



Welcome to CircuitPython!

Created by Kattni Rembor



Last updated on 2018-03-09 08:15:56 PM UTC

Guide Contents

Guide Contents	2
Overview	5
This guide will get you started with CircuitPython!	5
What is CircuitPython?	6
CircuitPython is based on Python	6
Why would I use CircuitPython?	6
Installing CircuitPython	8
Download the latest version!	8
Windows 7 Drivers	8
Start the UF2 Bootloader	9
Installing Mu Editor	11
Installing Mu for Windows or Mac OS X	11
Installing Mu for Linux	12
Using Mu	12
PyCharm and CircuitPython	15
Creating and Editing Code	21
Creating Code	21
Editing Code	23
Your code changes are run as soon as the file is done saving.	23
1. Use an editor that writes out the file completely when you save it.	23
2. Eject or Sync the Drive After Writing	24
Oh No I Did Something Wrong and Now The CIRCUITPY Drive Doesn't Show Up!!!	24
Back to Editing Code...	24
Exploring Your First CircuitPython Program	24
Imports & Libraries	25
Setting Up The LED	25
Loop-de-loops	25
More Changes	26
Naming Your Program File	26
Connecting to the Serial Console	27
Are you using Mu?	27
Using Something Else?	28
Interacting with the Serial Console	29
The REPL	32
Returning to the serial console	35
CircuitPython Hardware	37
Trinket M0	37
Gemma M0	38
Circuit Playground Express	38
Feather M0 Express	39
Metro M0 Express	39
What's Next?	40
CircuitPython Libraries	41

Installing the CircuitPython Library Bundle	41
Express Boards	42
Non-Express Boards	43
Example: ImportError Due to Missing Library	43
Library Install on Non-Express Boards	44
Updating CircuitPython Libraries	44
Welcome to the Community!	45
Adafruit Discord	45
Adafruit Forums	46
Adafruit Github	47
ReadTheDocs	48
CircuitPython for ESP8266	49
Installing CircuitPython on the ESP8266	49
Download esptool	49
Download Latest CircuitPython Firmware	49
Get ESP8266 Ready For Bootloading	50
Erase ESP8266	50
Program ESP8266	50
Upload Libraries & Files Using Ampy!	51
Other Stuff To Know!	51
Advanced Serial Console on Windows	53
Windows 7 Driver	53
What's the COM?	53
Install Putty	54
Advanced Serial Console on Mac and Linux	56
What's the Port?	56
Connect with screen	58
Permissions on Linux	59
Non-UF2 Installation	61
Flashing with Bossac - For Non-Express Feather M0's & Arduino Zero	61
Command-Line ahoy!	61
Download Latest CircuitPython Firmware	61
Download BOSSA	61
Test bossac	61
Get Into the Bootloader	62
Run bossac command	62
Troubleshooting	64
CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present	64
You may have a different board.	64
MakeCode	64
Windows 10	64
Windows 7	64
CircuitPython RGB Status Light	65
CIRCUITPY Drive Issues	65
For the Circuit Playground Express, Feather M0 Express, and Metro M0 Express:	66
For Non-Express Boards with a UF2 bootloader (Gemma M0, Trinket M0):	66
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	6767
Delete something!	67
Use tabs	67
Mac OSX loves to add extra files.	67
Prevent & Remove Mac OSX Hidden Files	67
Copy Files on Mac OSX Without Creating Hidden Files	68
Other Mac OSX Space-Saving Tips	68

Overview



So, you've got this new **CircuitPython compatible board**. You plugged it in. Maybe it showed up as a disk drive called CIRCUITPY. Maybe it didn't! Either way, you need to know where to go from here. Well, we've got you covered!

This guide will get you started with CircuitPython!

There are many amazing things about your new board. One of them is the ability to run CircuitPython. You may have seen that name on the Adafruit site somewhere. Not sure what it is? We can help!

"But I've never coded in my life. There's no way I do it!" You absolutely can! CircuitPython is designed to help you learn from the ground up. If you're new to everything, this is the place to start!

This guide will walk you through how to get started with CircuitPython. You'll learn how to install CircuitPython, get updated to the newest version of CircuitPython, how to setup a serial connection, and how to edit the files.

Welcome to CircuitPython!

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!

Installing CircuitPython

Some of the CircuitPython compatible boards come with CircuitPython installed. Others are *CircuitPython-ready*, but need to have it installed. As well, you may want to update the version of CircuitPython already installed on your board. The steps are the same for installing and updating. Here we will cover how to install or update CircuitPython on your board.

You only have to install CircuitPython ONCE, after that you are free to code all you like without going through this process again until you want to upgrade!

Download the latest version!










The first thing you'll want to do is download the most recent version of CircuitPython.

Click here to see the latest CircuitPython release

<https://adafru.it/v1F>

Scroll down to the list of CircuitPython files, and choose the file appropriate to your board. Each file includes the name of the board it's compatible with. Download the file for your board.

Downloads

 adafruit-circuitpython-arduino_zero-2.1.0.bin	180 KB
 adafruit-circuitpython-circuitplayground_express-2.1.0.uf2	442 KB
 adafruit-circuitpython-feather_huzzah-2.1.0.bin	581 KB
 adafruit-circuitpython-feather_m0_adalogger-2.1.0.bin	180 KB
 adafruit-circuitpython-feather_m0_basic-2.1.0.bin	180 KB
 adafruit-circuitpython-feather_m0_express-2.1.0.uf2	414 KB
 adafruit-circuitpython-gemma_m0-2.1.0.uf2	361 KB
 adafruit-circuitpython-metro_m0_express-2.1.0.uf2	415 KB
 adafruit-circuitpython-trinket_m0-2.1.0.uf2	361 KB

Next, you'll want to plug in your board using a known-good USB data cable. Make sure the USB cable is a data cable! There are some that work only for charging and can lead to a lot of frustration.

Windows 7 Drivers

If you're using Windows 7, you need to install a driver before plugging in your board.

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

Download Adafruit Windows 7 Driver Installer

Start the UF2 Bootloader

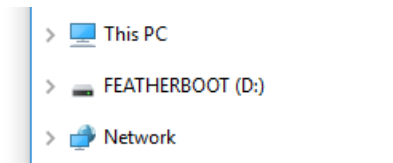
Nearly all CircuitPython boards ship with a bootloader called UF2 (USB Flasher version 2) that makes installing and updating CircuitPython a quick and easy process. The bootloader is the mode your board needs to be in for the CircuitPython .uf2 file you downloaded to work. If the file you downloaded that matches the board name ends in **uf2** then you want to continue with this section. [However, if the file ends in .bin then you have to do a more complex installation - go to this page for how to do that.](#)

Find the reset button on your board. It's a small, black button, and on most of the boards, it will be the only button available. (On Circuit Playground Express, it's the smaller button located in the center of the board.)



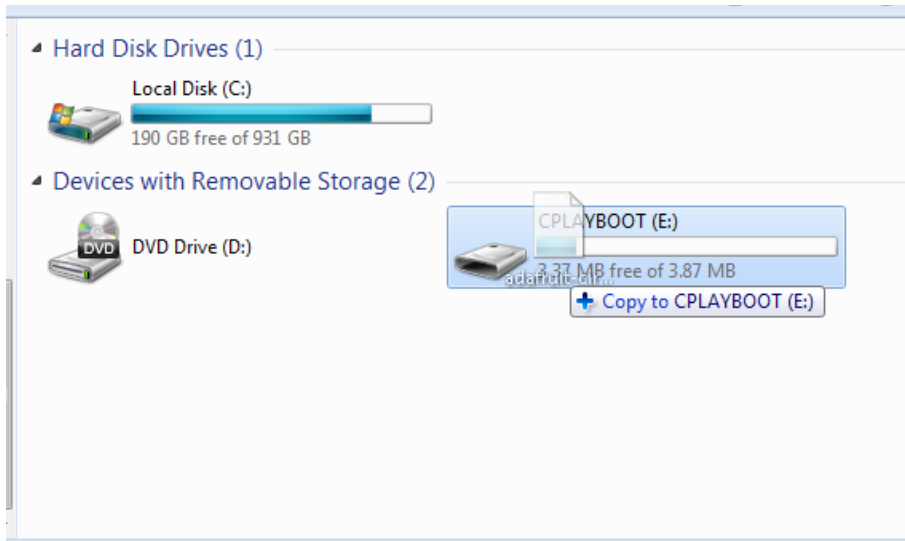
Tap this button twice to enter the bootloader. If it doesn't work on the first try, don't be discouraged. The rhythm of the taps needs to be correct and sometimes it takes a few tries. If you have a Circuit Playground Express, and it's fresh-out-of-the-bag try pressing the button once.

Once successful, the RGB LED on the board will flash red and then stay green. A new drive will show up on your computer. The drive will be called **boardnameBOOT** where **boardname** is a reference to your specific board. For example, a Feather will have **FEATHERBOOT** and a Trinket will have **TRINKETBOOT** etc. Going forward we'll just call the boot drive **BOOT**.

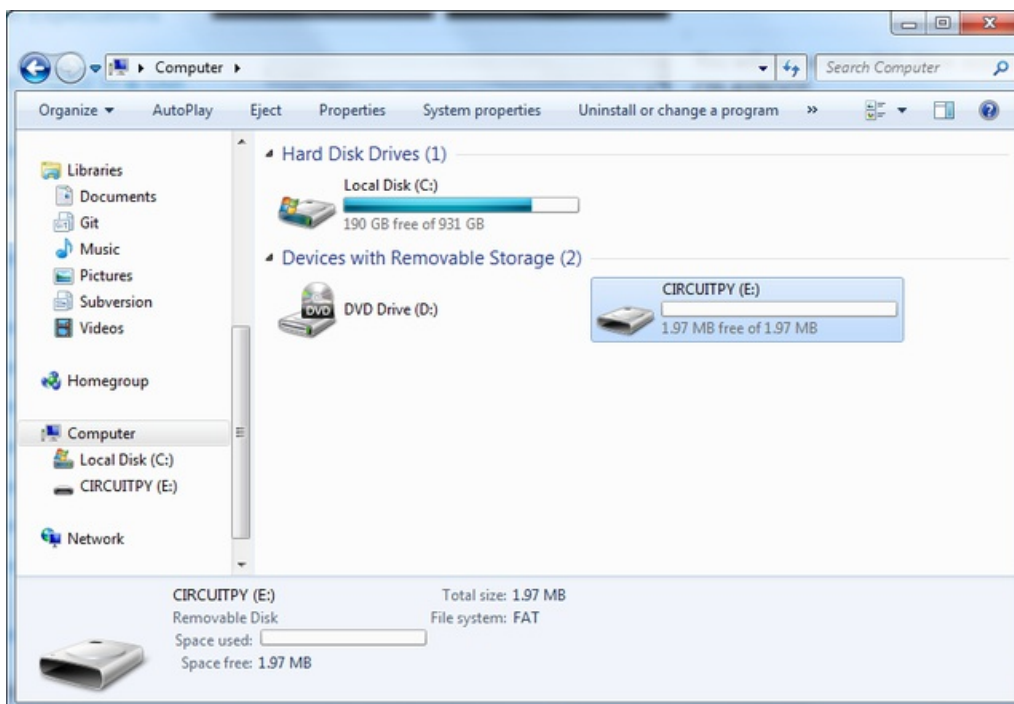


The board is now in bootloader mode! This is what we need to install or update CircuitPython.

Now find the file you downloaded. Drag that file to the **BOOT** drive on your computer.



The lights should flash again, **BOOT** will disappear and a new drive will show up on your computer called **CIRCUITPY**.



Congratulations! You've successfully installed or updated 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/AL7>

Download the latest Mu for Mac OS X

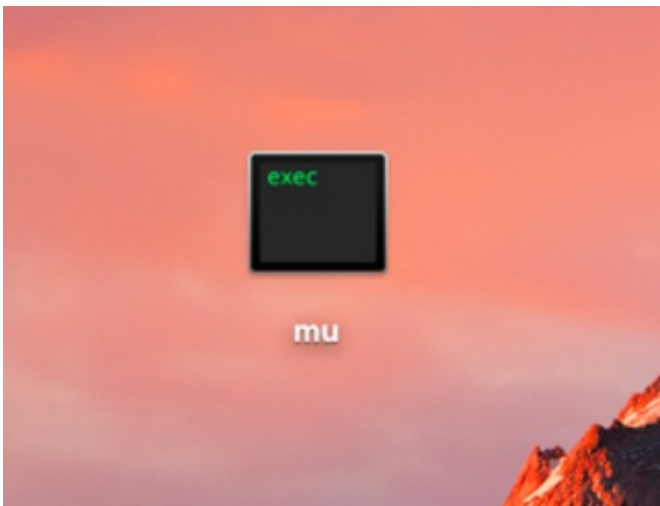
<https://adafru.it/ADx>



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 10.11 (El Capitan) 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!

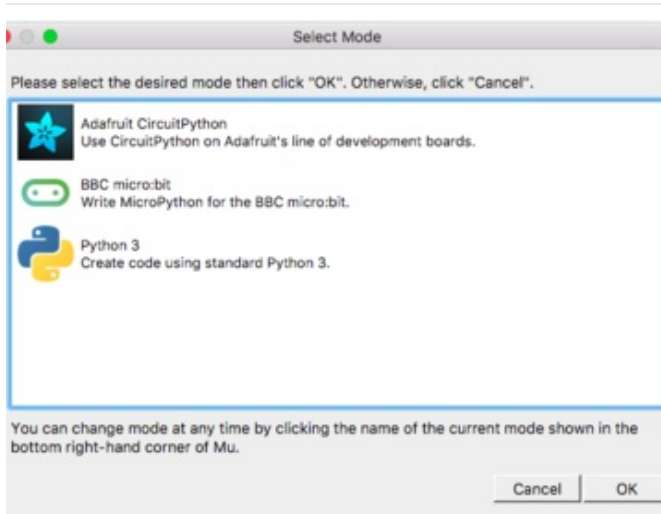


Installing Mu for Linux

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....



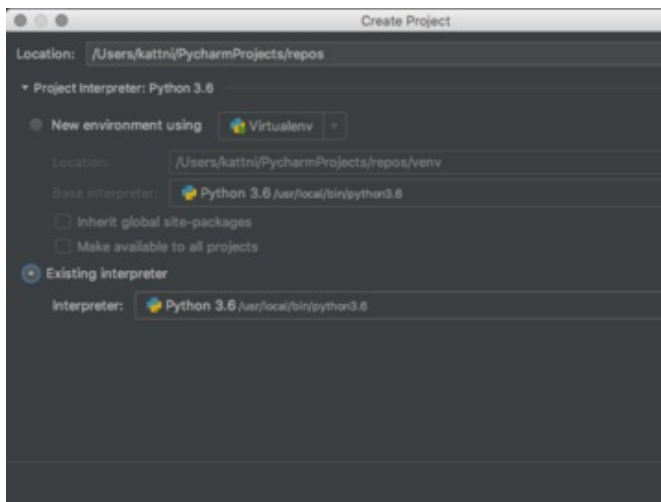
PyCharm and CircuitPython

[PyCharm](#) is a full featured Python editor including super helpful things like code completion and error highlighting. It's available for free in a Community Edition.

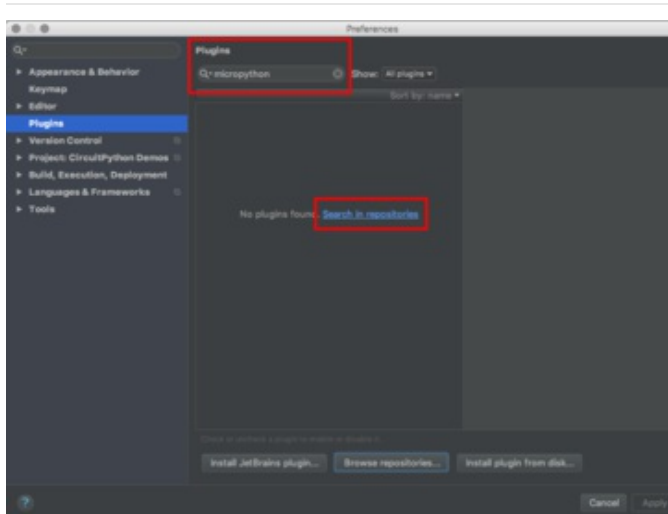


Recently, they added a [MicroPython plugin](#) that enables use of the REPL from within the editor. CircuitPython isn't officially supported, however we have some steps to make it work!

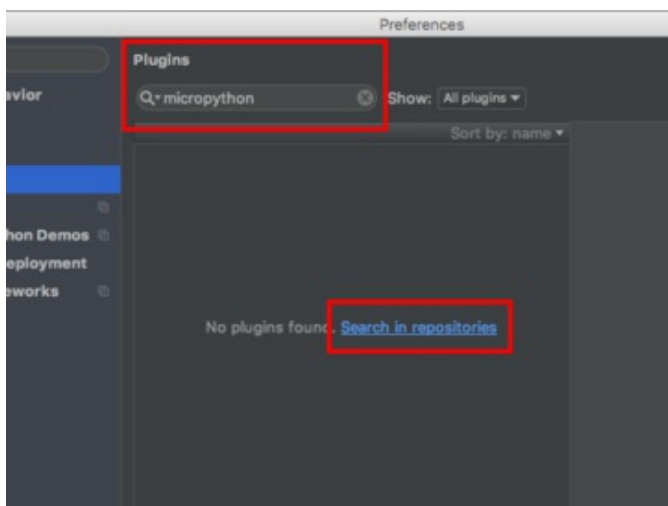
[Download](#) and install PyCharm on your computer. Then, plug in your board and follow the steps below!

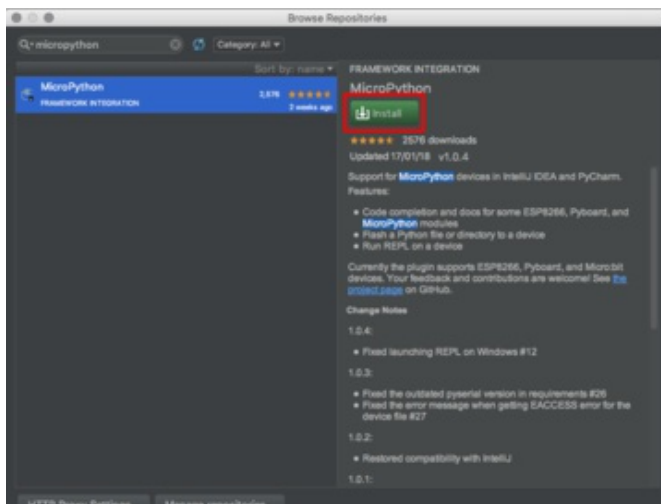


Create a new project or open an existing project.

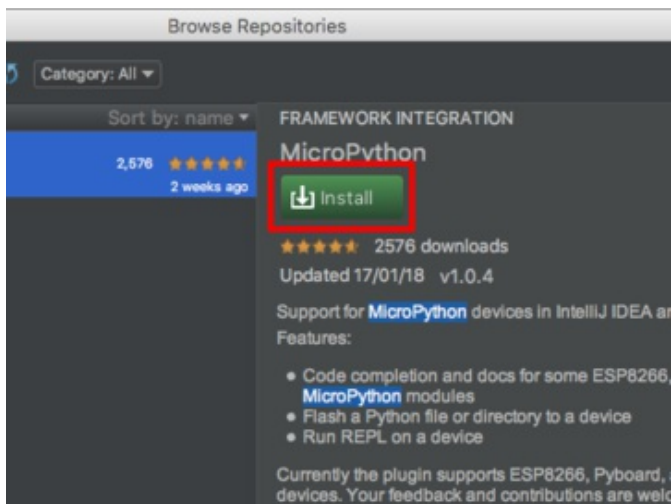


Open PyCharm Preferences/Settings. Click on **Plugins** and search for "micropython". Click on **Search in repositories**.

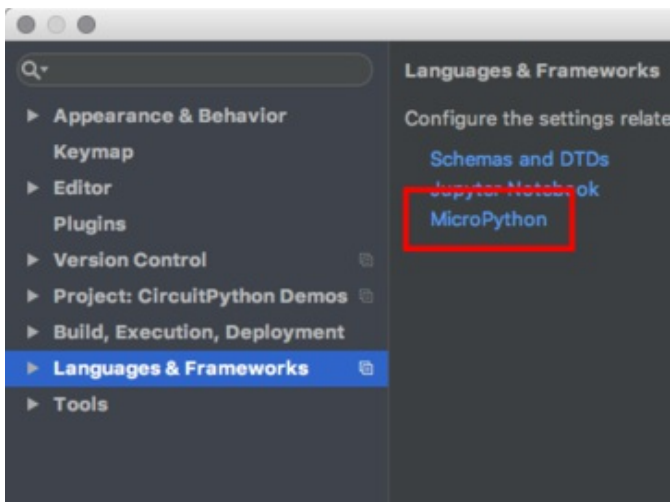




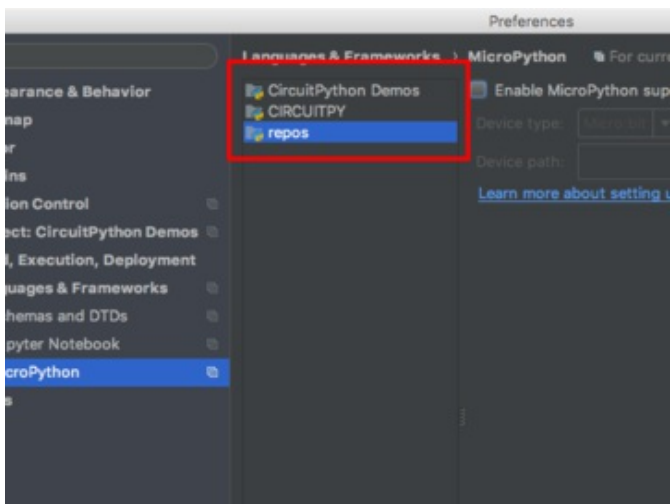
Click Install.



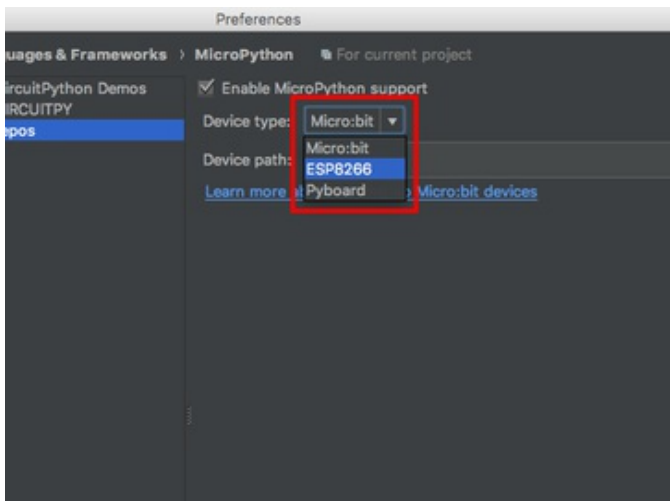
Once it's finished installing, click **Restart PyCharm**.



Once restarted, open Preferences/Settings. Click on **Languages & Frameworks** and choose **MicroPython**.

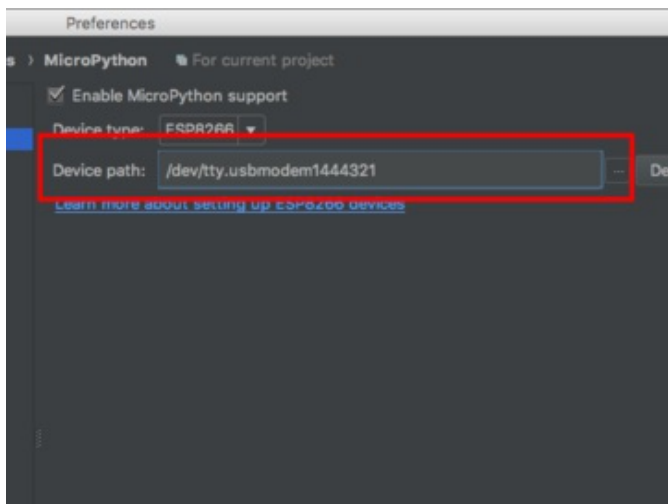


Choose your project directory from the list.



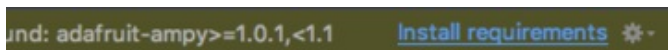
Choose **ESP8266** from the list of Device Names.

For now, you'll use this option regardless of what board you are using.



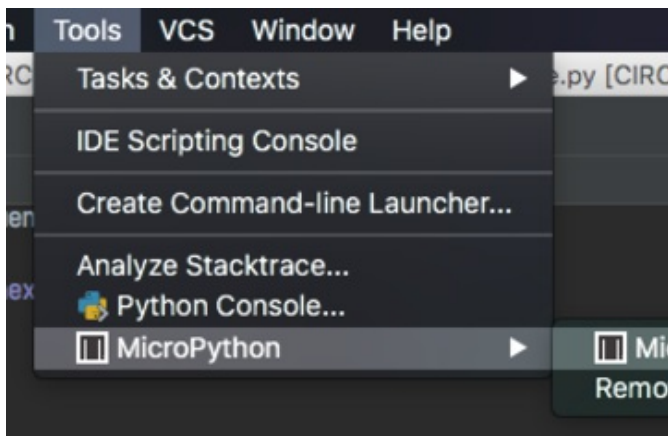
You'll need to manually add your **Device Path**. This is the path to your serial connection. Once entered, click **Ok**.

If you need help finding your device's serial connection, see [Advanced Serial Console on Windows \(https://adafruit.it/AAH\)](https://adafruit.it/AAH) and [Advanced Serial Console on Mac and Linux \(https://adafruit.it/AAI\)](https://adafruit.it/AAI).

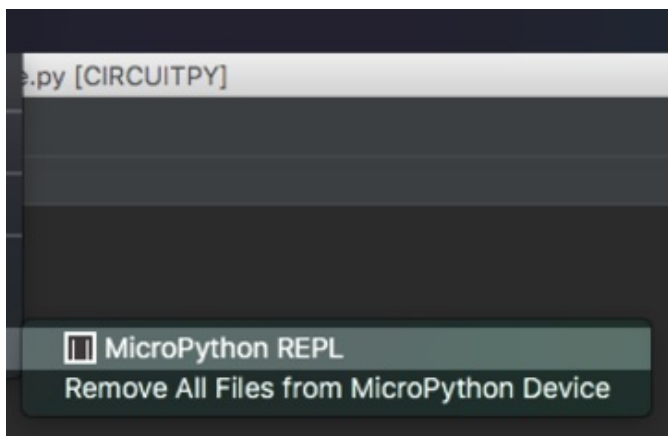


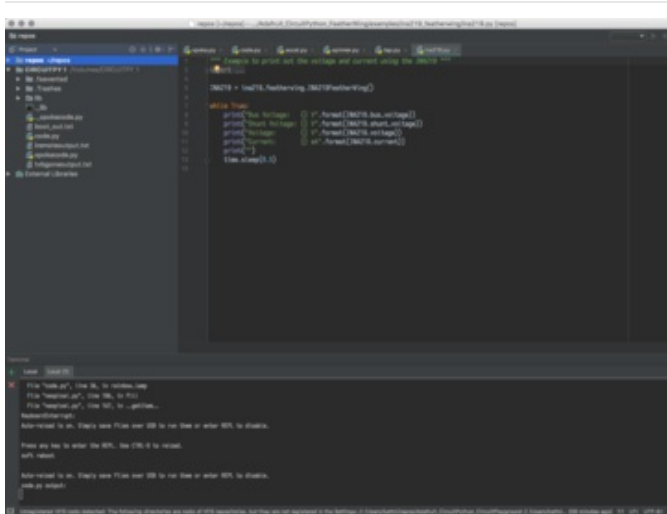
Now, open any Python file from the project directory you chose.

A message may pop up stating, "Packages required for ESP8266 support not found," followed by a list of packages. Click **Install Requirements** on this message to install the required packages.



Now, you can click on the **Tools** menu and you should find a **MicroPython** menu. Highlight this, and you'll find **MicroPython REPL** available.





The REPL will open at the bottom of the PyCharm window. Now you can begin coding!

If you want to use the built in REPL with a different board, you'll need to go in and change the Device Path to the path that matches the board you'd like to use.

To change the Device Path, follow the steps above starting with manually adding the Device Path.

The PyCharm REPL will not work with multiple boards at the same time. Going through the steps above on multiple projects did not result in consistently being able to open multiple REPL connections at the same time.

The "Remove All Files from MicroPython Device" and "Flash Project" features do not work, but could corrupt your board. Do not try to use these functions. Remember, CircuitPython isn't officially supported and the steps above are a workaround.

Creating and Editing Code

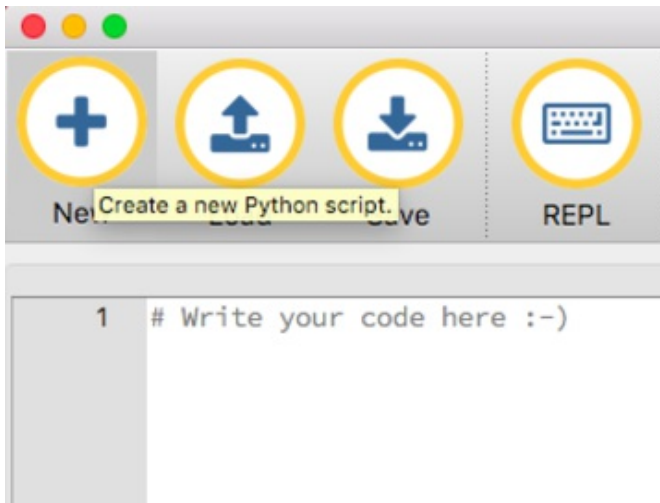
One of the best things about CircuitPython is how simple it is to get code up and running. In this section, we're going to cover how to create and edit your first CircuitPython program.

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



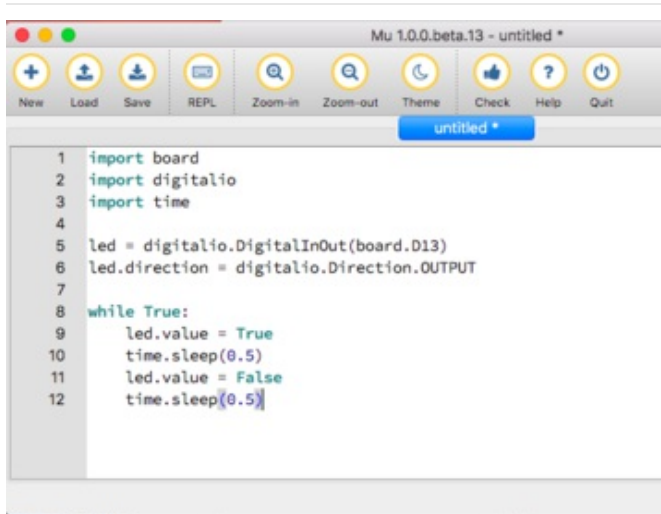
Open your editor, and create a new file. If you are using Mu, click the **New** button in the top left

Copy and paste the following code into your editor:

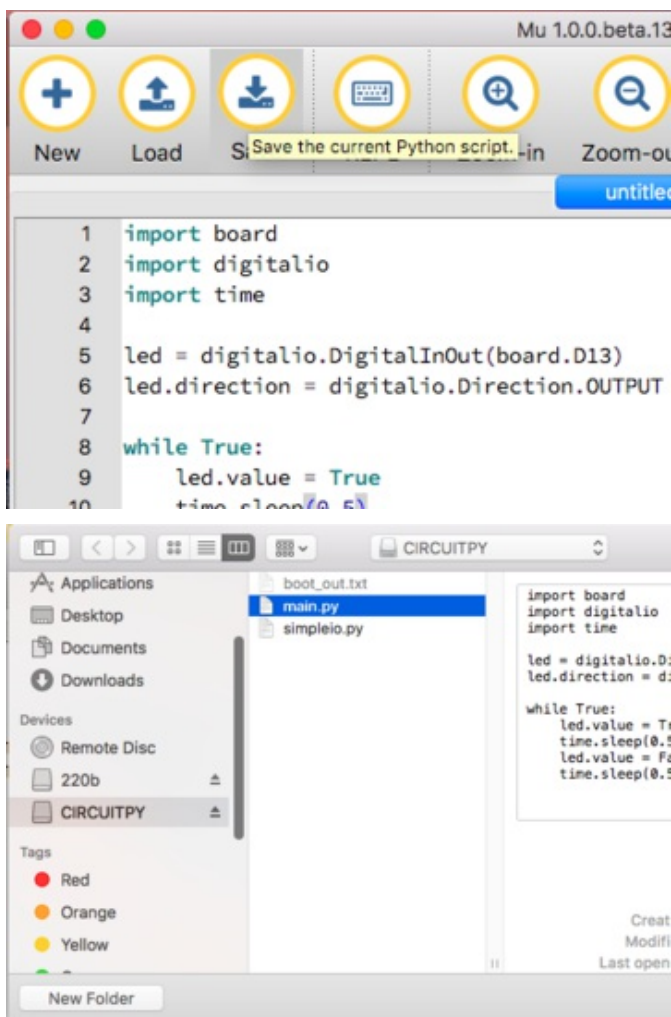
```
import board
import digitalio
import time

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.

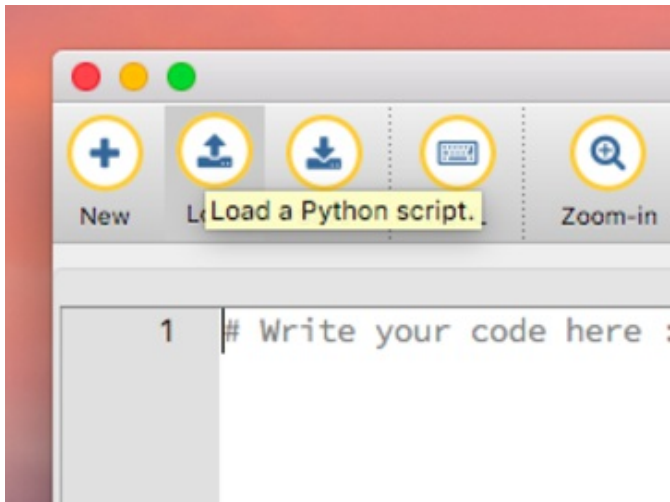


Save this file as **code.py** on your CIRCUIPTY drive.

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 out changes immediately
- **nano** (on Linux) does not force out changes
- **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 tell 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 anything 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` lets 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?

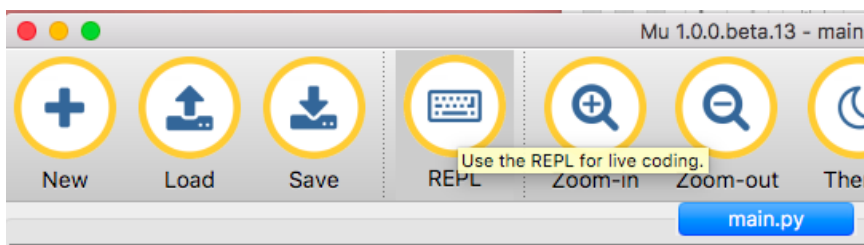
If so, good news! The serial console is **built into Mu** and will **autodetect your board** making using the REPL *really really easy*.

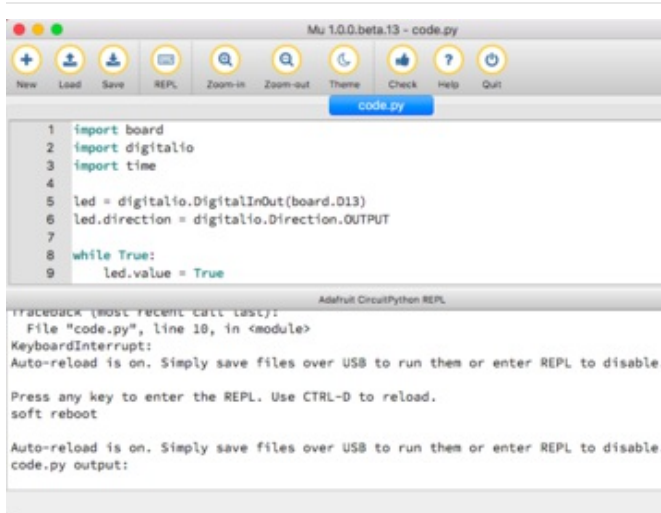
Please note that Mu does yet not work with nRF52 or ESP8266-based CircuitPython boards, skip down to the next section for details on using a terminal program.



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:

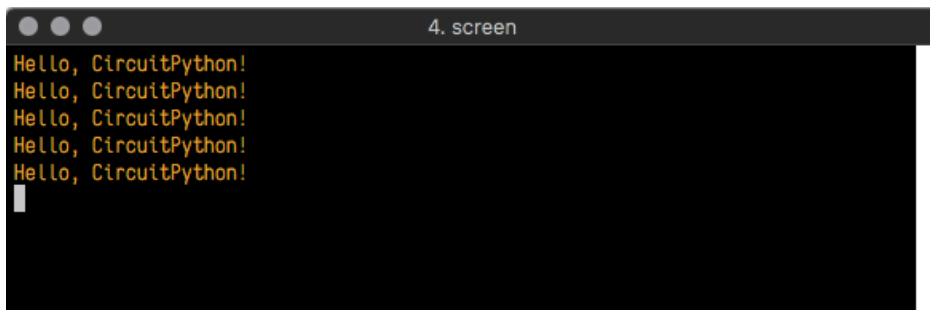
```
import board
import digitalio
import time

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

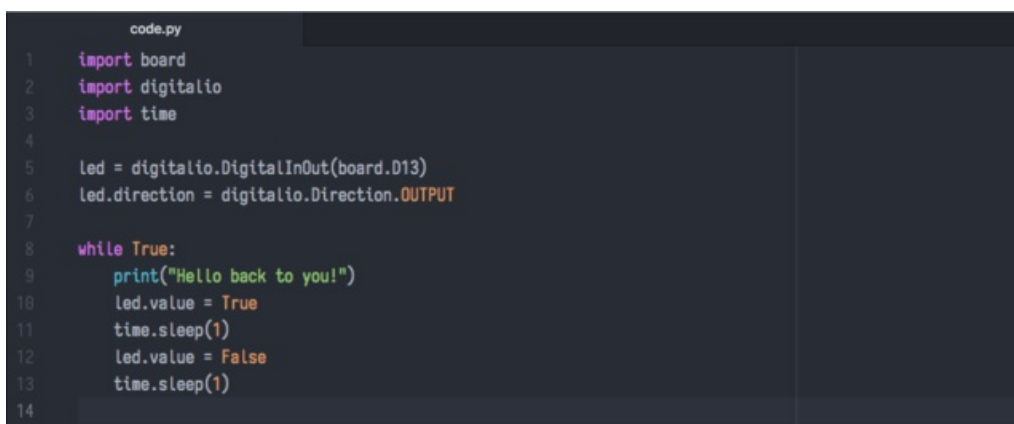
while True:
    print("Hello, CircuitPython!")
    led.value = True
    time.sleep(1)
    led.value = False
    time.sleep(1)
```

Save your file.

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

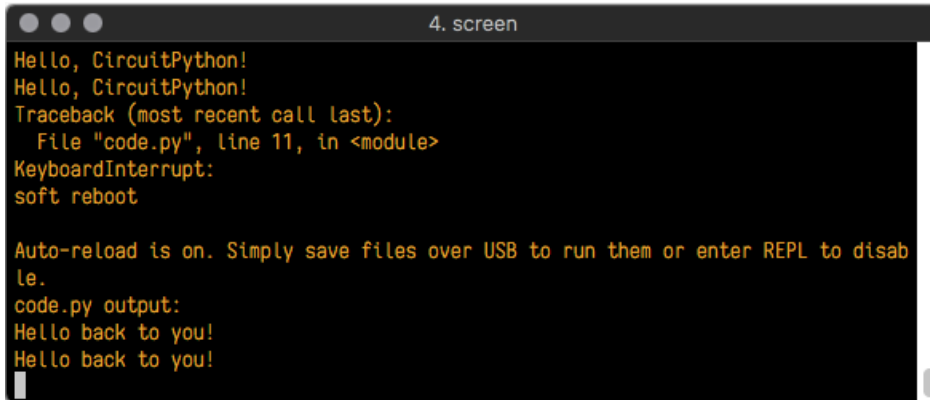


Excellent! Our print statement is showing up in our console! Try changing the printed text to something else.



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 disable.
code.py output:
Hello back to you!
Hello back to you!
```

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

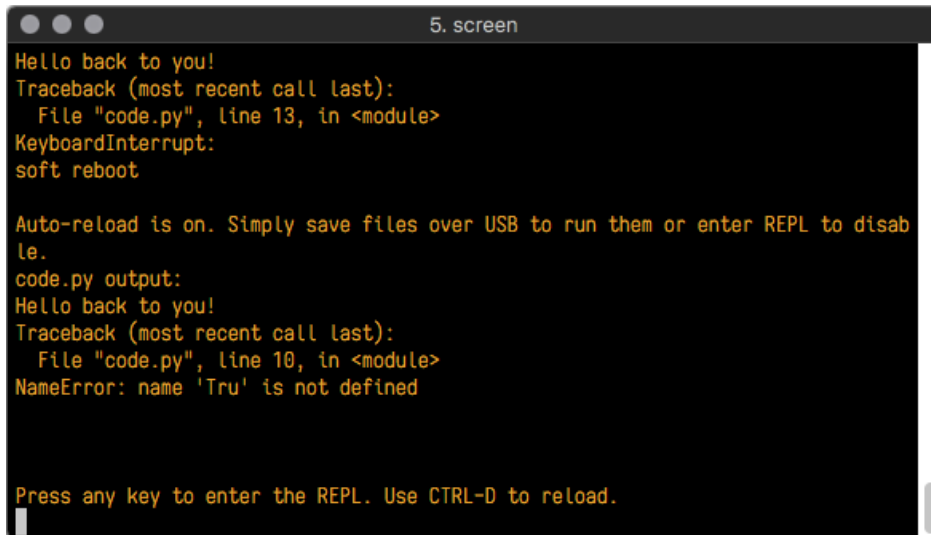
Delete the **e** at the end of **True** from the line **led.value = True** so that it says **led.value = Tru**



```
code.py
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.D13)
6 led.direction = digitalio.Direction.OUTPUT
7
8 while True:
9     print("Hello back to you!")
10    led.value = Tru
11    time.sleep(1)
12    led.value = False
13    time.sleep(1)
14
```

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!



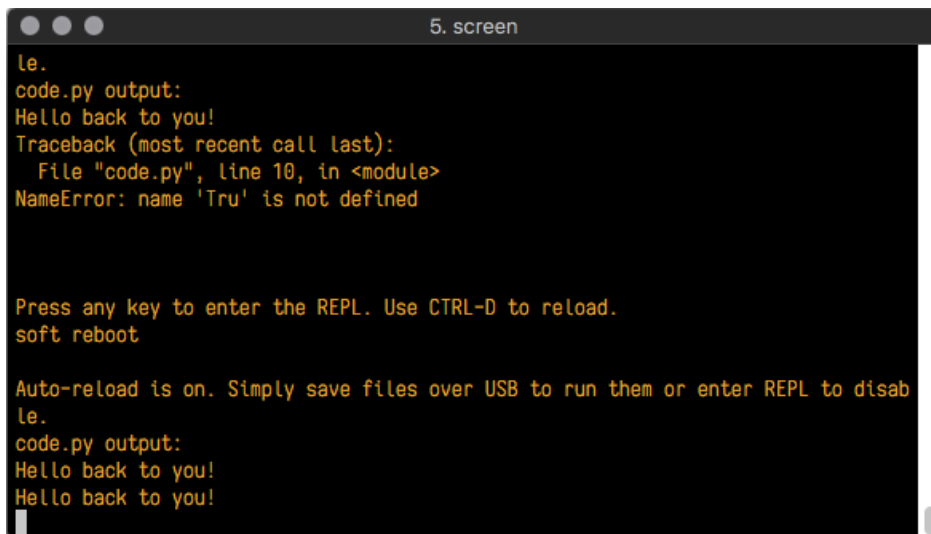
```
5. screen
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 13, in <module>
KeyboardInterrupt:
soft reboot

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

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

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

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



```
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 disable.
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!

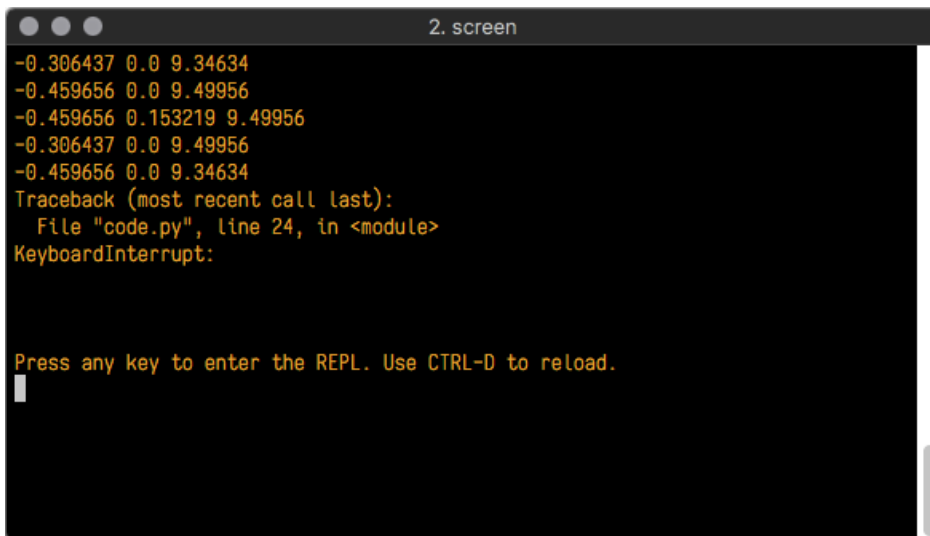
The REPL

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

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.

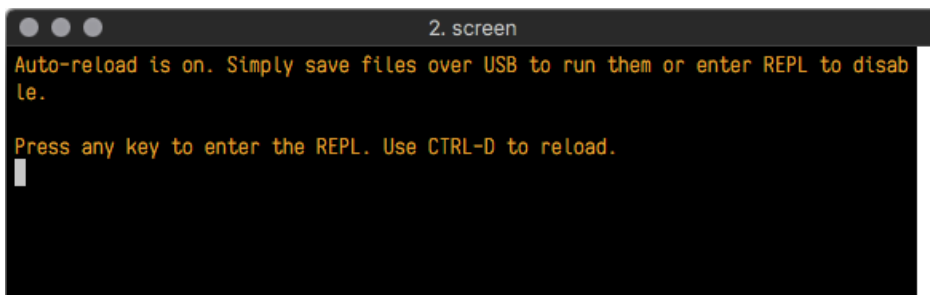
The **Traceback (most recent call last)** is telling you the last thing your board was doing before you pressed Ctrl + C and interrupted it. The **KeyboardInterrupt** is you pressing Ctrl + C. This information can be handy when troubleshooting, but for now, don't worry about it. Just note that it is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

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!



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.

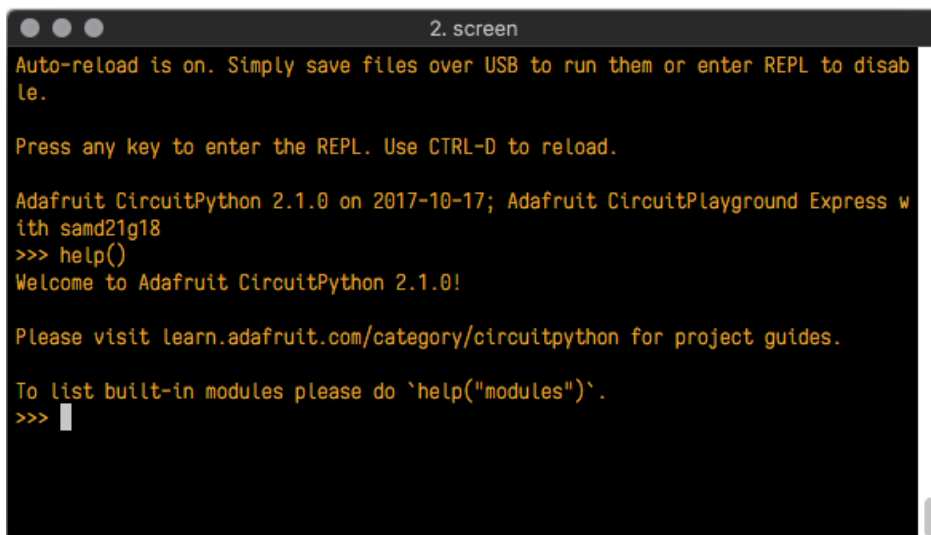
```
>>>
```

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 samd21g18
>>> help()
```

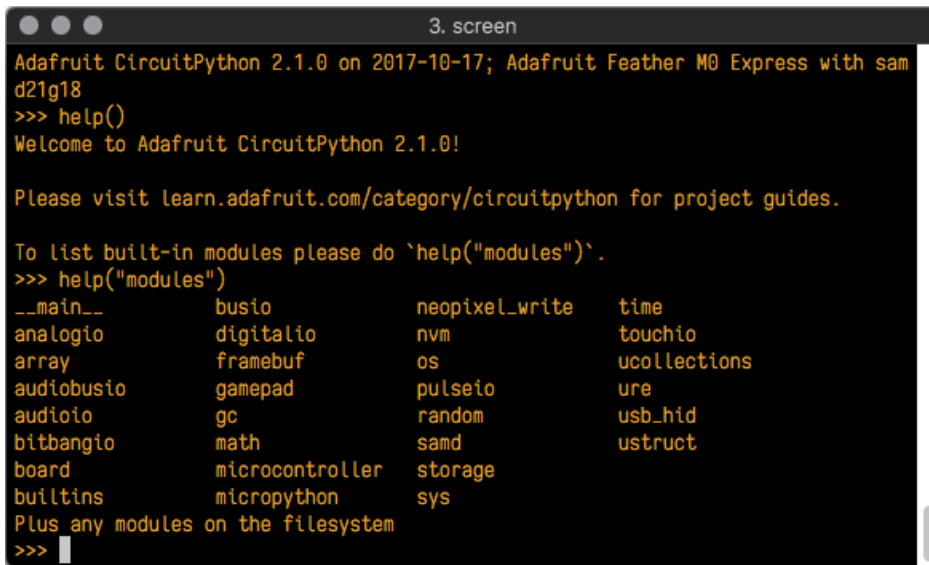
Then press enter. You should then see a message.



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.



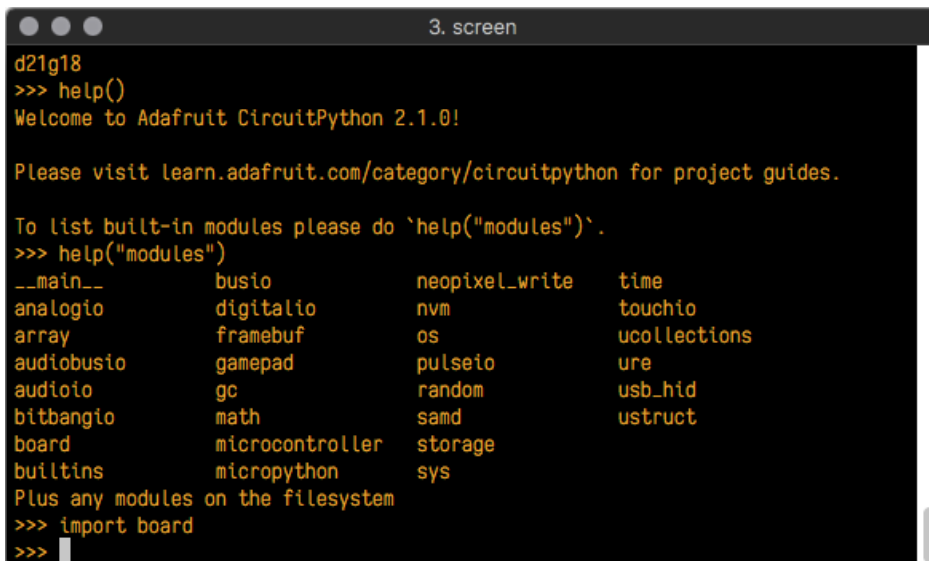
```
3. screen
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Feather M0 Express with sam
d21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

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         framebuffer    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         framebuffer    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         framebuffer    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.

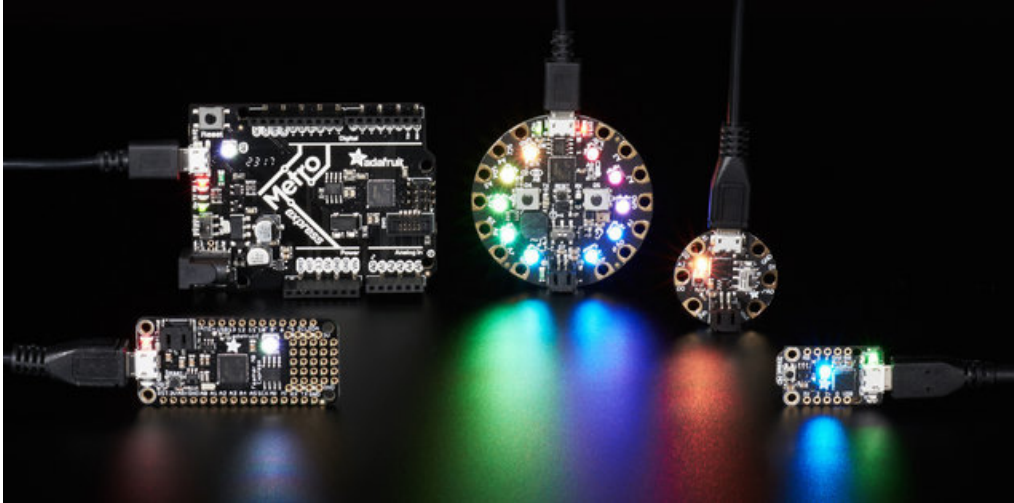
Try typing more into the REPL to see what happens!

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 Hardware



Now it's time to do something great with what you've learned! Every CircuitPython board is perfect for projects. However, each one excels in different areas. We're going to give you some details about each board, and highlight Learn guides where each one is used. You can try these out or get ideas for your own project!

Trinket M0

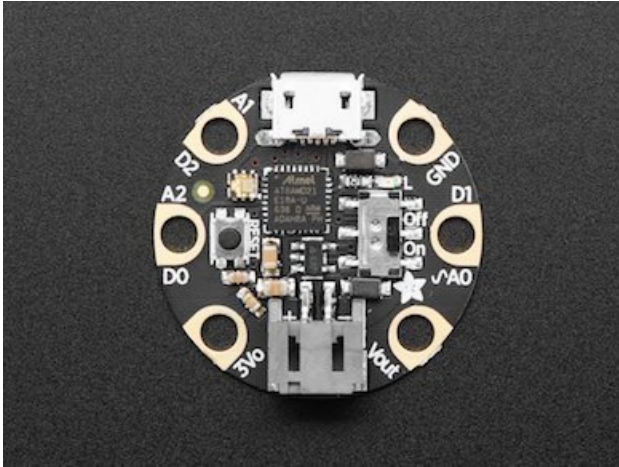


The [Adafruit Trinket M0](#) is the smallest CircuitPython board we carry. But don't let that fool you! It's a tiny board with a lot of power. We wanted to design a microcontroller board that was small enough to fit into any project, and low cost enough to use without hesitation. Planning to test a proof of concept and need a CircuitPython board to throw in? Not ready to disassemble the project you worked so hard to design to extract the board you used last time? Trinket M0 has you covered. It's the lowest cost CircuitPython board available but it easily holds its own with the bigger boards!

Trinket M0 ships with CircuitPython and comes with demo code already on the board. You can open and edit the main.py file found on the CIRCUITPY drive to get started, or create your own! The [Trinket M0 guide](#) gives you everything you need to know about your board. Check out the [CircuitPython](#) section to find a huge list of examples to try.

You can use Trinket M0 to make the [Chilled Drinkibot](#) which uses the Trinket to control thermoelectric cooling of a beverage. Or build a spooky Halloween project that turns your candy bucket into a [Screaming Cauldron!](#)

Gemma M0

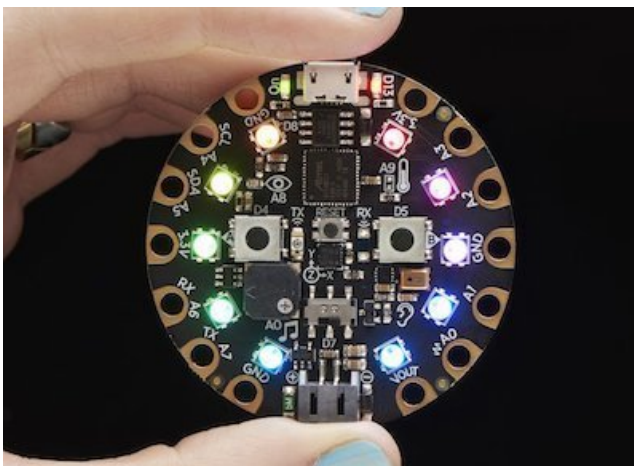


The [Adafruit Gemma M0](#) is a tiny CircuitPython board with just enough built-in to build many simple projects. It's designed to be worked into your wearable projects, with big holes around the outside for sewing (and they're alligator clip friendly too!). Gemma M0 will super-charge your wearables and is easier to use than ever. There are capacitive touch pads, an on-off switch and an RGB DotStar LED built right into the board so there's plenty you can do without adding a thing. Add conductive thread and LEDs and you'll have a blinky wearable in no time!

Like Trinket, Gemma M0 ships with CircuitPython and has demo code already on the board. You can open and edit the main.py file on the CIRCUITPY drive, or create your own! The [Gemma M0 guide](#) shows you all the info about your board, and has a great list of [CircuitPython examples](#) to try out.

Use Gemma M0 to create a pair of [Clockwork Goggles](#) with fun light patterns on NeoPixel rings. Or accessorise with this 3D printed [Sheikah Pendant](#) to add a bit of light to your next costume!

Circuit Playground Express



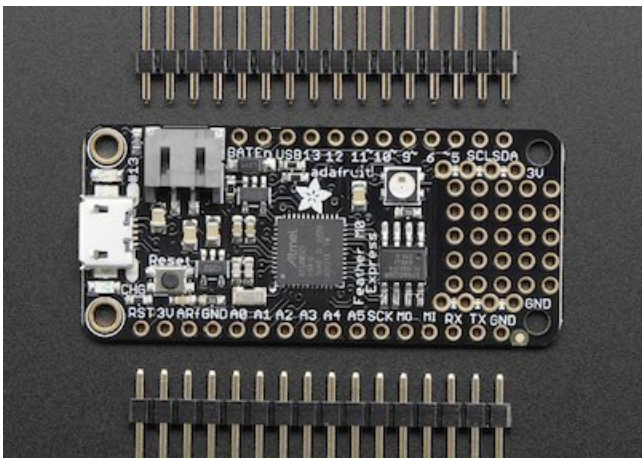
The [Adafruit Circuit Playground Express](#) is the next step towards a perfect introduction to electronics and programming. It's packed full of sensors, LEDs, buttons and switches, and it's super easy to get started with! This board is super versatile. Whether you're new to electronics and programming, or a seasoned veteran, Circuit Playground Express is an amazing board to work with. With so much built into the board, you can learn how different types of electronics work and learn to program them all without purchasing any other parts. All you need is a USB cable and the

board! But, that's just the beginning. Many of the pads around the outside of the board function in multiple ways allowing you to wire other things to the board. For example, you could wire up a servo or a potentiometer. The possibilities are endless!

The [Circuit Playground Express guide](#) has tons of information on all the fantastic features of the board. The [CircuitPython section](#) of the guide has an extensive list of examples using the built-in features of the CircuitPython and the board. There is also a section called [Python Playground](#) with more demos and a [Drum Machine project](#) to try out.

You can turn your Circuit Playground Express into a capacitive touch tone [Piano in the Key of Lime](#) using the touch pads on the board. Use the built in accelerometer to make a [UFO Flying Saucer](#) complete with lights and alien sounds using your board, and some extra supplies from around the house or a 3D printed saucer!

Feather M0 Express

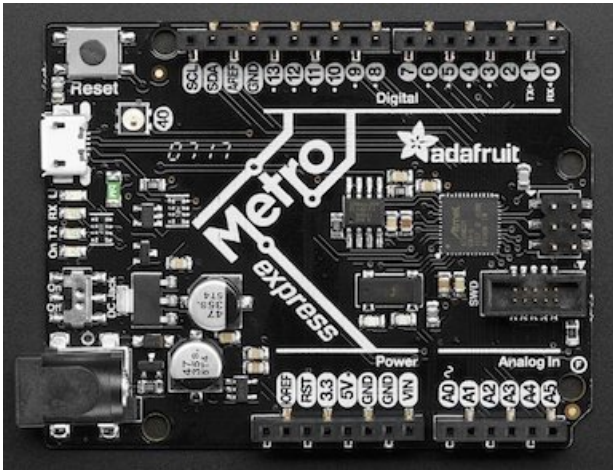


The [Adafruit Feather M0 Express](#) is the first Feather designed specifically for CircuitPython. It's part of a line of [Adafruit Feather development boards](#) designed to work standalone or stacked, and is powered by USB or lithium ion battery so it works for both stationary and on-the-go projects. Feather M0 Express comes with two headers for use with a solderless breadboard, or you can solder wires directly to the pins on the board. This allows for prototyping while you're working on your project and permanent installation when you're ready. One of the things that makes the Feather M0 Express amazing is the the huge array of boards called [Featherwings](#) which are designed to fit right on the Feather. There are CircuitPython libraries for many of these boards and more are being written all the time.

Feather M0 Express ships CircuitPython-ready with the UF2 bootloader installed and ready for you to install CircuitPython when you receive your board. Create your first program, save it to the board, and off you go! The [Feather M0 Express guide](#) has all the details about your board, and a [CircuitPython section](#) to get you started.

Feather M0 Express can be used to power all kinds of projects. Build a [CircuitPython Painter](#) POV LED wand using 3D printed parts and DotStar strips. Create an engraved edge-lit [LED Acrylic Sign](#) with NeoPixels. There are guides to go with the Featherwings that explain how to use them with CircuitPython, like the [OLED Display](#) and [Adalogger Featherwing](#).

Metro M0 Express



[Metro M0 Express](#) is the first Metro board designed to work with CircuitPython. This is not a beginner board. If you're just getting started, we'd recommend one of the previous boards. It has a lot of the same features as Feather M0 Express, as well as some development specific features (like the SWD port built in!). The Metro M0 Express is designed to work with the Arduino form-factor, so if you've already got Arduino shields, this board would be great for you. There are CircuitPython libraries for some shields already. It has 25 GPIO pins (the most of any of these boards!) so it's great if you're looking for a lot of options.

Metro M0 Express ships CircuitPython-ready with the UF2 bootloader installed and is ready for you to install CircuitPython when you receive your board. Create your first program, save it to the board, and you're good to go! The [Metro M0 Express guide](#) gives you all the details about your board, and the [CircuitPython section](#) is available to get you started.

All sensors and breakout boards with CircuitPython libraries will work with the Metro M0 Express running CircuitPython. Find the guide for your sensor and follow the guide to find out how to wire it up. There are a ton of options available.

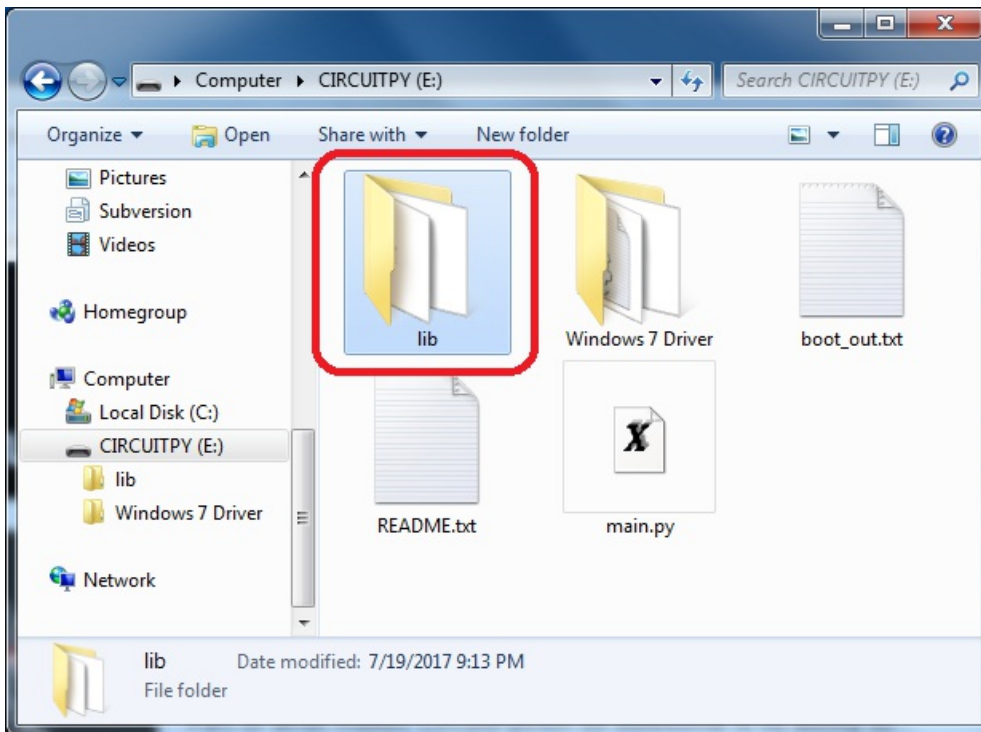
What's Next?

Now you're ready to jump into some more Learn Guides or simply get started with a brand new project. Great job, and good luck!

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 way 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 it's part of the Python path by default.

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

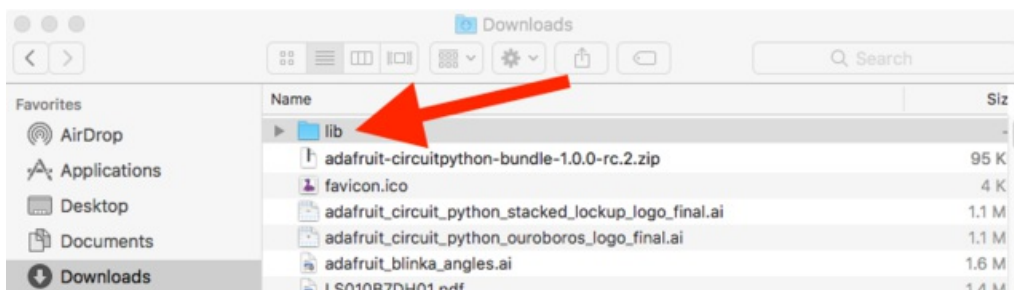
Click for the Latest Adafruit CircuitPython Library Bundle Release

<https://adafru.it/AgR>

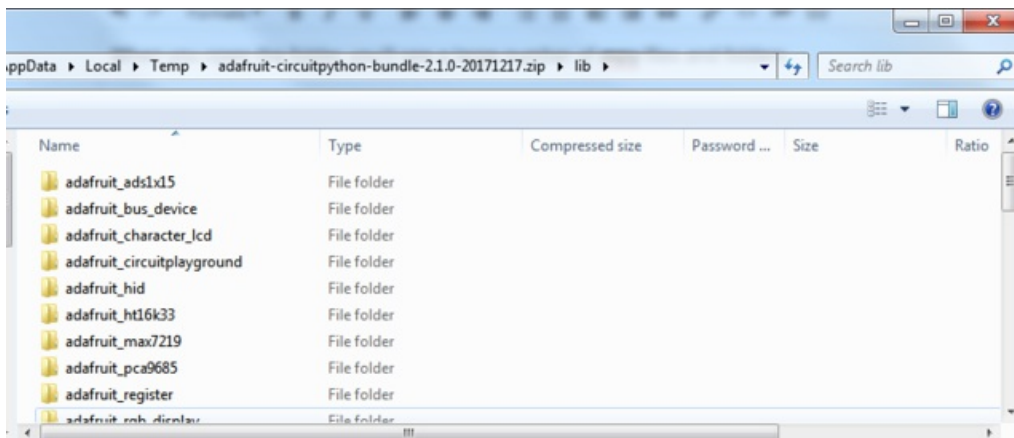
If you need another version, [you can also visit the bundle release page](#) which will let you select exactly what version you're looking for, as well as information about changes.

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.**

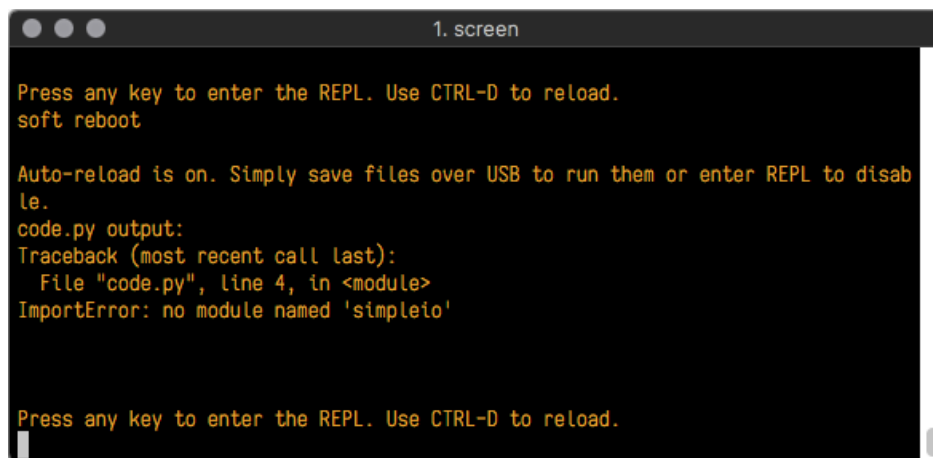
Let's use a modified version of the blinky example.

```
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.



```
1. screen

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

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

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

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

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

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

```
Press any key to enter the REPL. Use CTRL-D to reload.  
soft reboot  
  
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.  
code.py output:  
|
```

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

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

Library Install on Non-Express Boards

If you have 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

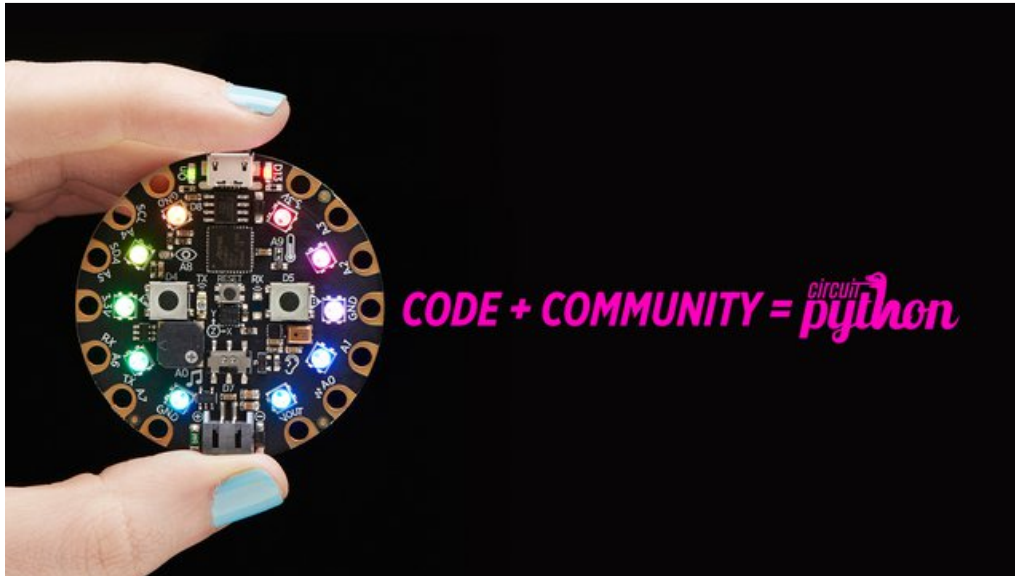
Libraries are updated from time to time, and it's important to update the files you have on your `CIRCUITPY` drive.

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 `CIRCUITPY` 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.

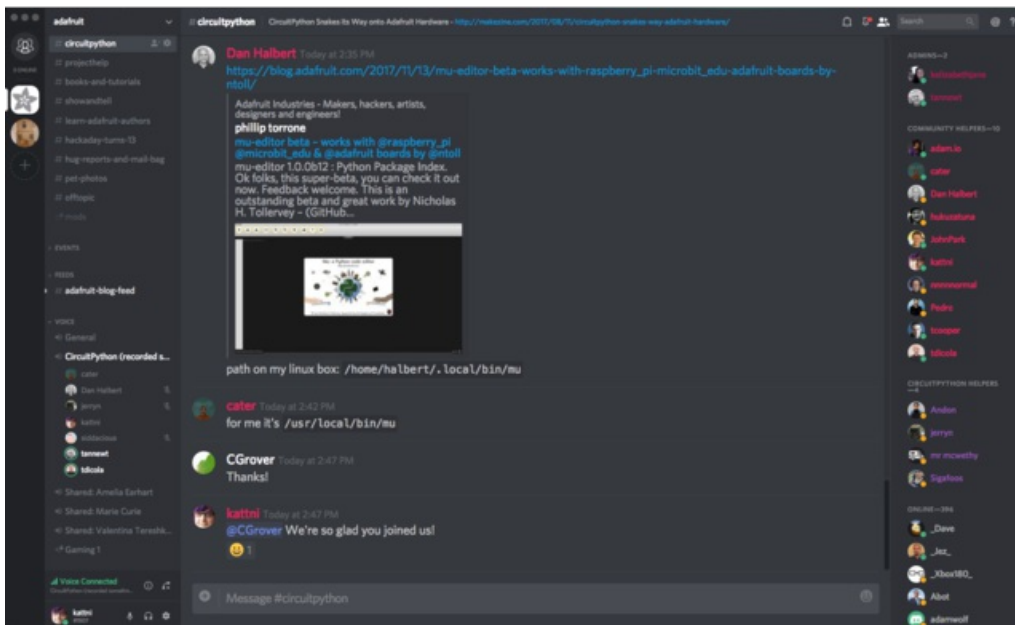
Welcome to the Community!



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.

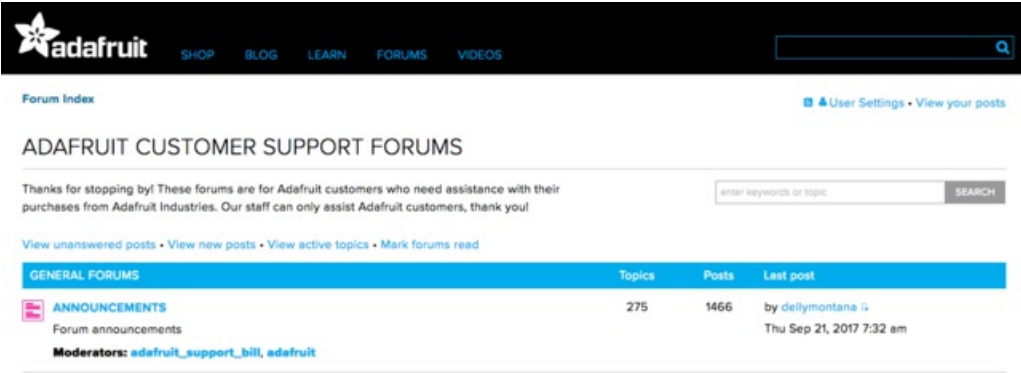
The CircuitPython channel is where to go with your CircuitPython questions. #circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. We'd love to hear what you have to say!

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 screenshot shows the Adafruit website's forum section. At the top is the Adafruit logo and navigation links: SHOP, BLOG, LEARN, FORUMS, and VIDEOS. Below the navigation bar is a search bar and links for "User Settings" and "View your posts". The main heading is "ADAFRUIT CUSTOMER SUPPORT FORUMS". A message states: "Thanks for stopping by! These forums are for Adafruit customers who need assistance with their purchases from Adafruit Industries. Our staff can only assist Adafruit customers, thank you!". Below this is a search bar with the placeholder "enter keywords or topic" and a "SEARCH" button. A link says "View unanswered posts • View new posts • View active topics • Mark forums read". A table titled "GENERAL FORUMS" lists forum categories. The first row is "ANNOUNCEMENTS" with 275 topics and 1466 posts, last post by "dellymontana" on "Thu Sep 21, 2017 7:32 am". Below the table, it says "Moderators: adafruit_support_bill, adafruit".

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

The [Adafruit Forums](#) 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.

Forum Index > Supported Products & Projects > Adafruit CircuitPython and MicroPython User Settings • View your posts

Adafruit CircuitPython and MicroPython

Moderators: [adafruit_support_bill](#), [adafruit](#)

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

POST A TOPIC Mark topics read • 179 topics • Page 1 of 4 • [1](#) [2](#) [3](#) [4](#)

Please be positive and constructive with your questions and comments.

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

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

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

Adafruit Github

GitHub repository page for **adafruit / circuitpython** (forked from [micropython/micropython](#))

Unwatch 69 Unstar 256 Fork 1,357

Code Issues 73 Pull requests 4 Insights

CircuitPython - a Python implementation for teaching coding with microcontrollers

circuitpython

9,856 commits 32 branches 73 releases 206 contributors

Whether you're just beginning or are life-long programmer who would like to contribute, there are ways for everyone to be a part of building CircuitPython. GitHub is the best source of ways to contribute to [CircuitPython](#) 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.

Issues list showing "good first issue" labels:

- OneWire BusDevice driver** (good first issue) #338 opened 29 days ago by tannewt Long term 2 comments
- Feather M0 Adalogger does not have D8 or D7** (good first issue) #323 opened on Oct 13 by ladyada 3.0 7 comments
- Audit and fix native API for methods that accept and ignore extra args.** (good first issue) #321 opened on Oct 12 by tannewt Long term

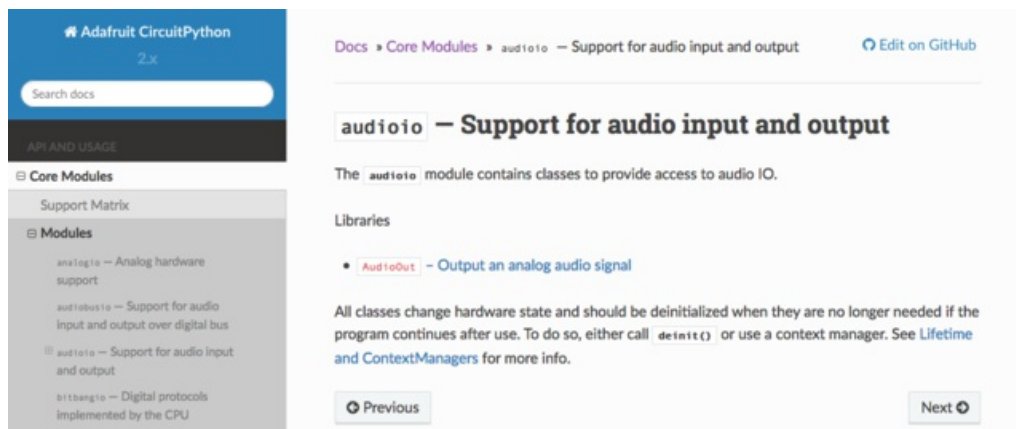
Already experienced and looking for a challenge? Checkout the rest of the issues list and you'll find plenty of ways to contribute. You'll find everything from new driver requests to core module updates. There's plenty of opportunities for everyone at any level!

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

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

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

ReadTheDocs



[ReadTheDocs](#) is 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 for ESP8266

We have two 'strains' of CircuitPython, the primary one is the ATSAM21/51-based boards that have native USB connectivity. Native USB means that the board can show up as a disk drive called **CIRCUITPY** and hold all your files on it.

There's also CircuitPython for boards like the ESP8266 and nRF52832, these are really nice chips with WiFi and Bluetooth, respectively, built in. **But they do not have native USB!** That means there is no way for the chip to appear as a disk drive. You can still use them with CircuitPython but it's a lot tougher, so we don't recommend them for beginners. (When USB-native versions of these chips come out, we'll take a look at making them show up like a drive!)

Here's what you have to know about using non-native chips for CircuitPython:

- You *only* get a REPL connection! No HID keyboard/mouse or other USB interface
- No disk drive for drag-n-drop file moving, files must be moved via a special tool such as **ampy** that 'types' the file in for you via the REPL
- Loading CircuitPython requires command line tools

Installing CircuitPython on the ESP8266

To use CircuitPython with the ESP8266 you'll need to first flash it with the latest firmware.

Download esptool

First install the **esptool.py** software which enables firmware flashing on the ESP8266. The easiest way to install this tool is from Python's pip package manager. If you don't have it already you'll need to [install Python 2.7](#) (make sure you check the box to put Python in your system path when installing on Windows) and then run the following command in a terminal: **pip install esptool**

Note on Mac OSX and Linux you might need to run the command as root with sudo, like: **sudo pip install esptool**



```
tony-imac:Downloads tony$ pip2 install esptool
Collecting esptool
  Using cached esptool-1.1-py2-none-any.whl
Requirement already satisfied (use --upgrade to upgrade): pyserial in /usr/local
/lib/python2.7/site-packages (from esptool)
Installing collected packages: esptool
Successfully installed esptool-1.1
tony-imac:Downloads tony$
```

If you receive an error that **esptool.py** only supports Python 2.x try running again with the **pip2** command instead of pip (likely your system is using Python 3 and the pip command is getting confused which version to use).

Download Latest CircuitPython Firmware

Next [download the latest CircuitPython ESP8266 firmware file](#). You can automatically download the latest binary by clicking here:

Download Latest ESP8266 Huzzah Firmware

<https://adafru.it/AIQ>

Get ESP8266 Ready For Bootloading

Now you'll need to put the ESP8266 into its firmware flashing mode. Each ESP8266 board is slightly different:

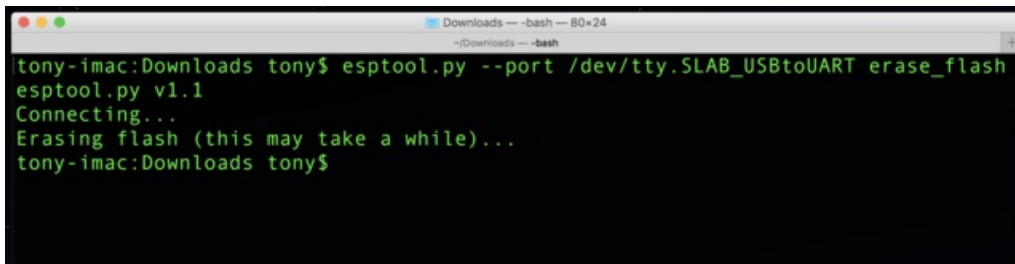
- For a raw ESP8266 module you'll need to wire up buttons to ground for the GPIO0 and RESET pins. Hold the GPIO0 button down (or connect the line to ground) and while still holding GPIO0 to ground press and release the RESET button (or connect and release the line from ground), then release GPIO0.
- For the [HUZZAH ESP8266 breakout](#) buttons for GPIO0 and RESET are built in to the board. Hold GPIO0 down, then press and release RESET (while still holding GPIO0), and finally release GPIO0.
- For the [Feather Huzzah ESP8266](#) you don't need to do anything special to go into firmware flashing mode. This board is built to detect when the serial port is opened for flashing and automatically configure the ESP8266 module to receive firmware. **Be sure to first [install the SiLabs CP210x driver on Windows and Mac OSX](#) to make the board's serial port visible!** On Windows you want the normal VCP driver, not the 'with Serial Enumeration' driver.

Erase ESP8266

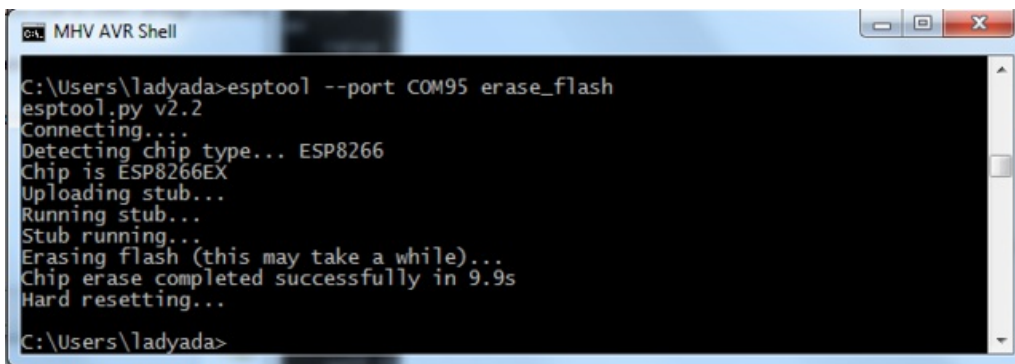
It's recommended to erase the entire flash memory of the ESP8266 board before uploading firmware. Run the following command in a terminal to perform this erase:

```
esptool.py --port ESP8266_PORTNAME erase_flash
```

Where `ESP8266_PORTNAME` is the path or name of the serial port that is connected to the ESP8266. The exact name of the device varies depending on the type of serial to USB converter chip so you might need to look at the serial ports with and without the device connected to find its name.



```
tony-imac:Downloads tony$ esptool.py --port /dev/tty.SLAB_USBtoUART erase_flash
esptool.py v1.1
Connecting...
Erasing flash (this may take a while)...
tony-imac:Downloads tony$
```



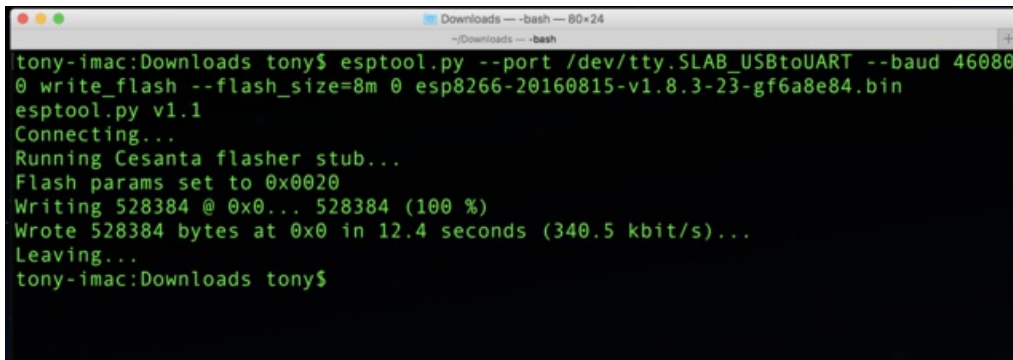
```
C:\Users\ladyada>esptool --port COM95 erase_flash
esptool.py v2.2
Connecting...
Detecting chip type... ESP8266
Chip is ESP8266EX
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 9.9s
Hard resetting...
C:\Users\ladyada>
```

Program ESP8266

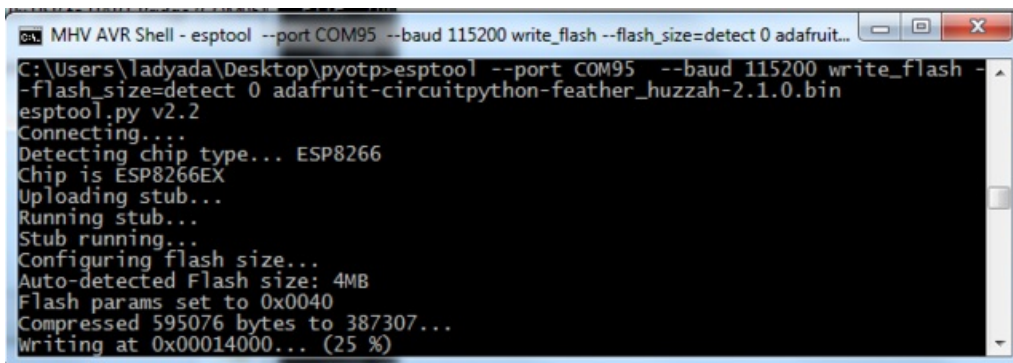
Now put the ESP8266 back into firmware flashing mode and run the following command to load the downloaded firmware file:

```
esptool.py --port ESP8266_PORTNAME --baud 115200 write_flash --flash_size=detect 0 firmware.bin
```

Again set `ESP8266_PORTNAME` to the path or name of the serial port that is connected to the ESP8266. In addition set `firmware.bin` to the name or path to the firmware file you would like to load.



```
tony-imac:Downloads tony$ esptool.py --port /dev/tty.SLAB_USBtoUART --baud 46080
0 write_flash --flash_size=8m 0 esp8266-20160815-v1.8.3-23-gf6a8e84.bin
esptool.py v1.1
Connecting...
Running Cesanta flasher stub...
Flash params set to 0x0020
Writing 528384 @ 0x0... 528384 (100 %)
Wrote 528384 bytes at 0x0 in 12.4 seconds (340.5 kbit/s)...
Leaving...
tony-imac:Downloads tony$
```



```
MHV AVR Shell - esptool --port COM95 --baud 115200 write_flash --flash_size=detect 0 adafruit...
C:\Users\ladyada\Desktop\pyotp>esptool --port COM95 --baud 115200 write_flash -
-flash_size=detect 0 adafruit-circuitpython-feather_huzzah-2.1.0.bin
esptool.py v2.2
Connecting...
Detecting chip type... ESP8266
Chip is ESP8266EX
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0040
Compressed 595076 bytes to 387307...
Writing at 0x00014000... (25 %)
```

Once the tool finishes flashing the firmware (you'll usually see a blue light on the ESP8266 module flashing during this process) press the RESET button on the ESP8266 board or disconnect and reconnect it to your computer. You should be all set to start using the latest CircuitPython firmware on the board!

Note that if you see an error that "detect is not a valid flash_size parameter" you might be using an older version of esptool.py. To upgrade to the latest version run the following command `pip install --upgrade esptool`

Upload Libraries & Files Using Ampy!

The biggest difference you'll find with ESP8266 is that you need to use a special tool to move files around. Check out `ampy` by reading this guide. It's about MicroPython but CircuitPython is nearly identical so the the overall installation and usage is identical!

Learn how to use ampy to move files on your
ESP8266

<https://adafru.it/q9C>

Other Stuff To Know!

- The REPL works as you'd expect, so check out that introductory page.
- File storage is in the same chip as CircuitPython so if you update, you may lose your files! Keep backups.
- Libraries and API are also the same as for other CircuitPython boards.
- Note that the ESP8266 does not have a ton of pins available, and only one analog input with 0-1.0V range. There

is no UART port available (it's the one used for the REPL!)

- There are no analog outputs.
- For SPI and I2C, you can use them! But you will need to use [bitbangio](#) to create the bus objects

Advanced Serial Console on Windows

Windows 7 Driver

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

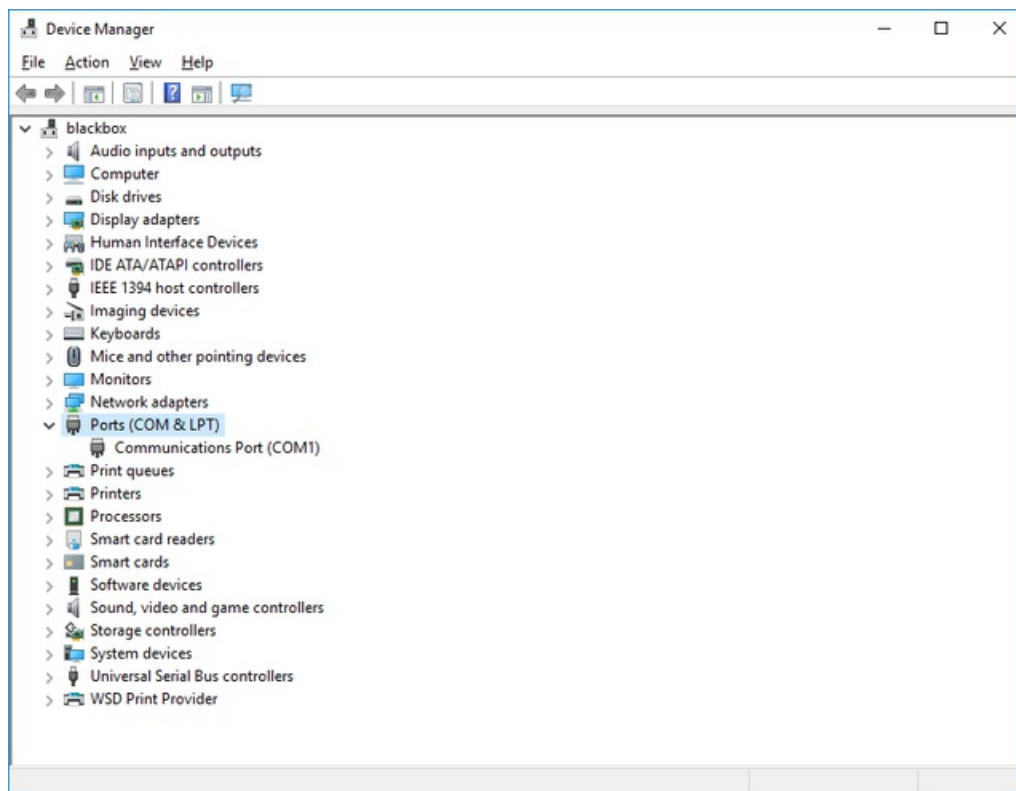
Download Windows 7 Drivers

<https://adafru.it/Aif>

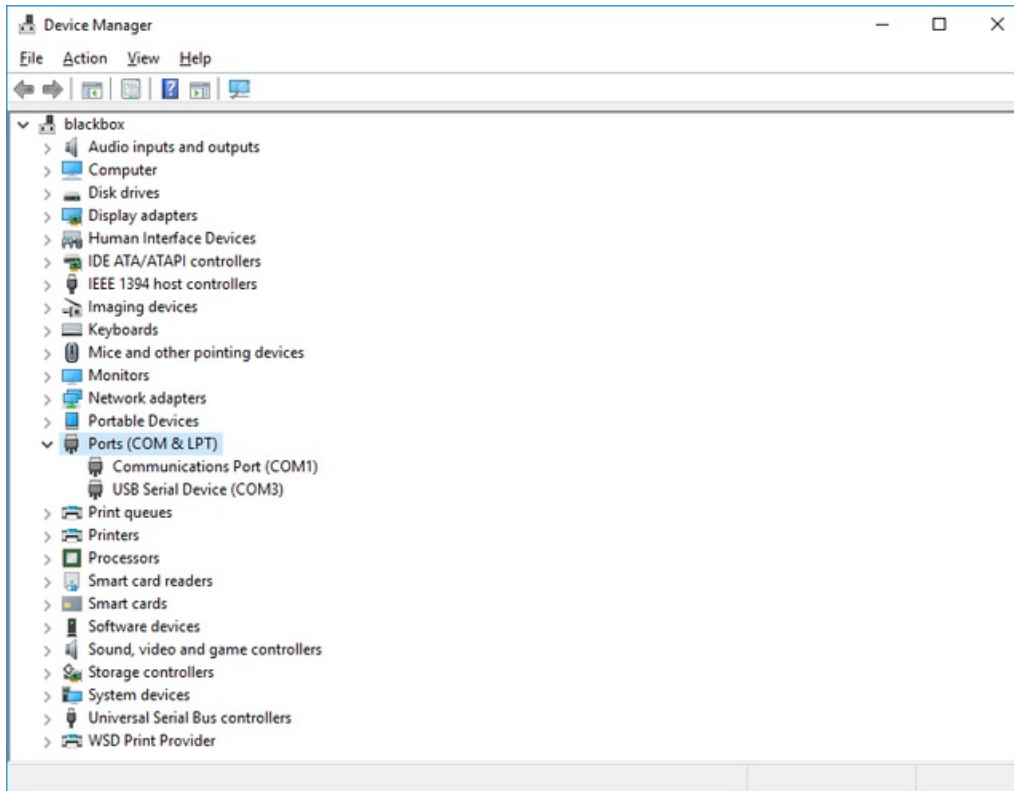
What's the COM?

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

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



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



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

Install Putty

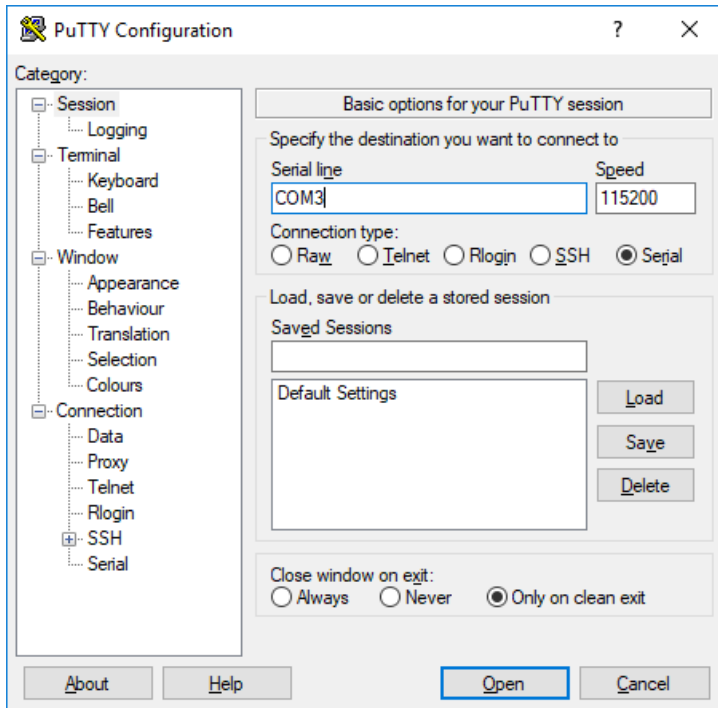
If you're using Windows, you'll need to download a terminal program. We're going to use PuTTY.

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

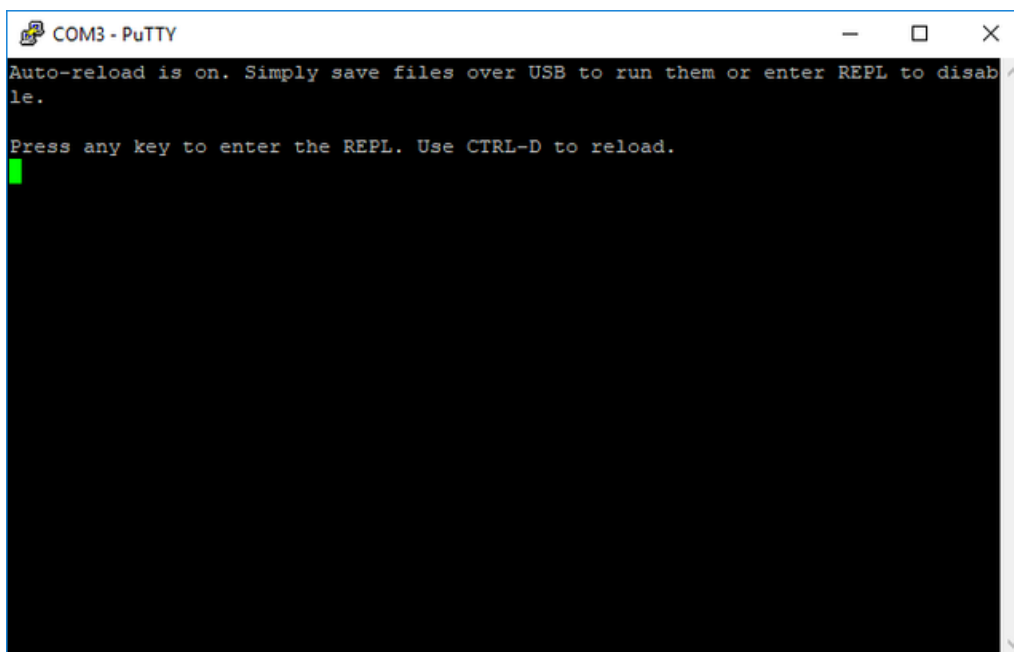
Now you need to open PuTTY.

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

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



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



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

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

Advanced Serial Console on Mac and Linux

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

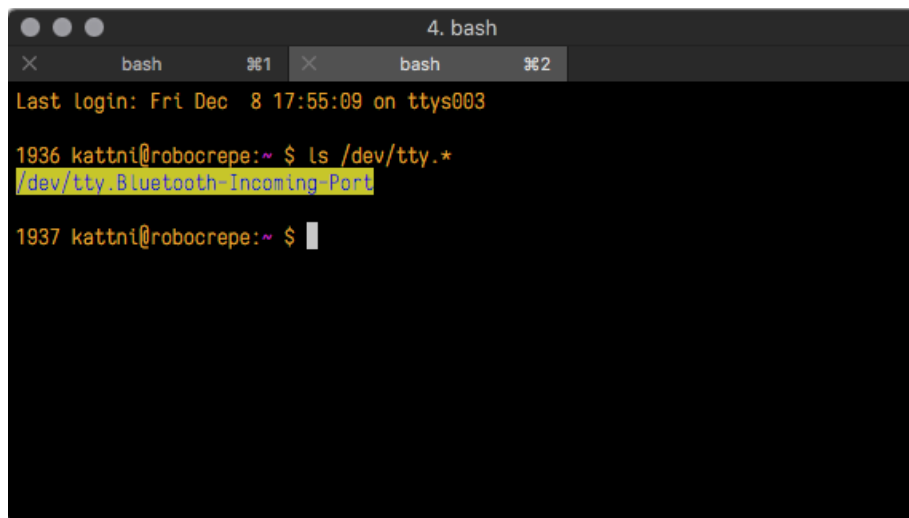
What's the Port?

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

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

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

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

A screenshot of a macOS Terminal window titled "4. bash". It shows two tabs, both labeled "bash". The first tab shows the login message "Last login: Fri Dec 8 17:55:09 on ttys003". The second tab shows the command prompt "1936 kattni@robocrepe:~ \$" followed by the command "ls /dev/tty.*". The output of the command is "/dev/tty.Bluetooth-Incoming-Port", which is highlighted in yellow. Below the output, the prompt "1937 kattni@robocrepe:~ \$" is visible with a cursor.

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

```
ls /dev/ttyACM*
```

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


```
jerryneedell@Ubuntu-Macmini: ~  
jerryneedell@Ubuntu-Macmini:~$ ls /dev/ttyACM*  
ls: cannot access '/dev/ttyACM*': No such file or directory  
jerryneedell@Ubuntu-Macmini:~$ ls /dev/ttyACM*  
/dev/ttyACM0  
jerryneedell@Ubuntu-Macmini:~$
```

Now, plug your board. Using Mac, type:

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

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

```
4. bash  
bash %1 bash %2  
Last login: Fri Dec 8 17:55:09 on ttys003  
1936 kattni@robocrepe:~$ ls /dev/tty.*  
/dev/tty.Bluetooth-Incoming-Port  
1937 kattni@robocrepe:~$ ls /dev/tty.*  
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441  
1937 kattni@robocrepe:~$
```

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

Using Linux, type:

```
ls /dev/ttyACM*
```

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

```
jerryneedell@Ubuntu-Macmini: ~  
jerryneedell@Ubuntu-Macmini:~$ ls /dev/ttyACM*  
ls: cannot access '/dev/ttyACM*': No such file or directory  
jerryneedell@Ubuntu-Macmini:~$ ls /dev/ttyACM*  
/dev/ttyACM0  
jerryneedell@Ubuntu-Macmini:~$
```

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

Connect with screen

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

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

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

```
4. bash  
bash %1 bash %2  
Last login: Fri Dec 8 17:55:09 on ttys003  
1936 kattni@robocrepe:~$ ls /dev/tty.*  
/dev/tty.Bluetooth-Incoming-Port  
1937 kattni@robocrepe:~$ ls /dev/tty.*  
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441  
1937 kattni@robocrepe:~$ screen /dev/tty.usbmodem141441 115200
```

```
jerryneedell@Ubuntu-Macmini:~$
jerryneedell@Ubuntu-Macmini:~$
jerryneedell@Ubuntu-Macmini:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
jerryneedell@Ubuntu-Macmini:~$ ls /dev/ttyACM*
/dev/ttyACM0
jerryneedell@Ubuntu-Macmini:~$ screen /dev/ttyACM0 115200
```

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

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

Permissions on Linux

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

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

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

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

Once you enter your password, you should be in:

```
ackbar@desk: ~
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

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

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

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

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

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

You'll need to logout and log back in. Then you can verify that you have been added to the group using the command `groups`.

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

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

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

And you're in:

```
ackbar@desk: ~
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

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

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

Non-UF2 Installation

This installation page is only required if you do not have UF2 bootloader installed (e.g. boardBOOT drag-n-drop)! This page is for non-Express Feather M0's, Arduino Zero's & M0's, and other custom ATSAMD21 boards.

Flashing with Bossac - For Non-Express Feather M0's & Arduino Zero

Our older Feather M0 boards don't come with UF2, instead they come with a simpler bootloader called **bossa**. This is also what is installed on Arduino Zero's and other Zero-compatible boards that use the ATSAMD21. It is the only method you can use if your CircuitPython installation file is a **.bin** rather than a **.uf2**

Command-Line ahoy!

Flashing with bossac requires the use of your computer's command line interface. On windows, that's the **cmd** or **powershell** tool. On mac and linux, use **Terminal**!

Download Latest CircuitPython Firmware

The first thing you'll want to do is download the most recent version of CircuitPython.

Download the .bin file for your CircuitPython board

<https://adafru.it/vIF>

Once downloaded, save the **.bin** file onto your desktop, you'll need it soon!

If you are running Windows 7, [you must install the driver set \(we cover it on this page\)](#) so you have access to the COM port.

Download BOSSA

Once you have a firmware image you'll need to download a special version of the BOSSA tool that can load firmware on SAMD21 boards. This tool is actually used internally by the Arduino IDE when it programs these boards, however you can use it yourself to load custom firmware

Be aware you **must** use this special **1.7.0 or higher** version of bossac to program SAMD21 boards! If you find an older version of BOSSA it won't work because it doesn't support the SAMD21 chip. To flash with **bossac** (BOSSA's command line tool) first download the latest version from [here](#). The **ming32** version is for Windows, **apple-darwin** for Mac OSX and various **linux** options for Linux.

bossac works with **.bin** files only, it will not work with **.uf2** files!

Test bossac

Open a terminal and navigate to the folder with the **bossac** tool. Then check the tool runs by running it with the **--help** option:

```

tony-mac:bossac-1.7.0 tony$ ./bossac --help
Usage: bossac [OPTION...] [FILE]
Basic Open Source SAM-BA Application (BOSSA) Version 1.7.0
Flash programmer for Atmel SAM devices.
Copyright (c) 2011-2012 ShumaTech (http://www.shumatech.com)

Examples:
  bossac -e -w -v -b image.bin  # Erase flash, write flash with image.bin,
  bossac -r0x10000 image.bin     # verify the write, and set boot from flash
                                # Read 64KB from flash and store in image.bin

Options:
  -e, --erase          erase the entire flash (keep the 8KB of bootloader for SAM Dxx)
  -w, --write          write FILE to the flash; accelerated when
                        combined with erase option
  -r, --read[=SIZE]    read SIZE from flash and store in FILE;
                        read entire flash if SIZE not specified
  -v, --verify         verify FILE matches flash contents
  -p, --port=PORT      use serial PORT to communicate to device;
                        default behavior is to auto-scan all serial ports
  -b, --boot[=BOOL]    boot from ROM if BOOL is 0;
                        boot from FLASH if BOOL is 1 [default];
                        option is ignored on unsupported devices
  -c, --bod[=BOOL]     no brownout detection if BOOL is 0;
                        brownout detection is on if BOOL is 1 [default]
  -t, --bor[=BOOL]     no brownout reset if BOOL is 0;
                        brownout reset is on if BOOL is 1 [default]
  -l, --lock[=REGION]  lock the flash REGION as a comma-separated list;
                        lock all if not given [default]
  -u, --unlock[=REGION] unlock the flash REGION as a comma-separated list;
                        unlock all if not given [default]
  -s, --security       set the flash security flag
  -i, --info           display device information

```

Open a terminal and navigate to the folder with the **bossac** tool. Then check the tool runs by running it with the `--help` option with `bossac --help`

Or if you're using Linux or Mac OSX you'll need to add a `./` to specify that bossac is run from the current directory like `./bossac --help`

Make sure you see BOSSA version 1.7.0 or higher! If you see a lower version then you accidentally downloaded an older version of the tool and it won't work to flash SAMD21 chips. Go back and [grab the latest release from this BOSSA GitHub repository \(https://adafru.it/s1B\)](https://adafru.it/s1B) as mentioned above.

Get Into the Bootloader

You'll have to 'kick' the board into the bootloader manually. Do so by double-clicking the reset button. The red "#13" LED should pulse on and off. If you are using an Arduino Zero, make sure you are connected to the **native USB** port not the debugging/programming port.

One special note, if you're using the Arduino M0 from Arduino.org you'll need to replace its bootloader with the Arduino Zero bootloader so it can work with BOSSA. To do this install the Arduino/Genuino Zero board in the Arduino IDE board manager and then [follow these steps to burn the Arduino Zero bootloader](#) (using the **programming** port on the board). Once you've loaded the Arduino Zero bootloader you should be able to use the M0 with bossac as described below.

Run bossac command

With your board plugged in and running the bootloader you're ready to flash CircuitPython firmware onto the board. Copy the firmware .bin file to the same directory as the **bossac** tool, then in a terminal navigate to that location and run the following command:

```
bossac -e -w -v -R adafruit-circuitpython-feather_m0_express-2.1.0.bin
```

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

This will **e**rase the chip, **w**rite the given file, **v**erify the write and **R**eset the board. On linux or Mac OS X you may need to run this command with `sudo ./bossac ...`

After BOSSA loads the firmware you should see output similar to the following:

```
tony-imac:bossac-1.7.0 tony$ ./bossac -e -w -v -R -p tty.usbmodem143411 firmware.bin
Atmel SMART device 0x10010005 found
Erase flash
done in 0.892 seconds

Write 182400 bytes to flash (2850 pages)
[=====] 100% (2850/2850 pages)
done in 1.628 seconds

Verify 182400 bytes of flash with checksum.
Verify successful
done in 0.723 seconds
CPU reset.
```

After reset, CircuitPython should be running and the **CIRCUITPY** drive will appear. You can always manually reset the board by clicking the reset button, sometimes that is needed to get it to 'wake up'. 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!

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.

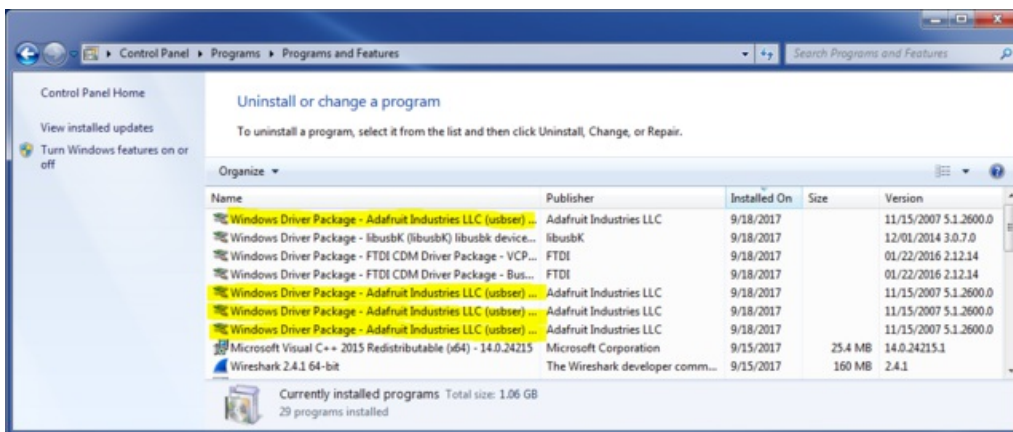
Windows 10

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 ...".

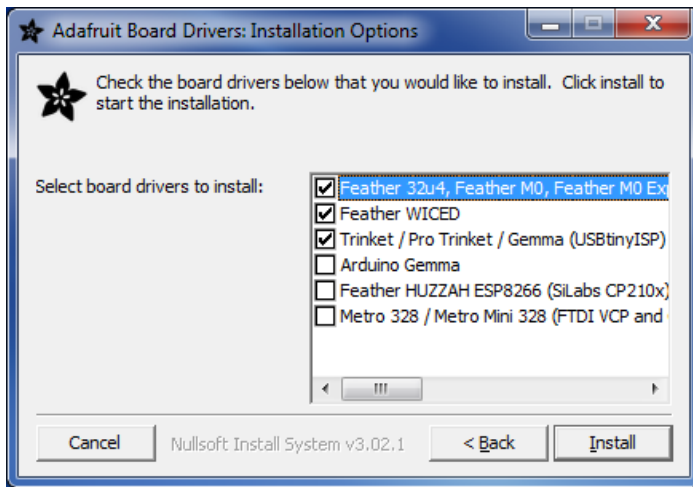


- Now install the new 2.0.0.0 (or higher) Adafruit Windows Drivers Package:

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.

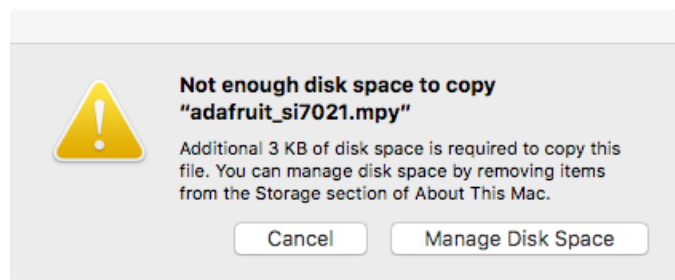
Delete something!

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 .{,._}{fsevents,Spotlight-V*,Trashes}
mkdir .fsevents
touch .fsevents/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:

```
1. bash
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem      Size  Used Avail Capacity iused ifree %used  Mounted on
/dev/disk3s1    59Ki   54Ki  5.5Ki   91%    128     0  100%  /Volumes/CIRCUITPY

(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./
../
.TemporaryItems/
.Trashes/
..TemporaryItems*
Windows 7 Driver/
._Trashes*
._original_code.py*
.fseventsd/
README.txt*
boot_out.txt*
code.py*
lib/
original_code.py*
```

Lets remove the `._` files first.

```
1. bash
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem      Size  Used Avail Capacity iused ifree %used  Mounted on
/dev/disk3s1    59Ki   54Ki  5.5Ki   91%    128     0  100%  /Volumes/CIRCUITPY

(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./
../
.TemporaryItems/
.Trashes/
..TemporaryItems*
Windows 7 Driver/
._Trashes*
._original_code.py*
.fseventsd/
README.txt*
boot_out.txt*
code.py*
lib/
original_code.py*

(venv) tannewt@shallan:/Volumes $ rm CIRCUITPY/._*
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem      Size  Used Avail Capacity iused ifree %used  Mounted on
/dev/disk3s1    59Ki   42Ki  18Ki   71%    128     0  100%  /Volumes/CIRCUITPY

(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./
../
.Trashes/
.fseventsd/
Windows 7 Driver/
lib/
boot_out.txt*
original_code.py*
.TemporaryItems/
README.txt*
code.py*
```

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