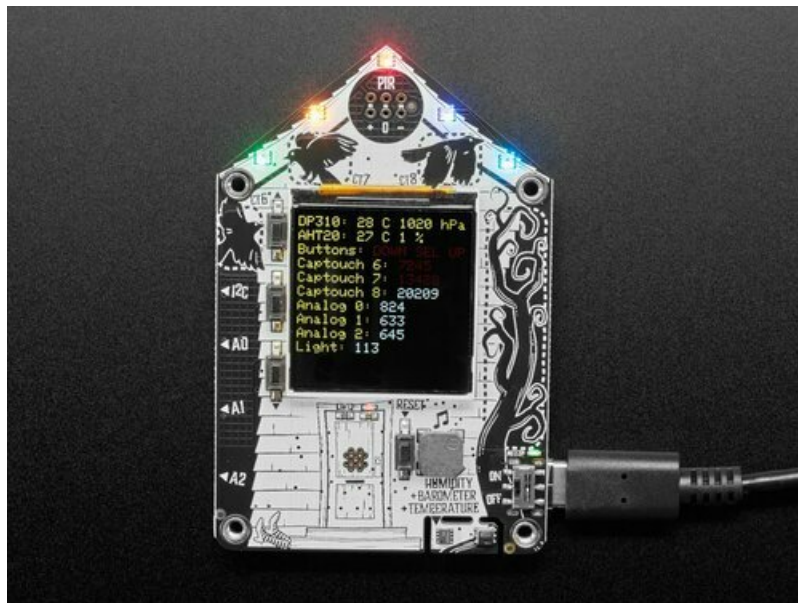


Adafruit FunHouse

Created by Melissa LeBlanc-Williams



Last updated on 2021-10-16 03:46:26 PM EDT

Guide Contents

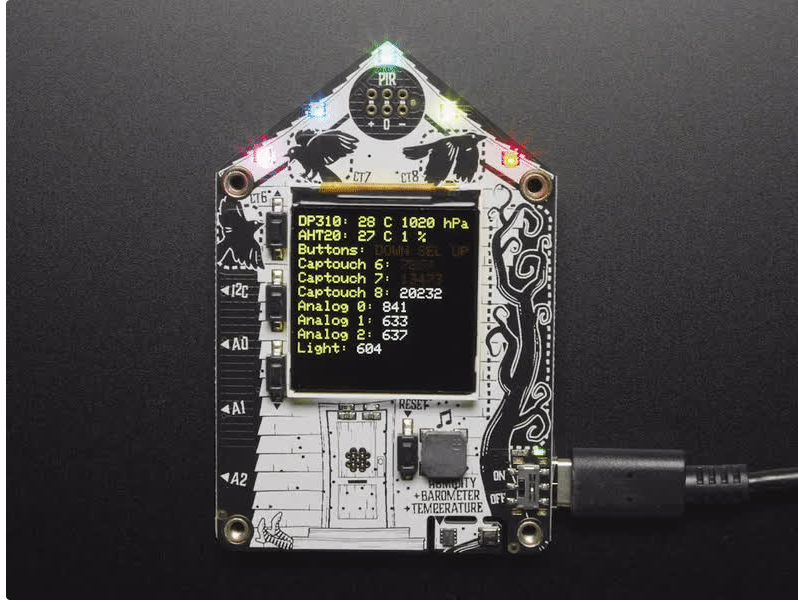
Guide Contents	2
Overview	6
Pinouts	9
TFT Display and Display Connector	10
Power	10
ESP32-S2 WiFi Module	11
DotStar LEDs and Red LED	11
STEMMA QT	11
Digital/Analog Connectors	12
Speaker and Sensors	12
Reset and Boot Buttons	12
User Buttons	13
Capacitive Touch Pads and Slider	13
PIR Sensor Port	13
Install UF2 Bootloader	14
Step 1. Download the tinyuf2_combo.bin file here	14
Step 2. Place your board in bootloader mode	14
Check for a new serial / COM port	14
Step 3 Option A. Use the Web Serial ESPTool to upload	15
Enable Web Serial (For older chrome)	15
Connecting	16
Erasing the Contents	16
Programming the Microcontroller	16
Step 3. Option B. Use esptool.py to upload (for advanced users)	17
Install ESPTool.py	17
Test the Installation	17
Installing the Bootloader	18
Step 4. Reset the board	18
CircuitPython	19
Set Up CircuitPython	19
Option 1 - Load with UF2 Bootloader	19
Try Launching UF2 Bootloader	19
Option 2 - Use Chrome Browser To Upload BIN file	20
Option 3 - Use esptool to load BIN file	20
CircuitPython Internet Libraries	22
Adafruit CircuitPython Library Bundle	22
CircuitPython Internet Test	23
Secrets File	23
Connect to WiFi	23
Getting The Date & Time	27
Step 1) Make an Adafruit account	27
Step 2) Sign into Adafruit IO	27
Step 3) Get your Adafruit IO Key	27
Step 4) Upload Test Python Code	27
FunHouse-Specific CircuitPython Libraries	29
Get Latest Adafruit CircuitPython Bundle	29
Secrets	29
Welcome To CircuitPython	30
This guide will get you started with CircuitPython!	30
Installing the Mu Editor	31
Download and Install Mu	31
Using Mu	31

Creating and Editing Code	33
Creating Code	33
Editing Code	34
Your code changes are run as soon as the file is done saving.	34
1. Use an editor that writes out the file completely when you save it.	35
2. Eject or Sync the Drive After Writing	35
Oh No I Did Something Wrong and Now The CIRCUITPY Drive Doesn't Show Up!!!	35
Back to Editing Code...	36
Exploring Your First CircuitPython Program	36
Imports & Libraries	37
Setting Up The LED	37
Loop-de-loops	37
What Happens When My Code Finishes Running?	37
What if I don't have the loop?	38
More Changes	39
Naming Your Program File	39
Connecting to the Serial Console	40
Are you using Mu?	40
Setting Permissions on Linux	40
Using Something Else?	41
Interacting with the Serial Console	42
The REPL	44
Returning to the serial console	46
Advanced Serial Console on Windows	47
Windows 7 Driver	47
What's the COM?	47
Install Putty	48
CircuitPython Libraries	50
Installing the CircuitPython Library Bundle	50
Example Files	51
Copying Libraries to Your Board	51
Example: ImportError Due to Missing Library	52
Library Install on Non-Express Boards	52
Updating CircuitPython Libraries/Examples	52
CircuitPython Pins and Modules	54
CircuitPython Pins	54
import board	54
I2C, SPI, and UART	54
What Are All the Available Names?	55
Microcontroller Pin Names	56
CircuitPython Built-In Modules	56
Advanced Serial Console on Mac and Linux	57
What's the Port?	57
Connect with screen	58
Permissions on Linux	59
Frequently Asked Questions	61
I have to continue using an older version of CircuitPython; where can I find compatible libraries?	61
Is ESP8266 or ESP32 supported in CircuitPython? Why not?	61
How do I connect to the Internet with CircuitPython?	62
Is there asyncio support in CircuitPython?	63

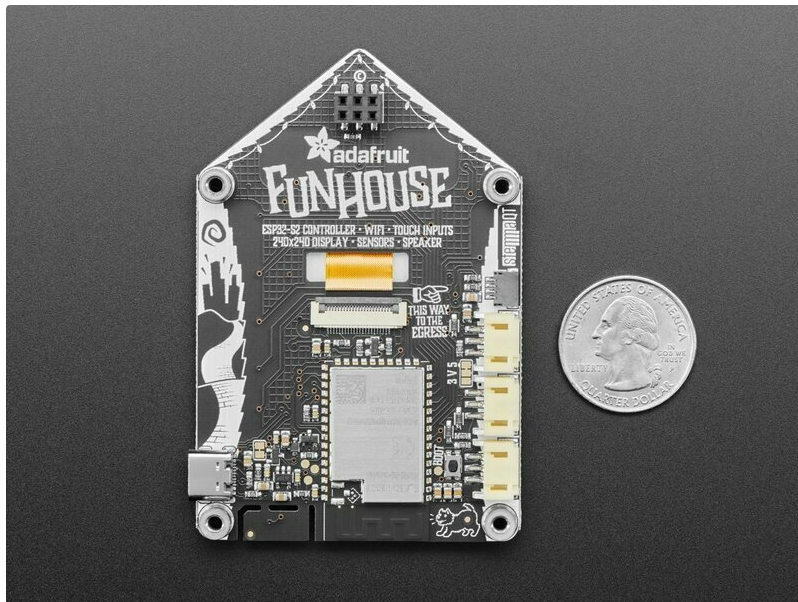
My RGB NeoPixel/DotStar LED is blinking funny colors - what does it mean?	64
What is a MemoryError?	65
What do I do when I encounter a MemoryError?	65
Can the order of my import statements affect memory?	65
How can I create my own .mpy files?	65
How do I check how much memory I have free?	65
Does CircuitPython support interrupts?	66
Does Feather M0 support WINC1500?	66
Can AVRs such as ATmega328 or ATmega2560 run CircuitPython?	66
Commonly Used Acronyms	66
ESP32-S2 Bugs & Limitations	67
Cannot reinitialize certain peripherals (especially busio.I2C)	67
No DAC-based audio output	68
Deep Sleep & Wake-up sources	69
Troubleshooting	71
Always Run the Latest Version of CircuitPython and Libraries	71
I have to continue using CircuitPython 5.x or earlier. Where can I find compatible libraries?	
CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present	71
You may have a different board.	71
MakeCode	71
MacOS	71
Windows 10	71
Windows 7 or 8.1	71
Windows Explorer Locks Up When Accessing boardnameBOOT Drive	72
Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied	72
CIRCUITPY Drive Does Not Appear	72
Device Errors or Problems on Windows	72
Serial Console in Mu Not Displaying Anything	73
CircuitPython RGB Status Light	73
CircuitPython 7.0.0 and Later	74
CircuitPython 6.3.0 and earlier	74
ValueError: Incompatible .mpy file.	74
CIRCUITPY Drive Issues	74
Easiest Way: Use storage.erase_filesystem()	75
Old Way: For the Circuit Playground Express, Feather M0 Express, and Metro M0 Express:	75
Old Way: For Non-Express Boards with a UF2 bootloader (Gemma M0, Trinket M0):	76
Old Way: For non-Express Boards without a UF2 bootloader (Feather M0 Basic Proto, Feather Adalogger, Arduino Zero):	76
Running Out of File Space on Non-Express Boards	76
Delete something!	76
Use tabs	76
MacOS loves to add extra files.	76
Prevent & Remove MacOS Hidden Files	76
Copy Files on MacOS Without Creating Hidden Files	77
Other MacOS Space-Saving Tips	77
Device Locked Up or Boot Looping	78
Welcome to the Community!	80
Adafruit Discord	80
Adafruit Forums	81
Adafruit Github	81
ReadTheDocs	82
CircuitPython Essentials	83

Blink	84
LED Location	84
Blinking an LED	84
Digital Input	85
LED and Button	85
Controlling the LED with a Button	85
Built-In DotStar LEDs	87
DotStar Location	87
DotStar Color and Brightness	87
RGB LED Colors	88
DotStar Rainbow	89
CPU Temperature	90
Microcontroller Location	90
Reading the Microcontroller Temperature	90
Arduino IDE Setup	92
Arduino Libraries	94
Install Libraries	94
Adafruit DotStar	94
Adafruit GFX	94
Adafruit ST7735 and ST7789	94
Adafruit ImageReader	94
Adafruit AHTX0	95
Adafruit DPS310	95
Arduino Basics	96
Using the Red LED	96
Reading the Buttons	96
Reading the Capacitive Touch Pads	97
Using On-Board DotStars	97
Using On-board Humidity and Temperature Sensor	98
Using On-board Pressure Sensor	99
Using the TFT Display	99
Arduino Self Test Example	101
Downloads	105
Files:	105
Schematic	105
Fab Print	105

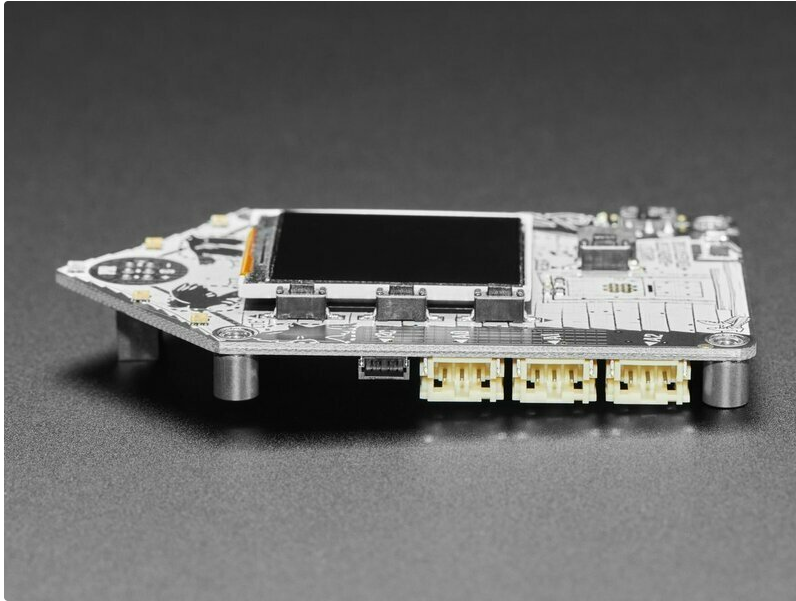
Overview



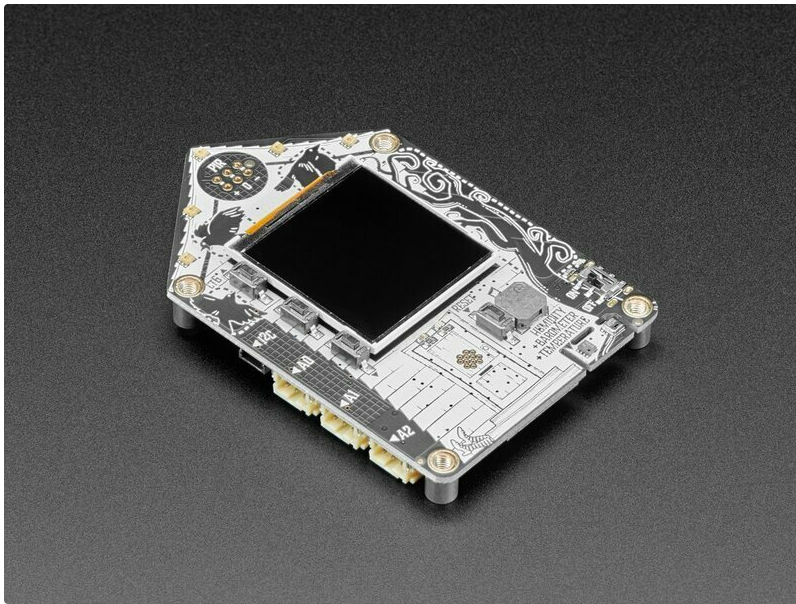
Home is where the heart is...it's also where we keep all our electronic bits. So why not wire it up with sensors and actuators to turn our house into an electronic wonderland. Whether it's tracking the environmental temperature and humidity in your laundry room, or notifying you when someone is detected in the kitchen, to sensing when a window was left open, or logging when your cat leaves through the pet door, this board is designed to make it way easy to make WiFi-connected home automation projects.



The main processor is the ESP32-S2, which has the advantage of the low cost and power of the ESP32 with the flexibility of CircuitPython support thanks to native USB support. There's also Arduino support for this chip, and many existing ESP32 projects seem to run as-is. Note this chip does not have BLE support, but for WiFi projects its great. You can use it to connect to IoT services or just the Internet in general, with SSL support and this module has plenty of PSRAM for any kind of data processing.



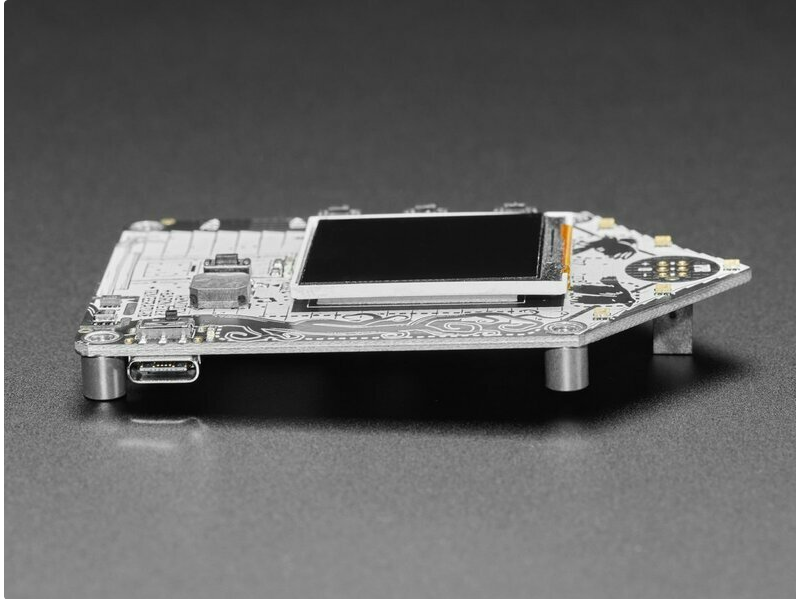
The board is designed to make it easy to wire up sensors with little or no soldering. There are built in sensors for light, pressure, humidity and temperature sensors. [Three JST PH plugs allow for quick connection of STEMMA boards \(https://adafru.it/RKb\)](https://adafru.it/RKb) that use digital or analog I/O, and there's a [STEMMA QT port for any I2C devices. \(https://adafru.it/NmD\)](https://adafru.it/NmD)



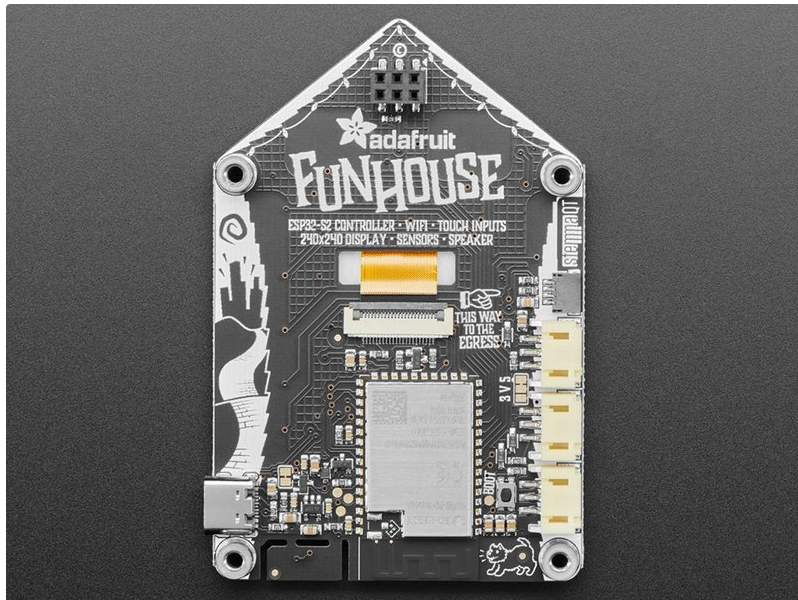
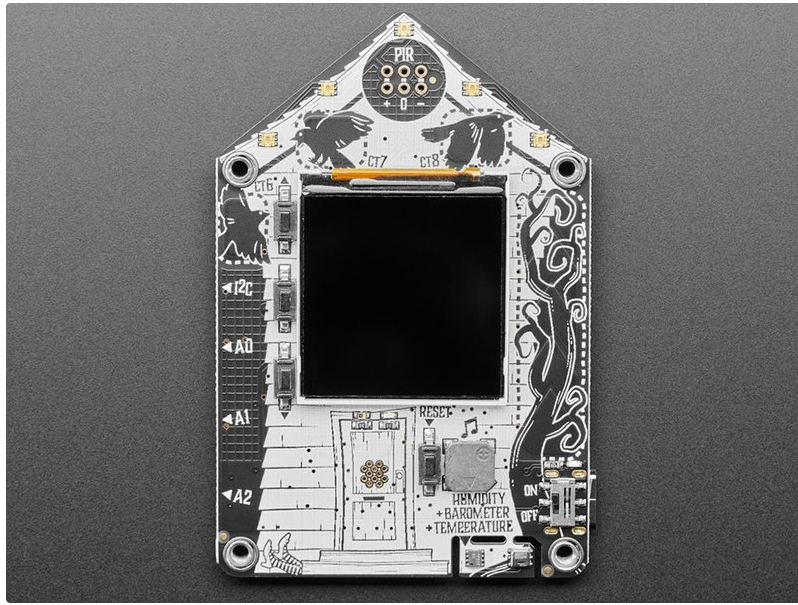
Here's what we included on this development board:

- **ESP32-S2 240MHz Tensilica processor** - the next generation of ESP32, now with native USB so it can act like a keyboard/mouse, MIDI device, disk drive, etc!
- **WROVER module** has FCC/CE certification and comes with 4 MByte of Flash and 2 MByte of PSRAM - you can have huge data buffers
- **1.54" Color TFT display with 240x240 pixels (https://adafru.it/RKc)**. This petite display is one of our favorites, with SPI interface and a controllable backlight.
- **USB C** power and data connector
- **Five mini RGB DotStar LEDs** on the top, for animations or easily-visible notification
- **Three buttons** can be used to wake up the ESP32 from deep-sleep, or select different modes
- **DPS310 (https://adafru.it/RKd)** barometric pressure and temperature sensor
- **AHT20 (https://adafru.it/RKe)** relative humidity and temperature sensor
- Plug in socket for **Mini PIR sensor (https://adafru.it/RKf)** (not included)
- **Front facing light sensor**
- **Speaker/Buzzer** can play tones and beeps for audible notification.
- **STEMMA QT port** for [attaching all sorts of I2C devices \(https://adafru.it/HMF\)](https://adafru.it/HMF)
- **Three STEMMA 3 pin JST connectors** for attaching [NeoPixels \(https://adafru.it/Cup\)](https://adafru.it/Cup), [speakers \(https://adafru.it/Gpf\)](https://adafru.it/Gpf), [servos \(https://adafru.it/FWq\)](https://adafru.it/FWq) or [relays \(https://adafru.it/NmC\)](https://adafru.it/NmC).
- **Three capacitive touch pads** and one **capacitive touch slider** with 5 elements.
- **On/Off switch**

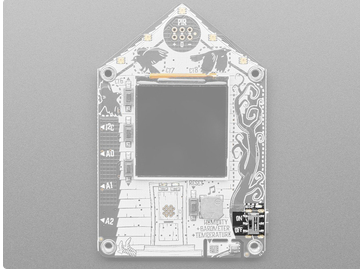
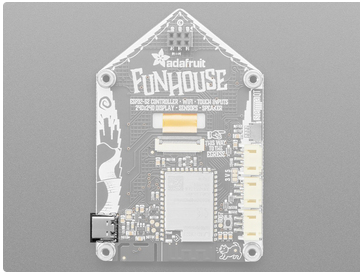
- **Boot and Reset buttons** for re-programming



Pinouts

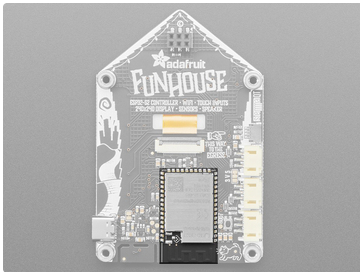


The Funhouse has a display and so many great features. It's packed with buttons, lights, connectors and sensors. Time to take a tour!



- **USB C port** - This is used for both powering and programming the board. You can power it with any USB C cable and will request 5V from a USB C PD.
- **On/Off switch** - This switch controls power to the board. If you plug in your board and nothing happens, make sure the switch is flipped to "ON"!

ESP32-S2 WiFi Module



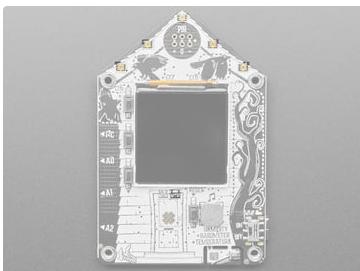
The **ESP32-S2 WROVER** module.

The ESP32-S2 is a highly-integrated, low-power, 2.4 GHz Wi-Fi System-on-Chip (SoC) solution that now has **built-in native USB** as well as some other interesting new technologies like Time of Flight distance measurements. With its state-of-the-art power and RF performance, this SoC is an ideal choice for a wide variety of application scenarios relating to the [Internet of Things \(IoT\)](https://adafru.it/Bwq) (<https://adafru.it/Bwq>), [wearable electronics](https://adafru.it/Osb) (<https://adafru.it/Osb>), and smart homes.

Please note, this is a single-core 240 MHz chip, so it won't be as fast as ESP32's with dual-core. Also, there is no Bluetooth support. However, we are super excited about the ESP32-S2's native USB which unlocks a lot of capabilities for advanced interfacing! This **WROVER** module comes with **4 MB flash** and **2 MB PSRAM**.

The 4 MB of flash is inside the module and is used for **both** program firmware and filesystem storage. For example, in CircuitPython, we have 3 MB set aside for program firmware (this includes two OTA option spots as well) and a 1MB section for CircuitPython scripts and files.

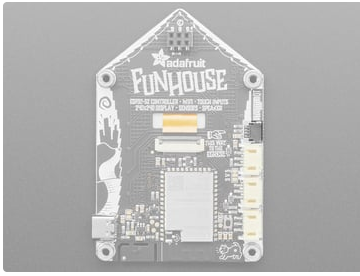
DotStar LEDs and Red LED



DotStar LEDs and red LED.

- On the front of the board, along the top, are **five addressable RGB DotStar LEDs**, so you can light up the display with any color or pattern. You can use **DOTSTAR_CLOCK** or **GPIO15** for the Clock pin and **DOTSTAR_DATA** or **GPIO14** for the Data pin to control the DotStars.
- Below the display, and slightly to the right, is a **red LED** positioned at the top right of the FunHouse door. It is user-controllable for blinky needs. You can blink this at any time. This LED is attached to **LED** or **GPIO37**.

STEMMA QT

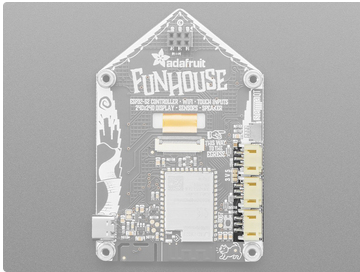


STEMMA QT (<https://adafru.it/Ft4>) - This JST SH 4-pin connector breaks out I2C (SCL, SDA, 3.3V, GND). It allows you to connect to **various breakouts and sensors with STEMMA QT connectors** (<https://adafru.it/HMF>) or to other things using **assorted associated accessories** (<https://adafru.it/Ft6>).

Works great with any STEMMA QT or Qwiic sensor/device

You can also use it with Grove I2C devices thanks to this handy cable (<https://adafru.it/Ndk>)

Digital/Analog Connectors



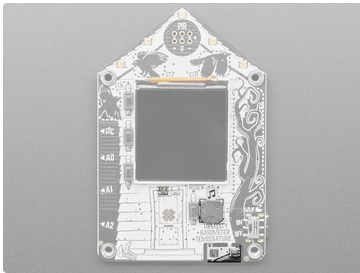
On the side are three connectors labeled **A0**, **A1**, and **A2**. These are **STEMMA 3 pin JST digital or analog connectors** for attaching **NeoPixels** (<https://adafru.it/Cup>), **speakers** (<https://adafru.it/Gpf>), **servos** (<https://adafru.it/HMF>). These pins can be analog inputs or digital I/O. They are connected to **GPIO17** for **A0**, **GPIO2** for **A1**, and **GPIO1** for **A2**.

All three connectors have protection 1K resistors + 3.6V zener diodes so you can drive an LED directly from the output. The maximum current from these connectors is 200mA.

All three ports are 'true' analog outputs and all three can be used for PWM as well as analog inputs. The maximum input voltage is 2.6V, after which the zener diodes will kick in to drain excess voltage.

The power output is 5V by default, but a jumper can be cut/soldered to change it to 3.3V.

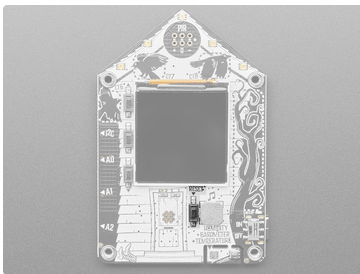
Speaker and Sensors



- Towards the middle of the board, at the bottom right corner of the display, is a **speaker/buzzer** labeled with a musical note. This includes a mini class D amplifier on DAC output **GPIO42** and can play tones or lo-fi audio clips.
- In the bottom right corner of the board, on the left side of the cutout region, is a **DPS310 pressure sensor**, that can be used to sense the barometric pressure. It is connected to the I2C port and available on I2C address **0x77**.
- Also in the bottom right corner of the board, on the right side of the cutout region, is an **AHT20 Humidity and Temperature sensor**, that can be used to sense the humidity and temperature. It is connected to the I2C port and available on I2C address **0x38**.
- Below the display, and slightly to the left, is a front-facing **light sensor** that is positioned at the top left of the FunHouse door.

Note: The light sensor is influenced by the display's backlight, use some tape to block the light if needed. More info, and a great chart [here](https://adafru.it/doW) (<https://adafru.it/doW>).

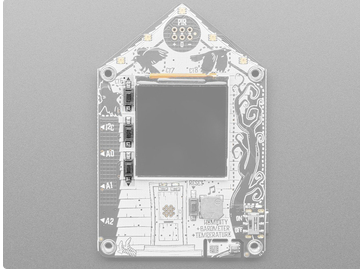
Reset and Boot Buttons



- **Reset button** - The reset button is on the front below the display and to the right of the FunHouse Door.
- **Boot button** - The boot button is on the back and situated between the Digital/Analog Ports and the ESP32-S2 module. This is connected to **BOOT0** and can be used to put the board into ROM bootloader mode. To enter ROM bootloader mode, **hold down DFU button while clicking reset button mentioned above**. When in the ROM bootloader, you can upload code and query the chip using **esptool**.



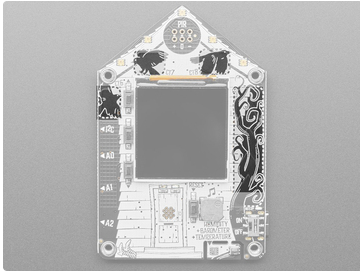
User Buttons



On the front of the board, to the left of the display, there are **three user-controllable buttons** labeled arrows for the top and bottom buttons. The buttons are on **BUTTON_DOWN** or **GPIO3**, **BUTTON_SELECT** or **GPIO4**, and **BUTTON_UP** or **GPIO5**. They can be used to wake up the ESP32-S2 from deep-sleep, or however you want to use them.

There are no pull-ups on board, use internal pulldowns for these pins - when the buttons are pressed the IO pin labeled is set to **HIGH**

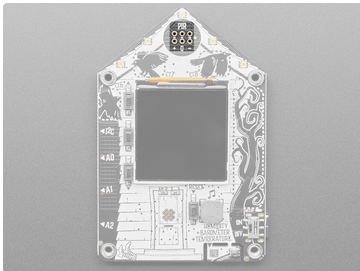
Capacitive Touch Pads and Slider



On the front of the board, to the left of the display and along the top, there are **three capacitive touch pads** with ravens on top labeled **CT6**, **CT7**, and **CT8**. The pads are on **CAP6** or **GPIO6**, **CAP7** or **GPIO7**, and **CAP8** or **GPIO8**. They can be used like buttons.

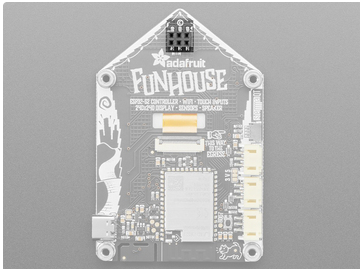
On the right side of the board, there is a **capacitive touch slider** made up of 5 capacitive touch pads. It has a tree on top. The pads are on **CAP9** or **GPIO9**, **CAP10** or **GPIO10**, **CAP11** or **GPIO11**, **CAP12** or **GPIO12**, and **CAP13** or **GPIO13**. They can be used as separate buttons or a positional slider.

PIR Sensor Port



On the front of the FunHouse is a location to add the PIR sensor. It is intended to be inserted through the front and into the connector on the back, but it could also work from the back. **When inserting, make sure the + and - symbols match up with the PIR sensor's markings or it will short out the board.**

The PIR sensor is connected to **PIR_SENSE** or **GPIO16**.



Install UF2 Bootloader

The FunHouse comes with a bootloader pre-installed, so you should only need to follow these steps if you are having issues.

If you're familiar with our other products and chipsets you may be familiar with our drag-n-drop bootloader, a.k.a UF2. We have a UF2 bootloader for the ESP32-S2, that will let you drag firmware on/off a USB disk drive.

Unlike the M0 (SAMD21) and M4 (SAMD51) boards, there is no bootloader protection for the UF2 bootloader. That means it is possible to erase or damage the bootloader, especially if you upload Arduino sketches to ESP32S2 boards that doesn't "know" there's a bootloader it should not overwrite!

However, thanks to the ROM bootloader, you don't have to worry about it if the UF2 bootloader is damaged. The ROM bootloader can never be disabled or erased, so it's always there if you need it! You can simply re-load the UF2 bootloader (USB-disk-style) with the ROM bootloader (non-USB-drive)

You can use the TinyUF2 bootloader to load code directly, say CircuitPython or the binary output of an Arduino compilation or you can use it to load a *second* bootloader on, like UF2 which has a drag-n-drop interface.

Installing the UF2 bootloader will erase your board's firmware which is also used for storing CircuitPython/Arduino/Files! Be sure to back up your data first.

Step 1. Download the tinyuf2_combo.bin file here

Note that this file is 3MB but that's because the bootloader is near the end of the available flash. It's not actually 3MB large, most of the file is empty but it's easier to program if we give you one combined 'swiss cheese' file. Save this file to your desktop or wherever you plan to run esptool from

This is a link to the latest .zip files. Just click the one for the FunHouse.

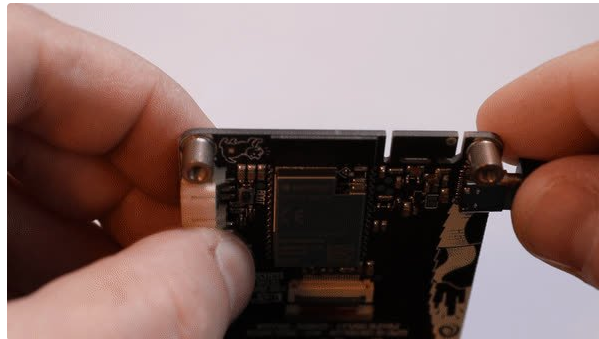
<https://adafru.it/TSA>

<https://adafru.it/TSA>

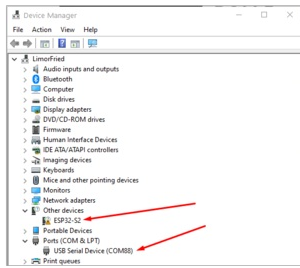
Step 2. Place your board in bootloader mode

Entering the bootloader is easy. Complete the following steps.

1. **Make sure your ESP32-S2 is plugged into USB port to your computer using a data/sync cable.** Charge-only cables will not work!
2. **Turn on the On/Off switch** - check that you see the OK light on so you know the board is powered, a prerequisite!
3. **Press and hold the DFU / Boot0 button down.** Don't let go of it yet!
4. **Press and release the Reset button.** You should have the DFU/Boot0 button pressed while you do this.
5. **Now you can release the DFU / Boot0 button**

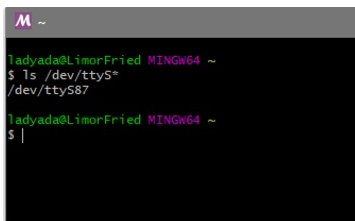


Check for a new serial / COM port



On Windows check the Device manager - you will see a COM port, for example here its COM8. You may also see another "Other device" called ESP32-S2

It's best to do this with no other dev boards plugged in so you don't get confused about which COM port is the ESP32-S2

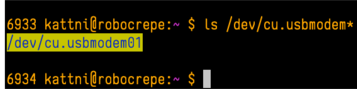


On Mac/Linux you will need to find the tty name which lives under /dev

On Linux, try `ls /dev/ttyS*` for example, to find the matching serial port name. In this case it shows up as `/dev/ttyS87`. If you don't see it listed try `ls /dev/ttyA*` on some Linux systems it might show up like `/dev/ttyACMO`

On Mac, try `ls /dev/cu.usbmodem*` for example, to find the matching serial port name. In this case, it shows up as `/dev/cu.usbmodem01`

It's best to do this with no other dev boards plugged in so you don't get confused about which serial port is the ESP32-S2



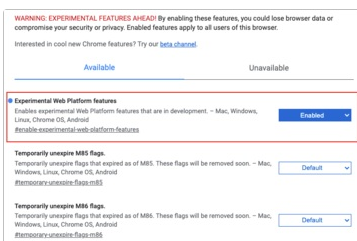
Step 3 Option A. Use the Web Serial ESPTool to upload

The WebSerial ESPTool was designed to be a web-capable option for programming ESP32-S2 boards. It allows you to erase the contents of the microcontroller and program up to 4 files at different offsets.

You will have to use the Chrome browser for this to work, Safari and Firefox, etc are *not* supported because we need Web Serial and only Chrome is supporting it to the level needed.

Enable Web Serial (For older chrome)

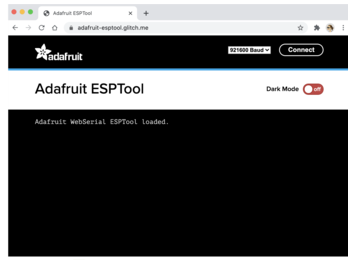
As of chrome 89, Web Serial is already enabled, so this step is only necessary on older browsers.



Visit `chrome://flags` from within Chrome. Find and enable the **Experimental Web Platform features**

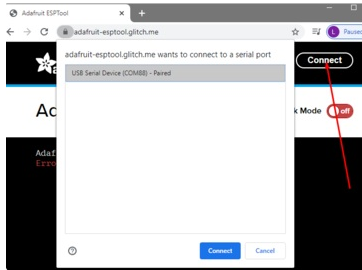
Restart Chrome

Connecting

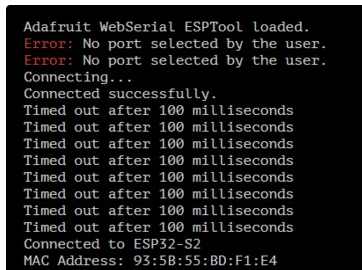


In the **Chrome browser**

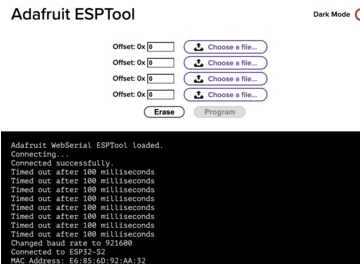
visit https://adafruit.github.io/Adafruit_WebSerial_ESPTool/ (<https://adafru.it/PMB>) it should look like the image to the left



Press the **Connect** button in the top right of the web browser. You will get a pop up asking you to select the COM or Serial port. You may want to remove all other USB devices so *only* the ESP32-S2 board is attached, that way there's no confusion over multiple ports!



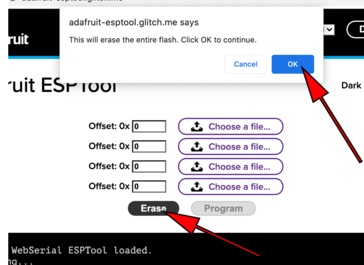
The Javascript code will now try to connect to the ROM bootloader. It may timeout for a bit until it succeeds. On success, you will see that it is **Connected** and will print out a unique **MAC** address identifying the board.



Once you have successfully connected, the command toolbar will appear.

Erasing the Contents

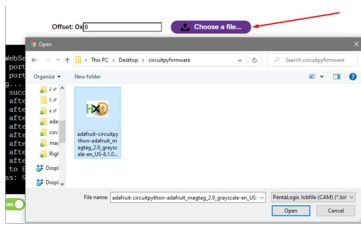
If you would like to erase the entire flash area so that you can start with a clean slate, you can use the erase feature. We recommend doing this if you are having issues.



To erase the contents, click the Erase button. You will be prompted whether you want to continue. Click OK to continue or if you changed your mind, just click cancel.

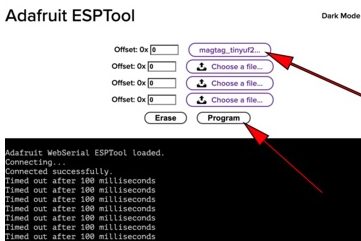
Programming the Microcontroller

Programming the microcontroller can be done with up to 4 files at different locations, but with the **tinyuf2combo BIN** file, which you should have downloaded under **Step 1** on this page, you only need to use 1 file.

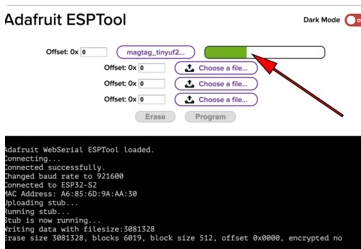


You can click on **Choose a file...** from any of the available buttons. It will only attempt to program buttons with a file and a unique location. Then select the Adafruit CircuitPython **BIN** files (not the UF2 file!)

Verify that the **Offset** box next to the file location you used is 0x0.



Once you choose a file, the button text will change to match your filename. You can then select the **Program** button to start flashing.



A progress bar will appear and after a minute or two, you will have written the firmware.

After using the tool, press the reset button to get out of bootloader mode and launch the new firmware!

Step 3. Option B. Use esptool.py to upload (for advanced users)

Once you have entered ROM bootloader mode, you can then [use Espressif's esptool program \(https://adafruit.it/E9p\)](https://adafruit.it/E9p) to communicate with the chip! **esptool** is the 'official' programming tool and is the most common/complete way to program an ESP chip.

Install ESPTool.py

You will need to use the command line / Terminal to install and run **esptool**.

You will also need to have pip and Python installed (any version!)

Install the latest version using pip (you may be able to run **pip** without the **3** depending on your setup):

```
pip3 install --upgrade esptool
```

Then, you can run:

```
esptool.py
```

Test the Installation

Run **esptool.py** in a new terminal/command line and verify you get something like the below:

```
C:\Users\ladyada>python3 C:\ESP32\esp-idf\components\esptool_py\esptool\esptool.py
esptool.py v3.0-dev
usage: esptool [-h] [--chip {auto,esp8266,esp32,esp32s2}] [--port PORT] [--baud BAUD]
              [--before {default_reset,no_reset,no_reset_no_sync}]
              [--after {hard_reset,soft_reset,no_reset}] [--no-stub] [--trace]
              [--override-vddsdio [{1.8V,1.0V,OFF}]] [--connect-attempts CONNECT_ATTEMPTS]
              [load_ram,dump_mem,read_mem,write_mem,write_flash,run,image_info,make_image,elf2image,read_mac,chip_id,flash_id,read_flash_status,write_flash_status,read_flash,verify_flash,erase_flash,erase_regi
on,version,get_security_info]
              ...
```

Make sure you are running esptool v3.0 or higher, which adds ESP32-S2 support

Run the following command, replacing the identifier after `--port` with the `COMxx`, `/dev/cu.usbmodemxx` or `/dev/ttySxx` you found above.

```
esptool.py --port COM88 chip_id
```

You should get a notice that it connected over that port and found an ESP32-S2

```
C:\Users\ladyada>python3 C:\ESP32\esp-idf\components\esptool_py\esptool\esptool.py --port COM88 chip_id
esptool.py v3.0-dev
Serial port COM88
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:00:3f:3e
Uploading stub...
Running stub...
Stub running...
Warning: ESP32-S2 has no Chip ID. Reading MAC instead.
MAC: 7c:df:a1:00:3f:3e
Hard resetting via RTS pin...
ERROR: ESP32-S2 chip was placed into download mode using GPIO0.
esptool.py can not exit the download mode over USB. To run the app, reset the chip manually.
To suppress this error, set --after option to 'no_reset'.
```

Installing the Bootloader

Run this command and replace the serial port name with your matching port and the file you just downloaded

```
esptool.py --port COM88 write_flash 0x0 tinyuf2_combo.bin
```

Don't forget to change the `--port` name to match.

There might be a bit of a 'wait' when programming, where it doesn't seem like it's working. Give it a minute, it has to erase the old flash code which can cause it to seem like it's not running.

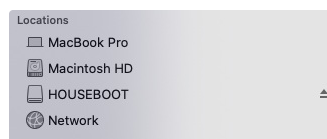
You'll finally get an output like this:

```
esptool.py v3.0-dev
Serial port COM88
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:05:f8:9a
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 3081264 bytes to 98937...
Wrote 3081264 bytes (98937 compressed) at 0x00000000 in 22.8 seconds (effective 1080.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
ERROR: ESP32-S2 chip was placed into download mode using GPIO0.
esptool.py can not exit the download mode over USB. To run the app, reset the chip manually.
To suppress this error, set --after option to 'no_reset'.
```

Step 4. Reset the board

Double-click the RESET button to launch the bootloader. **About a half second pause between clicks while the DotStars are purple seems to work well.** You'll see a new disk drive on your computer with the name HOUSEBOOT.



You're now ready to copy the CircuitPython UF2 on to the drive which will set up CircuitPython!

CircuitPython

[CircuitPython \(https://adafru.it/tB7\)](https://adafru.it/tB7) is a derivative of [MicroPython \(https://adafru.it/f9W\)](https://adafru.it/f9W) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

Set Up CircuitPython

Follow the steps to get CircuitPython installed on your FunHouse.

<https://adafru.it/RKB>

<https://adafru.it/RKB>

CircuitPython 6.2.0

This is the latest **stable** release of CircuitPython that will work with the FunHouse - WiFi Home Automation Development Board.

Start here if you are new to CircuitPython.

Release Notes for 6.2.0

ENGLISH (US)

DOWNLOAD .BIN NOW

DOWNLOAD .UF2 NOW

Built-in modules available: _bleio_, pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel, serial, sys, time, os, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi

Click the link above and download the latest .BIN and .UF2 file

(depending on how you program the ESP32S2 board you may need one or the other, might as well get both)

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



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

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

Option 1 - Load with UF2 Bootloader

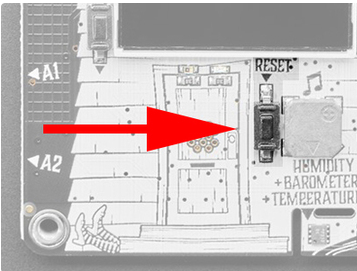
This is by far the easiest way to load CircuitPython. However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.

Still, try this first!



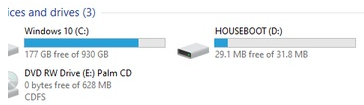
Try Launching UF2 Bootloader

Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it.

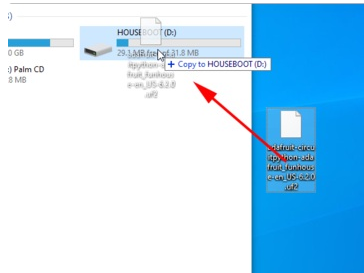


Launch UF2 by **double-clicking** the Reset button (the one next to the USB C port). You may have to try a few times to get the timing right.

About a half second pause between clicks while the DotStars are purple seems to work well.

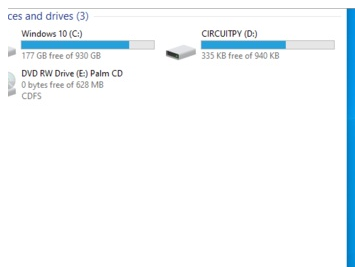


If the UF2 bootloader is installed, you will see a new disk drive appear called **HOUSEBOOT**



Copy the **UF2** file you downloaded at the first step of this tutorial onto the **HOUSEBOOT** drive

If you're using Windows and you get an error at the end of the file copy that says **Error from the file copy, Error 0x800701B1: A device which does not exist was specified.** You can ignore this error, the bootloader sometimes disconnects without telling Windows, the install completed just fine and you can continue. [If its really annoying, you can also upgrade the bootloader \(the latest version of the UF2 bootloader fixes this warning\) \(https://adafru.it/RLc\)](https://adafru.it/RLc)



Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

Option 2 - Use Chrome Browser To Upload BIN file

You will need to do a full erase prior to uploading new firmware.

The next best option is to try using the Chrome-browser version of esptool we have written. This is handy if you don't have Python on your computer, or something is really weird with your setup that makes esptool not run (which happens sometimes and isn't worth debugging!) You can follow along on the [Install UF2 Bootloader \(https://adafru.it/RLc\)](https://adafru.it/RLc) page and either load the UF2 bootloader and then come back to Option 1 on this page, or you can download the CircuitPython BIN file directly using the tool in the same manner as the bootloader.

Option 3 - Use esptool to load BIN file

For more advanced users, you can upload with **esptool** to the ROM (hardware) bootloader instead!

```

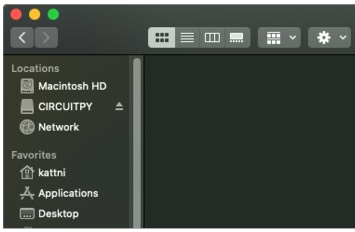
$ python3 esptool.py --port /dev/ttyUSB0 --after-reset
write_flash 0x0 - /adafruit-circuitpython-adafruit-astro-esp32-en_US-20201103-567925.bin
python3.py v1.14.0
Serial port /dev/ttyUSB0
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of eFuse
Crystal is 40MHz
MCU: esp32-s1-00-4a2
Uploading stub...
Bootloading stub...
Stub running...
Self-programming (flash chip)...
Compressed 138184 bytes to 844814...
Wrote 138184 bytes (844814 compressed) at 0x00000000 in 11.9 seconds (effective 978.2 kbit/s)...
Hash of data verified.
Erasing...
Erasing in bootloader.

```

Follow the initial steps found in the [Run esptool and check connection section of the Install UF2 Bootloader page \(https://adafru.it/RLc\)](https://adafru.it/RLc) to verify your environment is set up, your board is successfully connected, and which port it's using.

In the final command to write a binary file to the board, replace the port with your port, and replace "firmware.bin" with the file you downloaded above.

The output should look something like the output in the image.



Press reset to exit the bootloader.

Your **CIRCUITPY** drive should appear!

You're all set! Go to the next pages.

CircuitPython Internet Libraries

To use the internet-connectivity built into your ESP32-S2 with CircuitPython, you must first install a number of libraries. This page covers that process.

Adafruit CircuitPython Library Bundle

Download the Adafruit CircuitPython Bundle. You can find the latest release here:

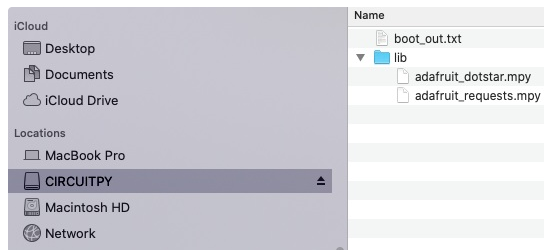
<https://adafru.it/ENC>

<https://adafru.it/ENC>

Download the **adafruit-circuitpython-bundle-version-mpy-*.zip** bundle zip file, and unzip a folder of the same name. Inside you'll find a **lib** folder. The entire collection of libraries is too large to fit on the **CIRCUITPY** drive. Instead, add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into **CIRCUITPY/lib** now

- **adafruit_requests.mpy** - A requests-like library for HTTP commands.
- **adafruit_dotstar.mpy** - Helper library to use DotStar LEDs, often built into the boards so they're great for quick feedback



Once you have added those files, please continue to the next page to set up and test Internet connectivity

CircuitPython Internet Test

Once you have CircuitPython installed and the minimum libraries installed we can get your board connected to the Internet.

To get connected, you will need to start by creating a `secrets.py` file.

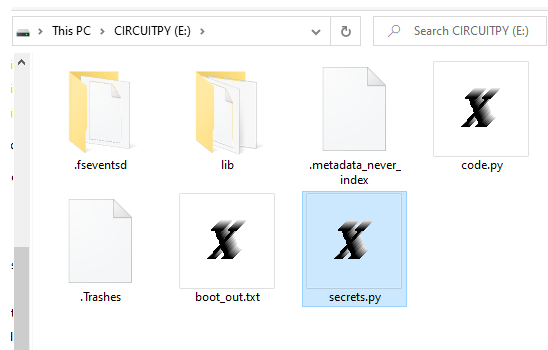
Secrets File

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a `secrets.py` file, that is in your `CIRCUITPY` drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your `secrets.py` file should look like this:

```
# This file is where you keep secret settings, passwords, and tokens!  
# If you put them in the code you risk committing that info or sharing it  
  
secrets = {  
    'ssid' : 'home_wifi_network',  
    'password' : 'wifi_password',  
    'aio_username' : 'my_adafruit_io_username',  
    'aio_key' : 'my_adafruit_io_key',  
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones  
}
```

Copy and paste that text/code into a file called `secrets.py` and save it to your `CIRCUITPY` folder like so:



Inside is a python dictionary named `secrets` with a line for each entry. Each entry has an entry name (say `'ssid'`) and then a colon to separate it from the entry key `'home ssid'` and finally a comma ,

At a minimum you'll need to adjust the `ssid` and `password` for your local WiFi setup so do that now!

As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing github or the hackaday API. Other non-secret data like your timezone can also go here, just cause its called secrets doesn't mean you can't have general customization data in there!

For the correct time zone string, look at <http://worldtimeapi.org/timezones> (<https://adafru.it/EcP>) and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your `secrets.py` - keep that out of GitHub, Discord or other project-sharing sites.

Don't share your `secrets.py` file, it has your passwords and API keys in it!

Connect to WiFi

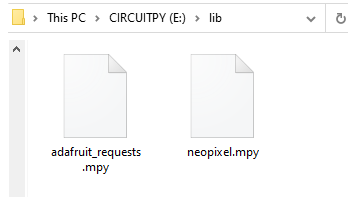
OK now you have your secrets setup - you can connect to the Internet using the Requests module.

First make sure you are running the [latest version of Adafruit CircuitPython](https://adafru.it/Amd) (<https://adafru.it/Amd>) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from [Adafruit's CircuitPython library bundle](https://adafru.it/zdx) (<https://adafru.it/zdx>). Our introduction guide has [a great page on how to install the library bundle](https://adafru.it/ABU) (<https://adafru.it/ABU>).

- `adafruit_requests`
- `neopixel`

Before continuing make sure your board's `CIRCUITPY/lib` folder or root filesystem has the above files copied over.



Once that's done, load up the following example using Mu or your favorite editor:

```
import ipaddress
import ssl
import wifi
import socketpool
import adafruit_requests

# URLs to fetch from
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_QUOTES_URL = "https://www.adafruit.com/api/quotes.php"
JSON_STARS_URL = "https://api.github.com/repos/adafruit/circuitpython"

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

print("ESP32-S2 WebClient Test")

print("My MAC addr:", [hex(i) for i in wifi.radio.mac_address])

print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
        network.rssi, network.channel))
wifi.radio.stop_scanning_networks()

print("Connecting to %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!"%secrets["ssid"])
print("My IP address is", wifi.radio.ipv4_address)

ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com: %f ms" % (wifi.radio.ping(ipv4)*1000))

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

print("Fetching text from", TEXT_URL)
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)

print("Fetching json from", JSON_QUOTES_URL)
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)

print()

print("Fetching and parsing json from", JSON_STARS_URL)
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print("CircuitPython GitHub Stars", response.json()["stargazers_count"])
print("-" * 40)

print("done")
```

And save it to your board. Make sure the file is named `code.py`.

Open up your REPL, you should see something like the following:


```

1: screen /Users/brentubell (screen)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
ESP32-S2 WebClient Test
My MAC addr: ['0x7c', '0xdf', '0xa1', '0x0', '0x52', '0xa0']
Avaliable WiFi networks:
  Brunelleschi      RSSI: -84      Channel: 6
  Transit           RSSI: -54      Channel: 1
  Fios-5dLNb        RSSI: -66      Channel: 1
  disconnectededer   RSSI: -86      Channel: 1
  SKJFios-ZV007     RSSI: -83      Channel: 11
  Fios-QIVUQ        RSSI: -83      Channel: 11
  Fios-ZV007        RSSI: -85      Channel: 11
  [REDACTED]         RSSI: -58      Channel: 2
  [REDACTED]         RSSI: -76      Channel: 8
  NETGEAR52         RSSI: -81      Channel: 10
Connecting to Transit
Connected to Transit!
None
My IP address is 192.168.1.182
Ping google.com: 0.065000 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
-----
This is a test of Adafruit WiFi!
If you can read this, its working :)
-----
Fetching json from https://www.adafruit.com/api/quotes.php
-----
[{'text': 'Science, my lad, is made up of mistakes, but they are mistakes which it is u
seful to make, because they lead little by little to the truth', 'author': 'Jules Verne
'}]
-----
Fetching and parsing json from https://api.github.com/repos/adafruit/circuitpython
-----
CircuitPython GitHub Stars 1896
-----
done

```

In order, the example code...

Checks the ESP32-S2's MAC address.

```
print("My MAC addr:", [hex(i) for i in wifi.radio.mac_address])
```

Performs a scan of all access points and prints out the access point's name (SSID), signal strength (RSSI), and channel.

```
print("Avaliable WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
        network.rssi, network.channel))
wifi.radio.stop_scanning_networks()
```

Connects to the access point you defined in the `secrets.py` file, prints out its local IP address, and attempts to ping google.com to check its network connectivity.

```
print("Connecting to %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print(print("Connected to %s!"%secrets["ssid"]))
print("My IP address is", wifi.radio.ipv4_address)

ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com: %f ms" % wifi.radio.ping(ipv4))
```

The code creates a socketpool using the wifi radio's available sockets. This is performed so we don't need to re-use sockets. Then, it initializes a new instance of the [requests \(https://adafruit.it/E9o\)](https://adafruit.it/E9o) interface - which makes getting data from the internet *really really easy*.

```
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
```

To read in plain-text from a web URL, call `requests.get` - you may pass in either a http, or a https url for SSL connectivity.

```
print("Fetching text from", TEXT_URL)
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)
```

Requests can also display a JSON-formatted response from a web URL using a call to `requests.get`.

```
print("Fetching json from", JSON_QUOTES_URL)
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)
```

Finally, you can fetch and parse a JSON URL using `requests.get`. This code snippet obtains the `stargazers_count` field from a call to the GitHub API.

```
print("Fetching and parsing json from", JSON_STARS_URL)
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print("CircuitPython GitHub Stars", response.json()["stargazers_count"])
print("-" * 40)
```

OK you now have your ESP32-S2 board set up with a proper `secrets.py` file and can connect over the Internet. If not, check that your `secrets.py` file has the right ssid and password and retrace your steps until you get the Internet connectivity working!

Getting The Date & Time

A very common need for projects is to know the current date and time. Especially when you want to deep sleep until an event, or you want to change your display based on what day, time, date, etc. it is

Determining the correct local time is really really hard. There are various time zones, Daylight Savings dates, leap seconds, etc. Trying to get NTP time and then back-calculating what the local time is, is extraordinarily hard on a microcontroller just isn't worth the effort and it will get out of sync as laws change anyways.

For that reason, we have the free adafruit.io time service. **Free for anyone, with a free adafruit.io account.** You *do need an account* because we have to keep accidentally mis-programmed-board from overwhelming adafruit.io and lock them out temporarily. Again, it's free!

There are other services like WorldTimeAPI, but we don't use those for our guides because they are nice people and we don't want to accidentally overload their site. Also, there's a chance it may eventually go down or also require an account.

Step 1) Make an Adafruit account

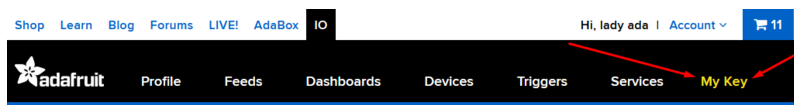
It's free! Visit <https://accounts.adafruit.com/> (<https://adafru.it/dyy>) to register and make an account if you do not already have one

Step 2) Sign into Adafruit IO

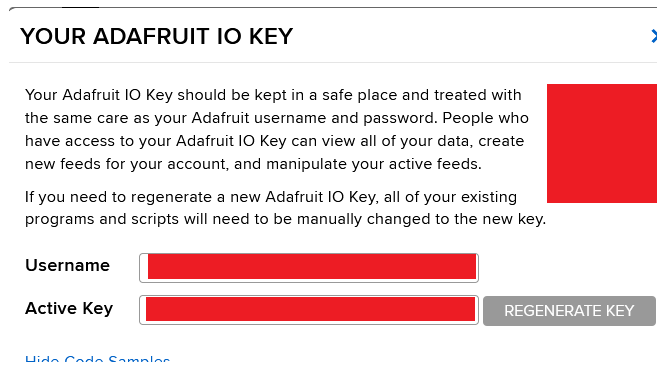
Head over to io.adafruit.com (<https://adafru.it/fsU>) and click **Sign In** to log into IO using your Adafruit account. It's free and fast to join.

Step 3) Get your Adafruit IO Key

Click on **My Key** in the top bar



You will get a popup with your **Username** and **Key** (In this screenshot, we've covered it with red blocks)



Go to your secrets.py file on your CIRCUITPY drive and add three lines for `aio_username`, `aio_key` and `timezone` so you get something like the following:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid' : 'home_wifi_network',
    'password' : 'wifi_password',
    'aio_username' : 'my_adafruit_io_username',
    'aio_key' : 'my_adafruit_io_key',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
}
```

The timezone is optional, if you don't have that entry, adafruit.io will guess your timezone based on geographic IP address lookup. You can visit <http://worldtimeapi.org/timezones> (<https://adafru.it/EcP>) to see all the time zones available (even though we do not use worldtimeapi for time-keeping we do use the same time zone table)

Step 4) Upload Test Python Code

This code is like the Internet Test code from before, but this time it will connect to adafruit.io and get the local time

```

import ipaddress
import ssl
import wifi
import socketpool
import adafruit_requests
import secrets

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_QUOTES_URL = "https://www.adafruit.com/api/quotes.php"
JSON_STARS_URL = "https://api.github.com/repos/adafruit/circuitpython"

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# Get our username, key and desired timezone
aio_username = secrets["aio_username"]
aio_key = secrets["aio_key"]
location = secrets.get("timezone", None)
TIME_URL = "https://io.adafruit.com/api/v2/%s/integrations/time/strftime?x-aio-key=%s" % (aio_username, aio_key)
TIME_URL += "&fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+%25Z"

print("ESP32-S2 Adafruit IO Time test")

print("My MAC addr:", [hex(i) for i in wifi.radio.mac_address])

print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
        network.rssi, network.channel))
wifi.radio.stop_scanning_networks()

print("Connecting to %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!"%secrets["ssid"])
print("My IP address is", wifi.radio.ipv4_address)

ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com: %f ms" % wifi.radio.ping(ipv4))

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

print("Fetching text from", TIME_URL)
response = requests.get(TIME_URL)
print("-" * 40)
print(response.text)
print("-" * 40)

```

After running this, you will see something like the below text. We have blocked out the part with the secret username and key data!

```

Connecting to adafruit
Connected to adafruit!
My IP address is 10.0.1.148
Ping google.com: 0.008000 ms
Fetching text from https://io.adafruit.com/api/v2/[REDACTED]/integrations/time/strftime?x-aio-
key=[REDACTED]&fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+%25Z
-----
2020-12-05 18:51:32.145 340 6 -0500 EST
-----

```

Note at the end you will get the date, time, and your timezone! If so, you have correctly configured your `secrets.py` and can continue to the next steps!

FunHouse-Specific CircuitPython Libraries

To use all the amazing features of your FunHouse with CircuitPython, you must first install a number of libraries. This page covers that process.

Get Latest Adafruit CircuitPython Bundle

Download the Adafruit CircuitPython Library Bundle. You can find the latest release here:

<https://adafru.it/ENC>

<https://adafru.it/ENC>

Download the **adafruit-circuitpython-bundle-version-mpy-*.zip** bundle zip file, and unzip a folder of the same name. Inside you'll find a **lib** folder. The entire collection of libraries is too large to fit on the **CIRCUITPY** drive. Therefore, you'll need to copy the necessary libraries to your board individually.

At a minimum, the following libraries are required. Copy the following folders or .mpy files to the **lib** folder on your **CIRCUITPY** drive. **If the library is a folder, copy the entire folder** to the **lib** folder on your board.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into **CIRCUITPY/lib** now

Library folders (copy the whole folder over to lib):

- **adafruit_funhouse** - This is a helper library designed for using all of the features of the FunHouse, including networking, buttons, DotStars, etc.
- **adafruit_portalbase** - This library is the base library that **adafruit_funhouse** is built on top of.
- **adafruit_bitmap_font** - There is fancy font support, and it's easy to make new fonts. This library reads and parses font files.
- **adafruit_display_text** - This library displays text on the screen.
- **adafruit_io** - This library helps connect the FunHouse to our free data logging and viewing service
- **adafruit_minimqtt** - MQTT library required for communicating with the MQTT Server

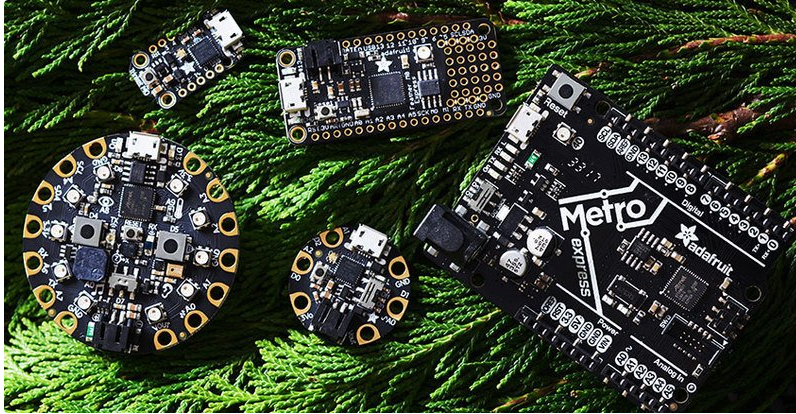
Library files:

- **adafruit_requests.mpy** - This library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- **adafruit_fakerequests.mpy** - This library allows you to create fake HTTP requests by using local files.
- **adafruit_mininqr.mpy** - QR creation library lets us add easy-to-scan 2D barcodes to the E-Ink display
- **adafruit_dotstar.mpy** - This library is used to control the onboard DotStars.
- **simpleio.mpy** - This library is used for tone generation.
- **adafruit_ahtx0.mpy** - This is used for the Humidity and Temperature Sensor
- **adafruit_dps310.mpy** - This is used for the Barometric Pressure Sensor

Secrets

Even if you aren't planning to go online with your FunHouse, you'll need to have a **secrets.py** file in the root directory (top level) of your **CIRCUITPY** drive. If you do not intend to connect to wireless, it does not need to have valid data in it. [Here's more info on the secrets.py file \(https://adafru.it/P3b\)](https://adafru.it/P3b).

Welcome To CircuitPython



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!

Installing the Mu Editor

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

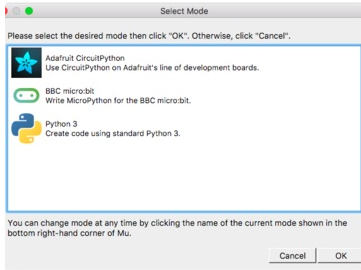
Mu is our recommended editor - please use it (unless you are an experienced coder with a favorite editor already!)

Download and Install Mu



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

Using Mu



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

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



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

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



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

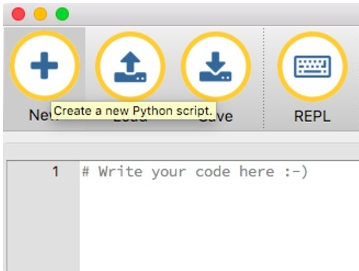
Creating and Editing Code

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

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

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

Creating Code



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

Copy and paste the following code into your editor:

```
import board
import digitalio
import time

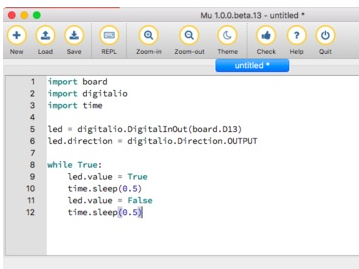
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

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

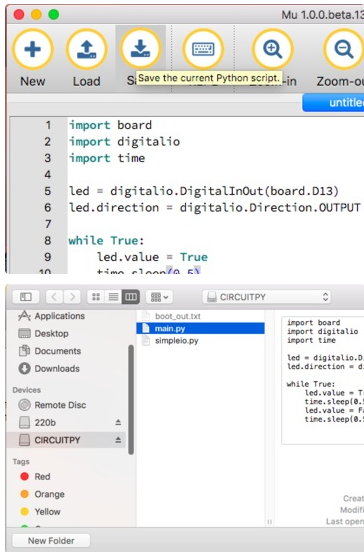
The QT Py and the Trinkeys do not have a built-in little red LED! There is an addressable RGB NeoPixel LED. The above example will NOT work on the QT Py or the Trinkeys!

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

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



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



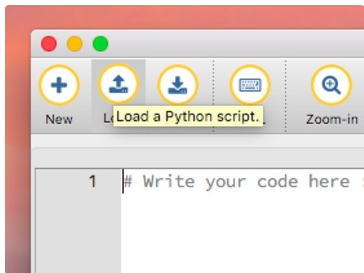
Save this file as `code.py` on your CIRCUITPY drive.

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

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

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

Editing Code



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

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

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

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

Don't Click Reset or Unplug!

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

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

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

There are a few ways to avoid this:

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

Recommended editors:

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

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

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

We *don't* recommend these editors:

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

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

2. Eject or Sync the Drive After Writing

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

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

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

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

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

Back to Editing Code...

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

```
import board
import digitalio
import time

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

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

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

Exploring Your First CircuitPython Program

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

Here is the original code again:

```
import board
import digitalio
import time

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

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

Imports & Libraries

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

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

The `import` statements tells the board that you're going to use a particular library in your code. In this example, we imported three modules: `board`, `digitalio`, and `time`. All three of these modules are built into CircuitPython, so no separate library files are needed. That's one of the things that makes this an excellent first example. You don't need 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.LED)
led.direction = digitalio.Direction.OUTPUT
```

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

Loop-de-loops

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

Inside our loop, we have four items:

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

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

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

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

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

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

What Happens When My Code Finishes Running?

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

For example, try reducing the above example to `led.value = True`. The LED will flash almost too quickly to see, and turn off. This is because the code finishes running and

resets the pin state, and the LED is no longer receiving a signal.

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

What if I don't have the loop?

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

The simplest loop would be:

```
while True:
```

```
    pass
```

And remember - you can press to exit the loop.

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

More Changes

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

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

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

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

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

Naming Your Program File

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

Connecting to the Serial Console

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

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

This line would result in:

```
Hello, world!
```

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

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

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

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

```
sudo apt purge modemmanager
```

Are you using Mu?

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

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



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

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



Setting Permissions on Linux

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



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

```
sudo adduser $USER dialout
```

After running the command above, reboot your machine to gain access to the group. On other Linux distributions, the group you need may be different. See [Advanced](#)

[Serial Console on Mac and Linux \(https://adafru.it/AAI\)](https://adafru.it/AAI) for details on how to add yourself to the right group.

Using Something Else?

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

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

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

Interacting with the Serial Console

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

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

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

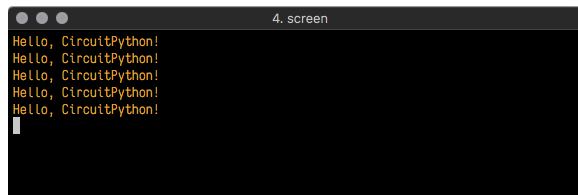
```
import board
import digitalio
import time

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

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

Save your file.

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



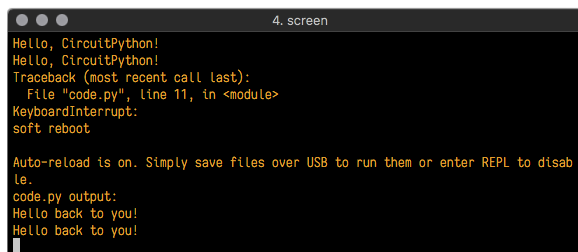
```
4. screen
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
```

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



```
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 = True
11    time.sleep(1)
12    led.value = False
13    time.sleep(1)
14
```

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!

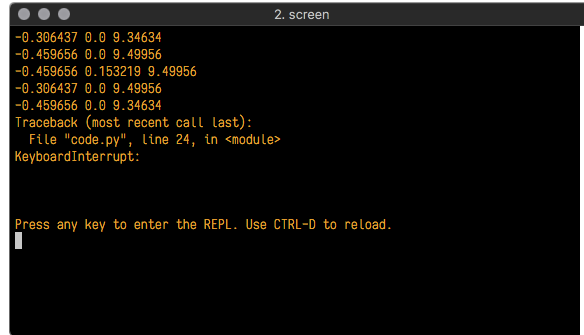
The REPL

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.
█
```

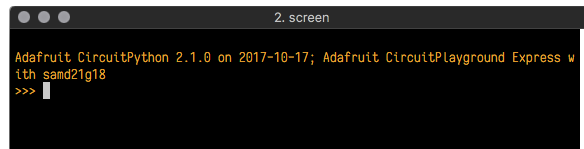
If there is no code running, you will enter the REPL immediately after pressing Ctrl + C. There is no information about what your board was doing before you interrupted it because there is no code running.



```
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!



```
2. screen
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit CircuitPlayground Express with samd21g18
>>> █
```

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

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

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

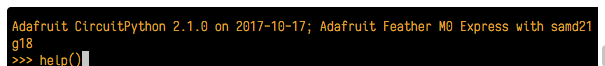
This line tells you the version of CircuitPython you're using and when it was released. Next, it gives you the type of board you're using and the type of microcontroller the board uses. Each part of this may be different for your board depending on the versions you're working with.

This is followed by the CircuitPython prompt.

```
>>> █
```

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

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



```
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Feather M0 Express with samd21g18
>>> help() █
```

Then press enter. You should then see a message.

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

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

Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Circuit Playground Express with samd21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

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

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

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

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

```
3. screen
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Feather M0 Express with samd21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

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

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

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

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

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

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

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

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

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

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

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

Then press enter.

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

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

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

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

Try typing more into the REPL to see what happens!

Returning to the serial console

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

You can return to the REPL at any time!

Advanced Serial Console on Windows

Windows 7 Driver

If you're using Windows 7 (or 8 or 8.1), use the link below to download the driver package. You will not need to install drivers on Mac, Linux or Windows 10.

We *strongly* encourage you to upgrade to Windows 10 if you are still using Windows 7 or Windows 8 or 8.1. Windows 7 has reached end-of-life and no longer receives security updates. A free upgrade to Windows 10 is [still available \(https://adafru.it/RWc\)](https://adafru.it/RWc).

<https://adafru.it/AB0>

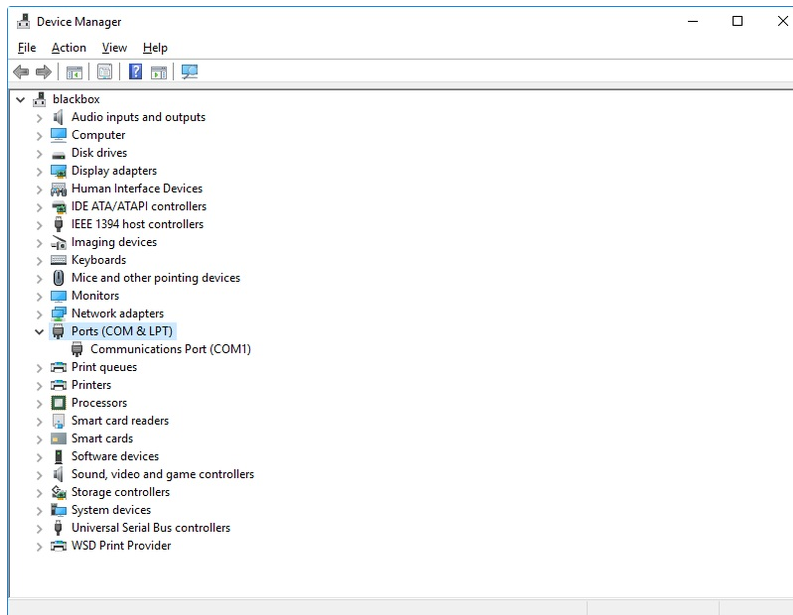
<https://adafru.it/AB0>

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

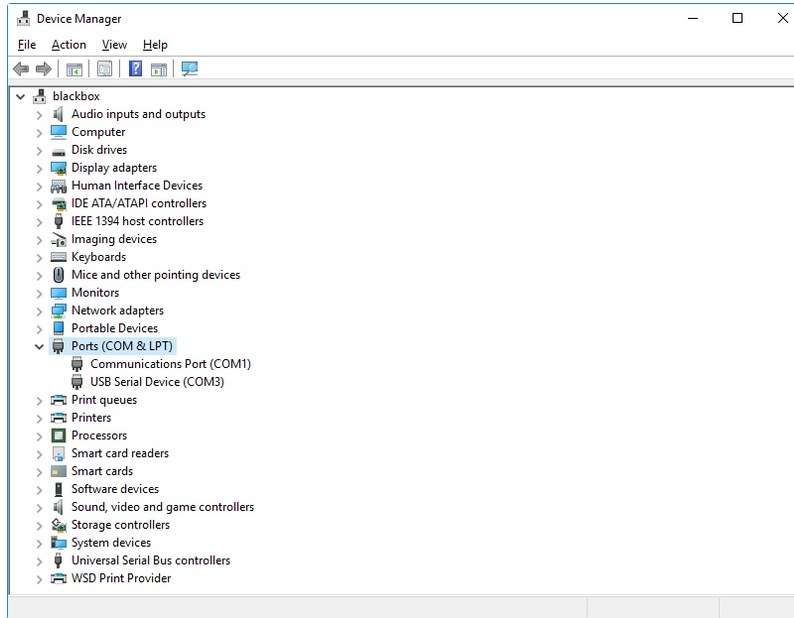
What's the COM?

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

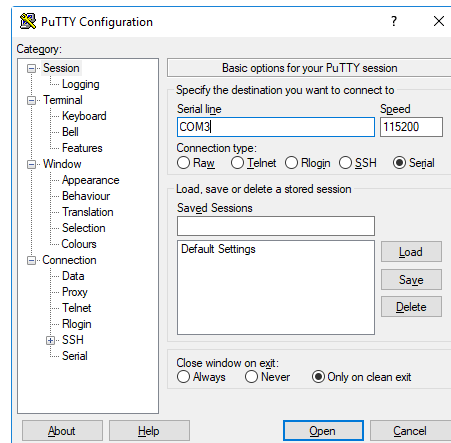
If you're using Windows, you'll need to download a terminal program. We're going to use PuTTY.

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

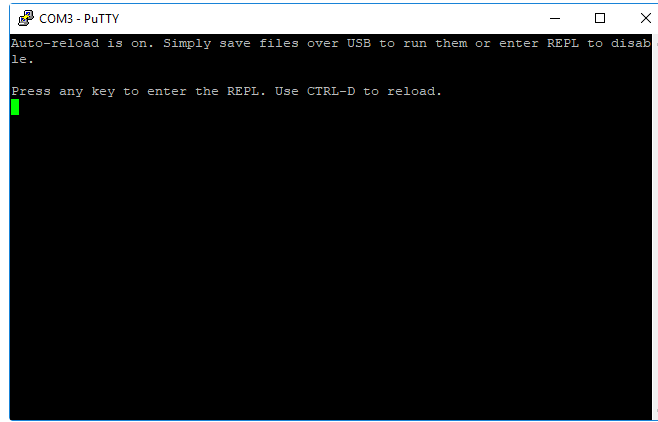
Now you need to open PuTTY.

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

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



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.

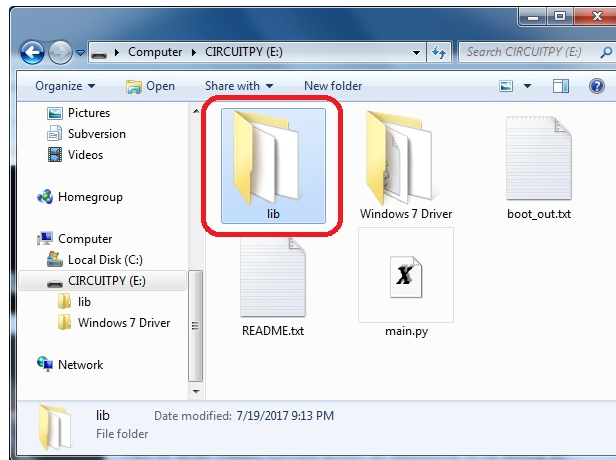
Great job! You've connected to the serial console!

CircuitPython Libraries

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

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

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



CircuitPython libraries work in the same way as regular Python modules so the [Python docs \(https://adafru.it/rar\)](https://adafru.it/rar) are a great reference for how it all should work. In Python terms, we can place our library files in the **lib** directory because 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 Bundle release by clicking the button below.

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

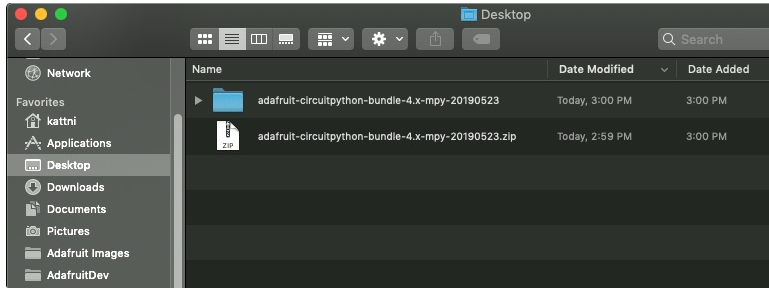
<https://adafru.it/ENC>

<https://adafru.it/ENC>

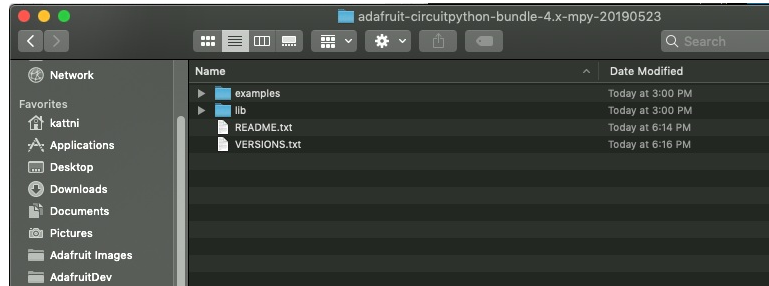
If you need another version, [you can also visit the bundle release page \(https://adafru.it/Ayy\)](https://adafru.it/Ayy) which will let you select exactly what version you're looking for, as well as information about changes.

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

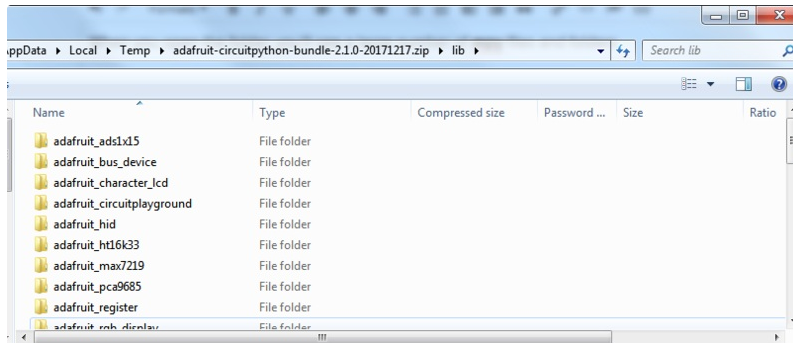
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



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



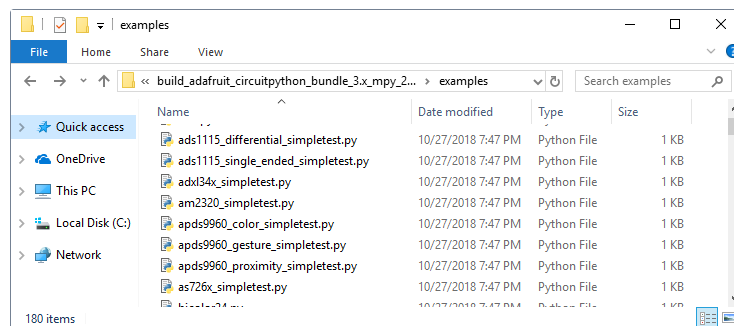
Now open the lib folder. When you open the folder, you'll see a large number of `mpy` files and folders



Example Files

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

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



Copying Libraries to Your Board

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

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

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

Example: `ImportError` Due to Missing Library

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

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

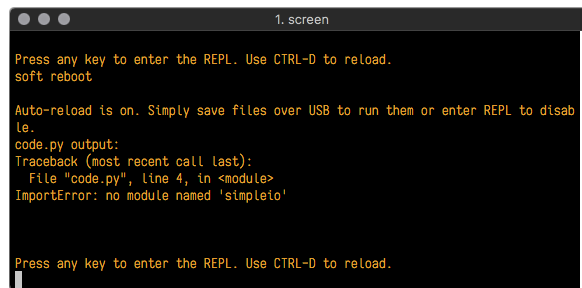
Let's use a modified version of the blinky example.

```
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.

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

Updating CircuitPython Libraries/Examples

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

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

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

CircuitPython Pins and Modules

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

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

CircuitPython Pins

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

import board

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

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

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

Here is the output for the QT Py.

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

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

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

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

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

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

```
>>> import board
>>> dir(board)
['_class_', 'A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'DEBUG_RX', 'DEBUG_TX', 'I2C',
'I01', 'I010', 'I011', 'I012', 'I013', 'I014', 'I015', 'I016', 'I017', 'I018',
'I02', 'I021', 'I03', 'I033', 'I034', 'I035', 'I036', 'I037', 'I04', 'I042', 'I045',
'I05', 'I06', 'I07', 'I08', 'I09', 'LED', 'MISO', 'MOSI', 'NEOPIXEL', 'RX',
'SCK', 'SCL', 'SDA', 'SPI', 'TX', 'UART']
```

Note that most of the pins are named in an IO# style, such as `I01` and `I02`. Those pins on the physical board are labeled only with a number, so an easy way to know how to access them in CircuitPython, is to run those commands in the REPL and find the pin naming scheme.

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

I2C, SPI, and UART

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

What's a singleton? When you create an object in CircuitPython, you are *instantiating* (creating) it. Instantiating an object means you are creating an instance of the

object with the unique values that are provided, or "passed", to it.

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

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

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

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

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

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

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

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

What Are All the Available Names?

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

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

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

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

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

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

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

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

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

Microcontroller Pin Names

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

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

CircuitPython Built-In Modules

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

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

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

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

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

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

Advanced Serial Console on Mac and Linux

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

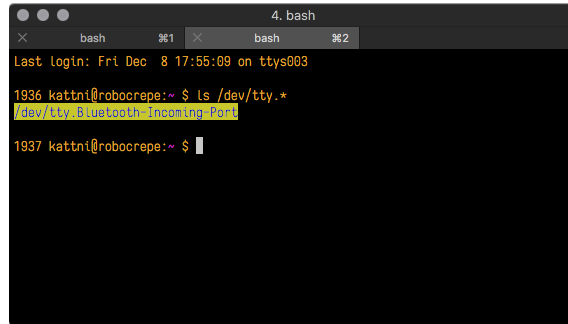
What's the Port?

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

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

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

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

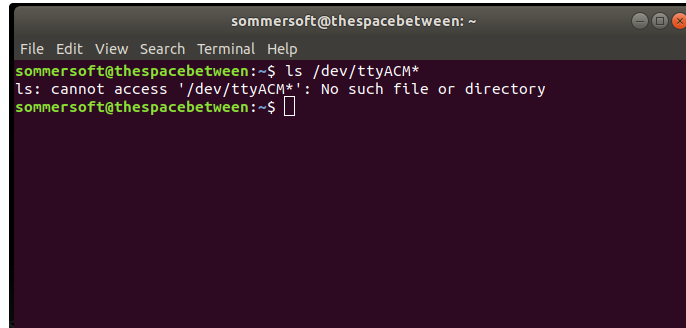


```
4. bash
bash %1 bash %2
Last login: Fri Dec 8 17:55:09 on ttys003
1936 kattni@robocrepe:~$ ls /dev/tty.*
dev/tty.Bluetooth-Incoming-Port
1937 kattni@robocrepe:~$
```

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`.

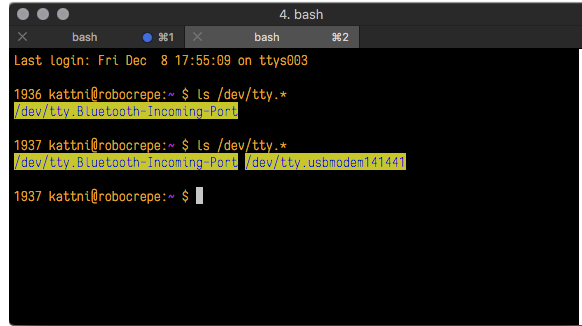


```
sommersoft@thespacebetween: ~
File Edit View Search Terminal Help
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
sommersoft@thespacebetween:~$
```

Now, plug your board. Using Mac, type:

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

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



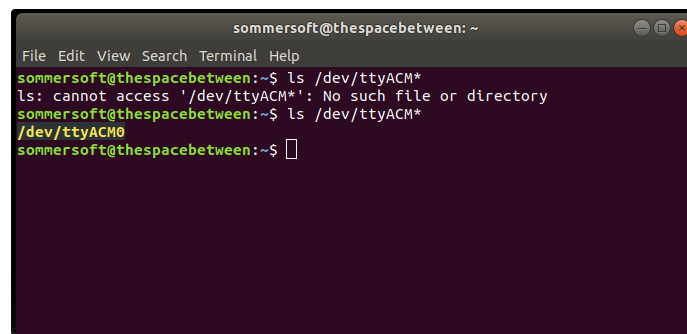
```
4. bash
bash %1 bash %2
Last login: Fri Dec 8 17:55:09 on ttys003
1936 kattni@robocrepe:~$ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port
1937 kattni@robocrepe:~$ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441
1937 kattni@robocrepe:~$
```

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.



```
sommersoft@thespacebetween: ~
File Edit View Search Terminal Help
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
/dev/ttyACM0
sommersoft@thespacebetween:~$
```

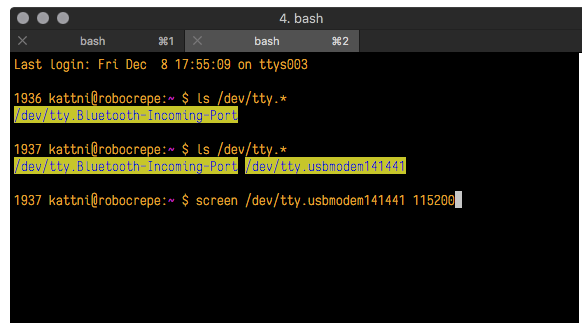
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

Connect with screen

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

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

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



```
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
```

```
sommersoft@thespacebetween: ~
File Edit View Search Terminal Help
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
/dev/ttyACM0
sommersoft@thespacebetween:~$ screen /dev/ttyACM0 115200
```

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

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

Permissions on Linux

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

```
ackbar@desk: ~
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:~$
```

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

```
ackbar@desk: ~
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
>>>
```

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

Frequently Asked Questions

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

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

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

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

- [2.x \(https://adafruit.it/FJA\)](https://adafruit.it/FJA)
- [3.x \(https://adafruit.it/FJB\)](https://adafruit.it/FJB)
- [4.x \(https://adafruit.it/QDL\)](https://adafruit.it/QDL)
- [5.x \(https://adafruit.it/QDJ\)](https://adafruit.it/QDJ)

Is ESP8266 or ESP32 supported in CircuitPython? Why not?

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

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

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

How do I connect to the Internet with CircuitPython?

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

Is there asyncio support in CircuitPython?

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

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

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

What is a `MemoryError`?

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

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

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

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

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

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

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

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

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

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

You can download `mpy-cross` for your operating system from <https://adafruit-circuit-python.s3.amazonaws.com/index.html?prefix=bin/mpy-cross/> (<https://adafru.it/QDK>). Builds are available for Windows, macOS, x64 Linux, and Raspberry Pi Linux. Choose the latest `mpy-cross` whose version matches the version of CircuitPython you are using.

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

How do I check how much memory I have free?

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

Will give you the number of bytes available for use.

Does CircuitPython support interrupts?

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

Does Feather M0 support WINC1500?

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

Can AVRs such as ATmega328 or ATmega2560 run CircuitPython?

No.

Commonly Used Acronyms

CP or CPy = [CircuitPython](https://adafru.it/cpy-welcome) (<https://adafru.it/cpy-welcome>)

CPC = [Circuit Playground Classic](https://adafru.it/ncE) (<https://adafru.it/ncE>)

CPX = [Circuit Playground Express](https://adafru.it/wpF) (<https://adafru.it/wpF>)

ESP32-S2 Bugs & Limitations

Nobody likes bugs, but all nontrivial software and hardware has some. The master list of problems is the [issues list on github](https://adafru.it/PEk) (<https://adafru.it/PEk>).

Adafruit considers CircuitPython for the ESP32-S2 to be beta quality software.

Cannot reinitialize certain peripherals (especially busio.I2C)

If you create a `busio.I2C` object, call `.deinit()` on it, and then create another one, CircuitPython will lock up.

Workaround: Do not deinitialize I2C objects, except by soft reload or entering deep sleep.

No DAC-based audio output

Current versions of esp-idf do not have the required APIs for DAC-based audio output. Once a future version of esp-idf that adds it, it will be possible to implement DAC-based AudioOut in CircuitPython.

Workaround: PWMOut can create tones and buzzes.

Workaround: I2SOut audio is currently being developed and will work with boards such as the [Adafruit I2S Stereo Decoder - UDA1334A Breakout \(https://adafru.it/PEI\)](https://adafru.it/PEI).

Deep Sleep & Wake-up sources

ESP32-S2 has hardware limitations on what kind of "pin alarms" can wake it. The following combinations are possible:

- EITHER one or two pins that wake from deep sleep when they are pulled LOW
- OR an arbitrary number of pins that wake from deep sleep when they are pulled HIGH, and optionally one pin that wakes from deep sleep when pulled LOW

This means that "wake" buttons should be wired so that pressing them pulls HIGH and a pull DOWN resistor is used with the pin. However, in some hardware designs including the original MagTag, the integrated buttons are pulled LOW when pressed and so only 1 or 2 buttons can be selected to wake the MagTag.

Troubleshooting

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

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

Always Run the Latest Version of CircuitPython and Libraries

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

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

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

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

We are no longer building or supporting the CircuitPython 5.x or earlier library bundles. We highly encourage you to [update CircuitPython to the latest version](#) (<https://adafru.it/Em8>) and use [the current version of the libraries](#) (<https://adafru.it/ENC>). However, if for some reason you cannot update, here are the last available library bundles for older versions:

- [2.x bundle](https://adafru.it/FJA) (<https://adafru.it/FJA>)
- [3.x bundle](https://adafru.it/FJB) (<https://adafru.it/FJB>)
- [4.x bundle](https://adafru.it/QDL) (<https://adafru.it/QDL>)
- [5.x bundle](https://adafru.it/QDJ) (<https://adafru.it/QDJ>)

CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present

You may have a different board.

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

MakeCode

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

MacOS

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

Windows 10

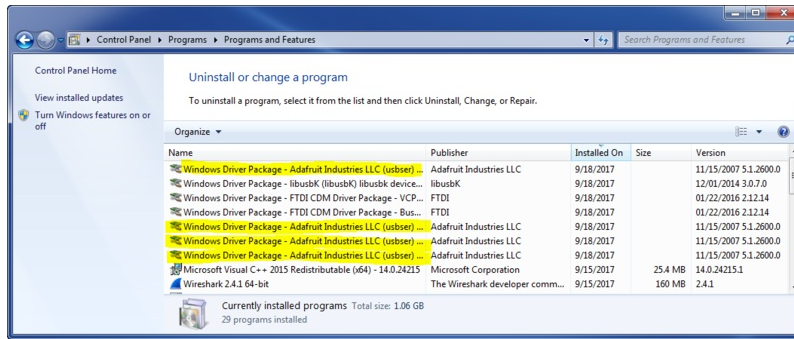
Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 with the driver package installed? You don't need to install this package on Windows 10 for most Adafruit boards. The old version (v1.5) can interfere with recognizing your device. Go to **Settings** ->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

Windows 7 or 8.1

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

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

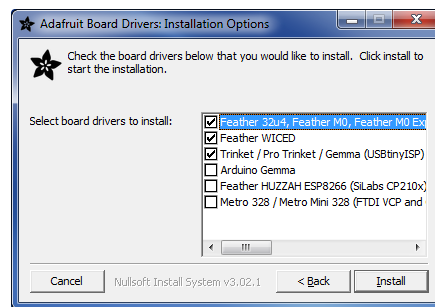
We [recommend](https://adafru.it/Amd) (<https://adafru.it/Amd>) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you: see the [link](https://adafru.it/Amd) (<https://adafru.it/Amd>).



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

<https://adafru.it/AB0>
<https://adafru.it/AB0>

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



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 \(https://adafru.it/jlf\)](https://adafru.it/jlf) or on the [Adafruit Discord \(\)](#) if this does not work for you!

Windows Explorer Locks Up When Accessing **boardnameBOOT** Drive

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

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

Copying UF2 to **boardnameBOOT** Drive Hangs at 0% Copied

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

CIRCUITPY Drive Does Not Appear

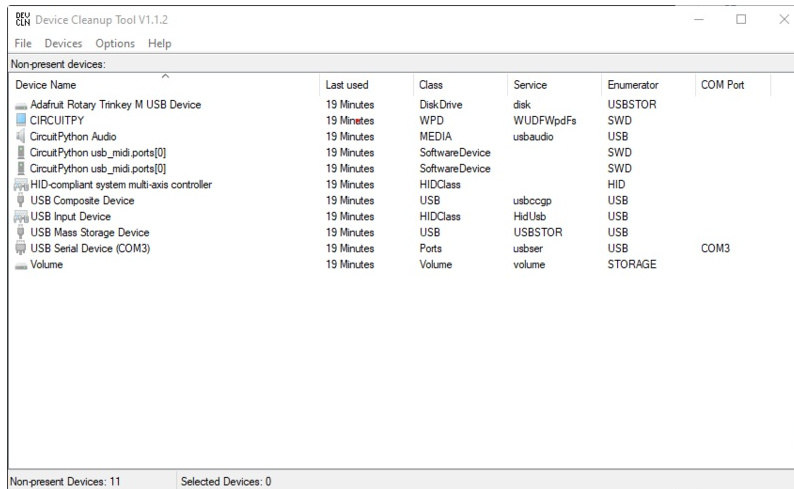
Kaspersky anti-virus can block the appearance of the **CIRCUITPY** drive. We haven't yet figured out a settings change that prevents this. Complete uninstallation of Kaspersky fixes the problem.

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

Device Errors or Problems on Windows

Windows can become confused about USB device installations. This is particularly true of Windows 7 and 8.1. We [recommend](https://adafru.it/Amd) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you: see this [link](https://adafru.it/V2a).

If not, try cleaning up your USB devices. Use [Uwe Sieber's Device Cleanup Tool](https://adafru.it/RWd). Download and unzip the tool. Unplug all the boards and other USB devices you want to clean up. Run the tool as Administrator. You will see a listing like this, probably with many more devices. It is listing all the USB devices that are *not* currently attached.



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

Serial Console in Mu Not Displaying Anything

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

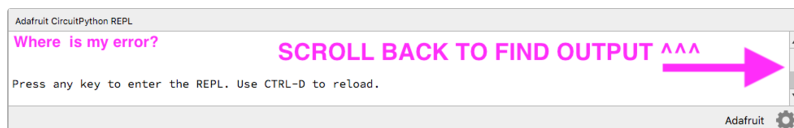
Depending on the size of your screen or Mu window, when you open the serial console, the serial console panel may be very small. This can be a problem. A basic CircuitPython error takes 10 lines to display!

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

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

More complex errors take even more lines!

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



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

CircuitPython RGB Status Light

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

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

In versions before 7.0.0, they do NOT indicate any status while running CircuitPython.

CircuitPython 7.0.0 and Later

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

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

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

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

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

CircuitPython 6.3.0 and earlier

Here's what the colors and blinking mean:

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

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

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

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

ValueError: Incompatible `.mpy` file.

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

So, for instance, if you upgraded to CircuitPython 7.x from 6.x you'll need to download a newer version of the library that triggered the error on `import`. They are all available in the [Adafruit bundle \(https://adafru.it/y8E\)](https://adafru.it/y8E).

CIRCUITPY Drive Issues

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

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

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

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

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

Easiest Way: Use `storage.erase_filesystem()`

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

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

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

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

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

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

1. Download the correct erase file:



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 yellow or blue, indicating the erase has started.
5. After approximately 15 seconds, the mainboard NeoPixel will light up green. On the NeoTrellis M4 this is the first NeoPixel on the grid
6. Double-click the reset button on the board to bring up the `boardnameBOOT` drive.

7. [Drag the appropriate latest release of CircuitPython](https://adafru.it/Amd) (<https://adafru.it/Amd>) `.uf2` file to the `boardnameBOOT` drive.

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

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

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

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

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

1. Download the erase file:



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

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

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

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

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

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

Running Out of File Space on Non-Express Boards

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

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

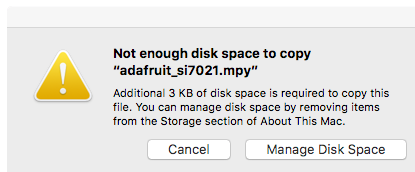
Delete something!

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

Use tabs

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

MacOS loves to add extra files.



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

Prevent & Remove MacOS Hidden Files

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

```
ls -l /Volumes
```

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

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

```
mdutil -i off /Volumes/CIRCUITPY
cd /Volumes/CIRCUITPY
rm -rf .{,._}{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.

Alternatively, with CircuitPython 4.x and above, the special files and folders mentioned above will be created automatically if you erase and reformat the filesystem.

WARNING: Save your files first! Do this in the REPL:

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

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

Copy Files on MacOS Without Creating Hidden Files

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

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

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

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

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

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

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

Other MacOS Space-Saving Tips

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

```

1. bash
bash  #1 bash  #2 bash  #3
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem      Size  Used Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1    59Ki  54Ki  5.5Ki   91%    128     0  100%  /Volumes/CIRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./
./
./TemporaryItems/
./Trashes/
./TemporaryItems*
./Trashes*
Windows 7 Driver/
boot_out.txt*
._original_code.py*
code.py*
.fsevents/
README.txt*
lib/
original_code.py*
(venv) tannewt@shallan:/Volumes $

```

Lets remove the `._` files first.

```

1. bash
bash  #1 bash  #2 bash  #3
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem      Size  Used Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1    59Ki  54Ki  5.5Ki   91%    128     0  100%  /Volumes/CIRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./
./
./TemporaryItems/
./Trashes/
./TemporaryItems*
./Trashes*
Windows 7 Driver/
boot_out.txt*
._original_code.py*
code.py*
.fsevents/
README.txt*
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 %iused  Mounted on
/dev/disk3s1    59Ki  42Ki  18Ki   71%    128     0  100%  /Volumes/CIRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./
./
./TemporaryItems/
./Trashes/
./TemporaryItems*
./Trashes*
Windows 7 Driver/
boot_out.txt*
code.py*
.fsevents/
README.txt*
lib/
original_code.py*
(venv) tannewt@shallan:/Volumes $

```

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

Device Locked Up or Boot Looping

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

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

For most devices:

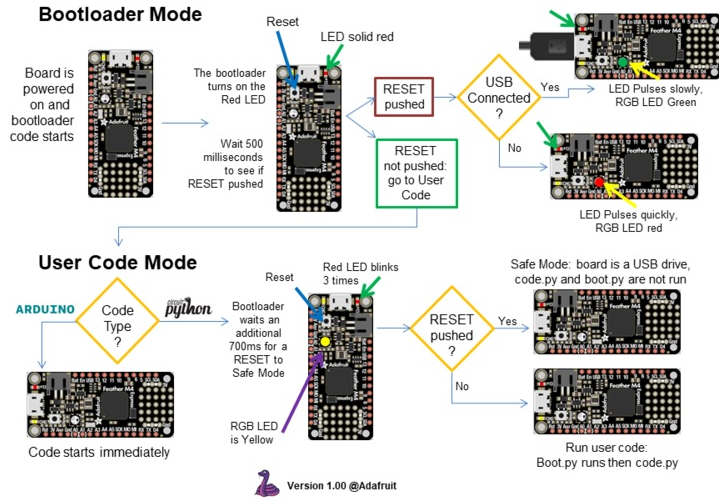
Press the reset button, and then when the RGB status LED blinks yellow, press the reset button again. Since your reaction time may not be that fast, try a "slow" double click, to catch the yellow LED on the second click.

For ESP32-S2 based devices:

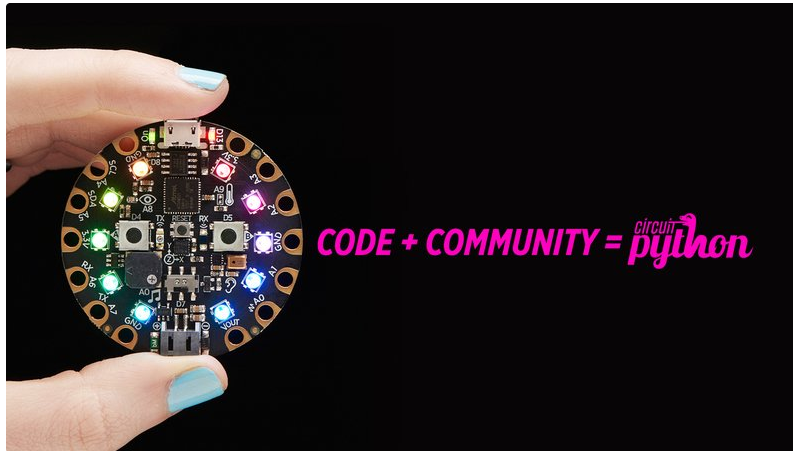
Press and release the reset button, then press and release the boot button about 3/4 of a second later.

Refer to the following diagram for boot sequence details:

The CircuitPython Boot Sequence



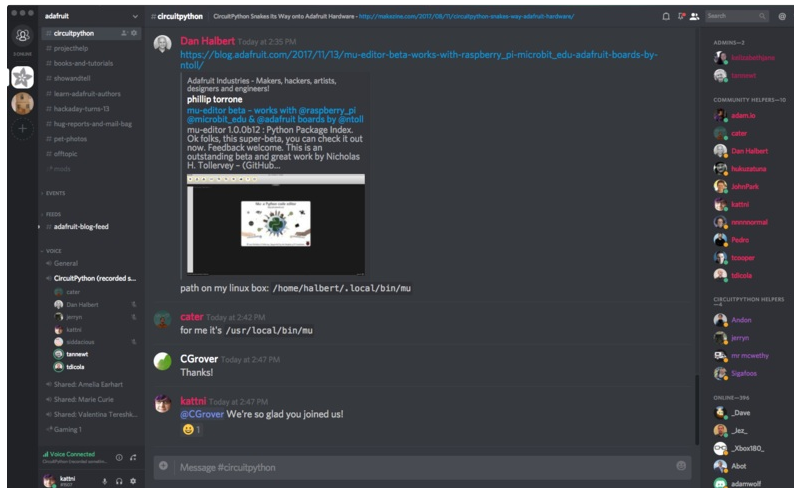
Welcome to the Community!



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 #help-with-projects channel for assistance with your current project or help coming up with ideas for your next one. There's the #showandtell channel for showing off your newest creation. Don't be afraid to ask a question in any channel! If you're unsure, #general is a great place to start. If another channel is more likely to provide you with a better answer, someone will guide you.

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

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

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

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

Adafruit Forums

Forum Index User Settings • View your posts

ADAFRUIT CUSTOMER SUPPORT FORUMS

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

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

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

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

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

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

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

Adafruit CircuitPython and MicroPython

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

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

POST A TOPIC Mark topics read • 179 topics • Page 1 of 4 • 1234

Please be positive and constructive with your questions and comments.

ANNOUNCEMENTS	Replies	Views	Last post
CIRCUITPYTHON 2.1.0 RELEASED! by danhalbert • Wed Oct 18, 2017 12:47 am	1	111	by danhalbert <small>is</small> 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

This repository Search Pull requests Issues Marketplace Explore

adafruit / circuitpython Unwatch 69 Unstar 256 Fork 1,357

forked from [micropython/micropython](#)

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 \(https://adafru.it/tB7\)](https://adafru.it/tB7) itself. If you need an account, visit <https://github.com/> (<https://adafru.it/d6C>) and sign up.

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

<input type="checkbox"/>	OneWire BusDevice driver good first issue	2
<small>#338 opened 29 days ago by tannevt Long term</small>		
<input type="checkbox"/>	Feather M0 Adalogger does not have D8 or D7 good first issue	7
<small>#323 opened on Oct 13 by ladyada 3.0</small>		
<input type="checkbox"/>	Audit and fix native API for methods that accept and ignore extra args. good first issue	
<small>#321 opened on Oct 12 by tannevt Long term</small>		

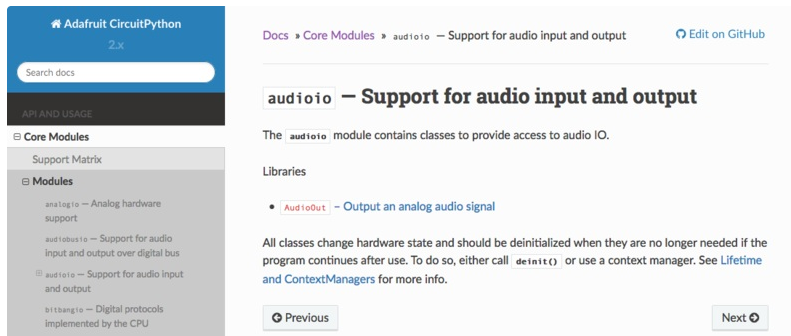
Already experienced and looking for a challenge? Check out the rest of the issues list and you'll find plenty of ways to contribute. You'll find everything from new driver requests to core module updates. There's plenty of opportunities for everyone at any level!

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



AdafuIt CircuitPython 2.x

Search docs

API AND USAGE

Core Modules

- Support Matrix
- Modules
 - analogio — Analog hardware support
 - audiobusio — Support for audio input and output over digital bus
 - audioio — Support for audio input and output
 - bitbangio — Digital protocols implemented by the CPU

Docs » Core Modules » audioio — Support for audio input and output [Edit on GitHub](#)

audioio — Support for audio input and output

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

Libraries

- `AudioOut` — Output an analog audio signal

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

[Previous](#) [Next](#)

[ReadTheDocs \(https://adafru.it/Beg\)](https://adafru.it/Beg) 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 \(https://adafru.it/Beh\)](https://adafru.it/Beh). Each module lists the available libraries. Each module library page lists the available parameters and an explanation for each. In many cases, you'll find quick code examples to help you understand how the modules and parameters work, however it won't have detailed explanations like the Learn Guides. If you want help understanding what's going on behind the scenes in any CircuitPython code you're writing, ReadTheDocs is there to help!

Here is blinky:

```
import digitalio
from board import *
import time

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

CircuitPython Essentials



You've been introduced to CircuitPython, and worked through getting everything set up. What's next? CircuitPython Essentials!

There are a number of core modules built into CircuitPython, which can be used along side the many CircuitPython libraries available. The following pages demonstrate some of these modules. Each page presents a different concept including a code example with an explanation. All of the examples are designed to work with your microcontroller board.

Time to get started learning the CircuitPython essentials!

Blink

In learning any programming language, you often begin with some sort of **Hello, World!** program. In CircuitPython, Hello, World! is blinking an LED. Blink is one of the simplest programs in CircuitPython. It involves three built-in modules, two lines of set up, and a short loop. Despite its simplicity, it shows you many of the basic concepts needed for most CircuitPython programs, and provides a solid basis for more complex projects. Time to get blinky!

LED Location



The built-in red LED (indicated in red in the image) is located in the upper right of the FunHouse door, towards the center of the board on the front.

Blinking an LED

Save the following as `code.py` on your `CIRCUITPY` drive.

```
"""CircuitPython Blink Example - the CircuitPython 'Hello, World!'"""
import time
import board
import digitalio

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

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

The built-in LED begins blinking!

Note that the code is a little less "Pythonic" than it could be. It could also be written as `led.value = not led.value` with a single `time.sleep(0.5)`. That way is more difficult to understand if you're new to programming, so the example is a bit longer than it needed to be to make it easier to read.

It's important to understand what is going on in this program.

First you `import` three modules: `time`, `board` and `digitalio`. This makes these modules available for use in your code. All three are built-in to CircuitPython, so you don't need to download anything to get started.

Next, you set up the LED. To interact with hardware in CircuitPython, your code must let the board know where to look for the hardware and what to do with it. So, you create a `digitalio.DigitalInOut()` object, provide it the LED pin using the `board` module, and save it to the variable `led`. Then, you tell the pin to act as an `OUTPUT`.

Finally, you create a `while True:` loop. This means all the code inside the loop will repeat indefinitely. Inside the loop, you set `led.value = True` which powers on the LED. Then, you use `time.sleep(0.5)` to tell the code to wait half a second before moving on to the next line. The next line sets `led.value = False` which turns the LED off. Then you use another `time.sleep(0.5)` to wait half a second before starting the loop over again.

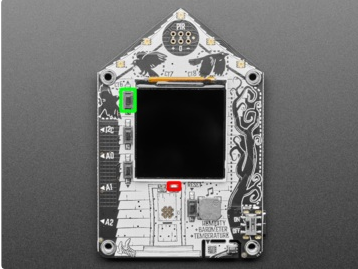
With only a small update, you can control the blink speed. The blink speed is controlled by the amount of time you tell the code to wait before moving on using `time.sleep()`. The example uses `0.5`, which is one half of one second. Try increasing or decreasing these values to see how the blinking changes.

That's all there is to blinking an LED using CircuitPython!

Digital Input

The CircuitPython `digitalio` module has many applications. The basic Blink program sets up the LED as a digital output. You can just as easily set up a **digital input** such as a button to control the LED. This example builds on the basic Blink example, but now includes setup for a button switch. Instead of using the `time` module to blink the LED, it uses the status of the button switch to control whether the LED is turned on or off.

LED and Button



- The red led (indicated by the red box in the image) is located in the upper right of the FunHouse door, towards the center of the board on the front.
- The top button (indicated by the green box in the image) is located near the upper left corner of the display below the up arrow on the board silk.

Controlling the LED with a Button

Save the following as `code.py` on your `CIRCUITPY` drive.

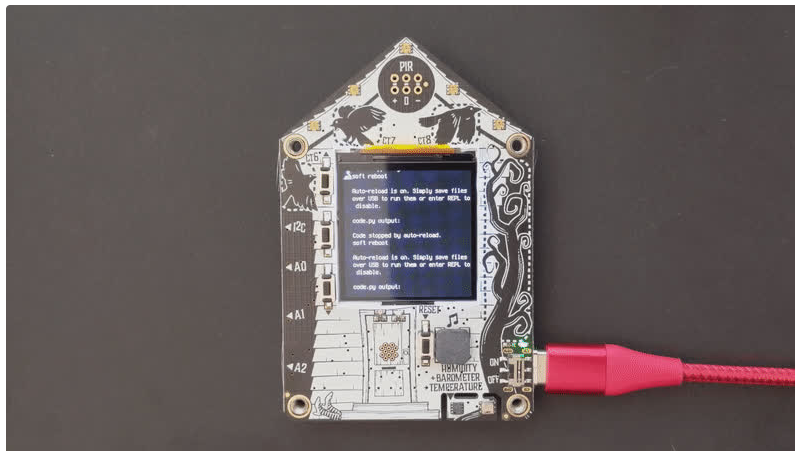
```
"""CircuitPython Digital Input Example for FunHouse"""
import board
import digitalio

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

button = digitalio.DigitalInOut(board.BUTTON_UP)
button.switch_to_input(pull=digitalio.Pull.DOWN)

while True:
    if not button.value:
        led.value = False
    else:
        led.value = True
```

Now, press the button. The LED lights up! Let go of the button and the LED turns off.



Note that the code is a little less "Pythonic" than it could be. It could also be written as `led.value = not button.value`. That way is more difficult to understand if you're new to programming, so the example is a bit longer than it needed to be to make it easier to read.

First you `import` two modules: `board` and `digitalio`. This makes these modules available for use in your code. Both are built-in to CircuitPython, so you don't need to download anything to get started.

Next, you set up the LED. To interact with hardware in CircuitPython, your code must let the board know where to look for the hardware and what to do with it. So, you create a `digitalio.DigitalInOut()` object, provide it the LED pin using the `board` module, and save it to the variable `led`. Then, you tell the pin to act as an `OUTPUT`.

You include setup for the button as well. It is similar to the LED setup, except the button is an `INPUT`, and requires a pull up.

Inside the loop, you check to see if the button is pressed, and if so, turn on the LED. Otherwise the LED is off.

That's all there is to controlling an LED with a button switch!

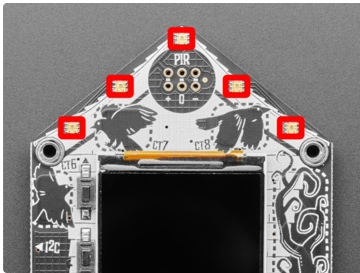
Built-In DotStar LEDs

Your board has multiple built-in RGB DotStar LEDs. You can use CircuitPython code to control the color and brightness of these LEDs. They are also used to indicate the bootloader status.

A DotStar refers to any 2-wire serial LED, typically APA102, but also possibly SK9822. Along side a driver chip, DotStars have have three LEDs: RGB DotStars have a red LED, a blue LED and a green LED, and white DotStars have three white LEDs. The LEDs on your microcontroller are RGB DotStars! DotStars operate over a generic 2-wire SPI bus, which means they aren't as strict about timing. They allow for extremely fast data and PWM rates so they're suitable for POV displays. They can be used individually (as in the built-in LED on your board), or chained together in strips or other creative form factors. DotStars do not light up on their own; they must be connected to a microcontroller. They require two pins, data and clock, to operate. The response time is faster when connected to a hardware SPI pair of pins, but will work connected to any two digital pins. You do not need to worry about connecting the DotStars because they're built into your microcontroller!

This page will cover using CircuitPython to control the RGB DotStars built into your microcontroller. You'll learn how to change the color and brightness, and how to make a rainbow. Time to get started!

DotStar Location



On the FunHouse, the DotStar LEDs (indicated by red boxes in the image) are spread out evenly along the top edges of the board, beginning at the top center, and then along the two edges to the mounting holes.

DotStar Color and Brightness

To use the built-in DotStars on your board, you need to first install the DotStar library into the `lib` folder on your **CIRCUITPY** drive.

Then you need to update `code.py`.

Click the **Download Project Bundle** button below to download the necessary libraries and the `code.py` file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the `code.py` file to your **CIRCUITPY** drive.

```
"""CircuitPython DotStar red, green, blue example for FunHouse"""
import time
import board
import adafruit_dotstar

dots = adafruit_dotstar.DotStar(board.DOTSTAR_CLOCK, board.DOTSTAR_DATA, 5)
dots.brightness = 0.3

while True:
    dots.fill((255, 0, 0))
    time.sleep(0.5)
    dots.fill((0, 255, 0))
    time.sleep(0.5)
    dots.fill((0, 0, 255))
    time.sleep(0.5)
```

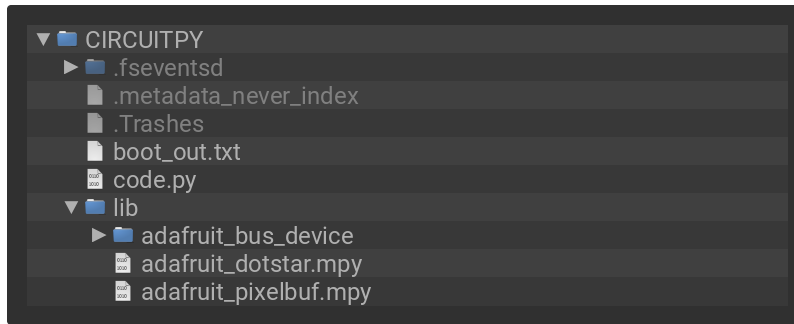
Your **CIRCUITPY** drive contents should resemble the image below.

You should have in / of the **CIRCUITPY** drive:

- `code.py`

And in the `lib` folder on your **CIRCUITPY** drive:

- `adafruit_bus_device/`
- `adafruit_dotstar.mpy`
- `adafruit_pixelbuf.mpy`



The DotStar LEDs being flashing red, green and blue!



If your DotStars do not start flashing red, green and blue, make sure you've copied all the necessary files and folders to the CIRCUITPY drive!

First you import the necessary modules, `time` and `board`, and the necessary library, `adafruit_dotstar`. This makes these modules and libraries available for use in your code. The first two are modules built-in to CircuitPython, so you don't need to download anything to use those. The `adafruit_dotstar` library is separate, which is why you needed to install it before getting started.

Next, you set up the Dotstar LEDs. To interact with hardware in CircuitPython, your code must let the board know where to look for the hardware and what to do with it. So, you create a `adafruit_dotstar.DotStar()` object, provide it the DotStar LED pins using the `board` module, and tell it the number of LEDs. You save this object to the variable `pixels`.

Then, you set the DotStar brightness using the `brightness` attribute. `brightness` expects float between `0` and `1.0`. A *float* is essentially a number with a decimal in it. The brightness value represents a percentage of maximum brightness; `0` is 0% and `1.0` is 100%. Therefore, setting `pixel.brightness = 0.3` sets the brightness to 30%. The default brightness, which is to say the brightness if you don't explicitly set it, is `1.0`. The default is really bright! That is why there is an option available to easily change the brightness.

Inside the loop, you turn the DotStars red for 0.5 seconds, green for 0.5 seconds, and blue for 0.5 seconds.

To turn the DotStars red, you "fill" them with an RGB value. Check out the section below for details on RGB colors. The RGB value for red is `(255, 0, 0)`. Note that the RGB value includes the parentheses. The `fill()` attribute expects the full RGB value including those parentheses. That is why there are two pairs of parentheses in the code.

You can change the RGB values to change the colors that the DotStars cycle through. Check out the list below for some examples. You can make any color of the rainbow with the right RGB value combination!

That's all there is to changing the color and setting the brightness of the built-in DotStar LEDs!

RGB LED Colors

RGB LED colors are set using a combination of red, green, and blue, in the form of an **(R, G, B)** tuple. Each member of the tuple is set to a number between 0 and 255 that determines the amount of each color present. Red, green and blue in different combinations can create all the colors in the rainbow! So, for example, to set an LED to red, the tuple would be `(255, 0, 0)`, which has the maximum level of red, and no green or blue. Green would be `(0, 255, 0)`, etc. For the colors between, you set a combination, such as cyan which is `(0, 255, 255)`, with equal amounts of green and blue. If you increase all values to the same level, you get white! If you decrease all the values to 0, you turn the LED off.

Common colors include:

- red: (255, 0, 0)
- green: (0, 255, 0)
- blue: (0, 0, 255)
- cyan: (0, 255, 255)
- purple: (255, 0, 255)
- yellow: (255, 255, 0)
- white: (255, 255, 255)
- black (off): (0, 0, 0)

DotStar Rainbow

You should have already installed the library necessary to use the built-in DotStar LEDs. If not, follow the steps at the beginning of the DotStar Color and Brightness section to install it.

Update your `code.py` to the following, and save.

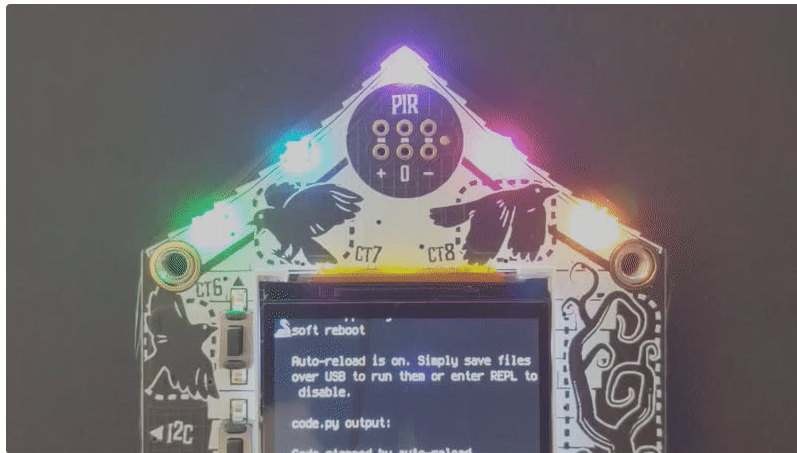
```
"""CircuitPython DotStar rainbow example for FunHouse"""
import time
import board
import adafruit_dotstar
from rainbowio import colorwheel

dots = adafruit_dotstar.DotStar(board.DOTSTAR_CLOCK, board.DOTSTAR_DATA, 5, auto_write=False)
dots.brightness = 0.3

def rainbow(delay):
    for color_value in range(255):
        for led in range(5):
            pixel_index = (led * 256 // 5) + color_value
            dots[led] = colorwheel(pixel_index & 255)
            dots.show()
            time.sleep(delay)

while True:
    rainbow(0.01)
```

The DotStars display a rainbow cycle!



This example builds on the previous example.

First, you import the same three modules and libraries. In addition to those, you import `colorwheel`.

The DotStar hardware setup is similar, but you now also set `auto_write` to `False`. This means that now the DotStar won't change unless you explicitly tell it to by calling `show()`. This is necessary for this example to speed up the rainbow animation. Brightness setting is the same.

Next, you have the `rainbow()` helper function. This helper displays the rainbow cycle. It expects a `delay` in seconds. The higher the number of seconds provided for `delay`, the slower the rainbow will cycle. The helper cycles through the values of the color wheel to create a rainbow of colors.

Inside the loop, you call the rainbow helper with a 0.01 second delay, by including `rainbow(0.01)`.

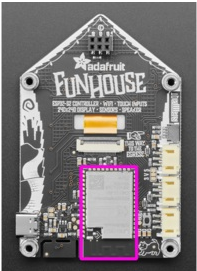
That's all there is to making rainbows using the built-in DotStar LEDs!

CPU Temperature

There is a temperature sensor built into the CPU on your microcontroller board. It reads the internal CPU temperature, which varies depending on how long the board has been running or how intense your code is.

CircuitPython makes it really simple to read this data from the temperature sensor built into the microcontroller. Using the built-in `microcontroller` module, you can easily read the temperature.

Microcontroller Location



The microcontroller on the FunHouse is located on the bottom of the back of the board, towards the center.

Reading the Microcontroller Temperature

The data is read using two lines of code. All necessary modules are built into CircuitPython, so you don't need to download any extra files to get started.

[Connect to the serial console \(https://adafruit.it/Bec\)](https://adafruit.it/Bec), and then update your `code.py` to the following and save.

```
"""CircuitPython CPU temperature example in Celsius"""
import time
import microcontroller

while True:
    print(microcontroller.cpu.temperature)
    time.sleep(0.15)
```

```
CircuitPython REPL
40.7144
40.2463
41.1825
41.6507
40.7144
40.7144
40.2463
```

The CPU temperature in Celsius is printed out to the serial console!

Try putting your finger on the microcontroller to see the temperature change.

The code is simple. First you import two modules: `time` and `microcontroller`. Then, inside the loop, you print the microcontroller CPU temperature, and the `time.sleep()` slows down the print enough to be readable. That's it!

You can easily print out the temperature in Fahrenheit by adding a little math to your code, using this simple formula: $Celsius * (9/5) + 32$.

Update your `code.py` to the following, and save.

```
"""CircuitPython CPU temperature example in Fahrenheit"""
import time
import microcontroller

while True:
    print(microcontroller.cpu.temperature * (9 / 5) + 32)
    time.sleep(0.15)
```

```
CircuitPython REPL
104.443
104.443
105.286
104.443
106.971
101.915
103.601
```

The CPU temperature in Fahrenheit is printed out to the serial console!

That's all there is to reading the CPU temperature using CircuitPython!

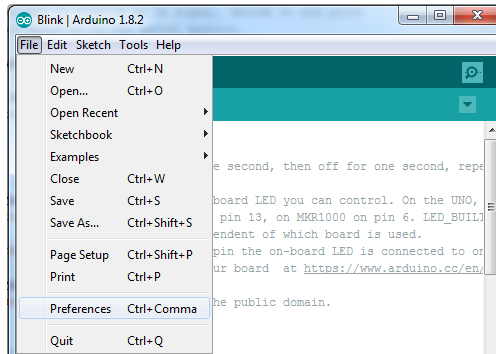
Arduino IDE Setup

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

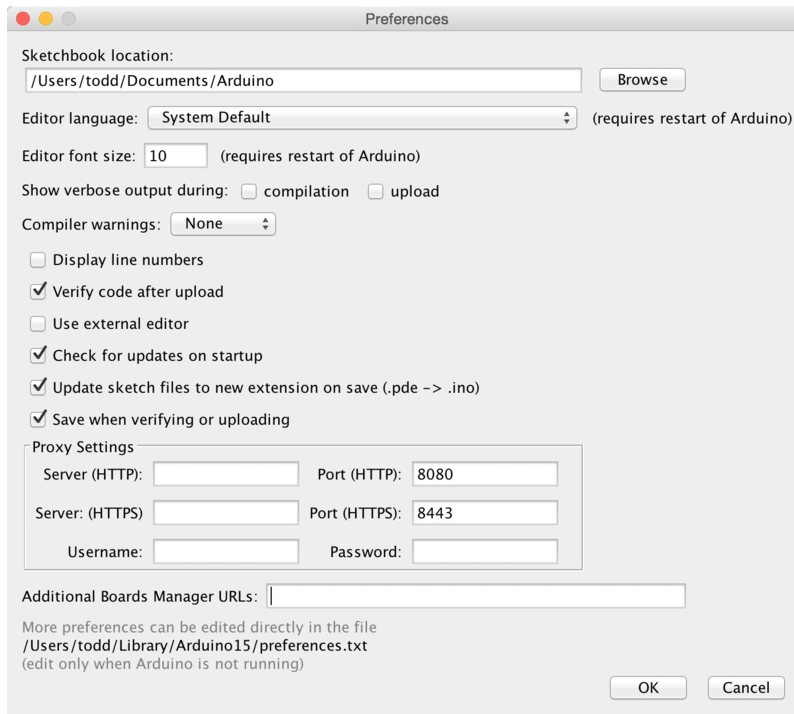
<https://adafru.it/f1P>
<https://adafru.it/f1P>

The ESP32-S2 Arduino board support package is currently part of the **2.0.0-alpha1** release. To use the ESP32-S2 with Arduino, you'll need to follow the steps below for your operating system. You can also [check out the Espressif Arduino repository for the most up to date details on how to install it \(https://adafru.it/weF\)](https://adafru.it/weF).

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



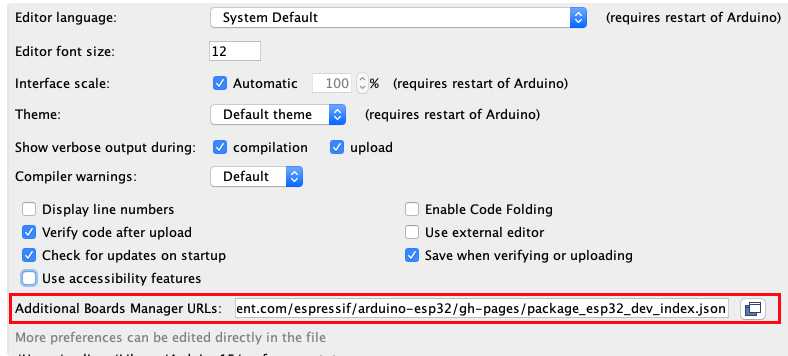
A dialog will pop up just like the one shown below.



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

To find the most up to date list of URLs you can add, you can visit the list of [third party board URLs on the Arduino IDE wiki \(https://adafru.it/f7U\)](https://adafru.it/f7U). We will only need to add one URL to the IDE in this example, but *you can add multiple URLs by separating them with commas*. Copy and paste the link below into the **Additional Boards Manager URLs** option in the Arduino IDE preferences.

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json

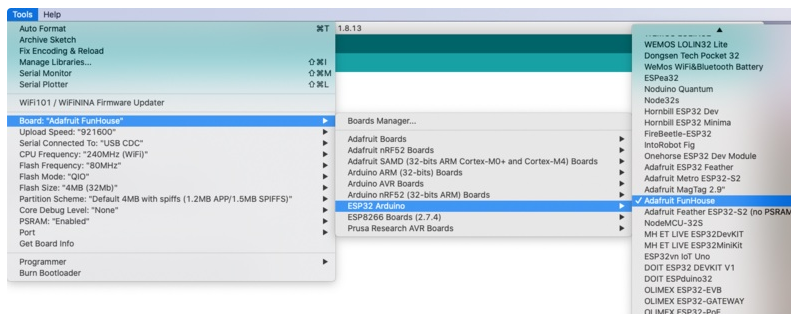


If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

Once done click **OK** to save the new preference settings.

In the **Tools** → **Board** submenu you should see **ESP32 Arduino** and in that dropdown it should contain the ESP32 boards along with all the latest ESP32-S2 boards.

Look for the board called **Adafruit FunHouse**.



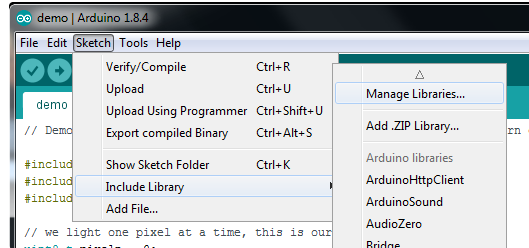
Arduino Libraries

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

There's a lot of libraries!

Install Libraries

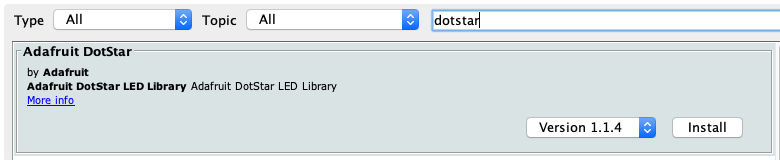
Open up the library manager...



And install the following libraries:

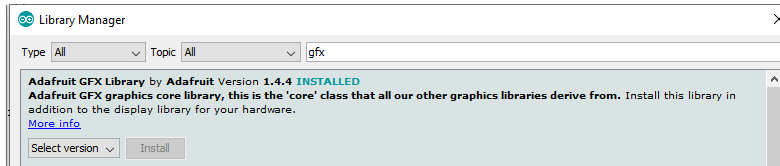
Adafruit DotStar

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



Adafruit GFX

This is the graphics library used to draw to the screen



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

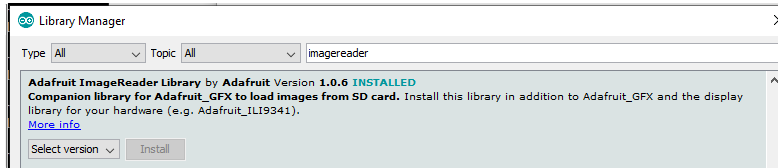
Adafruit ST7735 and ST7789

For using the display.



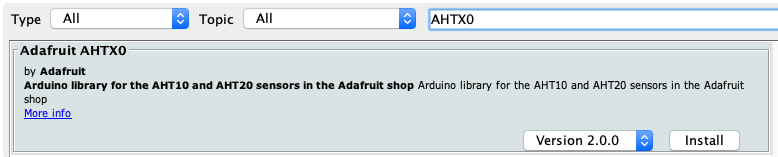
Adafruit ImageReader

For reading bitmaps from SD and displaying



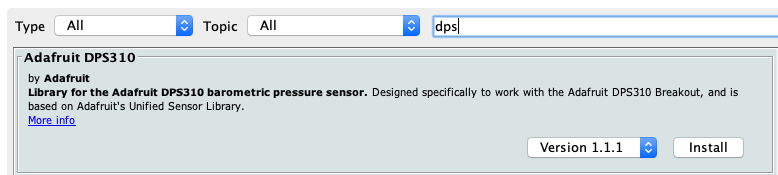
Adafruit AHTX0

For using the Humidity and Temperature Sensor



Adafruit DPS310

For using the Pressure Sensor



Arduino Basics

Arduino support for the ESP32-S2 is relatively new right now, so we recommend using CircuitPython!

Once you have Arduino installed and set up and you can upload simple blink sketches, you can move on to using each element of the FunHouse board.

Using the Red LED

It's always good to blink the LED when you want to verify if something is happening on your board. The LED is on IO #13, but we recommend you use the `LED_BUILTIN` macro and you can use this simple sketch example to blink the LED:

```
void setup() {
  // initialize built in LED pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  // initialize USB serial converter so we have a port created
  Serial.begin();
}

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

Reading the Buttons

There are three buttons on the front of the FunHouse - they're connected to digital pins IO 3, 4, and 5. However, we recommend you use the constants `BUTTON_DOWN`, `BUTTON_SELECT`, `BUTTON_UP`.

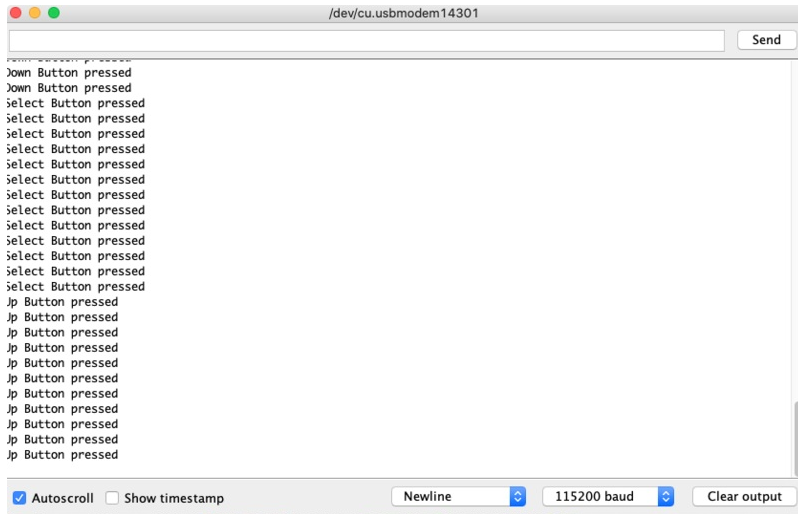
```
void setup() {
  Serial.begin(115200);

  pinMode(BUTTON_DOWN, INPUT_PULLDOWN);
  pinMode(BUTTON_SELECT, INPUT_PULLDOWN);
  pinMode(BUTTON_UP, INPUT_PULLDOWN);
}

void loop() {
  if (digitalRead(BUTTON_DOWN)) {
    Serial.println("Down Button pressed");
  }
  if (digitalRead(BUTTON_SELECT)) {
    Serial.println("Select Button pressed");
  }
  if (digitalRead(BUTTON_UP)) {
    Serial.println("Up Button pressed");
  }

  // small debugging delay
  delay(10);
}
```

Open the serial console and press buttons to see the serial output printed!



Reading the Capacitive Touch Pads

To read the value of the capacitive touch pads, you can use the `touchRead()` function, which is part of the ESP32 package. To use it, you only need to provide the GPIO pin number. The FunHouse uses IO #6, #7, and #8 for the button style touch pads and #9 through #13 along the slider.

So to get the value of IO #7, you would use the following code:

```
uint16_t touchread;

touchread = touchRead(7);
if (touchread > 20000 ) {
  // Do Something
}
```

You may want to try adjusting the threshold for your needs. A more complete example can be found on the [Arduino Self Test Example](#) page.

Using On-Board DotStars

There are 4 DotStar LEDs on pin IO #14 and #15 (we recommend using the macro `PIN_DOTSTAR_DATA` and `PIN_DOTSTAR_CLOCK`).

Here's an example sketch that initializes the DotStar LEDs and color cycles them through a rainbow of different colors.

```

#include <Adafruit_DotStar.h>

#define NUM_DOTSTAR 5

// LEDs!
Adafruit_DotStar pixels(NUM_DOTSTAR, PIN_DOTSTAR_DATA, PIN_DOTSTAR_CLOCK, DOTSTAR_BRG);

uint16_t firstPixelHue = 0;
uint8_t LED_dutycycle = 0;

void setup() {
  Serial.begin(115200);

  pinMode(LED_BUILTIN, OUTPUT);

  ledcSetup(0, 5000, 8);
  ledcAttachPin(LED_BUILTIN, 0);

  pixels.begin(); // Initialize pins for output
  pixels.show(); // Turn all LEDs off ASAP
  pixels.setBrightness(20);
}

void loop() {
  Serial.println("Hello!");

  // pulse red LED
  ledcWrite(0, LED_dutycycle++);

  // rainbow dotstars
  for (int i=0; i<pixels.numPixels(); i++) { // For each pixel in strip...
    int pixelHue = firstPixelHue + (i * 65536L / pixels.numPixels());
    pixels.setPixelColor(i, pixels.gamma32(pixels.ColorHSV(pixelHue)));
  }
  pixels.show(); // Update strip with new contents
  firstPixelHue += 256;

  delay(15);
}

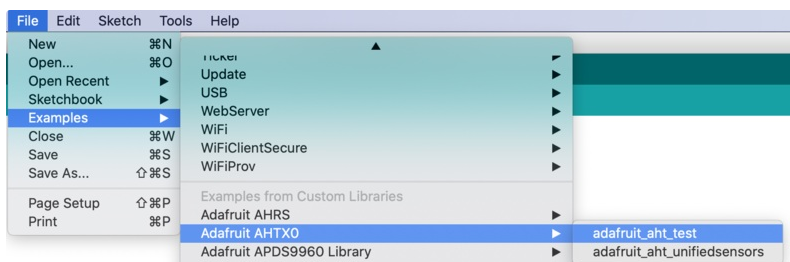
void rainbow(int wait) {
  for(long firstPixelHue = 0; firstPixelHue < 5*65536; firstPixelHue += 256) {
    for(int i=0; i<pixels.numPixels(); i++) { // For each pixel in strip...
      int pixelHue = firstPixelHue + (i * 65536L / pixels.numPixels());
      pixels.setPixelColor(i, pixels.gamma32(pixels.ColorHSV(pixelHue)));
    }
    pixels.show(); // Update strip with new contents
    delay(wait); // Pause for a moment
  }
}

```

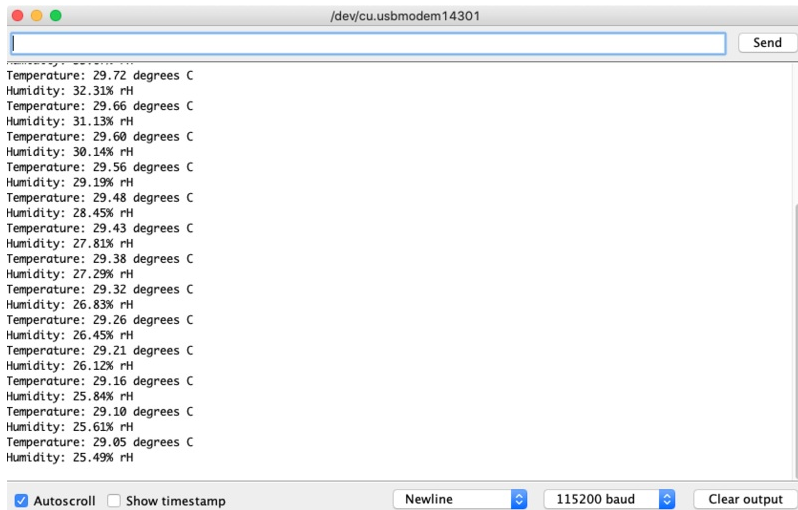
Using On-board Humidity and Temperature Sensor

There's a pre-soldered humidity and temperature sensor that you can use.

You can test the AHTX0 by loading the included `adafruit_aht_test` in the Arduino library



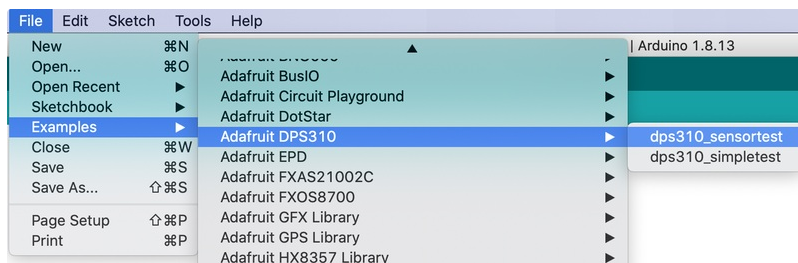
Now you can upload, reset, and check the serial port for temperature and humidity data!



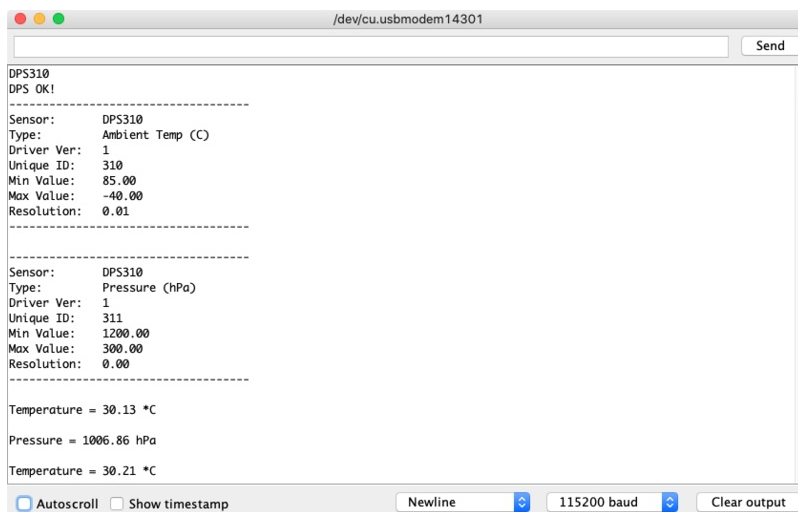
Using On-board Pressure Sensor

There's a pre-soldered pressure sensor that you can use.

You can test the DPS310 by loading the included `adafruit_sensor_test` in the Arduino library



Now you can upload, reset, and check the serial port for ambient temperature and pressure data!

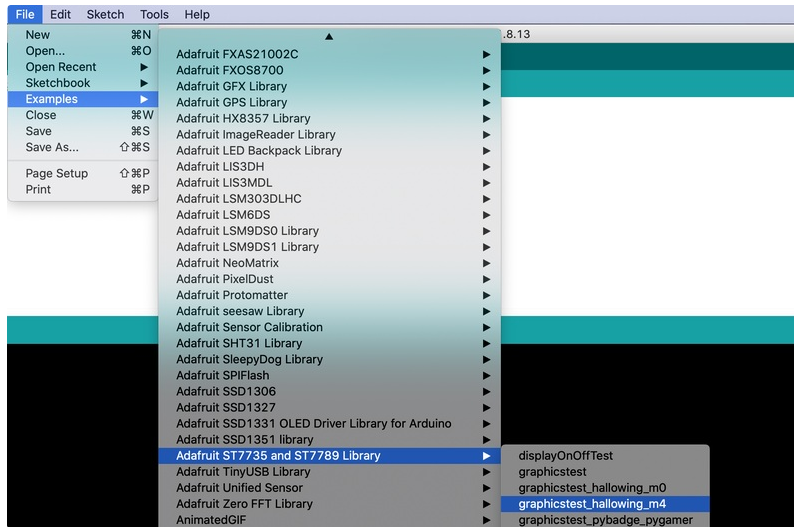


Using the TFT Display

You've been so patient, it's time to draw to the display!

We'll be using a demo that was written for the HalloWing M4, which has the same display, so only a couple of minor changes are needed.

From the `Adafruit ST7735 and ST7789 Library` folder, open the `graphicstest_hallowing_m4` example



Remove the pin definitions near the top since they are now part of the FunHouse Board Support Package.

```
#define TFT_CS      44 // PyBadge/PyGamer display control pins: chip select
#define TFT_RST    46 // Display reset
#define TFT_DC     45 // Display data/command select
#define TFT_BACKLIGHT 47 // Display backlight pin
```

Change the initialization line to the following:

```
Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RESET);
```

You can now upload the example to your FunHouse to see it display various graphics and text tests.

For more information on how to [display graphics and text, check out the Adafruit GFX guide \(https://adafru.it/doL\)](https://adafru.it/doL)

Arduino Self Test Example

The `selftest.ino` sketch will initialize all the sensors and respond to the buttons and capacitive touch pads. Go ahead and upload it to your FunHouse and start pressing buttons. This will also slowly color-cycle the DotStar LEDs and blink the red LED.

<https://adafru.it/RWB>

<https://adafru.it/RWB>

```
#include <Adafruit_DotStar.h>
#include <Adafruit_GFX.h> // Core graphics library
#include <Adafruit_ST7789.h> // Hardware-specific library for ST7789
#include <Adafruit_DPS310.h>
#include <Adafruit_AHTX0.h>

#define NUM_DOTSTAR 5
#define BG_COLOR ST77XX_BLACK

// display!
Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RESET);
// LEDs!
Adafruit_DotStar pixels(NUM_DOTSTAR, PIN_DOTSTAR_DATA, PIN_DOTSTAR_CLOCK, DOTSTAR_BRG);
// sensors!
Adafruit_DPS310 dps;
Adafruit_AHTX0 aht;

uint8_t LED_dutycycle = 0;
uint16_t firstPixelHue = 0;

void setup() {
  Serial.begin(115200);
  delay(100);

  pixels.begin(); // Initialize pins for output
  pixels.show(); // Turn all LEDs off ASAP
  pixels.setBrightness(20);

  pinMode(BUTTON_DOWN, INPUT_PULLDOWN);
  pinMode(BUTTON_SELECT, INPUT_PULLDOWN);
  pinMode(BUTTON_UP, INPUT_PULLDOWN);

  //analogReadResolution(13);

  tft.init(240, 240); // Initialize ST7789 screen
  pinMode(TFT_BACKLIGHT, OUTPUT);
  digitalWrite(TFT_BACKLIGHT, HIGH); // Backlight on

  tft.fillScreen(BG_COLOR);
  tft.setTextSize(2);
  tft.setTextColor(ST77XX_YELLOW);
  tft.setTextWrap(false);

  // check DPS!
  tft.setCursor(0, 0);
  tft.setTextColor(ST77XX_YELLOW);
  tft.print("DP310? ");

  if (!dps.begin_I2C()) {
    tft.setTextColor(ST77XX_RED);
    tft.println("FAIL!");
    while (1) delay(100);
  }
  tft.setTextColor(ST77XX_GREEN);
  tft.println("OK!");
  dps.configurePressure(DPS310_64HZ, DPS310_64SAMPLES);
  dps.configureTemperature(DPS310_64HZ, DPS310_64SAMPLES);

  // check AHT!
  tft.setCursor(0, 20);
  tft.setTextColor(ST77XX_YELLOW);
  tft.print("AHT20? ");

  if (!aht.begin()) {
    tft.setTextColor(ST77XX_RED);
    tft.println("FAIL!");
    while (1) delay(100);
  }
  tft.setTextColor(ST77XX_GREEN);
  tft.println("OK!");

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(SPEAKER, OUTPUT);

  ledcSetup(0, 2000 * 80, 8);
  ledcAttachPin(LED_BUILTIN, 0);
```

```

ledcSetup(1, 2000 * 80, 8);
ledcAttachPin(SPEAKER, 1);
ledcWrite(1, 0);
}

void loop() {

/***** sensors */
sensors_event_t humidity, temp, pressure;

tft.setCursor(0, 0);
tft.setTextColor(ST77XX_YELLOW, BG_COLOR);
dps.getEvents(&temp, &pressure);

tft.print("DP310: ");
tft.print(temp.temperature, 0);
tft.print(" C ");
tft.print(pressure.pressure, 0);
tft.print(" hPa");
tft.println(" ");
Serial.printf("DPS310: %0.1f *C %0.2f hPa\n", temp.temperature, pressure.pressure);

tft.setCursor(0, 20);
tft.setTextColor(ST77XX_YELLOW, BG_COLOR);
aht.getEvent(&humidity, &temp);

tft.print("AHT20: ");
tft.print(temp.temperature, 0);
tft.print(" C ");
tft.print(humidity.relative_humidity, 0);
tft.print(" %");
tft.println(" ");
Serial.printf("AHT20: %0.1f *C %0.2f rH\n", temp.temperature, humidity.relative_humidity);

/***** BUTTONS */
tft.setCursor(0, 40);
tft.setTextColor(ST77XX_YELLOW);
tft.print("Buttons: ");
if (!digitalRead(BUTTON_DOWN)) {
  tft.setTextColor(0x808080);
} else {
  Serial.println("DOWN pressed");
  tft.setTextColor(ST77XX_WHITE);
}
tft.print("DOWN ");

if (!digitalRead(BUTTON_SELECT)) {
  tft.setTextColor(0x808080);
} else {
  Serial.println("SELECT pressed");
  tft.setTextColor(ST77XX_WHITE);
}
tft.print("SEL ");

if (!digitalRead(BUTTON_UP)) {
  tft.setTextColor(0x808080);
} else {
  Serial.println("UP pressed");
  tft.setTextColor(ST77XX_WHITE);
}
tft.println("UP");

/***** CAPACITIVE */
uint16_t touchread;

tft.setCursor(0, 60);
tft.setTextColor(ST77XX_YELLOW, BG_COLOR);
tft.print("Captouch 6: ");
touchread = touchRead(6);
if (touchread < 10000) {
  tft.setTextColor(0x808080, BG_COLOR);
} else {
  tft.setTextColor(ST77XX_WHITE, BG_COLOR);
}
tft.print(touchread);
tft.println(" ");
Serial.printf("Captouch #6 reading: %d\n", touchread);

tft.setCursor(0, 80);
tft.setTextColor(ST77XX_YELLOW, BG_COLOR);
tft.print("Captouch 7: ");
touchread = touchRead(7);
if (touchread < 20000) {
  tft.setTextColor(0x808080, BG_COLOR);
} else {

```

```

    tft.setTextColor(ST77XX_WHITE, BG_COLOR);
}
tft.print(touchread);
tft.println(" ");
Serial.printf("Captouch #7 reading: %d\n", touchread);

tft.setCursor(0, 100);
tft.setTextColor(ST77XX_YELLOW, BG_COLOR);
tft.print("Captouch 8: ");
touchread = touchRead(8);
if (touchread < 20000 ) {
    tft.setTextColor(0x808080, BG_COLOR);
} else {
    tft.setTextColor(ST77XX_WHITE, BG_COLOR);
}
tft.print(touchread);
tft.println(" ");
Serial.printf("Captouch #8 reading: %d\n", touchread);

/***** ANALOG READ */
uint16_t analogread;

tft.setCursor(0, 120);
tft.setTextColor(ST77XX_YELLOW, BG_COLOR);
tft.print("Analog 0: ");
analogread = analogRead(A0);
if (analogread < 8000 ) {
    tft.setTextColor(ST77XX_WHITE, BG_COLOR);
} else {
    tft.setTextColor(ST77XX_RED, BG_COLOR);
}
tft.print(analogread);
tft.println(" ");
Serial.printf("Analog A0 reading: %d\n", analogread);

tft.setCursor(0, 140);
tft.setTextColor(ST77XX_YELLOW, BG_COLOR);
tft.print("Analog 1: ");
analogread = analogRead(A1);
if (analogread < 8000 ) {
    tft.setTextColor(ST77XX_WHITE, BG_COLOR);
} else {
    tft.setTextColor(ST77XX_RED, BG_COLOR);
}
tft.print(analogread);
tft.println(" ");
Serial.printf("Analog A1 reading: %d\n", analogread);

tft.setCursor(0, 160);
tft.setTextColor(ST77XX_YELLOW, BG_COLOR);
tft.print("Analog 2: ");
analogread = analogRead(A2);
if (analogread < 8000 ) {
    tft.setTextColor(ST77XX_WHITE, BG_COLOR);
} else {
    tft.setTextColor(ST77XX_RED, BG_COLOR);
}
tft.print(analogread);
tft.println(" ");
Serial.printf("Analog A2 reading: %d\n", analogread);

tft.setCursor(0, 180);
tft.setTextColor(ST77XX_YELLOW, BG_COLOR);
tft.print("Light: ");
analogread = analogRead(A3);
tft.setTextColor(ST77XX_WHITE, BG_COLOR);
tft.print(analogread);
tft.println(" ");
Serial.printf("Light sensor reading: %d\n", analogread);

/***** Beep! */
if (digitalRead(BUTTON_SELECT)) {
    Serial.println("*** Beep! ***");
    tone(SPEAKER, 988, 100); // tone1 - B5
    tone(SPEAKER, 1319, 200); // tone2 - E6
    delay(100);
    //tone(SPEAKER, 2000, 100);
}

/***** LEDs */
// pulse red LED
ledcWrite(0, LED_dutycycle);
LED_dutycycle += 32;

// rainbow dotstars
for (int i=0; i<pixels.numPixels(); i++) { // For each pixel in strip...
    int pixelHue = firstPixelHue + (i * 65536L / pixels.numPixels());

```

```
pixels.setPixelColor(i, pixels.gamma32(pixels.ColorHSV(pixelHue)));
}
pixels.show(); // Update strip with new contents
firstPixelHue += 256;
}

void tone(uint8_t pin, float frequency, float duration) {
  ledcSetup(1, frequency * 80, 8);
  ledcAttachPin(pin, 1);
  ledcWrite(1, 128);
  delay(duration);
  ledcWrite(1, 0);
}
```

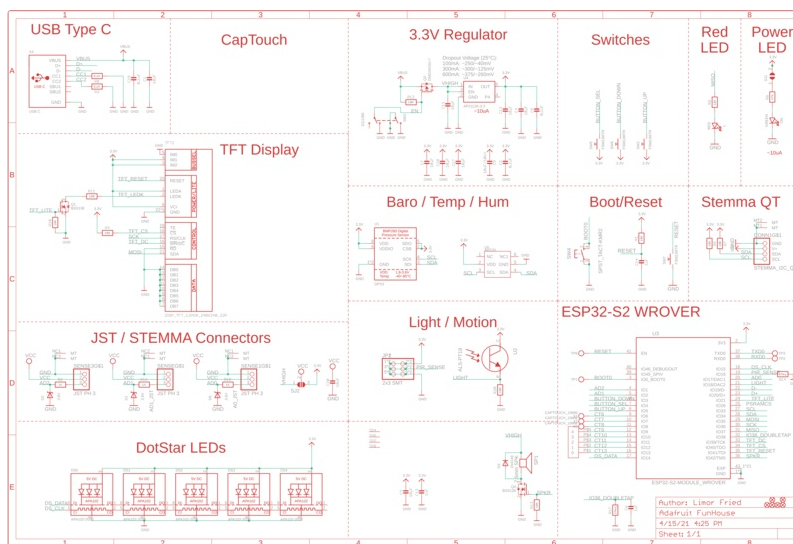

Downloads

Files:

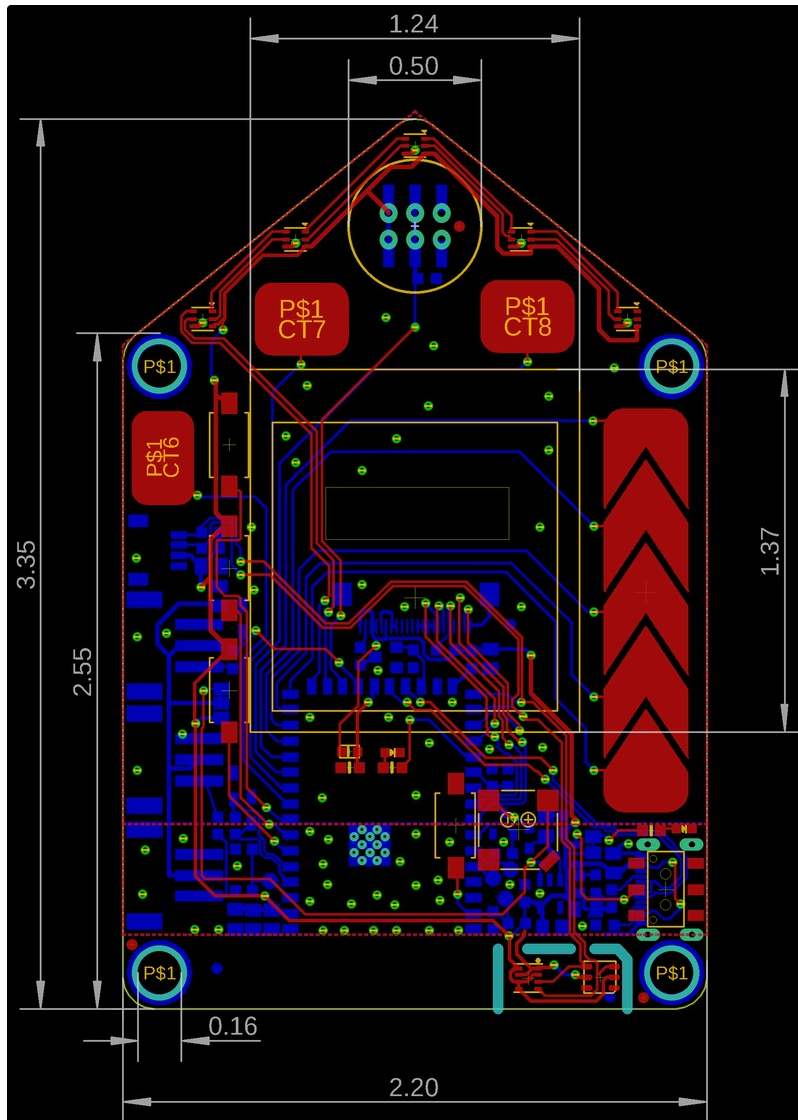
- [ESP32-S2 product page with resources](https://adafru.it/OpE) (https://adafru.it/OpE)
- [ESP32-S2 datasheet](https://adafru.it/OpF) (https://adafru.it/OpF)
- [ESP32-S2 WROVER datasheet](https://adafru.it/Oqg) (https://adafru.it/Oqg)
- [ESP32-S2 Technical Reference](https://adafru.it/Oqb) (https://adafru.it/Oqb)
- [DPS310 datasheet](https://adafru.it/RKC) (https://adafru.it/RKC)
- [AHT20 datasheet](https://adafru.it/LAm) (https://adafru.it/LAm)
- [EagleCAD PCB files on GitHub](https://adafru.it/RKD) (https://adafru.it/RKD)
- [3D Models on GitHub](https://adafru.it/Sbt) (https://adafru.it/Sbt)
- [Fritzing object in the Adafruit Fritzing Library](https://adafru.it/RKE) (https://adafru.it/RKE)

Schematic

The schematic shows a BMP280, but the FunHouse has a DPS310, which is pin-compatible.



Fab Print



Here's a design file for a mounting bracket – you can use it as a template for cutting your own by hand, with a laser cutter or mill, or as a jumping off point for modeling one for 3D printing.

<https://adafru.it/Sfy>

<https://adafru.it/Sfy>

