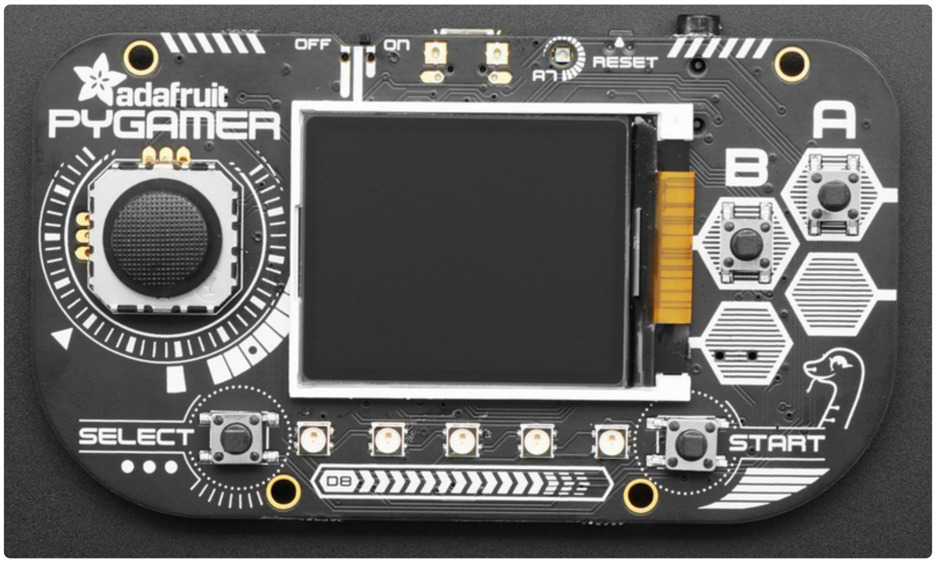


```
UF2 Bootloader v3.9.0 SFHWRO
Model: PyGamer
Board-ID: SAMD51J19A-PyGamer-M4
```

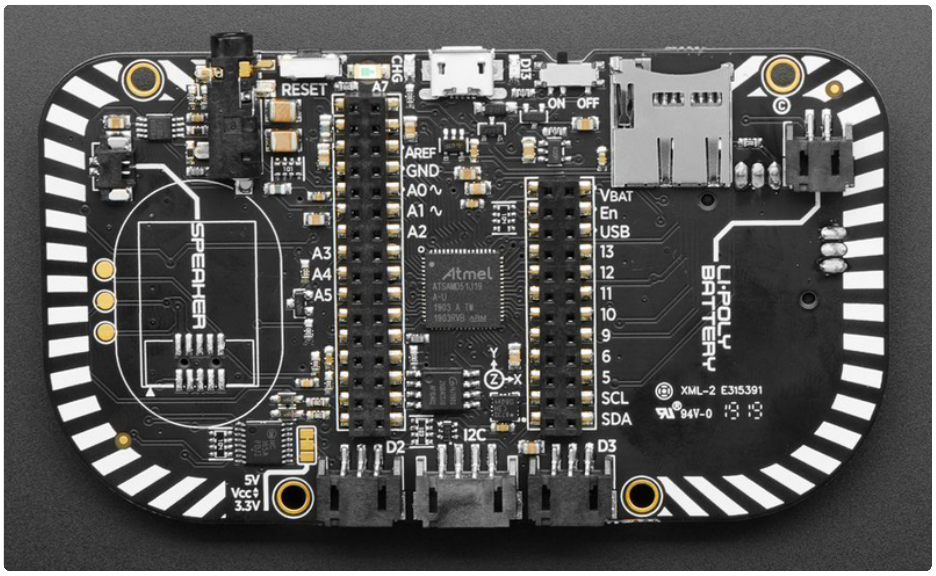
Oh no, I updated MacOS already and I can't see the boot drive!

If your Mac has already been updated to MacOS 10.14.4 and now you can't see a boot drivein 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 (https://adafru.it/Bf7), but then can work.

Pinouts

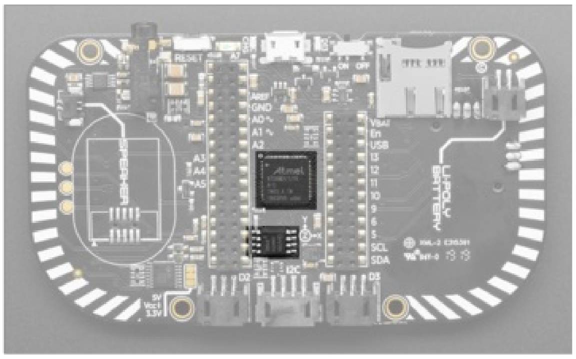


The PyGamer is a DIY handheld gaming development platform, and it's loaded with all sorts of goodies. The front features a display, buttons, a joystick, RGB LEDs, and a light sensor.



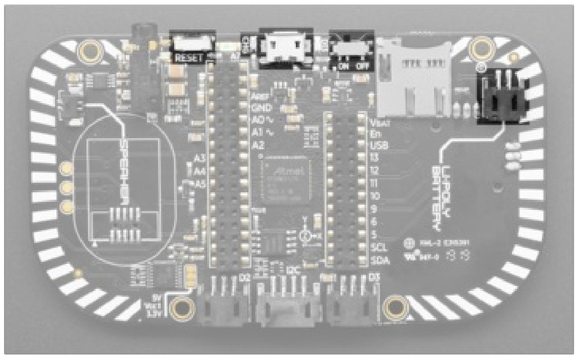
The back features a speaker connector, headphone jack, SD card slot, battery connector, peripheral connectors, an accelerometer and more. Let's take a look!

Microcontroller and Flash



- The main processor chip is the **ATSAMD51J19 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.

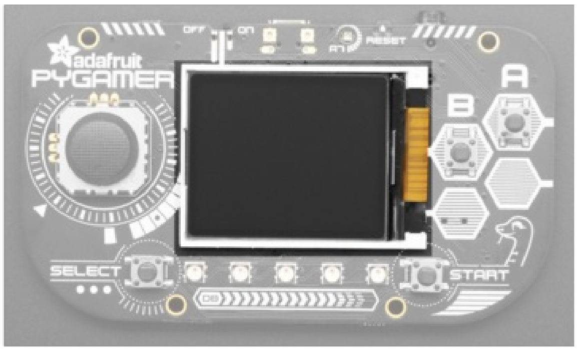
Power



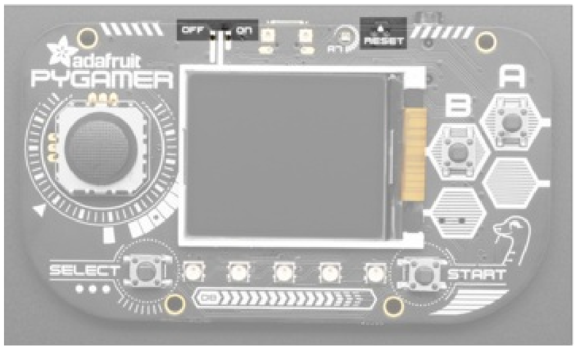
- The **On/Off switch** is located on the top.
- There is one USB port on the board. On the left side, towards the bottom, is a **USB Micro** port, which is used for powering and programming the board.
- There's two ways to power your PyBadge. The best way is to plug in a 3.7/4.2V Lipoly battery into 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.

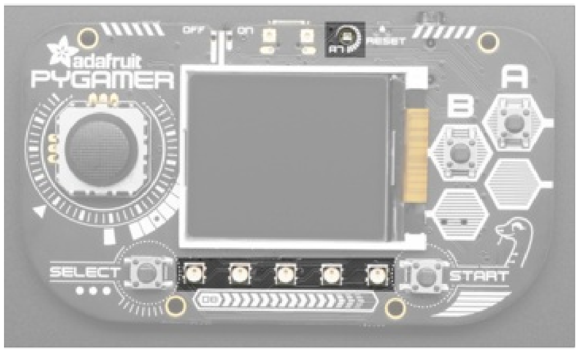
Display



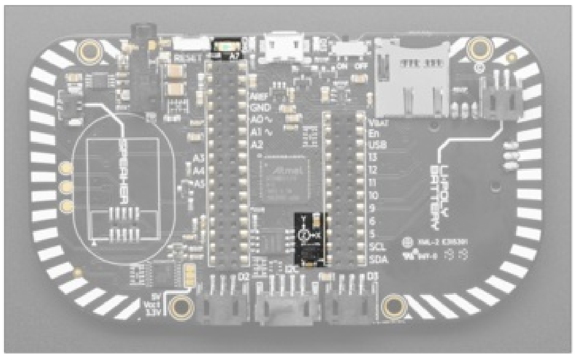
- **1.8" TFT display** - The front features a 160x128 pixel color display.



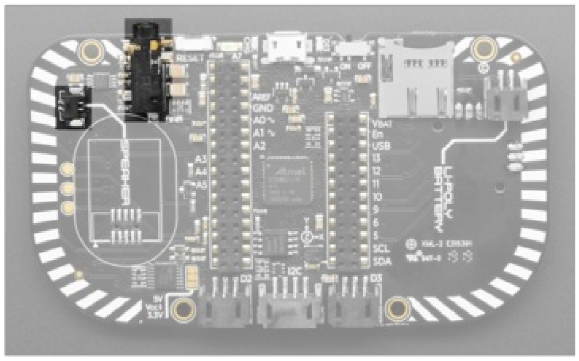
- **On/Off switch** - Powers off the board. Even if plugged in, the board won't work if switched to off. You can charge the battery when the board is off, but the USB device will not be active or any other electronics
- The **reset button** is located on the top to the right of center. Click it once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



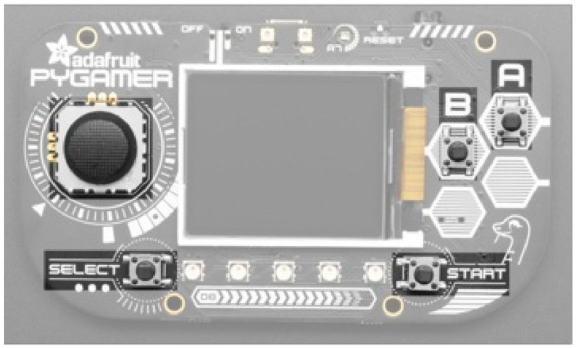
- **Light sensor (A7)** - There is an **ambient light sensor** on the top, which points through to the front. The light sensor is an analog input, connected to `board.LIGHT` (CircuitPython) or `A7` (Arduino) you can read it as any analog value ranging from 0 (dark) to 1023 (in Arduino) or 65535 (CircuitPython) when bright.
- **NeoPixels (D8)** - There are also 5 individually addressable **RGB NeoPixel LEDs** located on the front of the board along the bottom middle. The 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**.
- **Light sensor (A7)** - There is also 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.



- There is a **speaker connector** on the left side of the back, which is a Molex PicoBlade (https://adafru.it/C8p). 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 (https://adafru.it/Bep).

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.

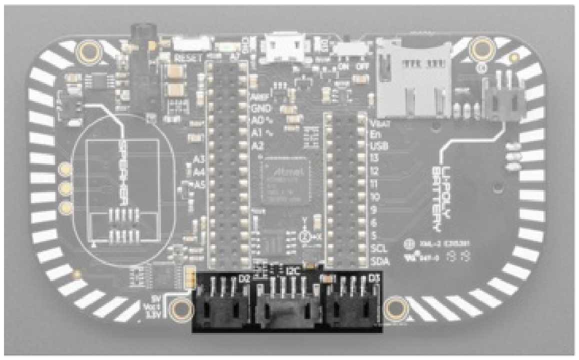
```
#define BUTTON_CLOCK   48
#define BUTTON_DATA    49
#define BUTTON_LATCH   50

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 (https://adafru.it/EF5)

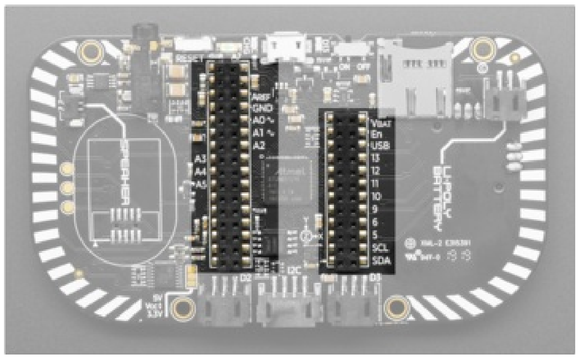
In CircuitPython, use the GamePadShift library to read the button presses for you:

```
pad = GamePadShift(digitalio.DigitalInOut(board.BUTTON_CLOCK),
                   digitalio.DigitalInOut(board.BUTTON_OUT),
       digitalio.DigitalInOut(board.BUTTON_LATCH))
pressed = pad.get_pressed()
```

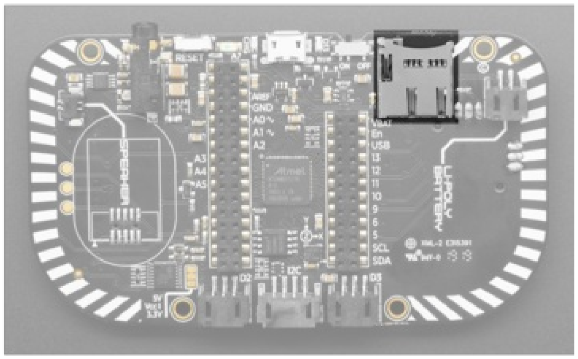
[More info on GamePadShift is here](https://adafru.it/F9O) (https://adafru.it/F9O)



- There is a 4-pin JST **I2C connector** in the center on the bottom, that is STEMMA and Grove compatible. You can access VCC power and ground as well as a level-shifted `SDA` & `SCL` connection. 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 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.

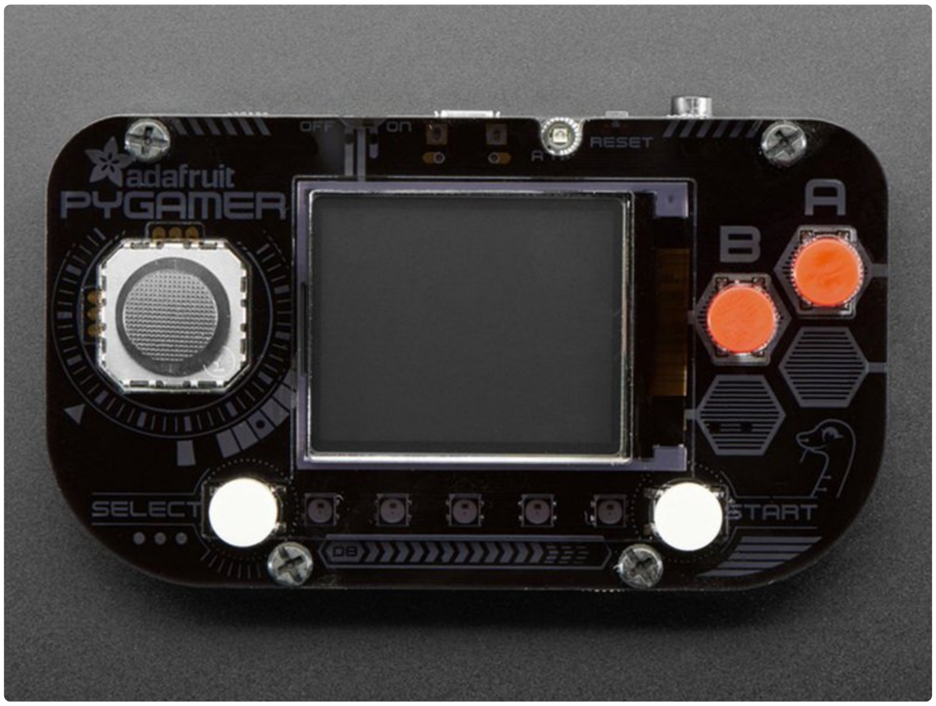


- You can easily attach FeatherWings to the back of your PyBadge using the convenient **"Feather Headers"** on the back. Located in the middle, they break out all the same pins you have access to on a Feather board, allowing use of any of our wide range of FeatherWings. Easily add all kinds of functionality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

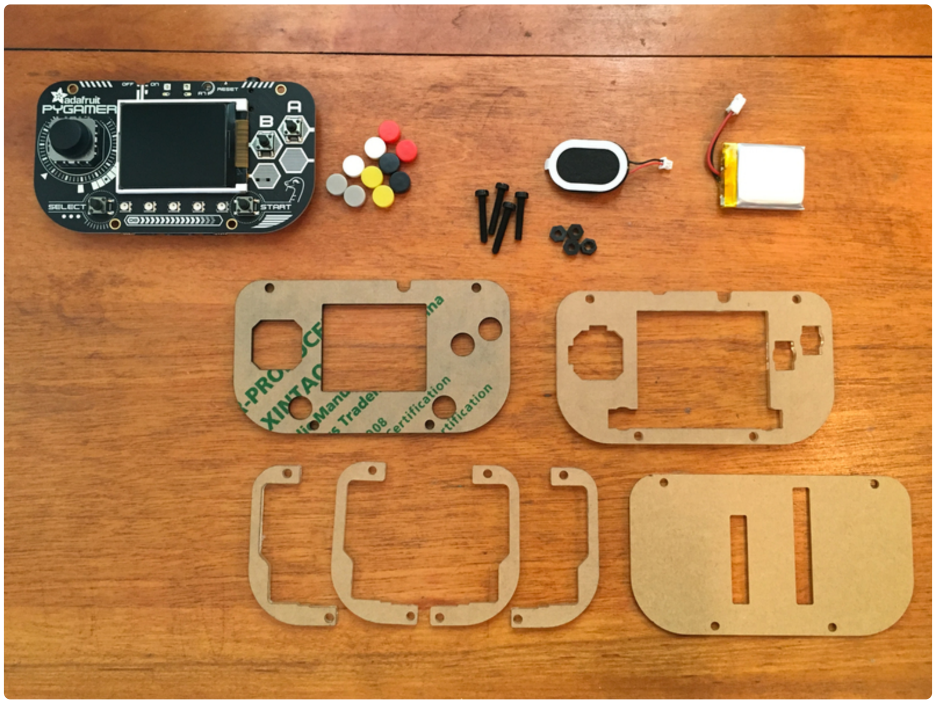


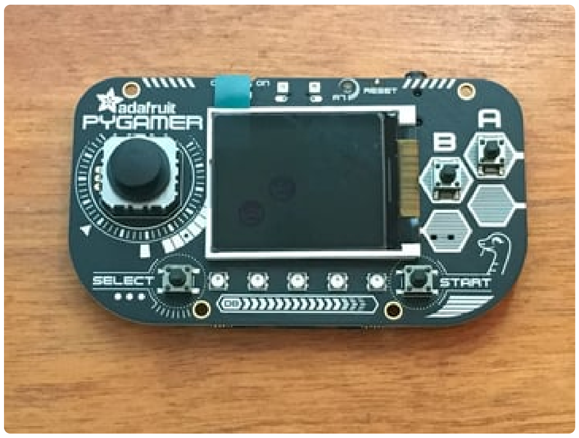
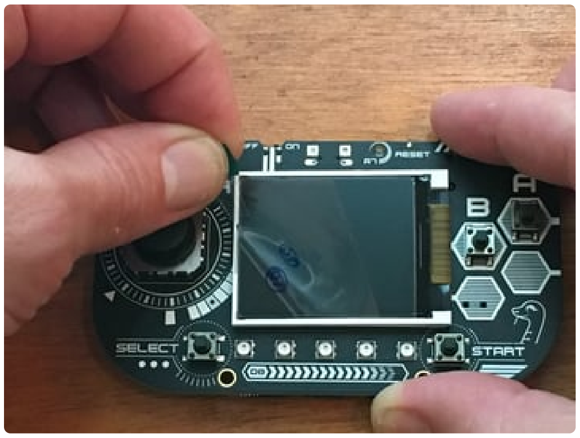
- On the back, there is a **micro SD card slot**. Insert a micro SD card for even more storage for more games, images, sounds, etc.

Build the PyGamer Case



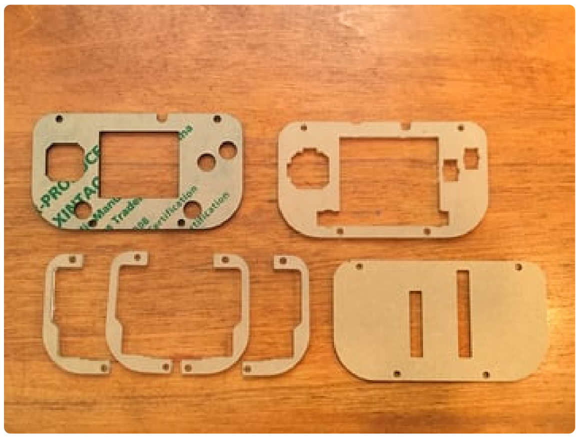
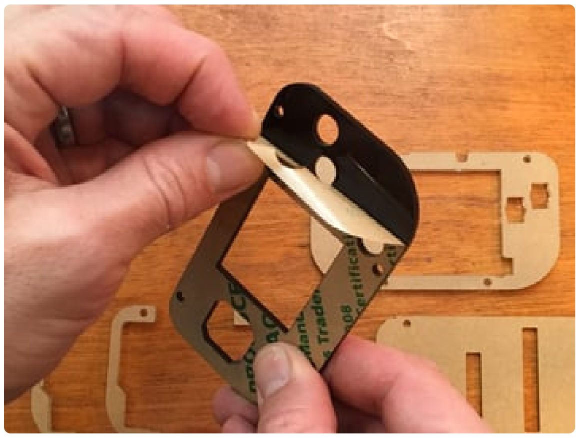
Here's how to assemble the laser cut acrylic case for the PyGamer. The kit comes with seven pieces of acrylic, and four screws and nuts. You've got ten button caps to pick from (you'll pick four), and you'll also want to connect the speaker and battery for the full portable experience.



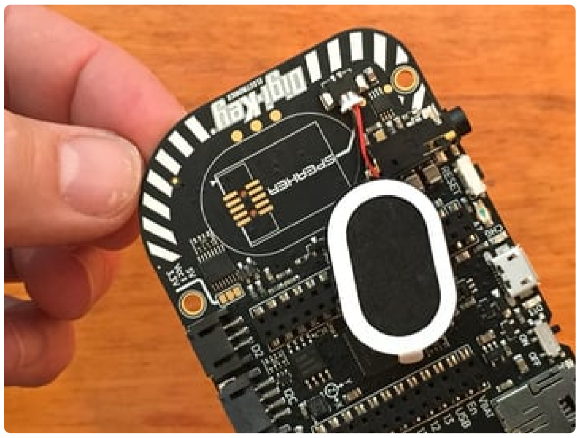
Prep

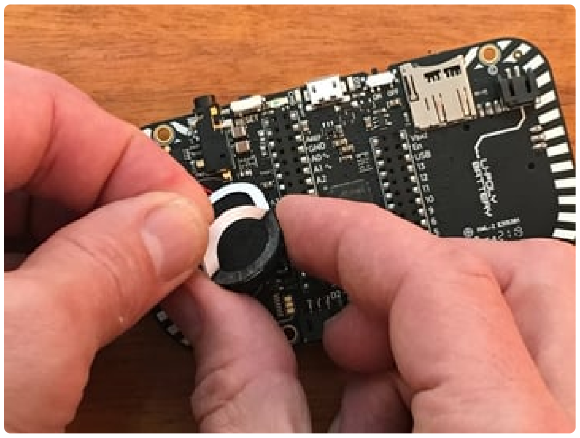
If you haven't already, remove the clear plastic screen protector film from the PyGamer display.

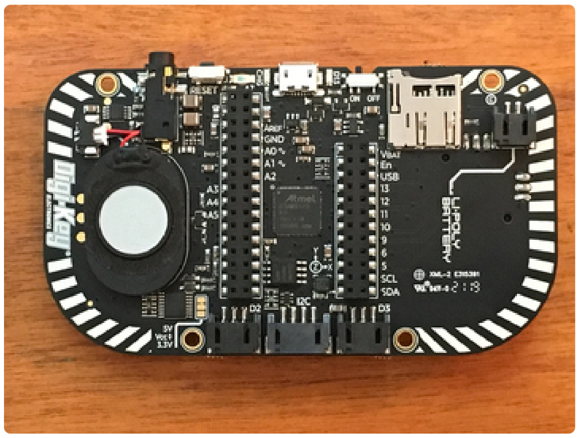



Paper Protection

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



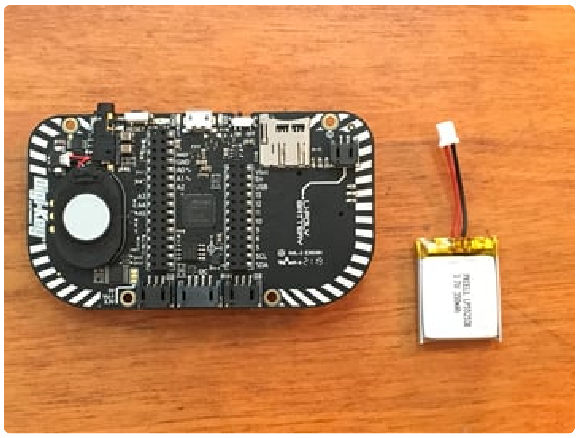
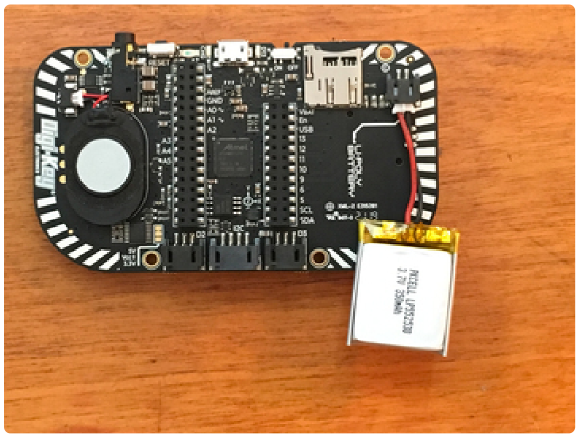
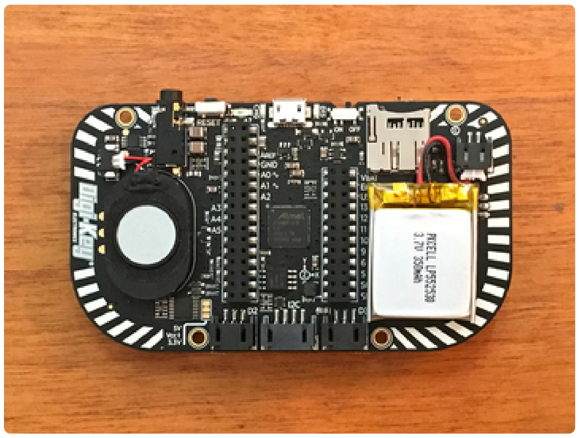




Speaker

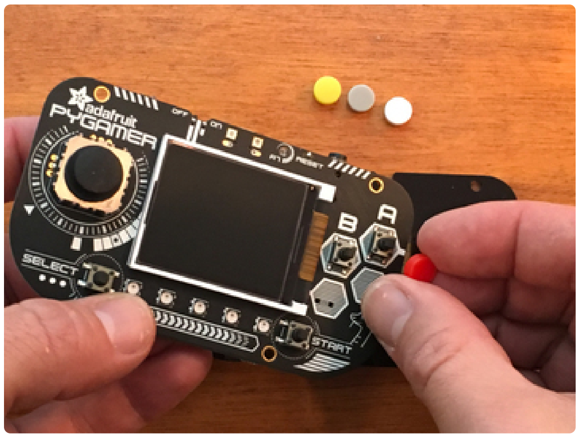
Plug the speaker into the speaker port on the PyGamer.

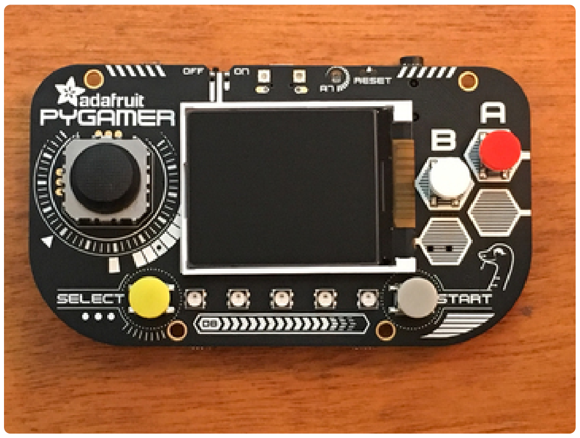
Then, remove the white oval plastic ring to expose the adhesive and press the speaker to the PyGamer where the silkscreen oval outline indicates.

Battery

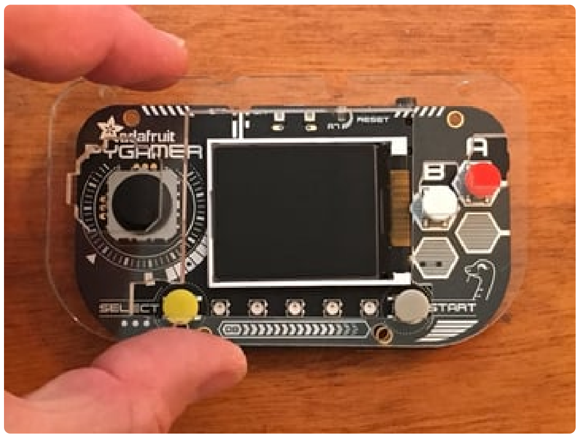
Plug the battery into the on-board connector. Very carefully, bend the wires so that the battery fits the spot shown.

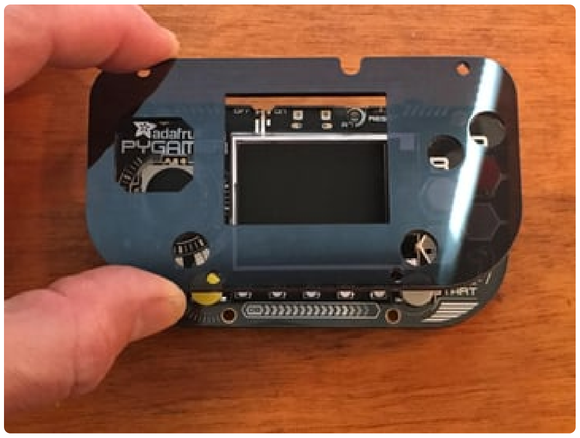




Button Caps

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

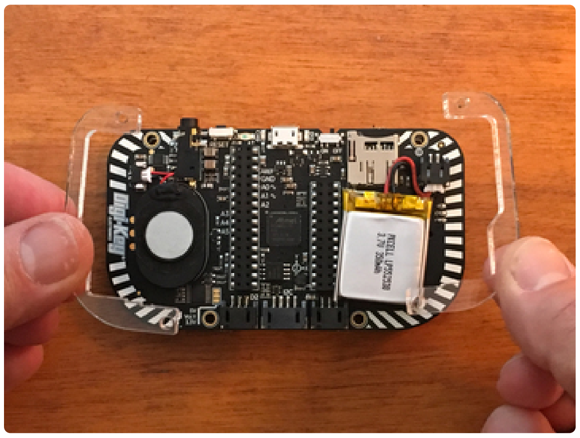
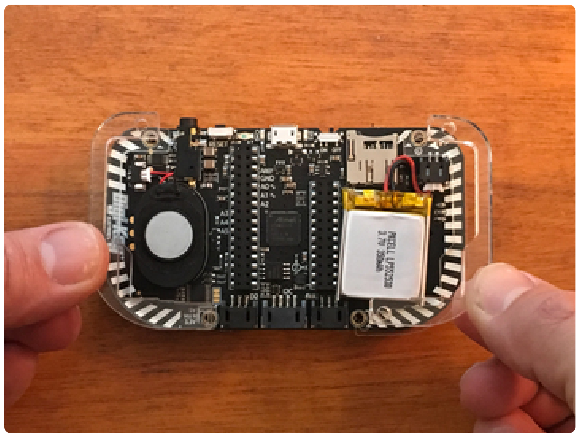




Case Layers

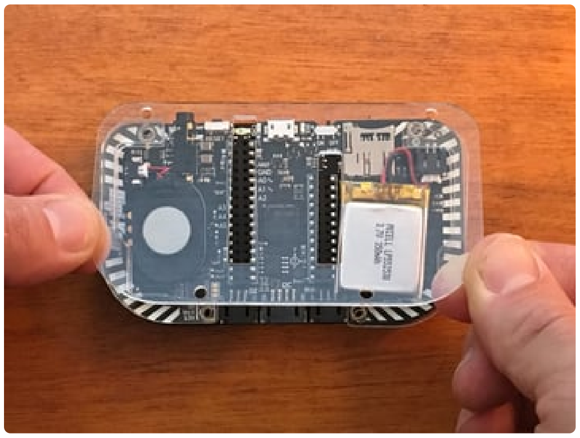
The case assembly is pretty simple. Place the clear top side piece on as shown.

Next, place the smoked gray piece on.

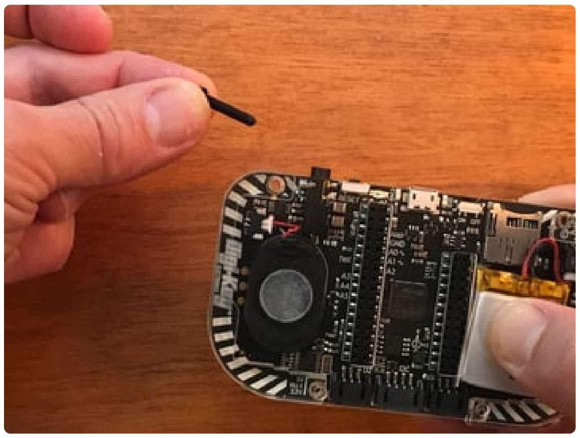
Spacers

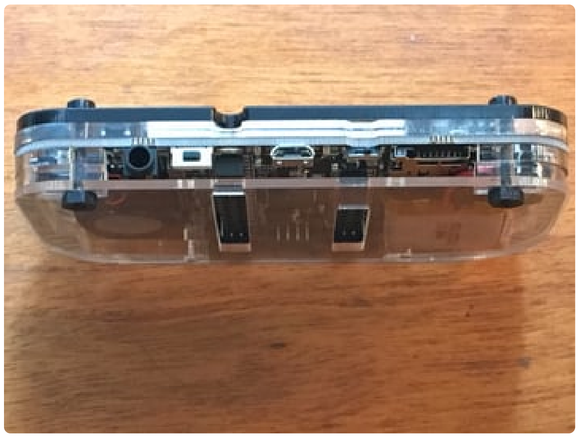
Flip the board over, then place the four spacer pieces onto the back as shown.



Backing

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

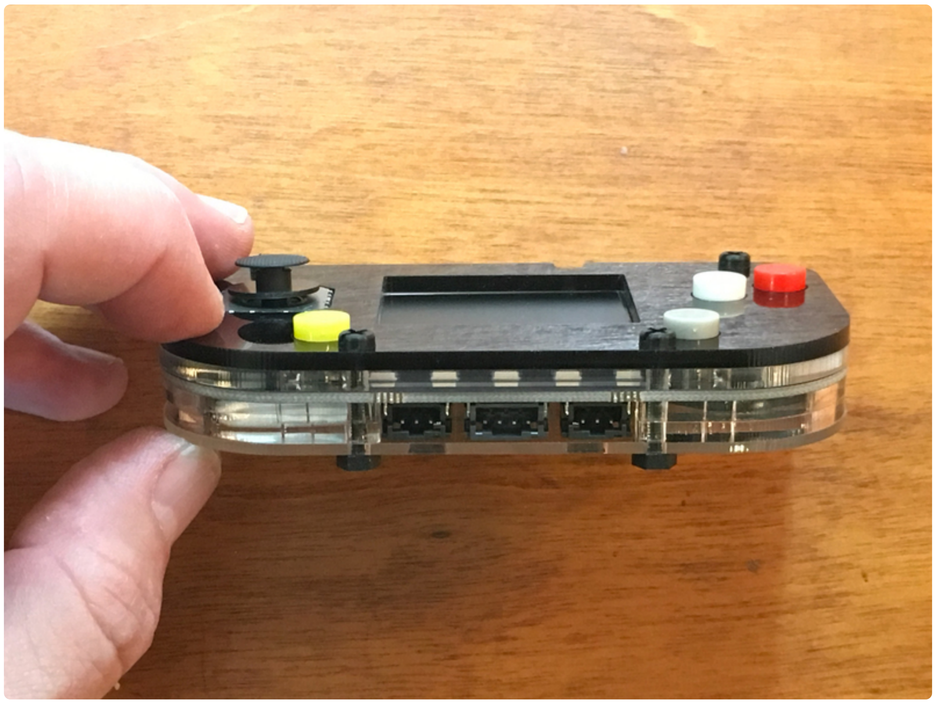




Fasteners

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.



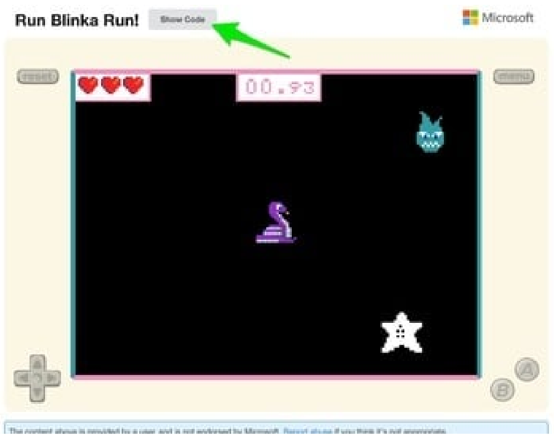
That's all there is to it -- you're ready to play with your PyGamer in its excellent, stylish case!

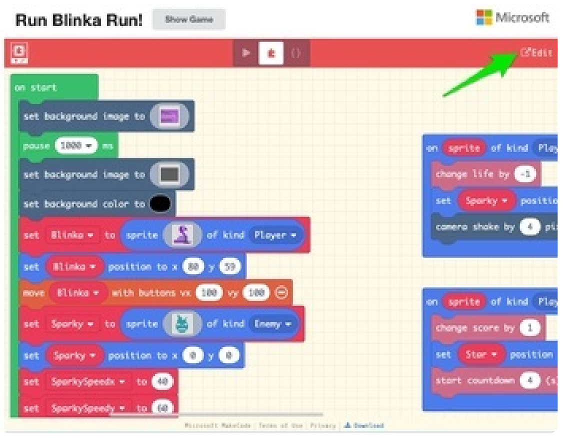
Load a MakeCode Game on PyGamer/PyBadge

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.

https://adafru.it/Fqf

https://adafru.it/Fqf



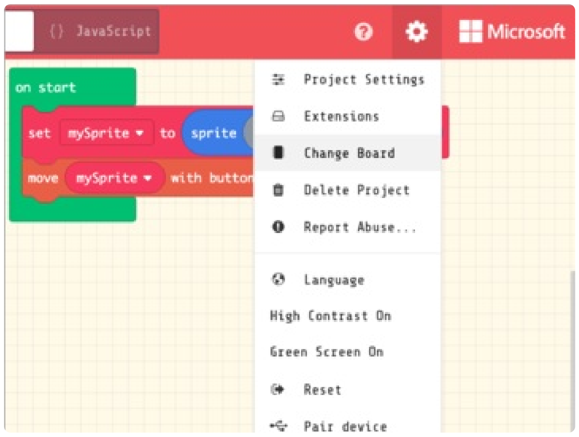


```
on start
    set background image to
    pause 1000 ms
    set background image to
    set background color to
    set Blinka to sprite of kind Player
    set Blinka position to x 80 y 59
    move Blinka with buttons vx 100 vy 100
    set Sparky to sprite of kind Enemy
    set Sparky position to x 0 y 0
    set SparkySpeedx to 40
    set SparkySpeedy to 60

on sprite of kind Play
    change life by -1
    set Sparky positio
    camera shake by 4 pi

on sprite of kind Play
    change score by 1
    set Star position
    start countdown 4 (s
```

Then, click on the Show Code button in the upper left corner. The 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.

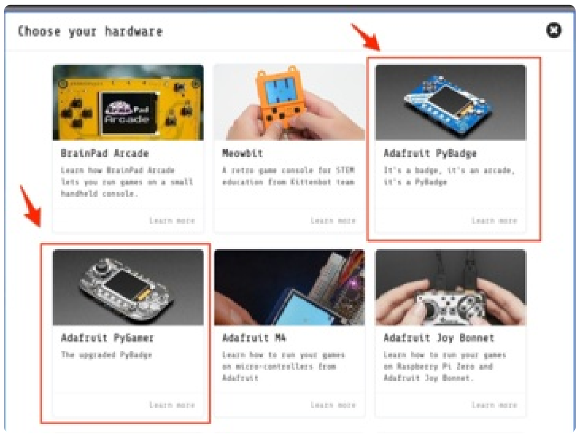
Once you have a game working on the MakeCode Arcade web editor, it's time to download it and flash it onto your board.

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



Board Definition

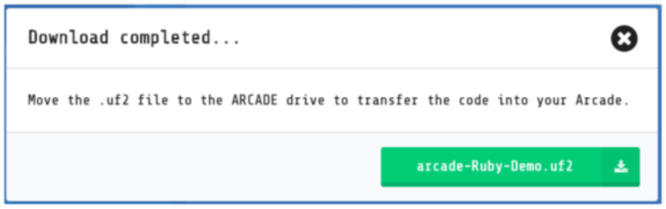
In order to load a game made in MakeCode Arcade onto the PyBadge, first choose the proper board definition inside of MakeCode. Click the gear icon and then the **Change Board** item.


Choose your hardware
BrainPad Arcade
Learn how BrainPad Arcade lets you run games on a small handheld console.
Learn more
Meowbit
A retro game console for STEM education from Kittenbot team
Learn more
Adafruit PyBadge
It's a badge, it's an arcade, it's a PyBadge
Learn more
Adafruit PyGamer
The upgraded PyBadge
Learn more
Adafruit M4
Learn how to run your games on micro-controllers from Adafruit
Learn more
Adafruit Joy Bonnet
Learn how to run your games on Raspberry Pi Zero and Adafruit Joy Bonnet.
Learn more

Change Board screen

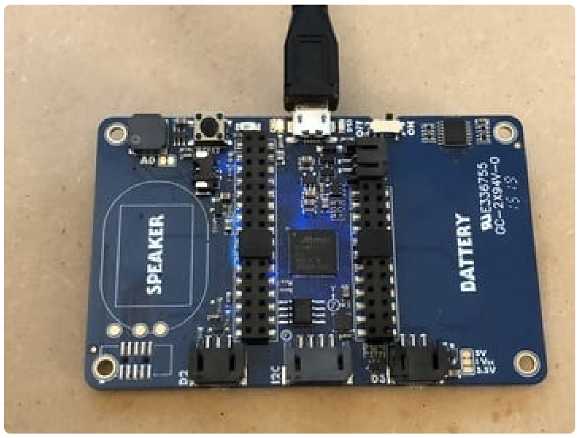
Click on the image of your board, either the PyBadge/PyBadge LC or the PyGamer

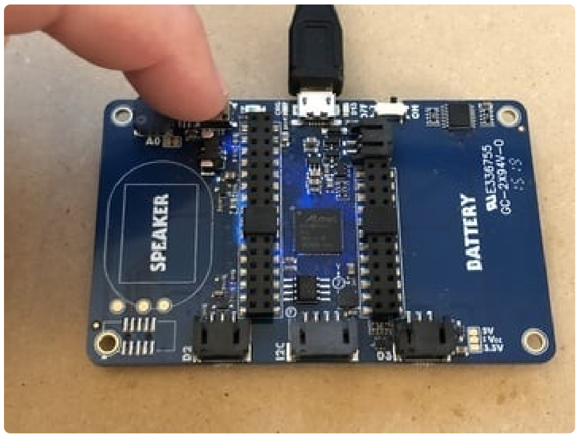
This will cause the game .uf2 file for your particular board to be saved to your hard drive. You only need to do this the first time you use a new board. Thereafter you can simply click the **Download** button on the MakeCode Arcade editor page.

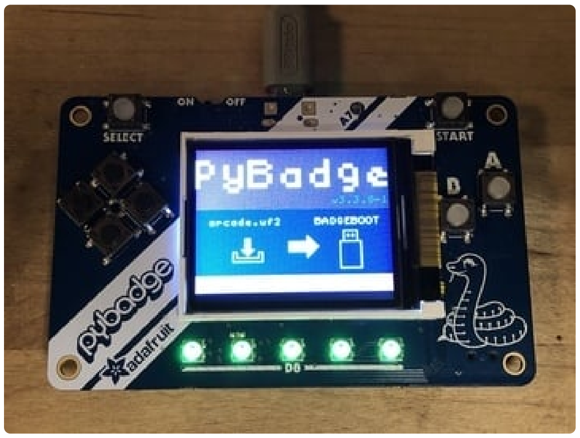

Download completed...

Move the .uf2 file to the ARCADE drive to transfer the code into your Arcade.

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!





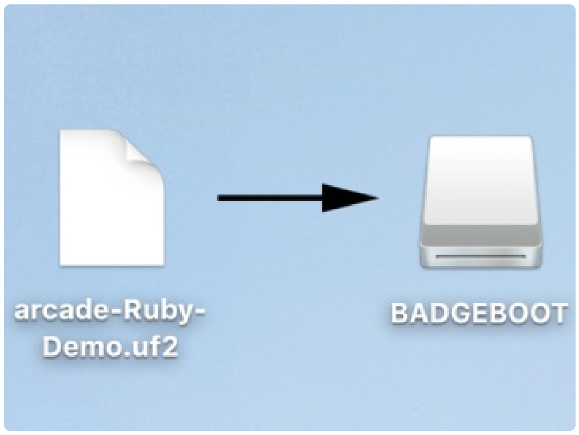


Bootloader Mode

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.

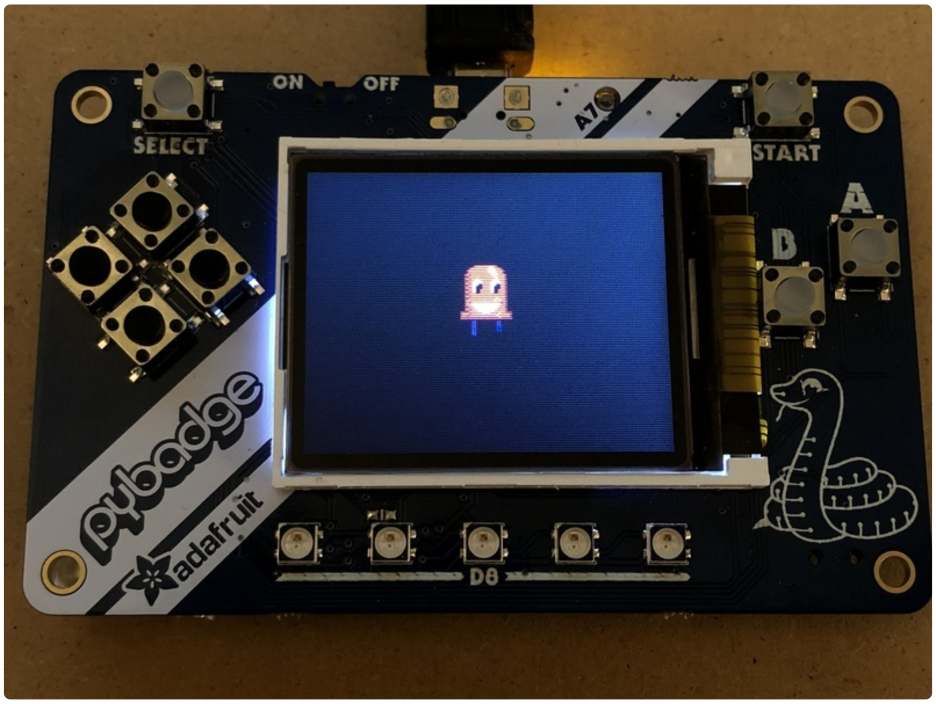


Drag and Drop

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 for additional game related content.

CircuitPython

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

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

Set up CircuitPython Quick Start!

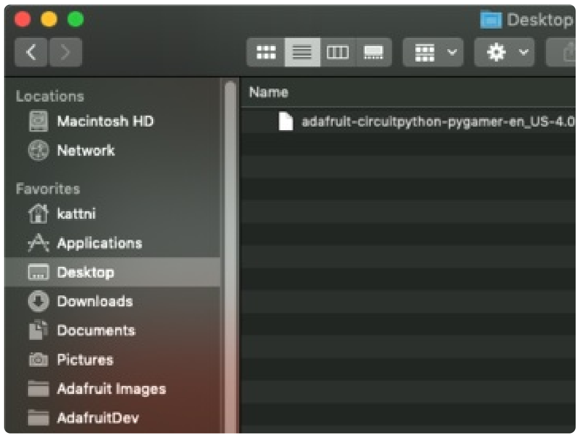
Follow this quick step-by-step for super-fast Python power :)

https://adafru.it/FxM

https://adafru.it/FxM

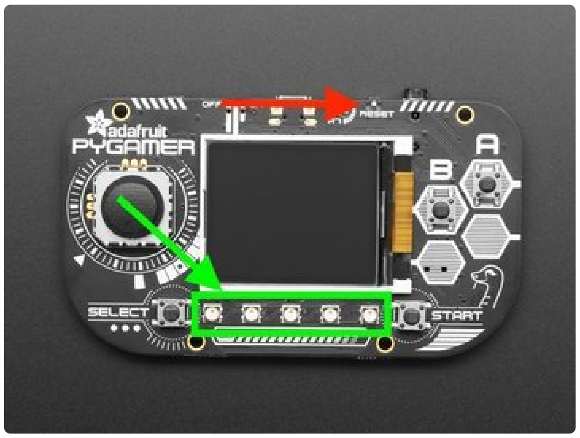
Further Information

For more detailed info on installing CircuitPython, check out Installing CircuitPython (https://adafru.it/Amd).



Click the link above and download the latest UF2 file.

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

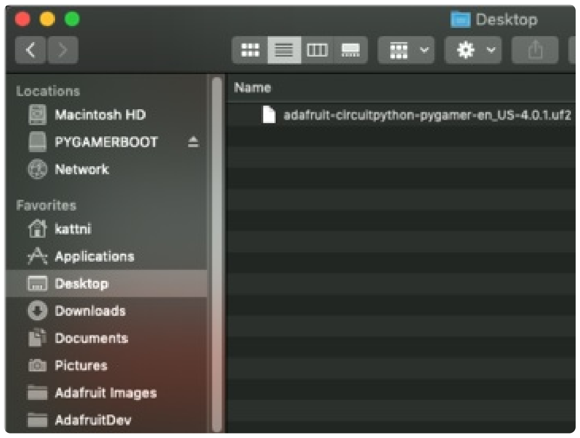


Plug your PyGamer into your computer using a known-good USB cable.

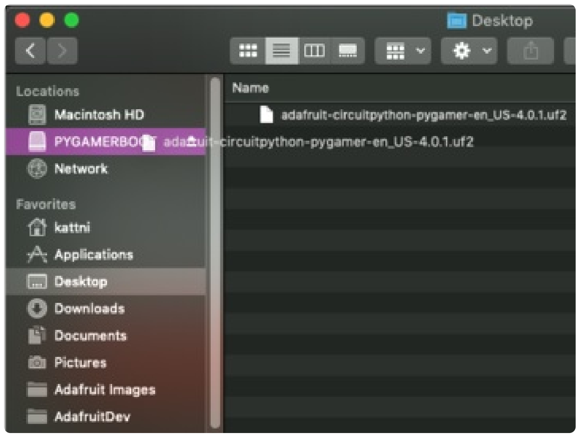
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

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.

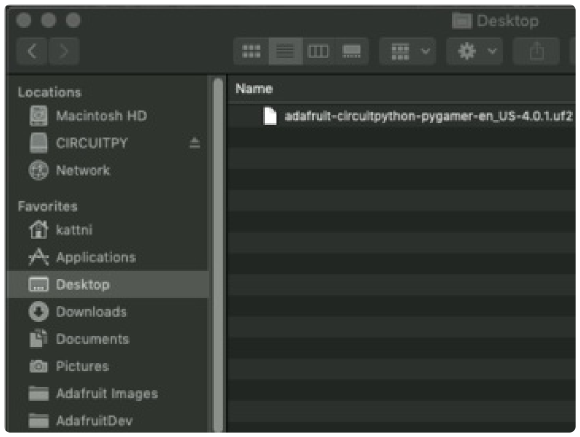
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



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.

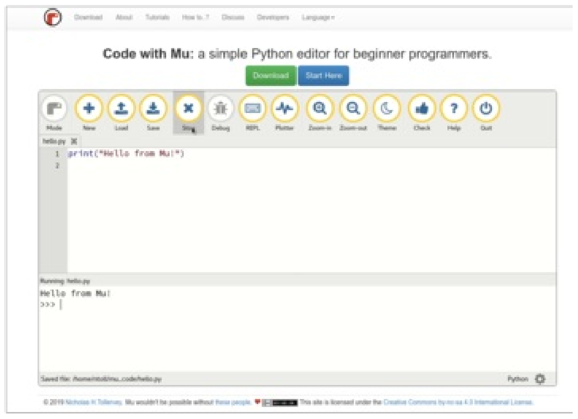
That's it, you're done! :)

Installing Mu Editor

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

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

Download and Install Mu



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

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

Running: hello.py

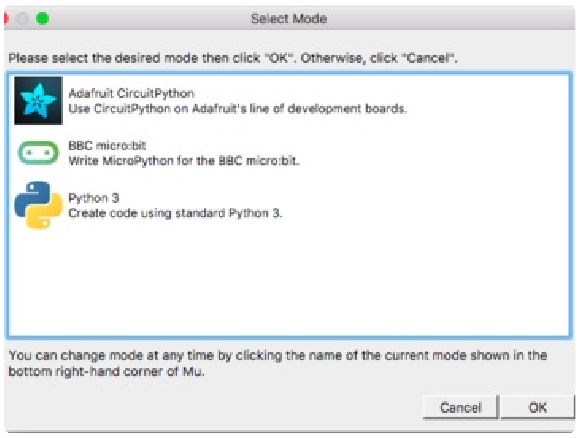
```
Hello from Mu!
>>> |
```

Saved file: /home/ntoll/mu_code/hello.py

© 2019 Nicholas H.Tollervey. Mu wouldn't be possible without these people. This site is licensed under the Creative Commons by-nc-sa 4.0 International License.

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

Using Mu



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

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

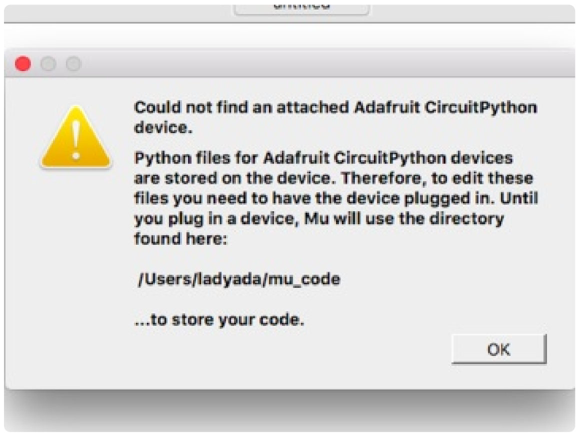
BBC micro:bit
Write MicroPython for the BBC micro:bit.

Python 3
Create code using standard Python 3.

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

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

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


Could not find an attached Adafruit CircuitPython device.

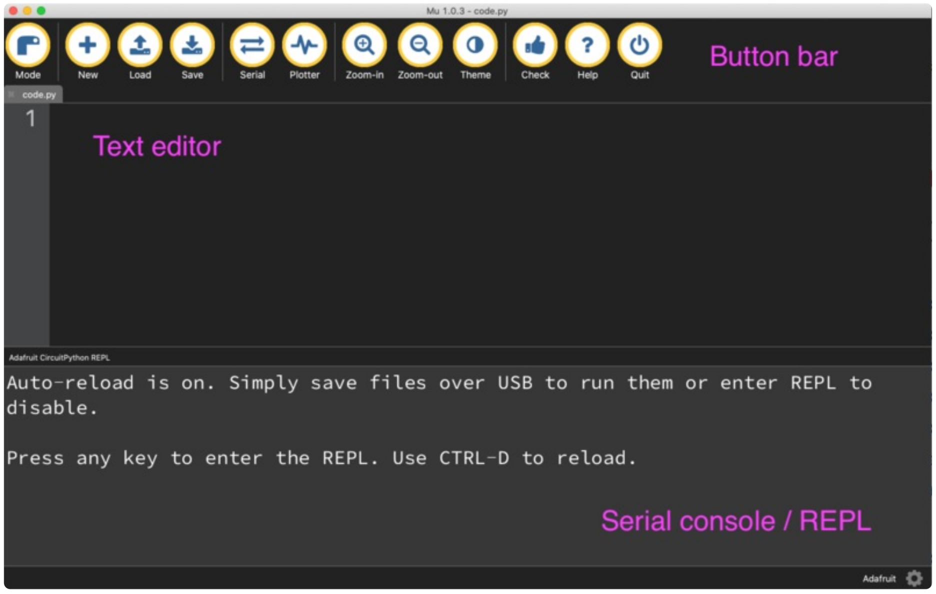
Python files for Adafruit CircuitPython devices are stored on the device. Therefore, to edit these files you need to have the device plugged in. Until you plug in a device, Mu will use the directory found here:

/Users/ladyada/mu_code

...to store your code.

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

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



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

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

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

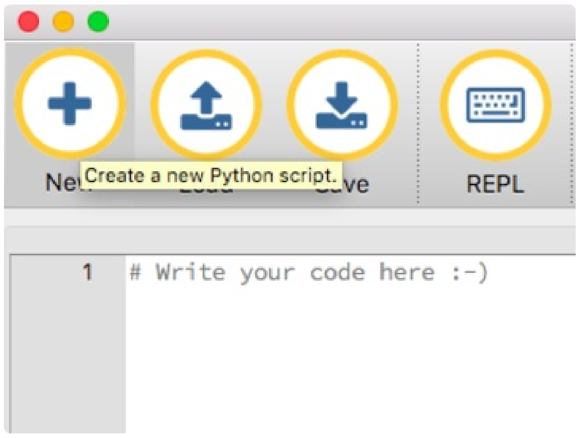
Creating and Editing Code

One of the best things about CircuitPython is how simple it is to get code up and running. In this section, we're going to cover how to create and edit your first CircuitPython program.

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

If you don't or can't use Mu, there are basic text editors built into every operating system such as Notepad on Windows, TextEdit on Mac, and gedit on Linux. However, many of these editors don't write back changes immediately to files that you edit. That can cause problems when using CircuitPython. See the Editing Code (https://adafru.it/id3) section below. If you want to skip that section for now, make sure you do "Eject" or "Safe Remove" on Windows or "sync" on Linux after writing a file if you aren't using Mu. (This is not a problem on MacOS.)

Creating Code



```
# Write your code here :-)
```

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

Copy and paste the following code into your editor:

```
import board
import digitalio
import time

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

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

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

If you're using QT Py or a Trinkey, please download the NeoPixel blink example (https://adafru.it/PE0).

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

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

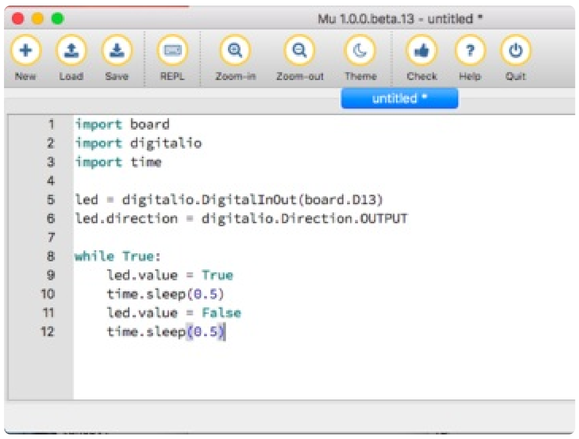
For Adafruit CLUE, you'll need to use `board.D17` instead of `board.LED`. The rest of the code remains the same. Make the following change to the `led =` line:

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

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

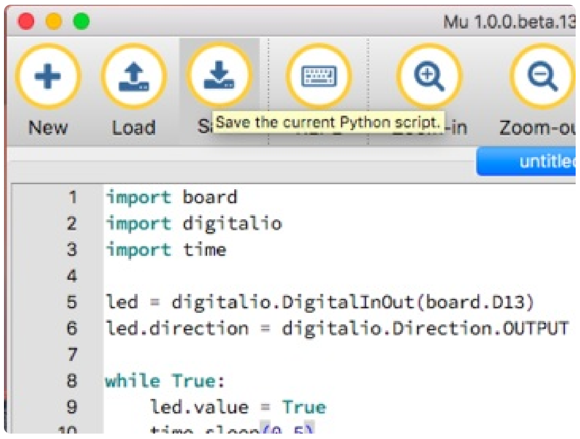
For Adafruit ItsyBitsy nRF52840, you'll need to use `board.BLUE_LED` instead of `board.LED`. The rest of the code remains the same. Make the following change to the `led =` line:

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

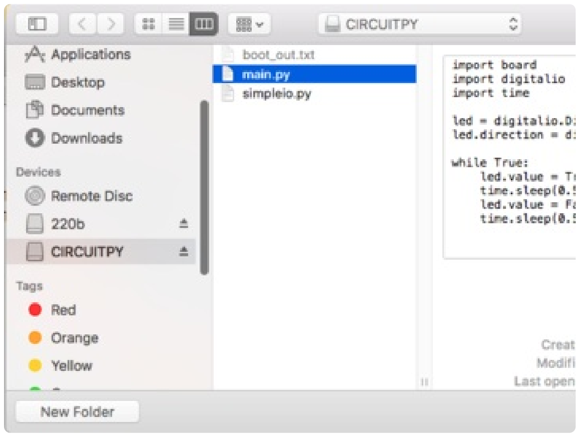


```
import board
import digitalio
import time

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

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

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



```
import board
import digitalio
import time

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

while True:
    led.value = True
```



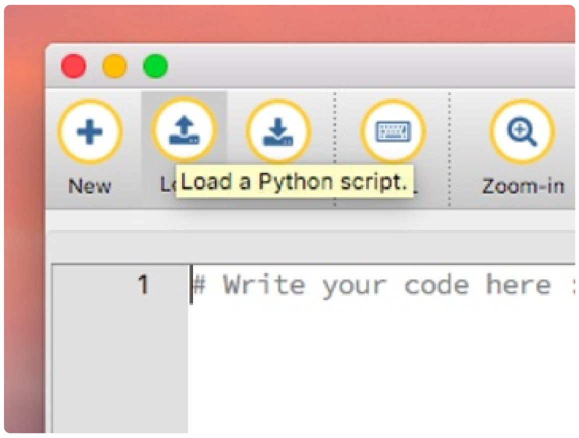
Save this file as **code.py** on your CIRCUITPY drive.

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

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

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

Editing Code



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

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

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

There's just one warning we have to give you before we continue...

Don't Click Reset or Unplug!

The CircuitPython code on your board detects when the files are changed or written and will automatically re-start your code. This makes coding very fast because you save, and it re-runs.

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

This is really important to be aware of. If you unplug or reset the board before your computer finishes writing the file to your board, you can corrupt the drive. If this happens, you may lose the code you've written, so it's important to backup your code to your computer regularly.

There are a few ways to avoid this:

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

Recommended editors:

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

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

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

We *don't* recommend these editors:

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

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

2. Eject or Sync the Drive After Writing

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

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

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

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

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

Back to Editing Code...

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

```
import board
import digitalio
import time

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

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

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

Exploring Your First CircuitPython Program

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

Here is the original code again:

```
import board
import digitalio
import time

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

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

Imports & Libraries

Each CircuitPython program you run needs to have a lot of information to work. The reason CircuitPython is so simple to use is that most of that information is stored in other files and works in the background. The files built into CircuitPython are called **modules**, and the files you load separately are called **libraries**. Modules are built into CircuitPython. Libraries are stored on your CIRCUITPY drive in a folder called **lib**.

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

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

Setting Up The LED

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

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

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

Loop-de-loops

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

Inside our loop, we have four items:

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

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

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

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

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

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

What Happens When My Code Finishes Running?

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

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

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

What if I don't have the loop?

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

The simplest loop would be:

```
while True:
    pass
```

And remember - you can press to exit the loop.

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

More Changes

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

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

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

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

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

Naming Your Program File

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

Connecting to the Serial Console

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

`print("Hello, world!")`

This line would result in:

`Hello, world!`

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

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

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

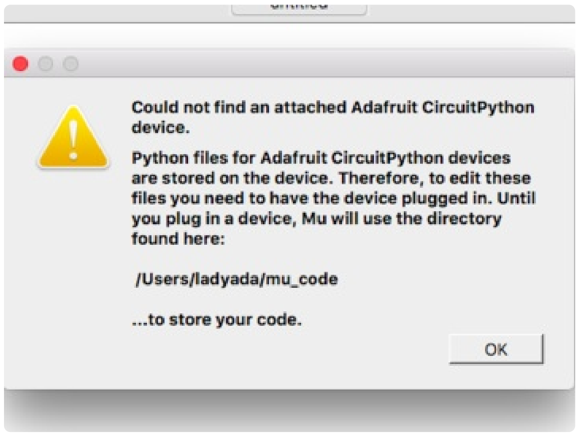
If you're on Linux, and are seeing multi-second delays connecting to the serial console, or are seeing "AT" and other gibberish when you connect, then the modemmanager service might be interfering. Just remove it; it doesn't have much use unless you're still using dial-up modems. To remove, type this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

```
sudo apt purge modemmanager
```

Are you using Mu?

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

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



Could not find an attached Adafruit CircuitPython device.

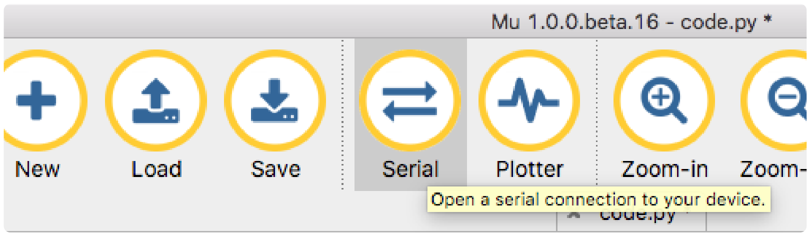
Python files for Adafruit CircuitPython devices are stored on the device. Therefore, to edit these files you need to have the device plugged in. Until you plug in a device, Mu will use the directory found here:

/Users/ladyada/mu_code

...to store your code.

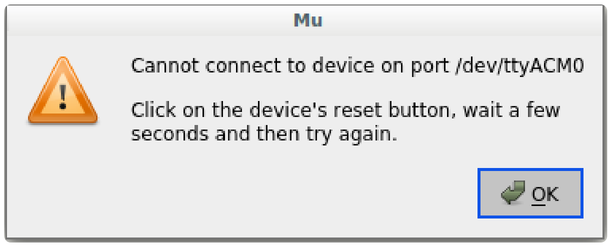
First, make sure your CircuitPython board is plugged in. If you are using Windows 7, make sure you installed the drivers (https://adafru.it/Amd).

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


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

Setting Permissions on Linux

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



Mu

Cannot connect to device on port /dev/ttyACM0

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

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

```
sudo adduser $USER dialout
```

After running the command above, reboot your machine to gain access to the group. On other Linux distributions, the group you need may be different. See Advanced Serial Console on Mac and

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

Using Something Else?

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

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

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

Interacting with the Serial Console

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

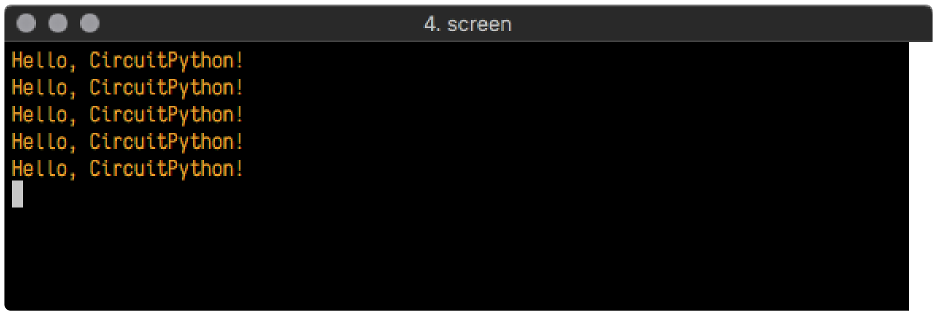
The code you wrote earlier has no output to the serial console. So, we're going to edit it to create some output.

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

```
import board
import digitalio
import time

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

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

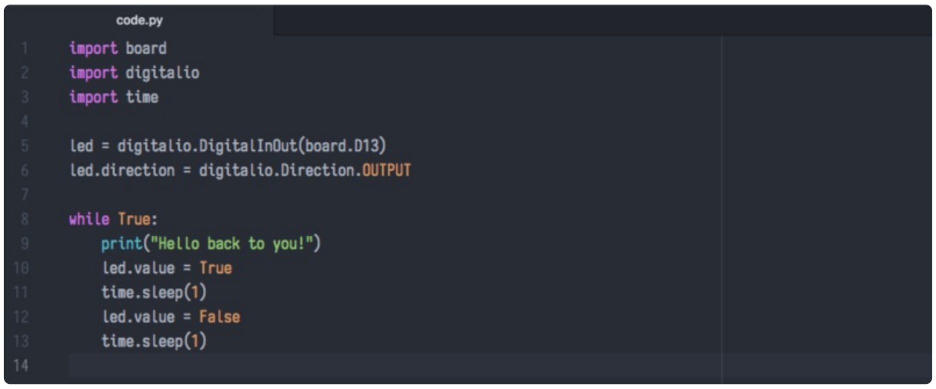
Save your file.

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

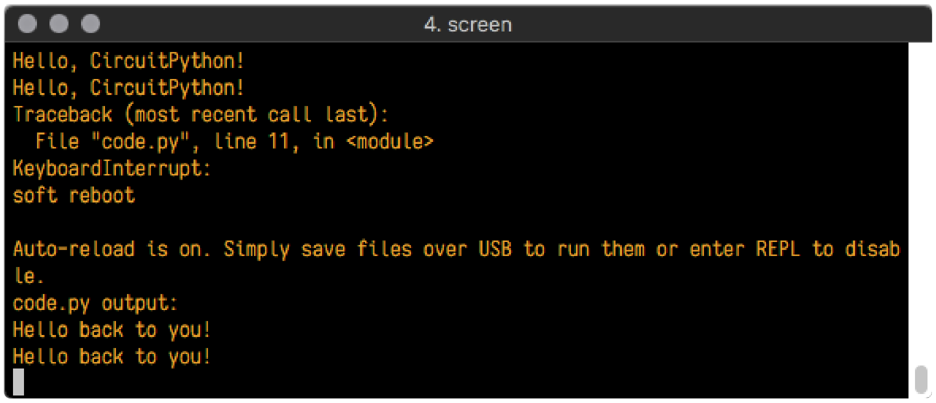


```
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
```

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

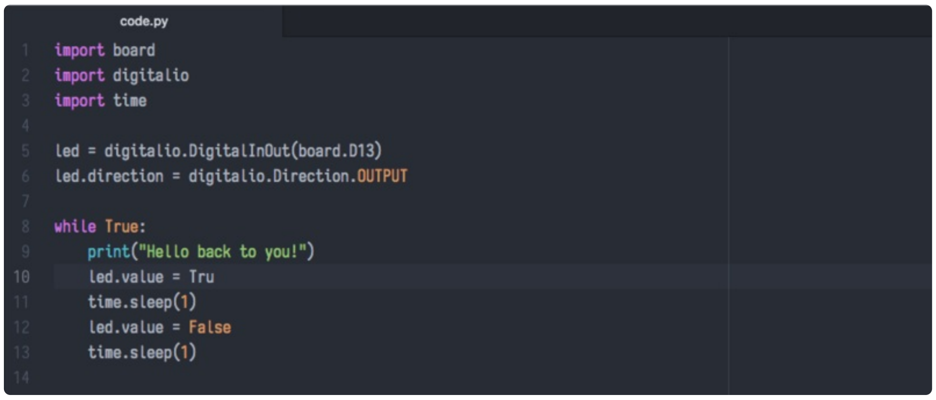

code.py
```python
import board
import digitalio
import time

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

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

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


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 disab
le.
code.py output:
Hello back to you!
Hello back to you!
```

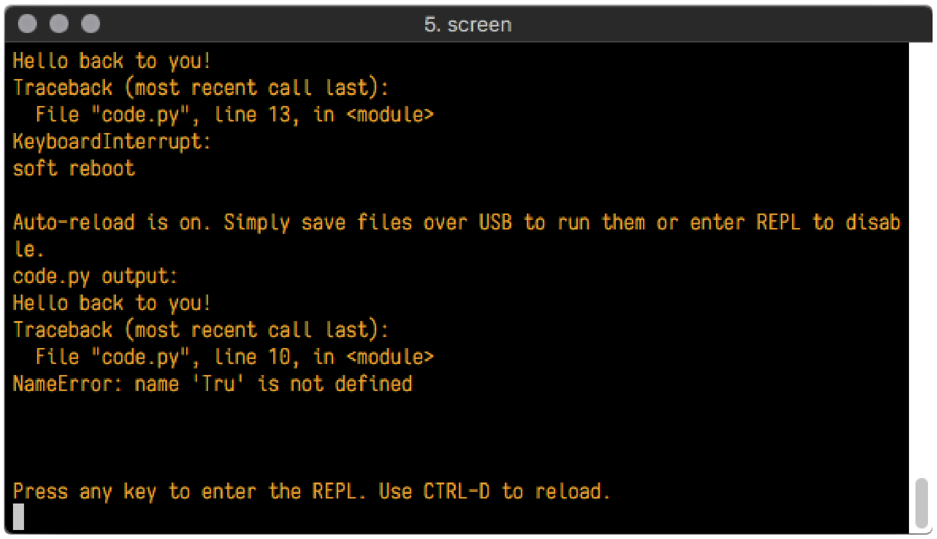
The `Traceback (most recent call last):` is telling you the last thing your board was doing before you saved your file. This is normal behavior and will happen every time the board resets. This is really handy for troubleshooting. Let's introduce an error so we can see how it is used.

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


code.py
```python
import board
import digitalio
import time

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

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

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

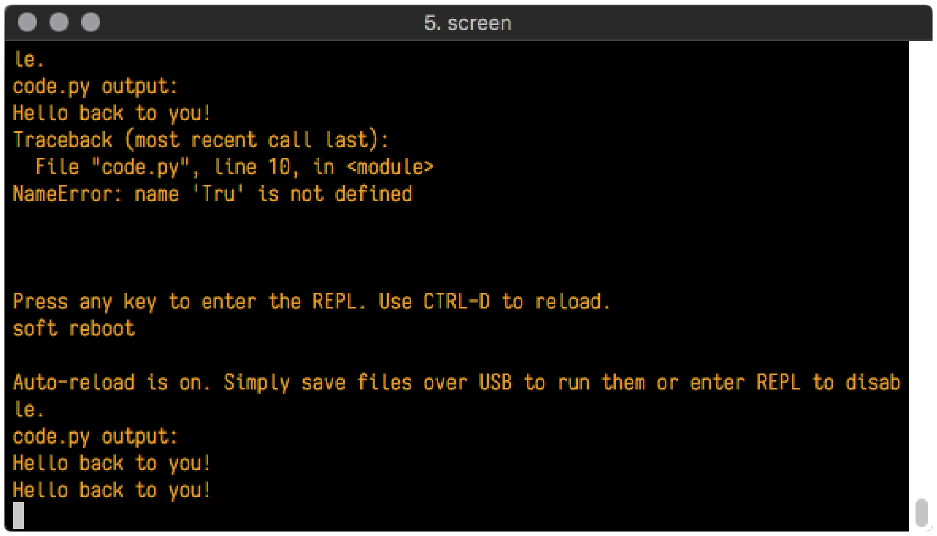
Usually when you run into errors, it's not because you introduced them on purpose. You may have 200 lines of code, and have no idea where your error could be hiding. This is where the serial console can help. Let's take a look!



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

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



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

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

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



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



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

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

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

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

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

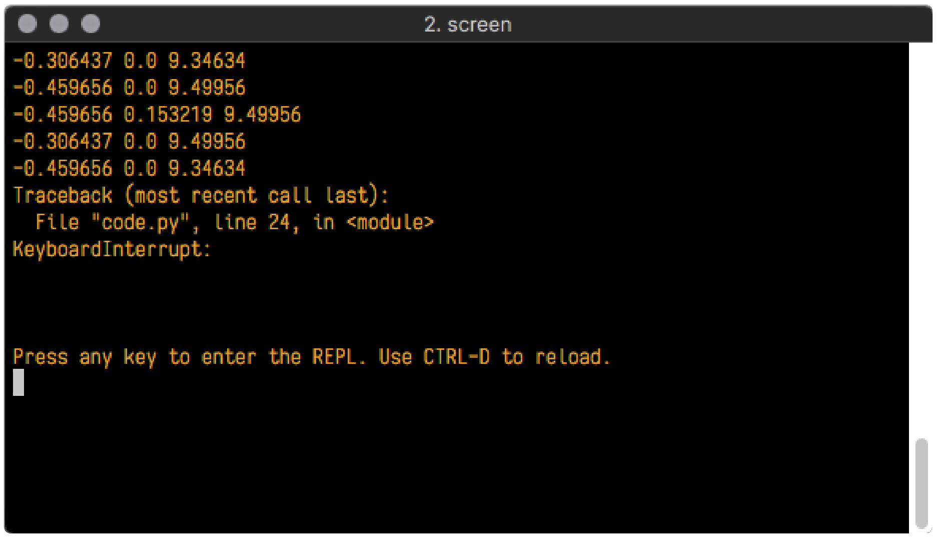
The REPL

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

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

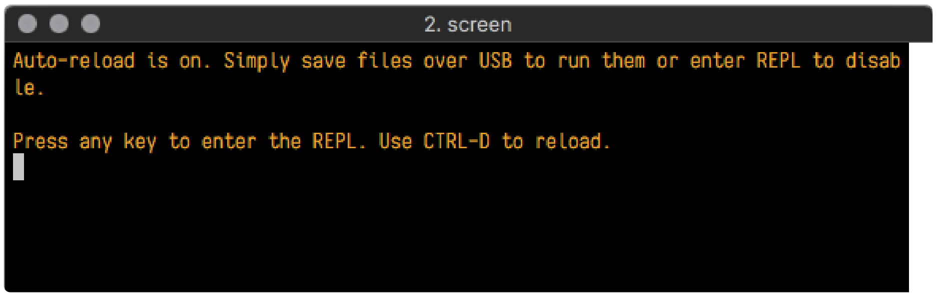
If there is code running, it will stop and you'll see `Press any key to enter the REPL. Use CTRL-D to reload.` Follow those instructions, and press any key on your keyboard.

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



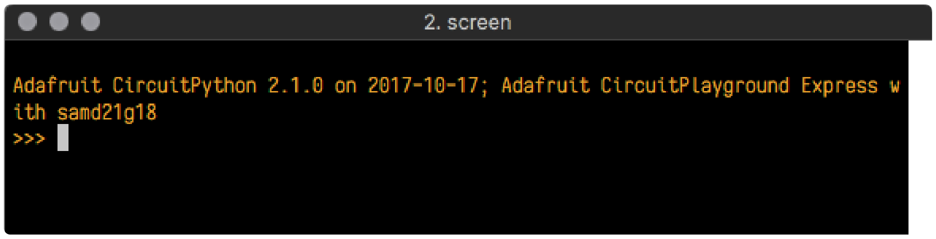
```
-0.306437 0.0 9.34634
-0.459656 0.0 9.49956
-0.459656 0.153219 9.49956
-0.306437 0.0 9.49956
-0.459656 0.0 9.34634
Traceback (most recent call last):
  File "code.py", line 24, in <module>
KeyboardInterrupt:



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

If there is no code running, you will enter the REPL immediately after pressing Ctrl + C. There is no information about what your board was doing before you interrupted it because there is no code running.



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

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

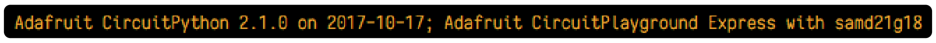
Either way, once you press a key you'll see a `>>>` prompt welcoming you to the REPL!


2. screen

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

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

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



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

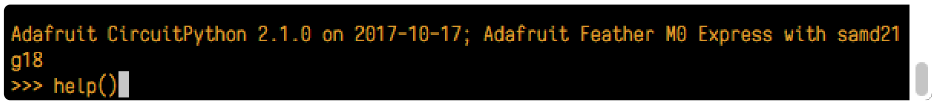
This line tells you the version of CircuitPython you're using and when it was released. Next, it gives you the type of board you're using and the type of microcontroller the board uses. Each part of this may be different for your board depending on the versions you're working with.

This is followed by the CircuitPython prompt.

```
>>>
```

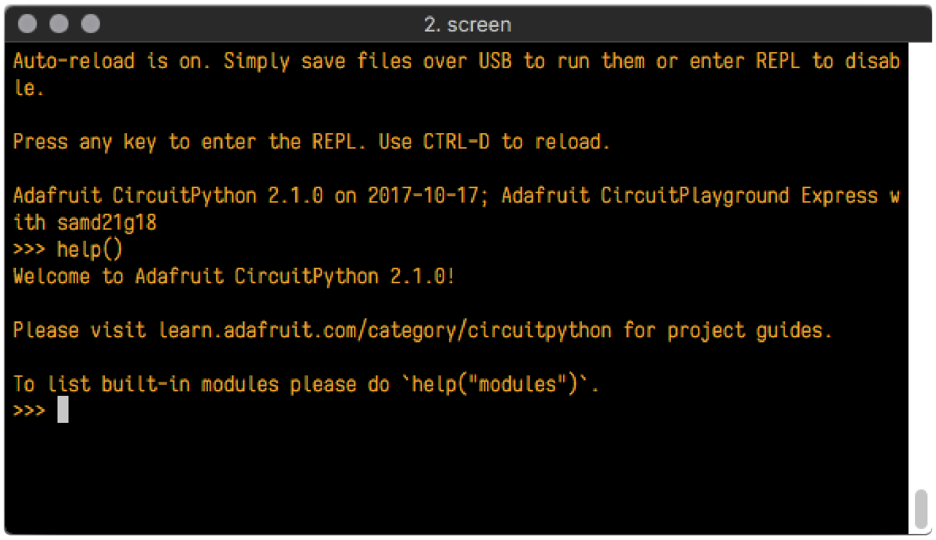
From this prompt you can run all sorts of commands and code. The first thing we'll do is run `help()`. This will tell us where to start exploring the REPL. To run code in the REPL, type it in next to the REPL prompt.

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



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

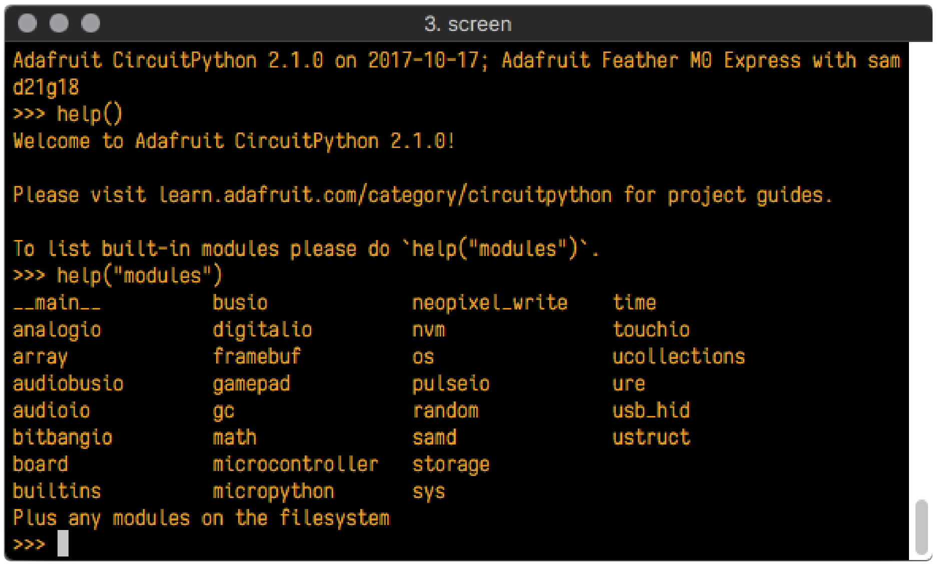
Then press enter. You should then see a message.


2. screen

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

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

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

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

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

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

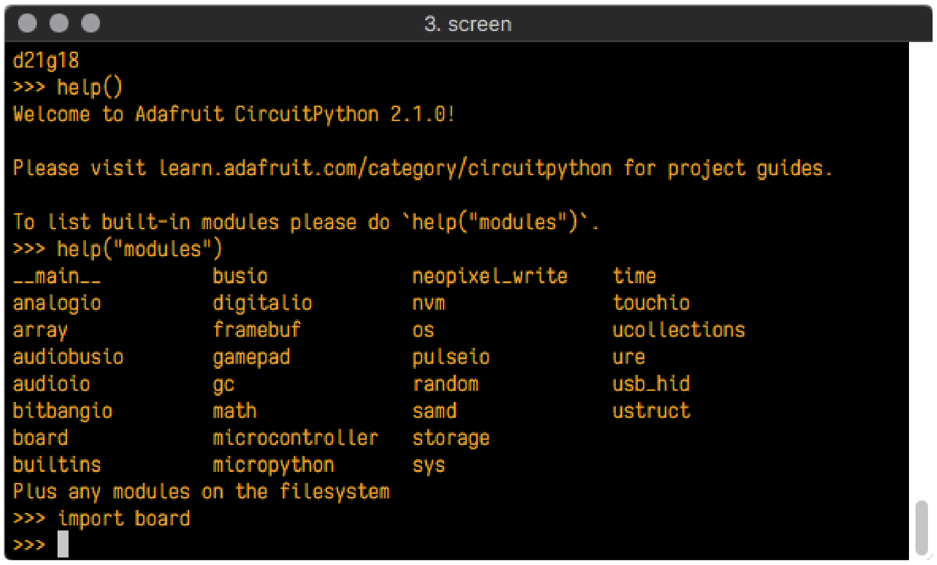
Type `help("modules")` into the REPL next to the prompt, and press enter.



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

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

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

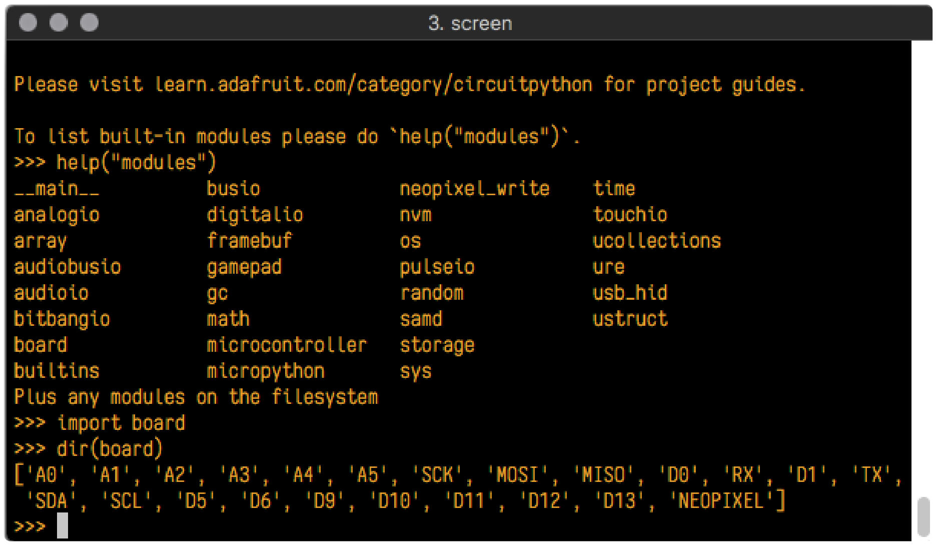
This is a list of all the core libraries built into CircuitPython. We discussed how board contains all of the pins on the board that you can use in your code. From the REPL, you are able to see that list!

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



```
3. screen
d21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

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

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

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



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

To list built-in modules please do `help("modules")`.
>>> help("modules")
__main__          busio             neopixel_write    time
analogio          digitalio         nvm               touchio
array             framebuf          os                ucollections
audiobusio        gamepad           pulseio           ure
audioio           gc                random            usb_hid
bitbangio         math              samd              ustruct
board             microcontroller   storage
builtins          micropython       sys
Plus any modules on the filesystem
>>> import board
>>> dir(board)
['A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'SCK', 'MOSI', 'MISO', 'D0', 'RX', 'D1', 'TX',
 'SDA', 'SCL', 'D5', 'D6', 'D9', 'D10', 'D11', 'D12', 'D13', 'NEOPIXEL']
>>> 
```

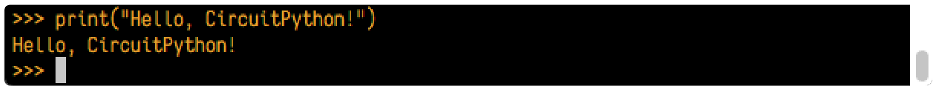
This is a list of all of the pins on your board that are available for you to use in your code. Each board's list will differ slightly depending on the number of pins available. Do you see D13 ? That's the pin you used to blink the red LED!

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

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

`print("Hello, CircuitPython!")`

Then press enter.



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

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

You can write single lines of code that run stand-alone. You can also write entire programs into the REPL to test them. As we said though, remember that nothing typed into the REPL is saved.

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

Try typing more into the REPL to see what happens!

Returning to the serial console

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

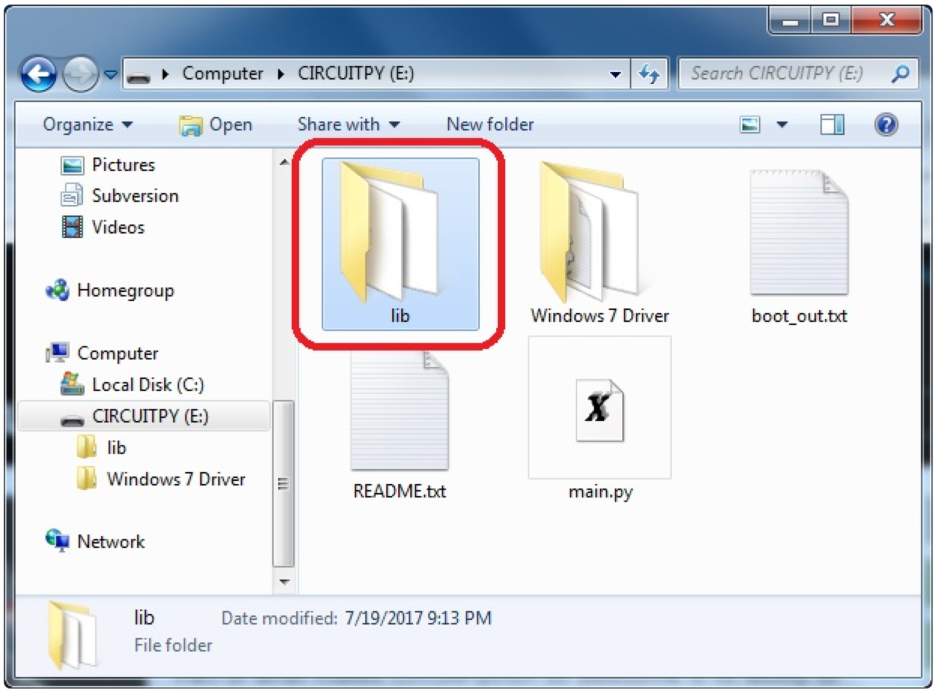
You can return to the REPL at any time!

CircuitPython Libraries

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

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

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



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

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

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

Installing the CircuitPython Library Bundle

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

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

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

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

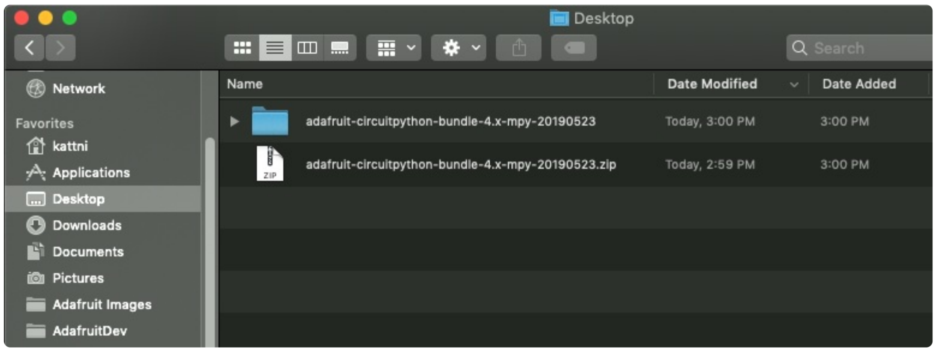
https://adafru.it/ENC

https://adafru.it/ENC

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

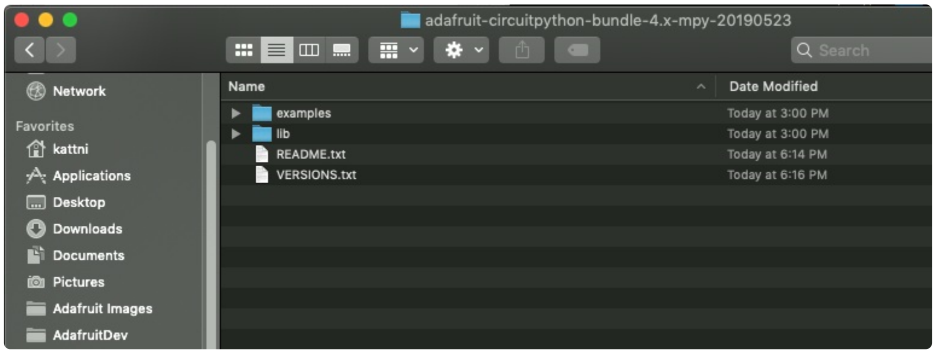
Either way, download the version that matches your CircuitPython firmware version. If you don't know the version, look at the initial prompt in the CircuitPython REPL, which reports the version. For example, if you're running v4.0.1, download the 4.x library bundle. There's also a **py** bundle which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

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



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

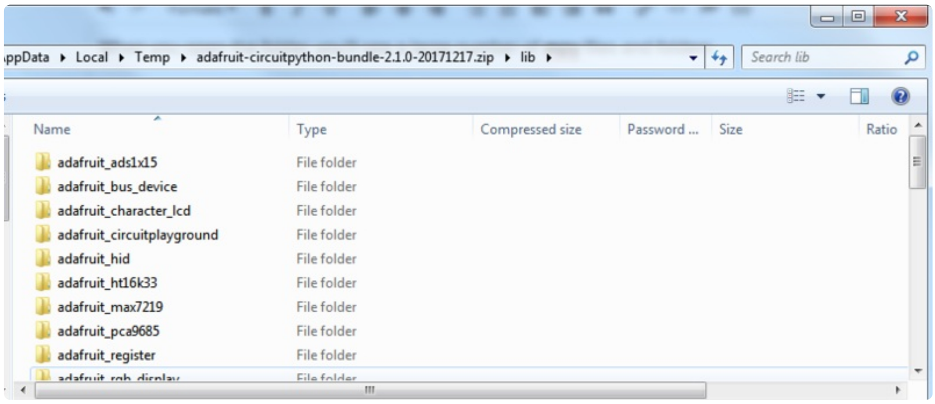
Open the bundle folder. Inside you'll find two information files, and two folders. One folder is the lib bundle, and the other folder is the examples bundle.


adafruit-circuitpython-bundle-4.x-mpy-20190523

Network
Favorites
kattni
Applications
Desktop
Downloads
Documents
Pictures
Adafruit Images
AdafruitDev

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

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

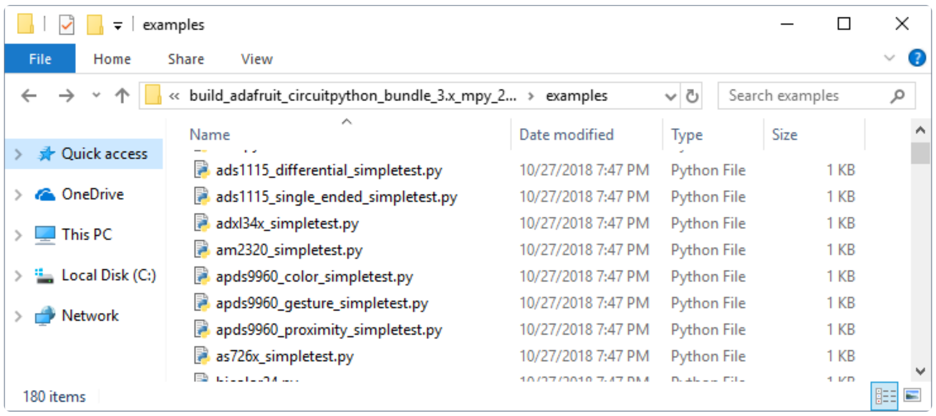

ppData ▸ Local ▸ Temp ▸ adafruit-circuitpython-bundle-2.1.0-20171217.zip ▸ lib ▸

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

Example Files

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

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


examples

File Home Share View

« build_adafruit_circuitpython_bundle_3.x_mpy_2... › examples

Quick access
OneDrive
This PC
Local Disk (C:)
Network

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

180 items

Copying Libraries to Your Board

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

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

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

Example: `ImportError` Due to Missing Library

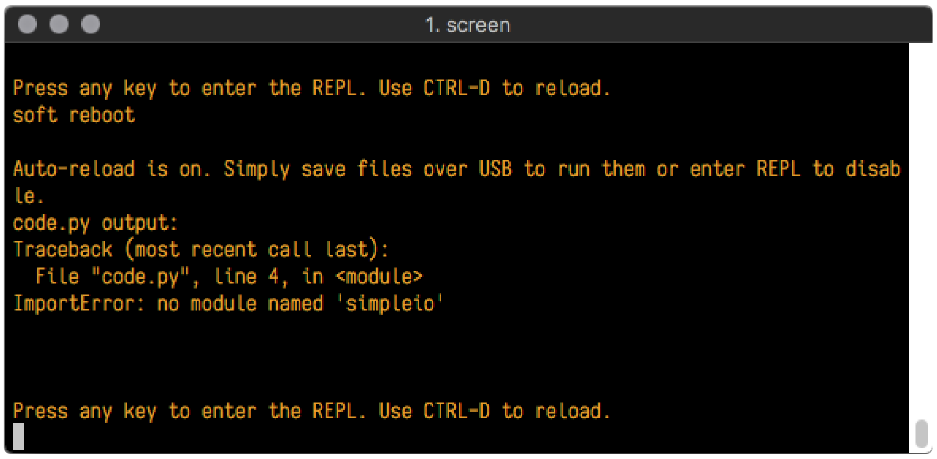
If you choose to load libraries as you need them, you may write up code that tries to use a library you haven't yet loaded. We're going to demonstrate what happens when you try to utilise a library that you don't have loaded on your board, and cover the steps required to resolve the issue.

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

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

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.D13)

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

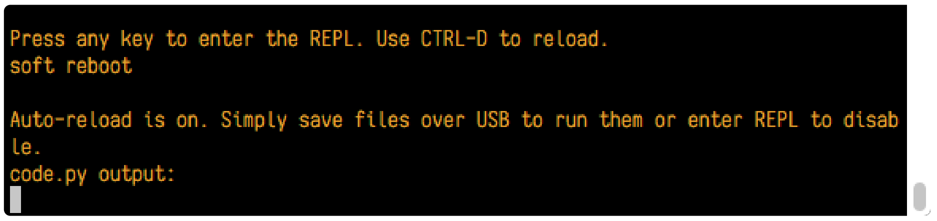
Save this file. Nothing happens to your board. Let's check the serial console to see what's going on.


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

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



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

We have an `ImportError`. It says there is `no module named 'simpleio'`. That's the one we just included in our code!

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

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


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

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

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

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

Library Install on Non-Express Boards

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

You may end up running out of space on your Trinket M0 or Gemma M0 even if you only load libraries as you need them. There are a number of steps you can use to try to resolve this issue. You'll find them in the Troubleshooting page in the Learn guides for your board.

Updating CircuitPython Libraries/Examples

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

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

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

CircuitPython Pins and Modules

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

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

CircuitPython Pins

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

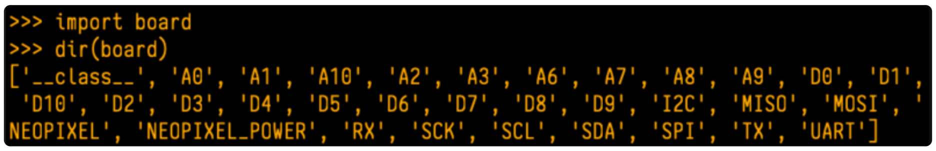
`import board`

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

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

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

Here is the output for the QT Py.



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

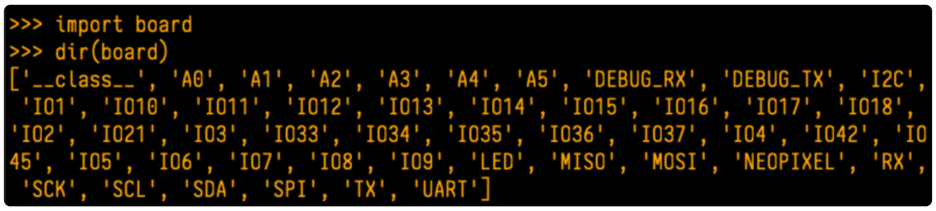
The following pins have labels on the physical QT Py board: A0, A1, A2, A3, SDA, SCL, TX, RX, SCK, MISO, and MOSI. You see that there are many more entries available in `board` than the labels on the QT Py.

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

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

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

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



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

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

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

I2C, SPI, and UART

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

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

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

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

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

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

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

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

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

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

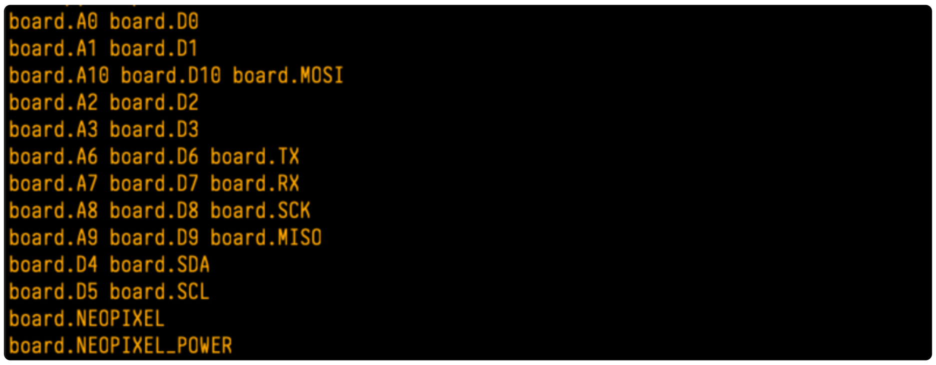
What Are All the Available Names?

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

On a microcontroller board running CircuitPython, connect to the serial console. Then, save the following as **code.py** on your **CIRCUITPY** drive.

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

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

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



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

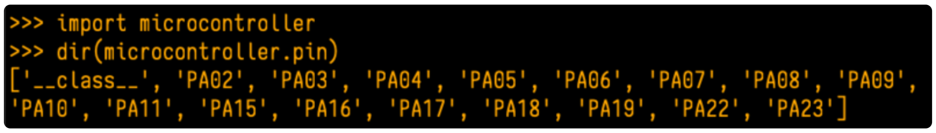
Each line represents a single pin. Find the line containing the pin name that's labeled on the physical board, and you'll find the other names available for that pin. For example, the first pin on the board is labeled **A0**. The first line in the output is `board.A0 board.D0`. This means that you can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

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

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

Microcontroller Pin Names

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



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

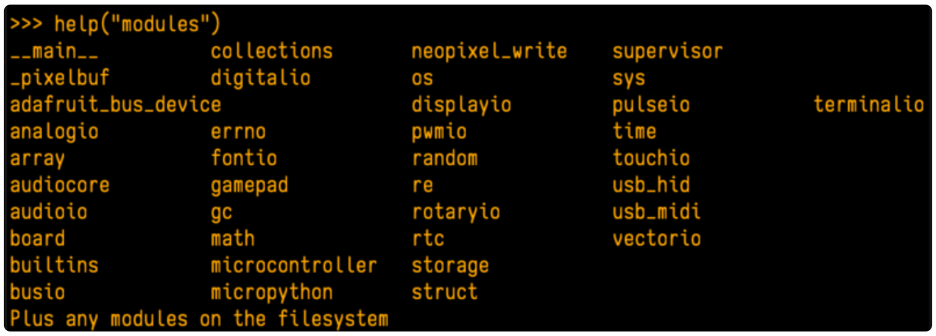
CircuitPython Built-In Modules

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

There are two options for this. You can check the support matrix (https://adafru.it/N2a), and search for your board by name. Or, you can use the REPL.

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

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



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

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

Advanced Serial Console on Windows

Windows 7 Driver

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

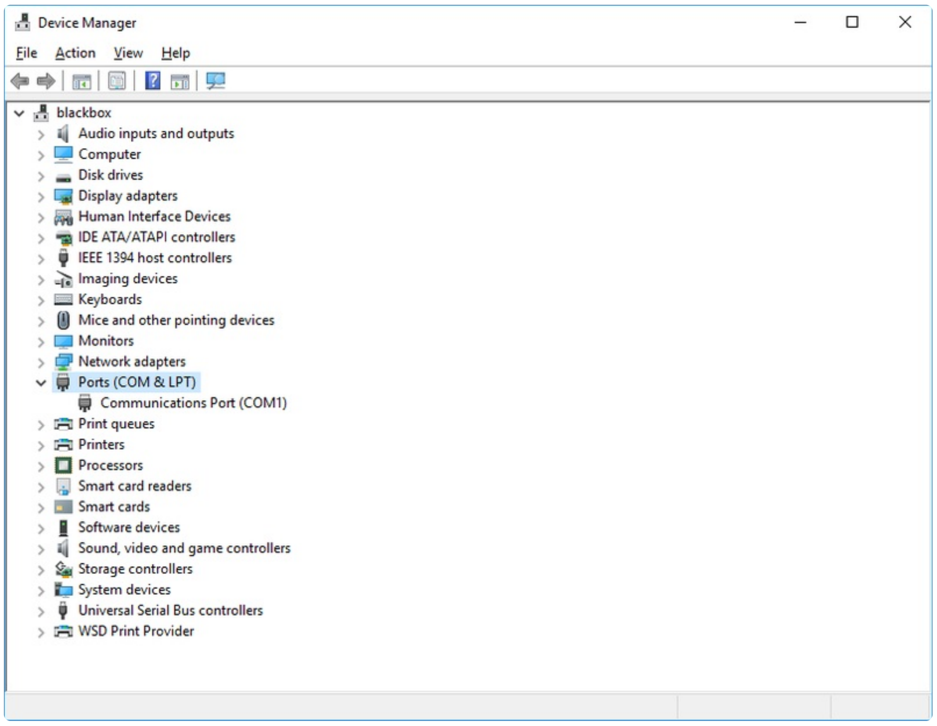
https://adafru.it/AB0

https://adafru.it/AB0

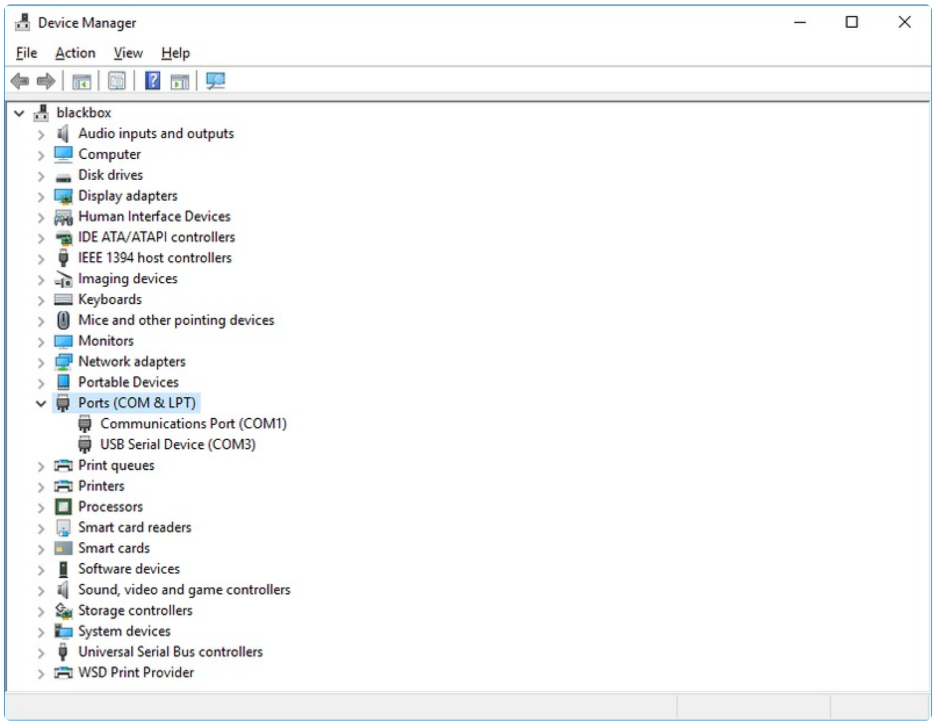
What's the COM?

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

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



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.



Device Manager

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

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

Install Putty

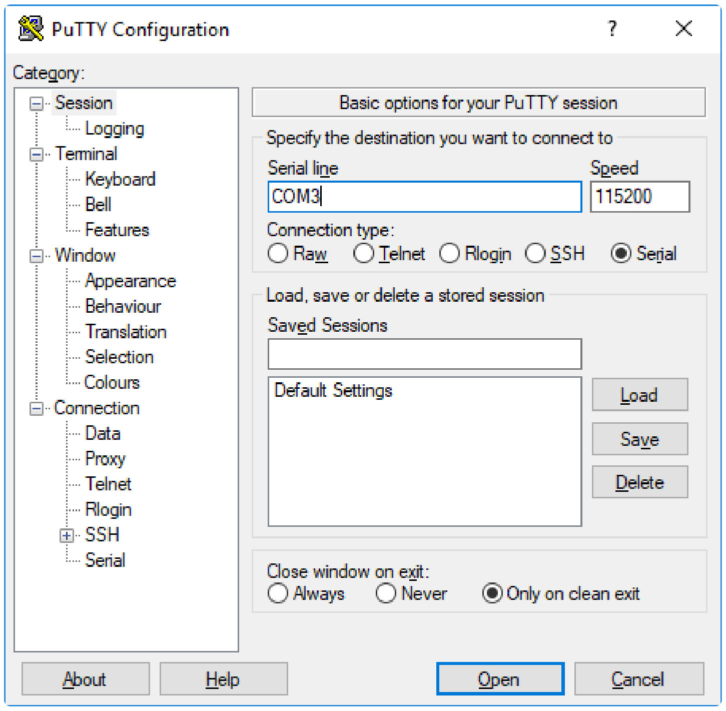
If you're using Windows, you'll need to download a terminal program. We're going to use PuTTY.

The first thing to do is download the latest version of PuTTY (https://adafru.it/Bf1). You'll want to download the Windows installer file. It is most likely that you'll need the 64-bit version. Download the file and install the program on your machine. If you run into issues, you can try downloading the 32-bit version instead. However, the 64-bit version will work on most PCs.

Now you need to open PuTTY.

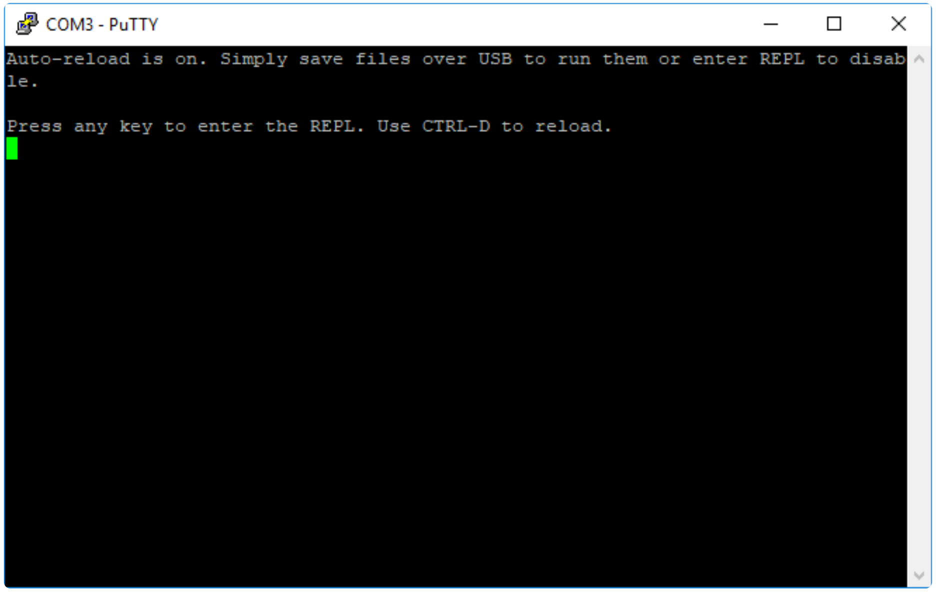
- Under **Connection type:** choose the button next to **Serial**.
- In the box under **Serial line**, enter the serial port you found that your board is using.
- In the box under **Speed**, enter 115200. This called the baud rate, which is the speed in bits per second that data is sent over the serial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

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



Serial line: COM3
Speed: 115200
Connection type: Raw, Telnet, Rlogin, SSH, [✓ Serial]
Close window on exit: Always, Never, [✓ Only on clean exit]

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



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

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

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

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

Advanced Serial Console on Mac and Linux

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

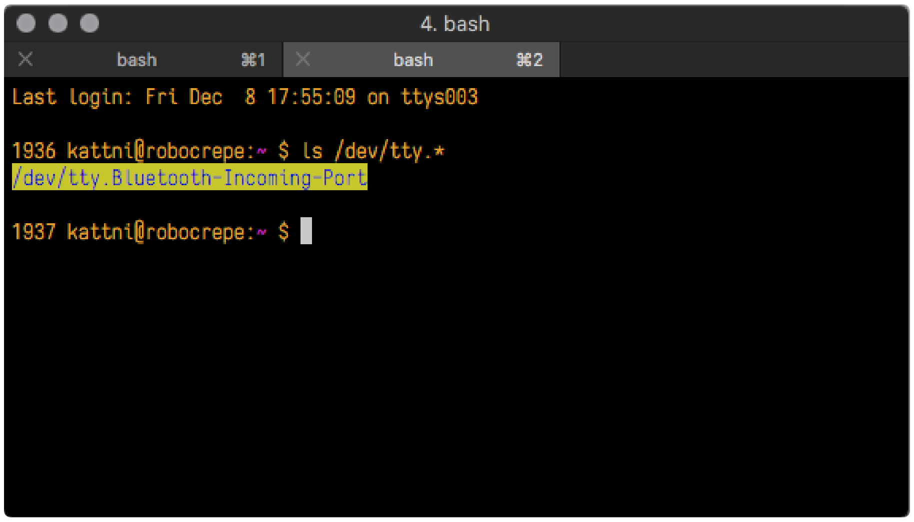
What's the Port?

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

We're going to use Terminal to determine what port the board is using. The easiest way to determine which port the board is using is to first check **without** the board plugged in. On Mac, open Terminal and type the following:

`ls /dev/tty.*`

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

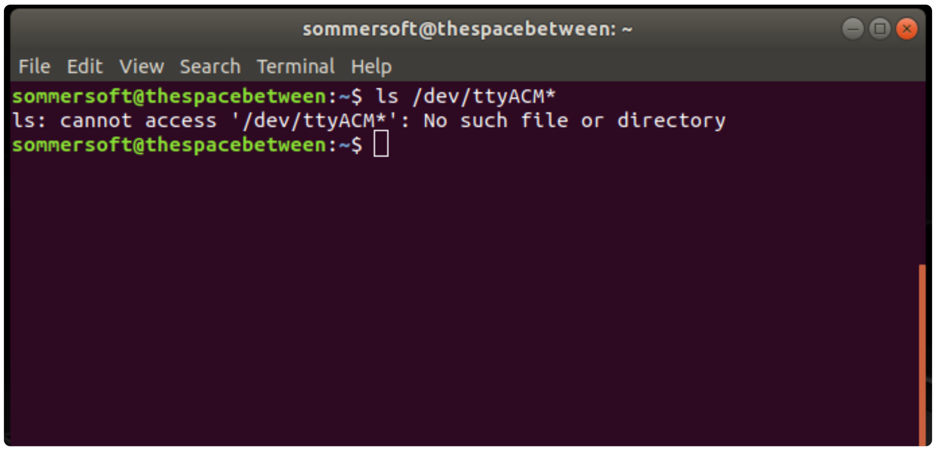


```
Last login: Fri Dec  8 17:55:09 on ttys003

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

1937 kattni@robocrepe:~ $
```

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

`ls /dev/ttyACM*`

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



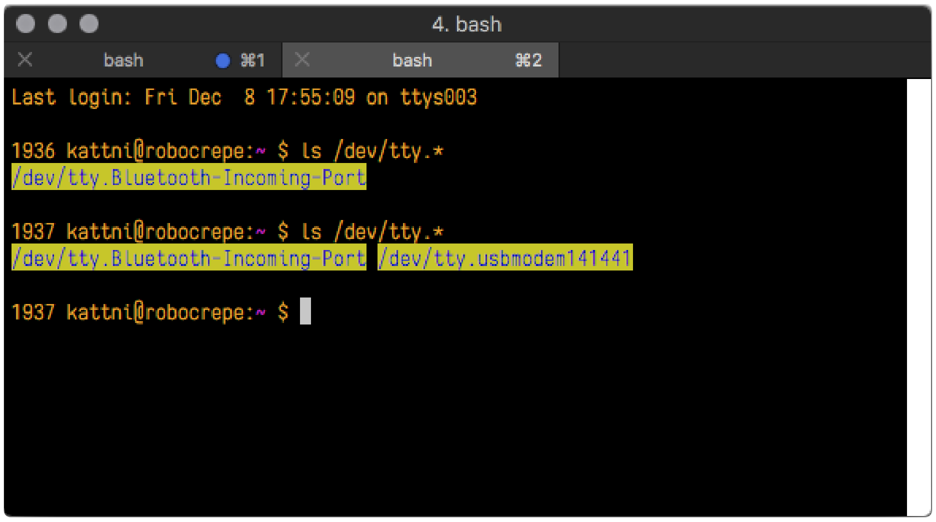
sommersoft@thespacebetween: ~
File Edit View Search Terminal Help

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

Now, plug your board. Using Mac, type:

`ls /dev/tty.*`

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



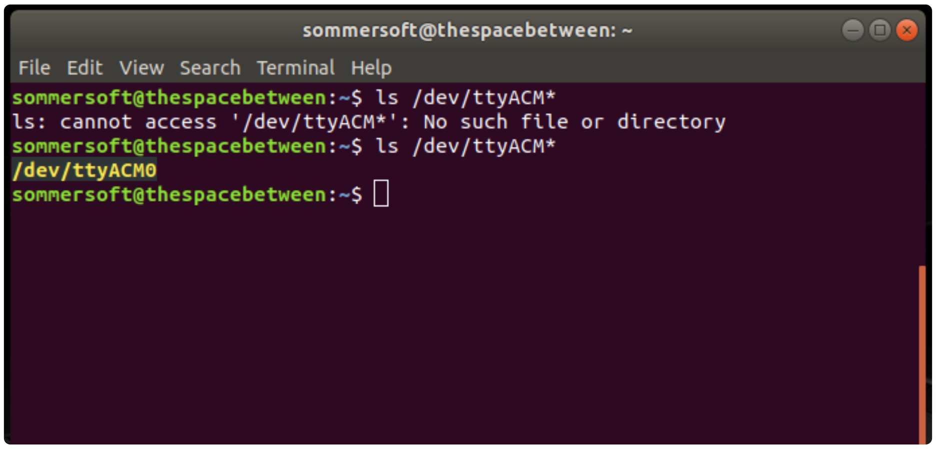
```
Last login: Fri Dec  8 17:55:09 on ttys003

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

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

1937 kattni@robocrepe:~ $
```

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

Using Linux, type:

`ls /dev/ttyACM*`

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


sommersoft@thespacebetween: ~
File Edit View Search Terminal Help

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

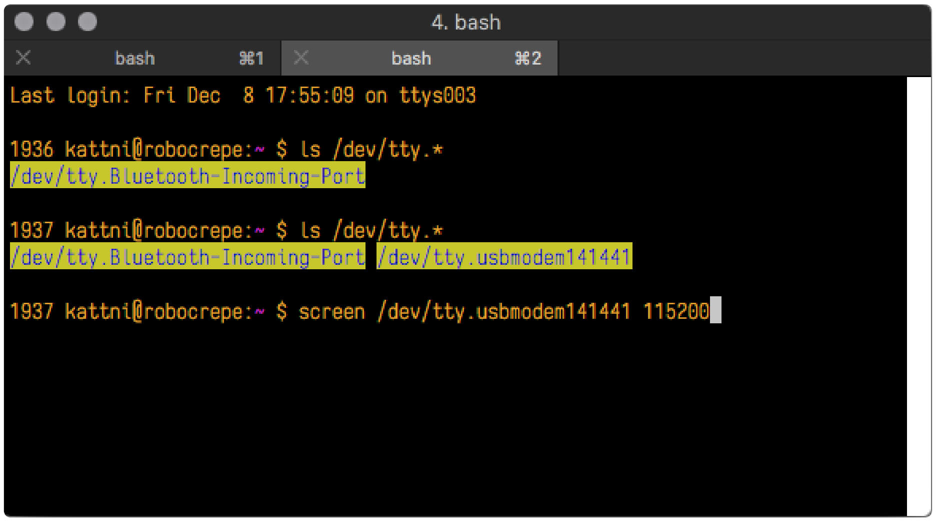
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

Connect with screen

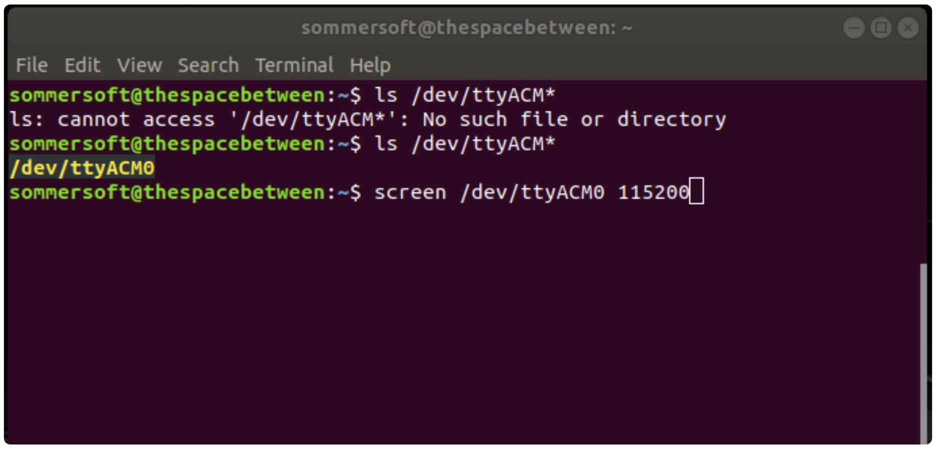
Now that you know the name your board is using, you're ready connect to the serial console. We're going to use a command called `screen`. The `screen` command is included with MacOS. Linux users may need to install it using their package manager. To connect to the serial console, use Terminal. Type the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

`screen /dev/tty.board_name 115200`

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



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

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

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

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


sommersoft@thespacebetween: ~
File Edit View Search Terminal Help

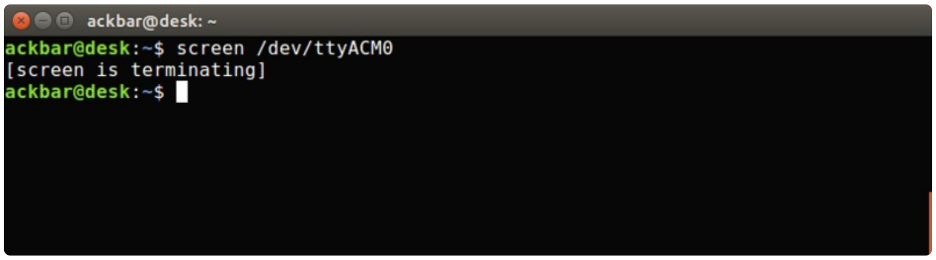
```
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
/dev/ttyACM0
sommersoft@thespacebetween:~$ screen /dev/ttyACM0 115200
```

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

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

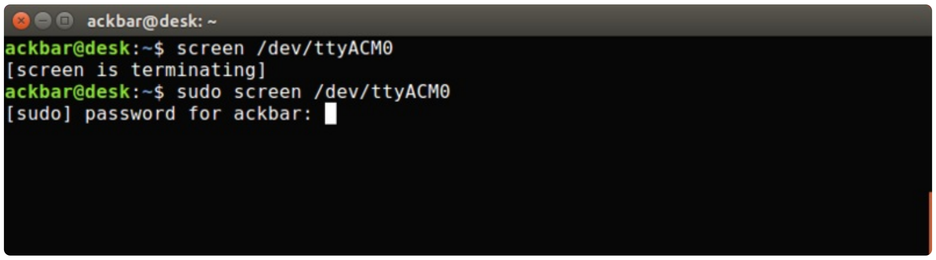
Permissions on Linux

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



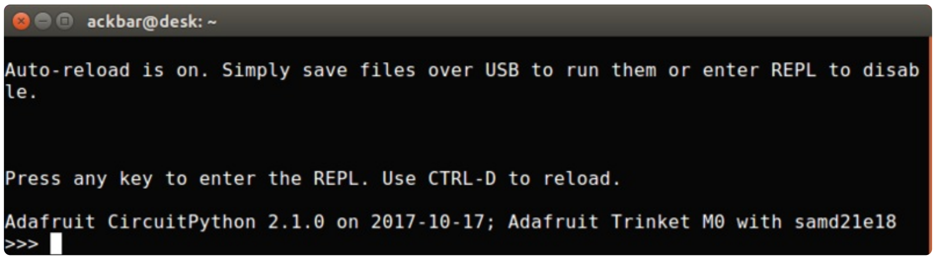
```
ackbar@desk:~$ screen /dev/ttyACM0
[screen is terminating]
ackbar@desk:~$
```

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



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

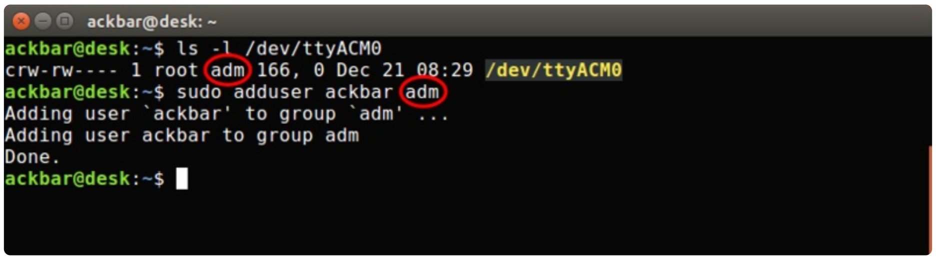
Once you enter your password, you should be in:



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


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

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

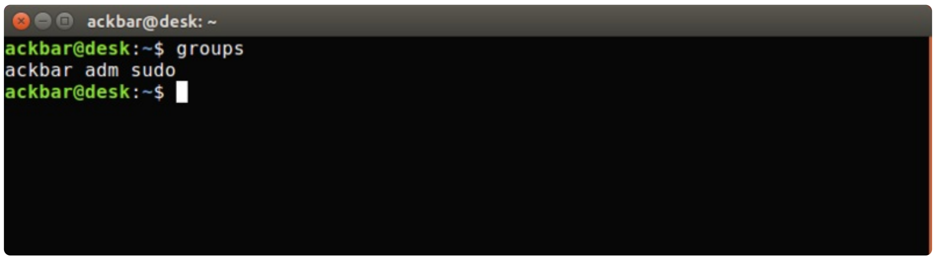
The second way is to add yourself to the group associated with the hardware. To figure out what that group is, use the command `ls -l` as shown below. The group name is circled in red.

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



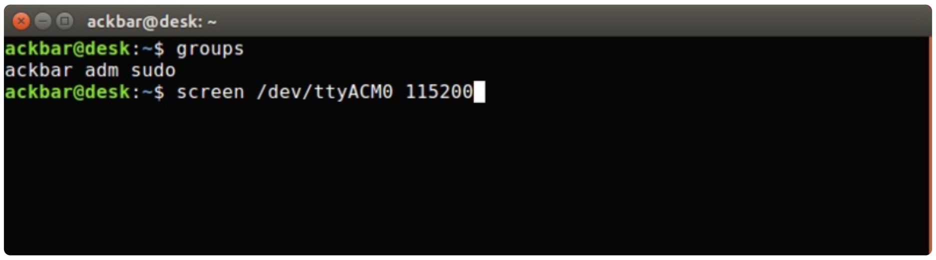
```
ackbar@desk:~$ ls -l /dev/ttyACM0
crw-rw---- 1 root adm 166, 0 Dec 21 08:29 /dev/ttyACM0
ackbar@desk:~$ sudo adduser ackbar adm
Adding user `ackbar' to group `adm' ...
Adding user ackbar to group adm
Done.
ackbar@desk:~$ 
```

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



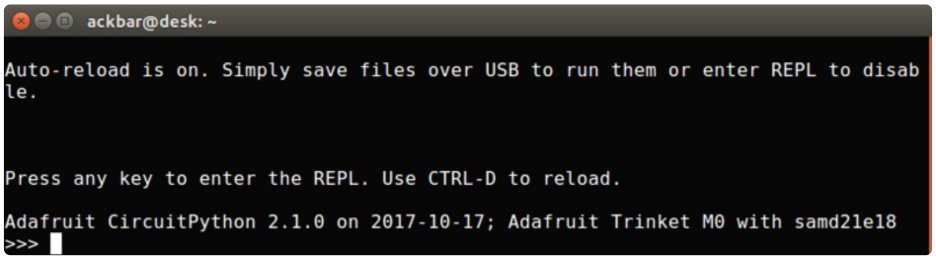
```
ackbar@desk:~$ groups
ackbar adm sudo
ackbar@desk:~$ 
```

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



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

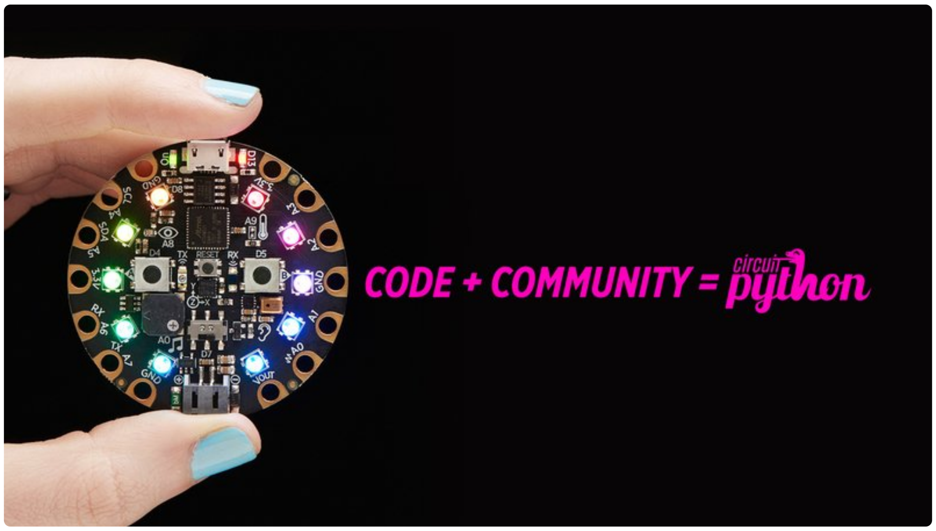
And you're in:



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


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

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

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

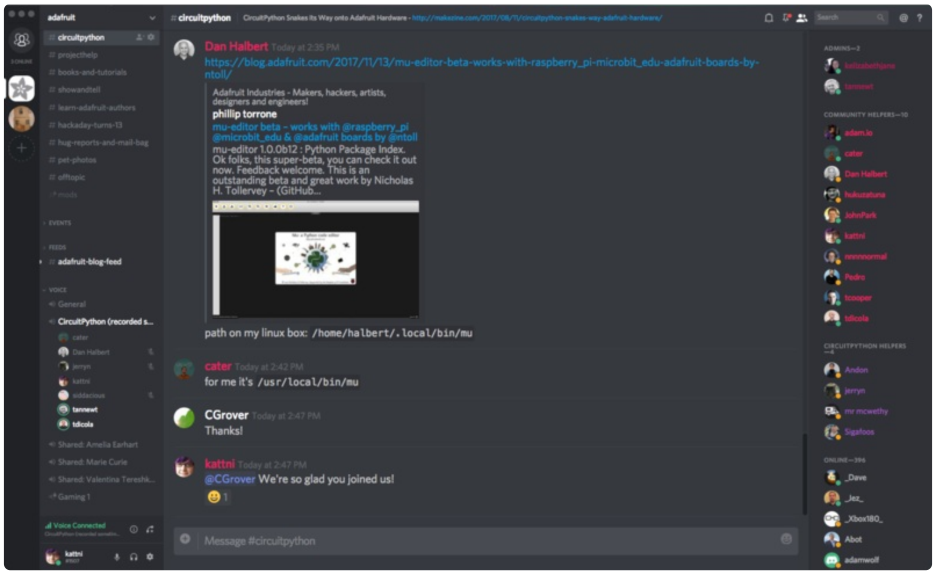
Welcome to the Community!



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

Everyone is welcome! CircuitPython is Open Source. This means it's available for anyone to use, edit, copy and improve upon. This also means CircuitPython becomes better because of you being a part of it. It doesn't matter whether this is your first microcontroller board or you're a computer engineer, you have something important to offer the Adafruit CircuitPython community. We're going to highlight some of the many ways you can be a part of it!

Adafruit Discord


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

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

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

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

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

CGrover Today at 2:47 PM
Thanks!

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

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

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

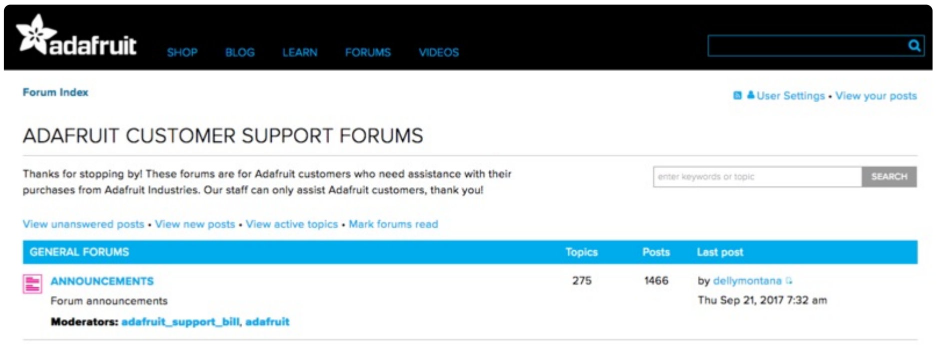
The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. We'd love to hear what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

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

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

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

Adafruit Forums



Forum Index

User Settings • View your posts

ADAFRUIT CUSTOMER SUPPORT FORUMS

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

enter keywords or topic SEARCH

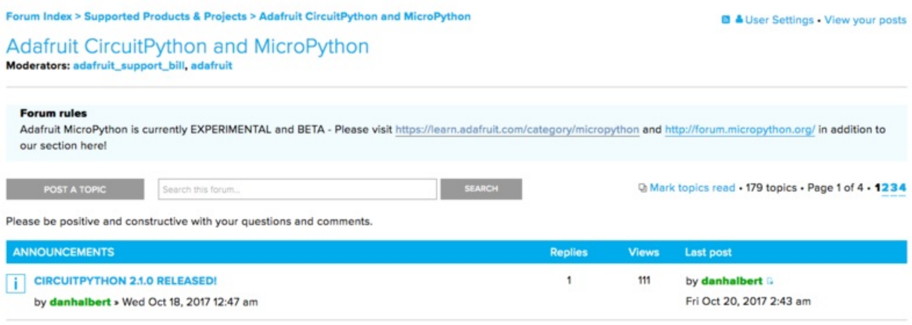
View unanswered posts • View new posts • View active topics • Mark forums read

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

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

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

There are forum categories that cover all kinds of topics, including everything Adafruit. The Adafruit CircuitPython and MicroPython (https://adafru.it/xXA) category under "Supported Products & Projects" is the best place to post your CircuitPython questions.


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

User Settings • View your posts

Adafruit CircuitPython and MicroPython

Moderators: adafruit_support_bill, adafruit

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

Mark topics read • 179 topics • Page 1 of 4 • 1234

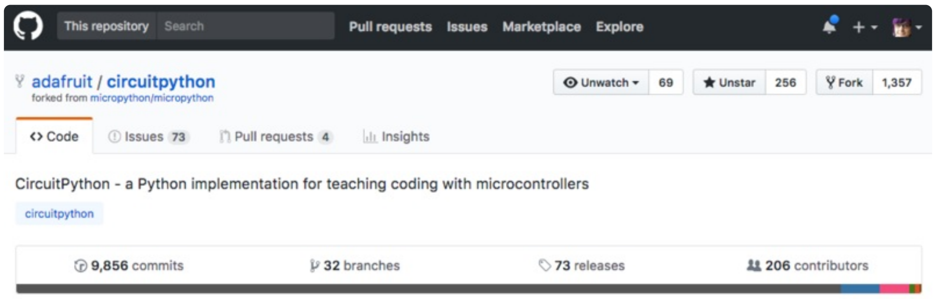
Please be positive and constructive with your questions and comments.

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

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

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

Adafruit Github



adafruit / circuitpython
forked from micropython/micropython

Unwatch 69 · Unstar 256 · Fork 1,357

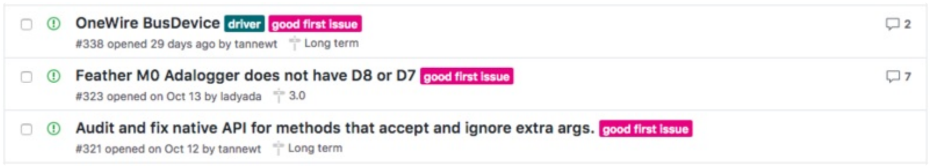
CircuitPython - a Python implementation for teaching coding with microcontrollers

circuitpython

9,856 commits · 32 branches · 73 releases · 206 contributors

Whether you're just beginning or are life-long programmer who would like to contribute, there are ways for everyone to be a part of building CircuitPython. GitHub is the best source of ways to contribute to CircuitPython (https://adafru.it/tB7) itself. If you need an account, visit https://github.com/ (https://adafru.it/d6C)and sign up.

If you're new to GitHub or programming in general, there are great opportunities for you. Head over to adafruit/circuitpython (https://adafru.it/tB7) on GitHub, click on "Issues (https://adafru.it/Bee)", and you'll find a list that includes issues labeled "good first issue (https://adafru.it/Bef)". These are things we've identified as something that someone with any level of experience can help with. These issues include options like updating documentation, providing feedback, and fixing simple bugs.


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

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

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

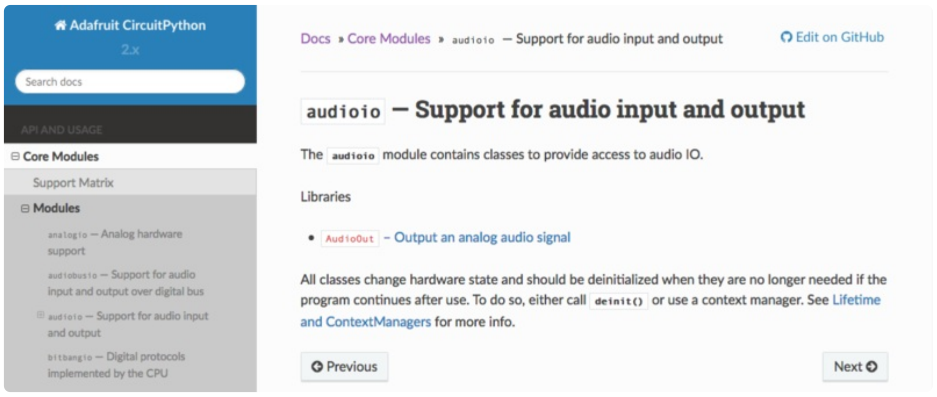
Already experienced and looking for a challenge? Checkout the rest of the issues list and you'll find plenty of ways to contribute. You'll find everything from new driver requests to core module updates. There's plenty of opportunities for everyone at any level!

When working with CircuitPython, you may find problems. If you find a bug, that's great! We love bugs! Posting a detailed issue to GitHub is an invaluable way to contribute to improving CircuitPython. Be sure to include the steps to replicate the issue as well as any other information you think is relevant. The more detail, the better!

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

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

ReadTheDocs


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

Docs » Core Modules » audioio — Support for audio input and output

audioio — Support for audio input and output

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

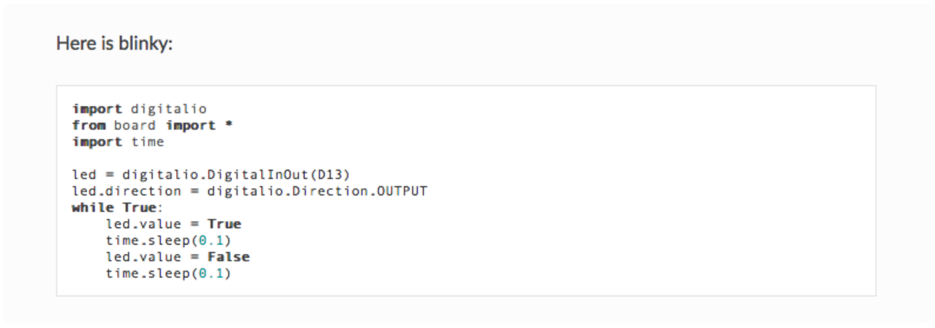
Libraries

- AudioOut – Output an analog audio signal

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

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

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


Here is blinky:

```
import digitalio
from board import *
import time

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

Frequently Asked Questions

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

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

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

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

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

Is ESP8266 or ESP32 supported in CircuitPython? Why not?

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

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

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

How do I connect to the Internet with CircuitPython?

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

Is there asyncio support in CircuitPython?

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

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

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

What is a `MemoryError`?

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

What do I do when I encounter a `MemoryError`?

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

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

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

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

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

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

How can I create my own `.mpy` files?

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

You can download `mpy-cross` for your operating system from [https://adafruit-circuit-python.s3.amazonaws.com/index.html?prefix=bin/mpy-cross/](https://adafru.it/QDK) (https://adafru.it/QDK). Almost any version will do. The format for .mpy files has not changed since CircuitPython 4.x.

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

How do I check how much memory I have free?

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

Will give you the number of bytes available for use.

Does CircuitPython support interrupts?

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

Does Feather M0 support WINC1500?

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

Can AVRs such as ATmega328 or ATmega2560 run CircuitPython?

No.

Commonly Used Acronyms

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

Troubleshooting

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

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

Always Run the Latest Version of CircuitPython and Libraries

As we continue to develop CircuitPython and create new releases, we will stop supporting older releases. **You need to update to the latest CircuitPython. (https://adafru.it/Em8).**

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

As we release new versions of CircuitPython, we will stop providing the previous bundles as automatically created downloads on the Adafruit CircuitPython Library Bundle repo. If you must continue to use an earlier version, you can still download the appropriate version of `mpy-cross` from the particular release of CircuitPython on the CircuitPython repo and create your own compatible .mpy library files. **However, it is best to update to the latest for both CircuitPython and the library bundle.**

I have to continue using CircuitPython 5.x, 4.x, 3.x or 2.x, where can I find compatible libraries?

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

CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present

You may have a different board.

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

MakeCode

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

MacOS

DriveDx and its accompanything **SAT SMART Driver** can interfere with seeing the BOOT drive. [See this forum post](https://adafru.it/sTc) (https://adafru.it/sTc) for how to fix the problem.

Windows 10

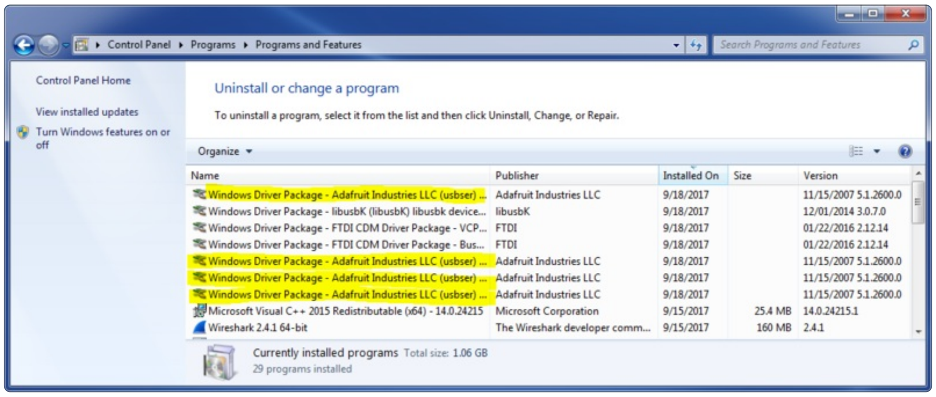
Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 with the driver package installed? You don't need to install this package on Windows 10 for most Adafruit boards. The old version (v1.5) can interfere with recognizing your device. Go to **Settings** ->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

Windows 7 or 8.1

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

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

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


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

Uninstall or change a program

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

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

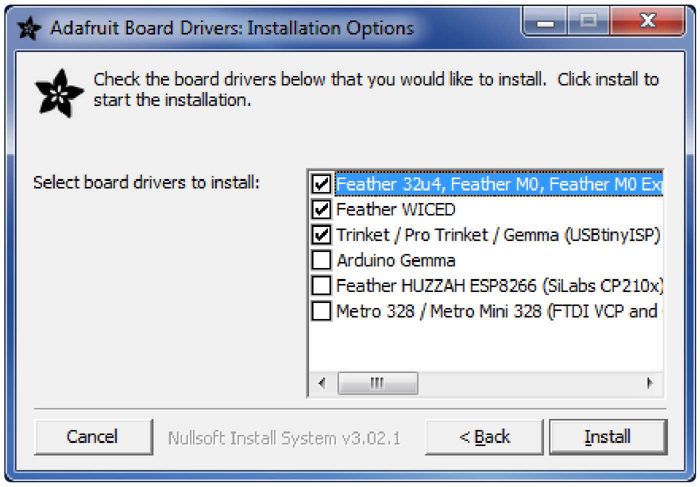
Currently installed programs Total size: 1.06 GB
29 programs installed

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

https://adafru.it/AB0

https://adafru.it/AB0

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


Adafruit Board Drivers: Installation Options

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

Select board drivers to install:

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

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

Let us know in the Adafruit support forums (https://adafru.it/jIf) or on the Adafruit Discord () if this does not work for you!

Windows Explorer Locks Up When Accessing boardnameBOOT Drive

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

- **AIDA64**: to fix, stop the program. This problem has been reported to AIDA64. They acquired hardware to test, and released a beta version that fixes the problem. This may have been incorporated into the latest release. Please let us know in the forums if you test this.
- **Hard Disk Sentinel**
- **Kaspersky anti-virus**: To fix, you may need to disable Kaspersky completely. Disabling some aspects of Kaspersky does not always solve the problem. This problem has been reported to Kaspersky.
- **ESET NOD32 anti-virus**: We have seen problems with at least version 9.0.386.0, solved by

uninstallation.

Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied

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

CIRCUITPY Drive Does Not Appear

Kaspersky anti-virus can block the appearance of the `CIRCUITPY` drive. We haven't yet figured out a settings change that prevents this. Complete uninstallation of Kaspersky fixes the problem.

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

Windows 7 and 8.1 Problems

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

Serial Console in Mu Not Displaying Anything

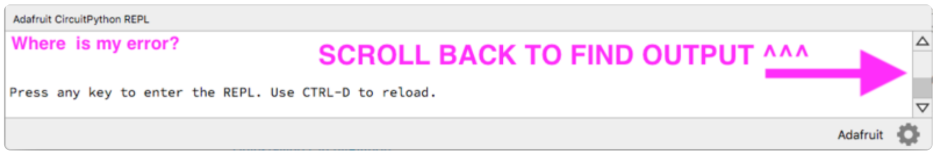
There are times when the serial console will accurately not display anything, such as, when no code is currently running, or when code with no serial output is already running before you open the console. However, if you find yourself in a situation where you feel it should be displaying something like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

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

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



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

More complex errors take even more lines!

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



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

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

CircuitPython RGB Status Light

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

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

Here's what the colors and blinking mean:

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

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

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

These are followed by flashes indicating the line number, including place value. **WHITE** flashes are thousands' place, **BLUE** are hundreds' place, **YELLOW** are tens' place, and **CYAN** are one's place. So for example, an error on line 32 would flash **YELLOW** three times and then **CYAN** two times. Zeroes are indicated by an extra-long dark gap.

ValueError: Incompatible `.mpy` file.

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

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

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

CIRCUITPY Drive Issues

You may find that you can no longer save files to your `CIRCUITPY` drive. You may find that your `CIRCUITPY` stops showing up in your file explorer, or shows up as `NO_NAME`. These are indicators that your filesystem has issues.

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

If still broken - When the `CIRCUITPY` disk is not safely ejected before being reset by the button or being disconnected from USB, it may corrupt the flash drive. It can happen on Windows, Mac or Linux.

In this situation, the board must be completely erased and CircuitPython must be reloaded onto the board.

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

Easiest Way: Use `storage.erase_filesystem()`

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

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

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

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

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

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

1. Download the correct erase file:

https://adafru.it/AdI

https://adafru.it/AdI

https://adafru.it/AdJ

https://adafru.it/AdJ

https://adafru.it/EVK

https://adafru.it/EVK

https://adafru.it/AdK

https://adafru.it/AdK

https://adafru.it/EoM

https://adafru.it/EoM

https://adafru.it/DjD

https://adafru.it/DjD

https://adafru.it/DBA

https://adafru.it/DBA

https://adafru.it/Eca

https://adafru.it/Eca

https://adafru.it/Gnc

https://adafru.it/Gnc

https://adafru.it/GAN

https://adafru.it/GAN

https://adafru.it/GAO

https://adafru.it/GAO

https://adafru.it/Jat

https://adafru.it/Jat

https://adafru.it/Q5B

https://adafru.it/Q5B

2. Double-click the reset button on the board to bring up the *boardname*BOOT drive.
3. Drag the erase .uf2 file to the *boardname*BOOT drive.
4. The onboard NeoPixel will turn yellow or blue, indicating the erase has started.
5. After approximately 15 seconds, the mainboard NeoPixel will light up green. On the NeoTrellis M4 this is the first NeoPixel on the grid
6. Double-click the reset button on the board to bring up the *boardname*BOOT drive.
7. Drag the appropriate latest release of CircuitPython (https://adafru.it/Amd) .uf2 file to the *boardname*BOOT drive.

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

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

If you haven't already downloaded the latest release of CircuitPython for your board, check out the [installation page](https://adafru.it/Amd) (https://adafru.it/Amd). You'll also need to install your libraries and code!

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

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

1. Download the erase file:

https://adafru.it/AdL

https://adafru.it/AdL

2. Double-click the reset button on the board to bring up the `*boardname*BOOT` drive.
3. Drag the erase `.uf2` file to the `*boardname*BOOT` drive.
4. The boot LED will start flashing again, and the `*boardname*BOOT` drive will reappear.
5. [Drag the appropriate latest release CircuitPython](https://adafru.it/Amd) (https://adafru.it/Amd) `.uf2` file to the `*boardname*BOOT` drive.

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

[If you haven't already downloaded the latest release of CircuitPython for your board, check out the installation page](https://adafru.it/Amd) (https://adafru.it/Amd) You'll also need to install your libraries and code!

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

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

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

Running Out of File Space on Non-Express Boards

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

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

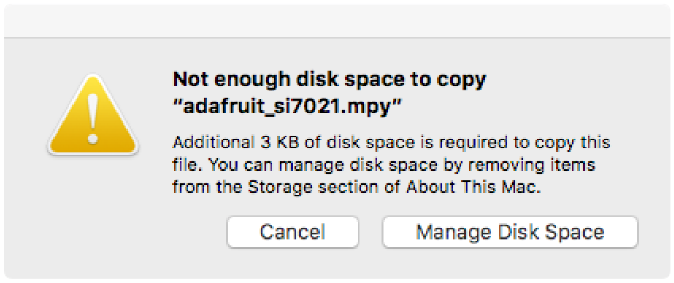
Delete something!

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

Use tabs

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

MacOS loves to add extra files.


Not enough disk space to copy "adafruit_si7021.mpy"

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

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

Prevent & Remove MacOS Hidden Files

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

```
ls -l /Volumes
```

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

Now follow the steps from this question (https://adafru.it/u1c) to run these terminal commands that stop hidden files from being created on the board:

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

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

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

`>>> import storage`
`>>> storage.erase_filesystem`

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

Copy Files on MacOS Without Creating Hidden Files

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

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

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

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

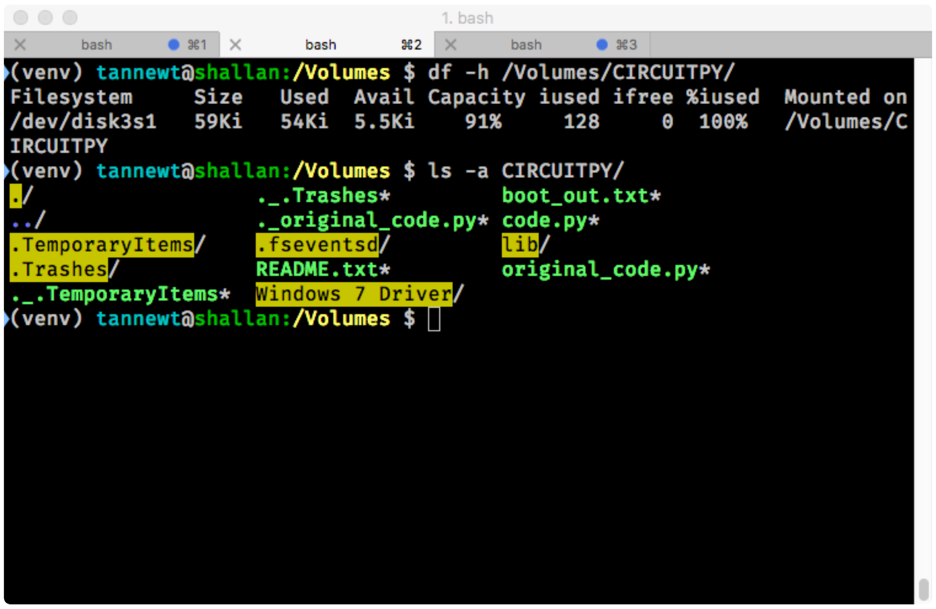
```
cp -rX folder_to_copy /Volumes/CIRCUITPY
```

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

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

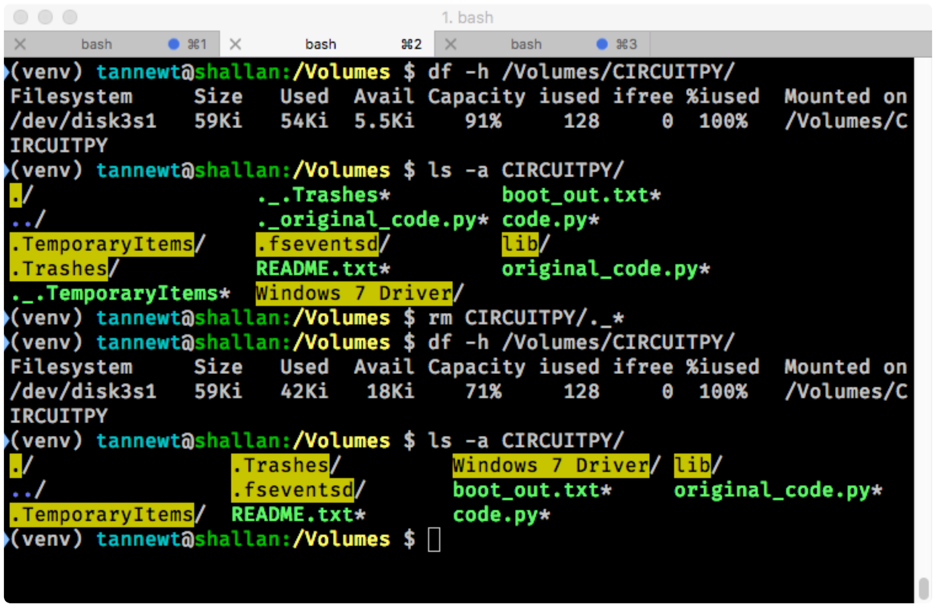
Other MacOS Space-Saving Tips

If you'd like to see the amount of space used on the drive and manually delete hidden files here's how to do so. First list the amount of space used on the CIRCUITPY drive with the **df** command:



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

Lets remove the ._ files first.



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

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

Device locked up or boot looping

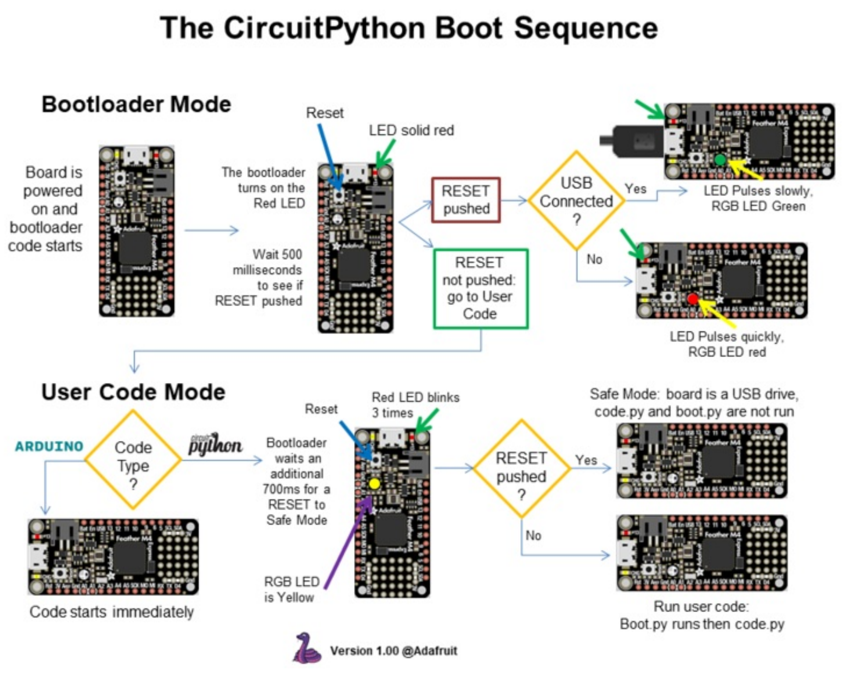
In rare cases, it may happen that something in your **code.py** or **boot.py** files causes the device to get locked up, or even go into a boot loop. These are not your everyday Python exceptions, typically it's the result of a deeper problem within CircuitPython. In this situation, it can be difficult to recover your device if **CIRCUITPY** is not allowing you to modify the **code.py** or **boot.py** files. Safe mode is one recovery option. When the device boots up in safe mode it will not run the **code.py** or **boot.py** scripts, but will still connect the **CIRCUITPY** drive so that you can remove or modify those files as needed.

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

For most devices:
Press the reset button, and then when the RGB status LED is yellow, press the reset button again.

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

Refer to the following diagram for boot sequence details:



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

Uninstalling CircuitPython

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

Maybe you tried CircuitPython and want to go back to MakeCode or Arduino? Not a problem

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

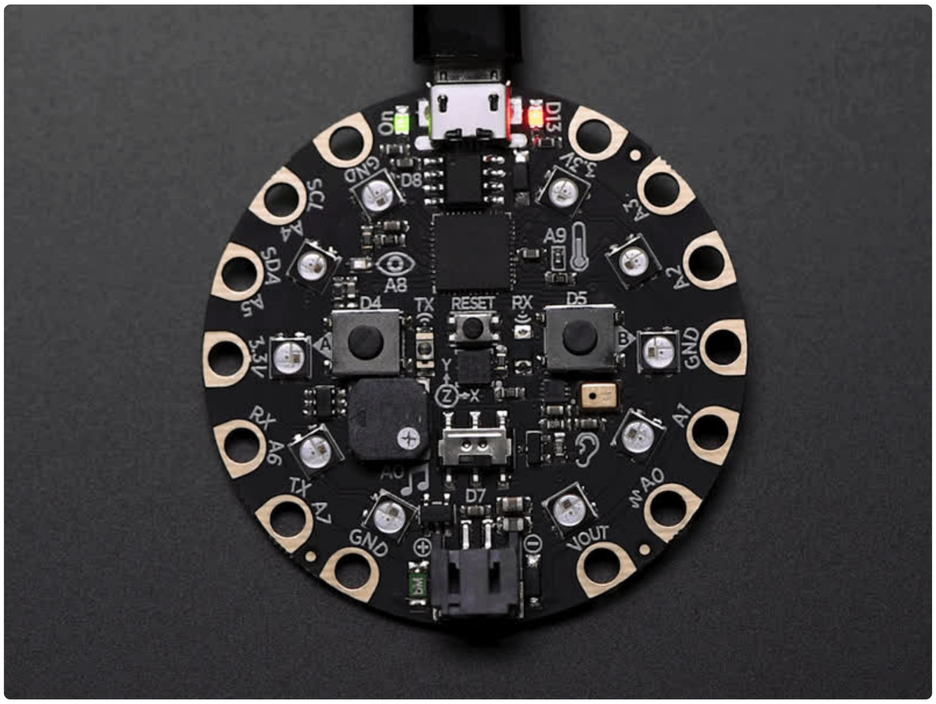
Backup Your Code

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

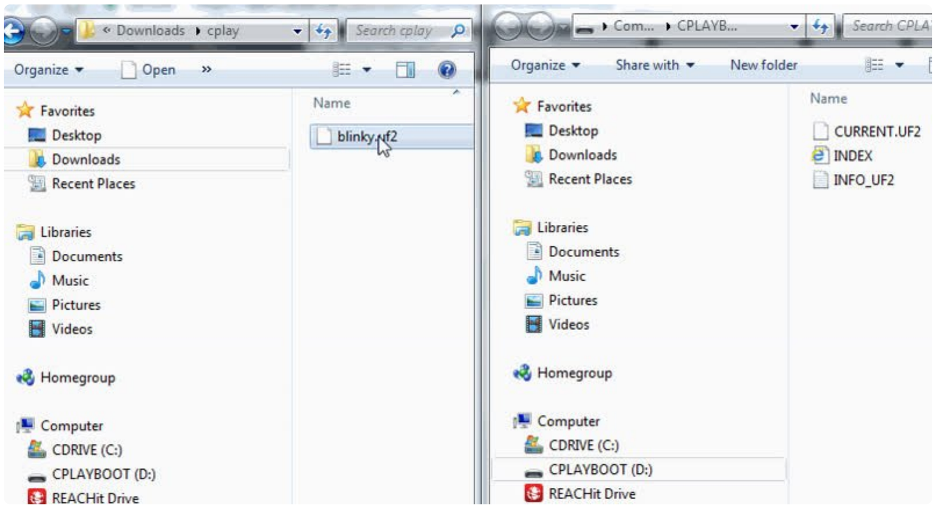
Moving Circuit Playground Express to MakeCode

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

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



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



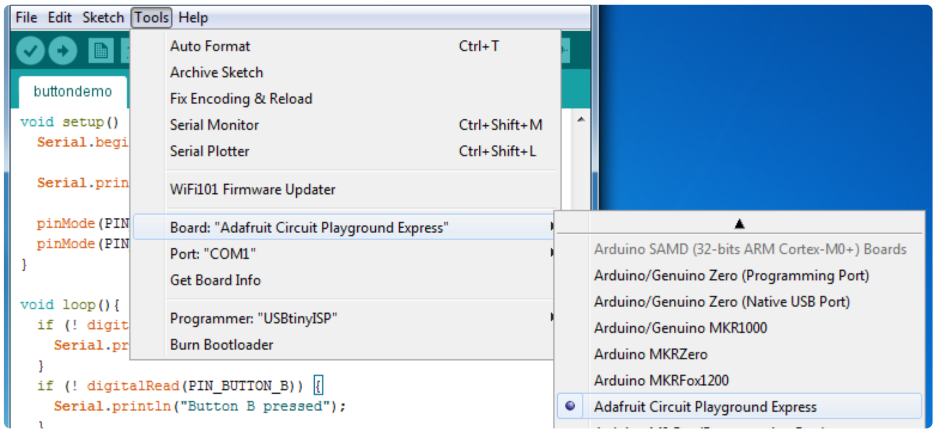
Your MakeCode is now running and CircuitPython has been removed. Going forward you only have to **single click** the reset button

Moving to Arduino

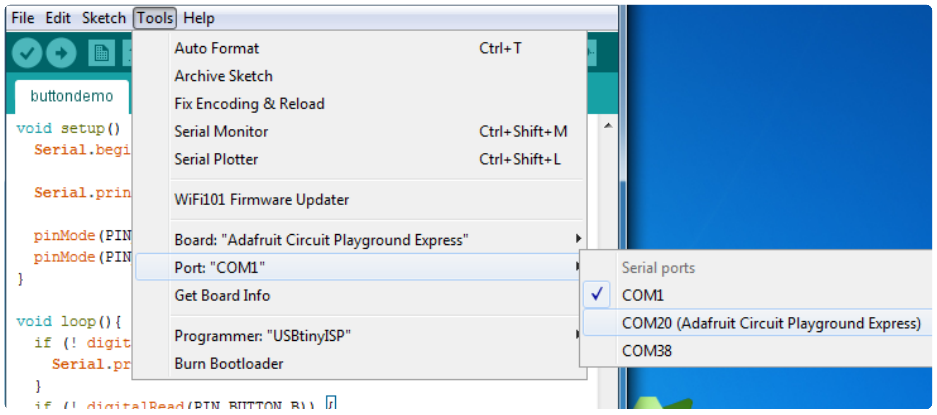
If you want to change your firmware to Arduino, it's also pretty easy.

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

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


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

Select the correct matching Port:


File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Adafruit Circuit Playground Express"
Port: "COM1"
Get Board Info
Programmer: "USBtinyISP"
Burn Bootloader
Serial ports
COM1
COM20 (Adafruit Circuit Playground Express)
COM38

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

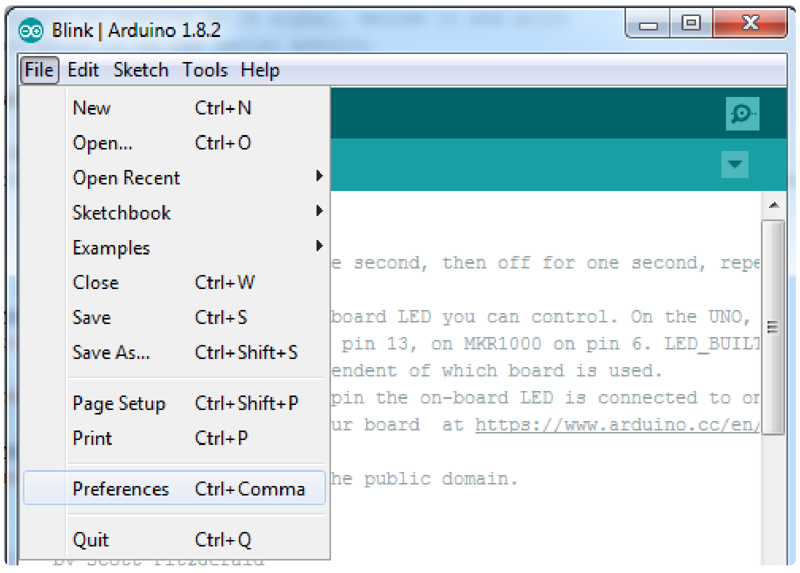
Arduino IDE Setup

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

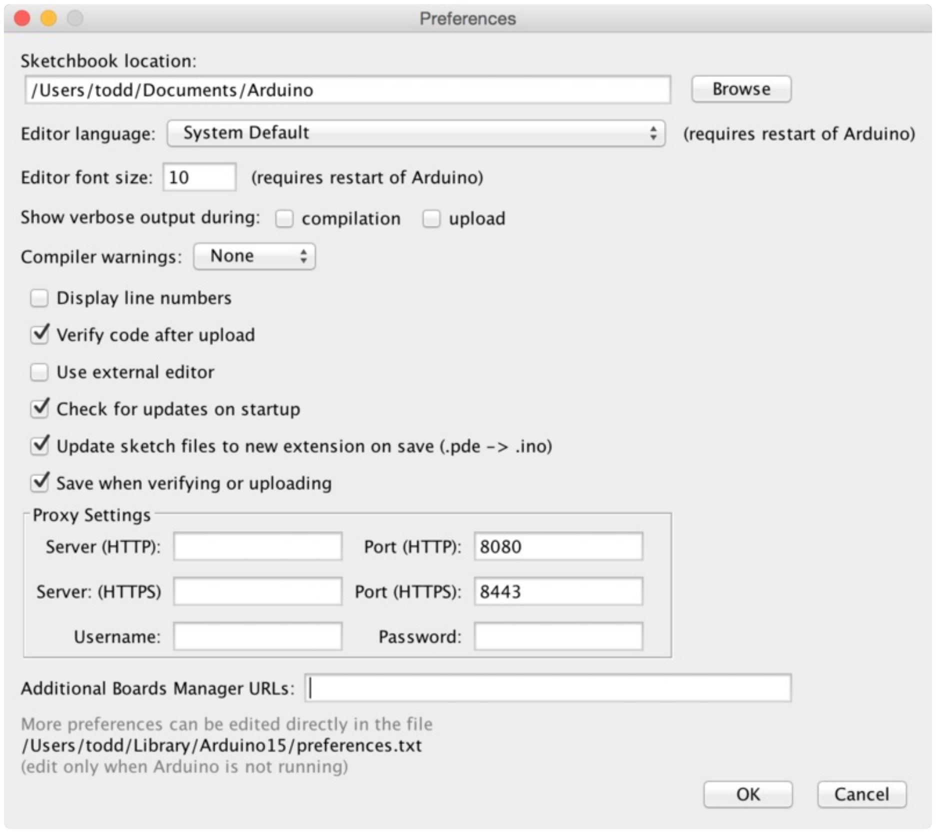
https://adafru.it/f1P

https://adafru.it/f1P

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.


Preferences

Sketchbook location: /Users/todd/Documents/Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 10 (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☐ Display line numbers

✓ Verify code after upload

☐ Use external editor

✓ Check for updates on startup

✓ Update sketch files to new extension on save (.pde -> .ino)

✓ Save when verifying or uploading

Proxy Settings

Server (HTTP): Port (HTTP): 8080

Server: (HTTPS) Port (HTTPS): 8443

Username: Password:

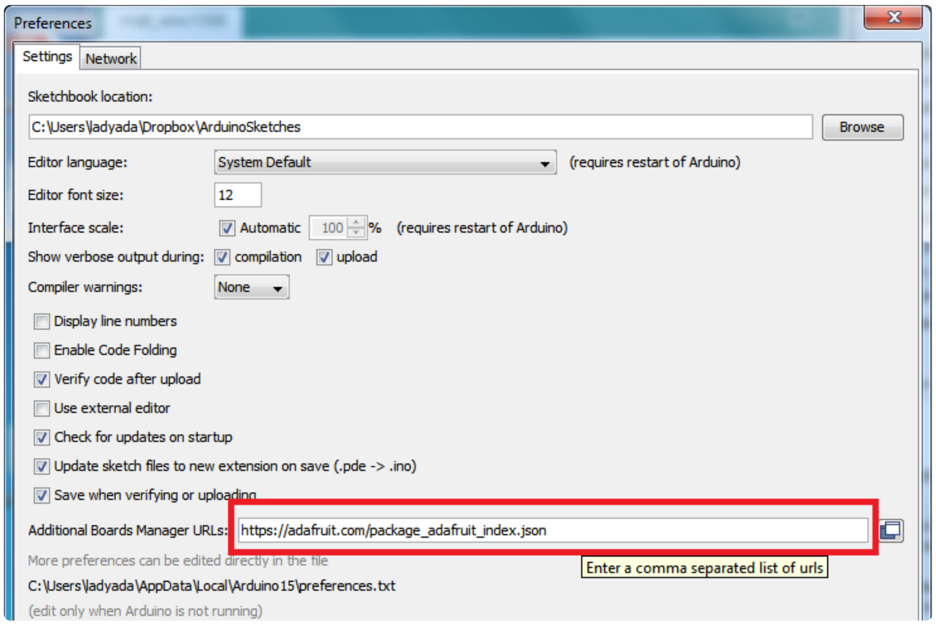
Additional Boards Manager URLs:

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

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 (https://adafru.it/f7U). 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`


Preferences

Settings | Network

Sketchbook location: C:\Users\ladyada\Dropbox\ArduinoSketches [Browse]

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

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

Show verbose output during: ✓ compilation ✓ upload

Compiler warnings: None

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

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

Enter a comma separated list of urls

More preferences can be edited directly in the file

C:\Users\ladyada\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

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, 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 (https://adafru.it/eSI).

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

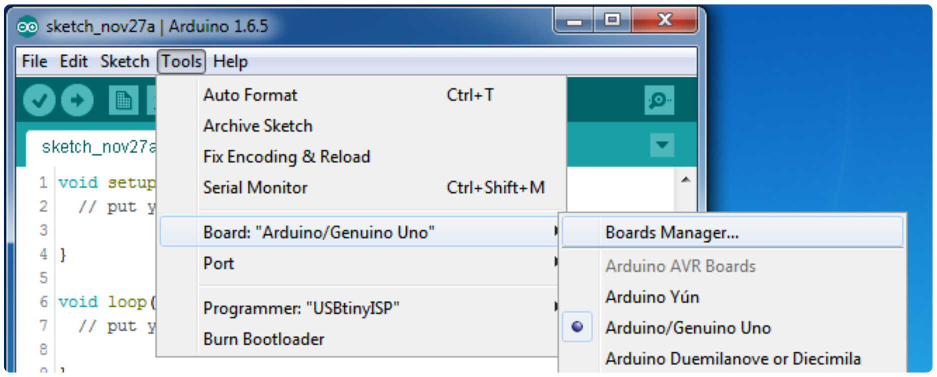
Once done click **OK** to save the new preference settings. Next we will look at installing boards with the Board Manager.

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

Using with Arduino IDE

The Feather/Metro/Gemma/QTPy/Trinket M0 and M4 use an ATSAMD21 or ATSAMD51 chip, and you can pretty easily get it working with the Arduino IDE. Most libraries (including the popular ones like NeoPixels and display) will work with the M0 and M4, especially devices & sensors that use I2C or SPI.

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



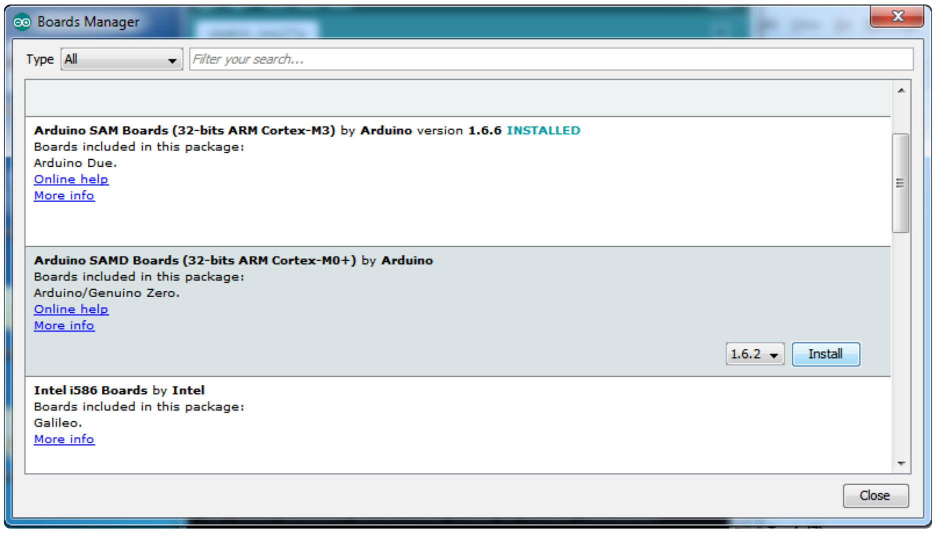
Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **All**. You will then be able to select and install the boards supplied by the URLs added to the preferences.

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

Install SAMD Support

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

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



Arduino SAM Boards (32-bits ARM Cortex-M3) by **Arduino** version **1.6.6** **INSTALLED**
Boards included in this package:
Arduino Due.
Online help
More info

Arduino SAMD Boards (32-bits ARM Cortex-M0+) by **Arduino**
Boards included in this package:
Arduino/Genuino Zero.
Online help
More info

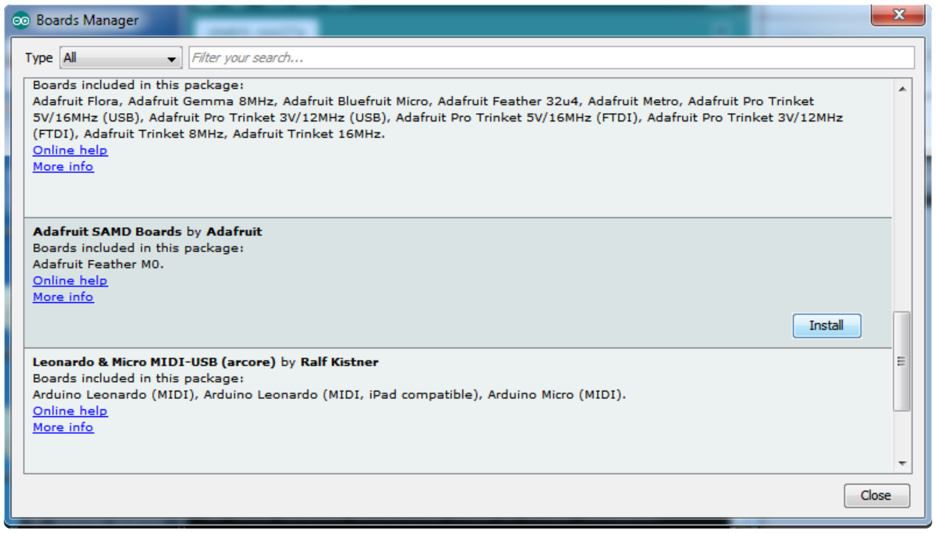
Intel i586 Boards by **Intel**
Boards included in this package:
Galileo.
More info

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**



Boards included in this package:
Adafruit Flora, Adafruit Gemma 8MHz, Adafruit Bluefruit Micro, Adafruit Feather 32u4, Adafruit Metro, Adafruit Pro Trinket 5V/16MHz (USB), Adafruit Pro Trinket 3V/12MHz (USB), Adafruit Pro Trinket 5V/16MHz (FTDI), Adafruit Pro Trinket 3V/12MHz (FTDI), Adafruit Trinket 8MHz, Adafruit Trinket 16MHz.
Online help
More info

Adafruit SAMD Boards by **Adafruit**
Boards included in this package:
Adafruit Feather M0.
Online help
More info

Leonardo & Micro MIDI-USB (arcore) by **Ralf Kistner**
Boards included in this package:
Arduino Leonardo (MIDI), Arduino Leonardo (MIDI, iPad compatible), Arduino Micro (MIDI).
Online help
More info

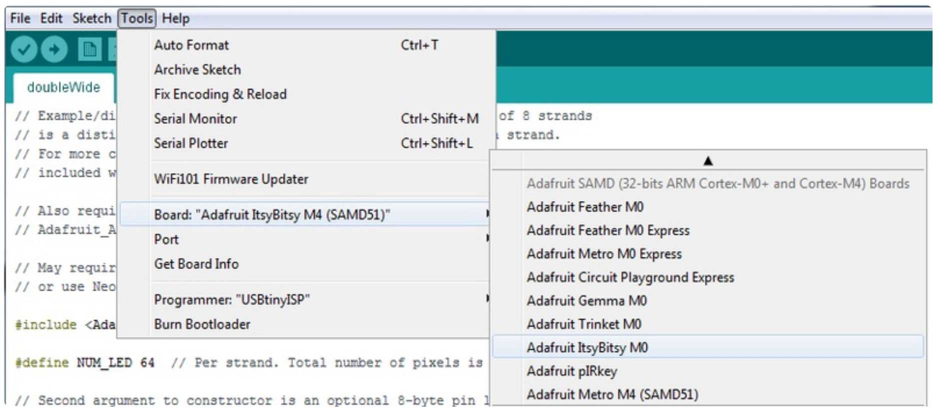
Even though in theory you don't need to - I recommend rebooting the IDE

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**


File Edit Sketch Tools Help
doubleWide
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Adafruit ItsyBitsy M4 (SAMD51)"
Port
Get Board Info
Programmer: "USBtinyISP"
Burn Bootloader
Adafruit SAMD (32-bits ARM Cortex-M0+ and Cortex-M4) Boards
Adafruit Feather M0
Adafruit Feather M0 Express
Adafruit Metro M0 Express
Adafruit Circuit Playground Express
Adafruit Gemma M0
Adafruit Trinket M0
Adafruit ItsyBitsy M0
Adafruit pIRkey
Adafruit Metro M4 (SAMD51)

```
// Example/di                                    of 8 strands
// is a disti                                    strand.
// For more c
// included w

// Also requi
// Adafruit_A

// May requir
// or use Neo

#include <Ada

#define NUM_LED 64  // Per strand. Total number of pixels is

// Second argument to constructor is an optional 8-byte pin l
```

Install Drivers (Windows 7 & 8 Only)

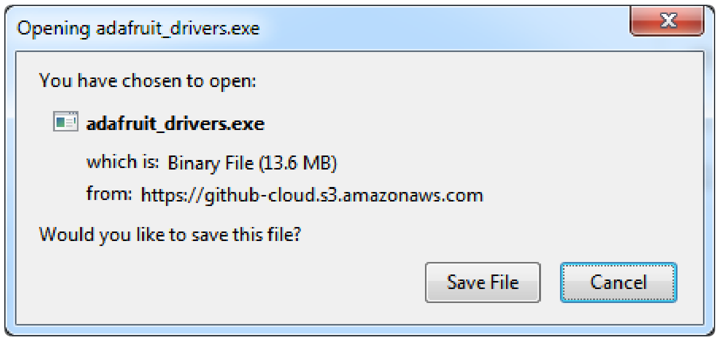
When you plug in the board, you'll need to possibly install a driver

Click below to download our Driver Installer

https://adafru.it/EC0

https://adafru.it/EC0

Download and run the installer



Opening adafruit_drivers.exe

You have chosen to open:

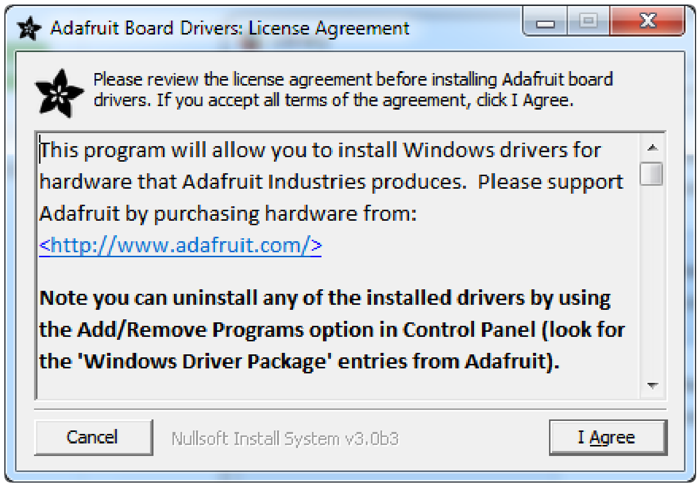
adafruit_drivers.exe

which is: Binary File (13.6 MB)

from: https://github-cloud.s3.amazonaws.com

Would you like to save this file?

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



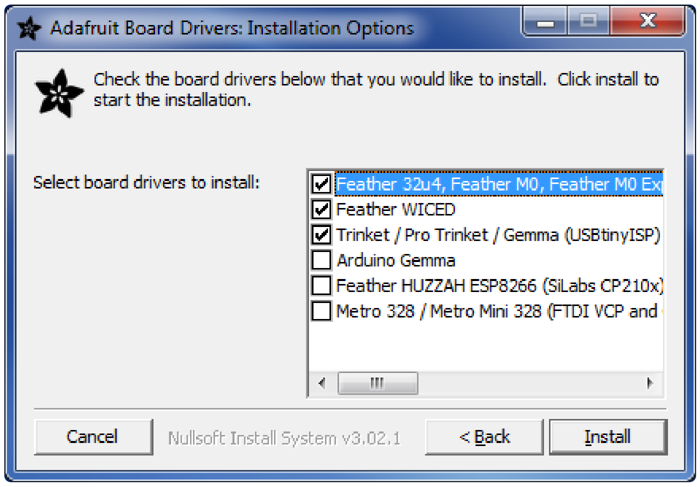
Adafruit Board Drivers: License Agreement

Please review the license agreement before installing Adafruit board drivers. If you accept all terms of the agreement, click I Agree.

This program will allow you to install Windows drivers for hardware that Adafruit Industries produces. Please support Adafruit by purchasing hardware from: <http://www.adafruit.com/>

Note you can uninstall any of the installed drivers by using the Add/Remove Programs option in Control Panel (look for the 'Windows Driver Package' entries from Adafruit).

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



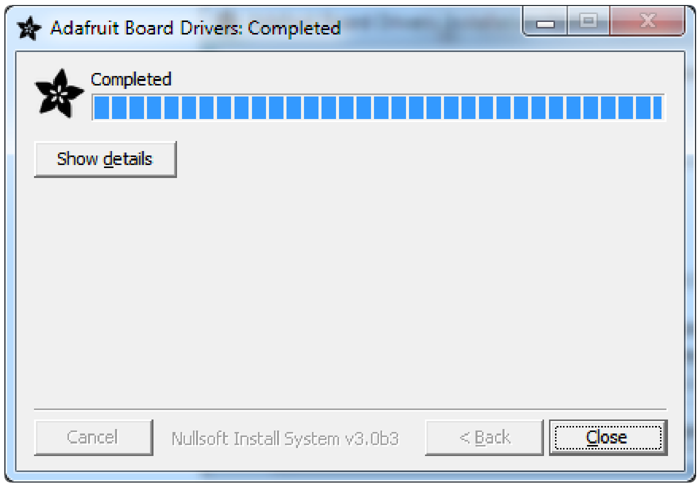
Adafruit Board Drivers: Installation Options

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

Select board drivers to install:

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

Click **Install** to do the installin'

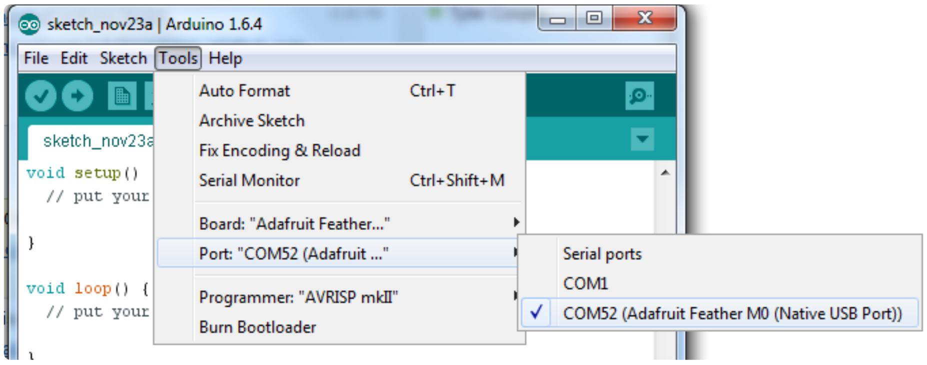

Adafruit Board Drivers: Completed
Completed
Show details
Cancel
Nullsoft Install System v3.0b3
< Back
Close

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!


sketch_nov23a | Arduino 1.6.4
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Adafruit Feather..."
Port: "COM52 (Adafruit ..."
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
✓ COM52 (Adafruit Feather M0 (Native USB Port))
sketch_nov23a

```
void setup()
  // put your
}

void loop() {
  // put your
```

Now load up the Blink example

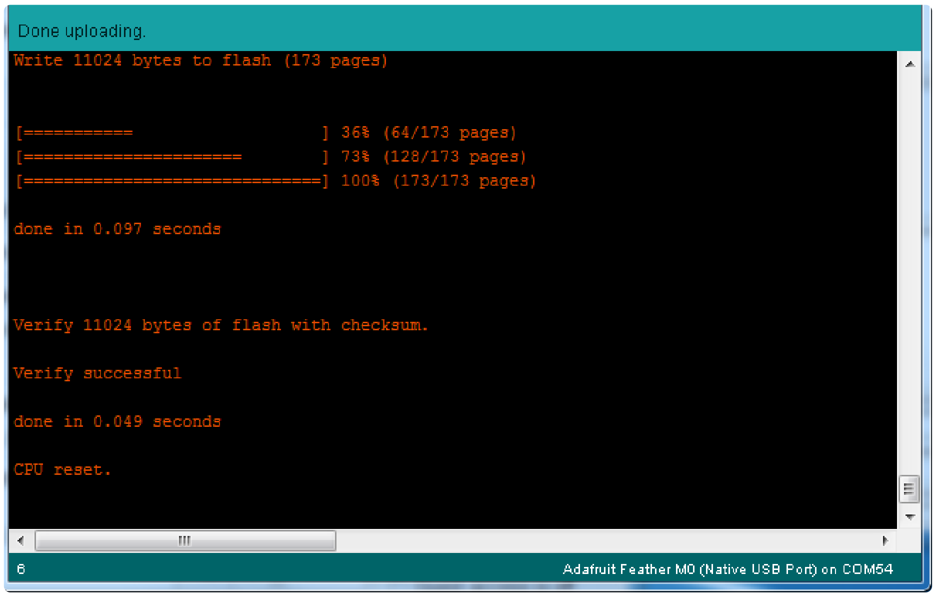
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// 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



```
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.
```

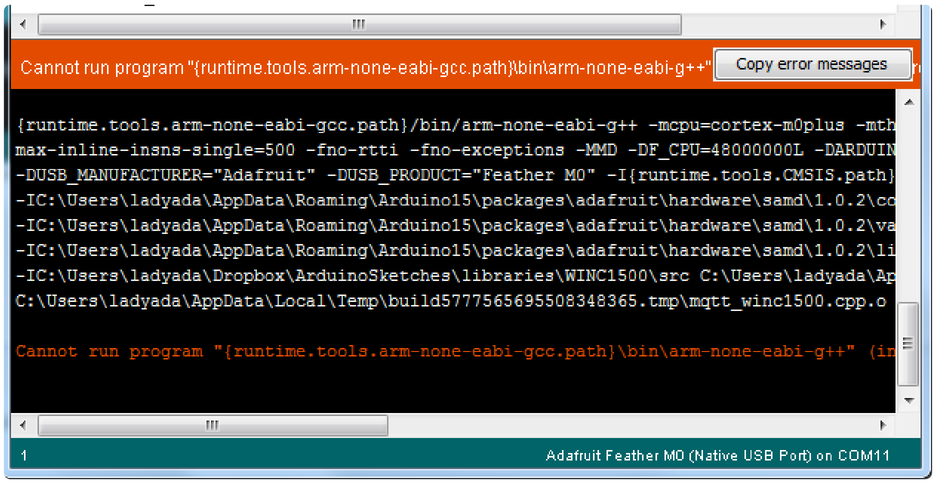
After uploading, you may see a message saying "Disk Not Ejected Properly" about the ...BOOT drive. You can ignore that message: it's an artifact of how the bootloader and uploading work.

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

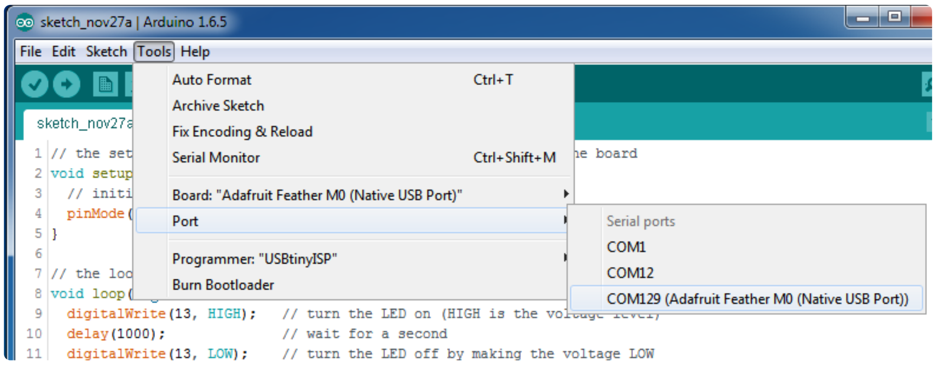

```
Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-none-eabi-g++"

{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-none-eabi-g++" (in
```

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.


sketch_nov27a | Arduino 1.6.5
File Edit Sketch Tools Help
sketch_nov27a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Adafruit Feather M0 (Native USB Port)"
Port
Programmer: "USBtinyISP"
Burn Bootloader
Serial ports
COM1
COM12
COM129 (Adafruit Feather M0 (Native USB Port))
```
// the set                                   he board
void setup
  // initi
  pinMode(
}

// the loo
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
```

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. (https://adafru.it/iOE)

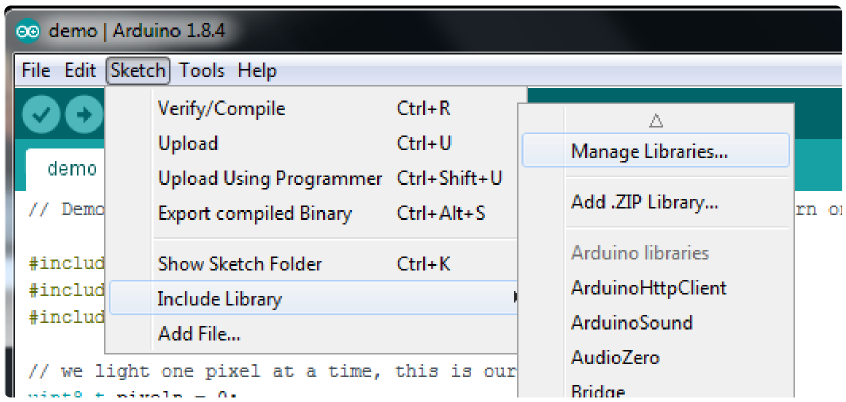
Arcada Libraries

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

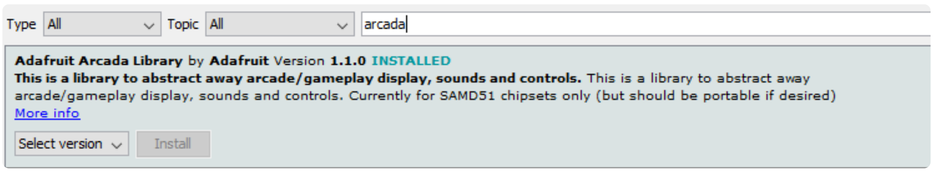
Open up the library manager...



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



Type All Topic All arcada

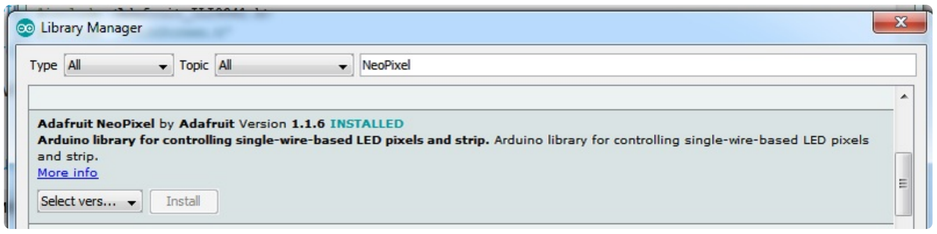
Adafruit Arcada Library by **Adafruit** Version **1.1.0** INSTALLED
This is a library to abstract away arcade/gameplay display, sounds and controls. This is a library to abstract away arcade/gameplay display, sounds and controls. Currently for SAMD51 chipsets only (but should be portable if desired)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

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

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

Adafruit NeoPixel

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


Library Manager

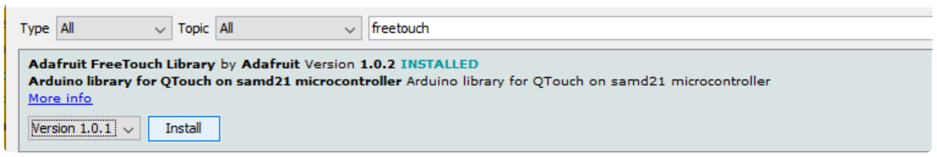
Type All Topic All NeoPixel

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

Select vers... Install

Adafruit FreeTouch

This is the open source version of QTouch for SAMD21 boards

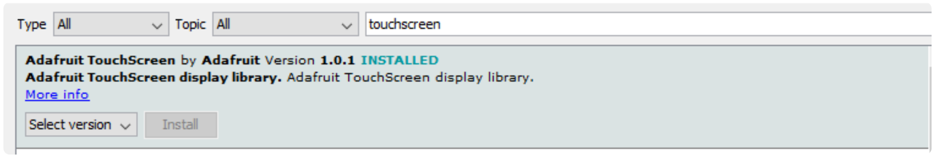

Type All Topic All freetouch

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

Version 1.0.1 Install

Adafruit Touchscreen

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

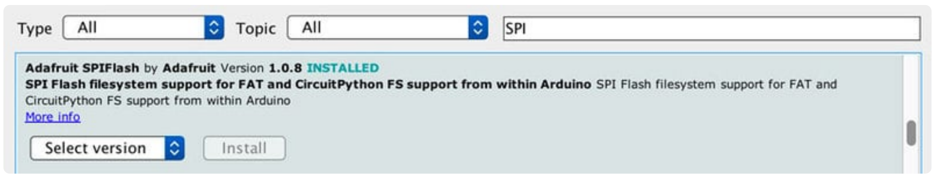

Type All Topic All touchscreen

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

Select version Install

Adafruit SPIFlash

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

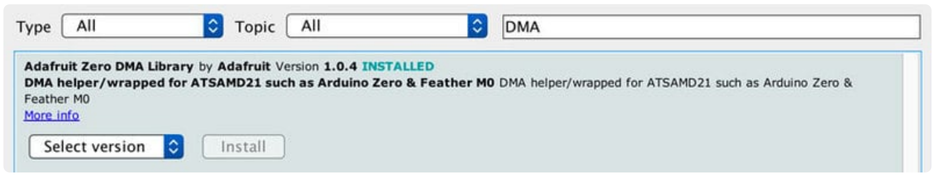

Type All Topic All SPI

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

Select version Install

Adafruit Zero DMA

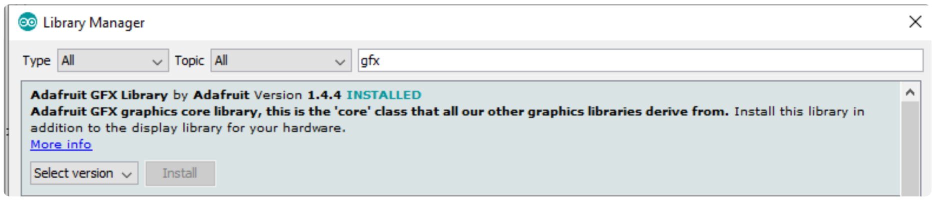
This is used by the Graphics Library if you choose to use DMA


Type All Topic All DMA

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

Adafruit GFX

This is the graphics library used to draw to the screen



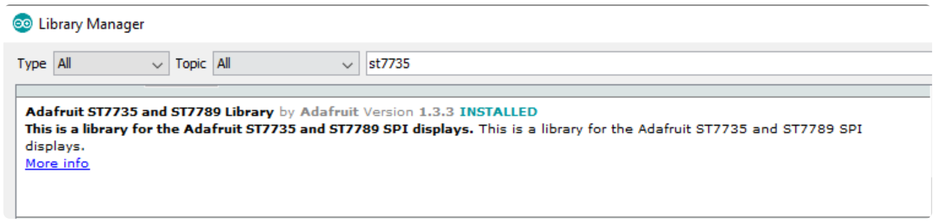
Type All Topic All gfx

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

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

Adafruit ST7735

The display on the PyBadge/PyGamer & other Arcada boards

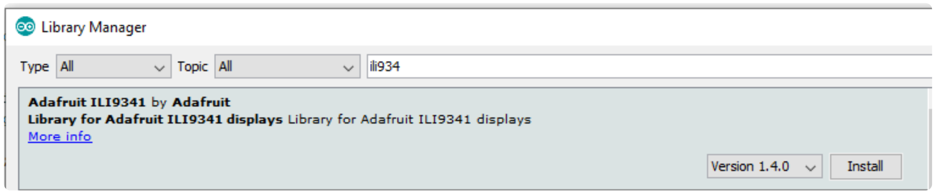


Type All Topic All st7735

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

Adafruit ILI9341

The display on the PyPortal & other Arcada boards

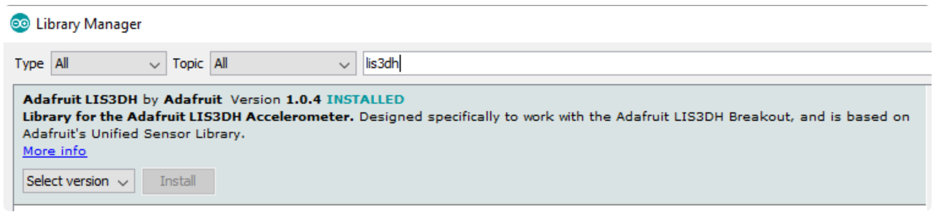


Type All Topic All ili934

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

Adafruit LIS3DH

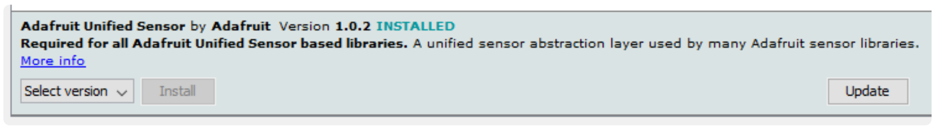
For reading the accelerometer data, required even if one is not on the board



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

Adafruit Sensor

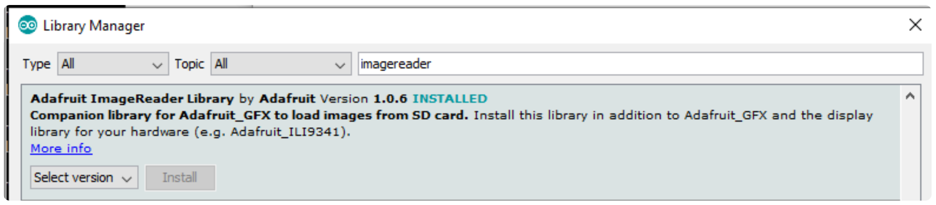
Needed by the LIS3DH Library, required even if one is not on the board



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

Adafruit ImageReader

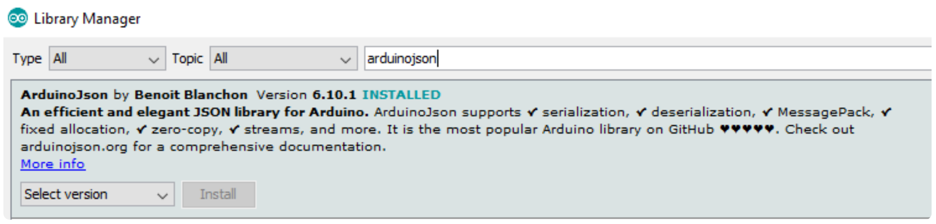
For reading bitmaps from SPI Flash or SD and displaying



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

ArduinoJson

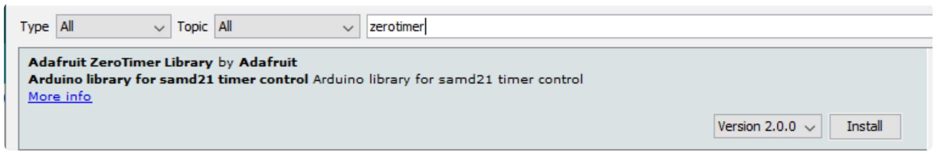
We use this library to read and write configuration files



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

Adafruit ZeroTimer

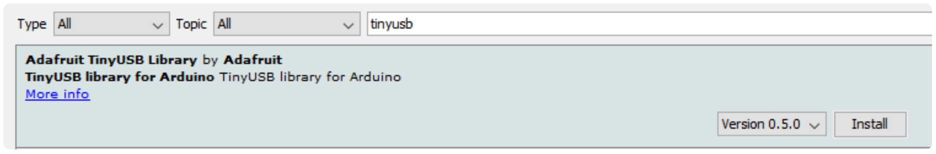
We use this library to easily set timers and callbacks on the SAMD processors



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

Adafruit TinyUSB

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



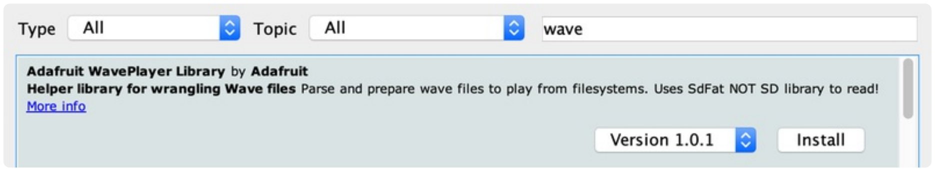
Type All Topic All tinyusb

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

Version 0.5.0 Install

Adafruit WavePlayer

Helps us play .WAV sound files.



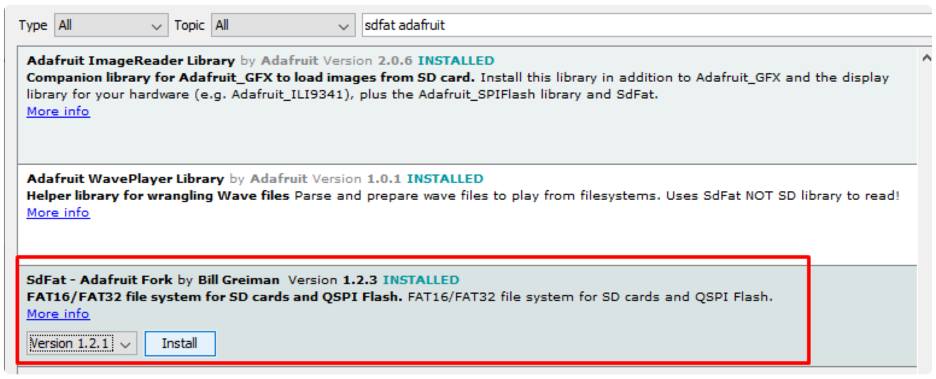
Type All Topic All wave

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

Version 1.0.1 Install

SdFat (Adafruit Fork)

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



Type All Topic All sdfat adafruit

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

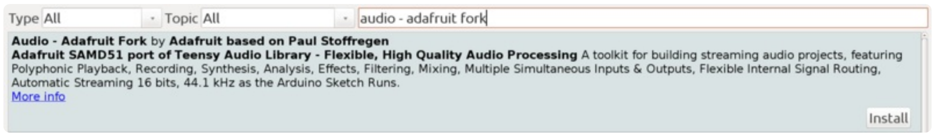
Adafruit WavePlayer Library by Adafruit Version 1.0.1 INSTALLED
Helper library for wrangling Wave files Parse and prepare wave files to play from filesystems. Uses SdFat NOT SD library to read!
More info

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

Version 1.2.1 Install

Audio - Adafruit Fork

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



Type All Topic All audio - adafruit fork

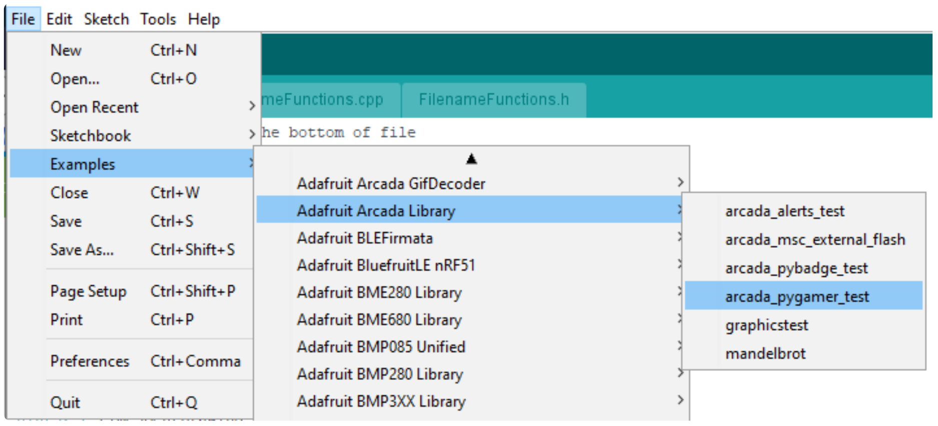
Audio - Adafruit Fork by **Adafruit based on Paul Stoffregen**
Adafruit SAMD51 port of Teensy Audio Library - Flexible, High Quality Audio Processing A toolkit for building streaming audio projects, featuring Polyphonic Playback, Recording, Synthesis, Analysis, Effects, Filtering, Mixing, Multiple Simultaneous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

Install

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

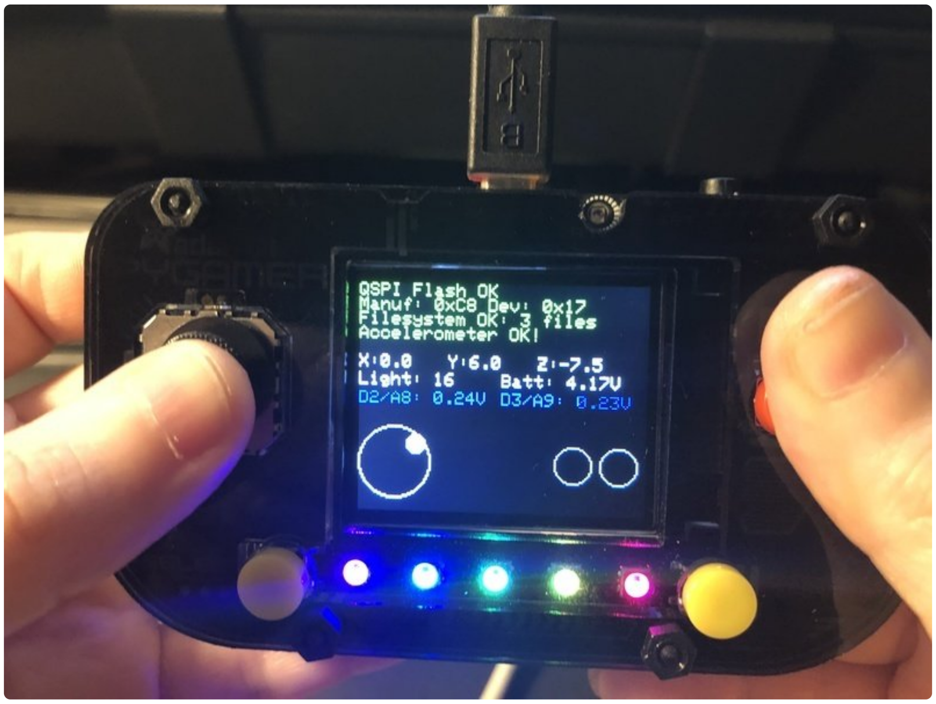
1. Checks the QSPI flash chip initialised correctly, and displays the manufacturer/device ID if so
2. Checks if the QPI flash has a filesystem on it (if not, try loading CircuitPython which will create a filesystem). It will print the # of files found in the root directory
3. Tests if an accelerometer 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 shaked 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.

https://adafru.it/ETU

https://adafru.it/ETU



```
QSPI Flash OK
Manuf: 0xC8 Dev: 0x17
Filesystem OK: 3 files
Accelerometer OK!

X:0.0   Y:6.0   Z:-7.5
Light: 16     Batt: 4.17V
D2/A8: 0.24V D3/A9: 0.23V
```

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)

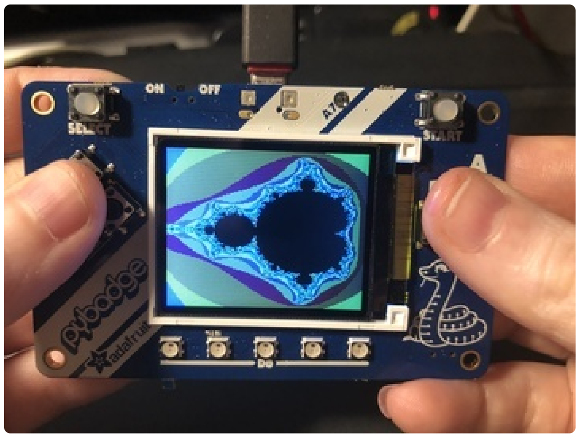
Arcada is a direct subclass of Adafruit_GFX, so if you want to use it to draw shapes and text, check out the GFX guide here first!

https://adafru.it/doL

https://adafru.it/doL

```
Hello World!
3.141593 Want pi?

845FED Print HEX!

Sketch has been
running for:
3 seconds.
```

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 explaination 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
- `filesysBeginMSD()` 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

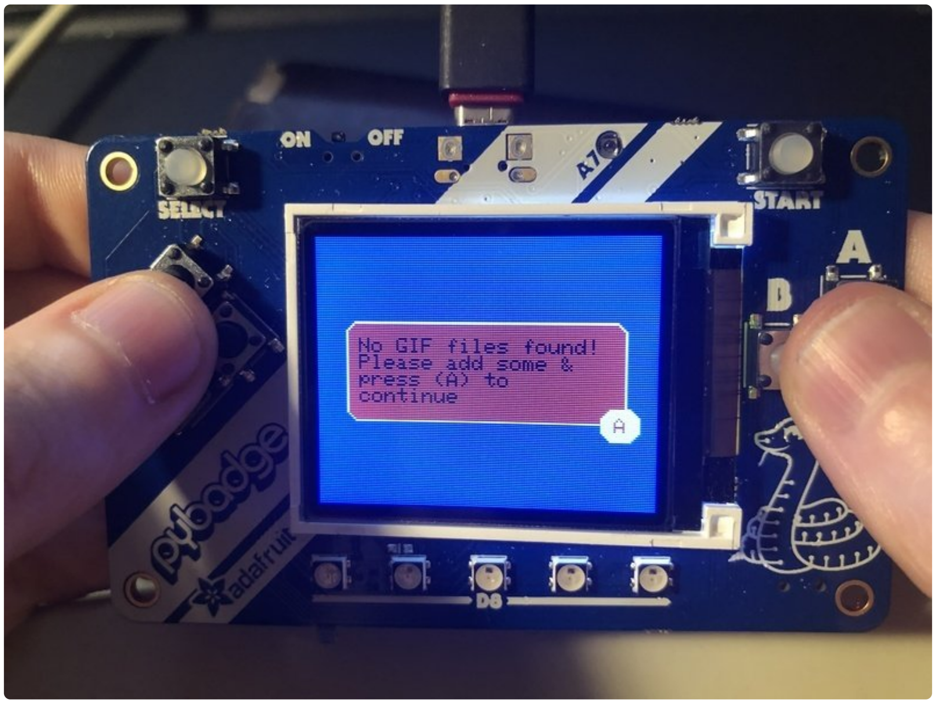
- Enable/disable speaker amplifier (if there is one) with `enableSpeaker` - this doesn't affect headphones if there are any
- `readBatterySensor` returns the battery voltage detected. You cannot detect whether a battery is being charged, only the voltage.
- `readLightSensor` will return 0 for dark, 1023 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



```
No GIF files found!
Please add some &
press (A) to
continue
```

Arcada Library Docs

[Arcada Library Docs](https://adafru.it/RiE) (https://adafru.it/RiE)

Adapting Sketches to M0 & M4

The ATSAMD21 and 51 are very nice little chips, but fairly new as Arduino-compatible cores go. **Most** sketches & libraries will work but here's a collection of things we noticed.

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

Analog References

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

Pin Outputs & Pullups

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

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

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

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

`pinMode(pin, INPUT_PULLUP)`

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

Serial vs SerialUSB

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

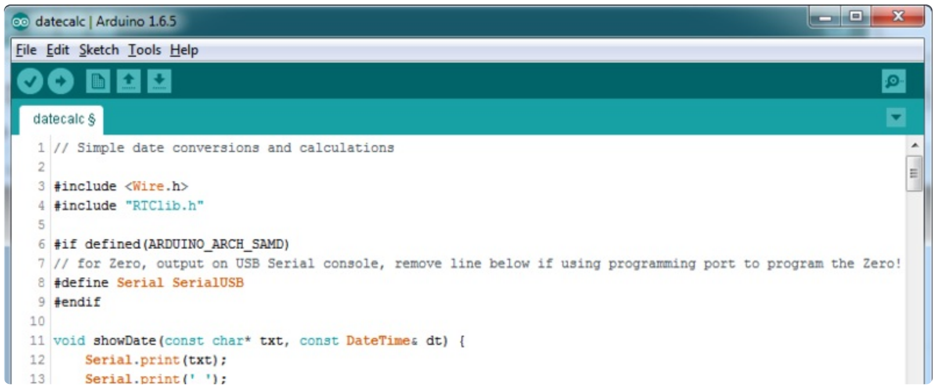
In the Adafruit M0/M4 Core, we fixed it so that **Serial goes to USB so it will automatically work just fine** .

However, on the off chance you are using the official Arduino SAMD core and *not* the Adafruit version (which really, we recommend you use our version because it's been tuned to our boards), and you want your Serial prints and reads to use the USB port, use *SerialUSB* instead of *Serial* in your sketch.

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

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

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



```
// Simple date conversions and calculations

#include <Wire.h>
#include "RTClib.h"

#if defined(ARDUINO_ARCH_SAMD)
// for Zero, output on USB Serial console, remove line below if using programming port to program the Zero!
#define Serial SerialUSB
#endif

void showDate(const char* txt, const DateTime& dt) {
    Serial.print(txt);
    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 AVRs, clicking **reset** while plugged into USB will launch the bootloader manually, the bootloader will time out after a few seconds. For the M0/M4, you'll need to ***double click*** the button. You will see a pulsing red LED to let you know you're in bootloader mode. Once in that mode, it wont time out! Click reset again if you want to go back to launching code.

Aligned Memory Access

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

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

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

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

Floating Point Conversion

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

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

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

http://forum.arduino.cc/index.php?topic=368720.0 (https://adafru.it/IFS)

How Much RAM Available?

The ATSAMD21G18 has 32K of RAM, but you still might need to track it for some reason. You can do so with this handy function:

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

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

Thx to http://forum.arduino.cc/index.php?topic=365830.msg2542879#msg2542879 (https://adafru.it/m6D) for the tip!

Storing data in FLASH

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

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

That string is now in FLASH. You can manipulate the string just like RAM data, the compiler will automatically read from FLASH so you dont need special progmem-knowledgeable functions.

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

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

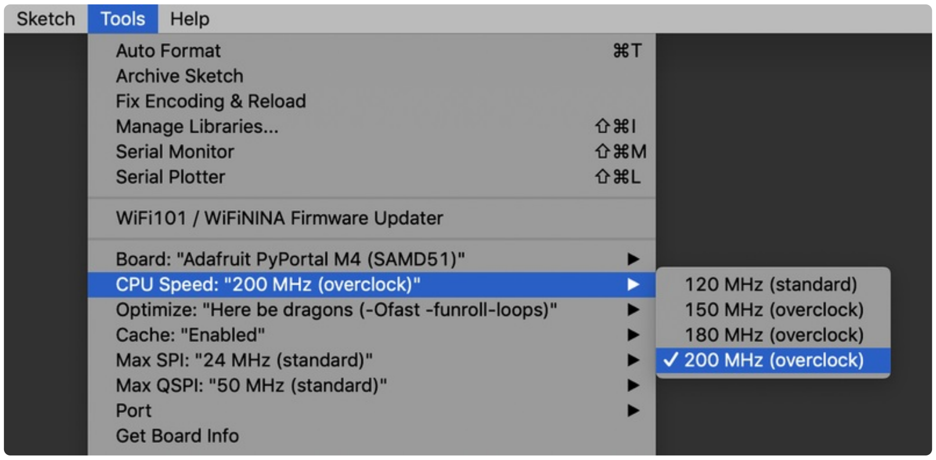
Pretty-Printing out registers

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

https://github.com/drewfish/arduino-ZeroRegs (https://adafru.it/Bet)

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.



Board: "Adafruit PyPortal M4 (SAMD51)"
CPU Speed: "200 MHz (overclock)"
Optimize: "Here be dragons (-Ofast -funroll-loops)"
Cache: "Enabled"
Max SPI: "24 MHz (standard)"
Max QSPI: "50 MHz (standard)"
✓ 200 MHz (overclock)

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

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

CPU Speed (overclocking)

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

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

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

Much more likely, **some code or libraries may not play well** with the nonstandard CPU speed. For example, currently the NeoPixel library assumes a 120 MHz CPU speed and won't issue the correct data at other settings (this will be worked on). Other libraries may exhibit similar problems, usually anything that strictly depends on CPU timing...you might encounter problems with audio- or servo-related code depending how it’s written. **If you encounter such code or libraries, set the CPU speed to the default 120 MHz and re-upload.**

Optimize

There’s usually more than one way to solve a problem, some more resource-intensive than others. Since Arduino got its start on resource-limited AVR microcontrollers, the C++ compiler has always aimed for the **smallest compiled program size**. The “Optimize” menu gives some choices for the compiler to take different and often faster approaches, at the expense of slightly larger program size...with the huge flash memory capacity of M4 devices, that’s rarely a problem now.

The “**Small**” setting will compile your code like it always has in the past, aiming for the smallest compiled program size.

The “**Fast**” setting invokes various speed optimizations. The resulting program should produce the same results, is slightly larger, and usually (but not always) noticably faster. It’s worth a shot!

“**Here be dragons**” invokes some more intensive optimizations...code will be larger still, faster still, but there’s a possibility these optimizations could cause unexpected behaviors. *Some code may not work the same as before.* Hence the name. Maybe you’ll discover treasure here, or maybe you’ll sail right off the edge of the world.

Most code and libraries will continue to function regardless of the optimizer settings. If you do encounter problems, **dial it back one notch and re-upload**.

Cache

This option allows a small collection of instructions and data to be accessed more quickly than from flash memory, boosting performance. It’s enabled by default and should work fine with all code and libraries. But if you encounter some esoteric situation, the cache can be disabled, then recompile and upload.

Max SPI and Max QSPI

These should probably be left at their defaults. They’re present mostly for our own experiments and can cause **serious headaches**.

Max SPI determines the clock source for the M4’s SPI peripherals. Under normal circumstances this allows transfers up to 24 MHz, and should usually be left at that setting. But...if you’re using write-only SPI devices (such as TFT or OLED displays), this option lets you drive them faster (we’ve successfully used 60 MHz with some TFT screens). The caveat is, if using *any* read/write devices (such as an SD card), *this will not work at all...*SPI reads *absolutely* max out at the default 24 MHz setting, and anything else will fail. **Write = OK. Read = FAIL.** This is true *even if your code is using a lower bitrate setting...* just having the different clock source prevents SPI reads.

Max QSPI does similarly for the extra flash storage on M4 “Express” boards. *Very few* Arduino sketches access this storage at all, let alone in a bandwidth-constrained context, so this will benefit next to nobody. Additionally, due to the way clock dividers are selected, this will only provide some benefit when certain “CPU Speed” settings are active. Our PyPortal Animated GIF Display (https://adafru.it/EkO) runs marginally better with it, if using the QSPI flash.

Enabling the Buck Converter on some M4 Boards

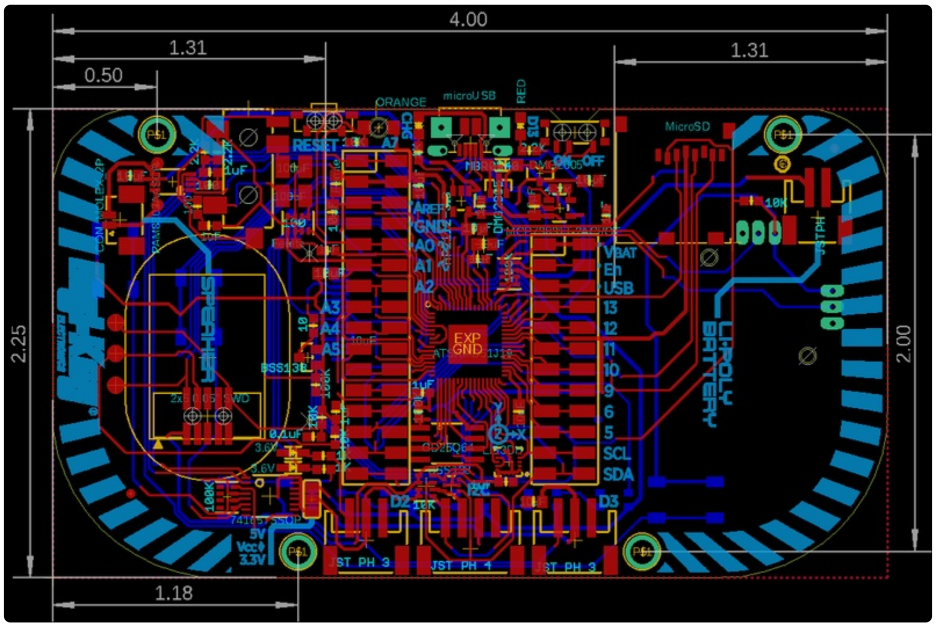
If you want to reduce power draw, some of our boards have an inductor so you can use the 1.8V buck converter instead of the built in linear regulator. If the board does have an inductor (see the schematic) you can add the line `SUPC->VREG.bit.SEL = 1;` to your code to switch to it. Note it will make ADC/DAC reads a bit noisier so we don't use it by default. You'll save ~4mA (https://adafru.it/F0H).

Downloads

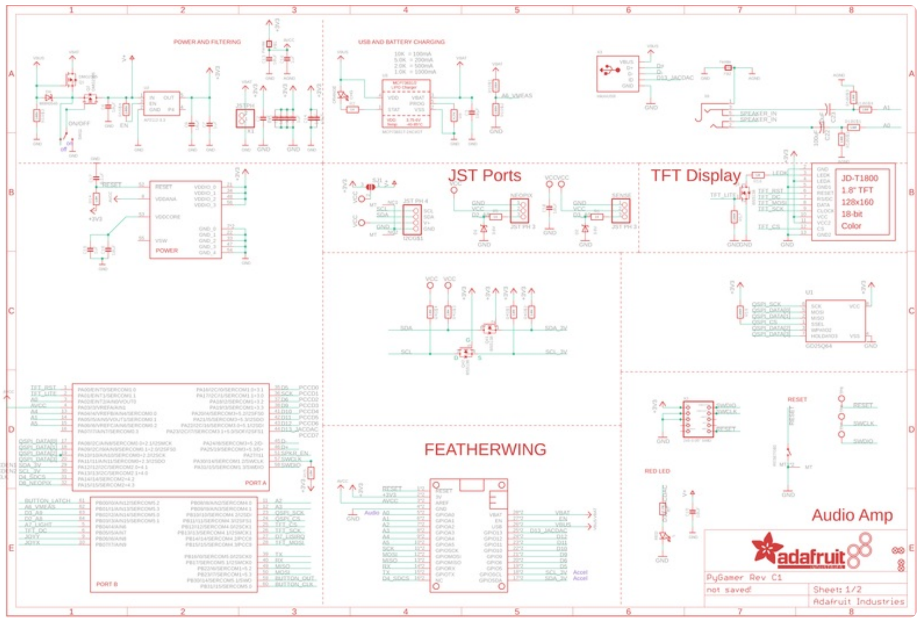
Files:

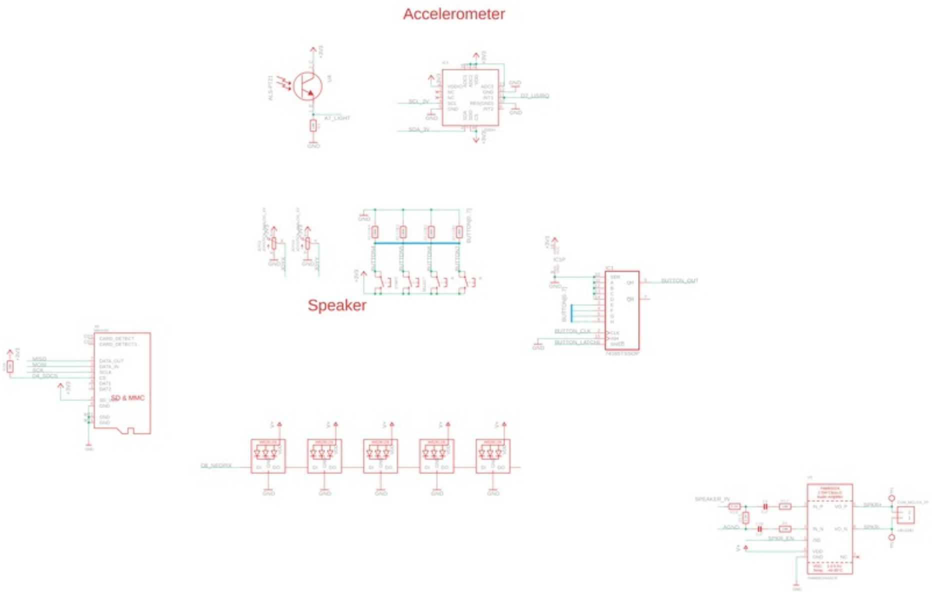
- ATSAMD51J19 Product page w/datasheets (https://adafru.it/Bf8)
- LIS3DH Datasheet (https://adafru.it/uBy)
- EagleCAD PCB files on GitHub (https://adafru.it/ETc)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/ETd)
- 3D Model on GitHub (https://adafru.it/EVr)

Fab Print



Schematic





Laser Cut Acrylic Case

https://adafru.it/ETW

https://adafru.it/ETW

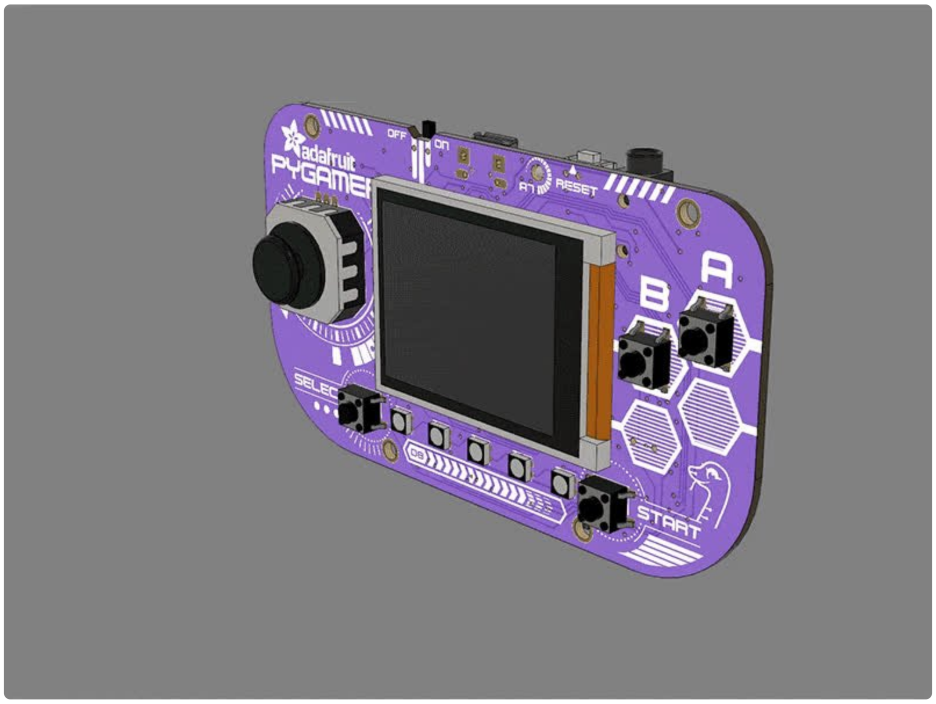
https://adafru.it/ETX

https://adafru.it/ETX

https://adafru.it/ETY

https://adafru.it/ETY

3D Model



https://adafru.it/EVr

https://adafru.it/EVr

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.

https://adafru.it/Fdg

https://adafru.it/Fdg

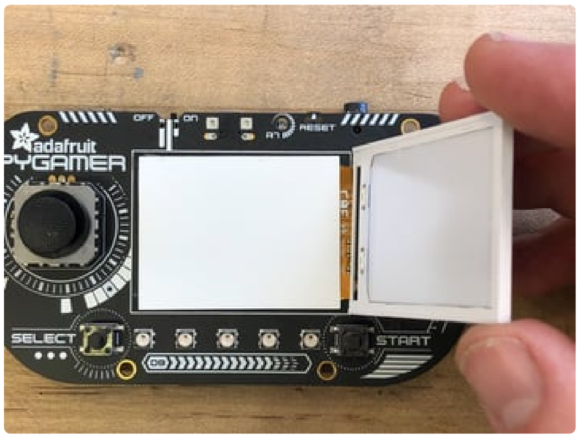
Plug in your PyGamer to your computer with a known good USB data cable (not a "charge-only" cable which should be killed and burned with fire) then turn on the PyGamer and press the reset button to bring up the bootloader. (Also, try both with and without a USB hub if you have one, sometimes they help, 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.



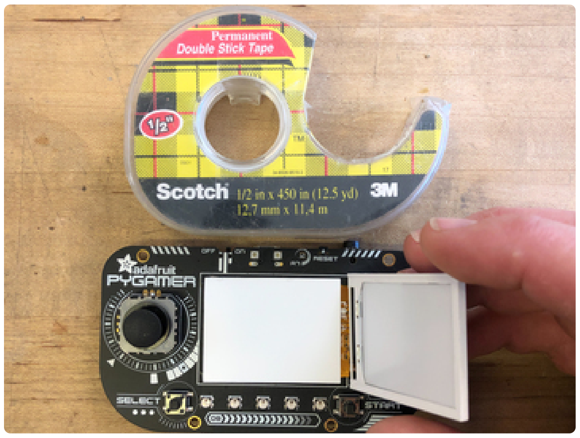
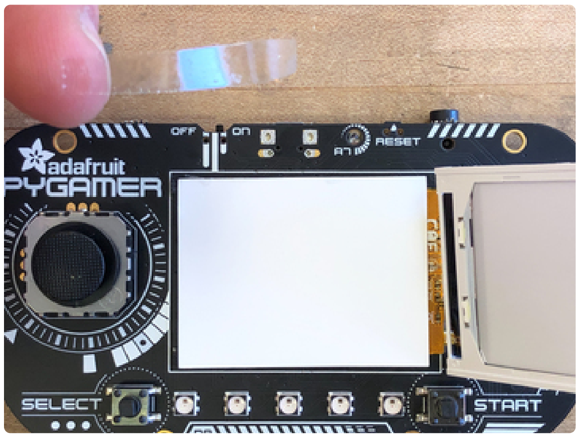
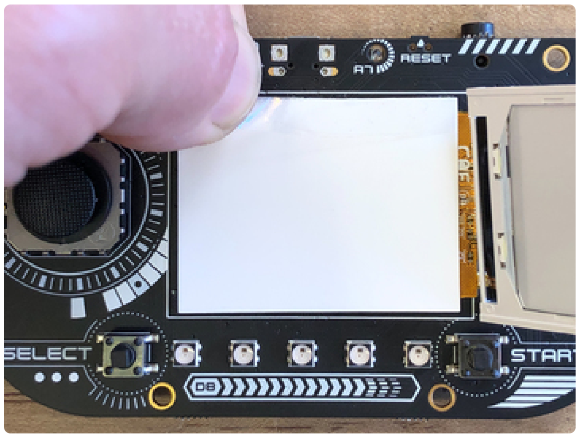
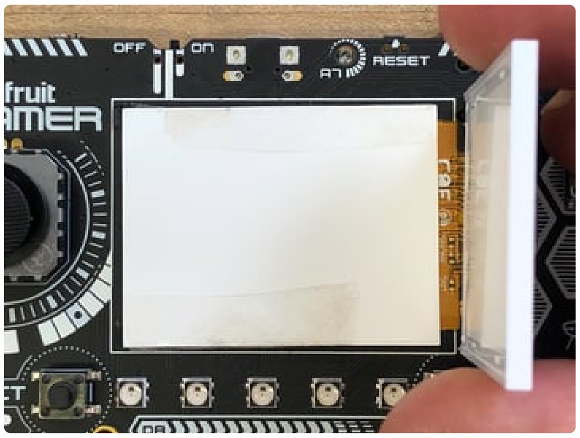
The small bits of adhesive may have given up the fight. Go ahead and lift the screen away from the backing, being careful to go gentle on the ribbon connector!

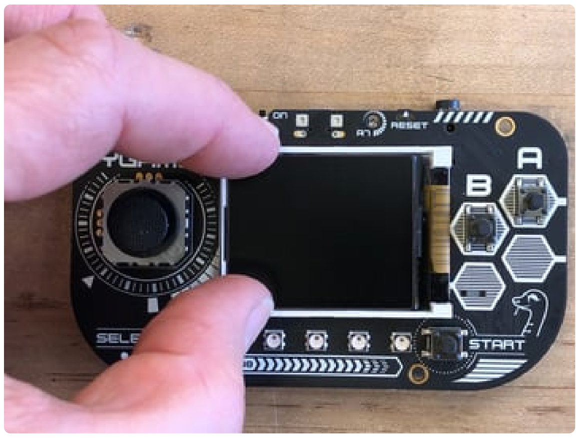
Get some good double stick tape, as shown.

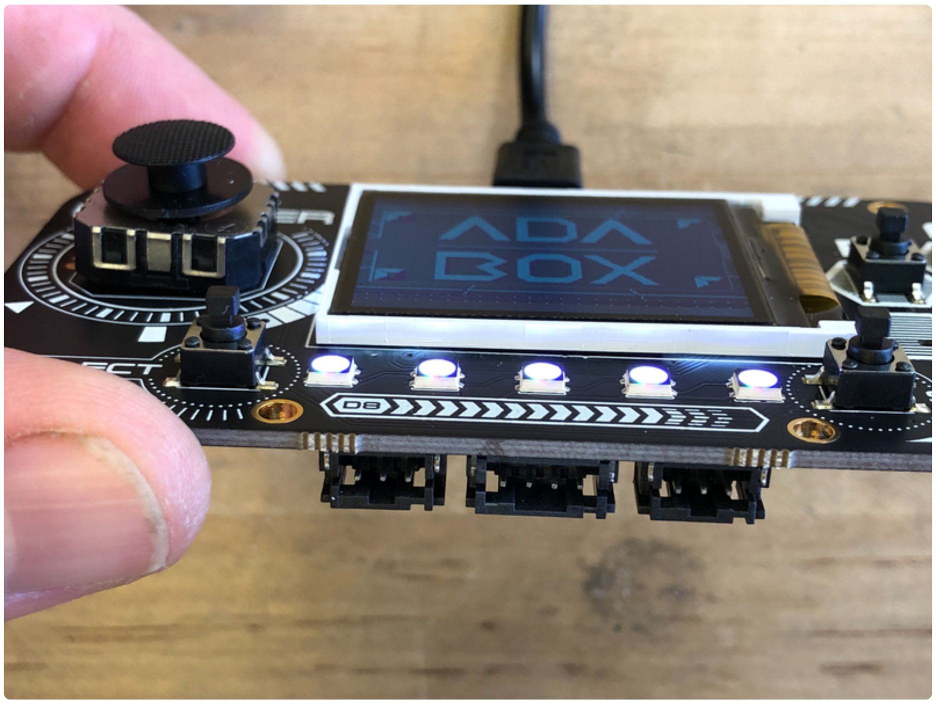
Cut off two short lengths of the tape, you can even use one piece and cut it in half length-wise to make narrow strips.

Place a piece of tape across the bottom and top edges of the backing.

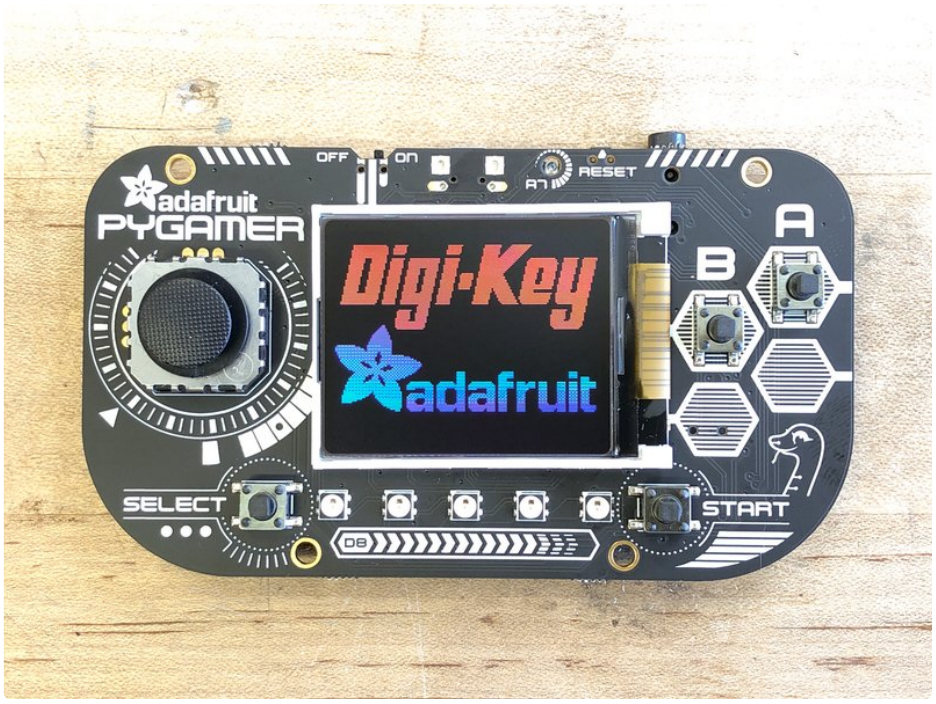
Press the screen down firmly and hold for 30 seconds to ensure good adhesion.


Permanent
Double Stick Tape
Scotch
1/2 in x 450 in (12.5 yd)
12,7 mm x 11,4 m
3M
adafruit
PYGAMER
SELECT
START

OFF
ON
RESET
adafruit
PYGAMER
SELECT
START

RESET
SELECT
START

OFF
ON
RESET





Your PyGamer should be better than ever now, and ready to play!


adafruit
PYGAMER
OFF
ON
A7
RESET
A
B
Digi-Key
adafruit
SELECT
START
D8