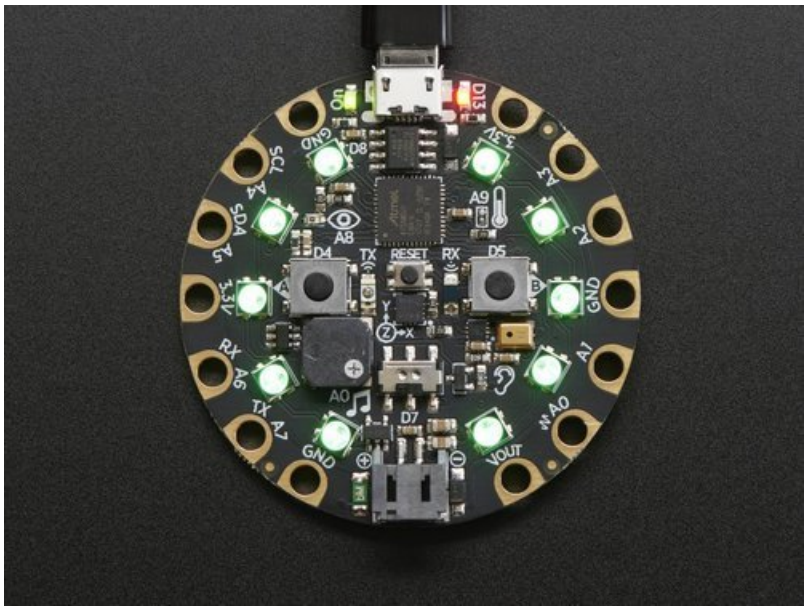
adafruit learning system

Adafruit Circuit Playground Express

Created by lady ada



Last updated on 2018-01-24 01:44:27 AM UTC

Guide Contents

Guide Contents 2
Overview 7
Classic vs. Express 10
How to tell if you have a Classic 10
How to tell if you have an Express 10
Guided Tour 12
Power and Data 12
- Micro B USB connector 12
- JST Battery Input 13

Alligator/Croc Clip Pads 13
Microchips 13
LEDs 14
- Green ON LED 15
- Red #13 LED 15
- 10 x Color NeoPixel LED 15

Speaker 16
Sensors 16
- Light Sensor 17
- Temperature Sensor 17
- Microphone Audio Sensor 18
- Motion Sensor 18
- Capacitive Touch 19

Switches & Buttons 19
Infrared Receive/Transmit & Proximity 20
Pinouts 22
Power Pads 22
Input/Output Pads 23
- Common to all pads 24

Each Pin! 24
Internally Used Pins! 25
Windows Driver Installation 26
Manual Driver Installation 27
MakeCode 29
What is MakeCode? 30
- Circuit Playground Express 30
- Circuit Playground Express - Base Kit 31

Editing Blocks 32
- Blinky! 32

Downloading and Flashing 33
- Step 1: Bootloader mode 33
- Step 2: Compile and Download 34
- Running MakeCode that's Already Loaded 35

Saving and Sharing 36

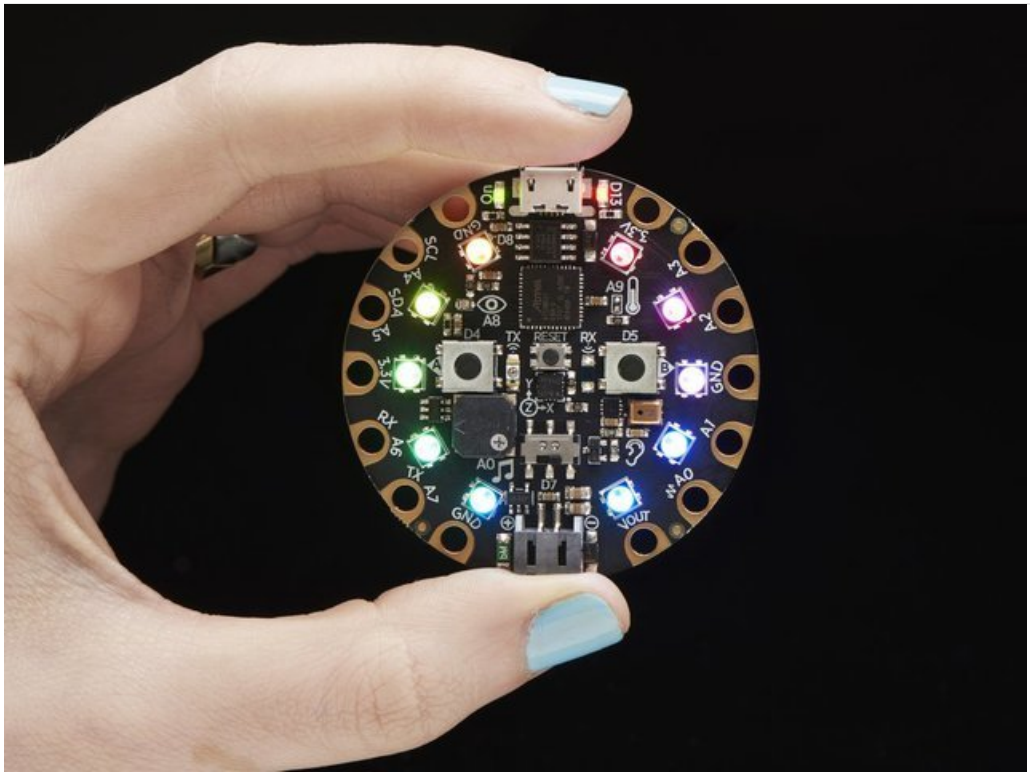
Extracting your code from the Circuit Playground 36
Sharing 36
Editing JavaScript 38
Apps 39
Windows Store 39
Node.JS 39
Other Good Stuff 40
GitHub packages 40
We are Open Source on GitHub 40
We have crowd-sourced translations 40
What is CircuitPython? 41
CircuitPython is based on Python 41
Why would I use CircuitPython? 41
CircuitPython 43
Install or update CircuitPython! 43
Further Information 45
Installing Mu Editor 46
Installing Mu for Windows or Mac OS X 46
Installing Mu for Linux 47
Using Mu 47
Creating and Editing Code 50
Creating Code 50
Editing Code 52
Your code changes are run as soon as the file is done saving. 52
1. Use an editor that writes out the file completely when you save it. 52
2. Eject or Sync the Drive After Writing 53
Oh No I Did Something Wrong and Now The CIRCUITPY Drive Doesn't Show Up!!! 53
Back to Editing Code... 53
Exploring Your First CircuitPython Program 53
Imports & Libraries 54
Setting Up The LED 54
Loop-de-loops 54
More Changes 55
Naming Your Program File 55
Connecting to the Serial Console 56
Are you using Mu? 56
Using Something Else? 57
Interacting with the Serial Console 58
The REPL 61
Returning to the serial console 64
CircuitPython Libraries 66
Installing the CircuitPython Library Bundle 66
Express Boards 67
Non-Express Boards 68
Example: ImportError Due to Missing Library 68
Library Install on Non-Express Boards 69

Updating CircuitPython Libraries 69
Troubleshooting 70
CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present 70
You may have a different board. 70
MakeCode 70
Windows 10 70
Windows 7 70
CircuitPython RGB Status Light 71
CIRCUITPY Drive Issues 71
For the Circuit Playground Express, Feather M0 Express, and Metro M0 Express: 72
For Non-Express Boards with a UF2 bootloader (Gemma M0, Trinket M0): 72
For non-Express Boards without a UF2 bootloader (Feather M0 Basic Proto, Feather Adalogger, Arduino Zero):
Running Out of File Space on Non-Express Boards 73 73
Delete something! 73
Use tabs 73
Mac OSX loves to add extra files. 73
Prevent & Remove Mac OSX Hidden Files 73
Copy Files on Mac OSX Without Creating Hidden Files 74
Other Mac OSX Space-Saving Tips 74
Welcome to the Community! 76
Adafruit Discord 76
Adafruit Forums 77
Adafruit Github 78
ReadTheDocs 79
CircuitPython Playground 80
CircuitPython Built-Ins 81
Things that are Built In and Work 81
flow control 81
math 81
tuples, lists, arrays, and dictionaries 81
classes/objects and functions 81
lambdas 81
Things to watch out for! 81
CircuitPython Digital In & Out 83
Going Beyond the Lesson! 84
Experiment 1 84
Experiment 2 84
CircuitPython Analog In 86
Creating an Analog Input 87
GetVoltage Helper 87
Main Loop 87
CircuitPython Analog Out 89
Creating an analog output 89
Setting the analog output 89
Main Loop 89
CircuitPython Audio Out 91
Basic Tones 91
Playing Audio Files 92

CircuitPython Cap Touch 95
Creating an capacitive touch input 96
Main Loop 96
Copper Foil Tape with Conductive Adhesive - 6mm x 15 meter roll 98
Copper Foil Tape with Conductive Adhesive - 25mm x 15 meter roll 99
Small Alligator Clip Test Lead (set of 12) 99
CircuitPython NeoPixel 100
CircuitPython HID Keyboard 104
CircuitPython CPU Temp 107
Playground Light Sensor 108
Playground Temperature 110
Playground Drum Machine 112
Playground Sound Meter 116
Handy Tips 119
Check Heap Memory Usage 119
Random Numbers 119
Arduino 120
Set Up Arduino IDE 121
Install SAMD Support 121
Install Drivers (Windows 7 Only) 122
Blink 122
Sucessful Upload 123
Compilation Issues 123
Manually bootloading 124
Ubuntu & Linux Issue Fix 124
Arduino Switches 125
Without Library Assist 125
Adapting Sketches to M0 127
Analog References 127
Pin Outputs & Pullups 127
Serial vs SerialUSB 127
AnalogWrite / PWM on Feather/Metro M0 128
analogWrite() PWM range 129
Missing header files 129
Bootloader Launching 129
Aligned Memory Access 129
Floating Point Conversion 130
How Much RAM Available? 130
Storing data in FLASH 130
UF2 Bootloader Details 131
Entering Bootloader Mode 131
Using the Mass Storage Bootloader 133
Using the BOSSA Bootloader 134

Windows 7 Drivers 134
Verifying Serial Port in Device Manager 135
Running bossac on the command line 136
Updating the bootloader 137
Getting Rid of Windows Pop-ups 138
Making your own UF2 139
Downloads 140
Files: 140
Circuit Playground Express Schematic 140

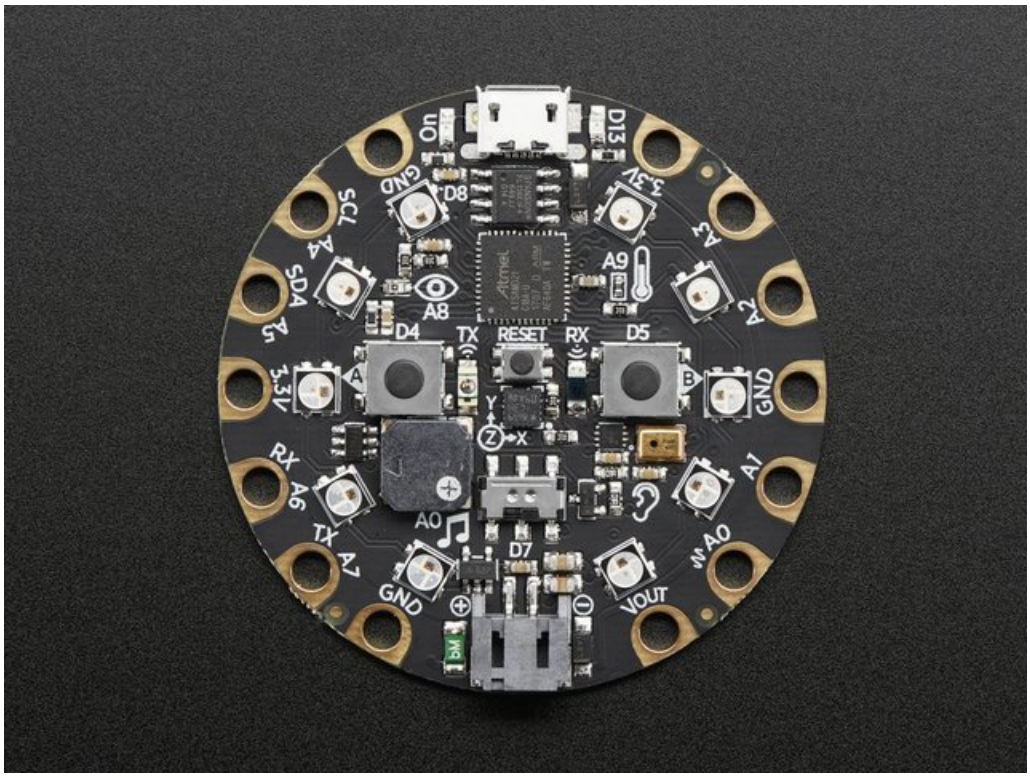
Overview



Circuit Playground Express is the next step towards a perfect introduction to electronics and programming. We've taken the original Circuit Playground Classic and made it even better! Not only did we pack even more sensors in, we also made it even easier to program.

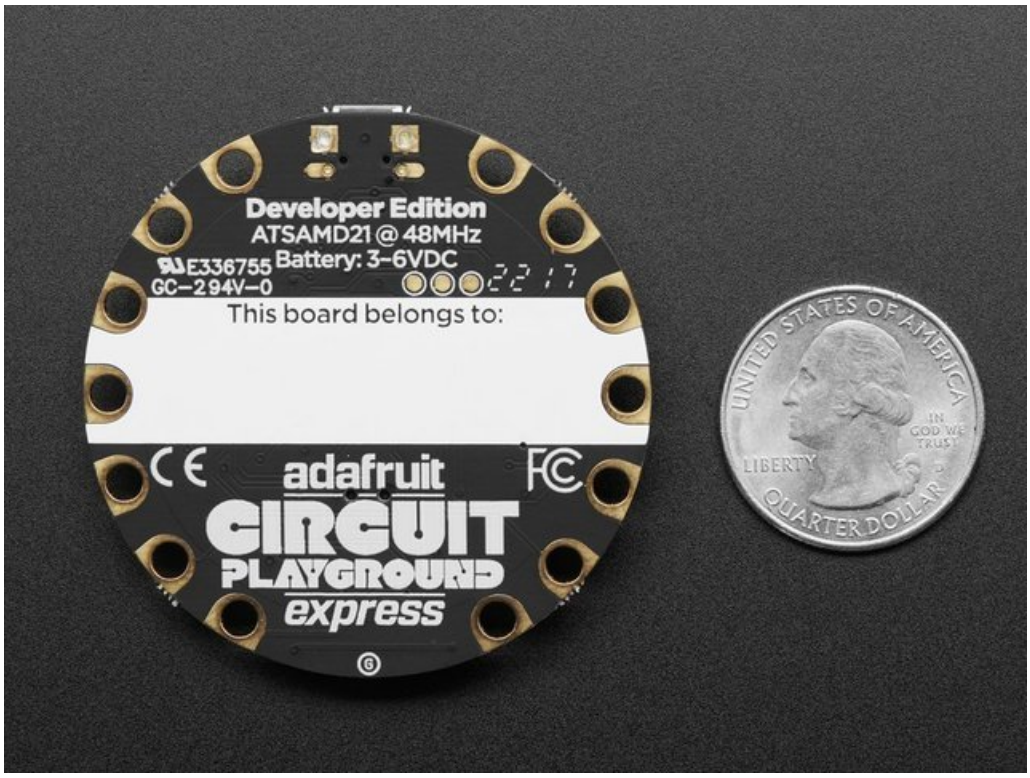
Start your journey with **Microsoft MakeCode** block-based or Javascript programming. Then, you can use the same board to try CircuitPython, with the Python interpreter running right on the Express. As you progress, you can advance to using **Arduino IDE**, which has full support of all the hardware down to the low level, so you can make powerful projects.

Because you can program the same board in 3 different ways - the Express has great value and re-usability. From beginners to experts, Circuit Playground Express has something for everyone.



Here's some of the great goodies baked in to each Circuit Playground Express:

- 10 x mini NeoPixels, each one can display any color
- 1 x Motion sensor (LIS3DH triple-axis accelerometer with tap detection, free-fall detection)
- 1 x Temperature sensor (thermistor)
- 1 x Light sensor (phototransistor). Can also act as a color sensor and pulse sensor.
- 1 x Sound sensor (MEMS microphone)
- 1 x Mini speaker with class D amplifier (7.5mm magnetic speaker/buzzer)
- 2 x Push buttons, labeled A and B
- 1 x Slide switch
- Infrared receiver and transmitter - can receive and transmit any remote control codes, as well as send messages between Circuit Playground Expresses. Can also act as a proximity sensor.
- 8 x alligator-clip friendly input/output pins
- Includes I2C, UART, 8 pins that can do analog inputs, multiple PWM output
- 7 pads can act as capacitive touch inputs and the 1 remaining is a true analog output
- Green "ON" LED so you know its powered
- Red "#13" LED for basic blinking
- Reset button
- ATSAMD21 ARM Cortex M0 Processor, running at 3.3V and 48MHz
- 2 MB of SPI Flash storage, used primarily with CircuitPython to store code and libraries.
- MicroUSB port for programming and debugging
- USB port can act like serial port, keyboard, mouse, joystick or MIDI!


Developer Edition
ATSAMD21 @ 48MHz
E336755
Battery: 3-6VDC
GC-294V-0
This board belongs to:
adafruit
CIRCUIT
PLAYGROUND
express
UNITED STATES OF AMERICA
LIBERTY
IN GOD WE TRUST
QUARTER DOLLAR

Classic vs. Express

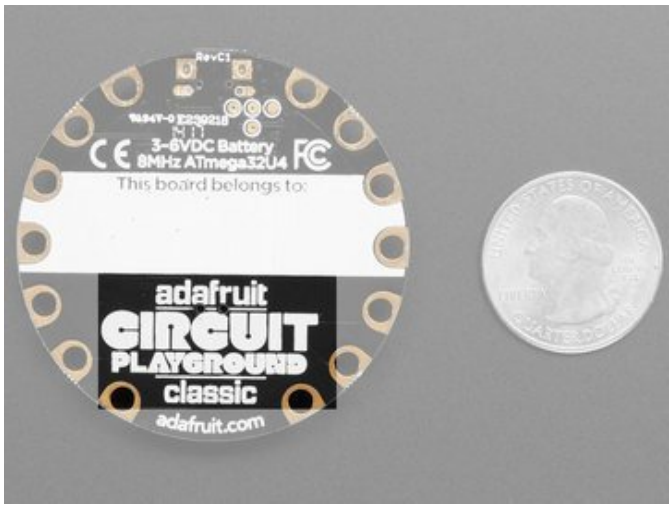
Circuit Playground started its life as a board with simple requirements - just work with Arduino IDE and Code.org. But since it's initial launch in 2015 we've learned a lot and improved the board greatly!

There are **TWO** Circuit Playgrounds - one **Classic** and one **Express**.

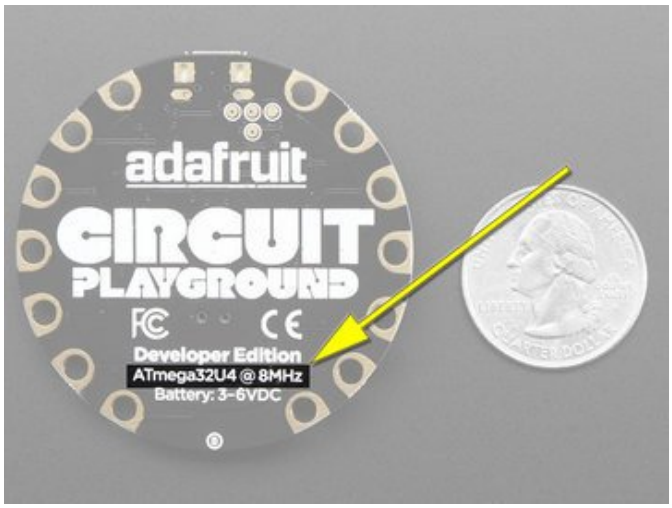
The Classic version can run Arduino and Code.org

The Express version can run MakeCode, CircuitPython and Arduino (Code.org support is not ready at this time)

How to tell if you have a Classic

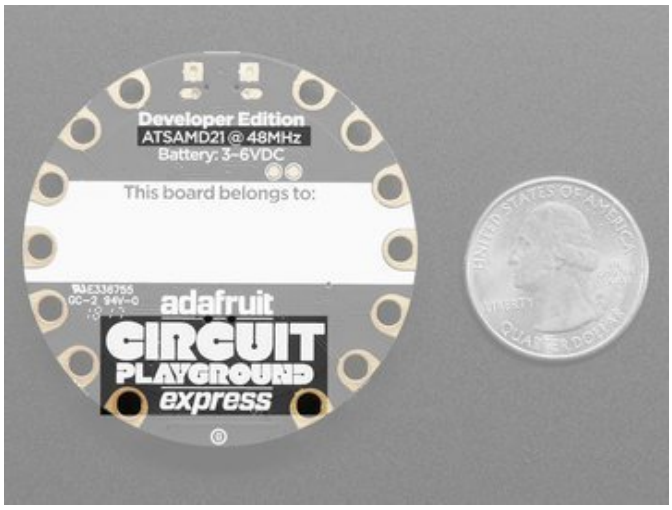


Current Circuit Playground Classic boards have **Classic** written on the back lower half



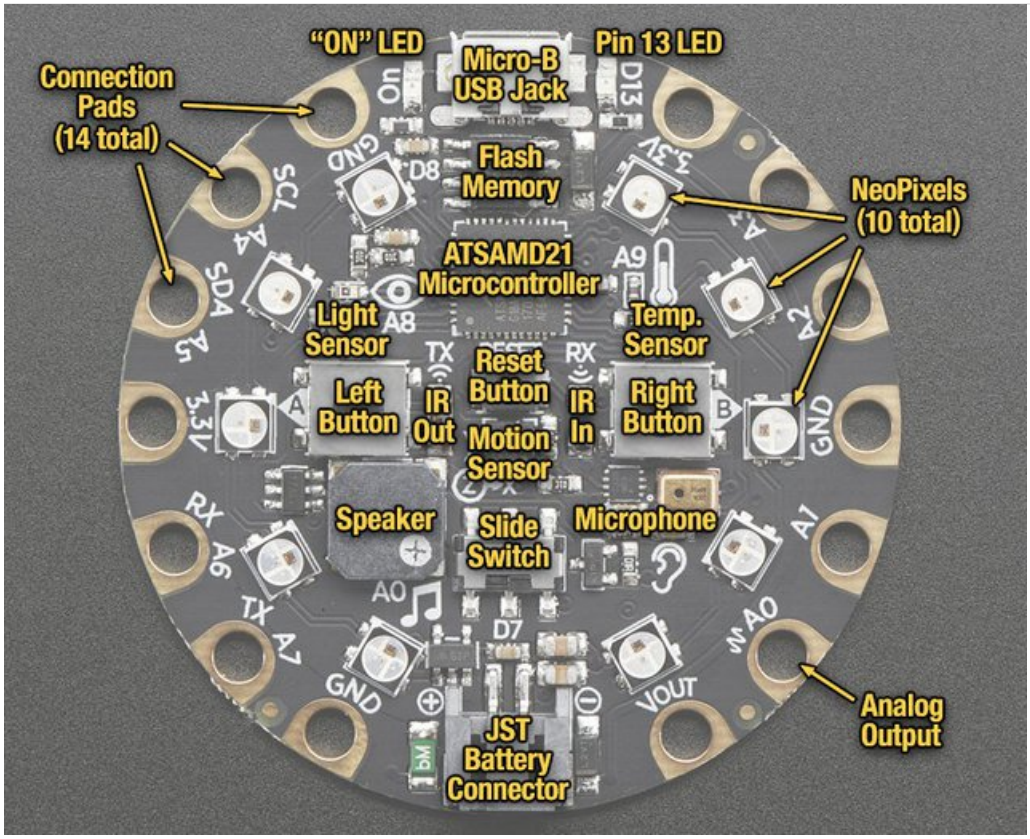
Older Circuit Playground Classics didn't have the word **Classic** on the back, but they do have text that says the chip type, **ATmega32U4**

How to tell if you have an Express



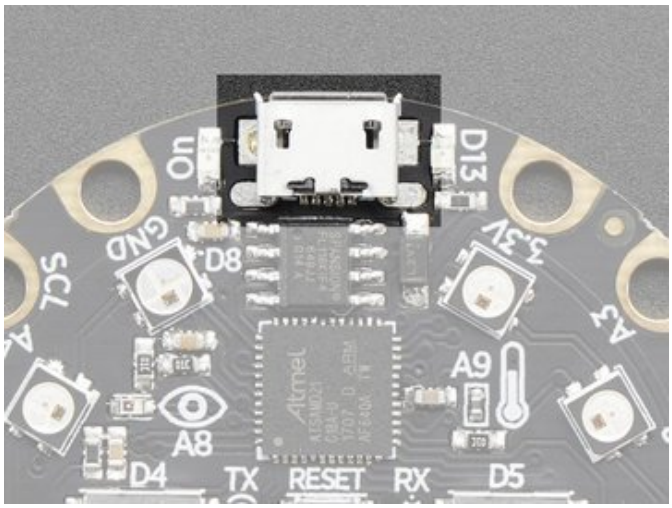
Express boards have **CIRCUIT PLAYGROUND EXPRESS** on the back lower half. They'll also note the chip is an **ATSAMD21**

Guided Tour



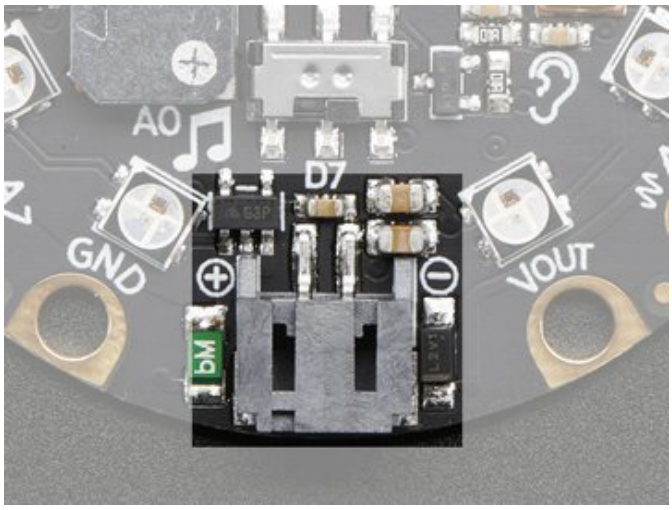
Let me take you on a tour of your Circuit Playground Express (we'll shorten that to **CPX**). Each CPX is assembled here at Adafruit and comes chock-full of good design to make it a joy to use.

Power and Data



Micro B USB connector

This is at the top of the board. We went with the tried and true micro-B USB connector for power and/or USB communication (bootloader, serial, HID, etc). Use with any computer with a standard data/sync cable.

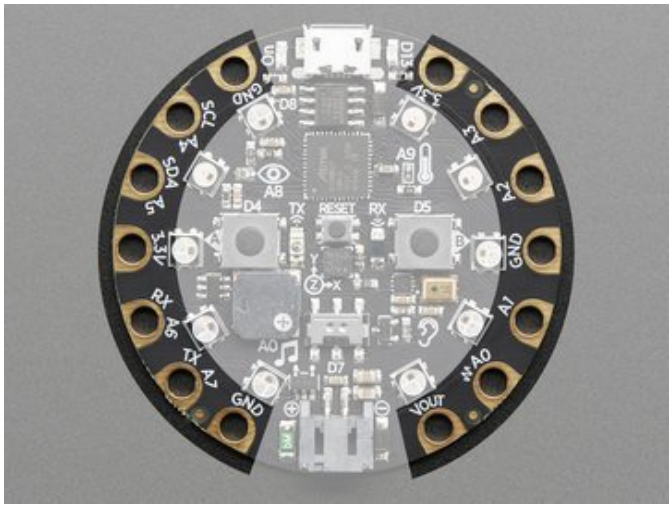


JST Battery Input

This is at the bottom of the board. You can take your CPX anywhere and power it from an external battery. This pin can take up 6V DC input, and has reverse-polarity, over-current and thermal protections. The circuitry inside will use either the battery input power or USB power, safely switching from one to the other. If both are connected, it will use whichever has the higher voltage. Works great with a Lithium Polymer battery or our 3xAAA battery packs with a JST connector on the end. There is no built in battery charging (so that you can use Alkaline *or* Lithium batteries safely)

Alligator/Croc Clip Pads

To make it super-easy to connect to the microcontroller, we have 14 connection pads. You can solder to them, use alligator/croc clips, sew with conductive thread, even use small metal screws!

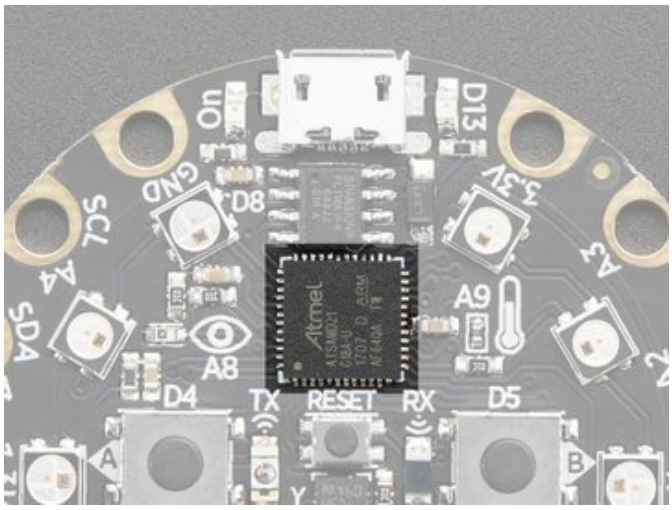


Of the 14 pads, you get a wide range of power pins, I2C, UART, Analog In, Digital In/Out, PWM, and Analog Out.

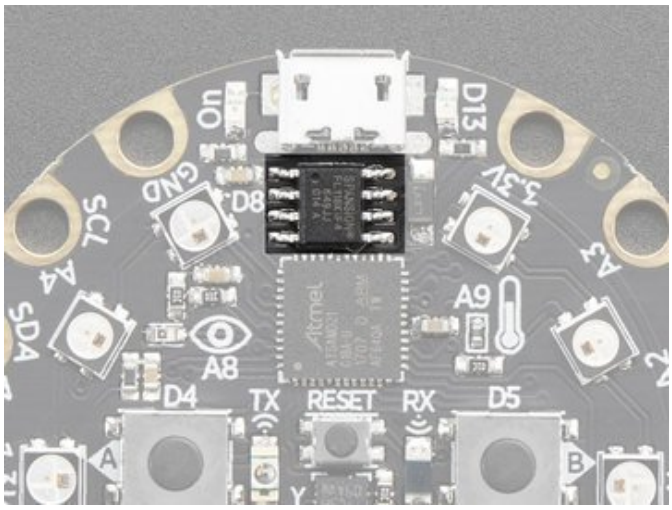
Some of them can even sense the touch of your finger!

See the next pinouts page for more details!

Microchips

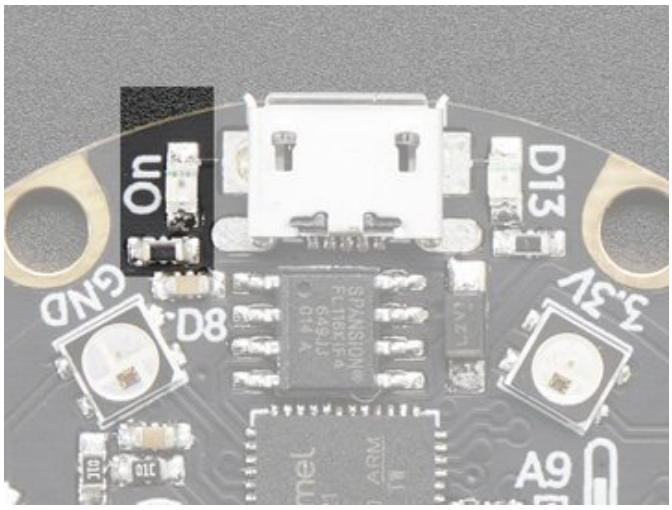


The **brains** of the operation here is the **ATSAMD21G18** a Cortex M0 microcontroller. This chip is much more powerful than the original Circuit Playground. It sits in the top center, and is what you use to run MakeCode, CircuitPython or Arduino!



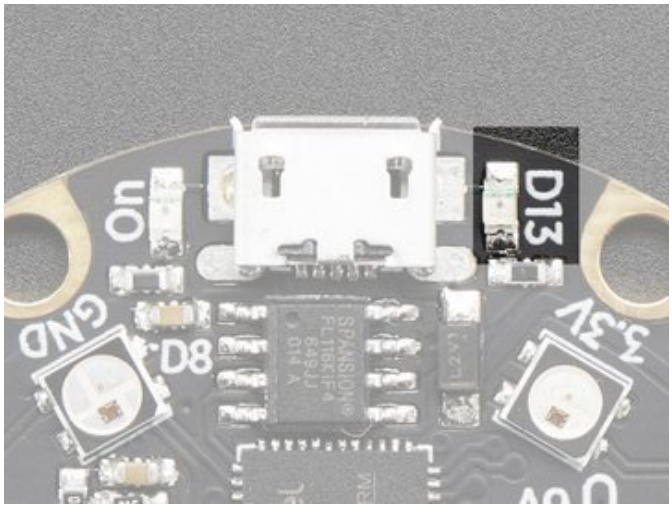
New in Circuit Playground Express, we have added a new storage chip, called SPI Flash. This is a very, very small disk drive, only 2 MB large. You can use this in Arduino or CircuitPython to store files. In CircuitPython this is where all your code lives, and what you see when you use the **CIRCUITPY** drive on your computer

LEDs



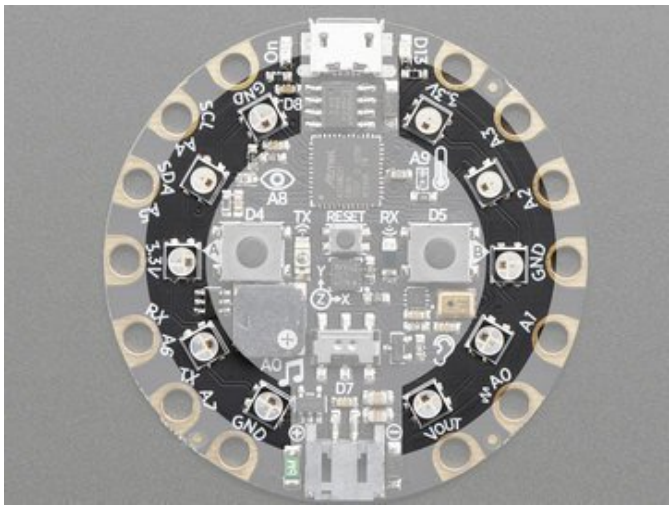
Green ON LED

To the left of the USB connector. This LED lets you know that the CPX is powered on. If it's lit, power is good! If it's dim, flickering or off, there's a power problem and you will have problems. You can't disable this light, but you *can* cover it with electrical tape if you want to make it black.



Red #13 LED

To the right of the USB connector. This LED does double duty. Its connected with a series resistor to the digital #13 GPIO pin. It pulses nicely when the CPX is in bootloader mode, and its also handy for when you want an indicator LED. Many first projects blink this LED to prove that programming worked.

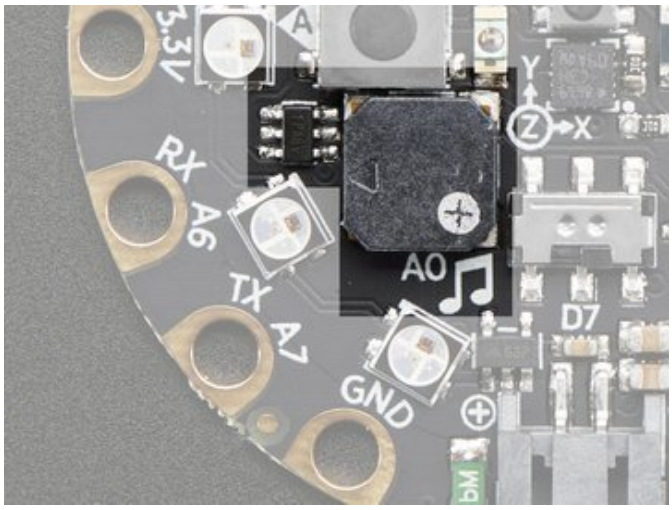


10 x Color NeoPixel LED

The ten LEDs surrounding the outer edge of the boards are all full color, RGB LEDs, each one can be set to any color in the rainbow. Great for beautiful lighting effects! The NeoPixels will also help you know when the bootloader is running (they will turn green) or if it failed to initialize USB when connected to a computer (they will turn red).

Speaker

We have upgraded the buzzer from the original Circuit Playground to be a real speaker, it's not going to compete with your HiFi stereo, but it can play simple songs and tones



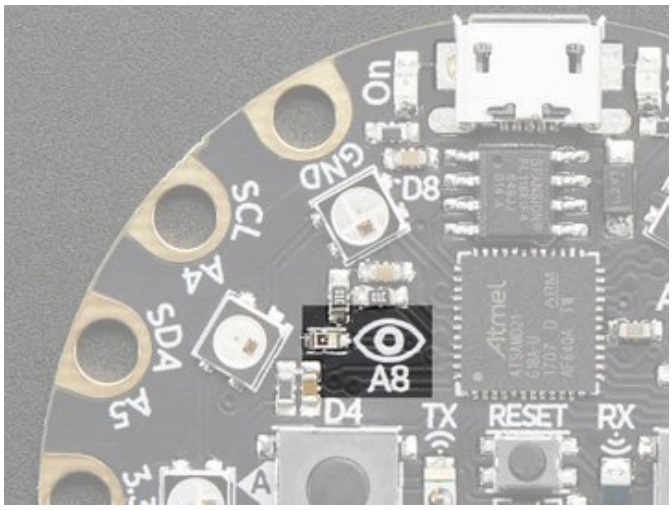
The speaker is the squarish gray chunk on the bottom left of the board. There is a small class D amplifier connected to the speaker so it can get quite loud! Note: it won't sound good if too loud, so some experimentation may be necessary

The amplifier is connected to the true analog output **A0** pin -- this pin is also available on one of the connection pads in the lower right. You can tell it's the analog output pin because of the squiggly symbol.

If you do not want the internal speaker to make noise, you can turn it off using the shutdown control on pin **#11**

Sensors

The Circuit Playground Express has a large number of sensor **inputs** that let you add all sorts of interactivity to your project.

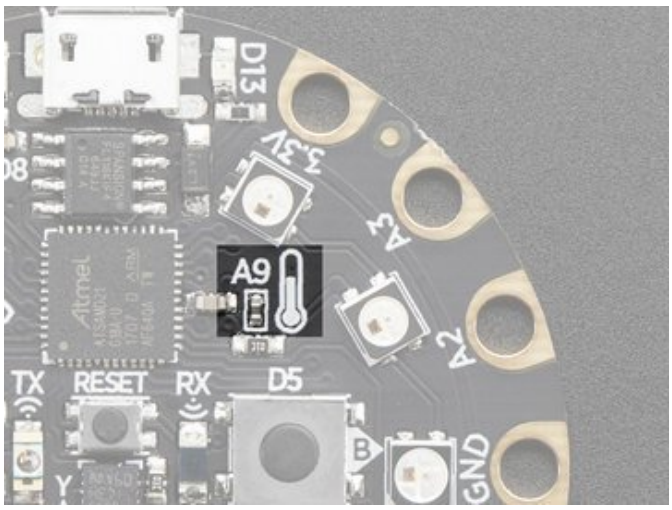


Light Sensor

There is an analog light sensor, part number ALS-PT19 (https://adafru.it/tC2), in the top left part of the board. This can be used to detect ambient light, with similar spectral response to the human eye.

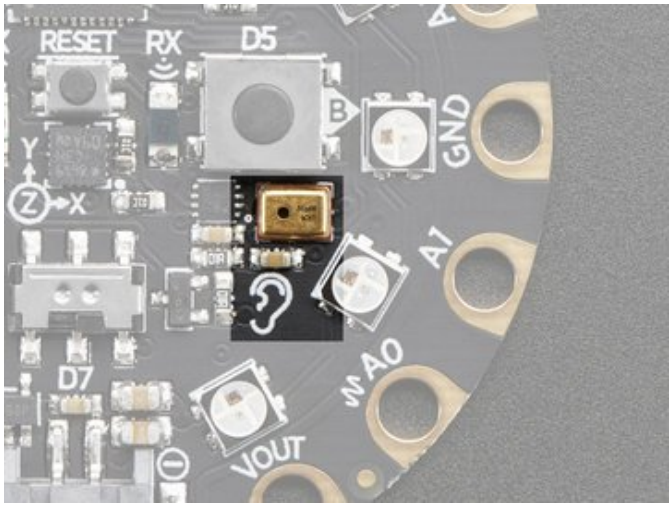
This sensor is connect to analog pin **A8** and will read between 0 and 1023 with higher values corresponding to higher light levels. A reading of about 300 is common for most indoor light levels.

With some clever code, you can use this as a color sensor or even a pulse sensor!



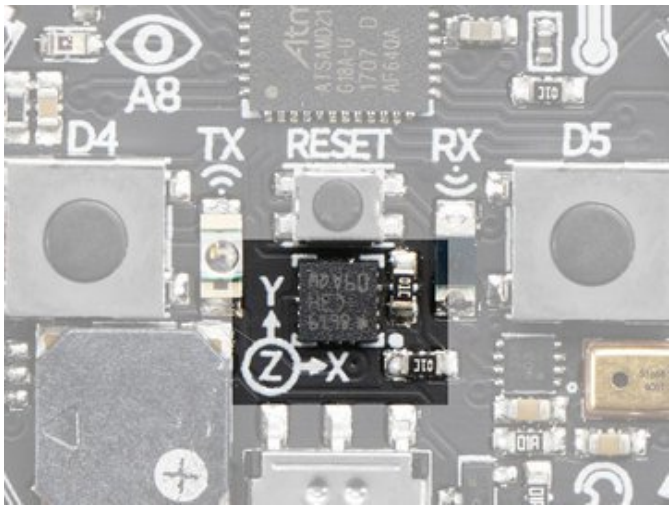
Temperature Sensor

There is an NTC thermistor (Murata NCP15XH103F03RC) that we use for temperature sensing. While it isn't an all-in-one temperature sensor, with linear output, it's easy to calculate the temperature based on the analog voltage on analog pin **#A9**. There's a 10K resistor connected to it as a pull down.



Microphone Audio Sensor

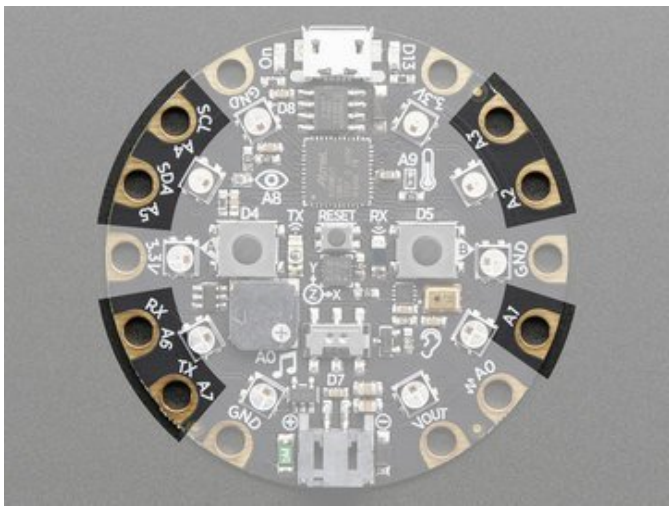
A MEMS microphone can be used to detect audio levels and even perform basic FFT functions. Instead of an analog microphone, that requires an external op-amp and level management, we've decided to go with a PDM microphone. This is a digital mic, and is a lot smaller and less expensive! You will have to use the MakeCode/CircuitPython/Arduino support libraries to read the audio volume, you cannot read it like an analog voltage.



Motion Sensor

We can sense motion with an accelerometer. This sensor detects *acceleration* which means it can be used to detect when its being moved around, as well as gravitational pull in order to detect orientation.

The LIS3DH 3-axis XYZ accelerometer is in the dead center of the board and you can use it to detect tilt, gravity, motion, as well as 'tap' and 'double tap' strikes on the board. The LIS3DH is connected to an internal I2C pinset (not the same as the ones on the pads) and has an optional interrupt output on digital pin **#36**

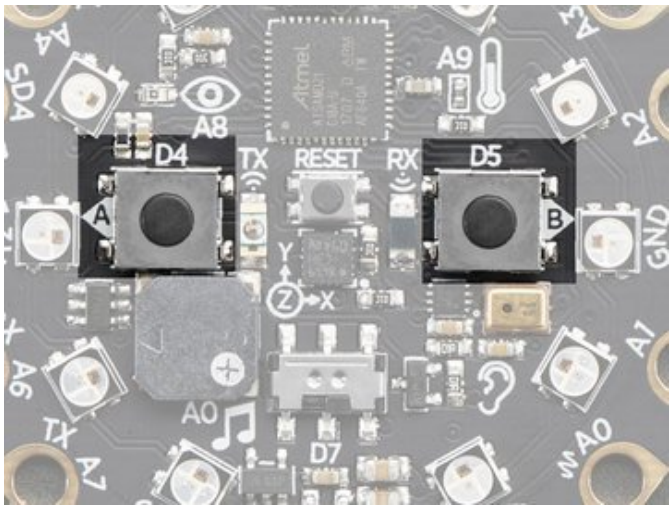


Capacitive Touch

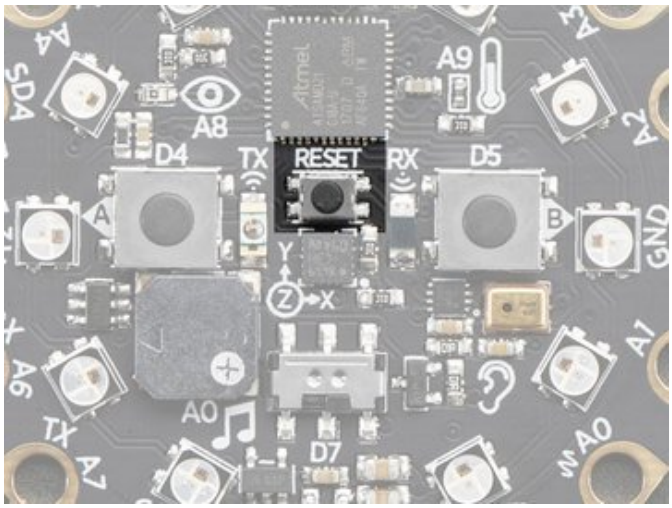
The chip in the CPX has a built in ability to perform capacitive touch readings. This is a great way to sense human touch without additional components. Even animals will work if its directly touching their skin!

On the Express you get **seven** capacitive touch pads. All pads but **A0** can do it. (**A0** is used for the audio output pad so cannot do captouch). Since we use the chip's built-in hardware support, you will use the provided MakeCode/Arduino/CircuitPython library code to perform the readings

Switches & Buttons



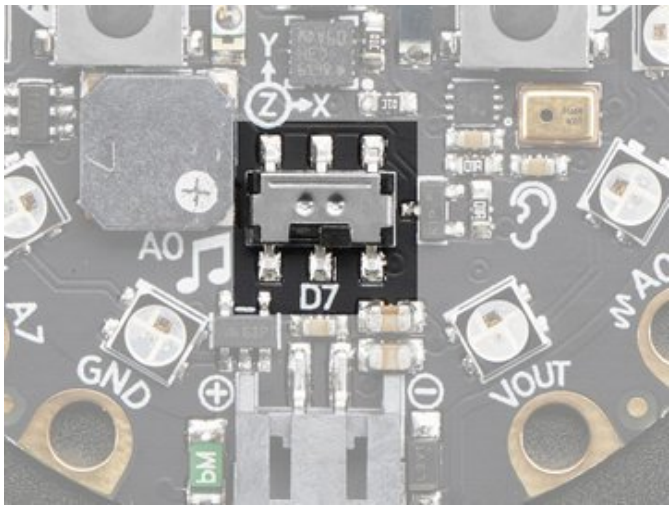
There are two large **A** and **B** buttons, connected to digital **#4** (Left) and **#5** (Right) each. These are unconnected when not pressed, and connected to 3.3V when pressed, so they read HIGH. Set the pins **#4** and **#5** to use an internal pull-down resistor when reading these pins so they will read LOW when not pressed.



This small button in the center of the board is for **Resetting** the board. You can use this button to restart or reset the CPX.

If using Arduino or CircuitPython, press this button once to reset, double-click to enter the bootloader manually.

If using MakeCode, press this button *twice* to reset, once to enter the bootloader manually.



There is a single slide switch near the center bottom of the Circuit Playground Express. It is connected to digital **#7**. The switch is unconnected when slid to the left and connected to ground when slid to the right. Set pin #7 to use an internal pull-up resistor so that the switch will read HIGH when slid to the left and LOW when slid to the right.

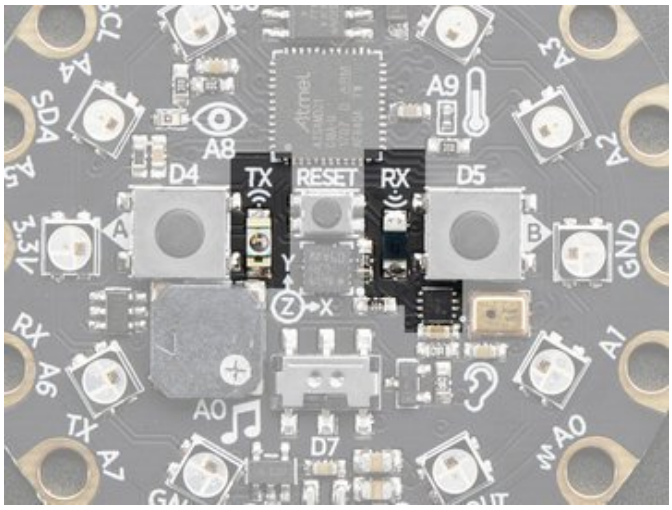
This is not an on-off switch, but you can use code to have this switch control how you want your project to behave

Note that you need to use an internal *pull-up* for the slide switch, but an internal *pull-down* for the pushbuttons.

Infrared Receive/Transmit & Proximity

New in Circuit Playground Express, we've added IR receive and transmit. This means you can communicate with TVs & other household devices to control them. You can also send commands to the CPX with a remote control. And, finally, you can send simple messages **between** multiple Circuit Playgrounds!

By bouncing IR light off of items and reading the reflected light, you can also do very simple proximity sensing!

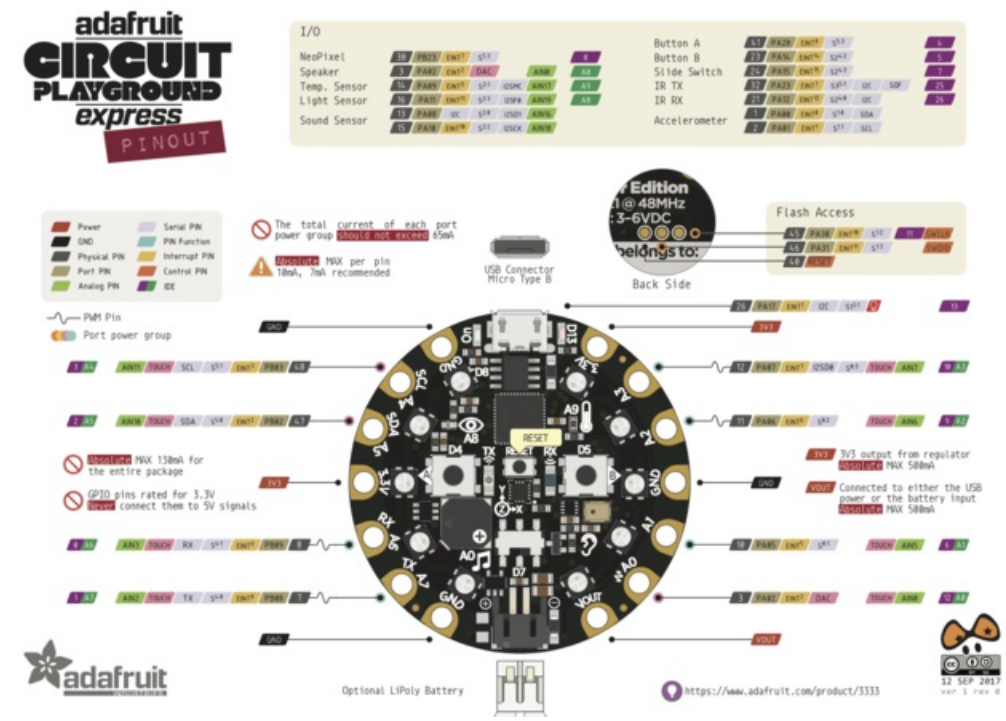


There's two elements for infrared. The **Transmitter** is on the left and is a big clear LED, this is connected via a transistor to pin **#29**

On the right is the darker **receiver** LED, which also has a decoder chip. This chip will receive the 38KHz signals and demodulate them for you. The demodulated output is available on pin **#39**

There's also a *secret* capability, you can read the 'raw' analog value from the receiver LED diode to do basic IR proximity sensing. The direct analog value is available from pin **A10**

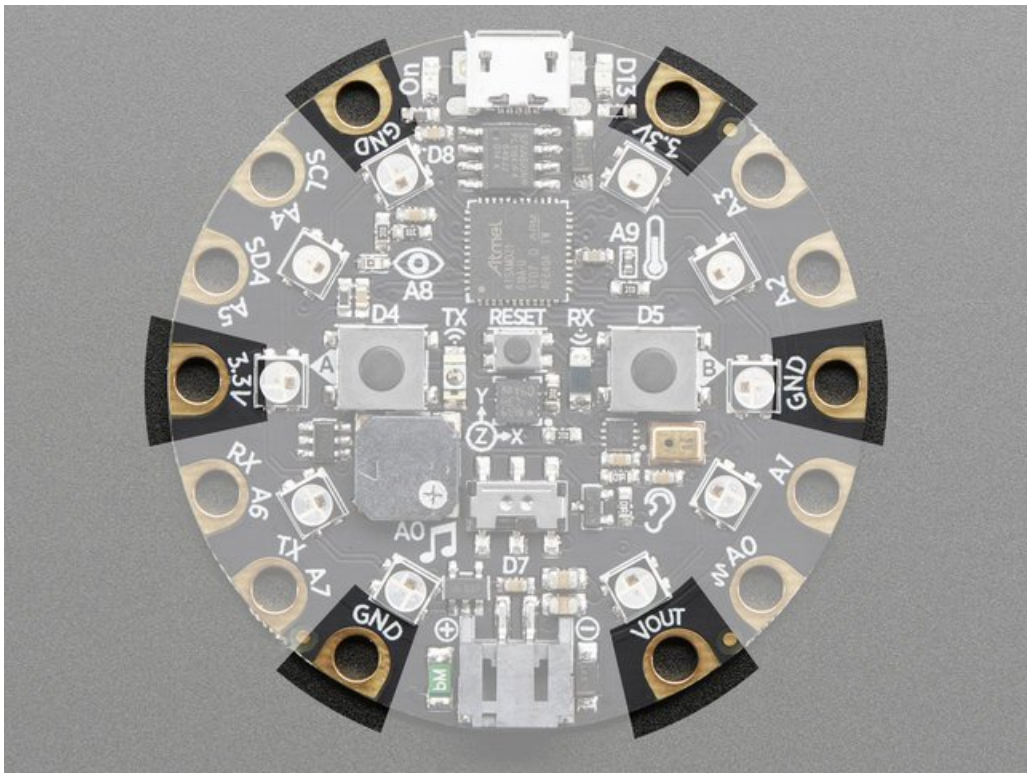
Pinouts



Despite having only 14 pads with 8 general purpose I/O pins available, there are a *lot* of possibilities with Circuit Playground Express. We went over all the internals in the last page. On this page we'll go through each pin/pad to explain what you can do with it.

Other than A0, no external I/O pads are shared with internal sensors/devices, so you do not need to worry about 'conflicting' pins or interactions!

Power Pads

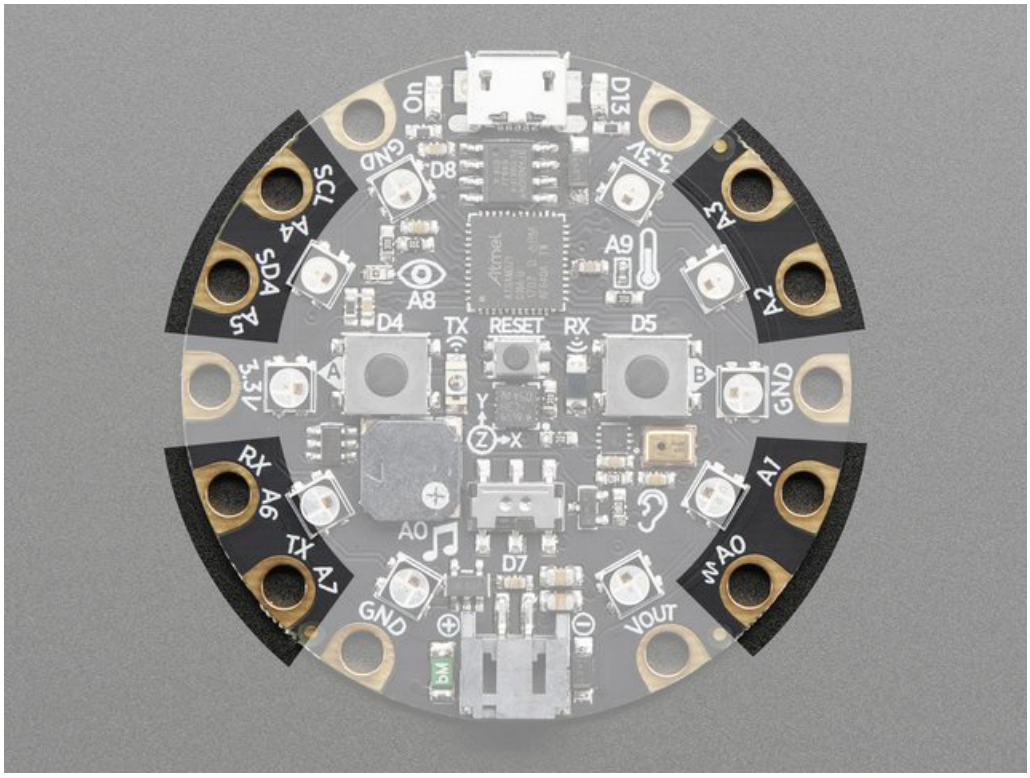


There are 6 power pads available, equally spaced around the perimeter.

- **GND** - there are 3 x **Ground** pads. They are all connected together, and are all the signal/power ground connections
- **3.3V** - there are two **3.3 Volt output** pads. They are connected to the output of the onboard regulator. The regulator can provide about 500mA max, but that includes all the built in parts too! So you should roughly budget about 300mA available for your usage (450mA if you are not using the onboard NeoPixels)
- **Vout** - there is one **Voltage Output** pad. This is a special power pad,it will be connected to *either* the USB power or the battery input, whichever has the higher voltage. This output does not connect to the regulator so you can draw as much current as your USB port / Battery can provide. There is a resettable fuse on this pin, so you can draw about 500mA continuous, and 1 Amp peak before it will trip. If the fuse trips, just wait a minute and it will automatically reset

Input/Output Pads

Next we will cover the 8 GPIO (General Purpose Input Ouput) pins! For reference you may want to also check out the datasheet-reference in the downloads section for the core ATSAMD21G18 pin. We picked pins that have *a lot* of capabilities.



Common to all pads

All the GPIO pads can be used as digital inputs, digital outputs, for LEDs, buttons and switches. In addition, all can be used as analog inputs (12-bit ADC). All but A0 can be used for hardware capacitive touch. All pads can also be used as hardware interrupt inputs.

Each pad can provide up to ~20mA of current. Don't connect a motor or other high-power component directly to the pins! Instead, use a transistor to power the DC motor on/off

All of the GPIO pads are 3.3V output level, and should not be used with 5V inputs. In general, most 5V devices are OK with 3.3V output though.

Other than A0, which is shared with the speaker, all of the pads are completely 'free' pins, they are not used by the USB connection, LEDs, sensors, etc so you never have to worry about interfering with them when programming.

Each Pin!

Let's start with **A0** which is in the bottom right corner, and work our way counter-clockwise

- **A0** (a.k.a **D12**) - This is a special pin that can do true analog output so it's great for playing audio clips. In can be digital I/O, or analog I/O, but if you do that it will interfere with the built-in speaker. This is the one pin that cannot be used for capacitive touch.
- **A1 / D6** - This pin can be digital I/O, or analog Input. This pin has PWM output and can be capacitive touch sensor
- **A2 / D9** - This pin can be digital I/O, or analog Input. This pin has PWM output and can be capacitive touch sensor
- **A3 / D10** - This pin can be digital I/O, or analog Input. This pin has PWM output and can be capacitive touch sensor
- **A4 / D3** - This pin can be digital I/O, or analog Input. This pin is also the **I2C SCL** pin, and can be capacitive touch

sensor
- **A5 / D2** - This pin can be digital I/O, or analog Input. This pin is also the **I2C SDA** pin, and can be capacitive touch sensor
- **A6 / D0** - This pin can be digital I/O, or analog Input. This pin has PWM output, **Serial Receive,** and can be capacitive touch sensor
- **A7 / D1** - This pin can be digital I/O, or analog Input. This pin has PWM output, **Serial Transmit,** and can be capacitive touch sensor

Internally Used Pins!

These are the names of the pins that are used for built in sensors and such!

- **D4** - Left Button A
- **D5** - Right Button B
- **D7** - Slide Switch
- **D8** - Built-in 10 NeoPixels
- **D13** - Red LED
- **D27** - Accelerometer interrupt
- **D29** - IR Transmitter
- **D39** - IR Receiver
- **A0** - Speaker analog output
- **A8** - Light Sensor
- **A9** - Temperature Sensor
- **A10** - IR Proximity Sensor

- **D28** - Internal I2C SDA (access with **Wire1**)
- **D29** - Internal I2C SCL (access with **Wire1**)
- **D30 (PIN_SPI_MISO)** - SPI FLASH MISO
- **D31 (PIN_SPI_SCK)** - SPI FLASH SCK
- **D32 (PIN_SPI_MOSI)** - SPI FLASH MOSI
- **D33** - SPI FLASH Chip Select

Windows Driver Installation

Mac and Linux do not require drivers, only Windows folks need to do this step

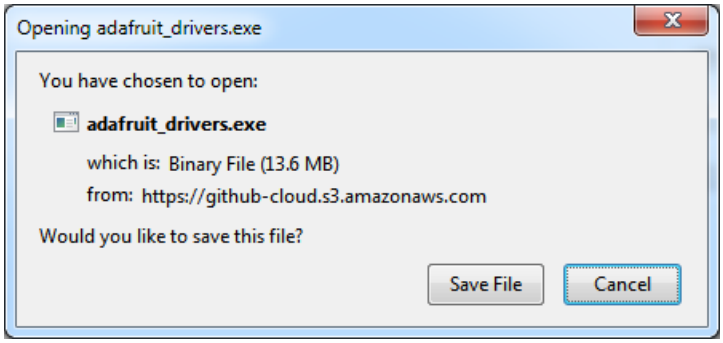
Before you plug in your board, you'll need to possibly install a driver!

Click below to download our Driver Installer.

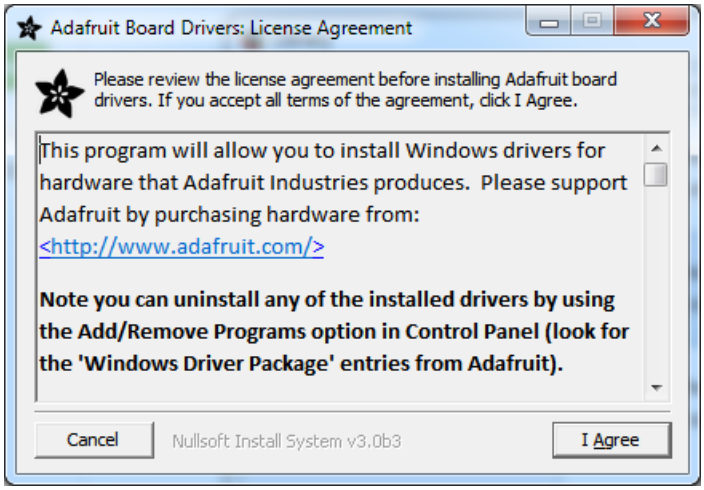
Download Latest Adafruit Windows Driver Installer

https://adafru.it/A0N

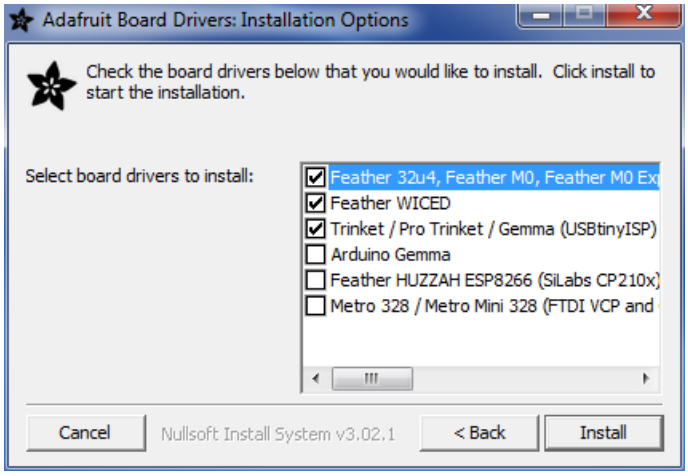
Download and run the installer.



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



Select which drivers you want to install, we suggest selecting all of them so you don't have to do this again!



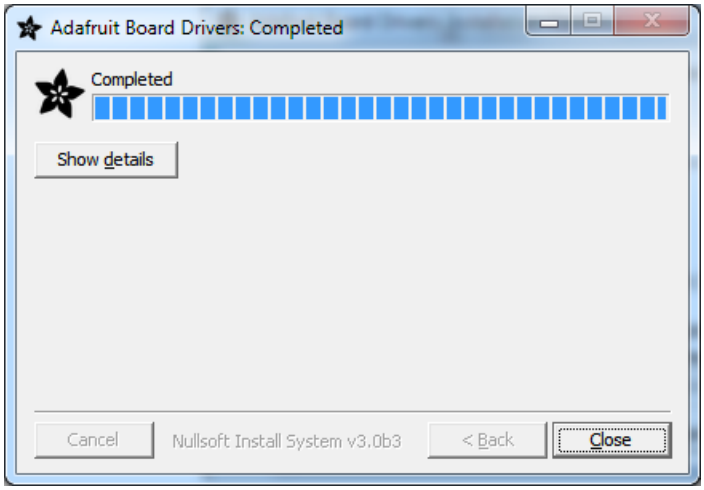
On Windows 7, by default, we install a single driver for most of Adafruit's boards, including the **Feather 32u4, the Feather M0, Feather M0, Express, Circuit Playground, Circuit Playground Express, Gemma M0, Trinket M0, Metro M0 Express**. On Windows 10 that driver is not necessary (it's built in to Windows) and it will not be listed.

The **Trinket / Pro Trinket / Gemma / USBtinyISP** drivers are also installed by default.

You can also, optionally, install the **Arduino Gemma** (different than the Adafruit Gemma!), **Huzzah and Metro 328** drivers.

Click **Install** to do the installin'.

Note that on Windows 10, support for many boards is built in. If you end up not checking any boxes, you don't need to run the installer at all!



Manual Driver Installation

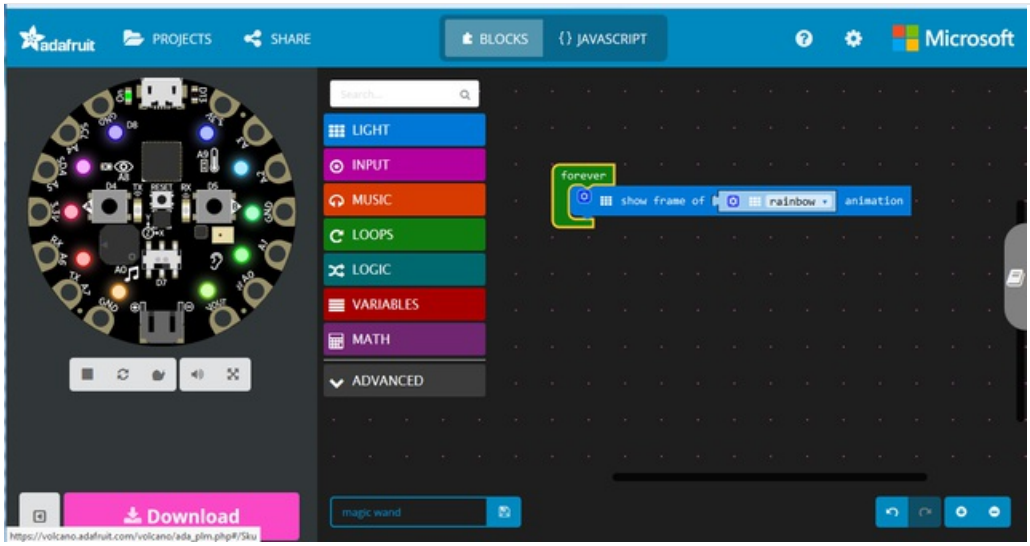
If windows needs the driver files (inf/cat) for some reason you can get all the drivers in a zip by clicking below:

Adafruit Windows Drivers source (v2.0.0.0)

https://adafru.it/zel

And point windows to the **Drivers** folder when it asks for the driver location

MakeCode



One of the beautiful things about Circuit Playground Express is that you can make **three ways**:

1. MakeCode
2. CircuitPython
3. Arduino IDE (or direct ARM GCC programming)

If this is your very first time programming or coding, we suggest starting with MakeCode - it is the super fast to get started not just with Circuit Playground Express, but coding in general! (Once you've got the hang of MakeCode you can then take a look at CircuitPython or Arduino.)

MakeCode does not require any software installation, it works on any computer with a web-browser. You can use it with Mac, Windows, Linux, Chromebooks.

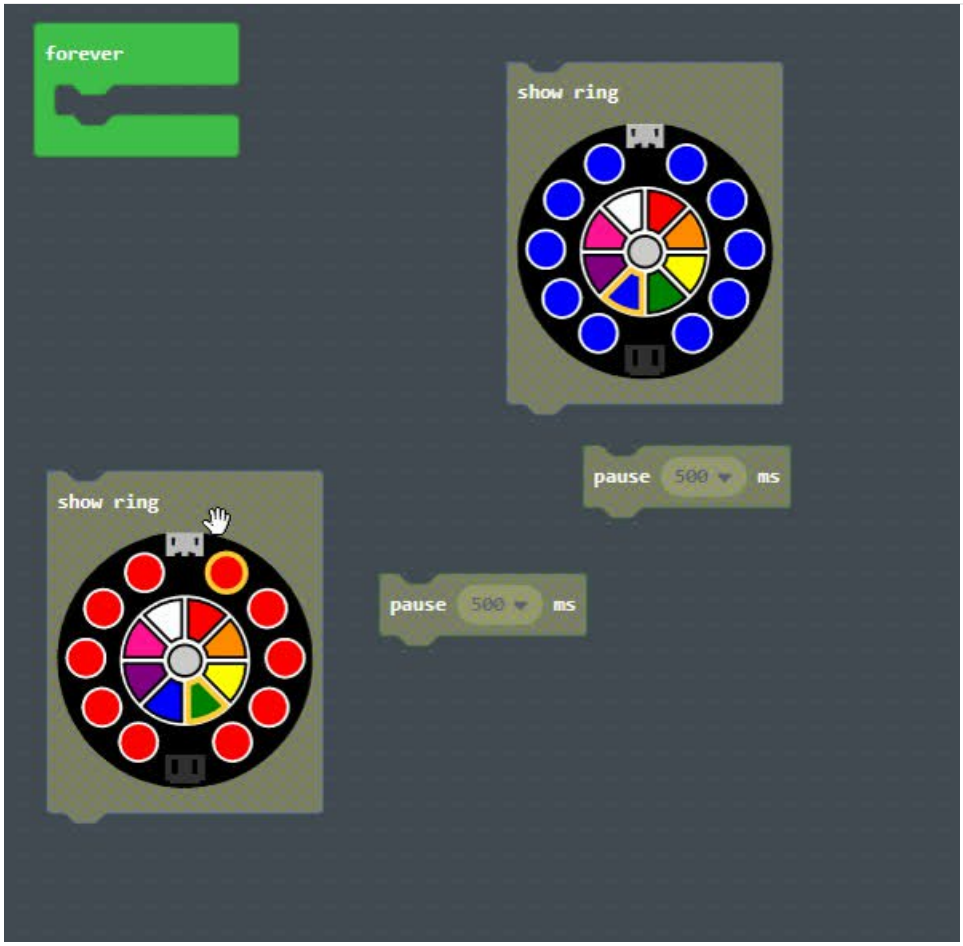
MakeCode uses drag-and-drop blocks just like Scratch, there is no syntax or semicolons. The blocks snap together to create larger and more complex projects

MakeCode lets you get started in 5 minutes or less, there are built in guides and projects, but its also just fun to play around.

MakeCode has surprisingly advanced runtime, despite looking very simple it can do very complex tasks that normally would require advanced programming. You can get creative very very quickly.

Read on to learn how to use MakeCode and build your first project!

What is MakeCode?



Microsoft MakeCode for Adafruit is a web-based code editor for physical computing. It provides a block editor, similar to Scratch or Code.org, and also a JavaScript editor for more advanced users. **Try it now at** https://makecode.adafruit.com/ **!**

Some of the key features of MakeCode are:

- **web based editor:** nothing to install
- **cross platform:** works in most modern browsers from tiny phone to giant touch screens
- **compilation in the browser:** the compiler runs in your browser, it's fast and works offline
- **blocks + JavaScript:** drag and drop blocks or type JavaScript, MakeCode let's you go back and forth between the two.
- **works offline:** once you've loaded the editor, it stays cached in your browser.
- **event based runtime:** easily respond to button clicks, shake gestures and more

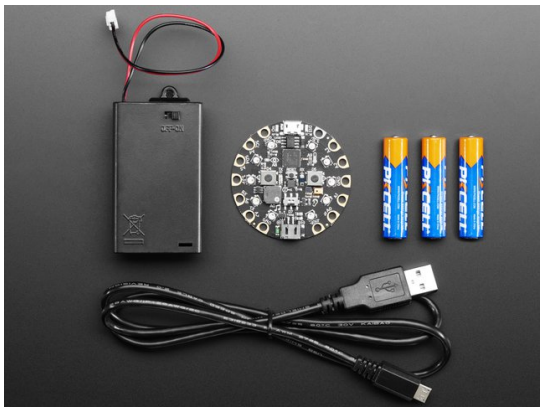
MakeCode currently supports the **Adafruit Circuit Playground Express.**

Your browser does not support the video tag.

Circuit Playground Express
PRODUCT ID: 3333

https://adafru.it/wpF

$24.95
IN STOCK



Circuit Playground Express - Base Kit
PRODUCT ID: 3517

https://adafru.it/wqb

$29.95
OUT OF STOCK

MakeCode works for the Express edition of the Circuit Playground, not the Classic.

Editing Blocks

The block editor is the easiest way to get started with MakeCode. You can drag and drop blocks from the category list. Each time you make a change to the blocks, the simulator will automatically restart and run the code. You can test your program in the browser!

- try our getting started tutorial that helps you build a siren program
- try one of our projects to learn more about the features of the Circuit Playground.
- take a deep dive in the documentation

Blinky!

Let's show how MakeCode works by building a simple program that blinks the 10 awesome NeoPixels.

Creating a blink effect is done by setting all the ring LEDs to red, **pause** for a little, then turn them off, pause for a little, then repeat **forever**.

Let's gather the blocks we need to convert the description above into Blocks that the Circuit Playground can understand and run:

- **forever** runs blocks in a loop with a 20ms pause in between (it is similar to Arduino**loop**).
- **show ring** sets the color on the 10 neopixels at once
- **pause** blocks the current thread for 100ms. If other events or forever loops are running, they have the opporunity to run at this time.

Do you want to select or change colors? The **show ring** block has a built-in color picker. Select the color from the color wheel to select a color, then click one of the ten Neopixel rings to modify its color.

How do I disable a Neopixel? The grey dot in the middle of the color wheel indicates that the pixel is off. Select the grey from the color wheel and then click the Neopixel ring.

You can see the Blinky block program in action in the MakeCode editor below. You can also see how the blocks are "slotted together". Clicking the question marks on the box will pop-up a comment to explain what the block does.

In the next section, we'll load the Blinky code onto the Circuit Playground Express!

Downloading and Flashing

Getting your code into your device is very easy with MakeCode. You do not need to install any software on your machine and the process takes two steps:

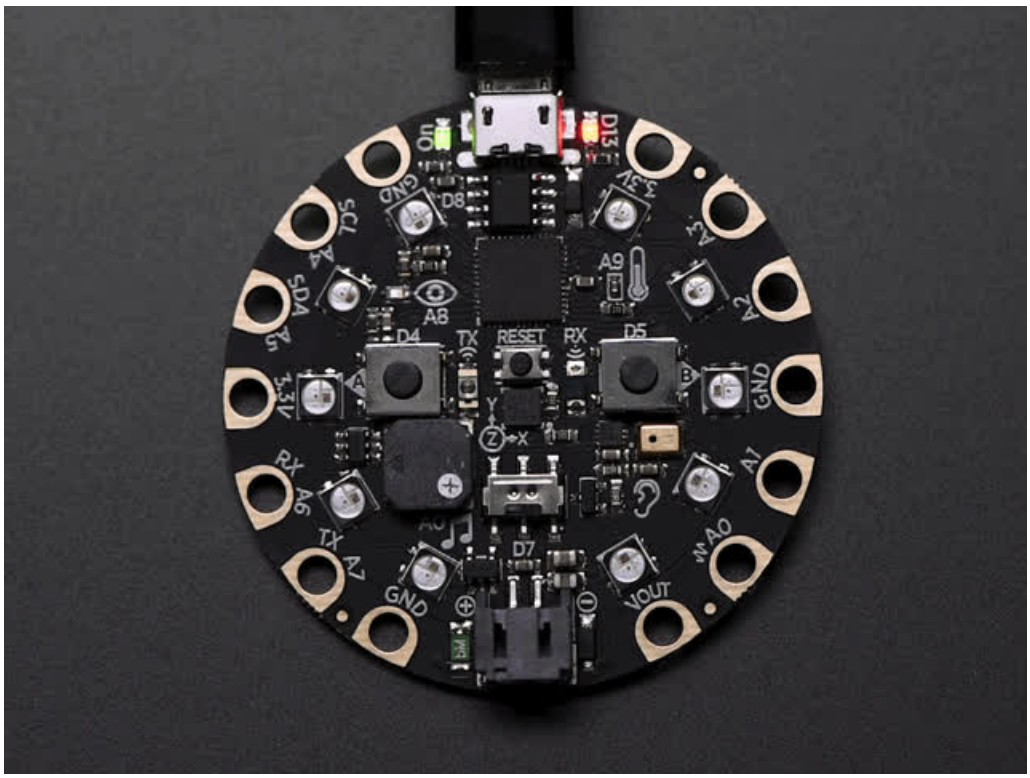
- Step 1: Connect your board via USB and enter **bootloader mode**
- Step 2: **Compile and Download** the .uf2 file into your board drive

We are going to go through these two steps in detail.

Step 1: Bootloader mode

Connect your board to your computer via a USB cable. Press the reset button once to put the board in bootloader mode.

> If it is your first time running MakeCode or if you have previously installed Arduino or CircuitPython, you may need to double press the reset button to get your board into bootloader mode.



When the Circuit Playground Express is in Bootloader mode, all the LEDs will turn **red briefly, then green. Verify your status LED is also pulsing red.** Your computer should show a new removable drive called "CPLAYBOOT"

If the LEDs are all red: Either the computer is still installing drivers (Please wait a minute, Windows takes some time to install updates.) or you have a bad USB connection. If you keep getting red LEDs - try a new USB cable (you may want to ensure your USB cable is not charge only, it needs to transfer data) or a different USB port.

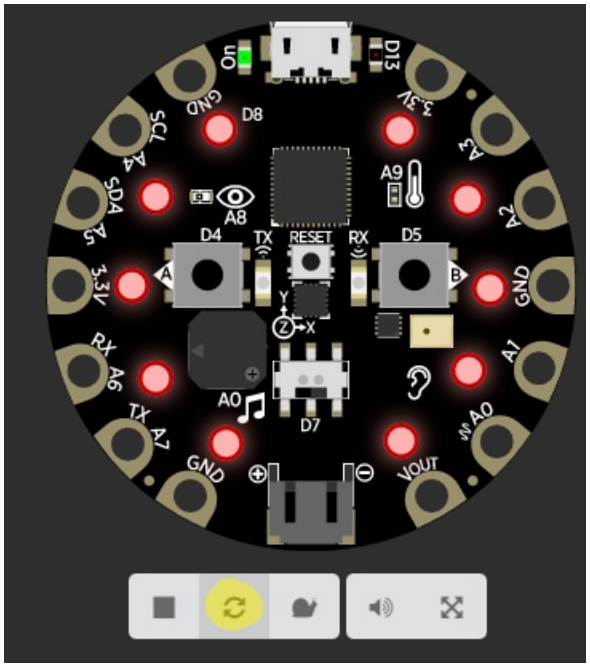
Once your LEDs are all green, you should see a CPLAYBOOT drive appear in your drive list in your file explorer.

We are now ready to compile our blinky code and download it to our board!

Step 2: Compile and Download

Let's first verify that our code compiles properly in MakeCode.

MakeCode has a built-in simulator that re-loads and re-runs code when restarted. This is an easy way to both ensure that our code compiles and simulate it before moving it onto the board. The refresh button re-loads the simulator with your latest version of block code.



If you receive a "we could not run this project" error, please check over your code for errors.

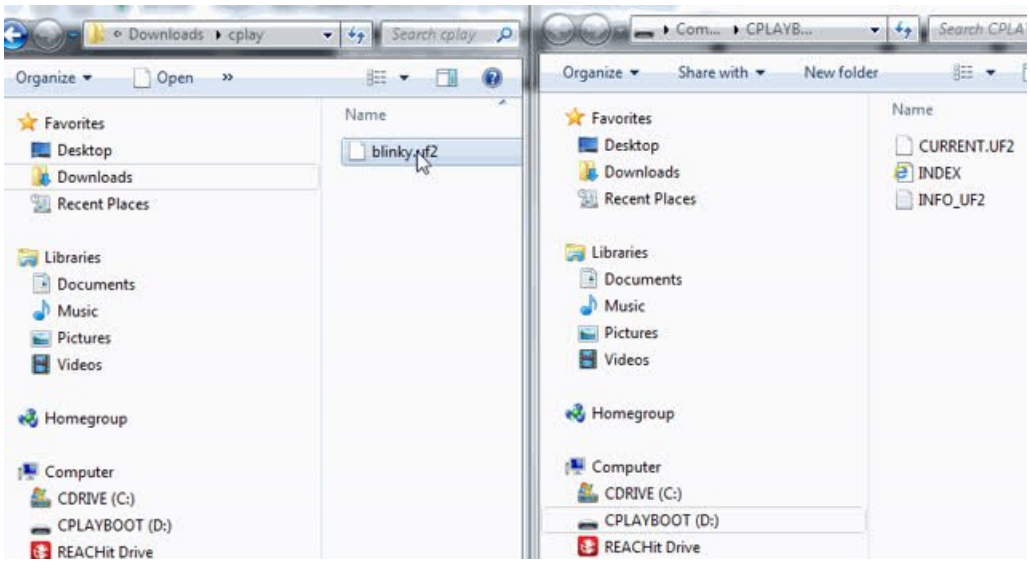
If your board is working in the simulator, it's time to download it to your actual board! Click the "Download" button. It will generate a .uf2 file and download it to your computer (UF2 is a file format designed to flash microcontrollers over USB).

General Steps to copy over your program (not specific to any Operating system)

1. Ensure your board is in bootloader mode.
2. Find the .uf2 file generated by MakeCode in your file explorer. Copy it to the "CPLAYBOOT" volume.
3. The status LED on the board will blink while the file is transferring. Once it's done transferring your file, the board will automatically reset and start running your code (just like in the simulator!)

Windows: Open Windows Explorer (Windows key + E key) and locate the "blinky.uf2" file you generated. It's probably in your **Downloads** folder!

You can copy/paste the file to your **CPLAYBOOT** volume or you can drag/drop it like in the GIF below.



macOS: Open Finder and locate the "blinky.uf2" file. You can copy/paste this file to the "cplayboot" volume or drag/drop it from the same finder window.

If you want to avoid the copying process: You can download your programs directly to the board. To do this: change the download location in Chrome, Firefox, Safari, or Opera to the main directory of your "CPLAYBOOT" volume.

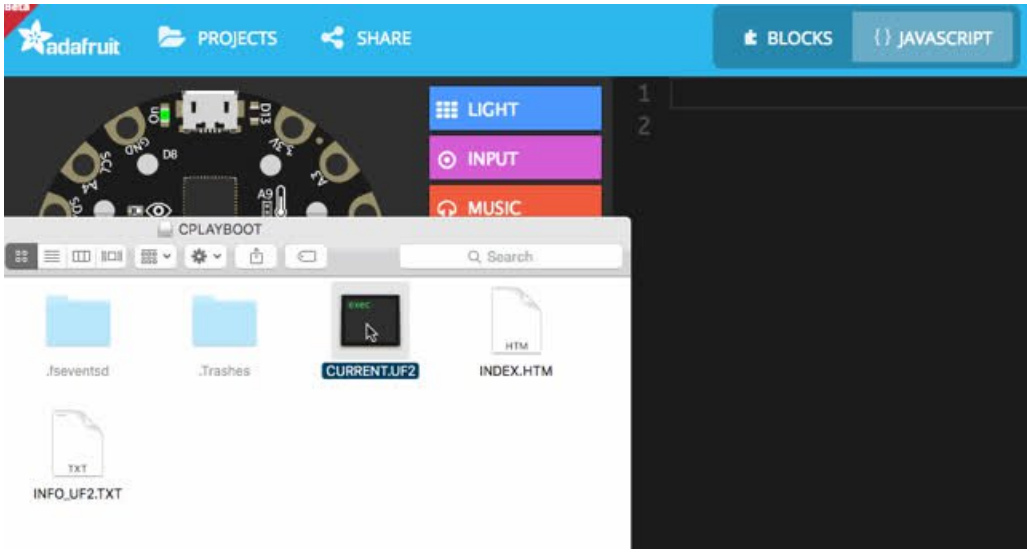
Running MakeCode that's Already Loaded

If you unplug your board and then plug it back in again to your computer, it may not run the program again automatically. If it instead shows all green NeoPixels, just press the reset button and your program will start.

Saving and Sharing

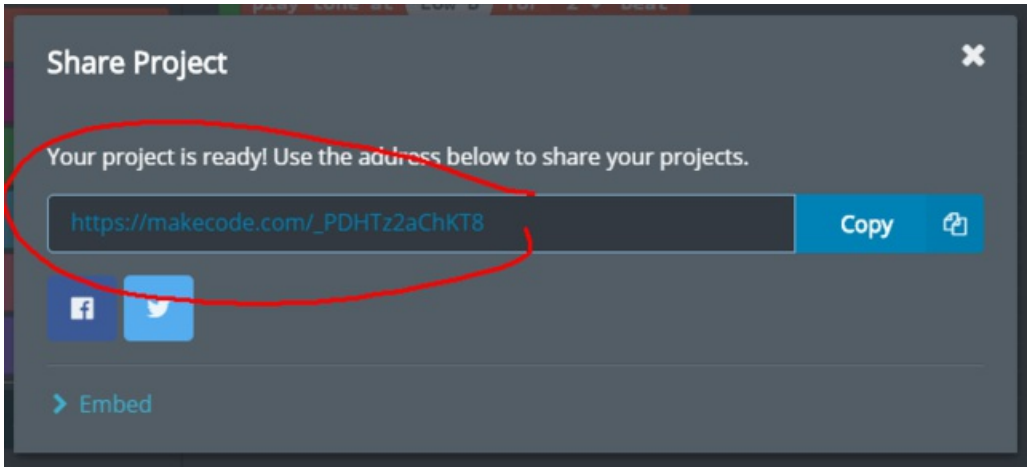
Want to change the Blinky program to display different colors, or make it play a sound? Editing the program on your Circuit Playground is super easy!

Extracting your code from the Circuit Playground



The .uf2 file **(CURRENT.UF2)** you created by clicking on the Compile button in MakeCode also contains the source code of your program!

You can open this file in MakeCode by dragging and dropping it into the browser to edit it.



Sharing

You can share your code by clicking on the **share** button. After confirmation, MakeCode will create a short unique URL for your code. Anyone with that URL will be able to reload the code.

These URLs can also be used to embed the editor your blog or web pages! Just copy paste the URL in your text editor and (if it supports oEmbed) it will automatically load it in your page.

Editing JavaScript

If you already have some experience coding or you feel ready to take the next step, MakeCode features a fully-feature JavaScript editor in the browser!

- read the JavaScript docs

Give it a try!

You can also switch between blocks and JavaScript by clicking the button on the top.

Apps

MakeCode also provides various apps to provide additional features not available in browsers.

Windows Store

The MakeCode for Adafruit Windows Store app.

- **Super fast HID-based flashing**: no more drag and drop.
- Reading serial message and surfacing them in the editor.

Node.JS

The (open source) GitHub repo contains instructions to run a local node.js web server with HID flashing and serial monitoring.

Other Good Stuff

This guide is meant as a starter block but it's worth mentioning briefly about other feature of MakeCode...

GitHub packages

Additional blocks or drivers can be packaged in github repo and loaded in the editor via the **Add Package** dialog. Packages can contain JavaScript, C++ and yes! ASM! https://makecode.adafruit.com/packages

We are Open Source on GitHub

Checkout https://makecode.com/about for more info about the various repos.

We have crowd-sourced translations

Whether you want to code in Klingon or your native non-English language, MakeCode supports crowd sourced translations at https://makecode.adafruit.com/translate.

What is CircuitPython?

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

CircuitPython is based on Python

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

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

Why would I use CircuitPython?

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

Other reasons to use CircuitPython include:

- **You want to get up and running quickly.** Create a file, edit your code, save the file, and it runs immediately.

There is no compiling, no downloading and no uploading needed.

- **You're new to programming.** CircuitPython is designed with education in mind. It's easy to start learning how to program and you get immediate feedback from the board.
- **Easily update your code.** Since your code lives on the disk drive, you can edit it whenever you like, you can also keep multiple files around for easy experimentation.
- **The serial console and REPL.** These allow for live feedback from your code and interactive programming.
- **File storage.** The internal storage for CircuitPython makes it great for data-logging, playing audio clips, and otherwise interacting with files.
- **Strong hardware support.** There are many libraries and drivers for sensors, breakout boards and other external components.
- **It's Python!** Python is the fastest-growing programming language. It's taught in schools and universities. CircuitPython is almost-completely compatible with Python. It simply adds hardware support.

This is just the beginning. CircuitPython continues to evolve, and is constantly being updated. We welcome and encourage feedback from the community, and we incorporate this into how we are developing CircuitPython. That's the core of the open source concept. This makes CircuitPython better for you and everyone who uses it!

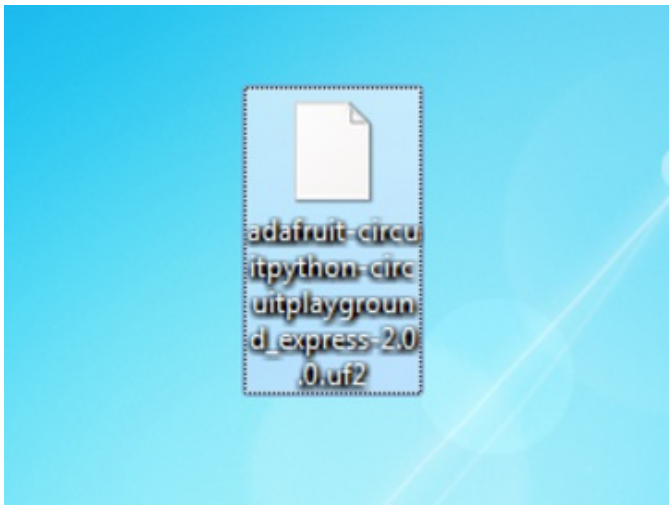
CircuitPython

Install or update CircuitPython!

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

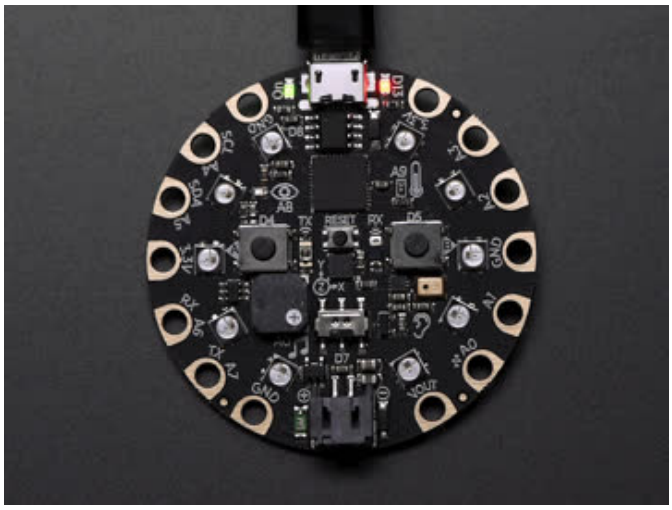
Download the latest CPX CircuitPython UF2

https://adafru.it/A0P



Click the link above to download the latest UF2 file

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

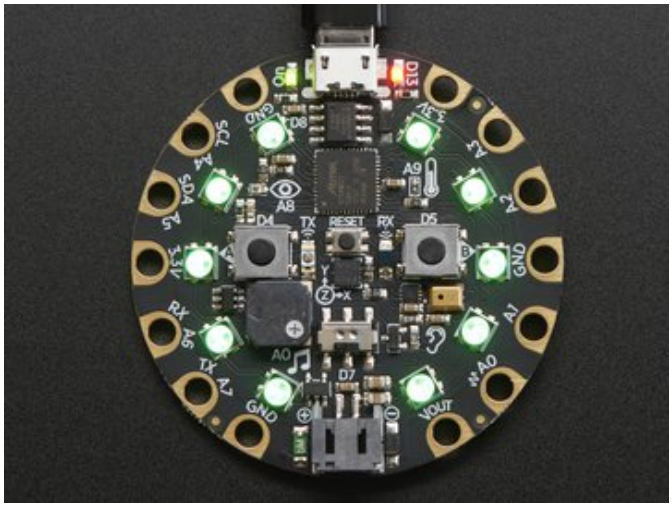


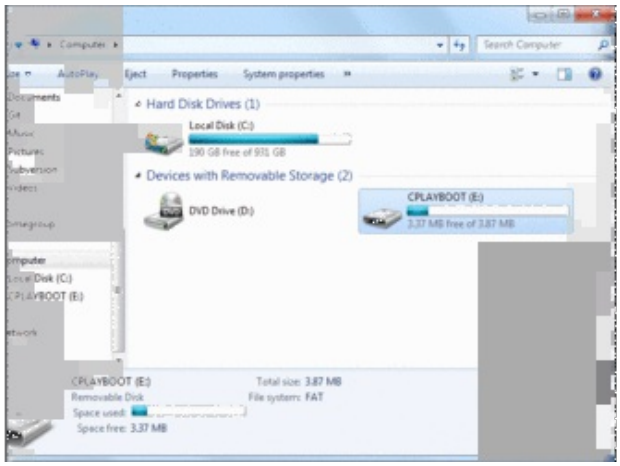
Plug your Circuit Playground Express into your computer using a known-good USB cable

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 small **Reset** button in the middle of the CPX, you will see all of the LEDs turn green. If they turn all red, check the USB cable, try another USB port, etc.

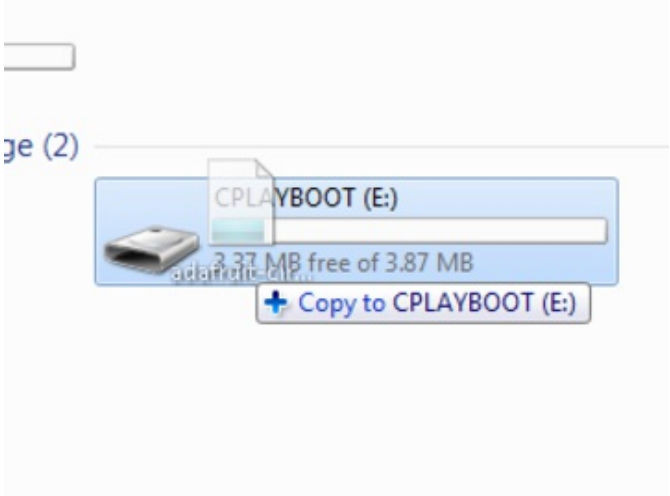
(If double-clicking doesn't do it, try a single-click!)

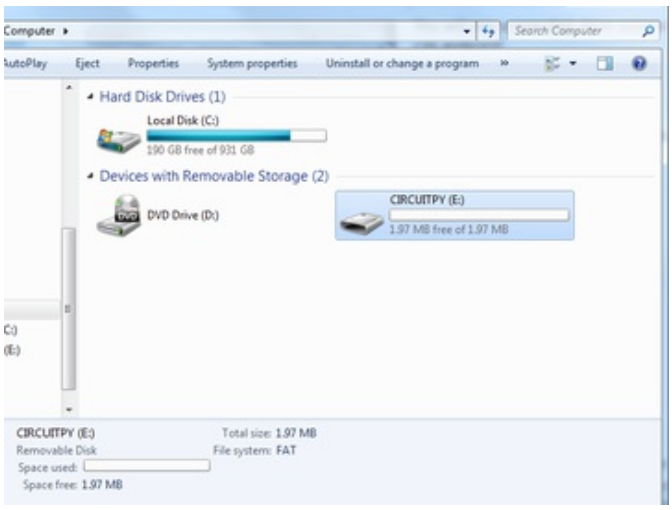




You will see a new disk drive appear called CPLAYBOOT

Drag the **adafruit-circuitpython-etc...uf2** file onto it





The **CPLAYBOOT** drive will disappear and a new disk drive will appear called **CIRCUITPY**

That's it! You're done :)

Further Information

For more detailed info on installing CircuitPython, check out Installing CircuitPython.

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!)

Installing Mu for Windows or Mac OS X

To install Mu for Windows, follow these steps:

Download the latest Mu for Windows

https://adafru.it/Amb

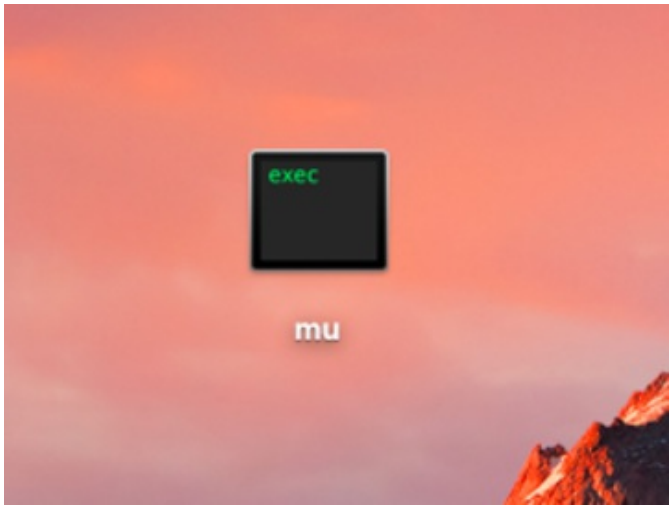
Download the latest Mu for Mac OS X

https://adafru.it/Amc

Click the link above to download the latest version of Mu. If you are using Windows, you must be running Windows 7 or higher. For Mac OS X you must be running 10.12 (Sierra) or higher (Mac users with lower versions can try the Linux instructions below, but YMMV)

Download and save the file to your desktop or wherever is handy.

Double-click the file to open Mu. You're ready to go!

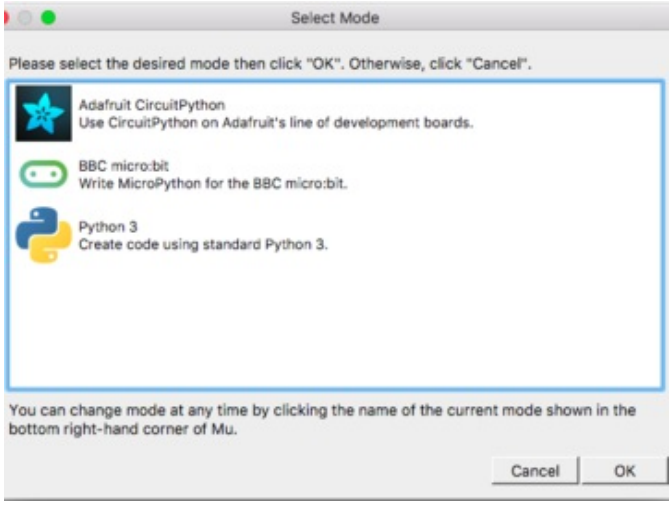


Installing Mu for Linux

Each Linux distro is a little different, so use this as a guideline! These instructions will also work for Mac OS X, but you'll want to use brew instead of apt-get

1. Mu require python version 3. If you haven't installed python yet, do so via your command line using something like `sudo apt-get install python3`
2. You'll also need pip3 (or pip if you only have python3 installed) - try running `pip3 --version`. If that didn't work, you ran `sudo apt-get install python3-pip`
3. Finally, run `pip3 install mu_editor`
4. You can now run `mu` directly from the command line

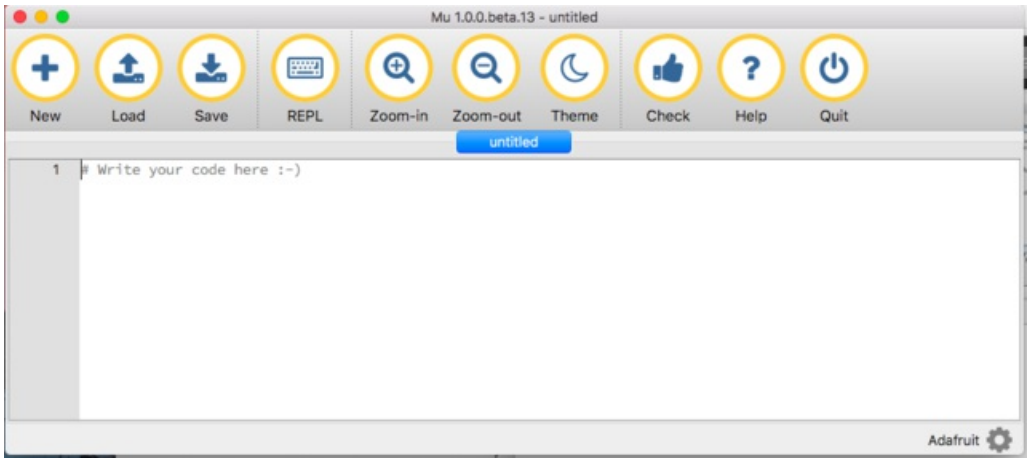
Using Mu



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

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

Now you're ready to code! Lets keep going....


Mu 1.0.0.beta.13 - untitled
New
Load
Save
REPL
Zoom-in
Zoom-out
Theme
Check
Help
Quit
untitled
1
Write your code here :-)
Adafruit

Creating and Editing Code

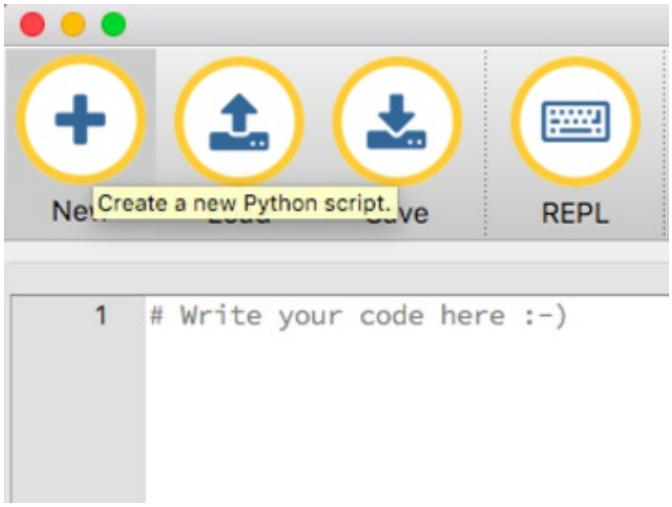
One of the best things about CircuitPython is how simple it is to get code up and running. In this section, we're going to cover how to create and edit your first CircuitPython program.

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. There are also excellent options available for download that are designed for editing code. Atom is a code editor that works on all three operating systems. There are many options for all operating systems.

Code editors have features that are specific to editing code, but any text editor will be fine.

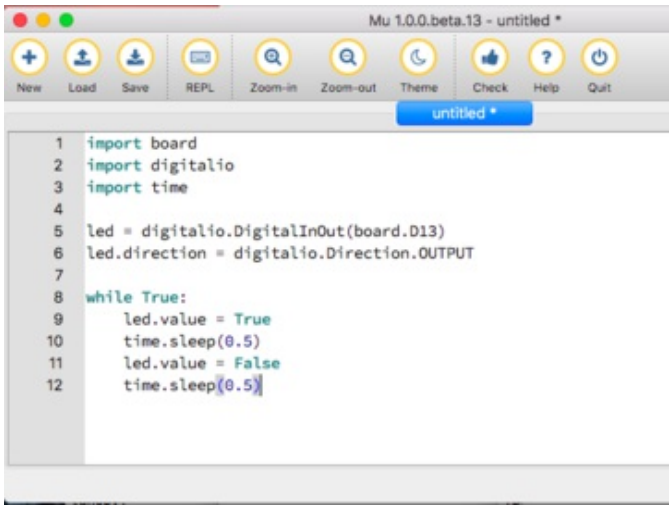
Creating Code



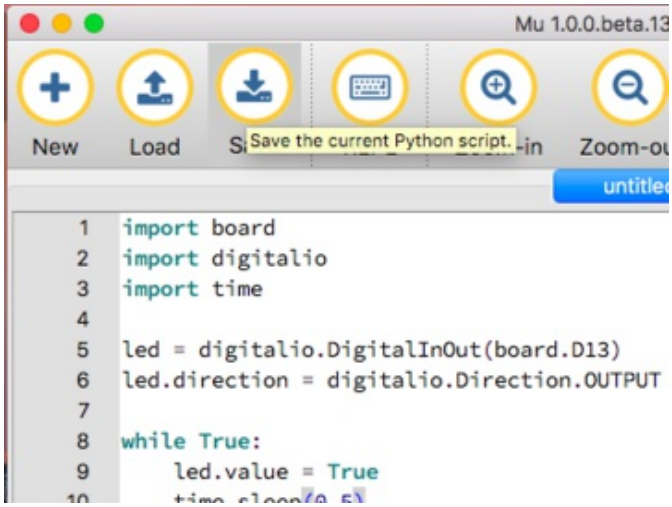
Open your editor, and create a new file. If you are using Mu, click the **New** button in the top left

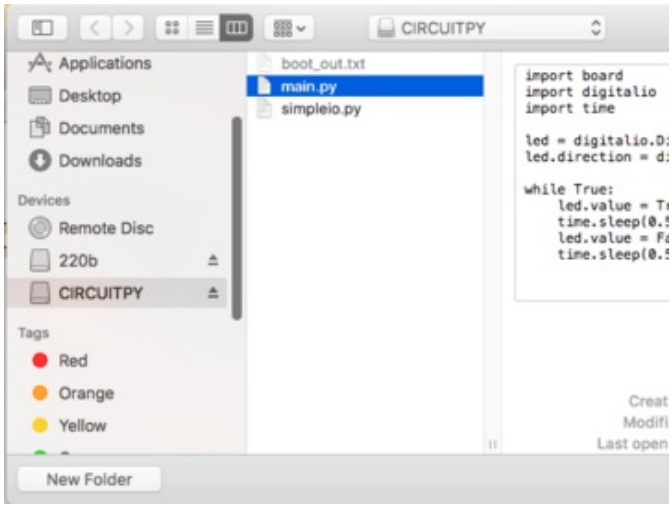
Copy and paste the following code into your editor:

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.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.



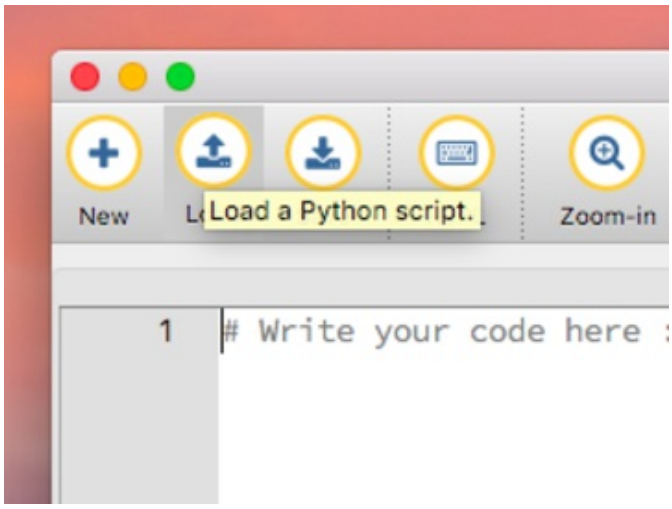


Save this file as **code.py** on your CIRCUITPY drive.

On each board you'll find a tiny red LED. It 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** is an editor that safely writes all changes (it's also our recommended editor!)
- **emacs** is also an editor that will fully write files on save
- **vim** / **vi** safely writes all changes
- **Sublime Text** safely writes all changes
- The **PyCharm IDE** is safe if "Safe Write" is turned on in Settings->System Settings->Synchronization (true by default).
- If you are using **Atom**, install this package so that it will always write out all changes to files on `CIRCUITPY`.

- **Visual Studio Code** appears to safely write all changes
- **gedit** on Linux appears to safely write all changes

We *don't* recommend these editors:

- **notepad** (the default windows editor) and **Notepad++** can be slow to write, so we recommend the editors above! If you are using notepad, be sure to eject the drive (see below)
- **IDLE** does not force-write out the file
- **Anything else** - we haven't tested other editors so please use a recommended one!

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.

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

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

Back to Editing Code...

Now! Let's try editing the program you added to your board. Open your **code.py** file into your editor. 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.D13)
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.D13)
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. These files are called **libraries**. Some of them are built into CircuitPython. Others 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 libraries: `board`, `digitalio`, and `time`. All three of these libraries are built into CircuitPython, so no separate 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.D13)
led.direction = digitalio.Direction.OUTPUT
```

Your board knows the red LED as `D13`. 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!

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.

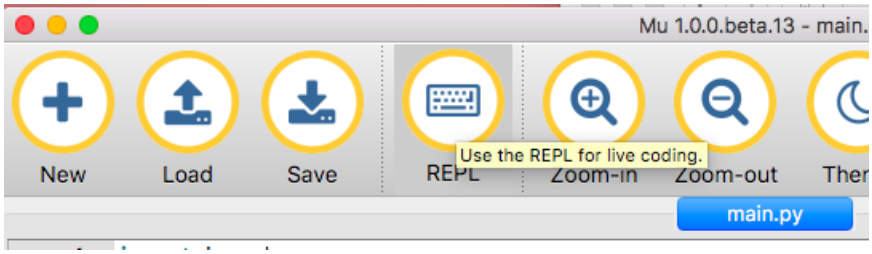
Are you using Mu?

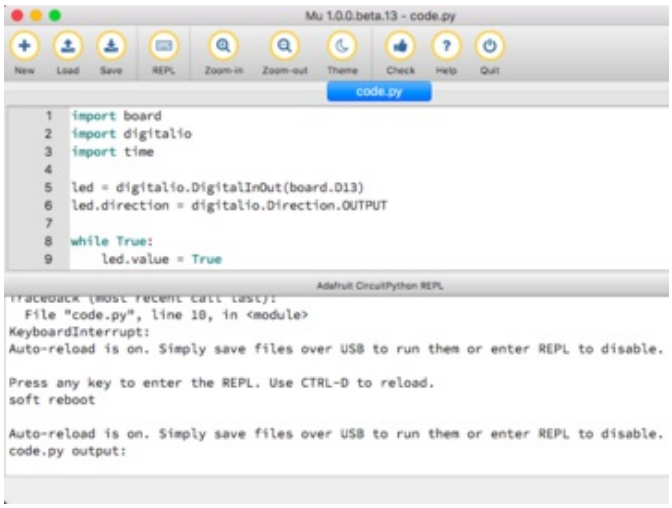
If so, good news! The serial console **is built into Mu** and will **autodetect your board** making using the REPL *really really easy*.

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.

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 **REPL** button in the menu and click it





The editor window will split in half.

The bottom half is your serial output/input. You can see text *from* the CircuitPython board as well as send text *to* the board.

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

Mac and Linux both have one built in, though other options are available for download, check this page for more details

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.

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

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
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!

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

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

```
5. screen
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.

```
2. screen
-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.

```
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.
```

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.

```
3. screen

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

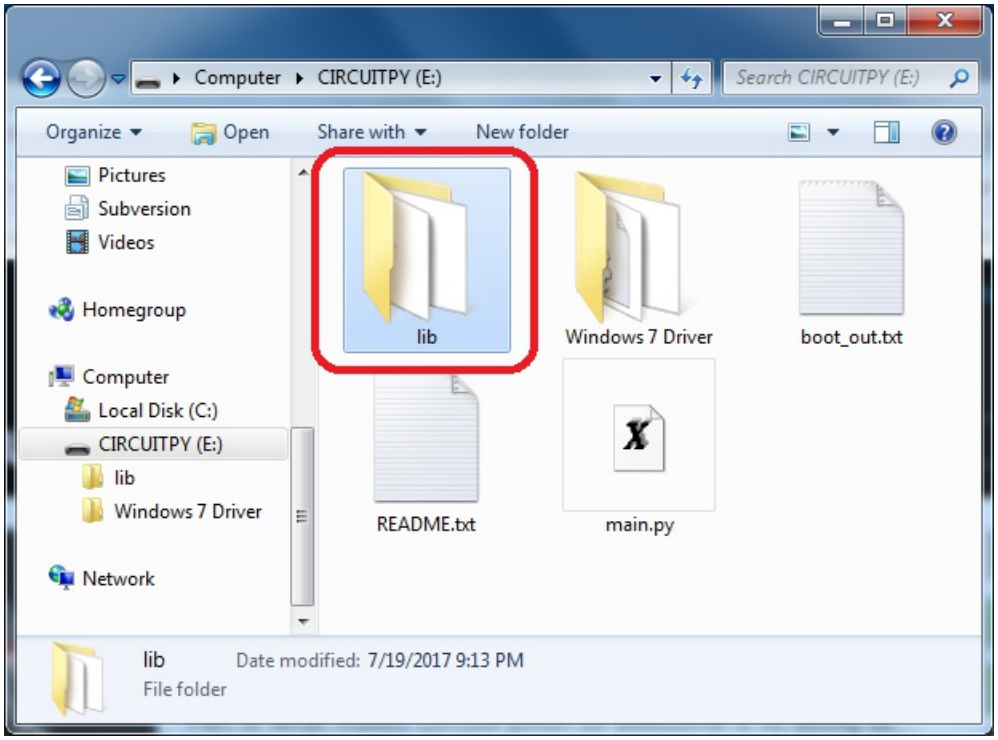
When you're ready to leave the REPL and return to the serial console, simply press **Ctrl + D**. This will reload your board and reenter the serial console. You will restart the program you had running before entering the REPL. In the console window, you'll see any output from the program you had running. 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

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, its in the base directory of the drive. If not, simply create the folder yourself.



CircuitPython libraries work in the same was as regular Python modules so the Python docs 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 2.x Bundle release by clicking this button:

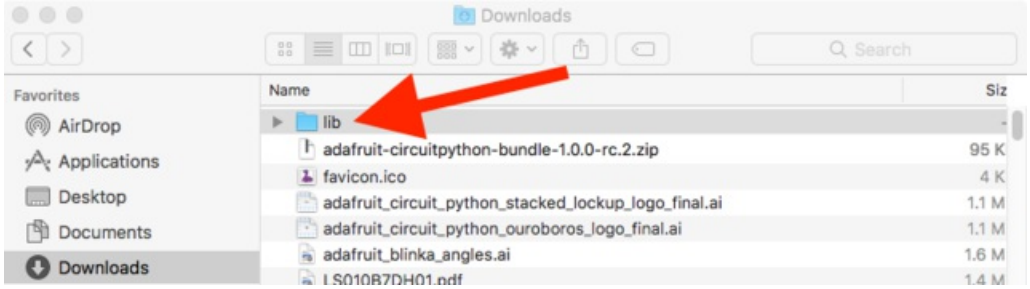
Click for the Latest Adafruit CircuitPython Library Bundle Release

https://adafru.it/AgR

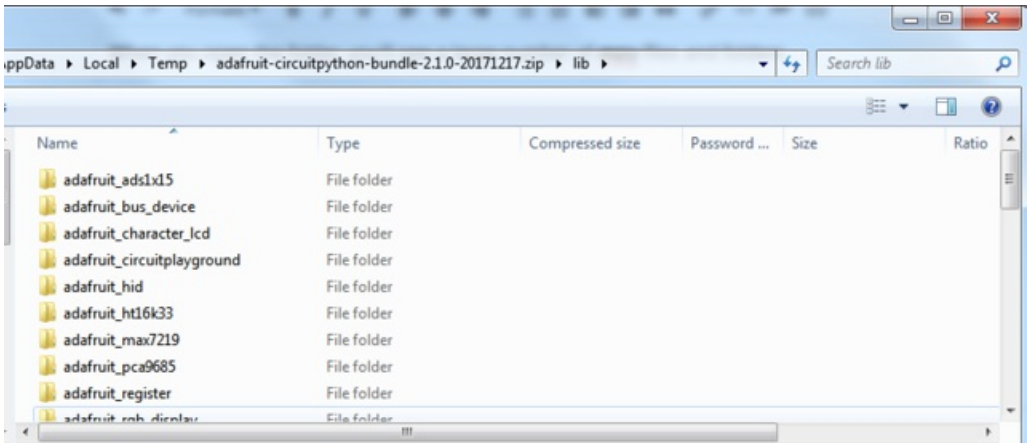
If you need another version, you can also visit the bundle release page which will let you select exactly what version you're looking for, as well as information about changes.

Either way, download the version that matches your CircuitPython run-time. For example, if you're running v2.2 download the v2 bundle. If you're running 3.0, download the v3 bundle. There's also a `py` bundle which contains the uncompressed python files, you probably *don't* want that!

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.



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



Express Boards

If you are using a Feather M0 Express, Metro M0 Express or Circuit Playground Express (or any other "Express" board) your CircuitPython board comes with at least 2 MB of Flash storage. This is *plenty* of space for all of our library files so we recommend you just install them all! (If you have a Gemma M0 or Trinket M0 or other non-Express board, skip down to the next section)

On Express boards, the `lib` directory can be copied directly to the CIRCUITPY drive.

Just drag the entire `lib` folder into the CIRCUITPY drive, and 'replace' any old files if your operating system prompts you

Non-Express Boards

If you are using Trinket M0 or Gemma M0, you will need to load the libraries individually, due to file space restrictions. If you are using a non-express board, or you would rather load libraries as you use them, you'll first 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.

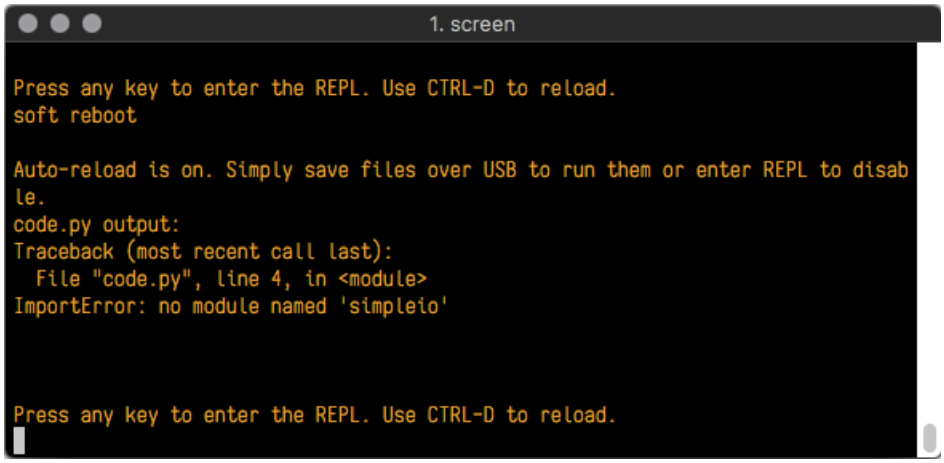
Example: `ImportError` Due to Missing Library

If you choose to load libraries as you need them, you may write up code that tries to use a library you haven't yet loaded. We're going to demonstrate what happens when you try to utilise a library that you don't have loaded on your board, and cover the steps required to resolve the issue. **This demonstration will only return an error if you do not have the required library loaded into the `lib` folder on your `CIRCUITPY` drive.**

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

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.D13)

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

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



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.

For these boards, your internal storage is from the chip itself. So, these boards don't have enough space for all of the libraries. If you try to copy over the entire `lib` folder you won't have enough space 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

Libraries 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, 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!

If you'd like to update the entire bundle at once, drag the lib folder to your CIRUCITPY drive. Different operating systems will have a different dialog pop up. You want to tell it to replace the current folder. Then you're updated and ready to go!

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.

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.

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 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 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.

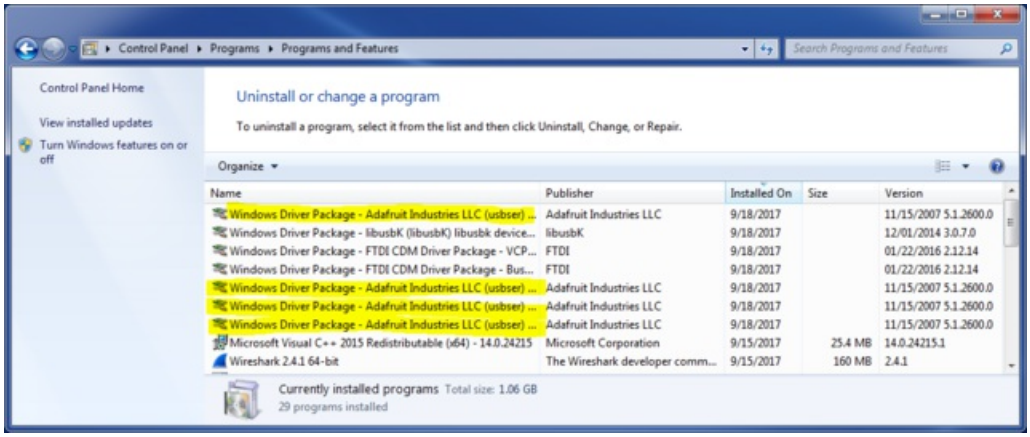
Windows 10

Did you install the Adafruit Windows Drivers package by mistake? 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

The latest version of the Adafruit Windows Drivers (version 2.0.0.0 or later) will fix the missing `boardnameBOOT` drive problem on Windows 7. 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 ...".

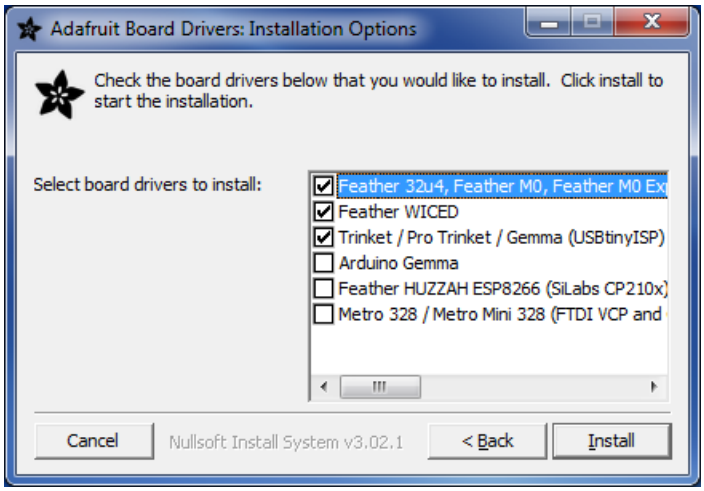


- Now install the new 2.0.0.0 (or higher) Adafruit Windows Drivers Package:

Install Latest Windows Drivers

https://adafru.it/A0N

- 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.



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

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

CircuitPython RGB Status Light

The Feather M0 Express, Metro M0 Express, Gemma M0, and Trinket M0 all have a single NeoPixel or DotStar RGB LED on the board that indicates the status of 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
- **YELLOW**: Circuit Python is in safe mode: it crashed and restarted
- **WHITE**: REPL is running
- **BLUE**: Circuit Python is starting up

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.

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 become corrupted.

This happens most often when the `CIRCUITPY` disk is not safely ejected before being reset by the button or being disconnected from USB. 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.

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

1. Download the correct erase file:

Circuit Playground Express

https://adafru.it/AdI

Feather M0 Express

https://adafru.it/AdJ

Metro M0 Express

https://adafru.it/AdK

2. Double-click the reset button on the board to bring up the `boardnameBOOT` drive.
3. Drag the erase `.uf2` file to the `boardnameBOOT` drive.
4. The onboard NeoPixel will turn blue, indicating the erase has started.
5. After approximately 15 seconds, the NeoPixel will start flashing green.
6. Double-click the reset button on the board to bring up the `boardnameBOOT` drive.
7. Drag the appropriate latest release of CircuitPython `.uf2` file to the `boardnameBOOT` drive.

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

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

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

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

1. Download the erase file:

Gemma M0, Trinket M0

https://adafru.it/AdL

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

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

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

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

Just follow these directions to reload CircuitPython using `bossac`, 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.

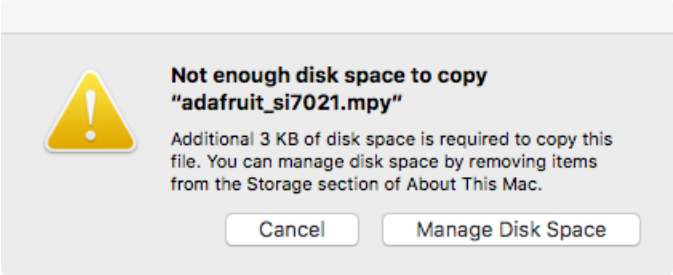
Delete something!

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.

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.

Mac OSX loves to add extra files.



Luckily you can disable some of the extra hidden files that Mac OSX 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 OSX:

Prevent & Remove Mac OSX Hidden Files

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

```
ls -l /Volumes
```

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

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

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

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

However there are still some cases where hidden files will be created by Mac OSX. In particular if you copy a file that was downloaded from the internet it will have special metadata that Mac OSX 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 Mac OSX Without Creating Hidden Files

Once you've disabled and removed hidden files with the above commands on Mac OSX 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
```

Or to copy a folder and all of its child files/folders use a command like:

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

Other Mac OSX 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:

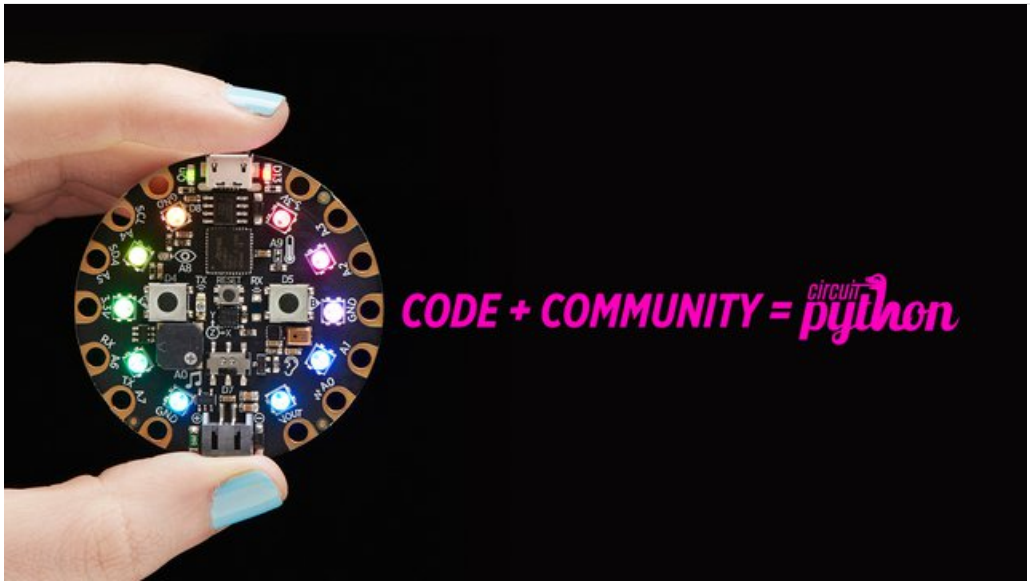
```
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem     Size   Used  Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1   59Ki   54Ki  5.5Ki    91%     128     0  100%   /Volumes/C
IRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./                  ._.Trashes*         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!

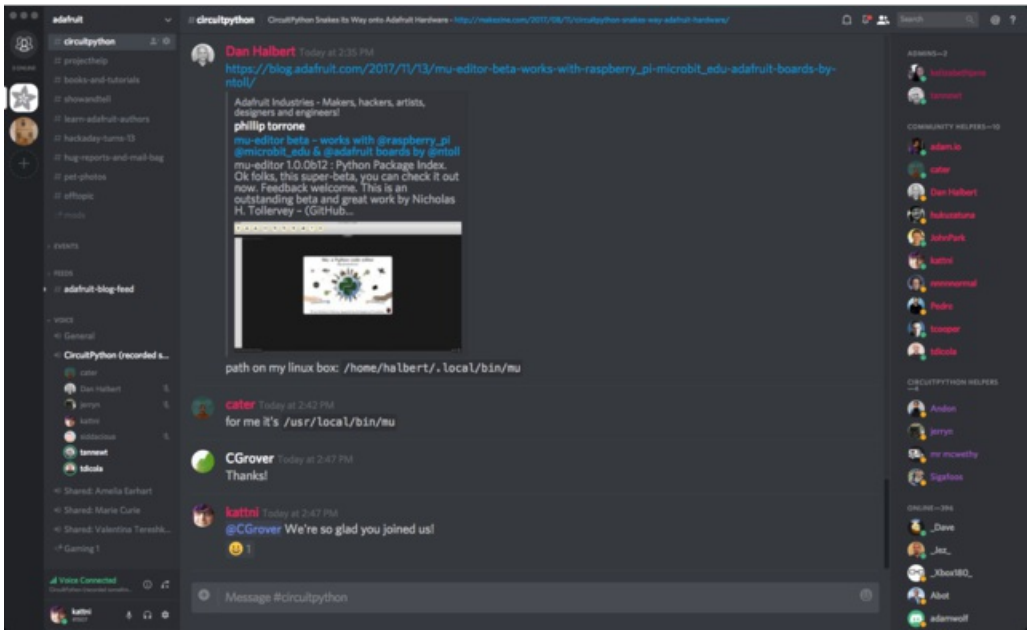
Welcome to the Community!



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



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 #projecthelp 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.

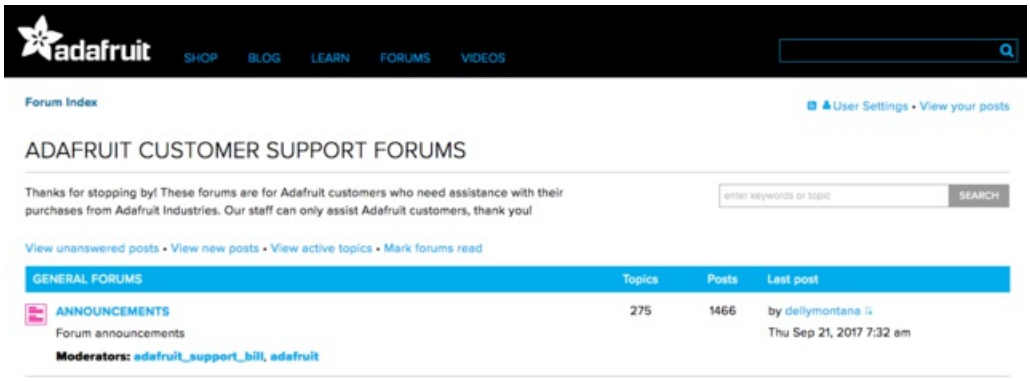
The CircuitPython channel is where to go with your CircuitPython questions. #circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. We'd love to hear what you have to say!

The 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!

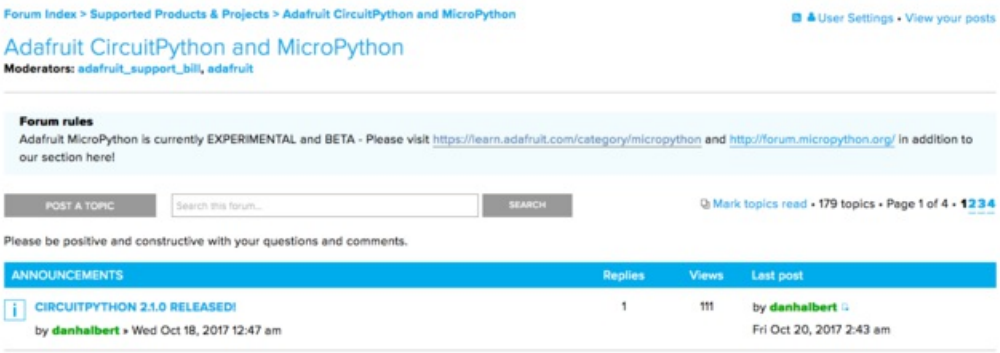
Adafruit Forums



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

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

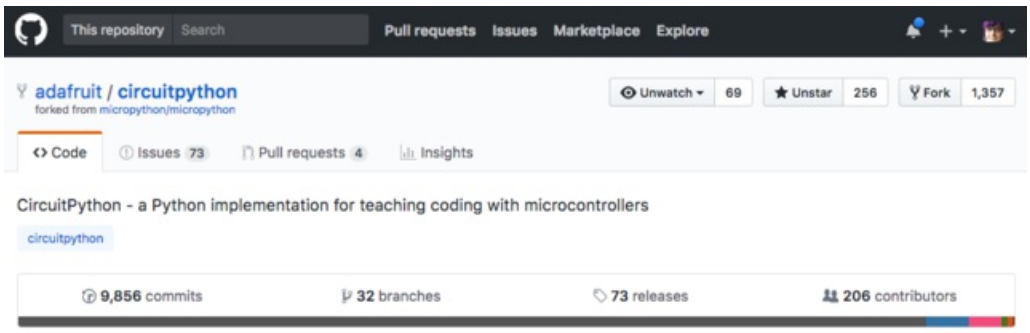
There are forum categories that cover all kinds of topics, including everything Adafruit. The Adafruit CircuitPython and MicroPython category under "Supported Products & Projects" is the best place to post your CircuitPython questions.



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

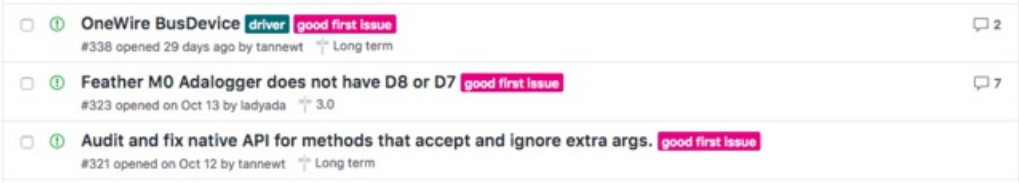
You might think you're just getting started, but you definitely know something that someone else doesn't. The great thing about the forums is that you can help others too! Everyone is welcome and encouraged to provide constructive feedback to any of the posted questions. This is an excellent way to contribute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

Adafruit Github



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

If you're new to GitHub or programming in general, there are great opportunities for you. Head over to adafruit/circuitpython on GitHub, click on "Issues", and you'll find a list that includes issues labeled "good first issue". 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.



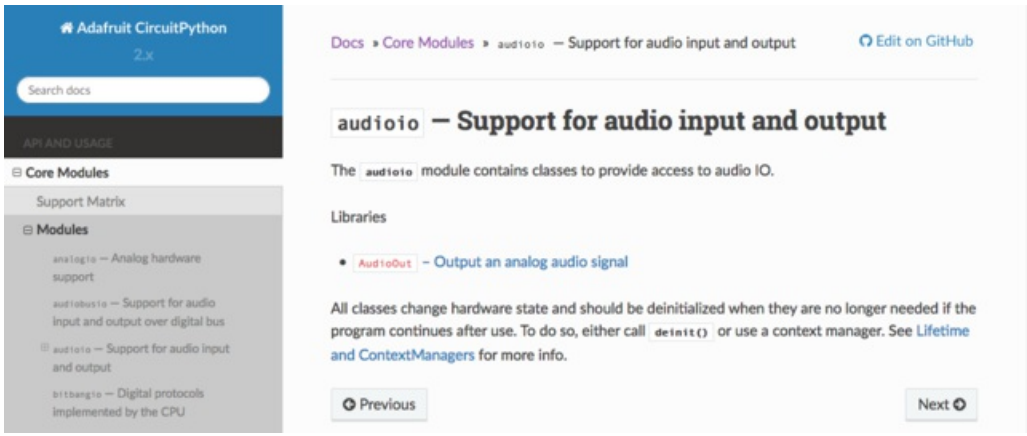
Already experienced and looking for a challenge? Checkout the rest of the issues list and you'll find plenty of ways to contribute. You'll find everything from new driver requests to core module updates. There's plenty of opportunities for everyone at any level!

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



ReadTheDocs 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. 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!

CircuitPython Playground

Here are some demos showing how to use the circuit playground with python!

CircuitPython Built-Ins

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

This is not an exhaustive list! It's just some of the many featuers you can use

Things that are Built In and Work

flow control

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

math

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

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

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

tuples, lists, arrays, and dictionaries

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

classes/objects and functions

We use objects and functions extensively in our libraries so check out one of our many examples like this MCP9808 library for class examples

lambdas

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

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

Things to watch out for!

- The wide body of python libraries have not been ported over, so while we wish you could `import numpy`, numpy isn't available. So you may have to port some code over yourself!
- For the ATSAMD21 based boards (Feather M0, Metro M0, Trinket M0, Gemma M0, Circuit PlayGround Express) there's a limited amount of RAM, we've found you can have about 250-ish lines of python (that's with various libraries) before you hit MemoryErrors. The upcoming SAMD51 chipset will help with that a ton but its not yet available)
- Non-Express boards like Trinket M0 and Gemma M0 and non-Express Feathers do not include all of the hardware support. For example, `audioio` and `bitbangio` are not included.
- Integers can only be up to 31 bits. Integers of unlimited size are not supported.

- We keep up with MicroPython stable releases, so check out the core 'differences' they document here.

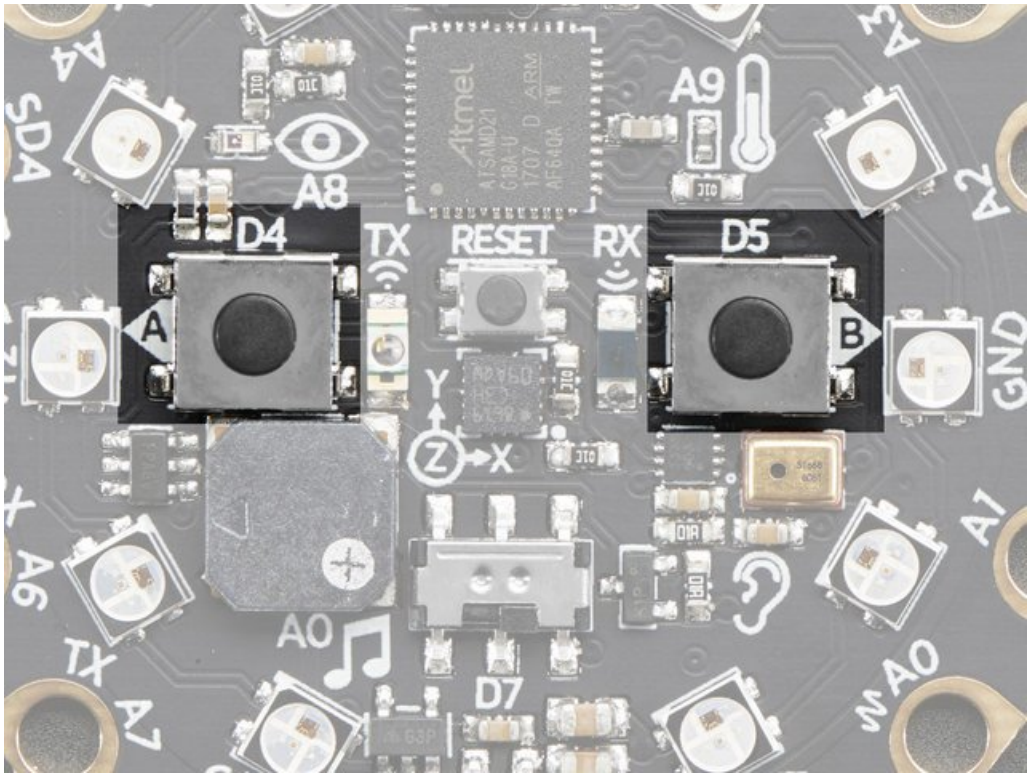
CircuitPython Digital In & Out

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

This quick-start example shows how you can use one of the Circuit Playground Express buttons as an *input* to control another digital *output* - the built in LED

Copy and paste the code block into **code.py** using your favorite text editor, and save the file, to run the demo

```
# CircuitPlaygroundExpress_DigitalIO

from digitalio import DigitalInOut, Direction, Pull
import board
import time

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

button = DigitalInOut(board.BUTTON_A)
button.direction = Direction.INPUT
button.pull = Pull.DOWN

while True:
    if button.value == True:  # button is pushed
        led.value = True
    else:
        led.value = False

    time.sleep(0.01)
```



We're using the built-in pushbuttons in this example because it's very easy to get started, but you can use ALL KINDS of different buttons and switches, even homemade ones such as tinfoil or pennies, as digital inputs connected to the Digital IO pads!

Note that we made the code a little less 'pythonic' than necessary, the if/then could be replaced with a simple `led.value = not button.value` but I wanted to make it super clear how to test the inputs. When the interpreter gets to evaluating **button.value** that is when it will read the digital input.

Press Button A (the one on the left), and the onboard red LED will turn on!

Note that on the M0/SAMD based CircuitPython boards, at least, you can also have internal pullups with **Pull.UP** when using external buttons, but the built in buttons require **Pull.DOWN**.

Maybe you're setting up your own external button with pullup or pulldown resistor. If you want to turn off the internal pullup/pulldown just assign **button.pull** = **None**

Going Beyond the Lesson!

It's time to flex your new learnings and try something different!!

Experiment 1

See if you can adjust your code so that you use Button B instead of Button A.

It only takes a small change to switch buttons. If you get stuck, click on the blurry text below to reveal a hint and then the answer:

[redacted]

Experiment 2

Perhaps you want to be sure there are no accidental illuminations of the red LED! Make it so that BOTH buttons must be pressed in order to light the red LED.

Hints:

[redacted]

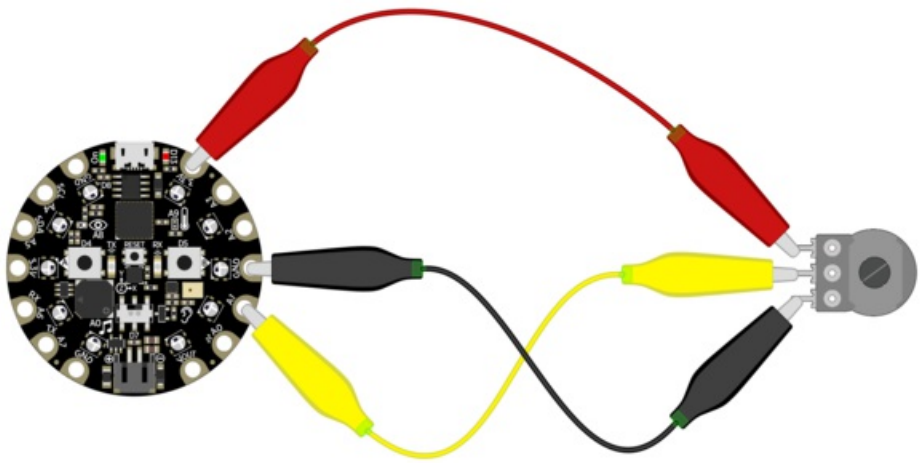
Experiment 3

Try testing the slide switch instead of the buttons. For the slide switch you need to use **Pull.UP** instead of **Pull.DOWN**.

Hints:

CircuitPython Analog In

This quick-start example shows how you can read the analog voltage of a potentiometer connected to the Circuit Playground Express.



First, connect your potentiometer to the Circuit Playground Express using three alligator clip leads, as shown. The connections are:

- Left pot connection to **3.3V**
- Center pot (wiper) to **A1**
- Right pot connection to **GND**

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

Copy and paste the code block into **code.py** using your favorite text editor, and save the file, to run the demo.

```
# CircuitPlaygroundExpress_AnalogIn
# reads the analog voltage level from a 10k potentiometer
# connected to GND, 3.3V, and pin A1
# and prints the results to the REPL

from analogio import AnalogIn
import board
import time

analogin = AnalogIn(board.A1)

def getVoltage(pin):  # helper
    return (pin.value * 3.3) / 65536

while True:
    print("Analog Voltage: %f" % getVoltage(analogin))
    time.sleep(0.1)
```

Creating an Analog Input

`analogin = AnalogIn(board.A1)` Creates an object named 'analogin' which is connected to the A1 pad on the Circuit Playground Express.

> You can use many of different kinds of external analog sensors connected to the Analog IO pads, such as distance sensors, flex sensors, and more!!

GetVoltage Helper

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

Main Loop

The main loop is simple, it will just print out the voltage as a floating point value (the %f indicates to print as **f**loating point) by calling getVoltage on each of our analog object, in this case the potentiometer.

If you connect to the serial port REPL, you'll see the voltage printed out. Try turning the knob of the potentiometer to see the voltage change!

main.py

```python
# CircuitPlaygroundExpress_AnalogIn
# reads the on-board light sensor's analog voltage level
# and prints the results

from digitalio import DigitalInOut, Direction
from analogio import AnalogIn
import board
import time

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

analogin = AnalogIn(board.LIGHT)


def getVoltage(pin): #helper
    return (pin.value * 3.3) / 65536

while True:
    print("Analog Voltage: %f" % getVoltage(analogin))
    time.sleep(0.1)
```

Adafruit CircuitPython REPL

```
Analog Voltage: 2.085964
Analog Voltage: 2.099056
Analog Voltage: 2.079268
Analog Voltage: 2.100969
Analog Voltage: 2.085562
Analog Voltage: 2.097546
Analog Voltage: 2.095430
Analog Voltage: 2.079317
Analog Voltage: 2.105955
Analog Voltage: 2.084051
Analog Voltage: 1.678399
Analog Voltage: 0.498605
Analog Voltage: 0.234851
```

Adafruit

CircuitPython Analog Out

This quick-start example shows how you can set the DAC (true analog voltage output) on pad **A0** (no other pins do analog out)

Copy and paste the code block into **main.py** using your favorite text editor, and save the file, to run the demo

```
# CircuitPython IO demo - analog output

from analogio import AnalogOut
import board
import time

aout = AnalogOut(board.A0)

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

Creating an analog output

`aout = AnalogOut(A0)`

Creates an object **aout** that is connected to the only DAC pin available - A0.

Setting the analog output

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

E.g. writing 0 will be the same as setting it to 0 - 0 Volts out

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

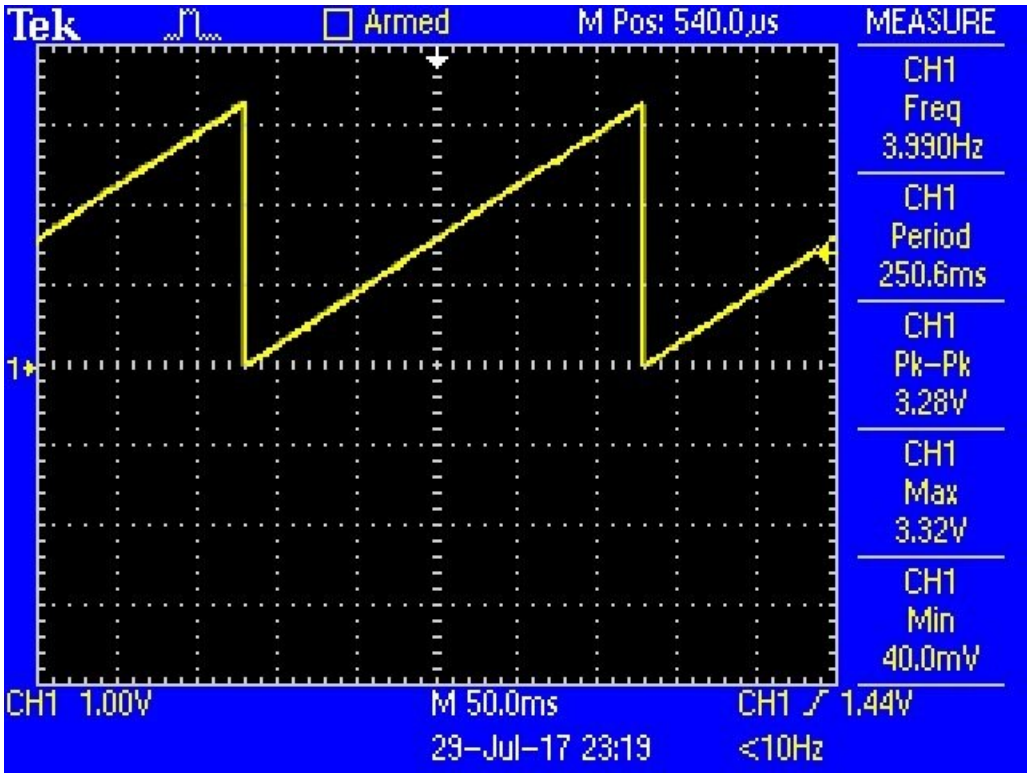
Writing 65535 is the same as 1023 which is the top range and you'll get 3.3V output

Main Loop

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

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

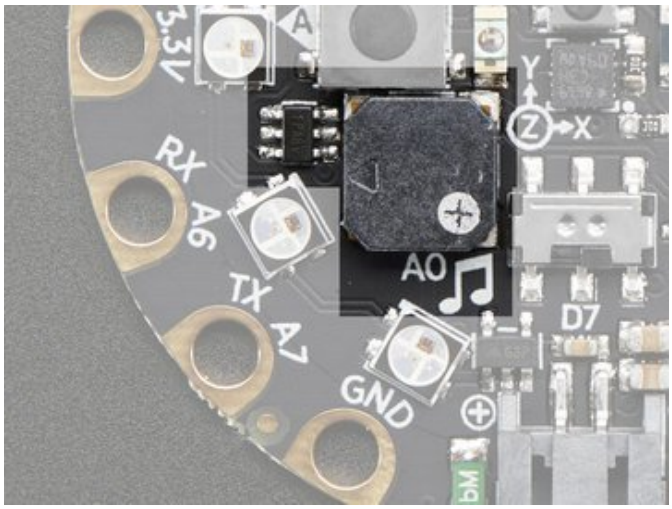
If you have an 'Express' board like the Circuit Playground Express, Metro M0 or Feather M0, these have more code space and can perform audio playback capabilities via the DAC. Gemma/Trinket M0 cannot!


Tek
Armed
M Pos: 540.0µs
MEASURE
CH1
Freq
3.990Hz
CH1
Period
250.6ms
CH1
Pk-Pk
3.28V
CH1
Max
3.32V
CH1
Min
40.0mV
CH1 1.00V
M 50.0ms
CH1 ∕ 1.44V
29-Jul-17 23:19
<10Hz

CircuitPython Audio Out

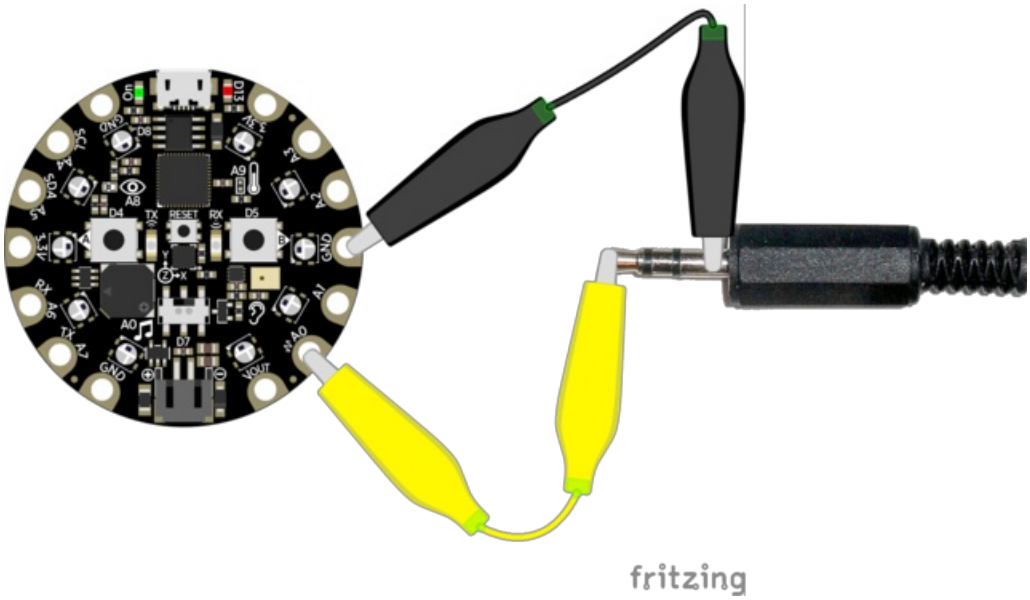
The Circuit Playground Express has some nice built in audio output capabilities.

There are **two** ways to get audio output, one is via the small built in speaker. The other is by using alligator clips to connect a headphone or speaker to the **A0** pin.



The speaker is over here, its small but can make some loud sounds! You can **ENABLE** or disable the speaker. If you disable the speaker, audio will only come out the **A0** pin. If you enable the speaker, audio will come out from both!

If you want to connect a speaker or headphones, use two alligator clips and connect **GND** to the sleeve of the headphone, and **A0** to the tip



Basic Tones

We can start by making simple tones. We will play sine waves. We first generate a single period of a sine wave in python, with the `math.sin` function, and stick it into `sine_wave`.

Then we enable the speaker by setting the `SPEAKER_ENABLE` pin to be an output and `True`.

We can create the audio object with this line that sets the output pin and you give it the sample array

`sample = audioio.AudioOut(board.SPEAKER, sine_wave)`

Since you can change the sample rate (higher samples mean higher frequencies, higher quality but more memory usage) you should set that next:

`sample.frequency = SAMPLERATE`

Finally you can run `sample.play()` - if you only want to play the sample once, call as is. If you want it to *loop* the sample, which we definitely do so its one long tone, pass in `loop=True`

You can then do whatever you like, the tone will play in the background until you call `sample.stop()`

```
from digitalio import DigitalInOut, Direction
import audioio
import board
import array
import time
import math

FREQUENCY = 440    # 440 Hz middle 'A'
SAMPLERATE = 8000  # 8000 samples/second, recommended!

# Generate one period of sine wav.
length = SAMPLERATE // FREQUENCY
sine_wave = array.array("H", [0] * length)
for i in range(length):
    sine_wave[i] = int(math.sin(math.pi * 2 * i / 18) * (2 ** 15) + 2 ** 15)

# enable the speaker
spkrenable = DigitalInOut(board.SPEAKER_ENABLE)
spkrenable.direction = Direction.OUTPUT
spkrenable.value = True

sample = audioio.AudioOut(board.SPEAKER, sine_wave)
sample.frequency = SAMPLERATE
sample.play(loop=True)  # keep playing the sample over and over
time.sleep(1)           # until...
sample.stop()           # we tell the board to stop
```

Playing Audio Files

Tones are lovely but lets play some music! You can drag-and-drop audio files onto the **CIRCUITPY** drive and then play them with a Python command

Here's the two files we're going to play:

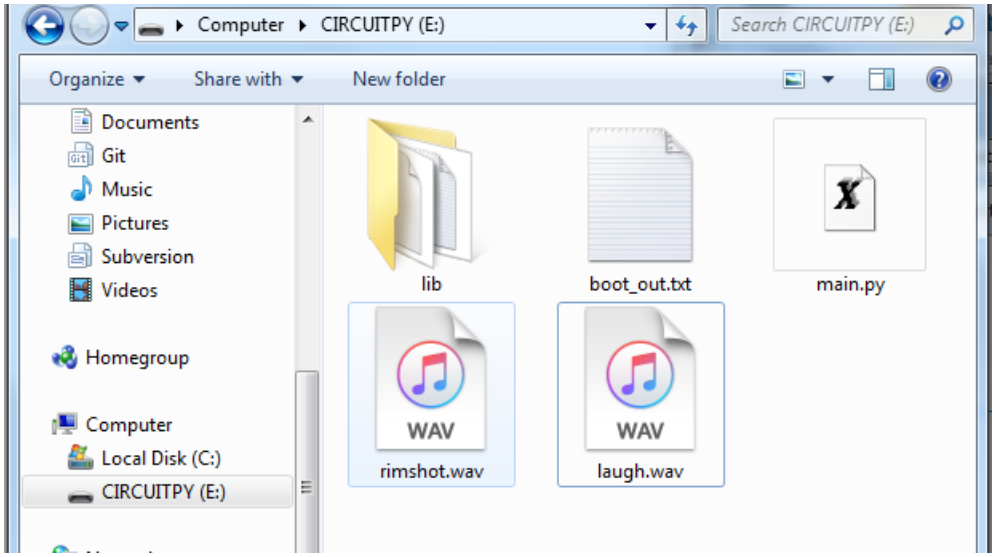
rimshot.wav

https://adafru.it/zFK

laugh.wav

https://adafru.it/zFL

Click to download, and save them onto your **CIRCUITPY** drive, alongside your **main.py** and **lib** files



This is the example code we'll be using

```
from digitalio import DigitalInOut, Direction, Pull
import audioio
import board

# enable the speaker
spkrenable = DigitalInOut(board.SPEAKER_ENABLE)
spkrenable.direction = Direction.OUTPUT
spkrenable.value = True

# make the 2 input buttons
buttonA = DigitalInOut(board.BUTTON_A)
buttonA.direction = Direction.INPUT
buttonA.pull = Pull.DOWN

buttonB = DigitalInOut(board.BUTTON_B)
buttonB.direction = Direction.INPUT
buttonB.pull = Pull.DOWN

# The two files assigned to buttons A & B
audiofiles = ["rimshot.wav", "laugh.wav"]

def play_file(filename):
    print("playing file "+filename)
    f = open(filename, "rb")
    a = audioio.AudioOut(board.A0, f)
    a.play()
    while a.playing:
        pass
    print("finished")

while True:
    if buttonA.value:
        play_file(audiofiles[0])
    if buttonB.value:
        play_file(audiofiles[1])
```

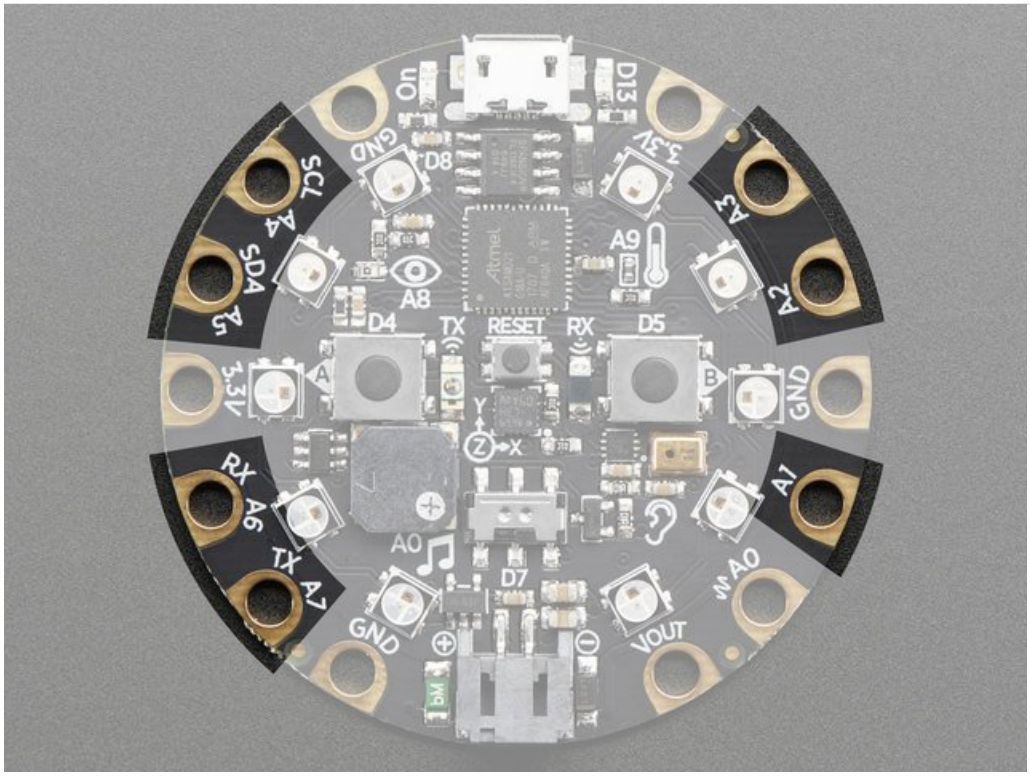
This example creates two input buttons using the onboard buttons, then has a helper that will:

1. open a file on the disk drive with `f = open(filename, "rb")`
2. create the audio playback object with `a = audioio.AudioOut(board.A0, f)`
3. and finally play it until its done:
   `a.play()`
   `while a.playing:`
   `  pass`

Upload the code then try pressing the two buttons one at a time to create your own laugh track!

If you want to use your own sound files, you can! Record, sample, remix, or simply download files from a sound file sight, such as freesample.org. Then, to make sure you have the files converted to the proper specifications, check out this guide here that'll show you how! Spoiler alert: you'll need to make a small, 22Khz (or lower), 16 bit PCM, mono .wav file!

CircuitPython Cap Touch



This quick-start example shows how you can read the capacitive touch sensors built into on seven of the CIrcuit Playground Express pads (pad **A0** is not a capacitive touch pad).

Copy and paste the code block into **code.py** using your favorite text editor, and save the file, to run the demo

```
#CircuitPlaygroundExpress_CapTouch

import touchio
import board
import time

touch1 = touchio.TouchIn(board.A1)
touch2 = touchio.TouchIn(board.A2)
touch3 = touchio.TouchIn(board.A3)
touch4 = touchio.TouchIn(board.A4)
touch5 = touchio.TouchIn(board.A5)
touch6 = touchio.TouchIn(board.A6)
touch7 = touchio.TouchIn(board.A7)


while True:
    if touch1.value:
        print("A1 touched!")
    if touch2.value:
        print("A2 touched!")
    if touch3.value:
        print("A3 touched!")
    if touch4.value:
        print("A4 touched!")
    if touch5.value:
        print("A5 touched!")
    if touch6.value:
        print("A6 touched!")
    if touch7.value:
        print("A7 touched!")

    time.sleep(0.01)
```

You can open up the serial console to see the touches detected and printed out.

Creating an capacitive touch input

Pads **A1 - A7** can be used as capacitive TouchIn devices:

```
touch1 = touchio.TouchIn(board.A1)
touch2 = touchio.TouchIn(board.A2)
touch3 = touchio.TouchIn(board.A3)
touch4 = touchio.TouchIn(board.A4)
touch5 = touchio.TouchIn(board.A5)
touch6 = touchio.TouchIn(board.A6)
touch7 = touchio.TouchIn(board.A7)
```

Creates seven objects, one connected to each of the cap touch pads.

Main Loop

The main loop checks each sensor one after the other, to determine if it has been touched. If `touch1.value` returns True, that means that that pad, `A1`, detected a touch. For each pad, if it has been touched, a message will print.

A small sleep delay is added at the end so the loop doesn't run *too* fast. You may want to change the delay from 0.1 seconds to 0 seconds to slow it down or speed it up.

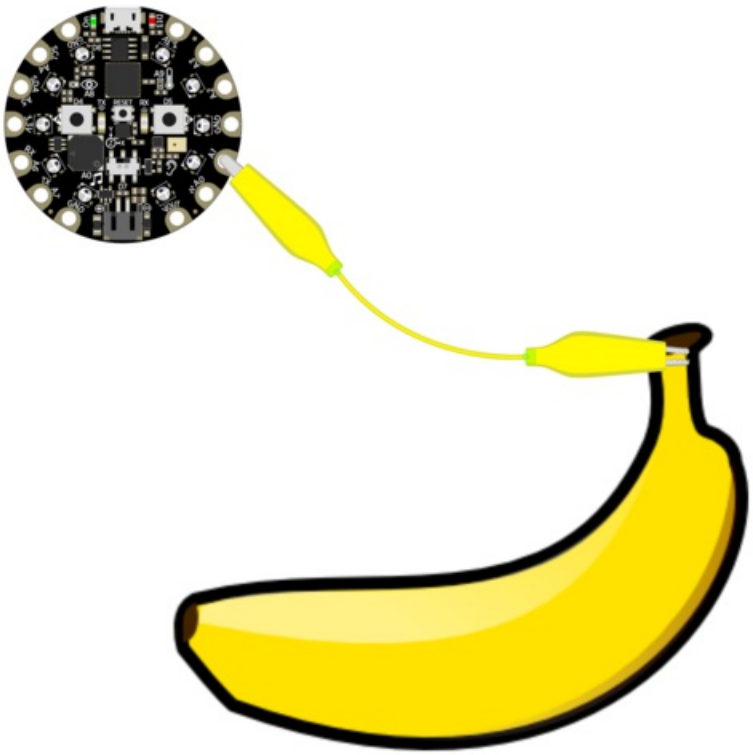
```python
#CircuitPlaygroundExpress_CapTouch

import touchio
import board
import time

touch1 = touchio.TouchIn(board.A1)
touch2 = touchio.TouchIn(board.A2)
touch3 = touchio.TouchIn(board.A3)
touch4 = touchio.TouchIn(board.A4)
touch5 = touchio.TouchIn(board.A5)
touch6 = touchio.TouchIn(board.A6)
touch7 = touchio.TouchIn(board.A7)


while True:
    if touch1.value:
        print("A1 touched!")
    if touch2.value:
        print("A2 touched!")
    if touch3.value:
        print("A3 touched!")
    if touch4.value:
        print("A4 touched!")
    if touch5.value:
        print("A5 touched!")
    if touch6.value:
        print("A6 touched!")
    if touch7.value:
        print("A7 touched!")

    time.sleep(0.01)
```

Adafruit CircuitPython REPL

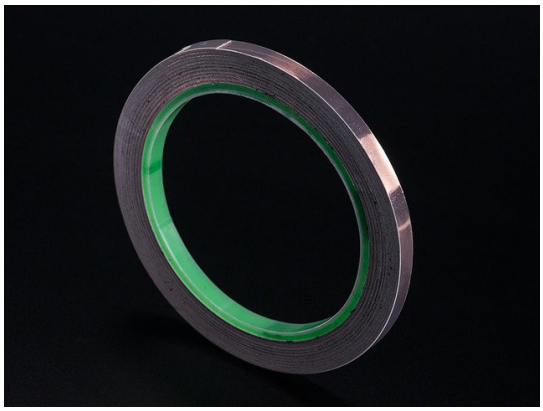
```
A5 touched!
A1 touched!
A5 touched!
A1 touched!
A5 touched!
A1 touched!
A5 touched!
A1 touched!
A1 touched!
A1 touched!
```

Adafruit

Note that no extra hardware is required, you can touch the pads directly, but you may want to attach alligator clips or foil tape to metallic or conductive objects. Try silverware, fruit or other food, liquid, aluminum foil, and items around your desk!



You may need to restart your code/board after changing the attached item because the capacitive touch code 'calibrates' based on what it sees when it first starts up. So if you get too many touch-signals or not enough, hit that reset button!



Copper Foil Tape with Conductive Adhesive - 6mm x 15 meter roll

PRODUCT ID: 1128

https://adafru.it/eNZ

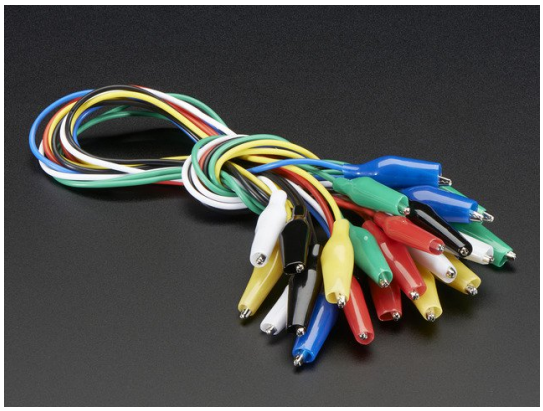
$5.95
IN STOCK

Copper Foil Tape with Conductive Adhesive - 25mm x 15 meter roll

PRODUCT ID: 1127

https://adafru.it/y8F

$19.95
IN STOCK



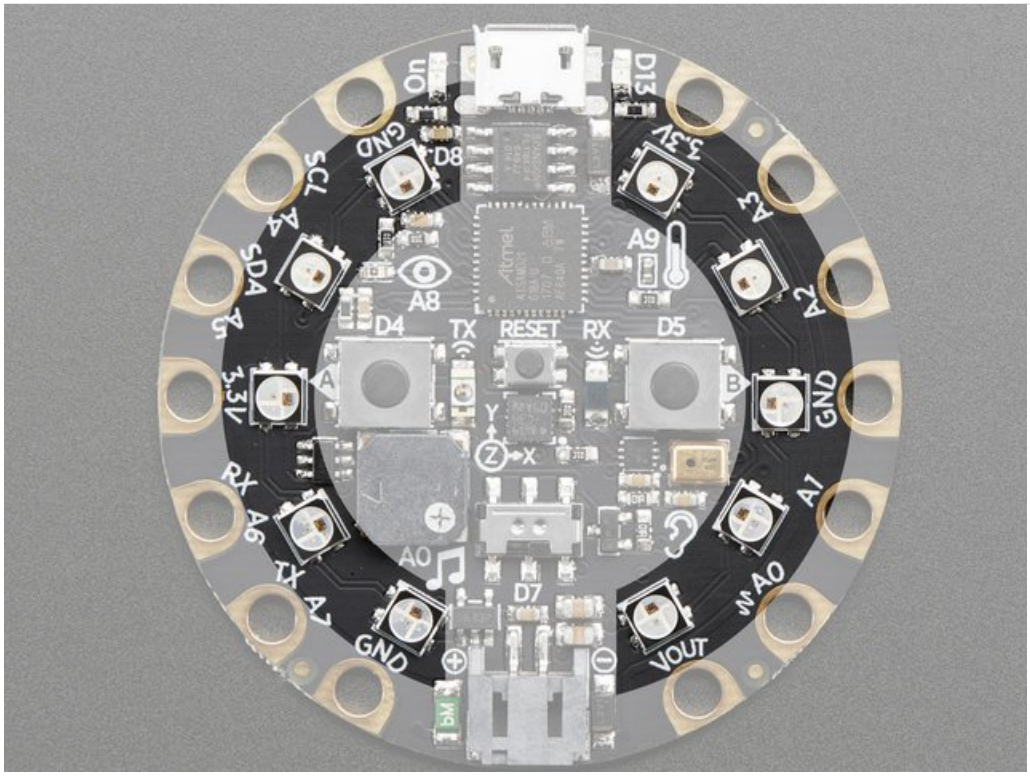
Small Alligator Clip Test Lead (set of 12)

PRODUCT ID: 1008

https://adafru.it/dWJ

$3.95
IN STOCK

CircuitPython NeoPixel



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

You can drive 300 pixels with brightness control and 1000 pixels without (set `brightness=1.0` in object creation). That's because to adjust the brighness we have to dynamically re-create the datastream each write.

Here's an example with a lot of different visual effects you can check out. You'll need the **neopixel.mpy** library file if you don't have it yet!

```
# CircuitPlaygroundExpress_NeoPixel

import board
import neopixel
import time

pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=.2)
pixels.fill((0,0,0))
pixels.show()

#choose which demos to play
# 1 means play, 0 means don't!
simpleCircleDemo = 1
flashDemo = 1
rainbowDemo = 1
rainbowCycleDemo = 1

def wheel(pos):
    # Input a value 0 to 255 to get a color value.
```

```
    # The colours are a transition r - g - b - back to r.
    if pos < 85:
        return (int(pos*3), int(255 - (pos*3)), 0)
    elif pos < 170:
        pos -= 85
        return (int(255 - (pos*3)), 0, int(pos*3))
    else:
        pos -= 170
        return (0, int(pos*3), int(255 - pos*3))

def rainbow_cycle(wait):
    for j in range(255):
        for i in range(len(pixels)):
            idx = int ((i * 256 / len(pixels)) + j*10)
            pixels[i] = wheel(idx & 255)
        pixels.show()
        time.sleep(wait)

def rainbow(wait):
    for j in range(255):
        for i in range(len(pixels)):
            idx = int (i+j)
            pixels[i] = wheel(idx & 255)
        pixels.show()
        time.sleep(wait)

def simpleCircle(wait):
    RED = 0x100000 # (0x10, 0, 0) also works
    YELLOW=(0x10, 0x10, 0)
    GREEN = (0, 0x10, 0)
    AQUA = (0, 0x10, 0x10)
    BLUE = (0, 0, 0x10)
    PURPLE = (0x10, 0, 0x10)
    BLACK = (0, 0, 0)

    for i in range(len(pixels)):
        pixels[i] = RED
        time.sleep(wait)
    time.sleep(1)

    for i in range(len(pixels)):
        pixels[i] = YELLOW
        time.sleep(wait)
    time.sleep(1)

    for i in range(len(pixels)):
        pixels[i] = GREEN
        time.sleep(wait)
    time.sleep(1)

    for i in range(len(pixels)):
        pixels[i] = AQUA
        time.sleep(wait)
    time.sleep(1)

    for i in range(len(pixels)):
        pixels[i] = BLUE
        time.sleep(wait)
    time.sleep(1)
```

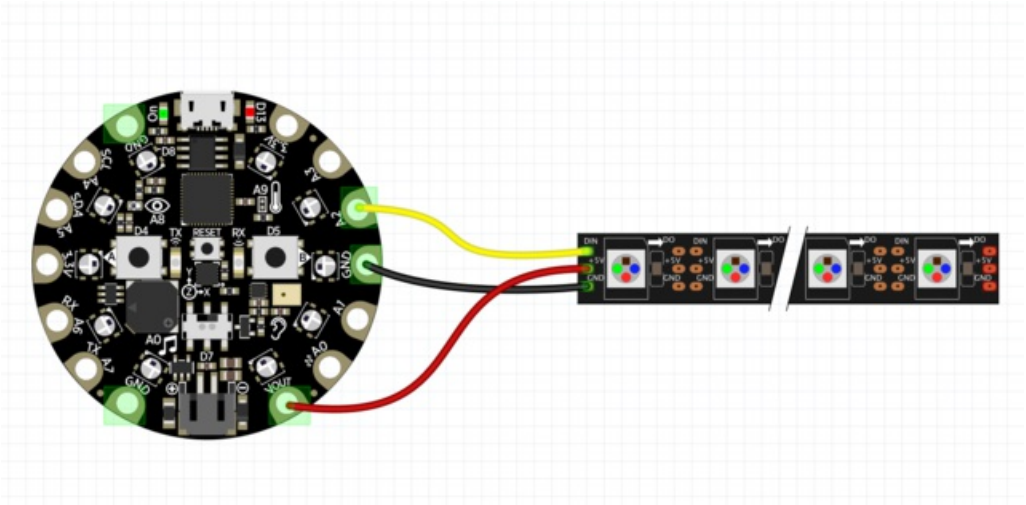
```
    for i in range(len(pixels)):
        pixels[i] = PURPLE
        time.sleep(wait)
    time.sleep(1)

    for i in range(len(pixels)):
        pixels[i] = BLACK
        time.sleep(wait)
    time.sleep(1)


while True:
    if simpleCircleDemo:
        print('Simple Circle Demo')
        simpleCircle(.05)

    if flashDemo: #this will play if flashDemo = 1 up above
        print('Flash Demo')
        pixels.fill((255, 0, 0))
        pixels.show()
        time.sleep(.25)

        pixels.fill((0, 255, 0))
        pixels.show()
        time.sleep(.25)

        pixels.fill((0, 0, 255))
        pixels.show()
        time.sleep(.25)

        pixels.fill((255, 255, 255))
        pixels.show()
        time.sleep(.25)

    if rainbowDemo:
        print('Rainbow Demo')
        rainbow(.001)

    if rainbowCycleDemo:
        print('Rainbow Cycle Demo')
        rainbow_cycle(.001)
```

You aren't limited to the on-board NeoPixels -- externally connected NeoPixels can be driven by any Digital IO pin.

For powering the pixels from the board, the 3.3V regulator output can handle about 500mA peak which is about 50 pixels with 'average' use. If you want really bright lights and a lot of pixels, we recommend powering direct from the power source. On the Circuit Playground Express this is the **Vout** pad - that pad has direct power from USB or BAT, depending on which is higher voltage.

The NeoPixel object's argument list requires the pin you'll use (any pin can be used) and the number of pixels. There's two optional arguments, `brightness` (range from 0 off to 1.0 full brightness) and auto_write. When `auto_write` default is set to True, where every change is immediately written to the strip of pixels, this is easier to use but *way* slower. if you set `auto_write=False` then you will have to call `strip.show()` when you want to actually write color data out.

You can easily set colors by indexing into the location `strip[n] = (red, green, blue)` . For example, `strip[0] = (100, 0, 0)` will set the first pixel to a medium-brightness red, and `strip[2] = (0, 255, 0)` will set the third pixel to bright green. Then, if you have `auto_write=False` don't forget to call `strip.show()`



Verify the wiring on your strip or device - plugging into the 'DOUT' side is a common mistake! Wire up NeoPixels only while the Circuit Playground Express is not on, to avoid possible damage!

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

We have a ton more information on general purpose NeoPixel know-how at our NeoPixel UberGuide https://learn.adafruit.com/adafruit-neopixel-uberguide

CircuitPython HID Keyboard

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

You'll need to install the driver bundle which comes with Keyboard, Keycode and Mouse support

Then try running this example code which will set the Circuit Playground Express **Button_A** and **Button_B** as HID keyboard "keys".

```
# CircuitPlaygroundExpress_HIDKeyboard

from digitalio import DigitalInOut, Direction, Pull
import touchio
import board
import time
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS

# A simple neat keyboard demo in circuitpython

# The button pins we'll use, each will have an internal pulldown
buttonpins = [board.BUTTON_A, board.BUTTON_B]
# our array of button objects
buttons = []
# The keycode sent for each button, will be paired with a control key
buttonkeys = [Keycode.A, "Hello World!\n"]
controlkey = Keycode.SHIFT

# the keyboard object!
kbd = Keyboard()
# we're americans :)
layout = KeyboardLayoutUS(kbd)

# make all pin objects, make them inputs w/pulldowns
```

```
# make all pin objects, make them inputs w/pulldowns
for pin in buttonpins:
    button = DigitalInOut(pin)
    button.direction = Direction.INPUT
    button.pull = Pull.DOWN
    buttons.append(button)

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

print("Waiting for button presses")

while True:
    # check each button
    # when pressed, the LED will light up,
    # when released, the keycode or string will be sent
    # this prevents rapid-fire repeats!
    for button in buttons:
        if  button.value:   # pressed?
            i = buttons.index(button)
            print("Button #%d Pressed" % i)

            # turn on the LED
            led.value = True

            while  button.value:
                pass  # wait for it to be released!
            # type the keycode or string
            k = buttonkeys[i]    # get the corresp. keycode/str
            if type(k) is str:
                layout.write(k)
            else:
                kbd.press(controlkey, k) # press...
                kbd.release_all()        # release!

            # turn off the LED
            led.value = False

    time.sleep(0.01)
```

Press Button A or Button B to have the keypresses sent.

The Keyboard and Layout object are created, we only have US right now (if you make other layouts please submit a GitHub pull request!)

```
# the keyboard object!
kbd = Keyboard()
# we're americans :)
layout = KeyboardLayoutUS(kbd)
```

Then you can send key-down's with `kbd.press(keycode, ...)` You can have up to 6 keycode presses at once. Note that these are **keycodes** so if you want to send a capital A, you need both SHIFT and A. Don't forget to call `kbd.release_all()` soon after or you'll have a stuck key which is really annoying!

You can also send full strings, with `layout.write("Hello World!\n")` - it will use the layout to determine the keycodes to press.

For more detail check out the documentation at https://circuitpython.readthedocs.io/projects/hid/en/latest/

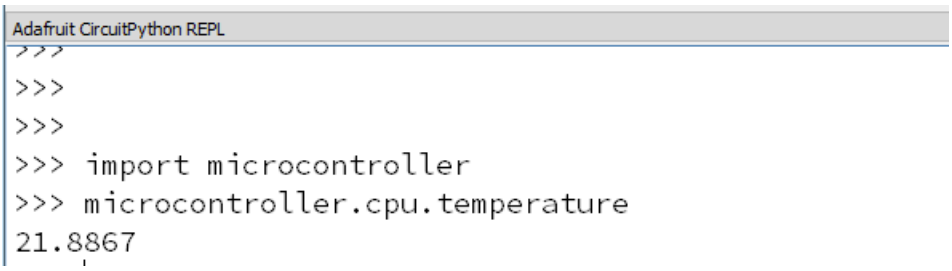
CircuitPython CPU Temp

This little built in sensor comes with all ATSAMD21 chips, and its really nice to have a temperature sensor so we let you read it via CircuitPython, its new since 2.0.0 and only available on the ATSAMD21-based boards (e.g. not ESP8266)

It's so easy, we'll just give you the two REPL commands

```
>>> import microcontroller
>>> microcontroller.cpu.temperature
```

That's it! You'll have the temperature in Centigrade printed out. Note it is not *exactly* the same as ambient temperature, and its not super precise. But it's kinda close!

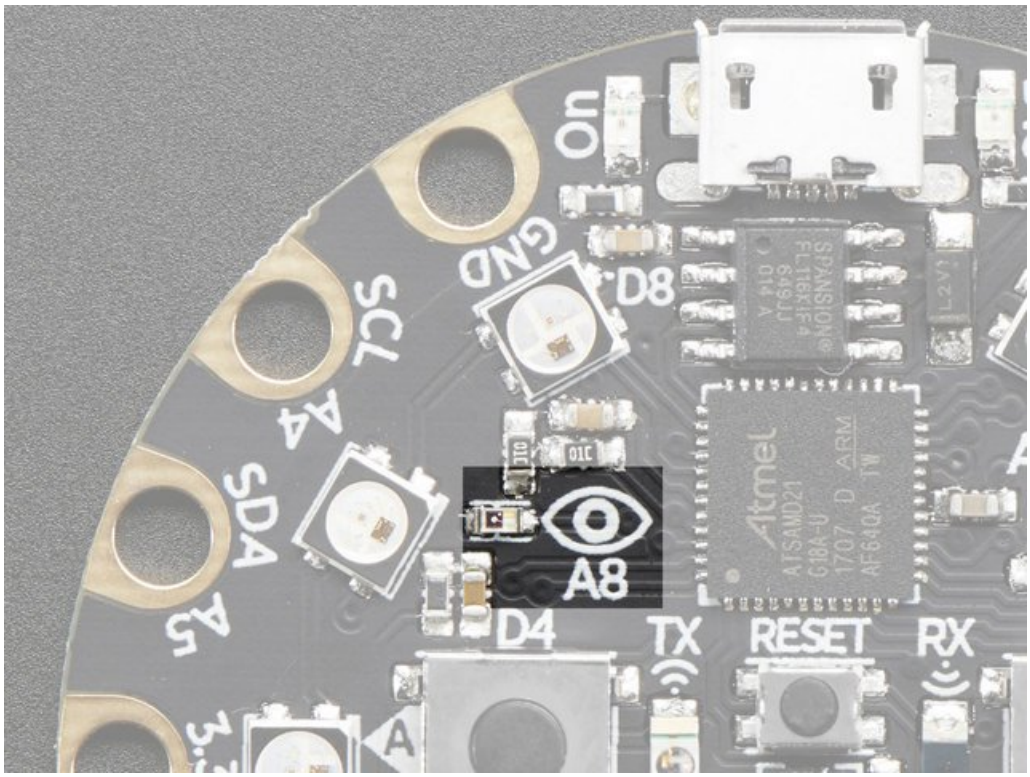


Playground Light Sensor

The Circuit Playground Express can see you! OK, not really. That would be creepy.

But, it can sense light and dark, as well as colors and even your pulse!!

The Light Sensor in the upper left of the board (look for the eye icon) is a phototransistor. Here's how to use it as a light sensor:



```
# CircuitPlaygroundExpress_LightSensor
# reads the on-board light sensor and graphs the brighness with NeoPixels

from simpleio import map_range
from analogio import AnalogIn
import board
import neopixel
import time

pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, auto_write=0, brightness=.05)
pixels.fill((0,0,0))
pixels.show()

analogin = AnalogIn(board.LIGHT)

while True:
    #light value remaped to pixel position
    peak = map_range(analogin.value, 2000, 62000, 0, 9)
    print(analogin.value)
    print(int(peak))

    for i in range(0, 9, 1):
         if i <= peak:
             pixels[i] = (0, 255, 0)
         else:
             pixels[i] = (0,0,0)
    pixels.show()

    time.sleep(0.01)
```

Copy and paste that code into a text editor and then save it to your Circuit Playground Express as **code.py**.

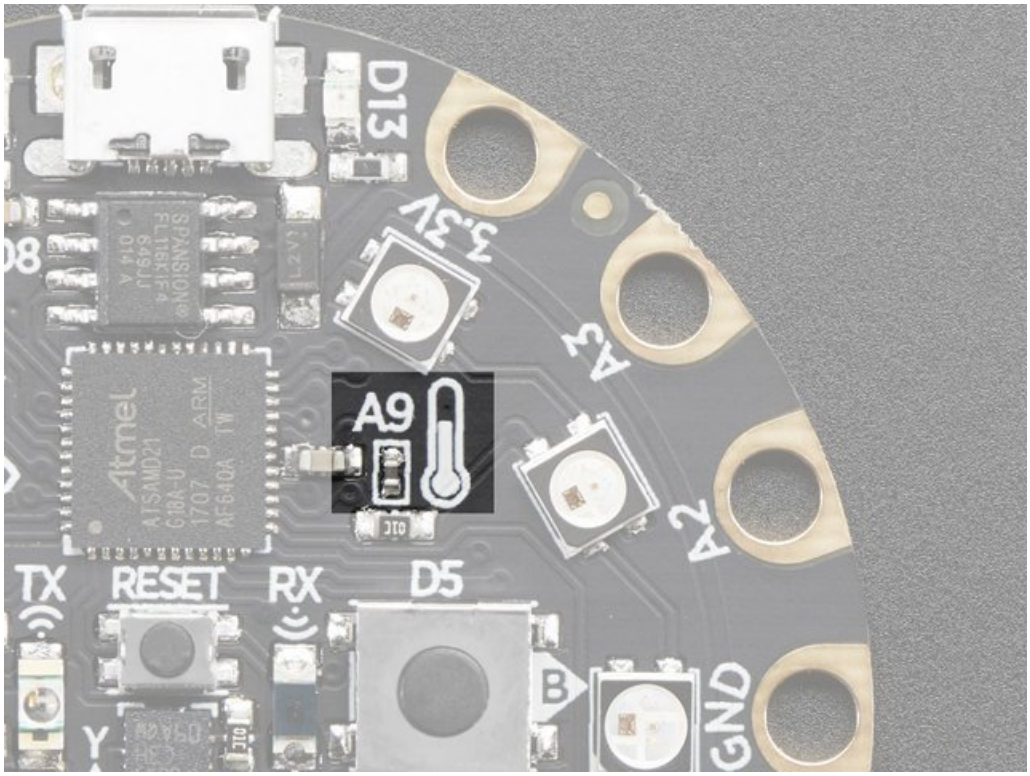
The code reads the light sensor and then lights up the NeoPixels like a bar graph depending on the light level. Try waving your hand over it, or shining it with a flashlight to see it change!

Playground Temperature

But wait! There's more -- the Circuit Playground Express can also tell the temperature!

How, you ask? With a build in thermistor. This little sensor is a thermally sensitive resistor, meaning it's resistance changes based on temperature.

We can access its readings in CircuitPython by importing the **adafruit_thermistor** library, and then using the `board.TEMPERATURE` pin to read the thermistor value.



Copy the code below in to a new file, then save it onto the board as main.py. Then, open up a REPL session and you'll see the temperature readings in both Celsius and Fahrenheit.

```
# CircuitPlaygroundExpress_Temperature
# reads the on-board temperature sensor and prints the value

import adafruit_thermistor
import board
import time

thermistor = adafruit_thermistor.Thermistor(board.TEMPERATURE, 10000, 10000, 25, 3950)


while True:
    print("Temperature is: %f C and %f F" % (thermistor.temperature,
                                             (thermistor.temperature*9/5+32)))

    time.sleep(0.25)
```

Try placing your finger over the sensor (you'll see a thermometer icon on the board) and watch the readings change.

```python
# CircuitPlaygroundExpress_Temperature
# reads the on-board temperature sensor and prints the value

import adafruit_thermistor
import board
import time

thermistor = adafruit_thermistor.Thermistor(board.TEMPERATURE, 10000, 10000, 25, 3950)

while True:
    print("Temperature is: %f C and %f F" % (thermistor.temperature,
                                             (thermistor.temperature*9/5+32)))

    time.sleep(0.25)
```

Adafruit CircuitPython REPL

```
Temperature is: 24.327760 C and 75.785127 F
Temperature is: 24.335938 C and 75.804687 F
Temperature is: 24.334595 C and 75.802245 F
Temperature is: 24.342773 C and 75.819397 F
Temperature is: 24.337280 C and 75.799837 F
Temperature is: 24.293580 C and 75.740724 F
Temperature is: 24.278443 C and 75.701175 F
Temperature is: 24.279907 C and 75.698762 F
Temperature is: 24.287965 C and 75.733275 F
Temperature is: 24.308593 C and 75.757875 F
Temperature is: 24.353759 C and 75.866427 F
Temperature is: 25.161011 C and 77.316833 F
Temperature is: 25.947754 C and 78.765502 F
Temperature is: 26.945558 C and 80.532103 F
Temperature is: 27.802856 C and 82.090607 F
Temperature is: 28.798950 C and 83.881598 F
Temperature is: 29.223022 C and 84.606695 F
Temperature is: 29.210205 C and 84.573307 F
```

Adafruit

Playground Drum Machine

A wise man once said, ***"Nothing sounds better than an Eight-O-Eight."***

(That wise man was Adam Horovitz of the Beastie Boys.)

The 808 to which Ad-Rock was referring is the Roland TR-808 drum machine. Let's build a little Circuit Playground Express tribute to the venerable 808! Instead of a full-blown drum pattern sequencer, we'll just focus on the machine's pads which are used for finger drumming to play back sampled drum sounds.

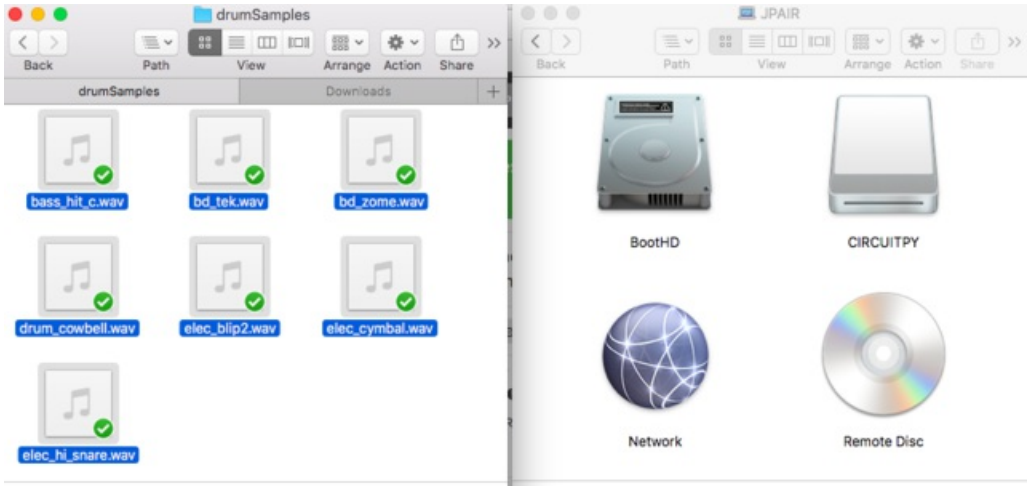
We can use the capacitive touch pads on the Circuit Playground Express as triggers, and small .wav files for our drum sounds!

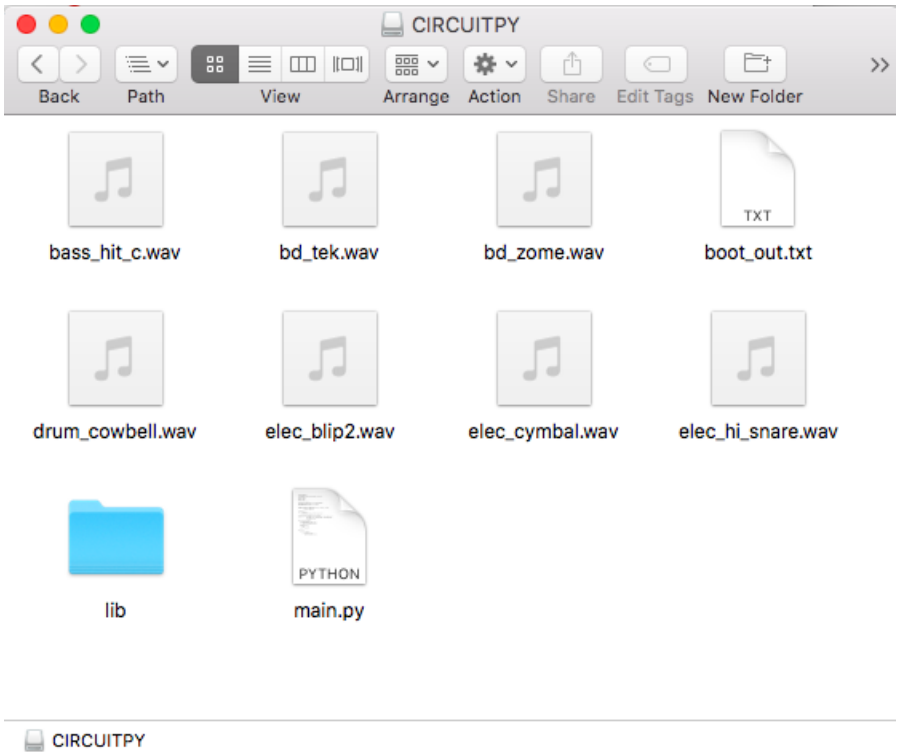
First, download the .zip file below, which contains all of the drum samples we'll be using. Save the file to your desktop or somewhere else easy to find, and then unzip it.

drumSamples.zip

https://adafru.it/zHc

You can plug in your Circuit Playground Express, and then drag the drum files onto it. It shows up as the CIRCUITPY drive.



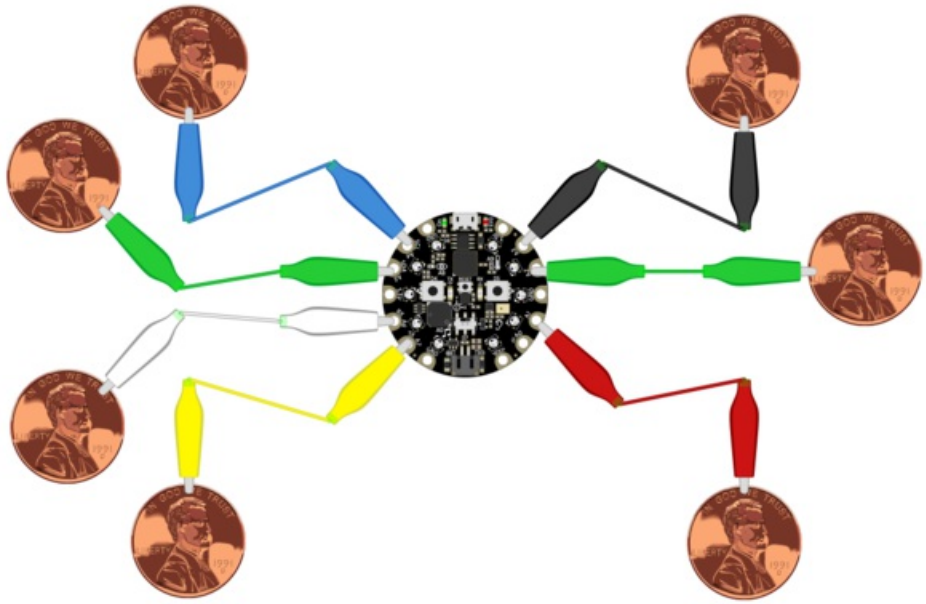


Now, it's time to code the Circuit Playground Express! Copy the code shown below, and then paste it into your text editor of choice. Save the file as **main.py** (or **code.py**) on your CIRCUITPY.

```
# Playground 808
# Drum machine
from digitalio import DigitalInOut, Direction
import audioio
import touchio
import board
import time

bpm = 120 #beats per minute for sustained hold, change this to suit your tempo

# enable the speaker
spkrenable = DigitalInOut(board.SPEAKER_ENABLE)
spkrenable.direction = Direction.OUTPUT
spkrenable.value = True

# make the input cap sense pads
capPins = (board.A1, board.A2, board.A3, board.A4, board.A5,
           board.A6, board.A7)

touchPad = []
for i in range(7):
    touchPad.append(touchio.TouchIn(capPins[i]))

# The seven files assigned to the touchpads
audiofiles = ["bd_tek.wav", "elec_hi_snare.wav", "elec_cymbal.wav",
               "elec_blip2.wav", "bd_zome.wav", "bass_hit_c.wav",
               "drum_cowbell.wav"]

def play_file(filename):
    print("playing file "+filename)
    f = open(filename, "rb")
    a = audioio.AudioOut(board.A0, f)
    a.play()
    time.sleep(bpm/960) #sixteenthNote delay

while True:

    for i in range(7):
        if touchPad[i].value:
            play_file(audiofiles[i])
```

Try it out! When you tap the capacitive pads, the corresponding drum sample is triggered!!

Things are a bit crammed, admittedly, so you can try adding foil, copper tape, alligator clips, etc. in order to increase the surface area and physical space you have for your drumming!



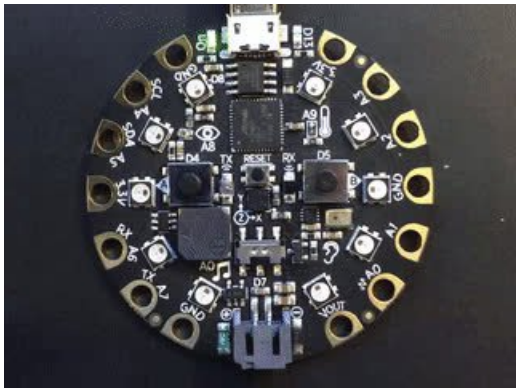
Capacitance is calibrated at startup, so you may need to reset the board after attaching leads to the pads!

If you want to use your own sound files, you can! Record, sample, remix, or simply download files from a sound file sight, such as freesample.org. Then, to make sure you have the files converted to the proper specifications, check out this guide here that'll show you how! Spoiler alert: you'll need to make a small, 22Khz (or lower), 16 bit PCM, mono .wav file!

Want to listen to your Drum Machine at body movin' volumes? No problem! Hook up an1/8" or 1/4" phono output to the **GND** and **A0** pads, then plug in to an amp!! I tried it on a small guitar amp and it sounds great.

Playground Sound Meter

Use the microphone on your Circuit Playground Express to measure sound levels and display them on a VU-meter-like display!



Note: This program will only work with the improved microphone support in CircuitPython 2.2.0 or later.

The program is below. There are many settings that you can change to make the readings more or less sensitive and the display more or less jumpy. Try changing `CURVE` to be 4 or 1 or 10 or -2 and see what happens.

The program samples audio for a short time and the computes the energy in the sample (corresponding to volume) using a Root-Mean-Square computation (RMS). You could try varying the sample size if you like.

The `log_scale()` function varies the number of NeoPixels lit in an exponential way, because sound levels can vary by many factors of 10 between loud and soft. Try varying how it's computed to see what happens.

The program takes one sample when it first starts to set a "quiet" sound level. If that doesn't work for you, set `input_floor` to be a fixed number. If the meter seems too sensitive, try changing `input_ceiling = input_floor + 500` to be `input_ceiling = input_floor + 2000` or higher. Or go the other way.

You can also change the colors. Try different ways of computing `volume_color(i)` for more of a rainbow effect, or make it a constant if you don't like changing colors.

```
# The MIT License (MIT)
#
# Copyright (c) 2017 Dan Halbert for Adafruit Industries
# Copyright (c) 2017 Kattni Rembor, Tony DiCola for Adafruit Industries
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
```

```
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.

import array
import audiobusio
import board
import math
import neopixel

# Exponential scaling factor.
# Should probably be in range -10 .. 10 to be reasonable.
CURVE = 2
SCALE_EXPONENT = math.pow(10, CURVE*-0.1)

PEAK_COLOR = (100, 0, 255)
NUM_PIXELS = 10

# Number of samples to read at once.
NUM_SAMPLES = 160

# Restrict value to be between floor and ceiling.
def constrain(value, floor, ceiling):
    return max(floor, min(value, ceiling))

# Scale input_value to be between output_min and output_max, in an exponential way.
def log_scale(input_value, input_min, input_max, output_min, output_max):
    normalized_input_value = (input_value - input_min) / (input_max - input_min)
    return output_min + math.pow(normalized_input_value, SCALE_EXPONENT) * (output_max - output_min)

# Remove DC bias before computing RMS.
def normalized_rms(values):
    minbuf = int(mean(values))
    return math.sqrt(sum(float((sample-minbuf)*(sample-minbuf)) for sample in values)/len(values))

def mean(values):
    return (sum(values) / len(values))

def volume_color(i):
    return (200, i*(255//NUM_PIXELS), 0)

# Main program

# Set up NeoPixels and turn them all off.
pixels = neopixel.NeoPixel(board.NEOPIXEL, NUM_PIXELS, brightness=0.1, auto_write=False)
pixels.fill(0)
pixels.show()

mic = audiobusio.PDMIn(board.MICROPHONE_CLOCK, board.MICROPHONE_DATA, frequency=16000, bit_depth=16)
# Record an initial sample to calibrate. Assume it's quiet when we start.
samples = array.array('H', [0] * NUM_SAMPLES)
mic.record(samples, len(samples))
# Set lowest level to expect, plus a little.
input_floor = normalized_rms(samples) + 10
# OR: used a fixed floor
# input_floor = 50

# You might want to print the input_floor to help adjust other values.
# print(input_floor)
```

```
# print(input_floor)

# Corresponds to sensitivity: lower means more pixels light up with lower sound
# Adjust this as you see fit.
input_ceiling = input_floor + 500

peak = 0
while True:
    mic.record(samples, len(samples))
    magnitude = normalized_rms(samples)
    # You might want to print this to see the values.
    # print(magnitude)

    # Compute scaled logarithmic reading in the range 0 to NUM_PIXELS
    c = log_scale(constrain(magnitude, input_floor, input_ceiling), input_floor, input_ceiling, 0, NUM_PI

    # Light up pixels that are below the scaled and interpolated magnitude.
    pixels.fill(0)
    for i in range(NUM_PIXELS):
        if i < c:
            pixels[i] = volume_color(i)
        # Light up the peak pixel and animate it slowly dropping.
        if c >= peak:
            peak = min(c, NUM_PIXELS-1)
        elif peak > 0:
            peak = peak - 1
        if peak > 0:
            pixels[int(peak)] = PEAK_COLOR
    pixels.show()
```

Handy Tips

Check Heap Memory Usage

`import gc`

`gc.mem_free()`

Will give you the number of bytes available for use.

Random Numbers

`import random`

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

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

Arduino

> Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.
>
> -- https://www.arduino.cc/en/Guide/Introduction

Arduino is the third and oldest of the progamming languages supported by Circuit Playground Express. Arduino has over a decade of projects and history, so you'll find a lot of existing code that you can use with your CPX.

Arduino is essentially C/C++ with a built in library of hardware interfaces. It is the most low-level of the three languages - you can embed assembly, write ultra-fast code, and you get the full use of the entire memory and filesystem. But... its harder for beginners to use! If you're an Arduino expert you may want to just use CPX with the Arduino IDE.

If you're starting out on your coding journey, check out MakeCode first.

If you want fast development without uploading/compiling or memory management, check out CircuitPython. It's not as clock-cycle-fast as Arduino but is faster to code in, simpler and more fun.

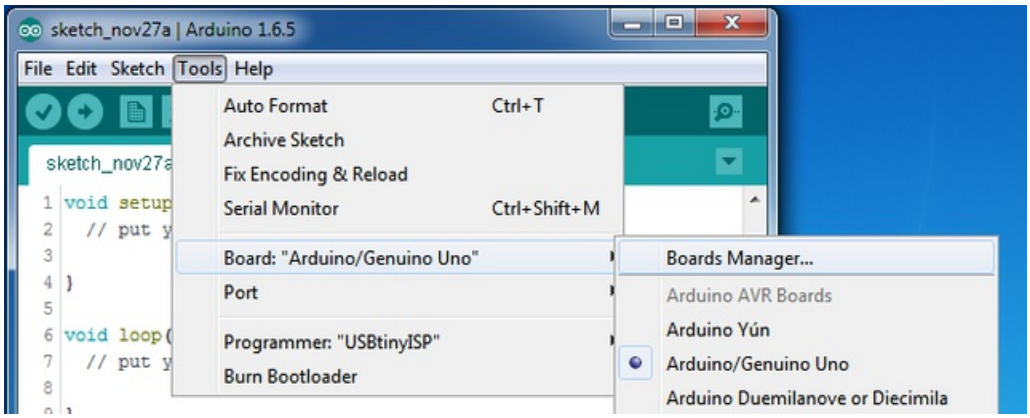
Set Up Arduino IDE

Since the Circuit Playground Express uses an ATSAMD21 chip running at 48 MHz, 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, especially devices & sensors that use i2c or SPI.

The Circuit Playground Express is 'natively' supported in the Arduino IDE so its **really easy** to set up!

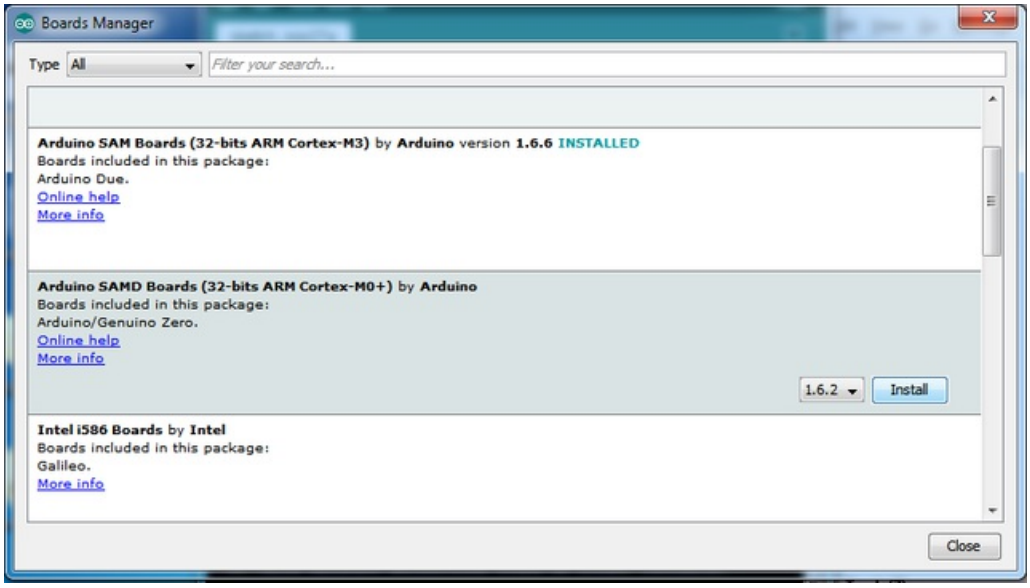
Install SAMD Support

Open the **Boards Manager** by navigating to the **Tools->Board** menu



First up, install the **Arduino SAMD Boards** version **1.6.16** or later

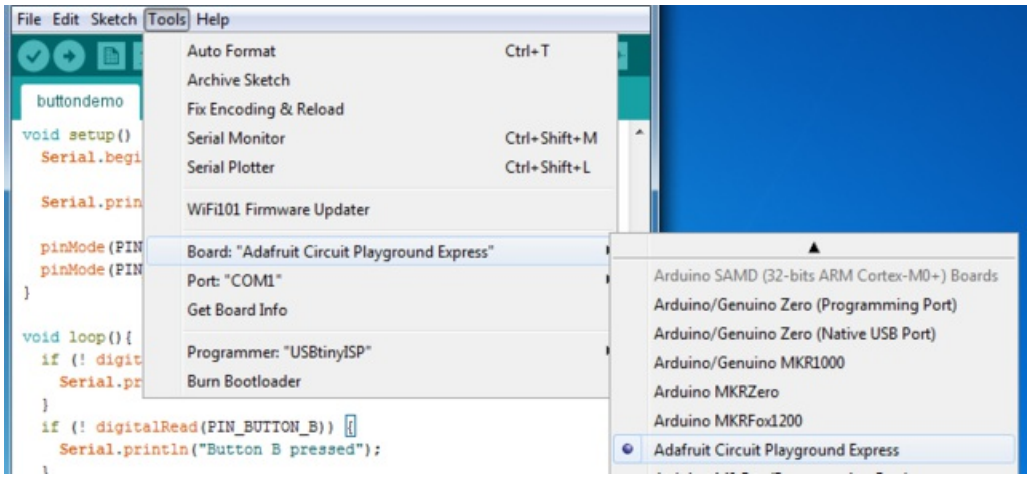
You can type **Arduino SAMD** in the top search bar, then when you see the entry, click **Install**



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

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

Select the Circuit Playground Express board:



Install Drivers (Windows 7 Only)

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

Click below to download our Driver Installer

Download Adafruit Driver Installer v2.0.0.0

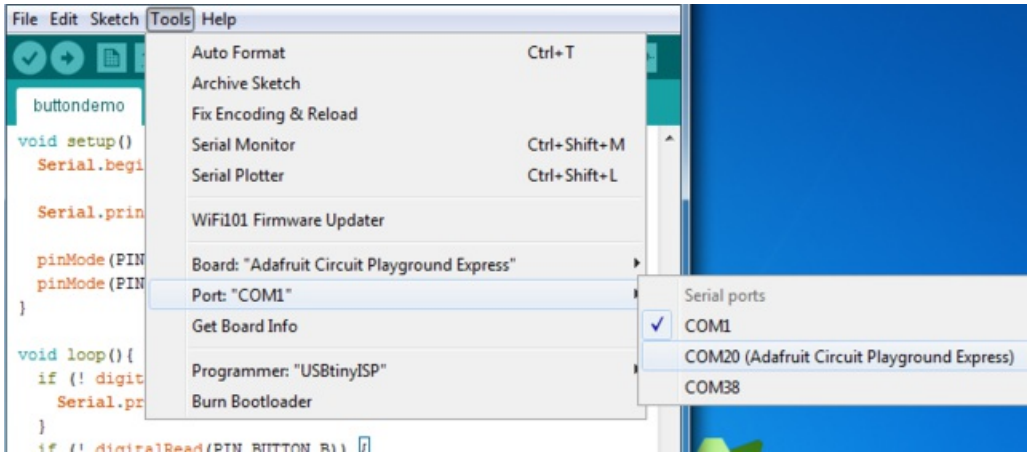
https://adafru.it/zek

Download and run the installer, more details are here

Blink

Now you can upload your first blink sketch!

Plug in the Circuit Playground Express 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 dropdown, it'll even be 'indicated' as a Circuit Playground Express board!



Now load up the Blink example

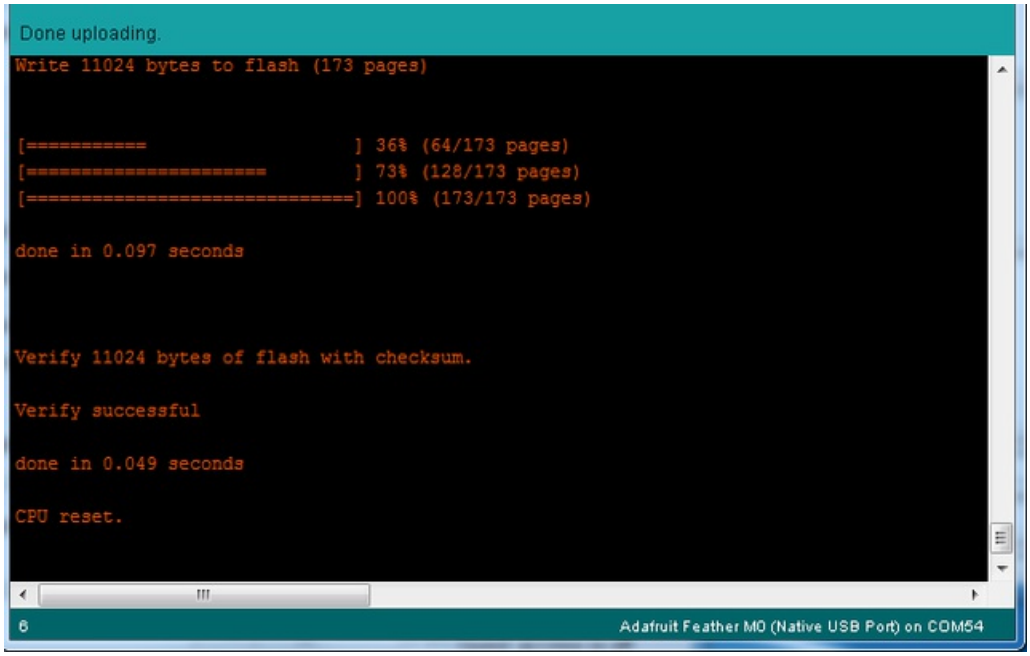
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// 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.

Sucessful 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



Compilation Issues

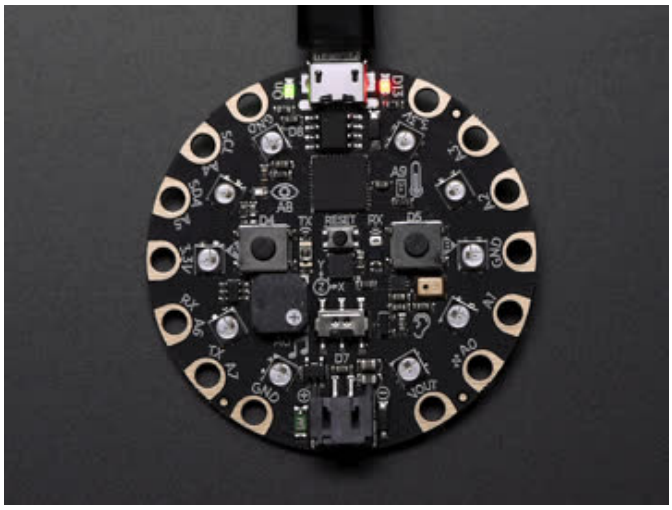
If you get an alert that looks like

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, try re-installing it or getting the latest version! Also try updating the IDE

Manually bootloading

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



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 **RESET** button **twice** (like a double-click) to get back into the bootloader.

The red LED will pulse, and all the NeoPixels will turn green so you know that its in bootloader mode.

In the Arduino IDE, **re-select the Serial Port** to the new port that has been created for the bootloader.

Then upload **Blink** - make sure that works!

Once that works, go back and re-select the 'normal' USB serial port next time you want to use the normal upload.

Ubuntu & Linux Issue Fix

Note if you're using Ubuntu 15.04 (or perhaps other more recent Linux distributions) there is an issue with the modem manager service which causes the Bluefruit LE micro to be difficult to program. If you run into errors like "device or resource busy", "bad file descriptor", or "port is busy" when attempting to program then you are hitting this issue.

The fix for this issue is to make sure Adafruit's custom udev rules are applied to your system. One of these rules is made to configure modem manager not to touch the Feather board and will fix the programming difficulty issue. Follow the steps for installing Adafruit's udev rules on this page.

Arduino Switches

The first part of interfacing with hardware is being able to manage digital inputs and outputs. With the built in Circuit Playground hardware & the Arduino library it's super easy!

This quick-start example shows how you can use one of the Circuit Playground buttons as an *input* to control another digital *output* - the built in LED

```
#include <Adafruit_CircuitPlayground.h>

void setup() {
  // Initialize the circuit playground
  CircuitPlayground.begin();
}

void loop() {
  // If the left button is pressed....
  if (CircuitPlayground.leftButton()) {
      CircuitPlayground.redLED(HIGH);  // LED on
  } else {
      CircuitPlayground.redLED(LOW);   // LED off
  }
}
```

Note that we made the code a little less elegant than necessary, the if/then could be replaced with a simple `CircuitPlayground.redLED(CircuitPlayground.leftButton())` but I wanted to make it super clear how to test the inputs.

Press Button A (the one on the left), and the onboard red LED will turn on!

Without Library Assist

If you are interested in 'lower level' interfacing to the hardware, you don't *have* to use the CircuitPlayground helper library. This code shows manually setting the button/LED pins to inputs & outputs as well as setting and reading the state of the pins:

```
#include <Adafruit_CircuitPlayground.h>

void setup() {
  pinMode(CPLAY_LEFTBUTTON, INPUT_PULLDOWN);
  pinMode(CPLAY_REDLED, OUTPUT);
}

void loop() {
  // If the left button is pressed....
  if (digitalRead(CPLAY_LEFTBUTTON)) {
      digitalWrite(CPLAY_REDLED, HIGH);  // LED on
  } else {
      digitalWrite(CPLAY_REDLED, LOW);  // LED off
  }
}
```

Note that on the M0/SAMD based CircuitPython boards, at least, you can also have internal pullups with `INPUT_PULLUP` when using external buttons, but the built in buttons require `INPUT_PULLDOWN`.

Maybe you're setting up your own external button with pullup or pulldown resistor. If you want to turn off the internal pullup/pulldown just set the `pinMode()` to plain `INPUT`

Adapting Sketches to M0

The ATSAMD21 is a very nice little chip but its fairly new as Arduino-compatible cores go. **Most** sketches & libraries will work but here's a few things we noticed!

The below note are for all M0 boards, but not all may apply (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 is the same as the output-selection register.

For the M0, you can't do this anymore! Instead, use

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

which has the benefit of being backwards compatible with AVR.

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 Core, we fixed it so that Serial goes to USB when you use a Feather M0 so it will automatically work just fine.

However, on the off chance you are using the official Arduino SAMD core not the Adafruit version (which really, we recommend you use our version because as you can see it can vary) & you want your Serial prints and reads to use the USB port, use SerialUSB instead of Serial in your sketch

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

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

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

AnalogWrite / PWM on Feather/Metro M0

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

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

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

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

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

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

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

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

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

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

- **Analog pin A5**

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

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

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

- **TX and SDA (Digital pins 1 and 20)**

analogWrite() PWM range

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

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_AR
 #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, you'll need to *double click* the button. You will see a pulsing red LED to let you know you're in bootloader mode. Once in that mode, it wont time out! Click reset again if you want to go back to launching code

Aligned Memory Access

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

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

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

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

Floating Point Conversion

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

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

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

http://forum.arduino.cc/index.php?topic=368720.0

How Much RAM Available?

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

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

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

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

Storing data in FLASH

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

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

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

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

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

UF2 Bootloader Details

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

Adafruit Express and Gemma/Trinket M0 boards feature an improved bootloader that makes it easier than ever to flash different code onto the microcontroller. This bootloader makes it easy to switch between Microsoft MakeCode, CircuitPython and Arduino.

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

The format of the file is a little special. Due to 'operating system woes' you cannot just drag a binary or hex file (trust us, we tried it, it isn't cross-platform compatible). Instead, the format of the file has extra information to help the bootloader know where the data goes. The format is called UF2 (USB Flashing Format). Microsoft MakeCode generates UF2s for flashing and CircuitPython releases are also available as UF2. You can also create your own UF2s from binary files using uf2tool, available here.

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

For more information about UF2, you can read a bunch more at the MakeCode blog, then check out the UF2 file format specification. Visit Adafruit's fork of the Microsoft UF2-samd bootloader GitHub repository for source code and releases of pre-built bootloaders.

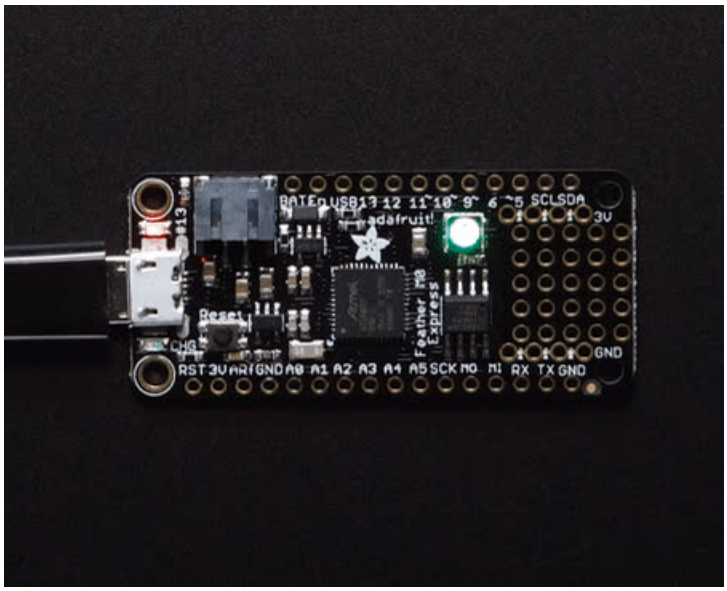
The bootloader is not needed when changing your CircuitPython code. Its only needed when upgrading the CircuitPython core or changing between CircuitPython, Arduino and Microsoft MakeCode.

Entering Bootloader Mode

The first step to loading new code onto your board is triggering the bootloader. It is easily done by double tapping the reset button. Once the bootloader is active you will see the small red LED fade in and out and a new drive will appear on your computer with a name ending in **BOOT**. For example, feathers show up as **FEATHERBOOT**, while the new CircuitPlayground shows up as **CPLAYBOOT**, Trinket M0 will show up as **TRINKETBOOT**, and Gemma M0 will show up as **GEMMABOOT**

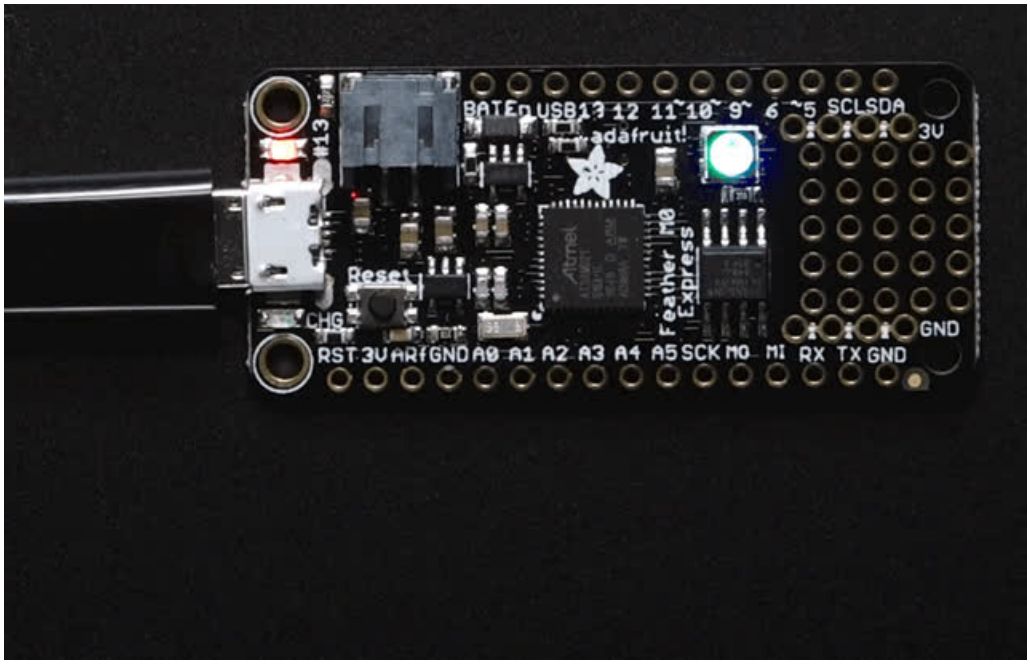
Furthermore, when the bootloader is active, it will change the color of one or more onboard neopixels to indicate the connection status, red for disconnected and green for connected. If the board is plugged in but still showing that its disconnected, try a different USB cable. Some cables only provide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

For example, here is a Feather M0 Express running a colorful Neopixel swirl. When the reset button is double clicked (about half second between each click) the NeoPixel will stay green to let you know the bootloader is active. When the reset button is clicked once, the 'user program' (NeoPixel color swirl) restarts.

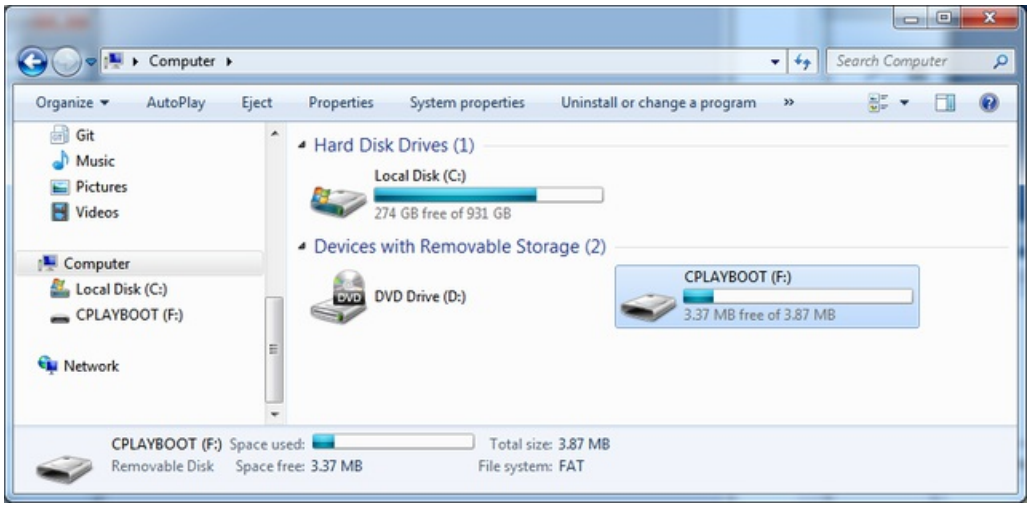


If the bootloader couldn't start, you will get a red NeoPixel LED.

That could mean that your USB cable is no good, it isn't connected to a computer, or maybe the drivers could not enumerate. Try a new USB cable first. Then try another port on your computer!

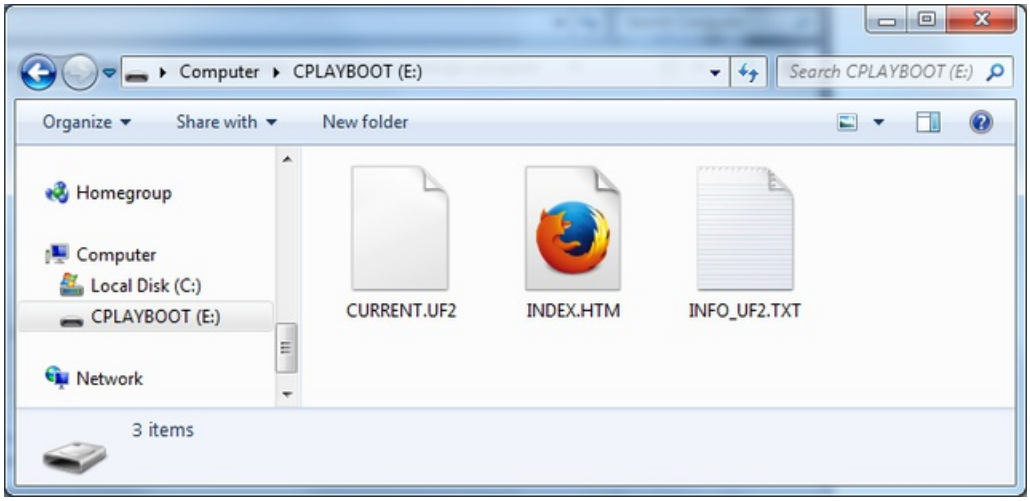


Once the bootloader is running, check your computer. You should see a USB Disk drive...



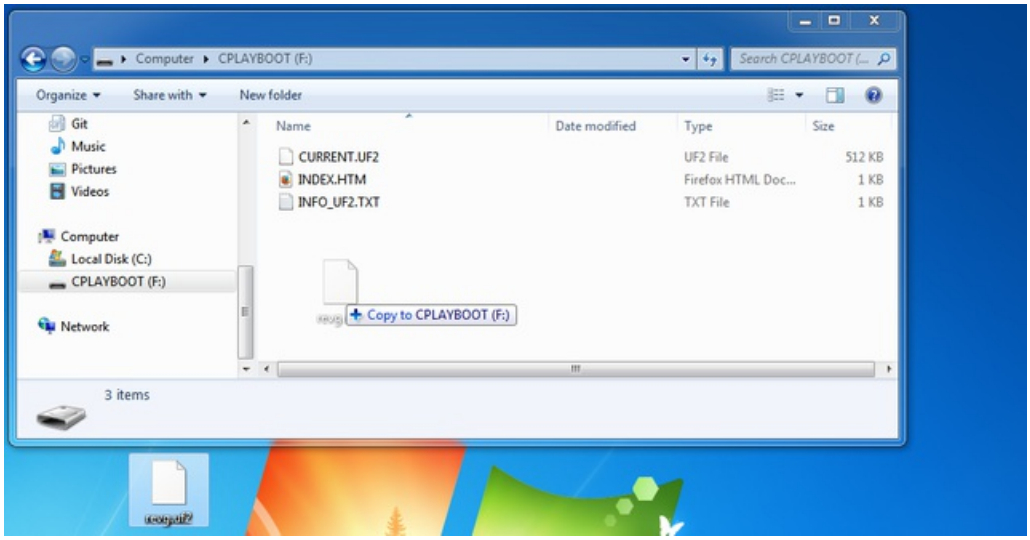
Once the bootloader is successfully connected you can open the drive and browse the virtual filesystem. This isn't the same filesystem as you use with CircuitPython or Arduino. It should have three files:

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

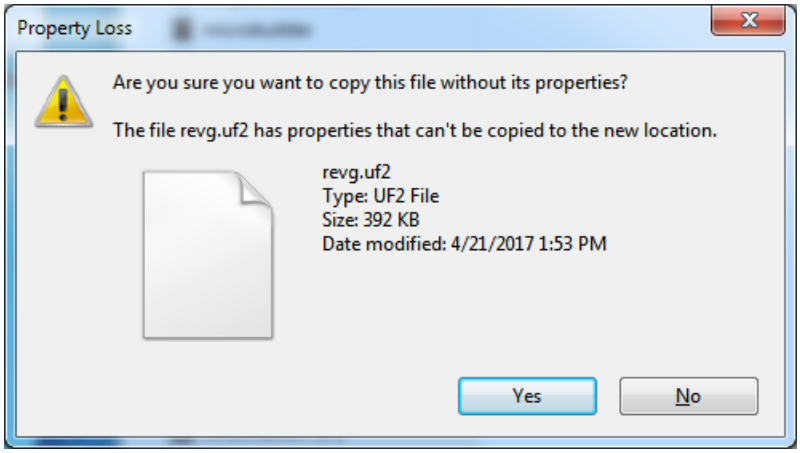


Using the Mass Storage Bootloader

To flash something new, simply drag any UF2 onto the drive. After the file is finished copying, the bootloader will automatically restart. This usually causes a warning about an unsafe eject of the drive. However, its not a problem. The bootloader knows when everything is copied successfully.



You may get an alert from the OS that the file is being copied without it's properties. You can just click **Yes**



You may also get get a complaint that the drive was ejected without warning. Don't worry about this. The drive only ejects once the bootloader has verified and completed the process of writing the new code

Using the BOSSA Bootloader

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

Windows 7 Drivers

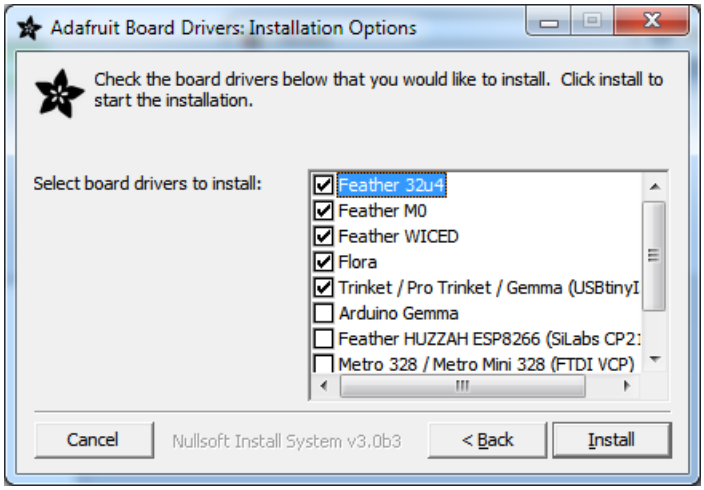
If you are running Windows 7 (or, goodness, something earlier?) You will need a Serial Port driver file. Windows 10 users do not need this so skip this step.

You can download our full driver package here:

Download Latest Adafruit Driver Installer

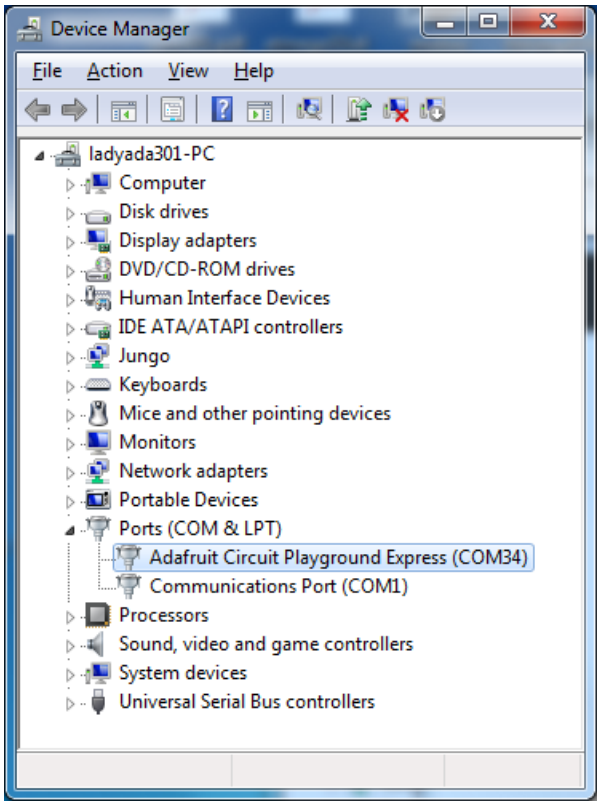
https://adafru.it/A0N

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

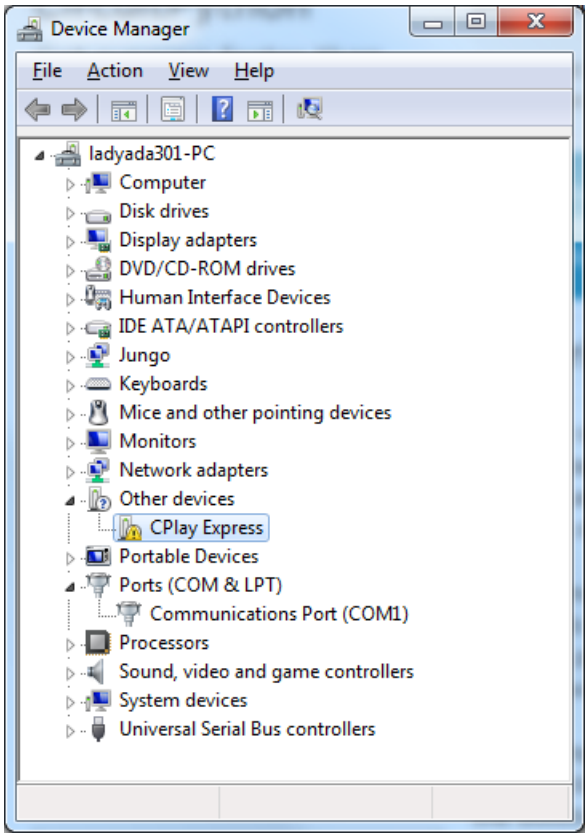


Verifying Serial Port in Device Manager

If you're running Windows, its a good idea to verify the device showed up. Open your Device Manager from the control panel and look under **Ports (COM & LPT)** for a device called **Feather M0** or **Circuit Playground** or whatever!



If you see something like this, it means you did not install the drivers. Go back and try again, then remove and re-plug the USB cable for your board



Running bossac on the command line

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

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

For example here's the command line you probably want to run:

```
bossac -e -w -v -R ~/Downloads/adafruit-circuitpython-feather_m0_express-0.9.3.bin
```

This will `-e` rase the chip, `-w` rite the given file, `-v` erify the write and `-R` eset the board. After reset, CircuitPython should be running. Express boards may cause a warning of an early eject of a USB drive but just ignore it. Nothing important was being written to the drive. A hard power-reset is also recommended after **bossac**, just in case.

```
(venv) tannewt@shallan:~/Downloads/bossac-1.7.0 $ ./bossac -e -w -v -R ~/Downloads/a
dafruit-circuitpython-feather_m0_express-0.9.3.bin
Device found on cu.usbmodem1441
Atmel SMART device 0x1001000a found
Erase flash
done in 0.658 seconds

Write 216080 bytes to flash (3377 pages)
[==============================] 100% (3377/3377 pages)
done in 1.371 seconds

Verify 216080 bytes of flash with checksum.
Verify successful
done in 0.305 seconds
CPU reset.
(venv) tannewt@shallan:~/Downloads/bossac-1.7.0 $
```

Updating the bootloader

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

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

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

For example:

```
UF2 Bootloader v1.20.0 SFHR
Model: Adafruit Feather M0
Board-ID: SAMD21G18A-Feather-v0
```

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

The latest updaters for various boards:

Circuit Playground Express v1.23 update-bootloader.uf2

https://adafru.it/yDv

Feather M0 Express v1.23 update-bootloader.uf2

https://adafru.it/yDw

Metro M0 Express v1.23 update-bootloader.uf2

https://adafru.it/yDx

Gemma M0 v1.23 update-bootloader.uf2

https://adafru.it/yDy

Trinket M0 v1.23 update-bootloader.uf2

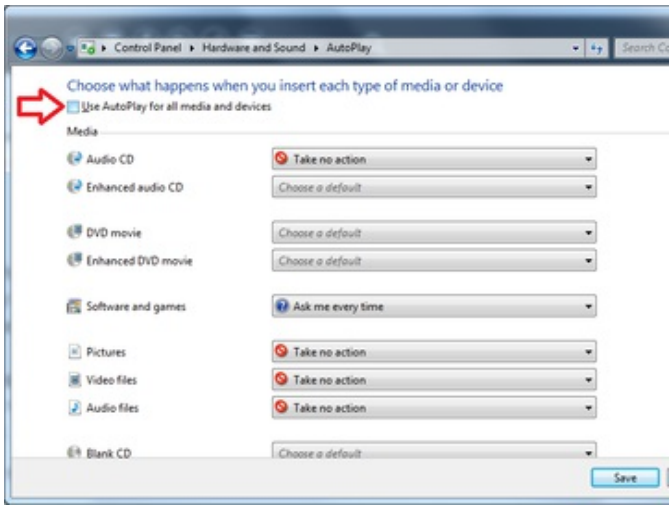
https://adafru.it/yDz

Getting Rid of Windows Pop-ups

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

Go to the Control Panel. Click on the **Hardware and Sound** header

Click on the **Autoplay** header



Uncheck the box at the top, labeled **Use Autoplay for all devices**

Making your own UF2

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

Once you have a .bin file, you simply need to run the Python conversion script over it. Here is an example from the directory with uf2conv.py:

```
uf2conv.py -c -o build-circuitplayground_express/revg.uf2 build-circuitplayground_express/revg.bin
```

This will produce a revg.uf2 file in the same directory as the source revg.bin. The uf2 can then be flashed in the same way as above.

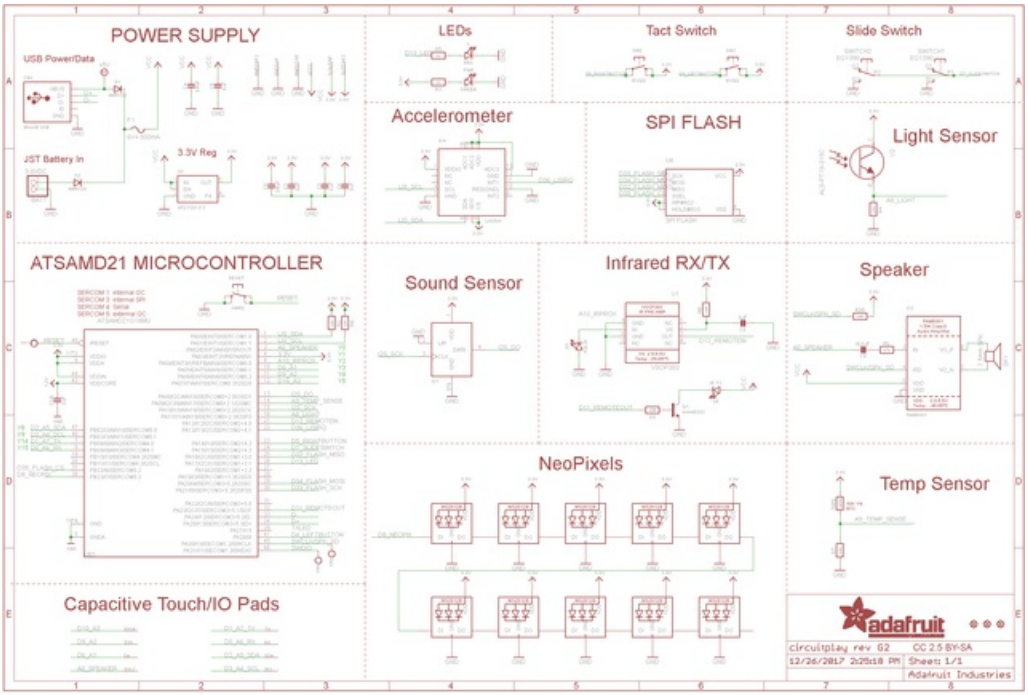
Downloads

Files:

- EagleCAD PCB files for Circuit Playground Express
- Fritzing files in Adafruit Fritzing Library
- PigHixxx Circuit Playground Express pinout diagram (PDF)
- Circuit Playground Express MakeCode Demo .uf2 (copy this to CPLAYBOOT)

Circuit Playground Express Schematic

click to embiggen



Original Arduino Demo shipped with Circuit Playground Express

https://adafru.it/AmO