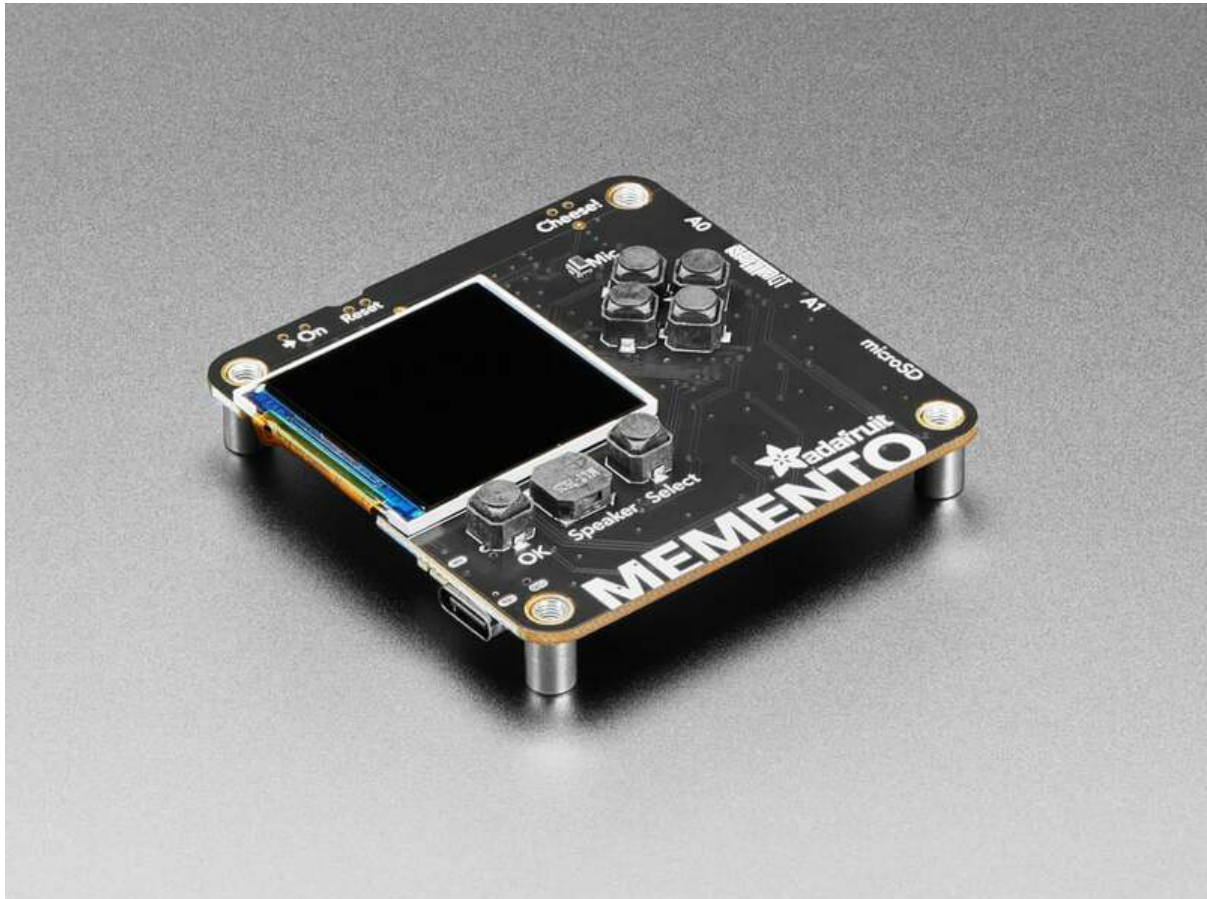




Adafruit MEMENTO Camera Board

Created by Liz Clark



<https://learn.adafruit.com/adafruit-memento-camera-board>

Last updated on 2025-02-08 04:28:24 PM EST

Table of Contents

Overview	7
Pinouts	9
<ul style="list-style-type: none">• Microcontroller and WiFi• OV5640 Camera Module• TFT Screen• Hardware UART• GPIO Expander• Accelerometer• User Buttons• Reset and Boot Buttons• Analog MEMS Microphone• Speaker• JST-PH Connectors• I2C/Stemma QT Connector• microSD Card Slot• NeoPixel• Power• On/Off Switch and Power LED	
Install CircuitPython	21
<ul style="list-style-type: none">• CircuitPython Quickstart	
Installing the Mu Editor	24
<ul style="list-style-type: none">• Download and Install Mu• Starting Up Mu• Using Mu	
The CIRCUITPY Drive	26
<ul style="list-style-type: none">• Boards Without CIRCUITPY	
Creating and Editing Code	27
<ul style="list-style-type: none">• Creating Code• Editing Code• Back to Editing Code...• Naming Your Program File	
Exploring Your First CircuitPython Program	32
<ul style="list-style-type: none">• Imports & Libraries• Setting Up The LED• Loop-de-loops• What Happens When My Code Finishes Running?• What if I Don't Have the Loop?	
Connecting to the Serial Console	35
<ul style="list-style-type: none">• Are you using Mu?• Serial Console Issues or Delays on Linux• Setting Permissions on Linux• Using Something Else?	
Interacting with the Serial Console	38

The REPL	41
<ul style="list-style-type: none">• Entering the REPL• Interacting with the REPL• Returning to the Serial Console	
CircuitPython Libraries	45
<ul style="list-style-type: none">• The Adafruit Learn Guide Project Bundle• The Adafruit CircuitPython Library Bundle• Downloading the Adafruit CircuitPython Library Bundle• The CircuitPython Community Library Bundle• Downloading the CircuitPython Community Library Bundle• Understanding the Bundle• Example Files• Copying Libraries to Your Board• Understanding Which Libraries to Install• Example: ImportError Due to Missing Library• Library Install on Non-Express Boards• Updating CircuitPython Libraries and Examples• CircUp CLI Tool	
CircuitPython Documentation	56
<ul style="list-style-type: none">• CircuitPython Core Documentation• CircuitPython Library Documentation	
Recommended Editors	62
<ul style="list-style-type: none">• Recommended editors• Recommended only with particular settings or add-ons• Editors that are NOT recommended	
Advanced Serial Console on Windows	64
<ul style="list-style-type: none">• Windows 7 and 8.1• What's the COM?• Install Putty	
Advanced Serial Console on Mac	68
<ul style="list-style-type: none">• What's the Port?• Connect with screen	
Advanced Serial Console on Linux	70
<ul style="list-style-type: none">• What's the Port?• Connect with screen• Permissions on Linux	
Frequently Asked Questions	74
<ul style="list-style-type: none">• Using Older Versions• Python Arithmetic• Wireless Connectivity• Asyncio and Interrupts• Status RGB LED• Memory Issues• Unsupported Hardware	
Troubleshooting	80
<ul style="list-style-type: none">• Always Run the Latest Version of CircuitPython and Libraries• I have to continue using CircuitPython 7.x or earlier. Where can I find compatible libraries?• macOS Sonoma before 14.4: Errors Writing to CIRCUITPYmacOS 14.4 - 15.1: Slow Writes to CIRCUITPY	

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

Welcome to the Community!

99

- [Adafruit Discord](#)
- [CircuitPython.org](#)
- [Adafruit GitHub](#)
- [Adafruit Forums](#)
- [Read the Docs](#)

microSD Card Formatting Notes

107

CircuitPython MEMENTO Starter Projects

108

MEMENTO Camera Quick Start Guide

109

CircuitPython Basic Camera

110

- [SD Card](#)
- [Download the Project Bundle](#)
- [Camera HUD](#)
- [Take a Photo](#)
- [Image Retrieval](#)
- [Change Resolution](#)
- [Effects](#)
- [LED Color](#)
- [More Camera!](#)

Fancy Camera

121

- [Download the Project Bundle](#)
- [Use the Camera](#)
- [Settings](#)

Timelapse	130
<ul style="list-style-type: none">• LAPS• Focus• Start/Stop	
Animated GIF Creation	135
<ul style="list-style-type: none">• GIF Mode• GIF Code• Effects• NeoPixel Lighting• Post Processing	
Stop Motion	138
<ul style="list-style-type: none">• Onion Skinning• Wave	
Frames to GIFs	141
<ul style="list-style-type: none">• GIF Maker• Select Files• Upload Files• Frame Order, Delay• Make a GIF• Save Your GIF	
Arduino IDE Setup	145
Arduino MEMENTO Library Installation and Starter Projects	146
<ul style="list-style-type: none">• Library Installation	
PyCamera Library Test	147
<ul style="list-style-type: none">• Factory Demo Code	
Basic Camera Example	151
Usage with PlatformIO	153
<ul style="list-style-type: none">• Installation• PyCamera Library Test	
Factory Reset	159
<ul style="list-style-type: none">• Factory Reset Firmware UF2• Factory Reset and Bootloader Repair• Download .bin and Enter Bootloader• Step 1. Download the factory-reset-and-bootloader.bin file• Step 2. Enter ROM bootloader mode• The WebSerial ESPTool Method• Connect• Erase the Contents• Program the ESP32-S2/S3• The esptool Method (for advanced users)• Install ESPTool.py• Test the Installation• Connect• Erase the Flash• Installing the Bootloader• Reset the board• Older Versions of Chrome	

- [The Flash an Arduino Sketch Method](#)
- [Arduino IDE Setup](#)
- [Load the Blink Sketch](#)

Downloads

172

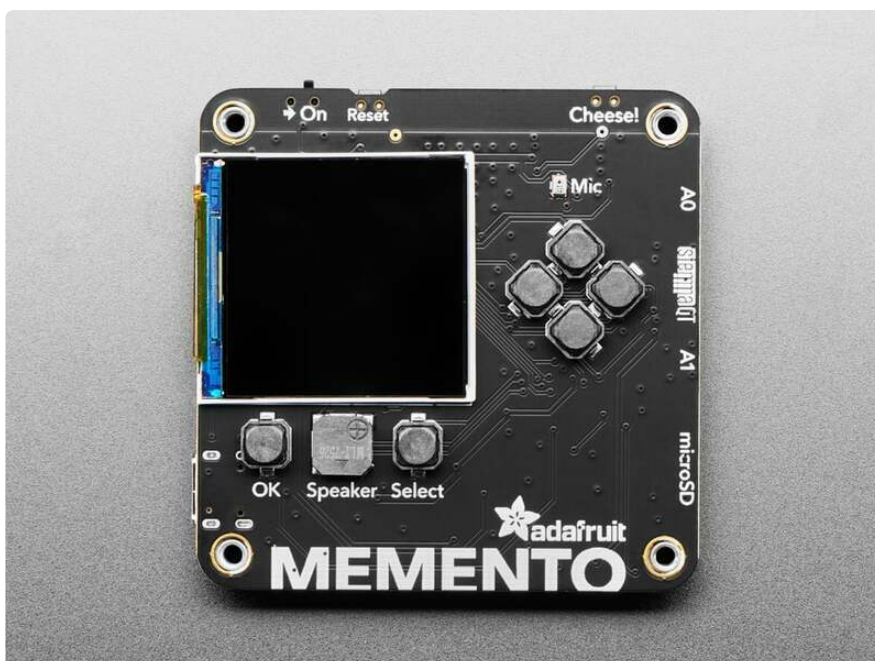
-
- [Files](#)
 - [Schematic and Fab Print](#)
 - [3D Model](#)

Overview

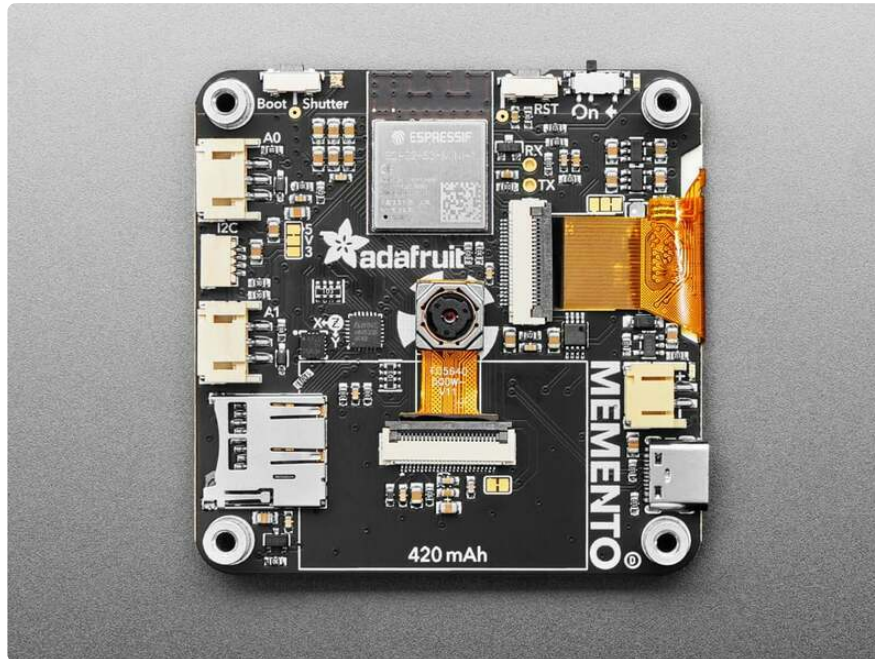


Make memories, or just a cool camera-based project, with **Adafruit's MEMENTO Camera Board**. It's a development board with everything you need to create programmable camera and vision projects: with a camera module, TFT preview screen, buttons, SD card slot and driven by a powerful ESP32-S3 with 2 MB of PSRAM for buffering 5 MegaPixel camera images.

This product is just the mainboard, and does not come with an enclosure, LED ring, hardware, SD card, or battery.



The ESP32-S3 is a WiFi and Bluetooth LE capable, 240 MHz dual core Tensilica processor - much like the famous ESP32. The S3 adds native USB support so it's great for use with Arduino or CircuitPython. The S3 also has the ability to interface with raw camera modules. The cameras require 12 GPIO pins and fast data transfer in order to get images off the sensor, and then a lot of memory for storing 2560 x 1920 images - which is why we picked an S3 module with 2MB of PSRAM so that we can read JPEGs into memory for saving onto a microSD card with up to 32GB capacity.



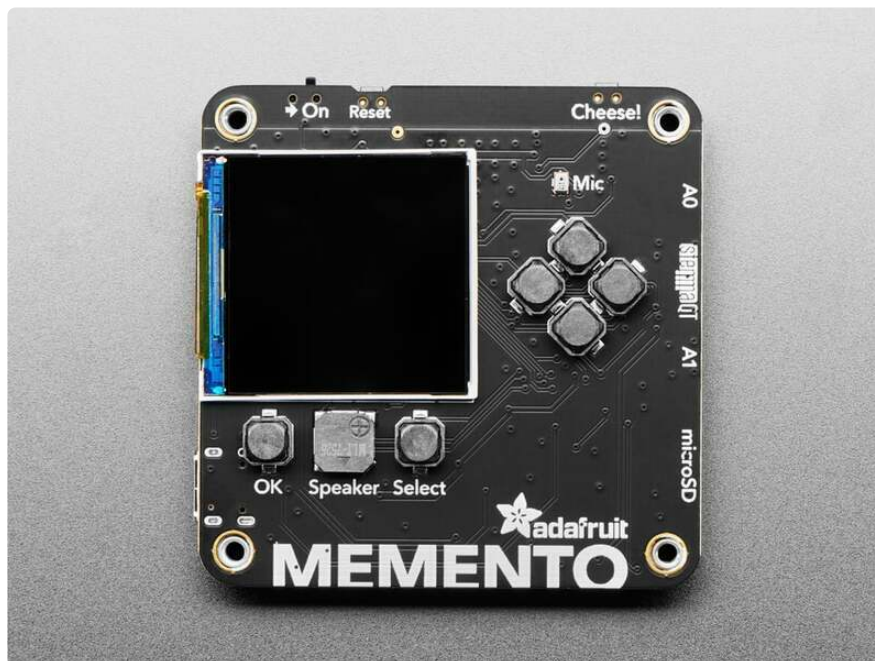
To make the board easy to use we added a ton of supporting hardware, here's a full list of the hardware included:

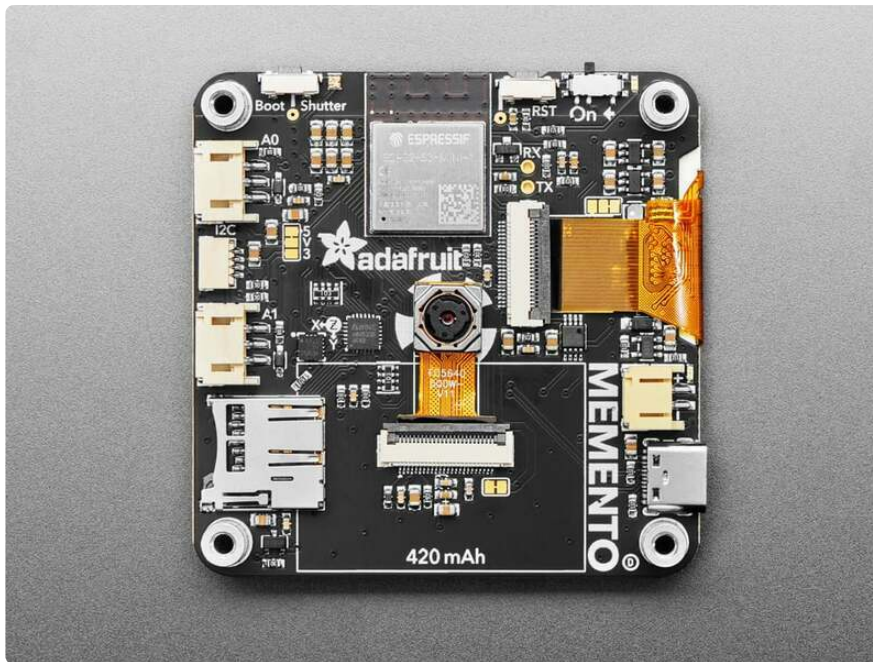
- **ESP32-S3 module with 4 MB Flash, 2 MB PSRAM** - dual core 240MHz Tensilica with WiFi and BTLE.
- **OV5640 camera module with 72 degree view and auto-focus motor** (<http://adafru.it/5840>) - 5MP camera sensor with JPEG encoder built in.
- **1.54" 240x240 Color TFT** (<http://adafru.it/4421>) - For previewing the camera images, or user interface design.
- **MicroSD card slot** - Store images or animations to any SPI-capable micro SD card. NOTE: You can use cards with up to 32GB capacity.
- **Two Digital/Analog Stemma Ports** - JST PH-3 connectors for A0, A1 and power+ground for adding external buttons, LEDs, or sensors. Can provide 3V or 5V power.
- **I2C Stemma QT Port** - Connect just about any I2C sensor you please with a Stemma QT JST SH port, provides 3.3V power and logic.
- **LIS3DH Accelerometer** - Triple-axis accelerometer can detect orientation, shaking or movement.

- **LiPoly battery charging support** - [Use a 3.7/4.2V 350mA](http://adafru.it/4237) (<http://adafru.it/4237>) or [420mA battery](http://adafru.it/4236) (<http://adafru.it/4236>) for on-the-go snaps.
- **6 User Buttons** - change modes, preview saved images, play DOOM (?).
Connected through a GPIO expander.
- **Buzzer** - play tones or alerts, or indicate when a photo was successfully taken.
- **Analog Microphone** - Can be used as a sensor to detect loud sounds, not for recording video with audio.
- **Shutter button** - Connected to GPIO 0 for entering the ROM bootloader.
- **Reset button** - For entering the bootloader or starting over.
- **On/Off switch** - Cut all power when using a battery.
- **USB Type C** for programming the ESP32-S3, as well as REPL access in CircuitPython and charging the optional LiPoly battery.
- **Breakout pads for hardware UART** - for more intense debugging needs, solder wires to the through-hole pads to connect to a console cable.
- **Four M3 standoffs** for mounting or enclosure attachment.

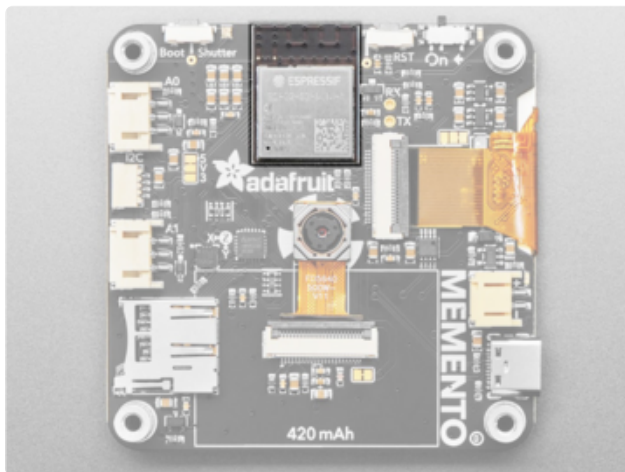
We've got both Arduino and CircuitPython example code that lets you preview the camera, adjust settings, and take photos that are saved to disk. However, we recommend CircuitPython because the compilation time in Arduino is pretty intense due to the huge amount of code required to run the camera. [CircuitPython is fast to develop for and our library will make it easy to start making custom camera projects](https://adafru.it/18e3) (<https://adafru.it/18e3>).

Pinouts





Microcontroller and WiFi

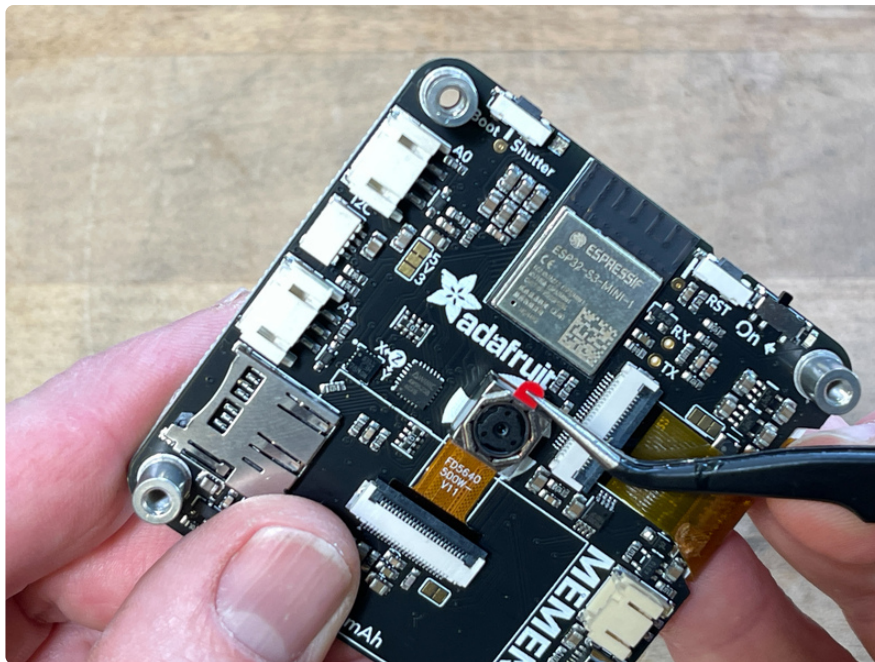


The main processor chip is the **Espressif ESP32-S3** with 3.3v logic/power. It has **4MB** of Flash and **2MB** of **PSRAM**.

The ESP32-S3 comes with WiFi and Bluetooth LE baked right in, though CircuitPython only supports WiFi at this time, not BLE on the S3 chip.

OV5640 Camera Module

Be sure to peel off the protective film that covers the lens!



In the center of the board is the **OV5640 camera module**. It has a 72 degree view, 5MP sensor and an auto-focus motor.

Below the camera connector is a **jumper** for providing 3.3V to the auto-focus motor. You can cut the jumper to disconnect DATA1 from 3.3V to disable auto-focus.

The camera utilizes the following pins:

VSYNC - GPIO5. Accessible in CircuitPython with `board.CAMERA_VSYNC` and Arduino with `VSYNC_GPIO_NUM`.

HREF - GPIO6. Accessible in CircuitPython with `board.CAMERA_HREF` and Arduino with `HREF_GPIO_NUM`.

XCLK - GPIO8. Accessible in CircuitPython with `board.CAMERA_XCLK` and Arduino with `XCLK_GPIO_NUM`.

PCLK - GPIO11. Accessible in CircuitPython with `board.CAMERA_PCLK` and Arduino with `PCLK_GPIO_NUM`.

PWDN - GPIO21. Accessible in CircuitPython with `board.CAMERA_PWDN` and Arduino with `PWDN_GPIO_NUM`.

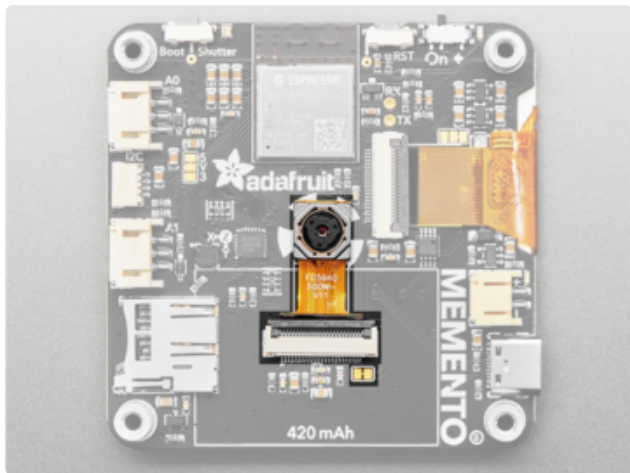
DATA2 - GPIO13. Accessible in CircuitPython with `board.CAMERA_DATA2` and Arduino with `Y2_GPIO_NUM`.

DATA3 - GPIO15. Accessible in CircuitPython with `board.CAMERA_DATA3` and Arduino with `Y3_GPIO_NUM`.

DATA4 - GPIO16. Accessible in CircuitPython with `board.CAMERA_DATA4` and Arduino with `Y4_GPIO_NUM`.

DATA5 - GPIO14. Accessible in CircuitPython with `board.CAMERA_DATA5` and Arduino with `Y5_GPIO_NUM`.

DATA6 - GPIO12. Accessible in CircuitPython with `board.CAMERA_DATA6` and Arduino with `Y6_GPIO_NUM`.



DATA7 - GPIO10. Accessible in CircuitPython with `board.CAMERA_DATA7` and Arduino with `Y7_GPIO_NUM`.

DATA8 - GPIO9. Accessible in CircuitPython with `board.CAMERA_DATA8` and Arduino with `Y8_GPIO_NUM`.

DATA9 - GPIO7. Accessible in CircuitPython with `board.CAMERA_DATA9` and Arduino with `Y9_GPIO_NUM`.

RESET - GPIO47. Accessible in CircuitPython with `board.CAMERA_RESET` and Arduino with `RESET_GPIO_NUM`.

For more information on the OV5640 pins, check out the Adafruit [Learn Guide for the module](https://adafru.it/18e4) (<https://adafru.it/18e4>).

TFT Screen



On the front of MEMENTO is the **1.54" 240x240 Color TFT**. It can be used for previewing the camera images or user interface design. It uses the ST7789 driver, which has both Arduino and CircuitPython support. It uses the following pins for SPI communication:

MOSI - GPIO35. Accessible in CircuitPython with `board.MOSI` and Arduino with `MOSI`.

SCK - GPIO36. Accessible in CircuitPython with `board.SCK` and Arduino with `SCK`.

MISO - GPIO37. Accessible in CircuitPython with `board.MISO` and Arduino with `MISO`.

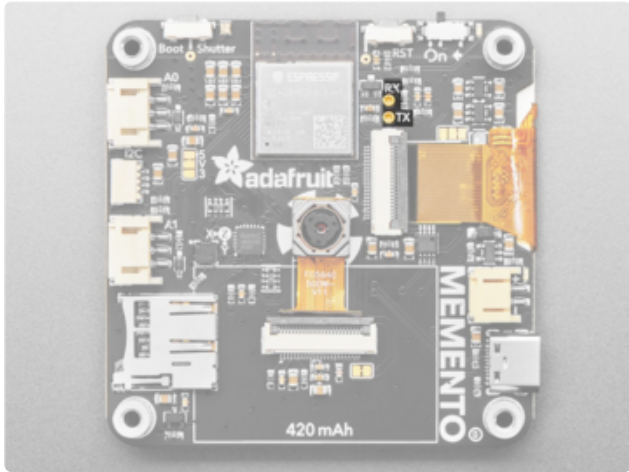
DC - GPIO40. Accessible in CircuitPython with `board.TFT_DC` and Arduino with `TFT_DC`.

CS - GPIO39. Accessible in CircuitPython with `board.TFT_CS` and Arduino with `TFT_CS`.

Backlight - GPIO45. Accessible in CircuitPython with `board.TFT_BACKLIGHT` and Arduino with `TFT_BACKLIGHT`.

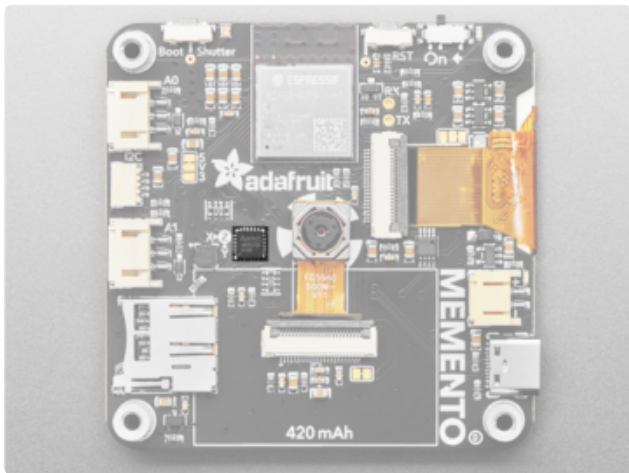
Reset - GPIO38. Accessible in CircuitPython with `board.TFT_RESET` and Arduino with `TFT_RESET` or `TFT_RST`.

Hardware UART



To the right of the ESP32-S3 module and above the TFT connector are the breakout pads for **hardware UART**. These pins, labeled **RX** and **TX** on the board silk, are for more intense debugging needs. You can solder wires to the through-hole pads to connect to a console cable.

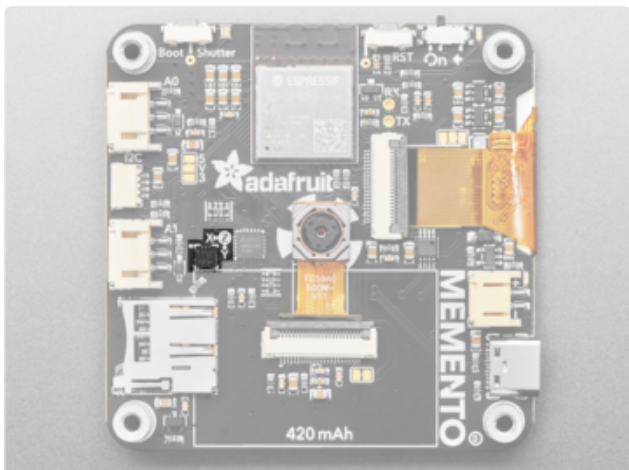
GPIO Expander



The MEMENTO includes an **AW9523 GPIO Expander**. The IO Expander is connected via the I2C bus. The main purpose of the expander is to add additional pins to communicate with the buttons

The I2C address of the GPIO expander is **0x58**.

Accelerometer



On the back of the board, to the left of the GPIO expander, is an **LIS3DH accelerometer**. It is a triple-access accelerometer that can detect orientation, shaking or movement. It communicates over I2C on address **0x19**. Its interrupt (**IRQ**) pin is connected to GPIO3, which is accessible in CircuitPython with `board.IRQ` and Arduino with `3`.

User Buttons

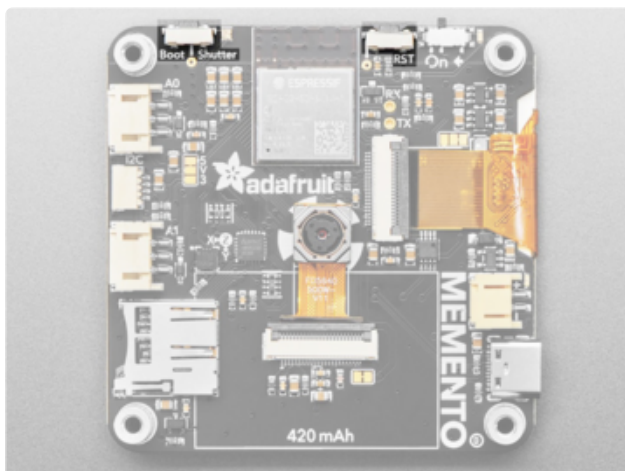


On the front of the board are six user buttons. They are connected through the **AW9523 GPIO expander**:

- OK button - pin 11
- Select button - pin 1
- Up button - pin 13
- Down button - pin 15
- Left button - pin 14
- Right button - pin 12

Both the [Arduino](https://adafru.it/18e5) (<https://adafru.it/18e5>) and [CircuitPython](https://adafru.it/18e3) (<https://adafru.it/18e3>) PyCamera libraries have built-in support for these buttons to easily access them.

Reset and Boot Buttons



Along the top edge of the board are two buttons. The **boot button**, labeled **Boot** and **Shutter** on the board silk, is connected to **GPIO0**. It's accessible in CircuitPython with `board.BUTTON` and Arduino with `SHUTTER_BUTTON`. You'll hold it while pressing reset to enter ROM Bootloader mode.

The **reset button**, labeled **RST** on the board silk, is connected to the ESP32-S3 reset pin. Click it once to restart your firmware. Click it again after about a half second to enter bootloader mode.

Analog MEMS Microphone



On the front of MEMENTO, above the cluster of four user buttons, is the **analog MEMS microphone**. It is labeled **Mic** on the board silk. It can be used as a sensor to detect loud sounds. It is **not** for recording video with audio. The analog input from the microphone is connected to GPIO2. It's accessible in CircuitPython with `board.MIC` and Arduino with `2`.

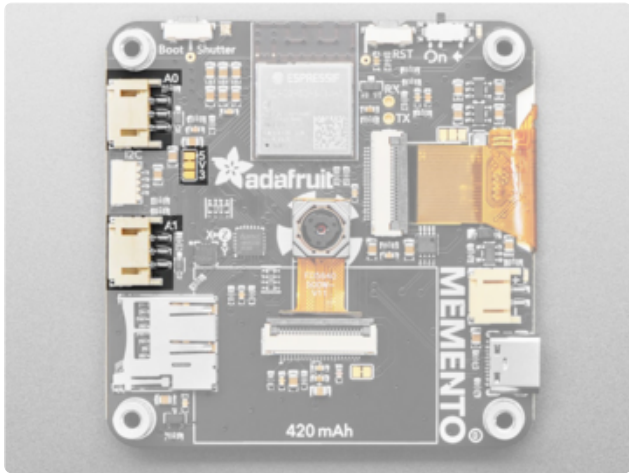
Speaker



Below the TFT display is the **speaker**. It is labeled **Speaker** on the board silk. The output from the speaker is connected to GPIO46. It is accessible in CircuitPython with `board.SPEAKER` and Arduino with `SPEAKER`.

The speaker is muted via pin 0 on the AW9523 GPIO expander, so don't forget to de-mute before expecting it to make noise!

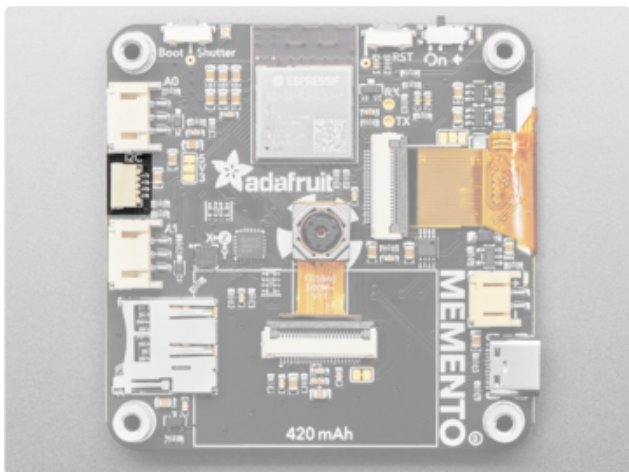
JST-PH Connectors



On the left side of the board are connectors labeled **A0** and **A1**. These are **3-pin JST connectors** for sensors, NeoPixels, or analog output or input. A0 is connected to GPIO17 (accessible with `board.A0` in CircuitPython and `A0` in Arduino) and A1 is connected to GPIO18 (accessible with `board.A1` in CircuitPython and `A1` in Arduino).

Between both connectors is a jumper labeled **5V3**. It can be cut and soldered to use **3V** instead of **5V** for the VCC signal on the JST connectors.

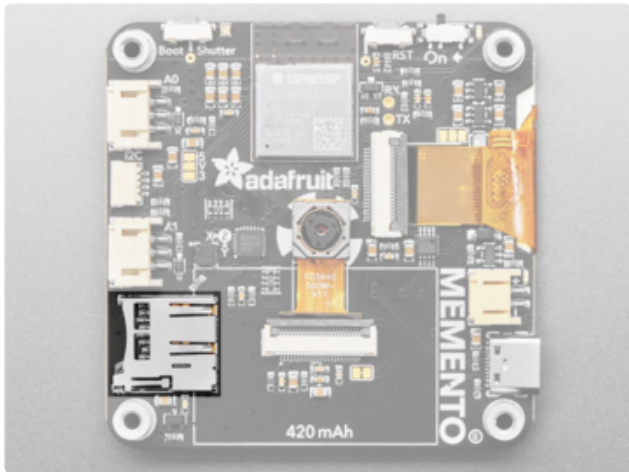
I2C/Stemma QT Connector



There is a 4-pin **Stemma QT connector** on the left. The I2C has pullups to 3.3V power.

The I2C pins are connected to GPIO34 (SDA) and GPIO33 (SCL). In Arduino, you can use the STEMMA connector with `Wire`. In CircuitPython, you can use the STEMMA connector with `board.SCL` and `board.SDA`, `board.I2C()` or `board.STEMMA_I2C()`.

microSD Card Slot

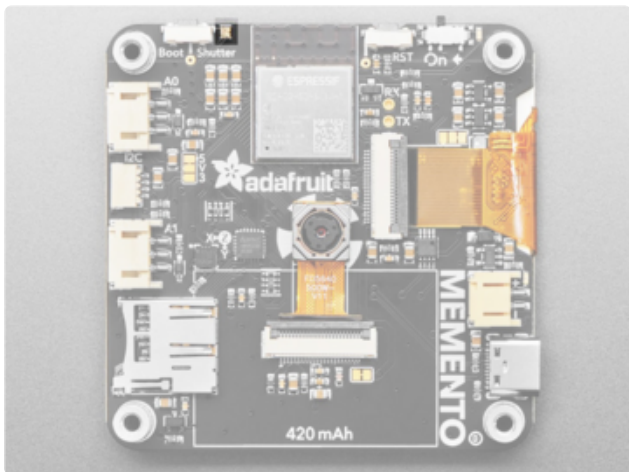


Towards the bottom left corner on the back of MEMENTO is the **microSD card slot**. You can store images or animations to any SPI-capable micro SD card.

SDCS (chip select pin) - connected to GPIO48. It is accessible in CircuitPython with `board.CARD_CS` and Arduino with `SD_CS` or `SD_CHIP_SELECT`.

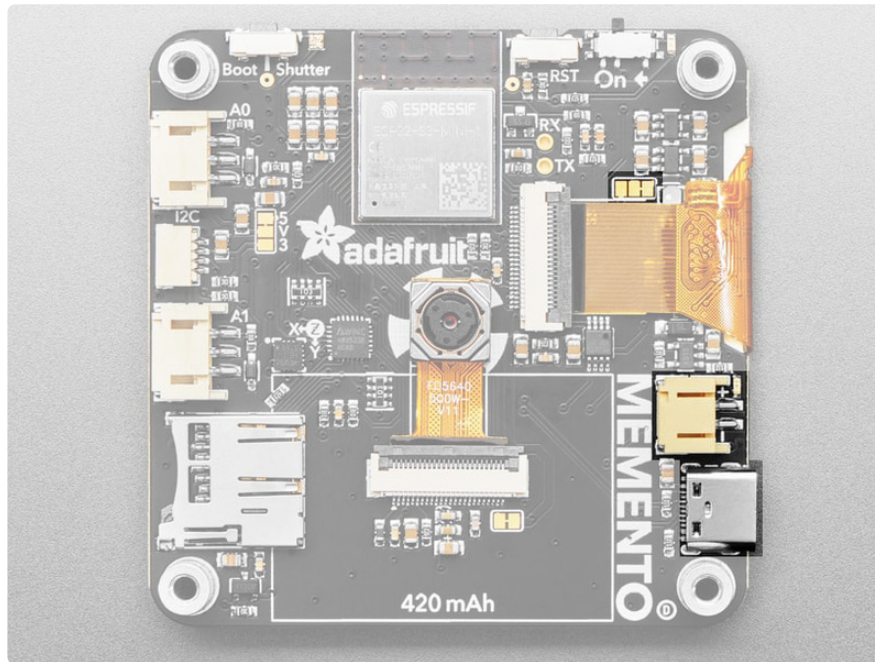
SDCD (card detect pin) - connected to pin 9 on the AW9523.

NeoPixel

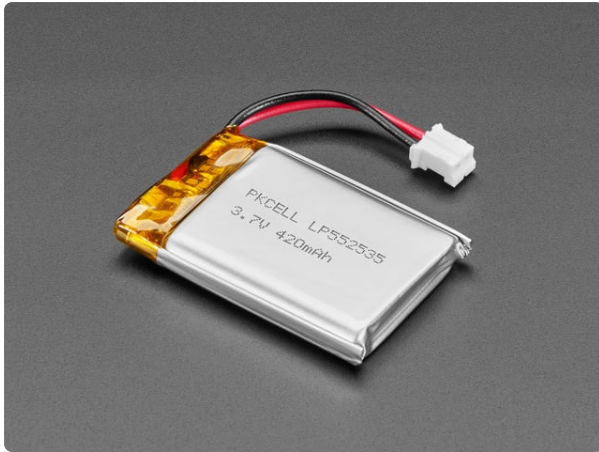


At the top of the board, to the right of the boot button, is the **NeoPixel**. This addressable RGB LED works both as a status LED (in CircuitPython and the bootloader), and can be controlled with code. It is connected to GPIO1 and is available in CircuitPython as `board.NEOPIXEL` and in Arduino as `PIN_NEOPIXEL` or `NEOPIXEL_PIN`.

Power



- **USB-C port** - This is used for both powering and programming the board. You can power it with any USB C cable. When USB is plugged in it will charge the LiPoly battery.
- **LiPoly connector/charger** - You can plug in any 350mAh or larger 3.7/4.2V LiPoly battery into this **JST 2-PH port** to both power your MEMENTO and charge the battery. If the battery is plugged in and USB is plugged in, the MEMENTO will power itself from USB and it will charge the battery up. There is an outline on the board silk that is perfectly sized for the 420mAh battery in the shop.
- **Battery Monitor Pin** - You can monitor the battery percentage via an ADC pin accessed with `board.BATTERY_MONITOR` in CircuitPython or `BATT_MONITOR` in Arduino.
- **Power Selection Jumper** - Above the TFT display ribbon cable is the power selection jumper. It selects whether the 2.8V supply is powered from 3.3V or 5V. It can be cut and soldered to use 5V instead of 3.3V.

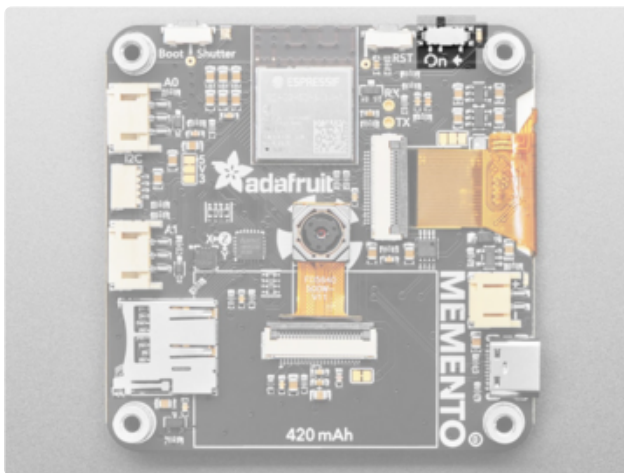


Lithium Ion Polymer Battery with Short Cable - 3.7V 420mAh

Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...

<https://www.adafruit.com/product/4236>

On/Off Switch and Power LED



On the top right edge of the board is the **on/off switch**, labeled **On** on the board silk. This sliding switch can cut all power to the board.

To the right of the switch is the **power LED**. It is a green LED and is lit up when the board has power.

Install CircuitPython

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

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

Make sure you use CircuitPython 9.0.0 final or later. CircuitPython 9.0.0-beta.1 and earlier have a bug that can corrupt the filesystem.

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

Your board will reboot after running this.

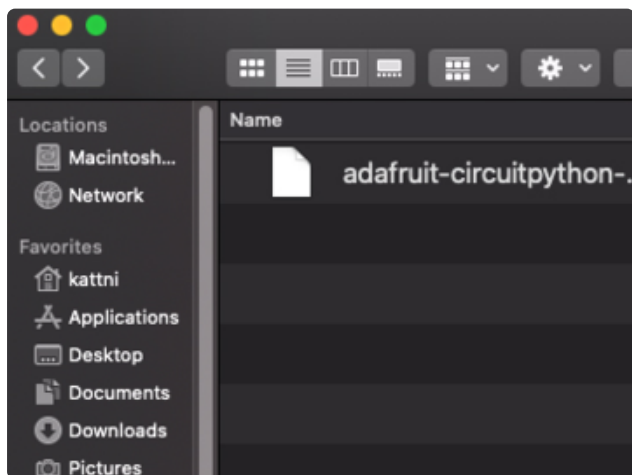
Download the latest version of
CircuitPython for this board via
circuitpython.org

<https://adafru.it/18e6>

As of CircuitPython 9, you'll need to create a folder called "sd" on your CIRCUITPY drive to mount the microSD card, if it's not already there.

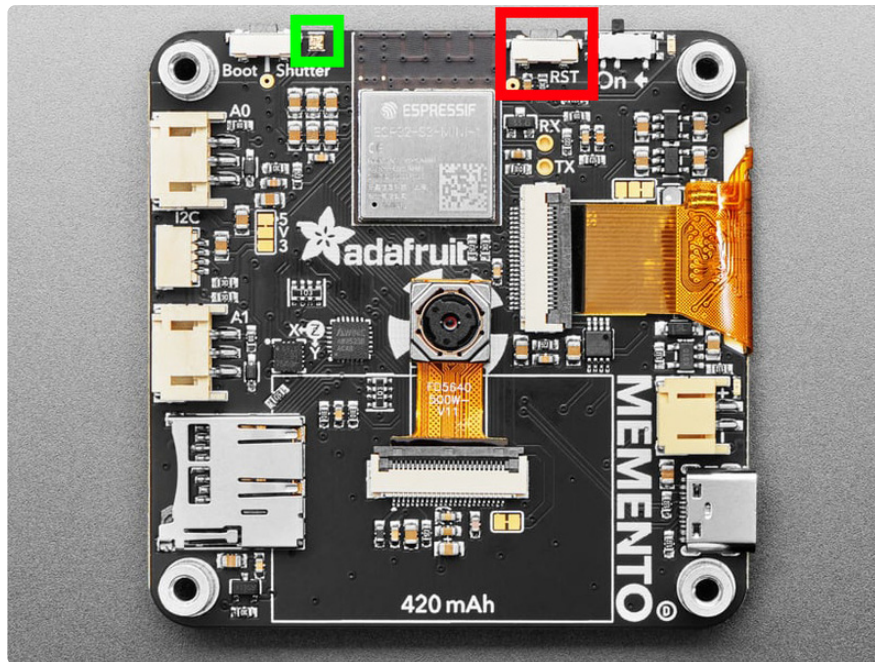
Follow these steps to create the /sd
directory

<https://adafru.it/19ei>



Click the link above to download the
latest CircuitPython UF2 file.

Save it wherever is convenient for you.



Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

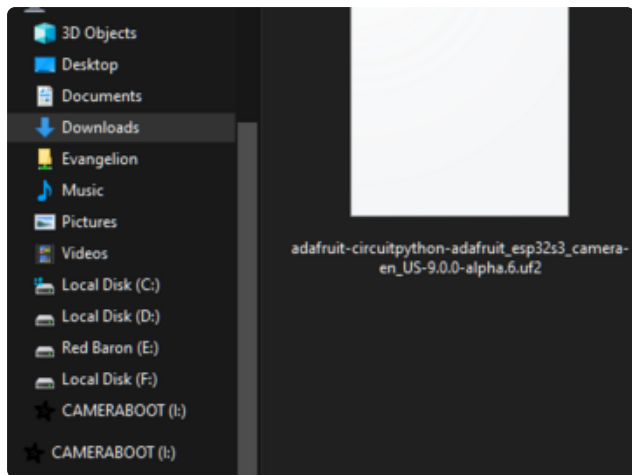
Double-click the **reset** button (highlighted in red above), and you will see the **RGB status LED(s)** turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

For this board, tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again. The second tap needs to happen while the LED is still purple.

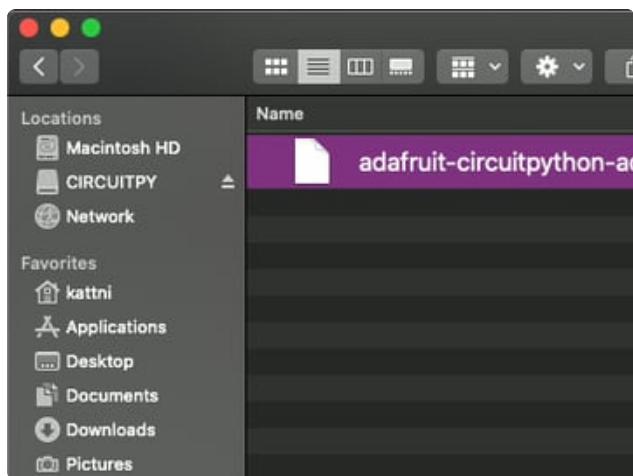
If you do not see the LED turning purple, you will need to reinstall the UF2 bootloader. See the **Factory Reset** page in this guide for details.

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

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



You will see a new disk drive appear called **CAMERABOOT**. Drag the **adafruit-circuitpython-adafruit_esp32s3_camera-etc.uf2** file to **CAMERABOOT**.



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

That's it!

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



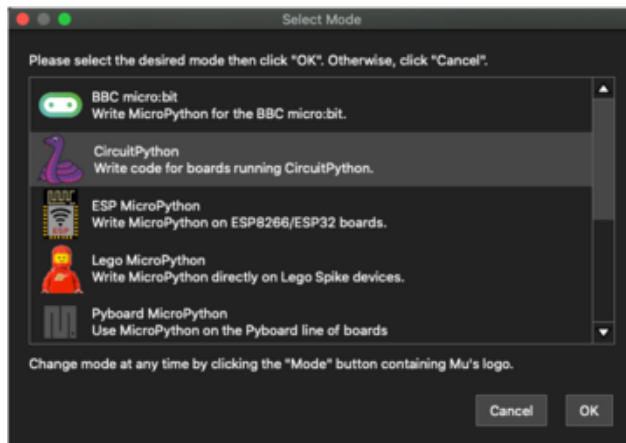
Download Mu from <https://codewith.mu> (<https://adafru.it/Be6>).

Click the **Download** link for downloads and installation instructions.

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

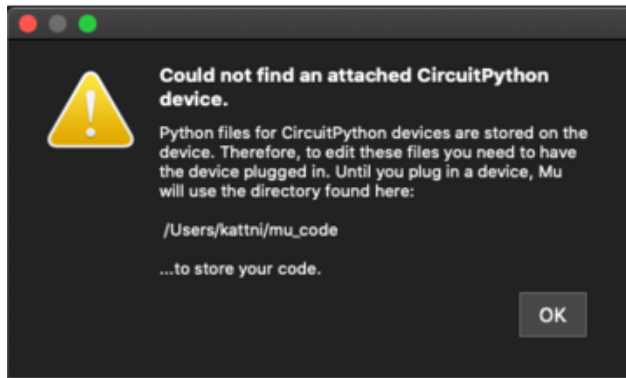
Windows users: due to the nature of MSI installers, please remove old versions of Mu before installing the latest version.

Starting Up Mu



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

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

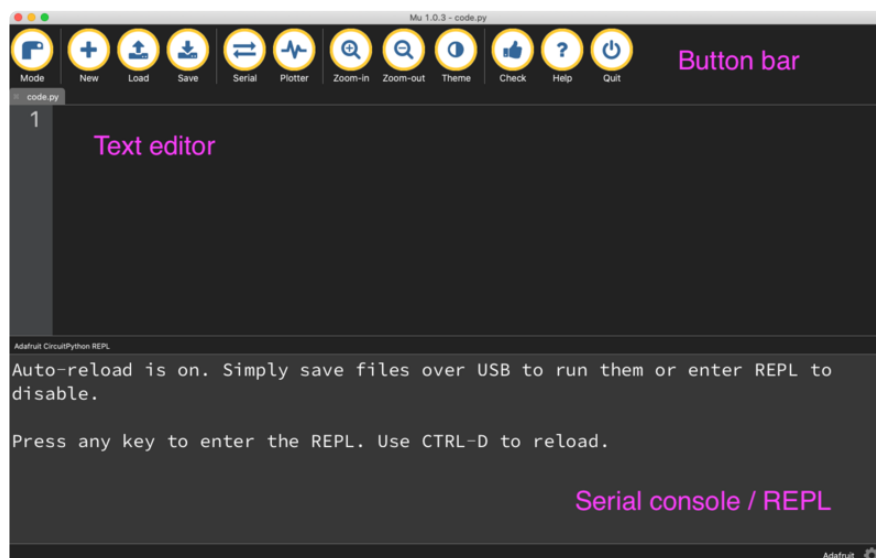


Mu attempts to auto-detect your board on startup, so if you do not have a CircuitPython board plugged in with a **CIRCUITPY** drive available, Mu will inform you where it will store any code you save until you plug in a board.

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

Using Mu

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



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

The CIRCUITPY Drive

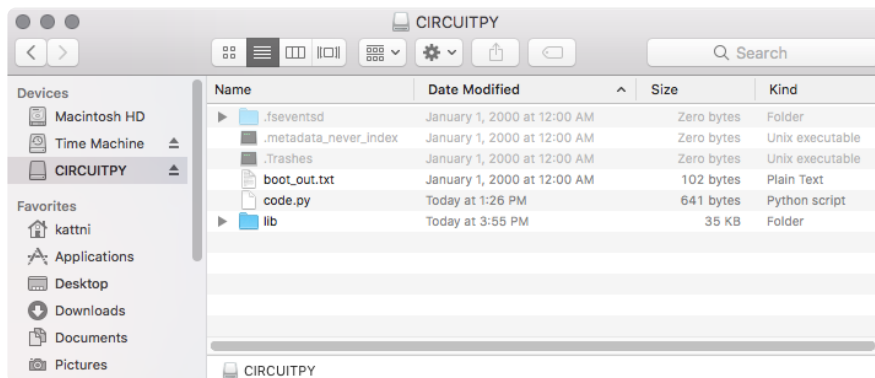
When CircuitPython finishes installing, or you plug a CircuitPython board into your computer with CircuitPython already installed, the board shows up on your computer as a USB drive called **CIRCUITPY**.

The **CIRCUITPY** drive is where your code and the necessary libraries and files will live. You can edit your code directly on this drive and when you save, it will run automatically. When you create and edit code, you'll save your code in a **code.py** file located on the **CIRCUITPY** drive. If you're following along with a Learn guide, you can

paste the contents of the tutorial example into **code.py** on the **CIRCUITPY** drive and save it to run the example.

With a fresh CircuitPython install, on your **CIRCUITPY** drive, you'll find a **code.py** file containing `print("Hello World!")` and an empty **lib** folder. If your **CIRCUITPY** drive does not contain a **code.py** file, you can easily create one and save it to the drive. CircuitPython looks for **code.py** and executes the code within the file automatically when the board starts up or resets. Following a change to the contents of **CIRCUITPY**, such as making a change to the **code.py** file, the board will reset, and the code will be run. You do not need to manually run the code. This is what makes it so easy to get started with your project and update your code!

Note that all changes to the contents of **CIRCUITPY**, such as saving a new file, renaming a current file, or deleting an existing file will trigger a reset of the board.



Boards Without CIRCUITPY

CircuitPython is available for some microcontrollers that do not support native USB. Those boards cannot present a **CIRCUITPY** drive. This includes boards using ESP32 or ESP32-C3 microcontrollers.

On these boards, there are alternative ways to transfer and edit files. You can use the [Thonny editor \(https://adafru.it/18e7\)](https://adafru.it/18e7), which uses hidden commands sent to the REPL to read and write files. Or you can use the CircuitPython web workflow, introduced in Circuitpython 8. The web workflow provides browser-based WiFi access to the CircuitPython filesystem. These guides will help you with the web workflow:

- [CircuitPython on ESP32 Quick Start \(https://adafru.it/10JF\)](https://adafru.it/10JF)
- [CircuitPython Web Workflow Code Editor Quick Start \(https://adafru.it/18e8\)](https://adafru.it/18e8)

Creating and Editing Code

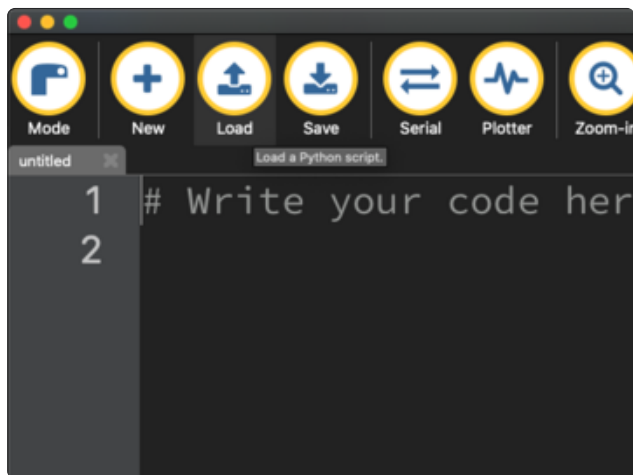
One of the best things about CircuitPython is how simple it is to get code up and running. This section covers how to create and edit your first CircuitPython program.

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

If you don't or can't use Mu, there are a number of other editors that work quite well. The [Recommended Editors page \(https://adafru.it/Vue\)](https://adafru.it/Vue) has more details. Otherwise, make sure you do "Eject" or "Safe Remove" on Windows or "sync" on Linux after writing a file if you aren't using Mu. (This was formerly not a problem on macOS, but see the warning below.)

macOS Sonoma 14.1 introduced a bug that delays writes to small drives such as CIRCUITPY drives. This caused errors when saving files to CIRCUITPY. There is a [workaround](#). The bug was fixed in Sonoma 14.4, but at the cost of greatly slowed writes to drives 1GB or smaller.

Creating Code



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

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

Copy and paste the following code into your editor:

```
import board
import digitalio
import time

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

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


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

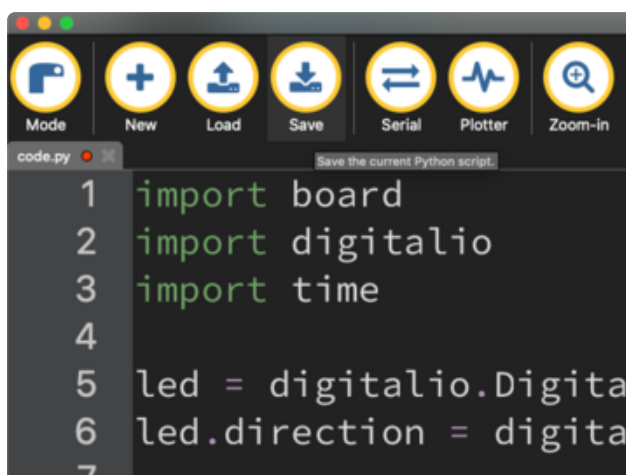
If you're using a KB2040, QT Py, Qualia, or a Trinkey, or any other board without a single-color LED that can blink, please download the [NeoPixel blink example](https://adafru.it/UDU) (<https://adafru.it/UDU>).

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.



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

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



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

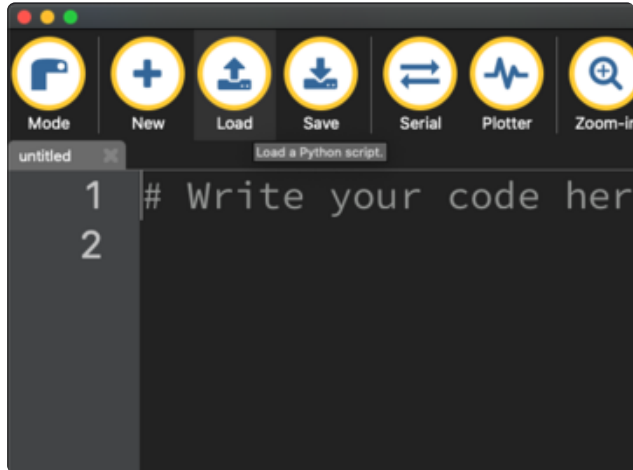
Save the `code.py` file on your **CIRCUITPY** drive.

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

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

On most boards you'll find a tiny red LED. On the ItsyBitsy nRF52840, you'll find a tiny blue LED. On QT Py M0, QT Py RP2040, Qualia, and the Trinky series, you will find only an RGB NeoPixel LED.

Editing Code



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

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

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

There's one warning before you continue...

Don't click reset or unplug your board!

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

There are a couple of ways to avoid filesystem corruption.

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

Check out the [Recommended Editors page \(https://adafru.it/Vue\)](https://adafru.it/Vue) for details on different editing options.

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

2. Eject or Sync the Drive After Writing

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

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

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



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

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

If you are having trouble saving code on Windows 10, try including this code snippet at the top of code.py:

```
import supervisor
supervisor.runtime.autoreload = False
```

Back to Editing Code...

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

```
import board
import digitalio
import time

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

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

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

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

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

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

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

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

Naming Your Program File

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

Exploring Your First CircuitPython Program

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

Here is the original code again for the LED blink example (if your board doesn't have a single-color LED to blink, look instead at the NeoPixel blink example):

```
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 tell the board that you're going to use a particular library or module in your code. In this example, you 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!

These three modules each have a purpose. The first one, `board`, gives you access to the hardware on your board. The second, `digitalio`, lets you access that hardware as inputs/outputs. The third, `time`, lets you control the flow of your code in multiple ways, including passing 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, you initialise that pin, and you set it to output. You set `led` to equal the rest of that information so you 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, you have four items:

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

First, you have `led.value = True`. This line tells the LED to turn on. On the next line, you 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 code snippet above by eliminating the loop entirely, and replacing it with `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 CTRL+C to exit the loop.

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

Connecting to the Serial Console

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

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

This line in your code.py would result in:

```
Hello, world!
```

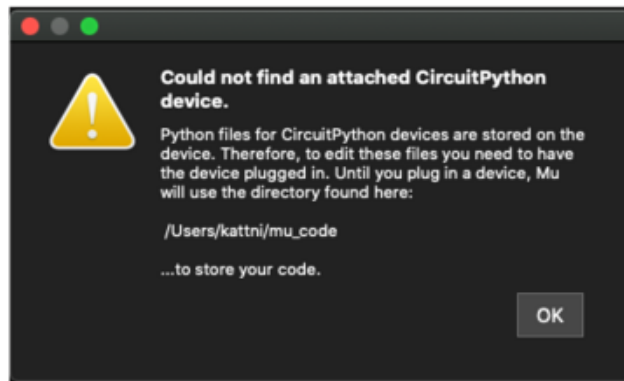
However, these print statements need somewhere to display. That's where the serial console comes in!

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

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

Are you using Mu?

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

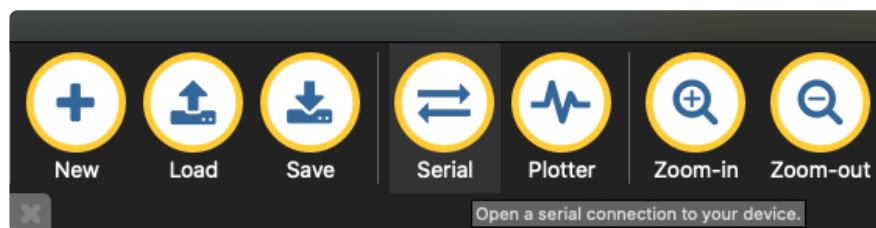


First, make sure your CircuitPython board is plugged in.

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

[If you are using Windows 7, make sure you installed the drivers \(https://adafru.it/VuB\).](https://adafru.it/VuB)

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



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



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

Serial Console Issues or Delays on Linux

If you're on Linux, and are seeing multi-second delays connecting to the serial console, or are seeing "AT" and other gibberish when you connect, then the

`modemmanager` service might be interfering. Just remove it; it doesn't have much use unless you're still using dial-up modems.

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

```
sudo apt purge modemmanager
```

Setting Permissions on Linux

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



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

```
sudo adduser $USER dialout
```

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

Using Something Else?

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

Windows requires you to download a terminal program. [Check out the Advanced Serial Console on Windows page for more details. \(https://adafru.it/AAH\)](https://adafru.it/AAH)

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

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

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

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

Code done running.

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

Interacting with the Serial Console

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

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

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

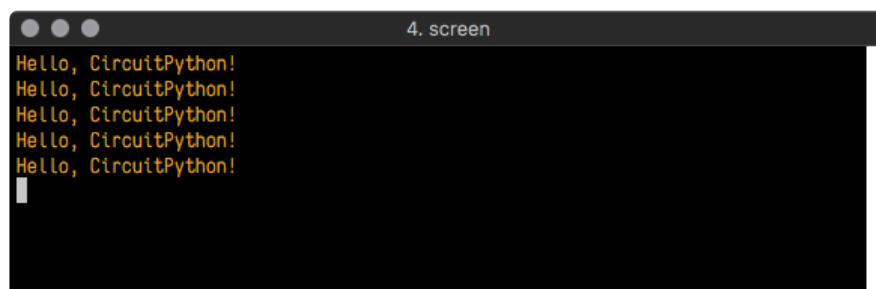
```
import board
import digitalio
import time

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

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

Save your file.

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



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

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

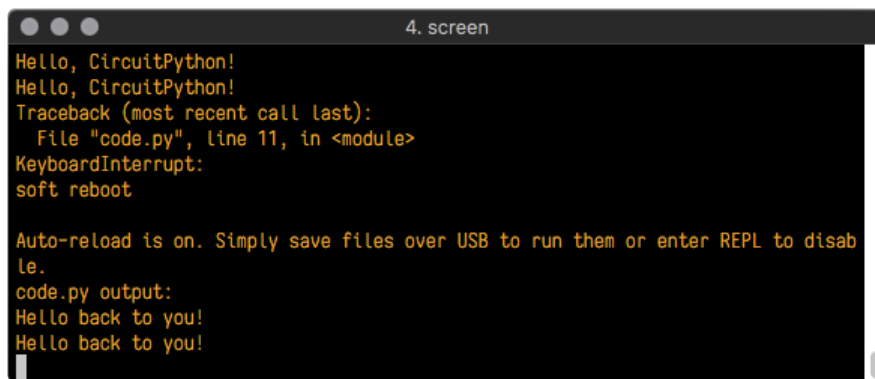
```
import board
import digitalio
```

```
import time

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

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

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



```
Hello, CircuitPython!
Hello, CircuitPython!
Traceback (most recent call last):
  File "code.py", line 11, in <module>
KeyboardInterrupt:
soft reboot

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

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

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

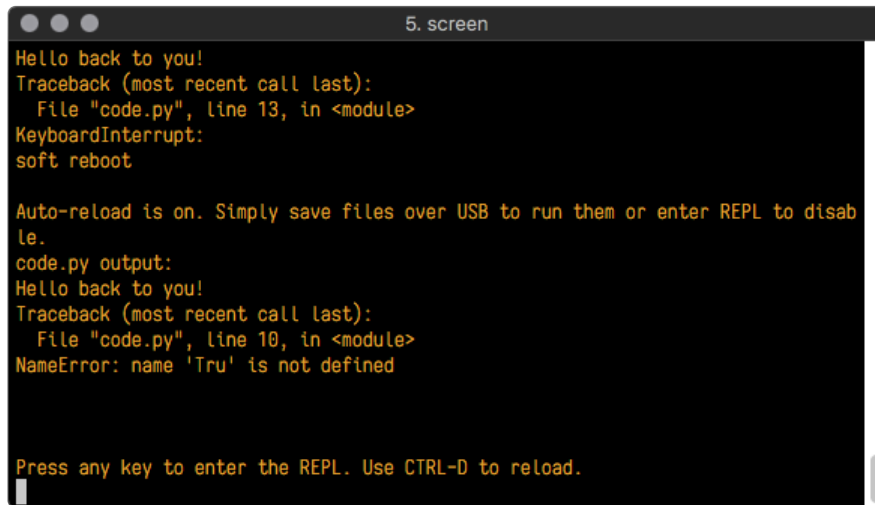
```
import board
import digitalio
import time

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

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

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

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



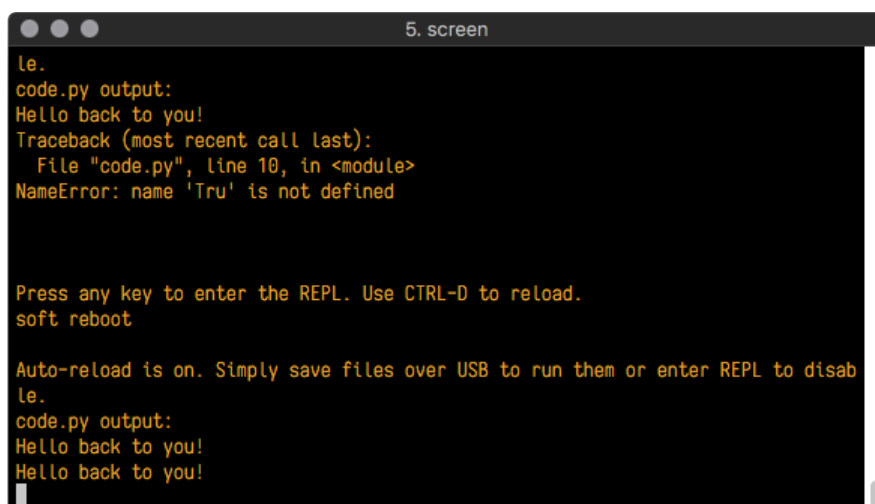
```
5. screen
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 13, in <module>
KeyboardInterrupt:
soft reboot

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

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

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

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



```
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, which is called "print debugging". Essentially, if your code isn't working, and you want to know where it's failing, you can put print statements in various places to see where it stops printing.

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

The REPL

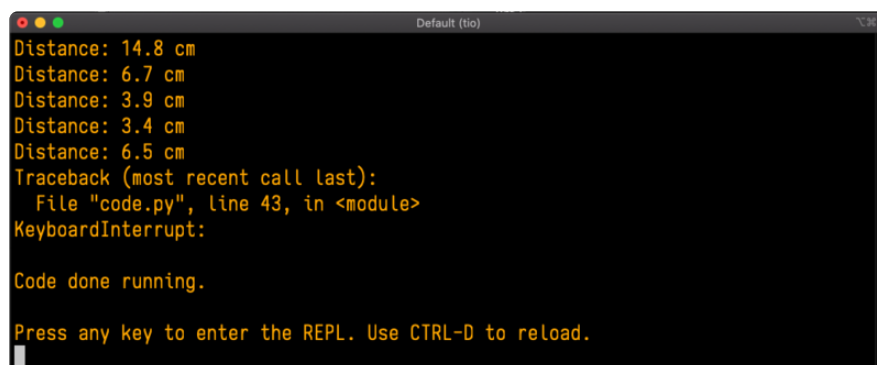
The other feature of the serial connection is the **Read-Evaluate-Print-Loop**, or REPL. The REPL allows you to enter individual lines of code and have them run immediately. It's really handy if you're running into trouble with a particular program and can't figure out why. It's interactive so it's great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

Entering the REPL

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

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

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



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

Code done running.

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

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

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

Code done running.

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

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

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

Code done running.

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

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

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

If you have trouble getting to the `>>>` prompt, try pressing Ctrl + C a few more times. The first thing you get from the REPL is information about your board.

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

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

This is followed by the CircuitPython prompt.

```
>>>
```

Interacting with the REPL

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

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


```

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

```

Then press enter. You should then see a message.

```

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

Visit circuitpython.org for more information.

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

```

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

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

```

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

```

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

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

```

>>> import board
>>>

```

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

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

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

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

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

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

Then press enter.

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

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

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

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

Try typing more into the REPL to see what happens!

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

Returning to the Serial Console

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

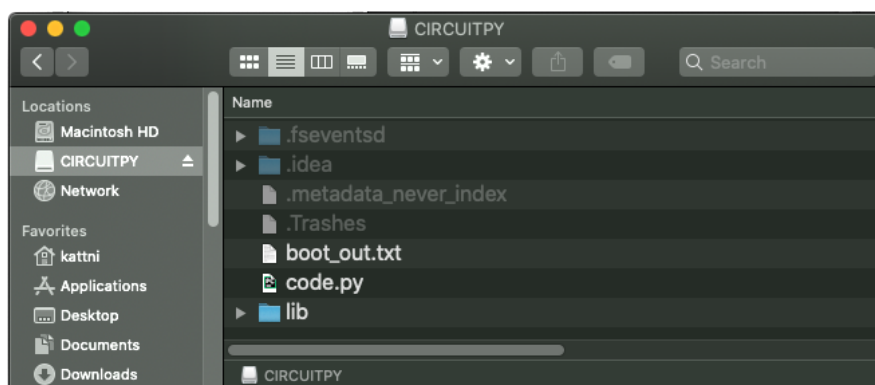
A terminal window titled "Default (tio)" with a black background and yellow text. The text inside the terminal reads: "Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.", "Code done running.", and "Press any key to enter the REPL. Use CTRL-D to reload."

CircuitPython Libraries

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

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

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



CircuitPython libraries work in the same way as regular Python modules so the [Python docs \(https://adafru.it/rar\)](https://adafru.it/rar) are an excellent reference for how it all should work. In

Python terms, you can place our library files in the **lib** directory because it's part of the Python path by default.

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

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

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

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

The Adafruit Learn Guide Project Bundle

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

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

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

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



Piano in the Key of Lime

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

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

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

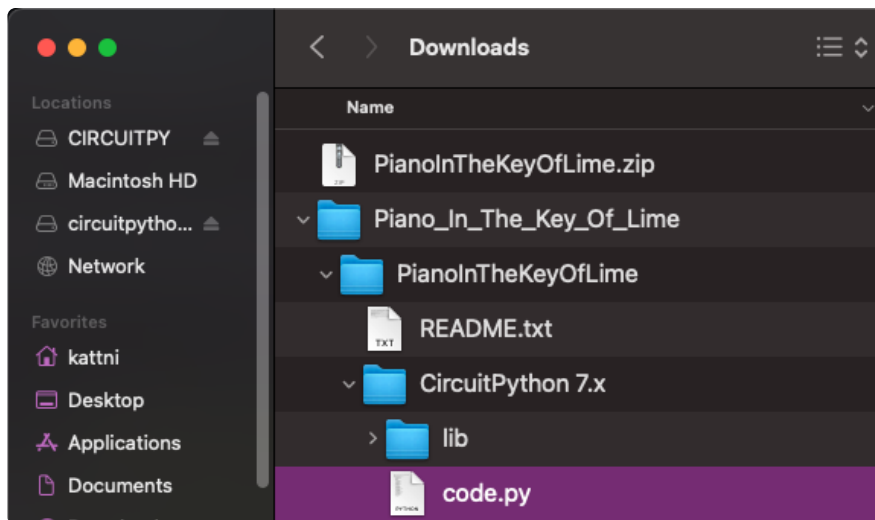
```
# SPDX-FileCopyrightText: 2017 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

from adafruit_circuitplayground import cp

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

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

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



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

When you open the zip, you'll find some nested directories. Navigate through them until you find what you need. You'll eventually find a directory for your CircuitPython version (in this case, 7.x). In the version directory, you'll find the file and directory you need: **code.py** and **lib/**. Once you find the content you need, you can copy it all over to your **CIRCUITPY** drive, replacing any files already on the drive with the files from the freshly downloaded zip.

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

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

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

The Adafruit CircuitPython Library Bundle

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

Downloading the Adafruit CircuitPython Library Bundle

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

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

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

<https://adafru.it/ENC>

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

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

The CircuitPython Community Library Bundle

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

These libraries are maintained by their authors and are not supported by Adafruit.

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

Downloading the CircuitPython Community Library Bundle

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

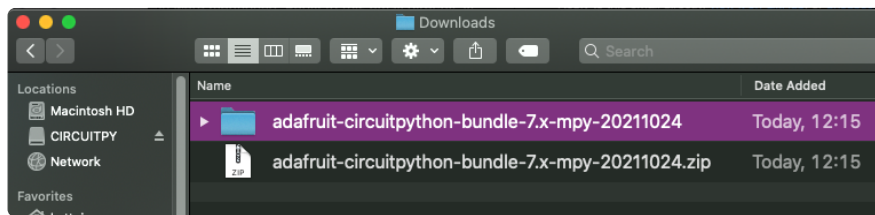
Click for the latest CircuitPython
Community Library Bundle release

<https://adafru.it/VCn>

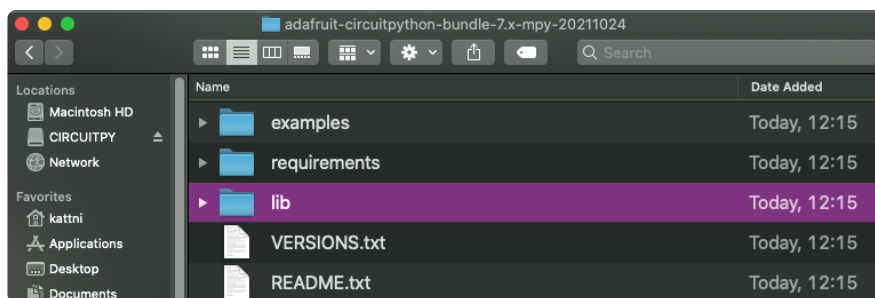
The link takes you to the latest release of the CircuitPython Community Library Bundle on GitHub. There are multiple versions of the bundle available. **Download the bundle version that matches your CircuitPython firmware version.** If you don't know the version, check the version info in `boot_out.txt`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

Understanding the Bundle

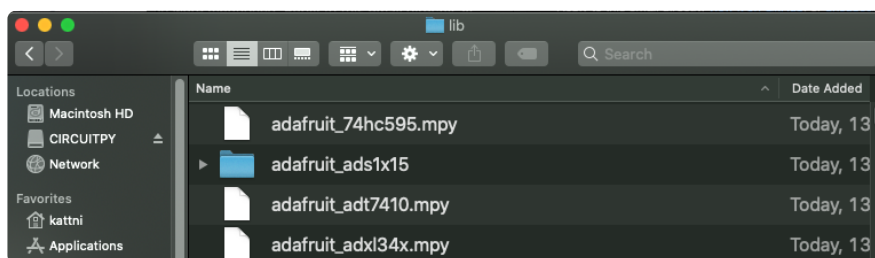
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



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



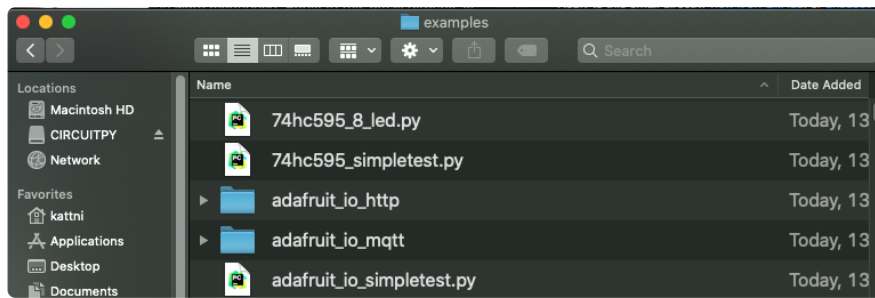
Now open the lib folder. When you open the folder, you'll see a large number of .mpy files, and folders.



Example Files

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

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



Copying Libraries to Your Board

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

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

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

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

Understanding Which Libraries to Install

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

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

- **import library_or_module**

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

- **from library_or_module import name**
- **from library_or_module.subpackage import name**

- `from library_or_module import name as local_name`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

There are cases, however, where libraries require other libraries internally. The internally required library is called a dependency. In the event of library

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

Example: `ImportError` Due to Missing Library

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

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

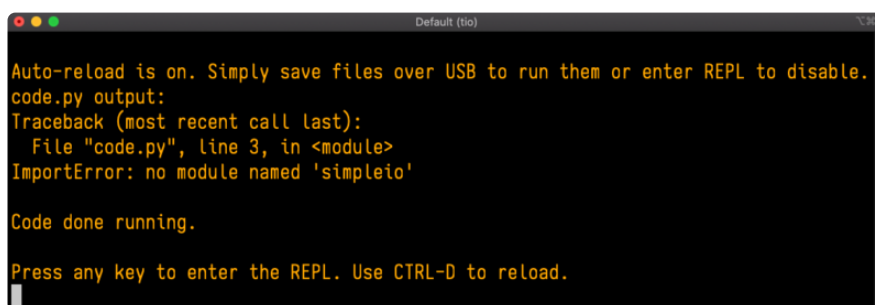
Let's use a modified version of the Blink example.

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.LED)

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

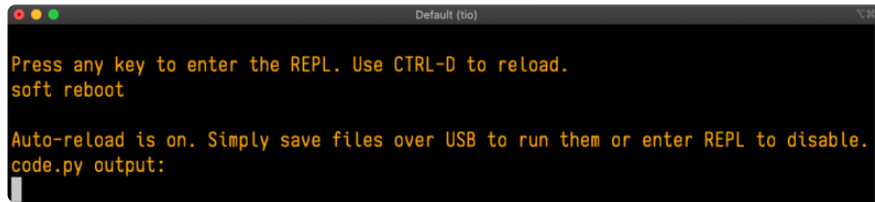
Save this file. Nothing happens to your board. Let's check the serial console to see what's going on.

A screenshot of a serial console window titled "Default (tio)". The text inside the window is as follows: "Auto-reload is on. Simply save files over USB to run them or enter REPL to disable." followed by "code.py output:" and a "Traceback (most recent call last):" section. The traceback shows "File 'code.py', line 3, in <module>" and "ImportError: no module named 'simpleio'". Below the traceback, it says "Code done running." and "Press any key to enter the REPL. Use CTRL-D to reload." The window has a dark background with light-colored text.

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

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

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



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

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

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

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

Library Install on Non-Express Boards

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

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

Updating CircuitPython Libraries and Examples

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

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

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

CircUp CLI Tool

There is a command line interface (CLI) utility called [CircUp \(https://adafru.it/Tfi\)](https://adafru.it/Tfi) that can be used to easily install and update libraries on your device. Follow the directions on the [install page within the CircUp learn guide \(https://adafru.it/-Ad\)](https://adafru.it/-Ad). Once you've got it installed you run the command `circup update` in a terminal to interactively

update all libraries on the connected CircuitPython device. See the [usage page in the CircUp guide](https://adafru.it/-Ah) (<https://adafru.it/-Ah>) for a full list of functionality

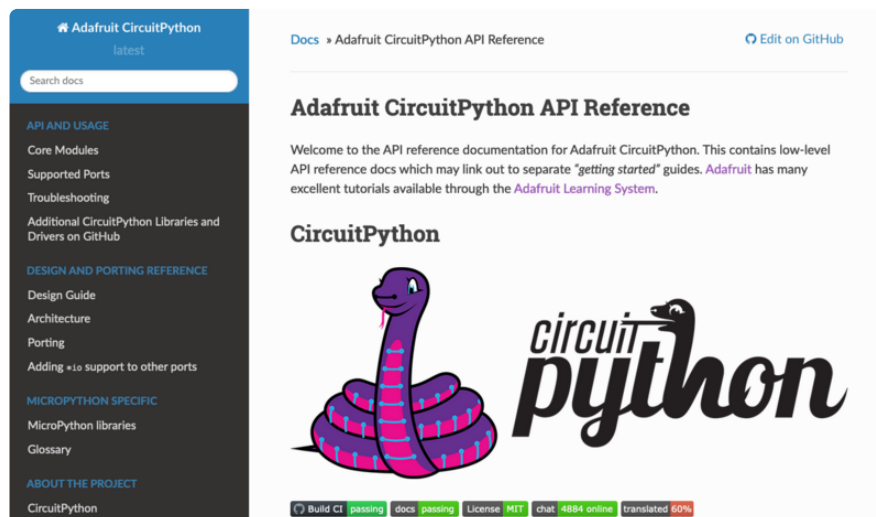
CircuitPython Documentation

You've learned about the CircuitPython built-in modules and external libraries. You know that you can find the modules in CircuitPython, and the libraries in the Library Bundles. There are guides available that explain the basics of many of the modules and libraries. However, there's sometimes more capabilities than are necessarily showcased in the guides, and often more to learn about a module or library. So, where can you find more detailed information? That's when you want to look at the API documentation.

The entire CircuitPython project comes with extensive documentation available on Read the Docs. This includes both the [CircuitPython core](https://adafru.it/Beg) (<https://adafru.it/Beg>) and the [Adafruit CircuitPython libraries](https://adafru.it/Tra) (<https://adafru.it/Tra>).

CircuitPython Core Documentation

The [CircuitPython core documentation](https://adafru.it/Beg) (<https://adafru.it/Beg>) covers many of the details you might want to know about the CircuitPython core and related topics. It includes API and usage info, a design guide and information about porting CircuitPython to new boards, MicroPython info with relation to CircuitPython, and general information about the project.



The main page covers the basics including where to **download CircuitPython**, how to **contribute**, **differences from MicroPython**, information about the **project structure**, and a **full table of contents** for the rest of the documentation.

The list along the left side leads to more information about specific topics.

The first section is **API and Usage**. This is where you can find information about how to use individual built-in **core modules**, such as `time` and `digitalio`, details about the **supported ports**, suggestions for **troubleshooting**, and basic info and links to the **library bundles**. The **Core Modules** section also includes the **Support Matrix**, which is a table of which core modules are available on which boards.

The second section is **Design and Porting Reference**. It includes a **design guide**, **architecture** information, details on **porting**, and **adding module support** to other ports.

The third section is **MicroPython Specific**. It includes information on **MicroPython and related libraries**, and a **glossary** of terms.

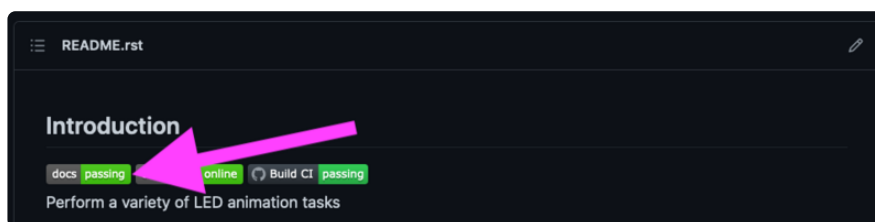
The fourth and final section is **About the Project**. It includes further information including details on **building, testing, and debugging CircuitPython**, along with various other useful links including the **Adafruit Community Code of Conduct**.

Whether you're a seasoned pro or new to electronics and programming, you'll find a wealth of information to help you along your CircuitPython journey in the documentation!

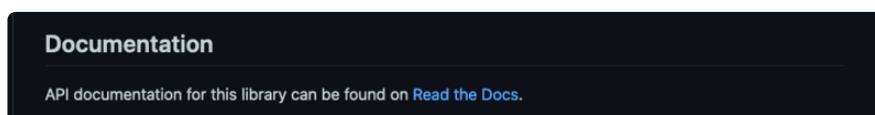
CircuitPython Library Documentation

The Adafruit CircuitPython libraries are documented in a very similar fashion. Each library has its own page on Read the Docs. There is a comprehensive list available [here](https://adafru.it/Tra) (<https://adafru.it/Tra>). Otherwise, to view the documentation for a specific library, you can visit the GitHub repository for the library, and find the link in the README.

For the purposes of this page, the [LED Animation library](https://adafru.it/O2d) (<https://adafru.it/O2d>) documentation will be featured. There are two links to the documentation in each library GitHub repo. The first one is the **docs badge** near the top of the README.



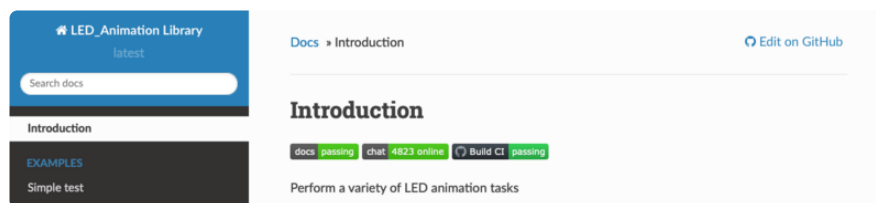
The second place is the **Documentation** section of the README. Scroll down to find it, and click on Read the Docs to get to the documentation.



Now that you know how to find it, it's time to take a look at what to expect.

Not all library documentation will look exactly the same, but this will give you some idea of what to expect from library docs.

The **Introduction** page is generated from the README, so it includes all the same info, such as PyPI installation instructions, a quick demo, and some build details. It also includes a full table of contents for the rest of the documentation (which is not part of the GitHub README). The page should look something like the following.



The left side contains links to the rest of the documentation, divided into three separate sections: **Examples**, **API Reference**, and **Other Links**.

Examples

The [Examples section \(https://adafru.it/VFD\)](https://adafru.it/VFD) is a list of library examples. This list contains anywhere from a small selection to the full list of the examples available for the library.

This section will always contain at least one example - the **simple test** example.



The simple test example is usually a basic example designed to show your setup is working. It may require other libraries to run. Keep in mind, it's simple - it won't showcase a comprehensive use of all the library features.

The LED Animation simple test demonstrates the Blink animation.

Simple test

Ensure your device works with this simple test.

`examples/led_animation_simpletest.py`

```
1 # SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
2 # SPDX-License-Identifier: MIT
3
4 """
5 This simpletest example displays the Blink animation.
6
7 For NeoPixel FeatherWing. Update pixel_pin and pixel_num to match your wiring if using
8 a different form of NeoPixels.
9 """
10 import board
11 import neopixel
12 from adafruit_led_animation.animation.blink import Blink
13 from adafruit_led_animation.color import RED
14
15 # Update to match the pin connected to your NeoPixels
16 pixel_pin = board.D6
17 # Update to match the number of NeoPixels you have connected
18 pixel_num = 32
19
20 pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)
21
22 blink = Blink(pixels, speed=0.5, color=RED)
23
24 while True:
25     blink.animate()
```

In some cases, you'll find a longer list, that may include examples that explore other features in the library. The LED Animation documentation includes a series of examples, all of which are available in the library. These examples include demonstrations of both basic and more complex features. Simply click on the example that interests you to view the associated code.

EXAMPLES

Simple test

Basic Animations

All Animations

Pixel Map

Animation Sequence

Animation Group

Blink

Basic Animations

Demonstrates the basic animations.

`examples/led_animation_basic_animations.py`

```
1 # SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
2 # SPDX-License-Identifier: MIT
3
4 """
5 This example displays the basic animations in sequence, at a five second interval.
6 """
```

When there are multiple links in the Examples section, all of the example content is, in actuality, on the same page. Each link after the first is an anchor link to the specified section of the page. Therefore, you can also view all the available examples by scrolling down the page.

You can view the rest of the examples by clicking through the list or scrolling down the page. These examples are fully working code. Which is to say, while they may rely on other libraries as well as the library for which you are viewing the documentation, they should not require modification to otherwise work.

API Reference

The [API Reference section \(https://adafru.it/Rqa\)](https://adafru.it/Rqa) includes a list of the library functions and classes. The API (Application Programming Interface) of a library is the set of functions and classes the library provides. Essentially, the API defines how your program interfaces with the functions and classes that you call in your code to use the library.

There is always at least one list item included. Libraries for which the code is included in a single Python (.py) file, will only have one item. Libraries for which the code is multiple Python files in a directory (called subpackages) will have multiple items in this list. The LED Animation library has a series of subpackages, and therefore, multiple items in this list.

Click on the first item in the list to begin viewing the API Reference section.



As with the Examples section, all of the API Reference content is on a single page, and the links under API Reference are anchor links to the specified section of the page.

When you click on an item in the API Reference section, you'll find details about the classes and functions in the library. In the case of only one item in this section, all the available functionality of the library will be contained within that first and only subsection. However, in the case of a library that has subpackages, each item will contain the features of the particular subpackage indicated by the link. The documentation will cover all of the available functions of the library, including more complex ones that may not interest you.

The first list item is the animation subpackage. If you scroll down, you'll begin to see the available features of animation. They are listed alphabetically. Each of these things can be called in your code. It includes the name and a description of the specific function you would call, and if any parameters are necessary, lists those with a description as well.


```
class adafruit_led_animation.Animation(pixel_object, speed, color, peers=None, paused=False, name=None)
```

Base class for animations.

```
add_cycle_complete_receiver(callback)
```

Adds an additional callback when the cycle completes.

Parameters

callback – Additional callback to trigger when a cycle completes. The callback is passed the animation object instance.

```
after_draw()
```

Animation subclasses may implement `after_draw()` to do operations after the main `draw()` is called.

You can view the other subpackages by clicking the link on the left or scrolling down the page. You may be interested in something a little more practical. Here is an example. To use the LED Animation library Comet animation, you would run the following example.

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

"""
This example animates a jade comet that bounces from end to end of the strip.

For QT Py Haxpress and a NeoPixel strip. Update pixel_pin and pixel_num to match
your wiring if
using a different board or form of NeoPixels.

This example will run on SAMD21 (M0) Express boards (such as Circuit Playground
Express or QT Py
Haxpress), but not on SAMD21 non-Express boards (such as QT Py or Trinket).
"""
import board
import neopixel

from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.color import JADE

# Update to match the pin connected to your NeoPixels
pixel_pin = board.A3
# Update to match the number of NeoPixels you have connected
pixel_num = 30

pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)
comet = Comet(pixels, speed=0.02, color=JADE, tail_length=10, bounce=True)

while True:
    comet.animate()
```

Note the line where you create the `comet` object. There are a number of items inside the parentheses. In this case, you're provided with a fully working example. But what if you want to change how the comet works? The code alone does not explain what the options mean.

So, in the API Reference documentation list, click the `adafruit_led_animation.animation.comet` link and scroll down a bit until you see the following.

```
class adafruit_led_animation.animation.comet.Comet(pixel_object, speed, color, tail_length=0, reverse=False, bounce=False, name=None, ring=False)
```

A comet animation.

Parameters

- pixel_object** – The initialised LED object.
- speed** (*float*) – Animation speed in seconds, e.g. `0.1`.
- color** – Animation color in `(r, g, b)` tuple, or `0x000000` hex format.
- tail_length** (*int*) – The length of the comet. Defaults to 25% of the length of the `pixel_object`. Automatically compensates for a minimum of 2 and a maximum of the length of the `pixel_object`.
- reverse** (*bool*) – Animates the comet in the reverse order. Defaults to `False`.
- bounce** (*bool*) – Comet will bounce back and forth. Defaults to `True`.
- ring** (*bool*) – Ring mode. Defaults to `False`.

Look familiar? It is! This is the documentation for setting up the comet object. It explains what each argument provided in the comet setup in the code meant, as well as the other available features. For example, the code includes `speed=0.02`. The documentation clarifies that this is the "Animation speed in seconds". The code doesn't include `ring`. The documentation indicates this is an available setting that enables "Ring mode".

This type of information is available for any function you would set up in your code. If you need clarification on something, wonder whether there's more options available, or are simply interested in the details involved in the code you're writing, check out the documentation for the CircuitPython libraries!

Other Links

This section is the same for every library. It includes a list of links to external sites, which you can visit for more information about the CircuitPython Project and Adafruit.

That covers the CircuitPython library documentation! When you are ready to go beyond the basic library features covered in a guide, or you're interested in understanding those features better, the library documentation on Read the Docs has you covered!

Recommended Editors

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.

To avoid the likelihood of filesystem corruption, use an editor that writes out the file completely when you save it. Check out the list of recommended editors below.

Recommended editors

- [mu](https://adafru.it/ANO) (<https://adafru.it/ANO>) 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
- [Notepad++](https://adafru.it/xNf) (<https://adafru.it/xNf>) flushes files after writes, as of several years ago. In addition, you can change the path used for "Enable session snapshot and periodic backup" to write somewhere else than the CIRCUITPY drive. This will save space on CIRCUITPY and reduce writes to the drive.

Recommended only with particular settings or 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>) or the [language-circuitpython package](https://adafru.it/Vuf) (<https://adafru.it/Vuf>) so that it will always write out all changes to files on CIRCUITPY.

- **SlickEdit** (<https://adafru.it/DdP>) works only if you [add a macro to flush the disk](https://adafru.it/ven) (<https://adafru.it/ven>).

The editors listed below are specifically NOT recommended!

Editors that are NOT recommended

- **notepad** (the default Windows editor) can be slow to write, so the editors above are recommended! If you are using notepad, be sure to eject the drive.
- **IDLE** in Python 3.8.0 or earlier does not force out changes immediately. Later versions do force out changes.
- **nano** (on Linux) does not force out changes.
- **geany** (on Linux) does not force out changes.
- **Anything else** - Other editors have not been tested so please use a recommended one!

Advanced Serial Console on Windows Windows 7 and 8.1

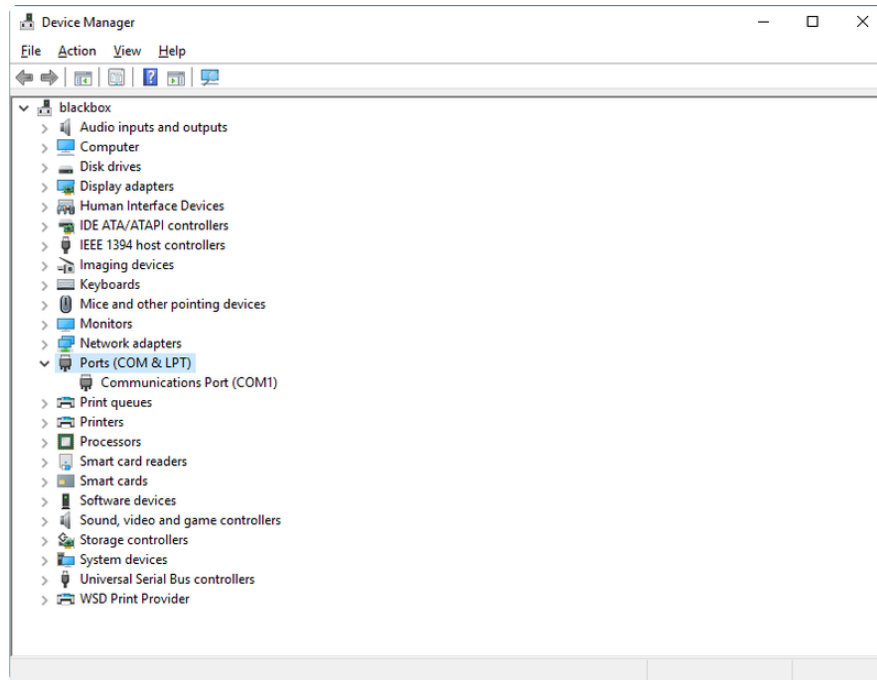
If you're using Windows 7 (or 8 or 8.1), you'll need to install drivers. See the [Windows 7 and 8.1 Drivers page](https://adafru.it/VuB) (<https://adafru.it/VuB>) for details. You will not need to install drivers on Mac, Linux or Windows 10.

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

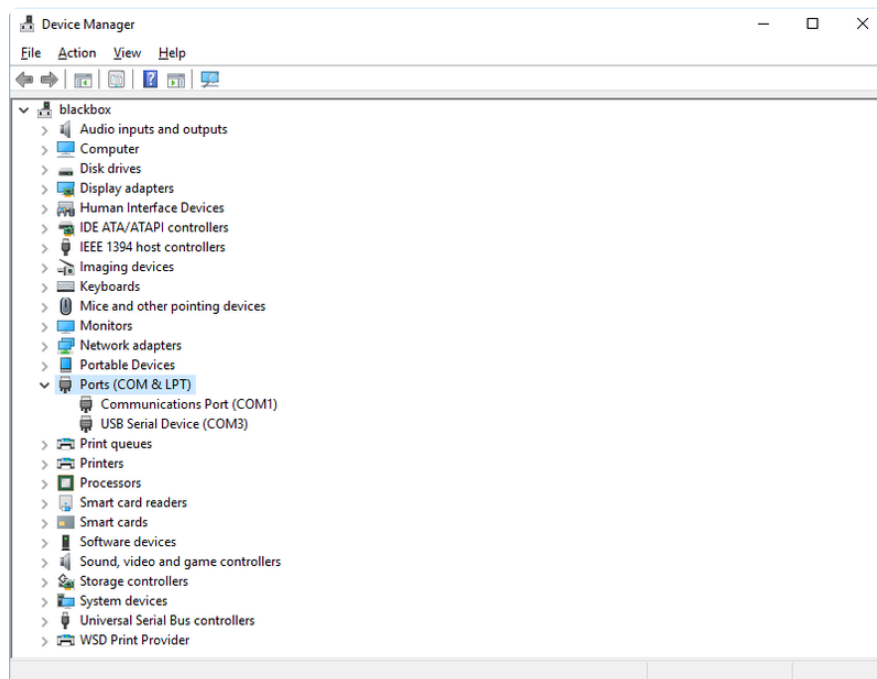
What's the COM?

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

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



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



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

Install Putty

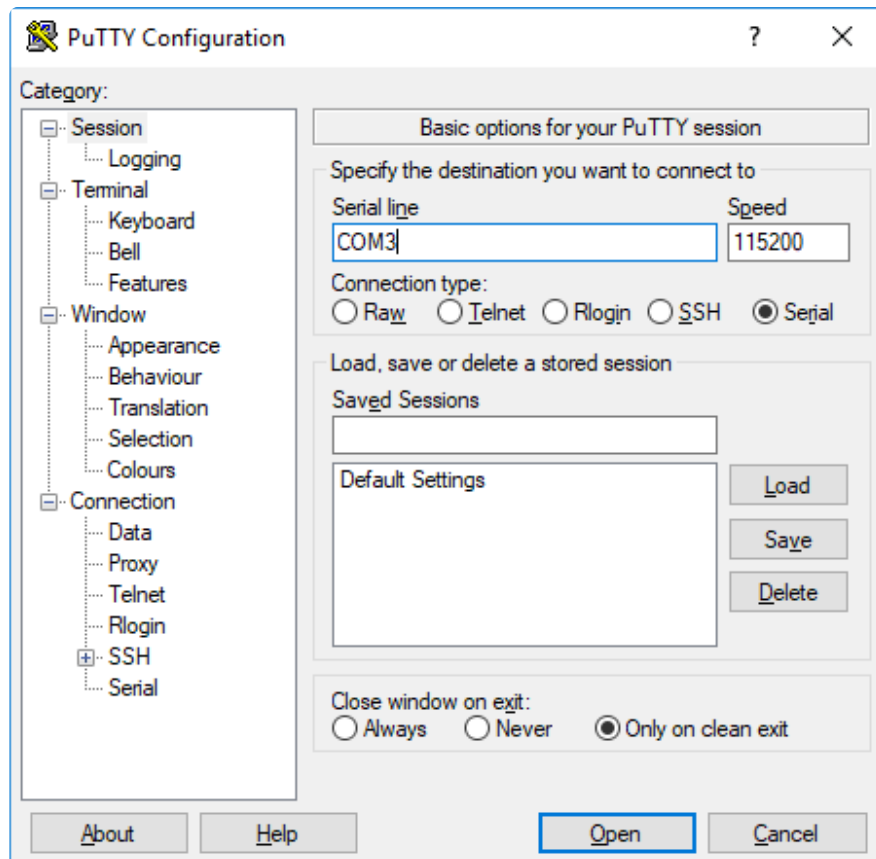
If you're using Windows, you'll need to download a terminal program. You're going to use PuTTY.

The first thing to do is download the [latest version of PuTTY \(https://adafru.it/Bf1\)](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.

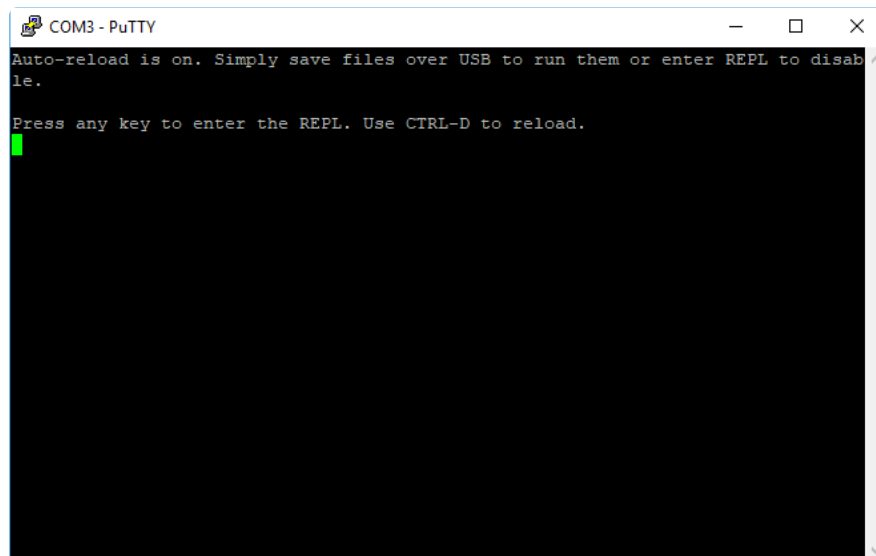
Now you need to open PuTTY.

- 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!

Advanced Serial Console on Mac

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

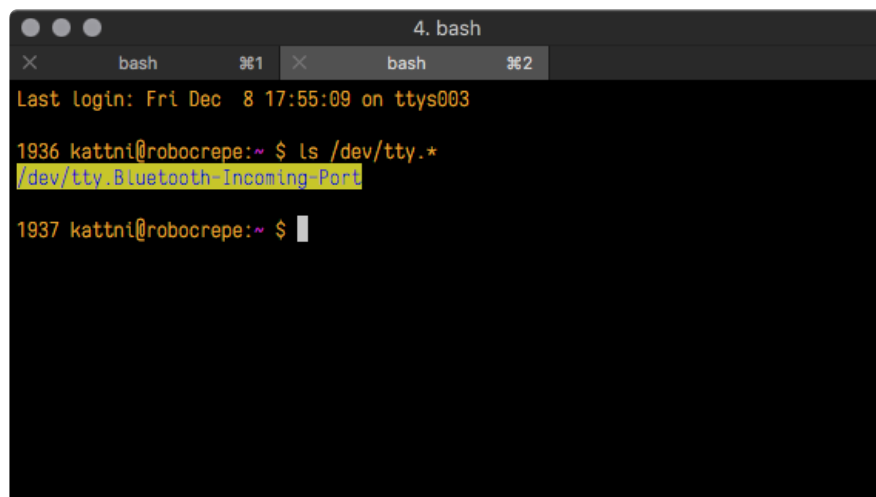
What's the Port?

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

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

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

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

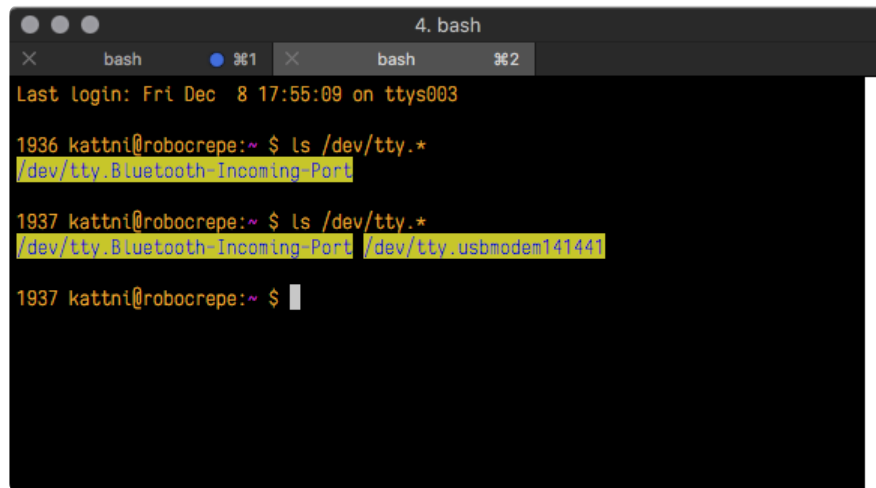
A screenshot of a macOS Terminal window titled "4. bash". It shows two tabs: "bash" and "bash". The first tab is active and displays the following text: "Last login: Fri Dec 8 17:55:09 on ttys003", "1936 kattni@robocrepe:~ \$ ls /dev/tty.*", and the output "/dev/tty.Blueetooth-Incoming-Port". The second tab is inactive and shows "1937 kattni@robocrepe:~ \$".

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

Now, plug your board. In Terminal, type:

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

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

A screenshot of a macOS Terminal window titled '4. bash'. It shows two tabs, both labeled 'bash'. The first tab shows the output of 'ls /dev/tty.*' as '/dev/tty.Bluetooth-Incoming-Port'. The second tab shows the output of the same command as '/dev/tty.Bluetooth-Incoming-Port' and '/dev/tty.usbmodem141441'. The prompt is 'kattni@robocrepe:~\$'.

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

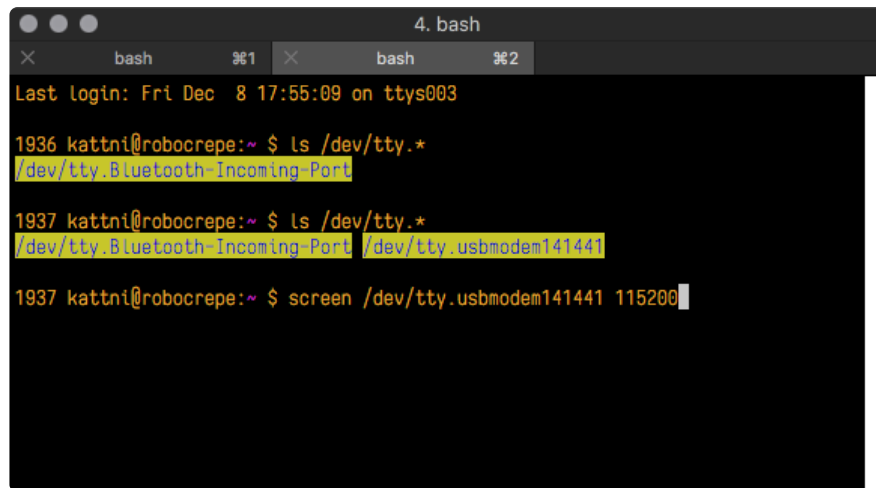
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

Connect with screen

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

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

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



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

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

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

Advanced Serial Console on Linux

Connecting to the serial console on Linux does not require installing any drivers, but you may need to install `screen` using your package manager. You'll use a terminal program to find your board, and `screen` to connect to it. There are a variety of terminal programs such as `gnome-terminal` (called Terminal) or Konsole on KDE.

The `tio` program works as well to connect to your board, and has the benefit of automatically reconnecting. You would need to install it using your package manager.

What's the Port?

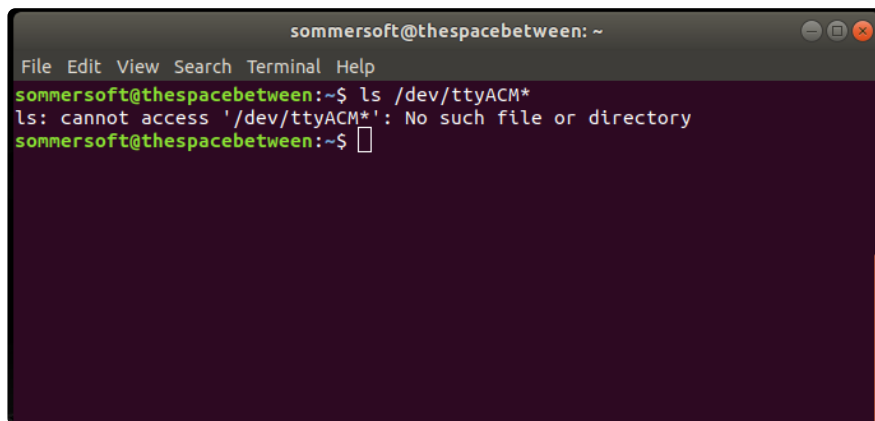
First you'll want to find out which serial port your board is using. When you plug your board in to USB on your computer, it connects to a serial port. The port is like a door through which your board can communicate with your computer using USB.

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

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

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

In the example below, the error is indicating that there are no current serial connections starting with **ttyACM**.

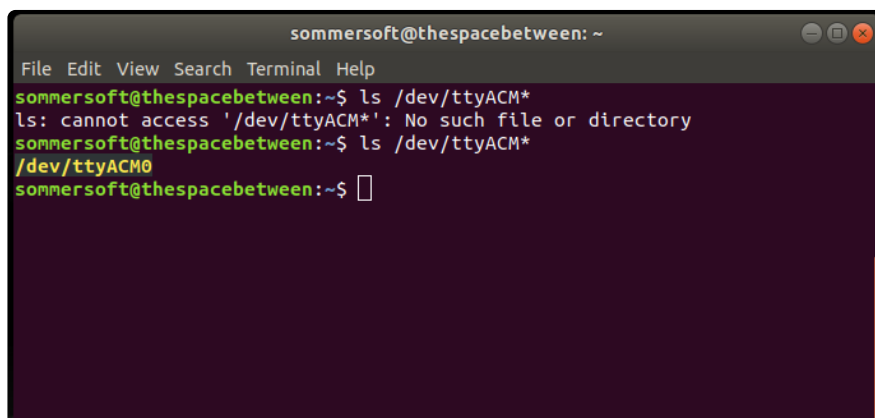
A terminal window titled 'sommersoft@thespacebetween: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'ls /dev/ttyACM*' being executed, which results in the error message 'ls: cannot access '/dev/ttyACM*': No such file or directory'. The prompt 'sommersoft@thespacebetween:~\$' is shown again.

```
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 in your board. In your terminal program, type:

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

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

A terminal window titled 'sommersoft@thespacebetween: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'ls /dev/ttyACM*' being executed, which results in the error message 'ls: cannot access '/dev/ttyACM*': No such file or directory'. The command is then repeated, and the output shows '/dev/ttyACM0'. The prompt 'sommersoft@thespacebetween:~\$' is shown again.

```
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:~$
```

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

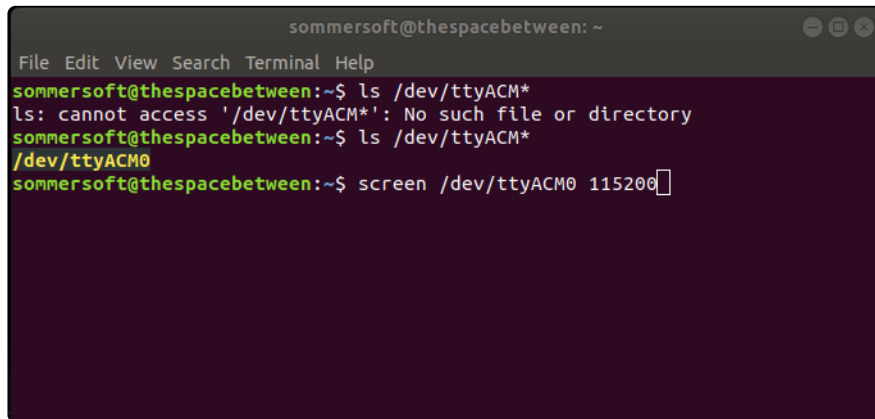
Now that you know the name your board is using, you're ready to connect to the serial console. You'll use a command called **screen**. You may need to install it using the package manager.

To connect to the serial console, use your terminal program. 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.



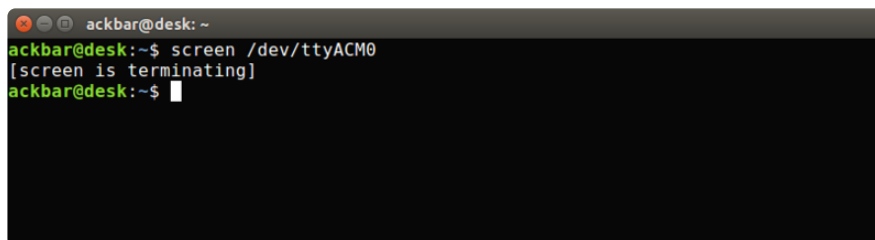
```
sommersoft@thespacebetween: ~  
File Edit View Search Terminal Help  
sommersoft@thespacebetween:~$ ls /dev/ttyACM*  
ls: cannot access '/dev/ttyACM*': No such file or directory  
sommersoft@thespacebetween:~$ ls /dev/ttyACM*  
/dev/ttyACM0  
sommersoft@thespacebetween:~$ screen /dev/ttyACM0 115200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

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!

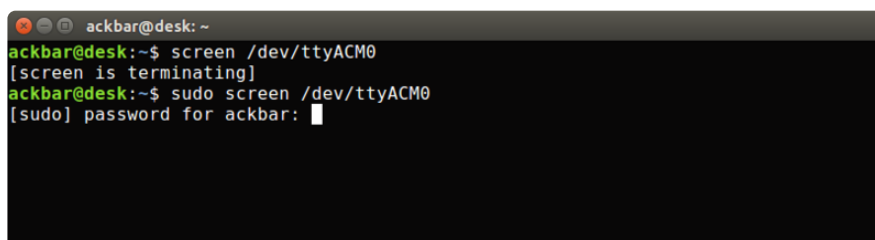
Permissions on Linux

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.

? What are some common acronyms to know?

CP or CPy = [CircuitPython \(https://adafru.it/KJD\)](https://adafru.it/KJD)

CPC = [Circuit Playground Classic \(http://adafru.it/3000\)](http://adafru.it/3000) (does not run CircuitPython)

CPX = [Circuit Playground Express \(http://adafru.it/3333\)](http://adafru.it/3333)

CPB = [Circuit Playground Bluefruit \(http://adafru.it/4333\)](http://adafru.it/4333)

Using Older Versions

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

? I have to continue using CircuitPython 8.x or earlier. Where can I find compatible libraries?

Python Arithmetic



Does CircuitPython support floating-point numbers?

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

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



Does CircuitPython support long integers, like regular Python?

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

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

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

Wireless Connectivity



How do I connect to the Internet with CircuitPython?

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

If your development board has an SPI port and at least 4 additional pins, you can check out [this guide \(https://adafru.it/F5X\)](https://adafru.it/F5X) on using AirLift with CircuitPython - extra wiring is required and some boards like the MacroPad or NeoTrellis do not have enough available pins to add the hardware support.

For further project examples, and guides about using AirLift with specific hardware, check out [the Adafruit Learn System \(https://adafru.it/VBr\)](https://adafru.it/VBr).



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

nRF52840, nRF52833, and as of **CircuitPython 9.1.0**, ESP32, ESP32-C3, and ESP32-S3 boards (with 8MB) have the most complete BLE implementation. Your program can act as both a BLE central and peripheral. As a central, you can scan for advertisements, and connect to an advertising board. As a peripheral, you can advertise, and you can create services available to a central. Pairing and bonding are supported.

Most Espressif boards with only 4MB of flash do not have enough room to include BLE in CircuitPython 9. Check the [Module Support Matrix \(https://adafru.it/-Cy\)](https://adafru.it/-Cy) to see if your board has support for `_bleio`. CircuitPython 10 is planned to support `_bleio` on Espressif boards with 4MB flash.

Note that the ESP32-S2 does not have Bluetooth capability.

On most other boards with adequate firmware space, [BLE is available for use with AirLift \(https://adafru.it/11Av\)](https://adafru.it/11Av) or other NINA-FW-based co-processors. Some boards have this coprocessor on board, such as the [PyPortal \(https://adafru.it/11Aw\)](https://adafru.it/11Aw). Currently, this implementation only supports acting as a BLE peripheral. Scanning and connecting as a central are not yet implemented. Bonding and pairing are not supported.



Are there other ways to communicate by radio with CircuitPython?

Check out [Adafruit's RFM boards \(https://adafru.it/11Ay\)](https://adafru.it/11Ay) for simple radio communication supported by CircuitPython, which can be used over distances of 100m to over a km, depending on the version. The RFM SAMD21 M0 boards can be used, but they were not designed for CircuitPython, and have limited RAM and flash space; using the RFM breakouts or FeatherWings with more capable boards will be easier.

Asyncio and Interrupts

Is there asyncio support in CircuitPython?

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

Does CircuitPython support interrupts?

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

Status RGB LED

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

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

Memory Issues



What is a MemoryError?

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



What do I do when I encounter a MemoryError?

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

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

On macOS and Linux, after you download `mpy-cross`, you must make the the file executable by doing `chmod +x name-of-the-mpy-cross-executable`.

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



How do I check how much memory I have free?

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

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

Unsupported Hardware



Is ESP8266 or ESP32 supported in CircuitPython? Why not?

We dropped ESP8266 support as of 4.x - For more information please read about it [here \(https://adafru.it/CiG\)](https://adafru.it/CiG)!

As of CircuitPython 8.x we have started to support ESP32 and ESP32-C3 and have added a WiFi workflow for wireless coding! (<https://adafru.it/10JF>)

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



Does Feather M0 support WINC1500?

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



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

No.

Troubleshooting

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

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

Always Run the Latest Version of CircuitPython and Libraries

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. **You need to [update to the latest CircuitPython](https://adafru.it/Em8)**. (<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)** (<https://adafru.it/ENC>).

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

I have to continue using CircuitPython 7.x or earlier.
Where can I find compatible libraries?

Adafruit is no longer building or supporting the CircuitPython 7.x or earlier library bundles. You are highly encouraged to [update CircuitPython to the latest version](https://adafru.it/Em8) (<https://adafru.it/Em8>) and use [the current version of the libraries](https://adafru.it/ENC) (<https://adafru.it/ENC>). However, if for some reason you cannot update, links to the previous bundles are available in the [FAQ](https://adafru.it/FwY) (<https://adafru.it/FwY>).

macOS Sonoma before 14.4: Errors Writing to CIRCUITPY macOS 14.4 - 15.1: Slow Writes to CIRCUITPY

macOS Sonoma before 14.4 took many seconds to complete writes to small FAT drives, 8MB or smaller. This causes errors when writing to CIRCUITPY. The best solution was to remount the CIRCUITPY drive after it is automatically mounted. Or consider downgrading back to Ventura if that works for you. This problem was tracked in [CircuitPython GitHub issue 8449](https://adafru.it/18ea) (<https://adafru.it/18ea>).

Below is a shell script to do this remount conveniently (courtesy [@czei in GitHub](https://adafru.it/18ea) (<https://adafru.it/18ea>)). Copy the code here into a file named, say, **remount-CIRCUITPY.sh**. Place the file in a directory on your PATH, or in some other convenient place.

macOS Sonoma 14.4 and versions of macOS before Sequoia 15.2 did not have the problem above, but did take an inordinately long time to write to FAT drives of size 1GB or less (40 times longer than 2GB drives). As of macOS 15.2, writes are no longer very slow. This problem was tracked in [CircuitPython GitHub issue 8918 \(https://adafru.it/19iD\)](https://adafru.it/19iD).

```
#!/bin/sh
#
# This works around bug where, by default,
# macOS 14.x before 14.4 writes part of a file immediately,
# and then doesn't update the directory for 20-60 seconds, causing
# the file system to be corrupted.
#

disky=`df | grep CIRCUITPY | cut -d" " -f1`
sudo umount /Volumes/CIRCUITPY
sudo mkdir /Volumes/CIRCUITPY
sleep 2
sudo mount -v -o noasync -t msdos $disky /Volumes/CIRCUITPY
```

Then in a Terminal window, do this to make this script executable:

```
chmod +x remount-CIRCUITPY.sh
```

Place the file in a directory on your **PATH**, or in some other convenient place.

Now, each time you plug in or reset your CIRCUITPY board, run the file **remount-CIRCUITPY.sh**. You can run it in a Terminal window or you may be able to place it on the desktop or in your dock to run it just by double-clicking.

This will be something of a nuisance but it is the safest solution.

This problem is being tracked in [this CircuitPython issue \(https://adafru.it/18ea\)](https://adafru.it/18ea).

Bootloader (boardnameBOOT) Drive Not Present

You may have a different board.

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

MakeCode

If you are running a [MakeCode \(https://adafru.it/zbY\)](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 or later

Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 or later with the driver package installed? You don't need to install this package on Windows 10 or 11 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

Windows 7 and 8.1 have reached end of life. It is [recommended \(https://adafru.it/Amd\)](https://adafru.it/Amd) that you upgrade to Windows 10 or 11 if possible. Drivers are available for some older CircuitPython boards, but there are no plans to release drivers for newer boards.

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 available. There are no plans to release drivers for newer boards. The boards work fine on Windows 10 and later.

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

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

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

Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied

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

CIRCUITPY Drive Does Not Appear or Disappears Quickly

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

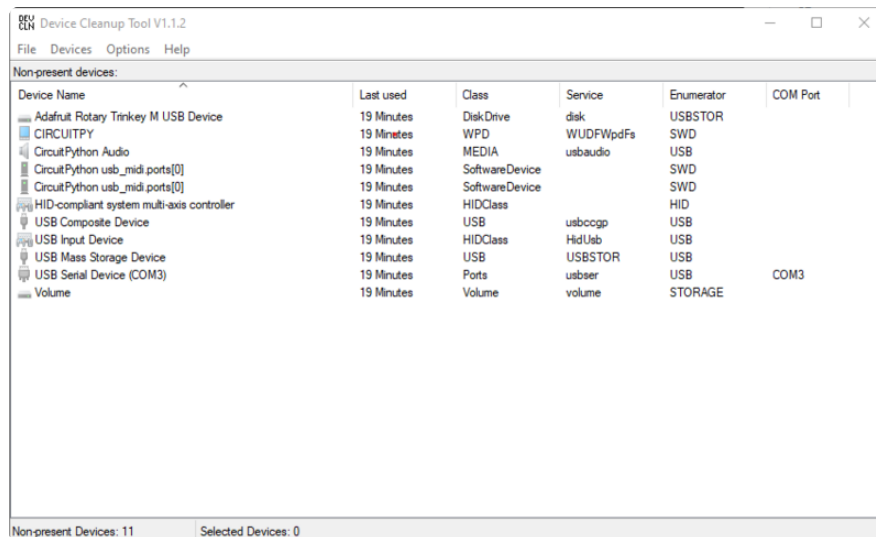
Norton anti-virus can interfere with **CIRCUITPY**. A user has reported this problem on Windows 7. The user turned off both Smart Firewall and Auto Protect, and **CIRCUITPY** then appeared.

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

Samsung Magician can cause CIRCUITPY to disappear (reported [here \(https://adafru.it/18eb\)](https://adafru.it/18eb) and [here \(https://adafru.it/18ec\)](https://adafru.it/18ec)).

Device Errors or Problems on Windows

Windows can become confused about USB device installations. Try cleaning up your USB devices. Use [Uwe Sieber's Device Cleanup Tool \(https://adafru.it/RWd\)](https://adafru.it/RWd) (on that page, scroll down to "Device Cleanup Tool"). Download and unzip the tool. Unplug all the boards and other USB devices you want to clean up. Run the tool as Administrator. You will see a listing like this, probably with many more devices. It is listing all the USB devices that are not currently attached.



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

Serial Console in Mu Not Displaying Anything

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

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

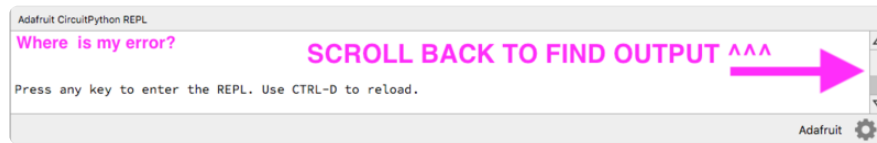
```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Traceback (most recent call last):
  File "code.py", line 7
SyntaxError: invalid syntax
```

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

More complex errors take even more lines!

Therefore, if your serial console panel is five lines tall or less, you may only see blank lines or blank lines followed by **Press any key to enter the REPL. Use CTRL-D**

to reload. . If this is the case, you need to either mouse over the top of the panel to utilise the option to resize the serial panel, or use the scrollbar on the right side to scroll up and find your message.



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

code.py Restarts Constantly

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

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

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

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

```
import supervisor
supervisor.runtime.autoreload = False
```

CircuitPython RGB Status Light

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

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

CircuitPython 7.0.0 and Later

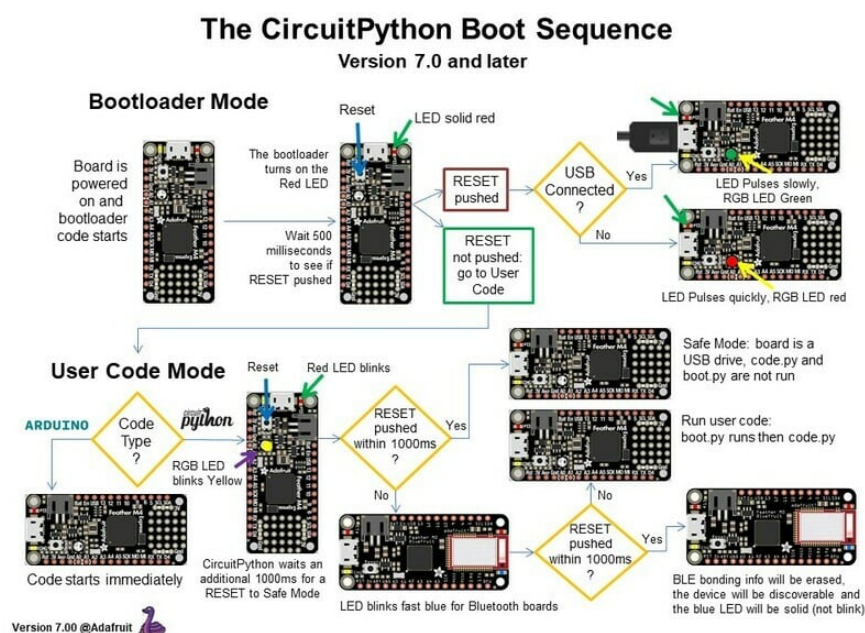
The status LED blinks were changed in CircuitPython 7.0.0 in order to save battery power and simplify the blinks. These blink patterns will occur on single color LEDs when the board does not have any RGB LEDs. Speed and blink count also vary for this reason.

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

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

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

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



CircuitPython 6.3.0 and earlier

Here's what the colors and blinking mean:

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

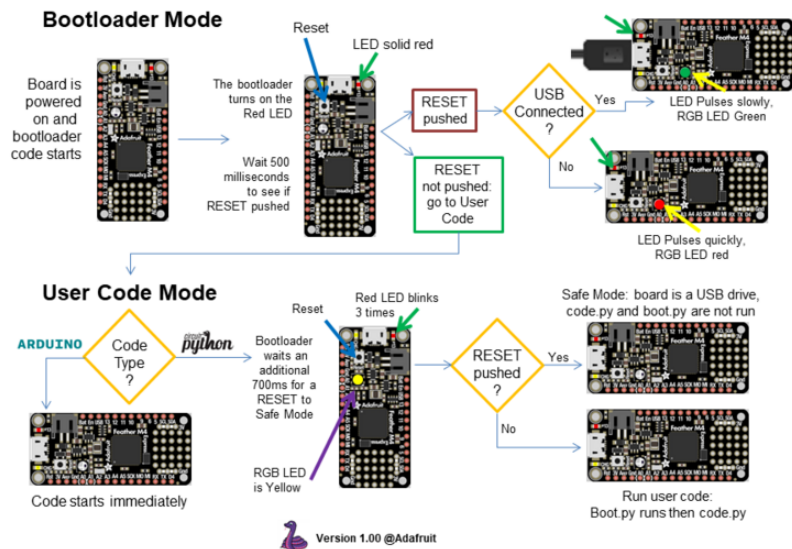
Colors with multiple flashes following indicate a Python exception and then indicate the line number of the error. The color of the first flash indicates the type of error:

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

These are followed by flashes indicating the line number, including place value.

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

The CircuitPython Boot Sequence



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

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

So, for instance, if you upgraded to CircuitPython 7.x from 6.x you'll need to download a newer version of the library that triggered the error on **import**. All libraries are available in the [Adafruit bundle \(https://adafru.it/y8E\)](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. When the **CIRCUITPY** disk is not safely ejected before being reset by the button or being disconnected from USB, it may corrupt the flash drive. It can happen on Windows, Mac or Linux, though it is more common on Windows.

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

The easiest first step is to reload CircuitPython. Double-tap reset on the board so you get a **boardnameBOOT** drive rather than a **CIRCUITPY** drive, and copy the latest

version of CircuitPython (.uf2) back to the board. This may restore **CIRCUITPY** functionality.

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

Safe Mode

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

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

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

Entering Safe Mode in CircuitPython 7.x and Later

You can enter safe by pressing reset during the right time when the board boots. Immediately after the board starts up or resets, it waits one second. On some boards, the onboard status LED will blink yellow during that time. If you press reset during that one second period, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a "slow" double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

Entering Safe Mode in CircuitPython 6.x

You can enter safe by pressing reset during the right time when the board boots.. Immediately after the board starts up or resets, it waits 0.7 seconds. On some boards, the onboard status LED (highlighted in green above) will turn solid yellow during this time. If you press reset during that 0.7 seconds, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

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

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

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

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

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

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

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

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

To erase CIRCUITPY: `storage.erase_filesystem()`

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

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

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

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

Erase CIRCUITPY Without Access to the REPL

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

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

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

For the specific boards listed below:

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

1. Download the correct erase file:

Circuit Playground Express

<https://adafru.it/AdI>

Feather M0 Express

<https://adafru.it/AdJ>

Feather M4 Express

<https://adafru.it/EVK>

Metro M0 Express

<https://adafru.it/AdK>

Metro M4 Express QSPI Eraser

<https://adafru.it/EoM>

Trellis M4 Express (QSPI)

<https://adafru.it/DjD>

Grand Central M4 Express (QSPI)

<https://adafru.it/DBA>

PyPortal M4 Express (QSPI)

<https://adafru.it/Eca>

Circuit Playground Bluefruit (QSPI)

<https://adafru.it/Gnc>

Monster M4SK (QSPI)

<https://adafru.it/GAN>

PyBadge/PyGamer QSPI Eraser.UF2

<https://adafru.it/GAO>

CLUE_Flash_Erase.UF2

<https://adafru.it/Jat>

Matrix_Portal_M4_(QSPI).UF2

<https://adafru.it/Q5B>

RP2040 boards (flash_nuke.uf2)

<https://adafru.it/18ed>

2. Double-click the reset button on the board to bring up the **boardnameBOOT** drive.

3. Drag the erase **.uf2** file to the **boardnameBOOT** drive.

4. The status LED will turn yellow or blue, indicating the erase has started.

5. After approximately 15 seconds, the status LED will light up green. On the NeoTrellis M4 this is the first NeoPixel on the grid

6. Double-click the reset button on the board to bring up the **boardnameBOOT** drive.

7. [Drag the appropriate latest release of CircuitPython \(https://adafru.it/Em8\)](https://adafru.it/Em8) **.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>). You'll also need to load your code and reinstall your libraries!

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

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

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

1. Download the erase file:

SAMD21 non-Express Boards

<https://adafru.it/VB->

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 \(<https://adafru.it/Em8>\)](https://adafru.it/Em8) **.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>) You'll also need to load your code and reinstall your libraries!

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

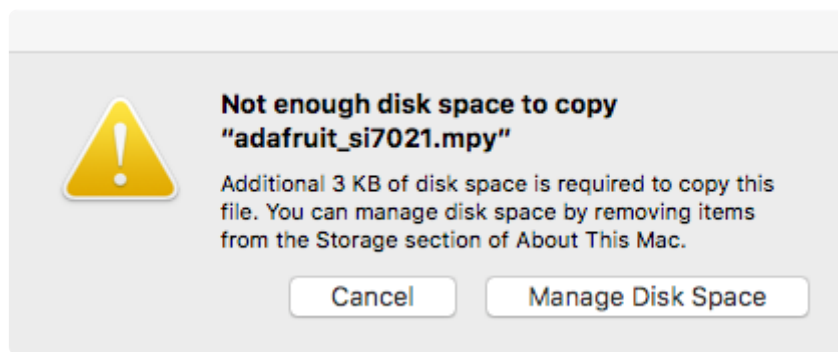
Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that do **not** have a UF2 bootloader include the Feather M0 Basic Proto, Feather Adalogger, or the Arduino Zero.

If you are trying to erase a non-Express board that does not have a UF2 bootloader, [follow these directions to reload CircuitPython using `bossac`](https://adafru.it/Bed) (<https://adafru.it/Bed>), which will erase and re-create **CIRCUITPY**.

Running Out of File Space on SAMD21 Non-Express Boards

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

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



Delete something!

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

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

Use tabs

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

On macOS?

MacOS loves to generate hidden files. Luckily you can disable some of the extra hidden files that macOS adds by running a few commands to disable search indexing

and create zero byte placeholders. Follow the steps below to maximize the amount of space available on macOS.

Prevent & Remove macOS Hidden Files

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

```
ls -l /Volumes
```

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

Now follow the [steps from this question \(https://adafru.it/u1c\)](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 .{,_.}{fseventsd,Spotlight-V*,Trashes}
mkdir .fseventsd
touch .fseventsd/no_log .metadata_never_index .Trashes
cd -
```

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

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

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

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

Copy Files on macOS Without Creating Hidden Files

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

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

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

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

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

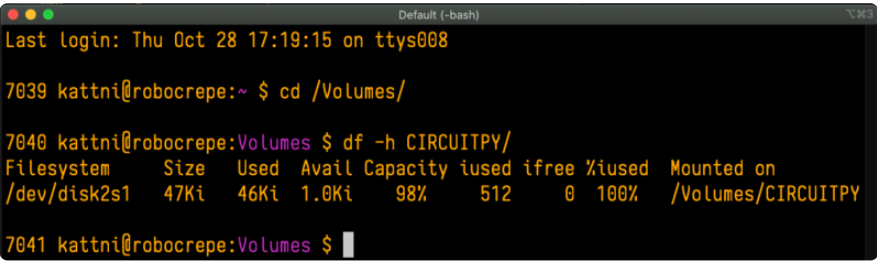
```
cp -rX folder_to_copy /Volumes/CIRCUITPY
```

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

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

Other macOS Space-Saving Tips

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



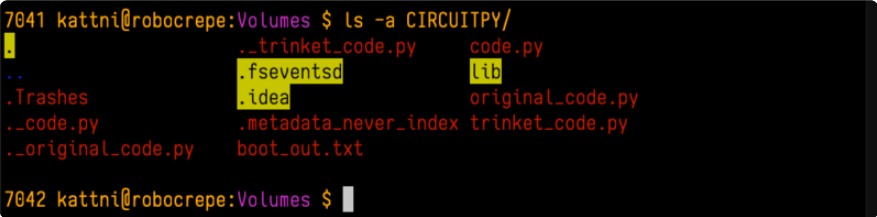
```
Default (-bash)
Last login: Thu Oct 28 17:19:15 on ttys008

7039 kattni@robocrepe:~ $ cd /Volumes/

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

7041 kattni@robocrepe:Volumes $
```

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



```
7041 kattni@robocrepe:Volumes $ ls -a CIRCUITPY/
.      .trinket_code.py  code.py
..     .fsevents         lib
.Trashes  .idea             original_code.py
._code.py .metadata_never_index trinket_code.py
._original_code.py boot_out.txt

7042 kattni@robocrepe:Volumes $
```

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

```
7042 kattni@robocrepe:Volumes $ rm CIRCUITPY/._*
7043 kattni@robocrepe:Volumes $
```

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

```
7043 kattni@robocrepe:Volumes $ df -h CIRCUITPY/
Filesystem      Size  Used Avail Capacity iused ifree %used  Mounted on
/dev/disk2s1    47Ki   34Ki   13Ki    73%     512     0  100%   /Volumes/CIRCUITPY
7044 kattni@robocrepe:Volumes $
```

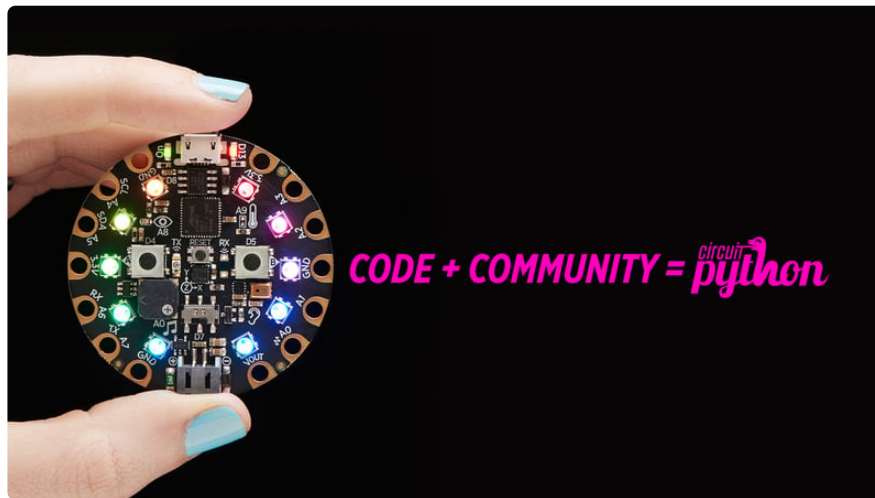
Nice! You have 12Ki more than before! This space can now be used for libraries and code!

Device Locked Up or Boot Looping

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

For more information on safe mode and how to enter safe mode, see the [Safe Mode section on this page \(https://adafru.it/Den\)](https://adafru.it/Den).

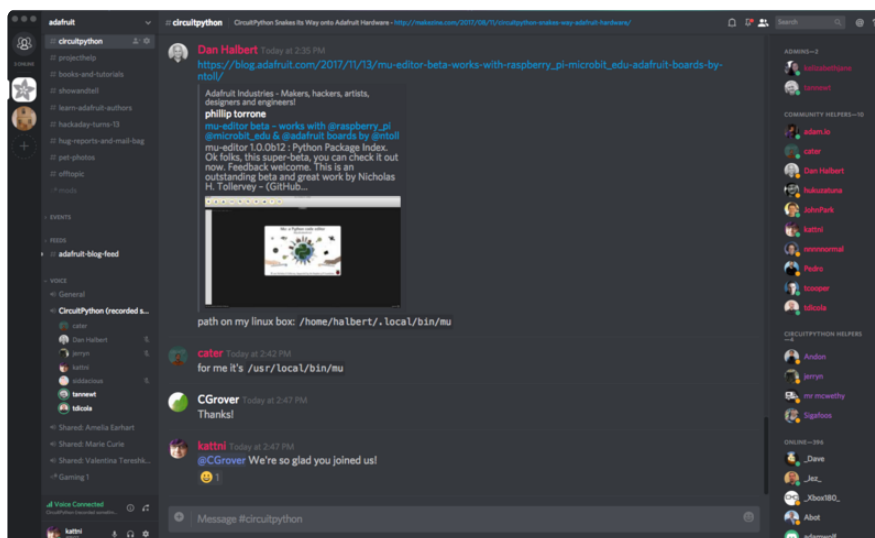
Welcome to the Community!



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

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

Adafruit Discord



The Adafruit Discord server is the best place to start. Discord is where the community comes together to volunteer and provide live support of all kinds. From general

discussion to detailed problem solving, and everything in between, Discord is a digital maker space with makers from around the world.

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

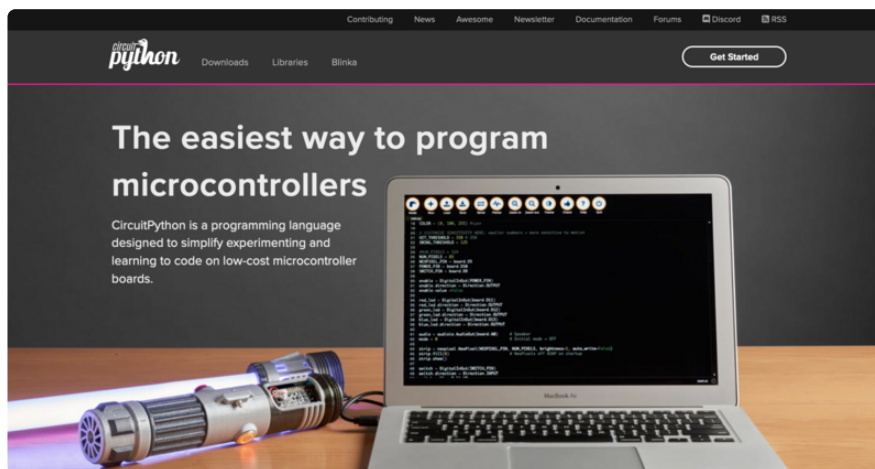
The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. Your contributions are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

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

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

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

CircuitPython.org



Beyond the Adafruit Learn System, which you are viewing right now, the best place to find information about CircuitPython is circuitpython.org (<https://adafru.it/KJD>). Everything you need to get started with your new microcontroller and beyond is

available. You can do things like [download CircuitPython for your microcontroller \(https://adafru.it/Em8\)](https://adafru.it/Em8) or [download the latest CircuitPython Library bundle \(https://adafru.it/ENC\)](https://adafru.it/ENC), or check out [which single board computers support Blinka \(https://adafru.it/EA8\)](https://adafru.it/EA8). You can also get to various other CircuitPython related things like Awesome CircuitPython or the Python for Microcontrollers newsletter. This is all incredibly useful, but it isn't necessarily community related. So why is it included here? The [Contributing page \(https://adafru.it/VD7\)](https://adafru.it/VD7).

Contributing

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

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

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

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

CircuitPython itself is written in C. However, all of the Adafruit CircuitPython libraries are written in Python. If you're interested in contributing to CircuitPython on the Python side of things, check out [circuitpython.org/contributing \(https://adafru.it/VD7\)](https://adafru.it/VD7). You'll find information pertaining to every Adafruit CircuitPython library GitHub repository, giving you the opportunity to join the community by finding a contributing option that works for you.

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

Current Status for Tue, Nov 02, 2021

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

Now, a look at the different options.

Pull Requests

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



Pull Requests Open Issues Library Infrastructure Issues CircuitPython Localization

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

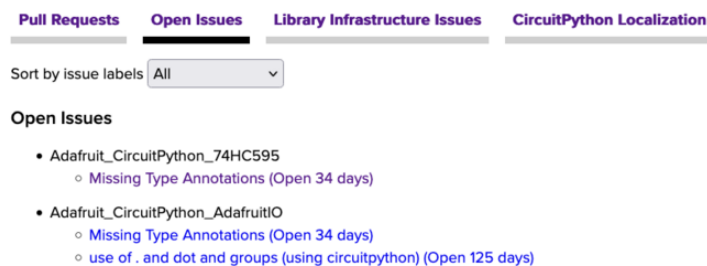
Open Pull Requests

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

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

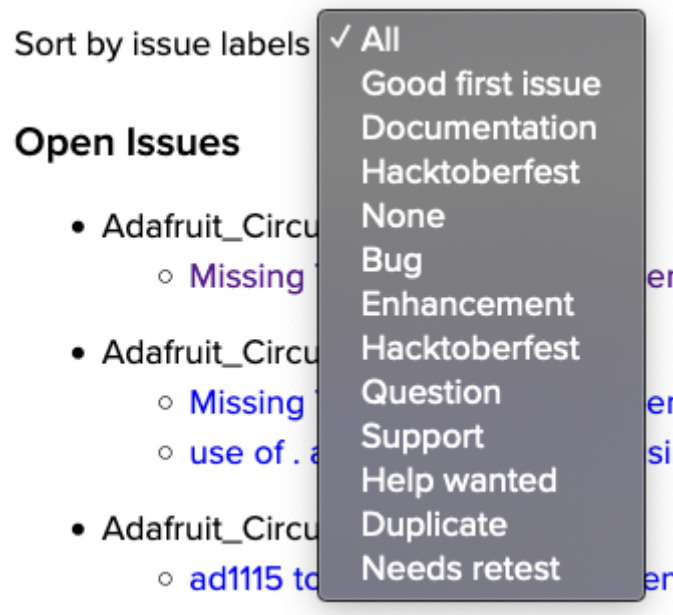
Open Issues

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



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

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



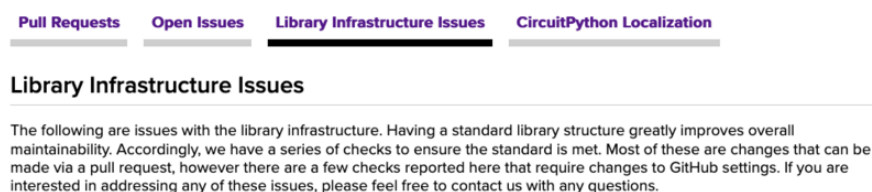
If you're new to everything, new to contributing to open source, or new to contributing to the CircuitPython project, you can choose "Good first issue". Issues with that label are well defined, with a finite scope, and are intended to be easy for someone new to figure out.

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

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

Library Infrastructure Issues

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

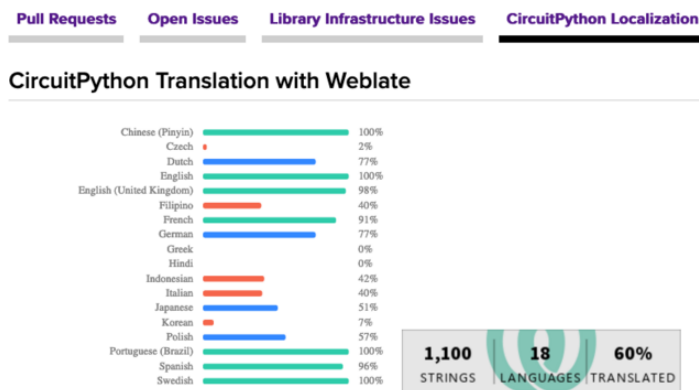


This section is generated by a script that runs checks on the libraries, and then reports back where there may be issues. It is made up of a list of subsections each containing links to the repositories that are experiencing that particular issue. This page is available mostly for internal use, but you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

to resolve that issue. Others can reply either way to let you know what the scope of it might be, and help you resolve it if necessary.

CircuitPython Localization

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

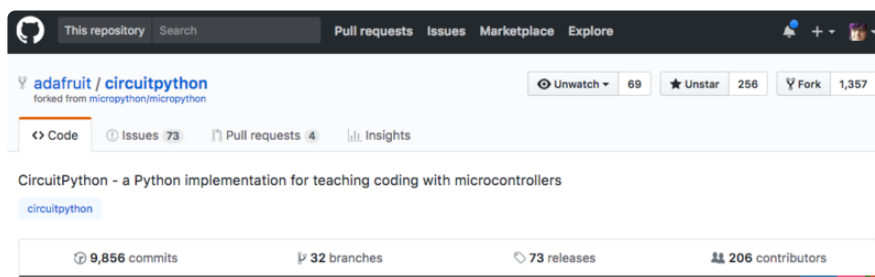


If you speak another language, you can help translate CircuitPython! The translations apply to informational and error messages that are within the CircuitPython core. It means that folks who do not speak English have the opportunity to have these messages shown to them in their own language when using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

CircuitPython uses Weblate to translate, which makes it much simpler to contribute translations. You will still need to know some CircuitPython-specific practices and a few basics about coding strings, but as with any CircuitPython contributions, folks are there to help.

Regardless of your skill level, or how you want to contribute to the CircuitPython project, there is an opportunity available. The [Contributing page \(https://adafru.it/VD7\)](https://adafru.it/VD7) is an excellent place to start!

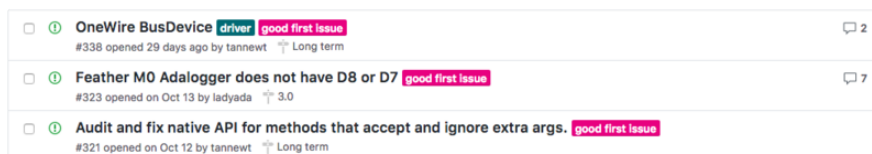
Adafruit GitHub



Whether you're just beginning or are life-long programmer who would like to contribute, there are ways for everyone to be a part of the CircuitPython project. The CircuitPython core is written in C. The libraries are written in Python. GitHub is the best source of ways to contribute to the [CircuitPython core \(https://adafru.it/tB7\)](https://adafru.it/tB7), and

the [CircuitPython libraries \(https://adafru.it/VFv\)](https://adafru.it/VFv). 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. For the CircuitPython core, head over to the CircuitPython repository on GitHub, click on "[Issues \(https://adafru.it/tBb\)](https://adafru.it/tBb)", and you'll find a list that includes issues labeled "[good first issue \(https://adafru.it/188e\)](https://adafru.it/188e)". For the libraries, head over to the [Contributing page Issues list \(https://adafru.it/VFv\)](https://adafru.it/VFv), and use the drop down menu to search for "[good first issue \(https://adafru.it/VFw\)](https://adafru.it/VFw)". These issues are things that have been identified as something that someone with any level of experience can help with. These issues include options like updating documentation, providing feedback, and fixing simple bugs. If you need help getting started with GitHub, there is an excellent guide on [Contributing to CircuitPython with Git and GitHub \(https://adafru.it/Dkh\)](https://adafru.it/Dkh).



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

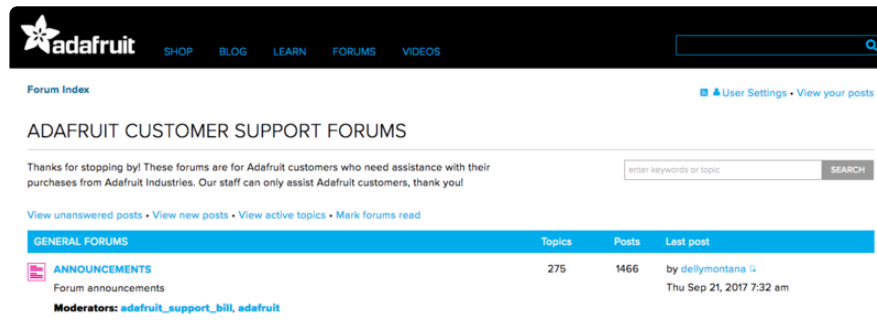
Already experienced and looking for a challenge? Checkout the rest of either issues list and you'll find plenty of ways to contribute. You'll find all sorts of things, from new driver requests, to library bugs, to core module updates. There's plenty of opportunities for everyone at any level!

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

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

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

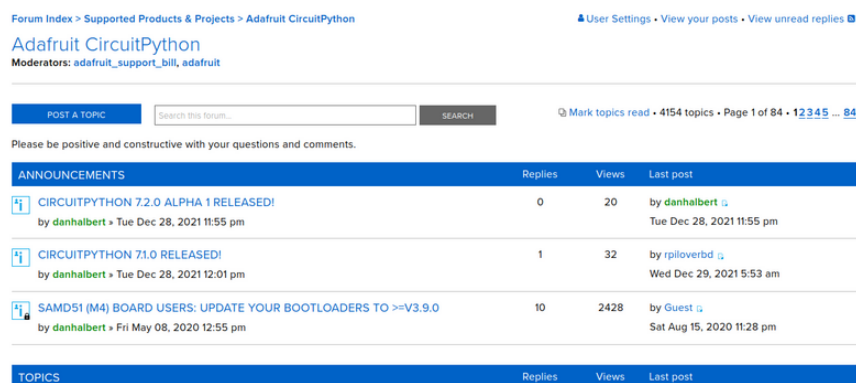
Adafruit Forums



The [Adafruit Forums](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.

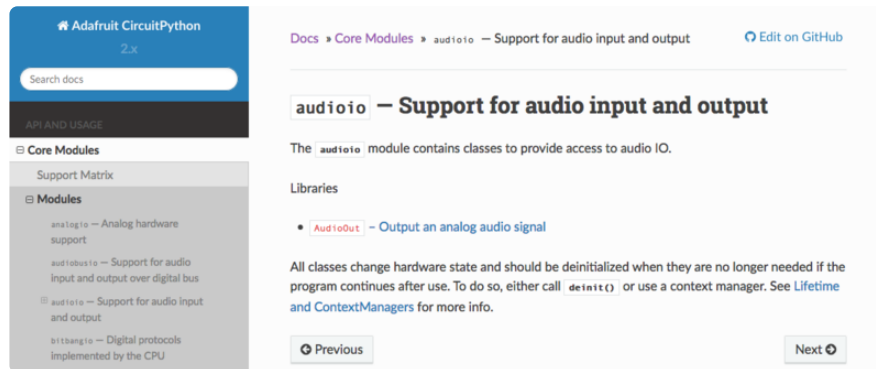
There are forum categories that cover all kinds of topics, including everything Adafruit. The [Adafruit CircuitPython](https://adafru.it/xXA) (<https://adafru.it/xXA>) category under "Supported Products & Projects" is the best place to post your CircuitPython questions.



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

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

Read the Docs



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

Here is blinky:

```
import time
import digitalio
import board

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

microSD Card Formatting Notes

Even though you can/could use your SD card 'raw' - it's most convenient to format the card to a filesystem. For the Arduino library we'll be discussing, and nearly every other SD library, the card must be formatted FAT16 or FAT32. Some only allow one or the other. The Arduino SD library can use either.

If you bought an SD card, chances are it's already pre-formatted with a FAT filesystem. However you may have problems with how the factory formats the card, or if it's an old card it needs to be reformatted. The Arduino SD library we use supports both **FAT16** and **FAT32** filesystems. If you have a very small SD card, say 8-32 Megabytes you might find it is formatted **FAT12** which isn't supported. You'll have to reformat these cards. Either way, it's **always** good idea to format the card before using, even if it's new! Note that formatting will erase the card so save anything you want first.

We strongly recommend you use the official SD card formatter utility - written by the SD association it solves many problems that come with bad formatting!

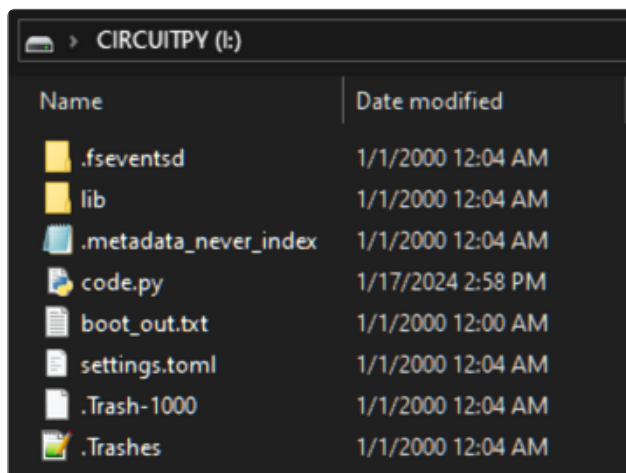
Download the formatter from <https://www.sdcard.org/downloads/formatter/> (<https://adafru.it/FKd>)

Download it and run it on your computer, there's also a manual linked from that page for use.

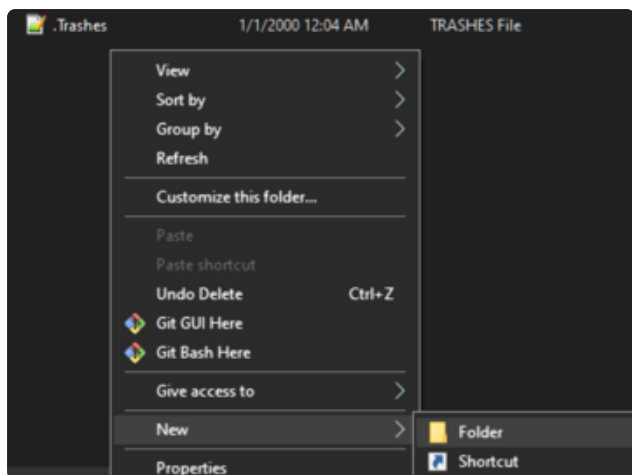
CircuitPython MEMENTO Starter Projects

Now that you've setup your MEMENTO with CircuitPython, you're ready to get started using the [PyCamera CircuitPython library](https://adafru.it/18e3) (<https://adafru.it/18e3>) for quick camera application development.

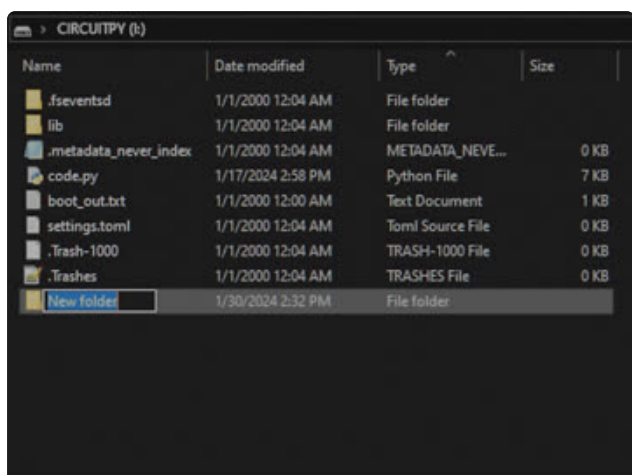
Before you load up the example project bundles though, you'll need to create an `/sd` directory on your MEMENTO **CIRCUITPY** drive. As of [CircuitPython 9](https://adafru.it/19eh) (<https://adafru.it/19eh>), this directory needs to be included on your **CIRCUITPY** drive when mounting an SD card. This change was made to allow for accessing the SD cards over web workflow.



To create this `/sd` directory, you'll open the **CIRCUITPY** drive in your operating system's file viewer window.

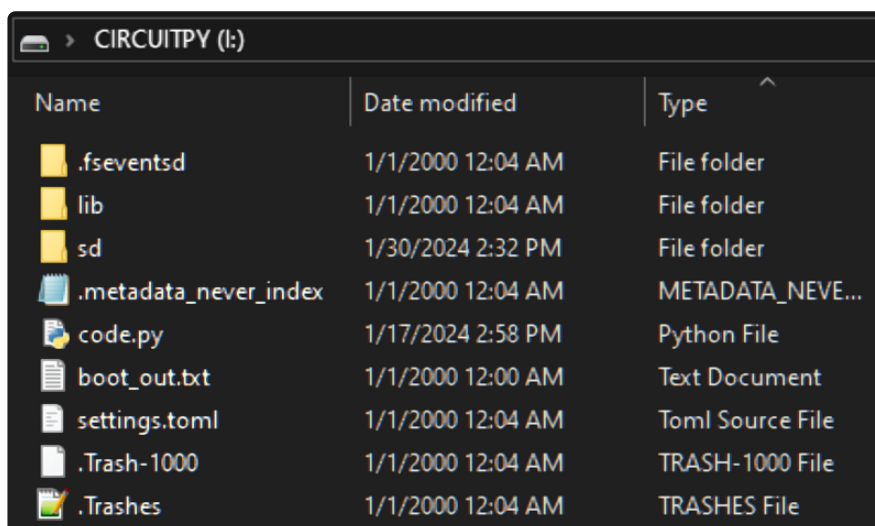


Then, create a new Folder on the **CIRCUITPY** drive.



Rename your new folder "**sd**".

Your **CIRCUITPY** directory will look like this after following these steps. Now you'll be able to mount SD cards with no problems in CircuitPython 9.



MEMENTO Camera Quick Start Guide

[MEMENTO Camera Quick Start Guide \(https://adafru.it/19Ha\)](https://adafru.it/19Ha)

CircuitPython Basic Camera



Ready to take some photos of your favorite goat-based still life*? Nice. Here you'll code a simple point-and-shoot camera as a way to get familiar with the MEMENTO's basic functions. It'll be just like that Instamatic you had in the 1970s, except digital and programmable!

* Other subjects are acceptable as well. We don't judge.

CircuitPython

Be sure you've [installed CircuitPython on your MEMENTO \(https://adafru.it/19ja\)](https://adafru.it/19ja) before proceeding.



SD Card

Format your SD card to FAT32 using the method shown on the [Formatting Notes](https://adafru.it/18ee) (<https://adafru.it/18ee>) page in this guide (don't rely on your operating system tools, the official formatter works more reliably).

With the MEMENTO turned off, insert the SD card into the MEMENTO's SD card reader as shown -- note the direction of the pins. Press it in until it bottoms out on the spring and then release pressure so it clicks into place.



NOTE: You can use cards with up to 32GB capacity.

A mount point named /sd is required on the CIRCUITPY drive. Make sure to create that directory after upgrading CircuitPython.

Follow these steps to create the /sd directory

<https://adafru.it/19ei>

Download the Project Bundle

Your project will use a specific set of CircuitPython libraries, and the **code.py** file. To get everything you need, click on the **Download Project Bundle** button below, and uncompress the .zip file.

Connect your computer to the board via a known good USB power+data cable. A new flash drive should show up as **CIRCUITPY**.

Drag the contents of the uncompressed bundle directory onto your board **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

```
# SPDX-FileCopyrightText: Copyright (c) 2023 john park for Adafruit Industries
#
# SPDX-License-Identifier: MIT
""" simple point-and-shoot camera example. No bells! Zero whistles! """

import time
import adafruit_pycamera # pylint: disable=import-error

pycam = adafruit_pycamera.PyCamera()
pycam.mode = 0 # only mode 0 (JPEG) will work in this example

# User settings - try changing these:
pycam.resolution = 8 # 0-12 preset resolutions:
#                      0: 240x240, 1: 320x240, 2: 640x480, 3: 800x600, 4: 1024x768,
#                      5: 1280x720, 6: 1280x1024, 7: 1600x1200, 8: 1920x1080, 9:
2048x1536,
#                      10: 2560x1440, 11: 2560x1600, 12: 2560x1920
pycam.led_level = 1 # 0-4 preset brightness levels
pycam.led_color = 0 # 0-7 preset colors: 0: white, 1: green, 2: yellow, 3: red,
#                                         4: pink, 5: blue, 6: teal, 7: rainbow
pycam.effect = 0 # 0-7 preset FX: 0: normal, 1: invert, 2: b&w, 3: red,
#                               4: green, 5: blue, 6: sepia, 7: solarize
```

```

print("Simple camera ready.")
pycam.tone(800, 0.1)
pycam.tone(1200, 0.05)

while True:
    pycam.blit(pycam.continuous_capture())
    pycam.keys_debounce()

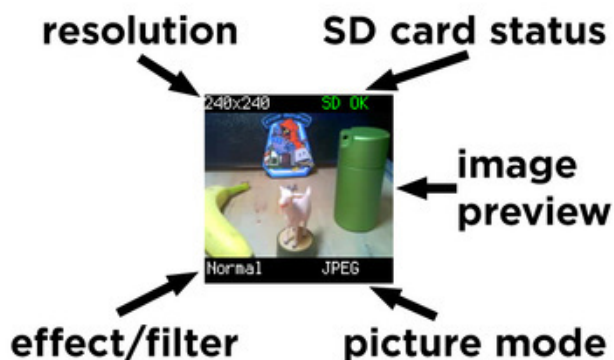
    if pycam.shutter.short_count:
        print("Shutter released")
        pycam.tone(1200, 0.05)
        pycam.tone(1600, 0.05)
        try:
            pycam.display_message("snap", color=0x00DD00)
            pycam.capture_jpeg()
            pycam.live_preview_mode()
        except TypeError as exception:
            pycam.display_message("Failed", color=0xFF0000)
            time.sleep(0.5)
            pycam.live_preview_mode()
        except RuntimeError as exception:
            pycam.display_message("Error\nNo SD Card", color=0xFF0000)
            time.sleep(0.5)

    if pycam.card_detect.fell:
        print("SD card removed")
        pycam.unmount_sd_card()
        pycam.display_refresh()

    if pycam.card_detect.rose:
        print("SD card inserted")
        pycam.display_message("Mounting\nSD Card", color=0xFFFFFF)
        for _ in range(3):
            try:
                print("Mounting card")
                pycam.mount_sd_card()
                print("Success!")
                break
            except OSError as exception:
                print("Retrying!", exception)
                time.sleep(0.5)
        else:
            pycam.display_message("SD Card\nFailed!", color=0xFF0000)
            time.sleep(0.5)
        pycam.display.refresh()

```

Flip the "-> On" switch and you're ready to go!



Camera HUD

The PyCamera library includes a real-time image preview and heads-up-display (HUD). The HUD provides the following information:

- image pixel resolution
- SD card status
- current effect/filter
- picture mode



Take a Photo

To snap a photo, flip the -> **On** switch to the right if the camera isn't already on, then once the camera has started and you see the preview image and "**SD OK**" simply press and release the shutter button (labeled with "**Cheese!**" or the 📷 sign if you have the back panel in place). You'll see "**snap**" show on screen and hear a beep.

The image will be saved to the SD card as a high-quality JPEG with a unique filename.



Image Retrieval

To admire your photo and send it to other goat fanciers, press and release the microSD card to pop it out of the camera. Be careful not to fling it across the room, that spring is pretty springy.

Put your microSD card into an SD card reader on your computer and open up the image (you may need a microSD to SD adapter such as [this](http://adafru.it/5447) (<http://adafru.it/5447>)). You can copy/paste or drag the images onto your computer's hard drive, too. Be sure to eject the microSD card drive from your computer before physically removing it.

Here you can see an 800x600 resolution image (the thumbnail of the preview and HUD won't be there, those were added for clarity in this guide).



Change Resolution

In code you can adjust the resolution by setting a different value for `pycam.resolution`.

```
pycam.resolution = 0
# 0-12 preset resolutions:
# 0: 240x240, 1: 320x240, 2: 640x480, 3: 800x600, 4: 1024x768,
# 5: 1280x720, 6: 1280x1024, 7: 1600x1200, 8: 1920x1080, 9: 2048x1536,
# 10: 2560x1440, 11: 2560x1600, 12: 2560x1920
```

Here's the `0` setting, which is a 240x240 image.



This is the 1920x1080 option, click the image for the option to see the original at full size.



This is the 2560x1920 option, click the image for the option to see the original at full size.

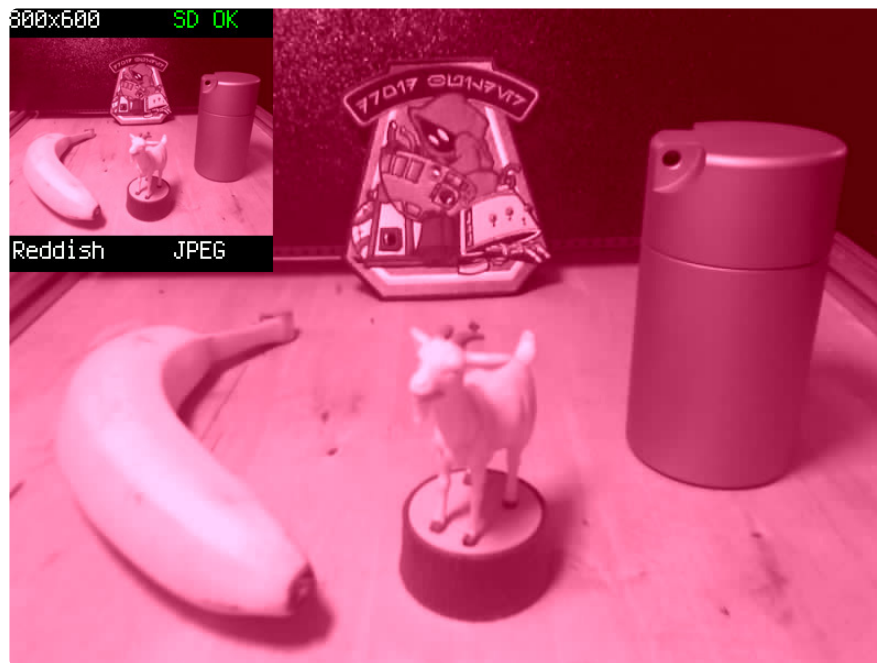


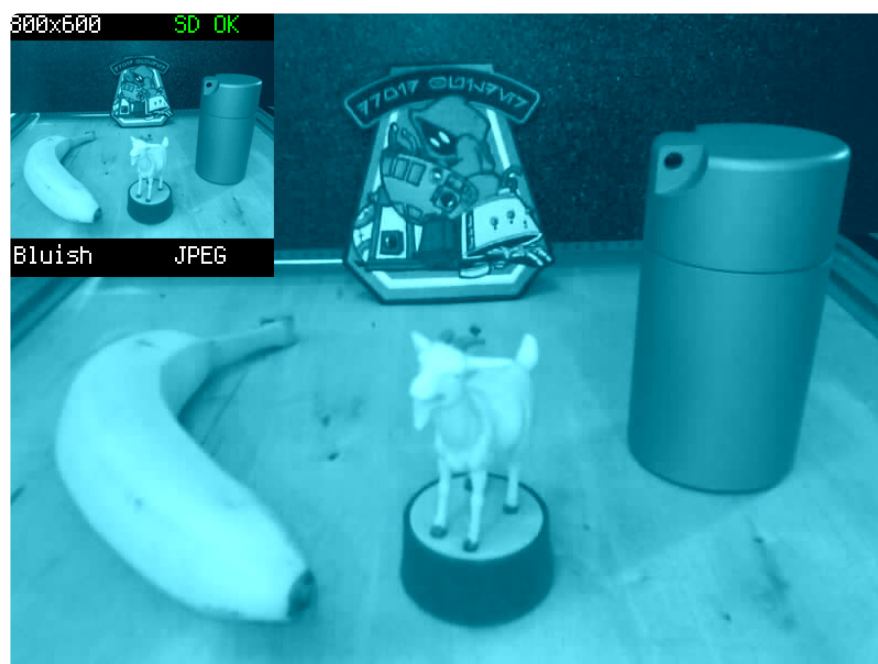
Effects

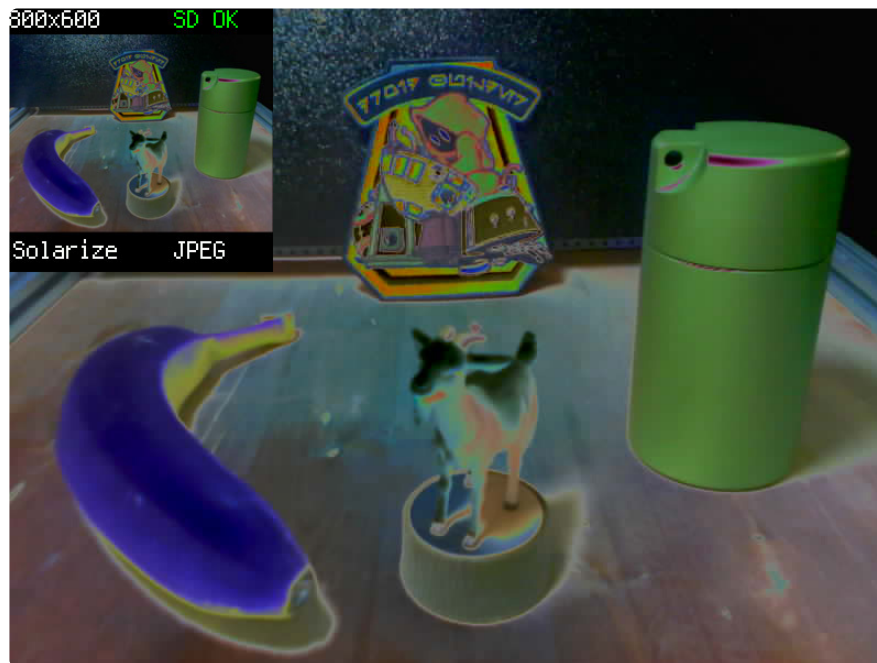
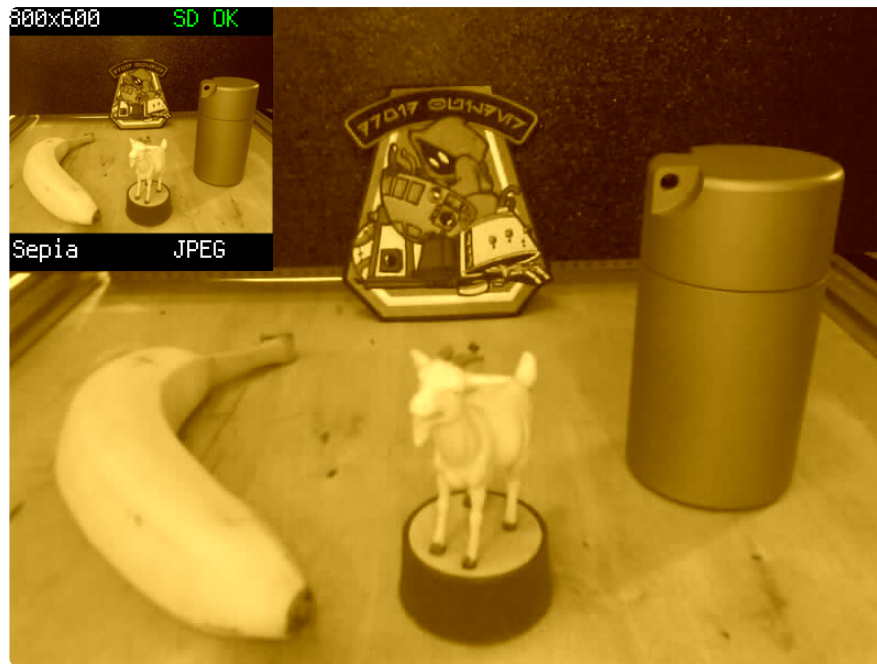
There are some built-in effects/filters you can try by adjusting the `pycam.effect` value.

```
pycam.effect = 0 # 0-7 preset FX: 0: normal, 1: invert, 2: b&w, 3: red,
# 4: green, 5: blue, 6: sepia, 7: solarize
```









LED Color

If you have the LED light ring faceplate connected, you can set LED brightness and color values using `pycam.led_level` and `pycam.led_color`.

```
pycam.led_level = 1 # 0-4 preset brightness levels
pycam.led_color = 0 # 0-7 preset colors: 0: white, 1: green, 2: yellow, 3: red,
# 4: pink, 5: blue, 6: teal, 7: rainbow
```



Here are some examples of colored lighting on the scene. There is a lot of ambient light here, so the color is reasonably subtle. With less ambient light the color will be more intense.

More Camera!

Want to see an example with even more features? Check out the example on the next page!

Fancy Camera

This example from the Adafruit CircuitPython PyCamera library is similar to the Basic Camera, but adds **autofocus** and on-camera user interaction via the MEMENTO's buttons to give you more of a point-and-shoot camera experience.

You can use the buttons to adjust:

- resolutions
- effects
- modes
- optional NeoPixel LED colors

Format your SD card if necessary and insert it in the MEMENTO SD card reader as shown on the [Basic Camera page \(https://adafru.it/18ef\)](https://adafru.it/18ef).

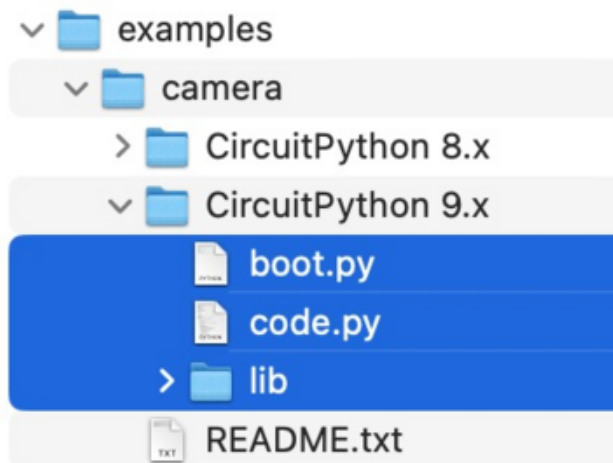
CircuitPython

Be sure you've [installed CircuitPython on your MEMENTO \(https://adafru.it/19ja\)](https://adafru.it/19ja) before proceeding.

Download the Project Bundle

Your project will use a specific set of CircuitPython libraries, and the **code.py** file. To get everything you need, click on the **Download Project Bundle** button below, and uncompress the .zip file.

Connect your computer to the board via a known good USB power+data cable. A new flash drive should show up as **CIRCUITPY**.



Drag the contents of the uncompressed bundle directory for the version of CircuitPython you're running (e.g., / **examples/camera/CircuitPython 9.x/**) onto your board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

```
# SPDX-FileCopyrightText: 2023 Jeff Epler for Adafruit Industries
# SPDX-FileCopyrightText: 2023 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import ssl
import os
import time
import socketpool
import adafruit_requests
import rtc
import adafruit_ntp
import wifi
import bitmaptools
import displayio
```

```

import gifio
import ulab.numpy as np

import adafruit_pycamera

# Wifi details are in settings.toml file, also,
# timezone info should be included to allow local time and DST adjustments
# # UTC_OFFSET, if present, will override TZ and DST and no API query will be done
# UTC_OFFSET=-25200
# # TZ="America/Phoenix"

UTC_OFFSET = os.getenv("UTC_OFFSET")
TZ = os.getenv("TZ")

print(f"Connecting to {os.getenv('CIRCUITPY_WIFI_SSID')}")
SSID = os.getenv("CIRCUITPY_WIFI_SSID")
PASSWORD = os.getenv("CIRCUITPY_WIFI_PASSWORD")

if SSID and PASSWORD:
    wifi.radio.connect(
        os.getenv("CIRCUITPY_WIFI_SSID"), os.getenv("CIRCUITPY_WIFI_PASSWORD")
    )
    if wifi.radio.connected:
        print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}!")
        print("My IP address is", wifi.radio.ipv4_address)
        pool = socketpool.SocketPool(wifi.radio)

        if UTC_OFFSET is None:
            requests = adafruit_requests.Session(pool, ssl.create_default_context())
            response = requests.get("http://worldtimeapi.org/api/timezone/" + TZ)
            response_as_json = response.json()
            UTC_OFFSET = response_as_json["raw_offset"] +
response_as_json["dst_offset"]
            print(f"UTC_OFFSET: {UTC_OFFSET}")

            ntp = adafruit_ntp.NTP(
                pool, server="pool.ntp.org", tz_offset=UTC_OFFSET // 3600
            )

            print(f"ntp time: {ntp.datetime}")
            rtc.RTC().datetime = ntp.datetime
        else:
            print("Wifi failed to connect. Time not set.")
    else:
        print("Wifi config not found in settintgs.toml. Time not set.")

pycam = adafruit_pycamera.PyCamera()
# pycam.live_preview_mode()

settings = (
    None,
    "resolution",
    "effect",
    "mode",
    "led_level",
    "led_color",
    "timelapse_rate",
)
curr_setting = 0

print("Starting!")
# pycam.tone(200, 0.1)
last_frame = displayio.Bitmap(pycam.camera.width, pycam.camera.height, 65535)
onionskin = displayio.Bitmap(pycam.camera.width, pycam.camera.height, 65535)
timelapse_remaining = None
timelapse_timestamp = None

while True:
    if pycam.mode_text == "STOP" and pycam.stop_motion_frame != 0:

```

```

        # alpha blend
        new_frame = pycam.continuous_capture()
        bitmaptools.alphablend(
            onionskin, last_frame, new_frame, displayio.Colorspace.RGB565_SWAPPED
        )
        pycam.blit(onionskin)
    elif pycam.mode_text == "GB0Y":
        bitmaptools.dither(
            last_frame, pycam.continuous_capture(),
displayio.Colorspace.RGB565_SWAPPED
        )
        pycam.blit(last_frame)
    elif pycam.mode_text == "LAPS":
        if timelapse_remaining is None:
            pycam.timelapsestatus_label.text = "STOP"
        else:
            timelapse_remaining = timelapse_timestamp - time.time()
            pycam.timelapsestatus_label.text = f"{timelapse_remaining}s /      "
            # Manually updating the label text a second time ensures that the label
            # is re-painted over the blitted preview.
            pycam.timelapse_rate_label.text = pycam.timelapse_rate_label.text
            pycam.timelapse_submode_label.text = pycam.timelapse_submode_label.text

            # only in high power mode do we continuously preview
            if (timelapse_remaining is None) or (
                pycam.timelapse_submode_label.text == "HiPwr"
            ):
                pycam.blit(pycam.continuous_capture())
            if pycam.timelapse_submode_label.text == "LowPwr" and (
                timelapse_remaining is not None
            ):
                pycam.display.brightness = 0.05
            else:
                pycam.display.brightness = 1
            pycam.display.refresh()

        if timelapse_remaining is not None and timelapse_remaining <= 0:
            # no matter what, show what was just on the camera
            pycam.blit(pycam.continuous_capture())
            # pycam.tone(200, 0.1) # uncomment to add a beep when a photo is taken
            try:
                pycam.display_message("Snap!", color=0x0000FF)
                pycam.capture_jpeg()
            except TypeError as e:
                pycam.display_message("Failed", color=0xFF0000)
                time.sleep(0.5)
            except RuntimeError as e:
                pycam.display_message("Error\nNo SD Card", color=0xFF0000)
                time.sleep(0.5)
            pycam.live_preview_mode()
            pycam.display.refresh()
            pycam.blit(pycam.continuous_capture())
            timelapse_timestamp = (
                time.time() + pycam.timelapse_rates[pycam.timelapse_rate] + 1
            )
        else:
            pycam.blit(pycam.continuous_capture())
            # print("\t\t", capture_time, blit_time)

        pycam.keys_debounce()
        # test shutter button
        if pycam.shutter.long_press:
            print("FOCUS")
            print(pycam.autofocus_status)
            pycam.autofocus()
            print(pycam.autofocus_status)
        if pycam.shutter.short_count:
            print("Shutter released")
            if pycam.mode_text == "STOP":

```

```

pycam.capture_into_bitmap(last_frame)
pycam.stop_motion_frame += 1
try:
    pycam.display_message("Snap!", color=0x0000FF)
    pycam.capture_jpeg()
except TypeError as e:
    pycam.display_message("Failed", color=0xFF0000)
    time.sleep(0.5)
except RuntimeError as e:
    pycam.display_message("Error\nNo SD Card", color=0xFF0000)
    time.sleep(0.5)
pycam.live_preview_mode()

if pycam.mode_text == "GB0Y":
    try:
        f = pycam.open_next_image("gif")
    except RuntimeError as e:
        pycam.display_message("Error\nNo SD Card", color=0xFF0000)
        time.sleep(0.5)
        continue

    with gifio.GifWriter(
        f,
        pycam.camera.width,
        pycam.camera.height,
        displayio.Colorspace.RGB565_SWAPPED,
        dither=True,
    ) as g:
        g.add_frame(last_frame, 1)

if pycam.mode_text == "GIF":
    try:
        f = pycam.open_next_image("gif")
    except RuntimeError as e:
        pycam.display_message("Error\nNo SD Card", color=0xFF0000)
        time.sleep(0.5)
        continue
    i = 0
    ft = []
    pycam._mode_label.text = "RECORDING" # pylint: disable=protected-access

    pycam.display.refresh()
    with gifio.GifWriter(
        f,
        pycam.camera.width,
        pycam.camera.height,
        displayio.Colorspace.RGB565_SWAPPED,
        dither=True,
    ) as g:
        t00 = t0 = time.monotonic()
        while (i < 15) or not pycam.shutter_button.value:
            i += 1
            _gifframe = pycam.continuous_capture()
            g.add_frame(_gifframe, 0.12)
            pycam.blit(_gifframe)
            t1 = time.monotonic()
            ft.append(1 / (t1 - t0))
            print(end=".")
            t0 = t1
        pycam._mode_label.text = "GIF" # pylint: disable=protected-access
        print(f"\nfinal size {f.tell()} for {i} frames")
        print(f"average framerate {i / (t1 - t00)}fps")
        print(f"best {max(ft)} worst {min(ft)} std. deviation {np.std(ft)}")
        f.close()
        pycam.display.refresh()

if pycam.mode_text == "JPEG":
    pycam.tone(200, 0.1)
    try:

```

```

        pycam.display_message("Snap!", color=0x0000FF)
        pycam.capture_jpeg()
        pycam.live_preview_mode()
    except TypeError as e:
        pycam.display_message("Failed", color=0xFF0000)
        time.sleep(0.5)
        pycam.live_preview_mode()
    except RuntimeError as e:
        pycam.display_message("Error\nNo SD Card", color=0xFF0000)
        time.sleep(0.5)

if pycam.card_detect.fell:
    print("SD card removed")
    pycam.unmount_sd_card()
    pycam.display.refresh()
if pycam.card_detect.rose:
    print("SD card inserted")
    pycam.display_message("Mounting\nSD Card", color=0xFFFFFF)
    for _ in range(3):
        try:
            print("Mounting card")
            pycam.mount_sd_card()
            print("Success!")
            break
        except OSError as e:
            print("Retrying!", e)
            time.sleep(0.5)
    else:
        pycam.display_message("SD Card\nFailed!", color=0xFF0000)
        time.sleep(0.5)
    pycam.display.refresh()

if pycam.up.fell:
    print("UP")
    key = settings[curr_setting]
    if key:
        print("getting", key, getattr(pycam, key))
        setattr(pycam, key, getattr(pycam, key) + 1)
if pycam.down.fell:
    print("DN")
    key = settings[curr_setting]
    if key:
        setattr(pycam, key, getattr(pycam, key) - 1)
if pycam.right.fell:
    print("RT")
    curr_setting = (curr_setting + 1) % len(settings)
    if pycam.mode_text != "LAPS" and settings[curr_setting] == "timelapse_rate":
        curr_setting = (curr_setting + 1) % len(settings)
    print(settings[curr_setting])
    # new_res = min(len(pycam.resolutions)-1, pycam.get_resolution()+1)
    # pycam.set_resolution(pycam.resolutions[new_res])
    pycam.select_setting(settings[curr_setting])
if pycam.left.fell:
    print("LF")
    curr_setting = (curr_setting - 1 + len(settings)) % len(settings)
    if pycam.mode_text != "LAPS" and settings[curr_setting] == "timelaps_rate":
        curr_setting = (curr_setting + 1) % len(settings)
    print(settings[curr_setting])
    pycam.select_setting(settings[curr_setting])
    # new_res = max(1, pycam.get_resolution()-1)
    # pycam.set_resolution(pycam.resolutions[new_res])
if pycam.select.fell:
    print("SEL")
    if pycam.mode_text == "LAPS":
        pycam.timelapse_submode += 1
        pycam.display.refresh()
if pycam.ok.fell:
    print("OK")
    if pycam.mode_text == "LAPS":

```

```

if timelapse_remaining is None: # stopped
    print("Starting timelapse")
    timelapse_remaining = pycam.timelapse_rates[pycam.timelapse_rate]
    timelapse_timestamp = time.time() + timelapse_remaining + 1
    # dont let the camera take over auto-settings
    saved_settings = pycam.get_camera_autosettings()
    # print(f"Current exposure {saved_settings}")
    pycam.set_camera_exposure(saved_settings["exposure"])
    pycam.set_camera_gain(saved_settings["gain"])
    pycam.set_camera_wb(saved_settings["wb"])
else: # is running, turn off
    print("Stopping timelapse")

    timelapse_remaining = None
    pycam.camera.exposure_ctrl = True
    pycam.set_camera_gain(None) # go back to autogain
    pycam.set_camera_wb(None) # go back to autobalance
    pycam.set_camera_exposure(None) # go back to auto shutter

```

Use the Camera

Taking pictures is just as simple as with the Basic Camera -- you could say it's a snap -- simply follow these two steps:

1. point the MEMENTO at a subject while framing it in the display
2. press-and-release the shutter button fairly quickly

Autofocus

However, now you can use the autofocus feature to help keep closer subjects looking sharp:

1. point the MEMENTO at a subject while framing it in the display
2. press-and-hold the shutter button until you hear the beep (you should also see the focus change if there is a close subject in frame)
3. release the shutter button to finish enabling autofocus
4. press-and-release the shutter button to snap your pic!

I ate the banana. So now we have a can of Liquid Wrench to take its place. Which is great, because it is still yellow, but has a lot more detail on which to focus!

In the first photo, the MEMENTO's default distant focus is shown. Note how the Droid Builders patch is in focus but the mid and foreground objects aren't.

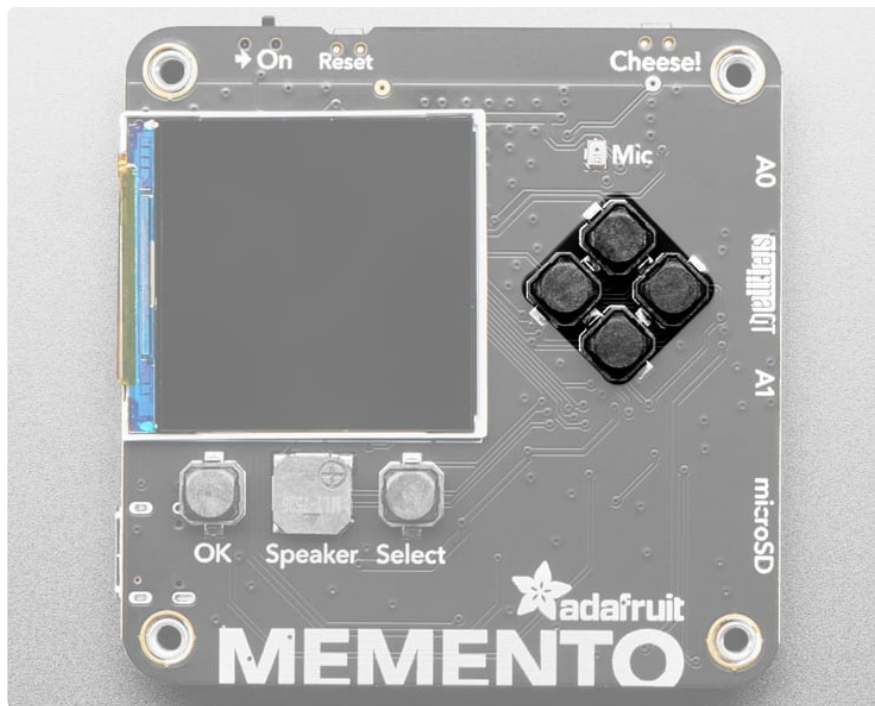


For this second shot I pointed the camera at the can of Liquid Wrench, held the shutter button to set the MEMENTO's autofocus, released it, then aimed it back at the patch and snapped the pic. Such nice close-up focus!



Settings

While the Basic Camera example settings could only be changed directly in code, in the Fancy Camera example you can use the MEMENTO's four directional buttons to change settings on the fly.



- press **Right** button to pick a menu category **Resolution**, **Effect**, or **Mode**
- the category will be highlighted
- press **Up** or **Down** buttons to select the choices within the mode:
 - **Resolution** will page through the different available resolutions
 - **Effect** picks the different effects, such as **Normal**, **Invert**, **B&W**, **Reddish**, **Sepia**, **Solarize**, etc.
 - **Mode** will flip between **JPEG**, **GIF**, **GBOY**, **STOP**, **LAPS** (we'll cover stop motion mode and timelapse modes elsewhere in this guide)



Timelapse



A sprout growing quickly from the soil. Clouds racing across the sky. A building constructed from scratch in a matter of minutes. Jigsaw puzzles coming together in moments. These are some of the things you can do with timelapse photography. All

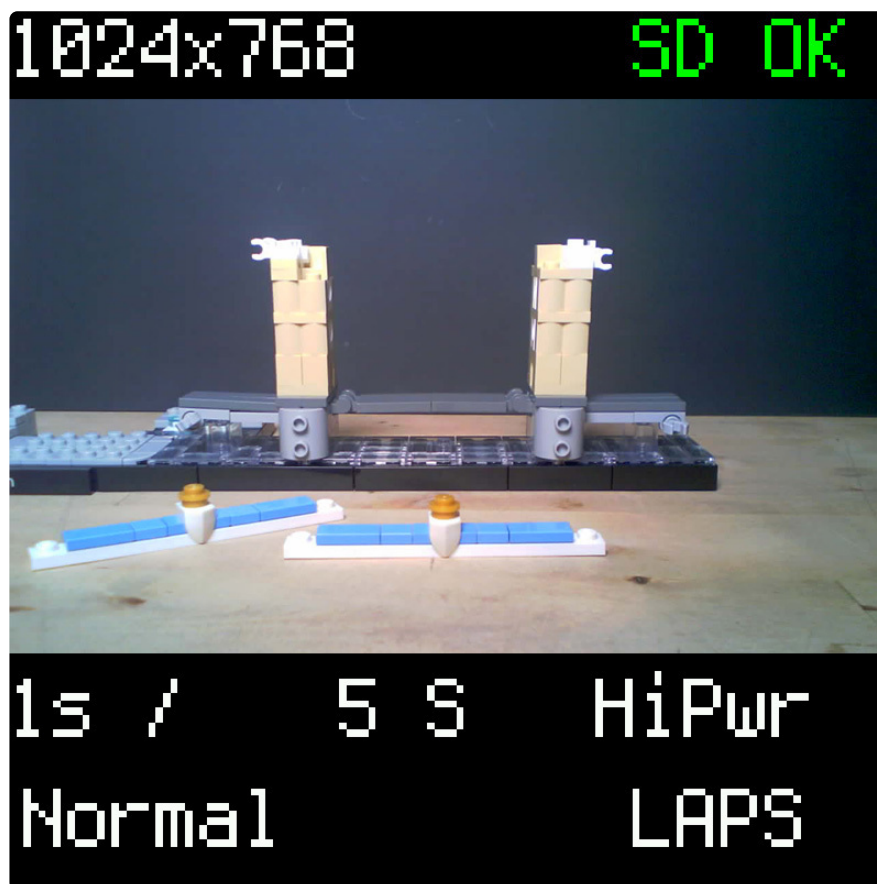
you need is a camera and an intervalometer. Luckily for us, the MEMENTO can be both!

The camera part is straightforward, but what about this so-called "intervalometer"? That's a fancy term for a gizmo that can press the shutter for you (hardware or software) at a pre-set interval until you tell it to stop. It never gets tired, so you can set it to shoot once per second or per hour or even just once per day, to capture a long event.

Then, you'll play back those individual still frames rapidly as a movie or GIF. Time flies when you're having fun!

CircuitPython

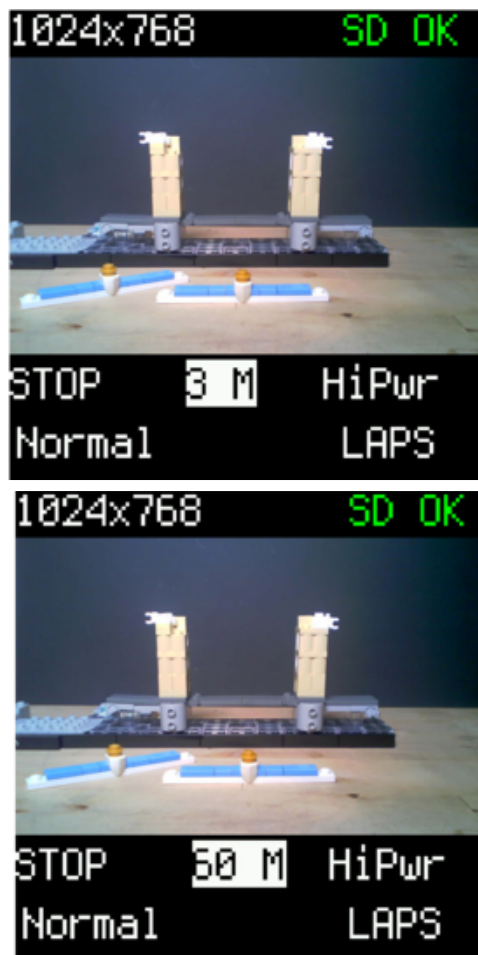
Be sure you've [installed CircuitPython on your MEMENTO](https://adafru.it/19ja) before proceeding.



LAPS

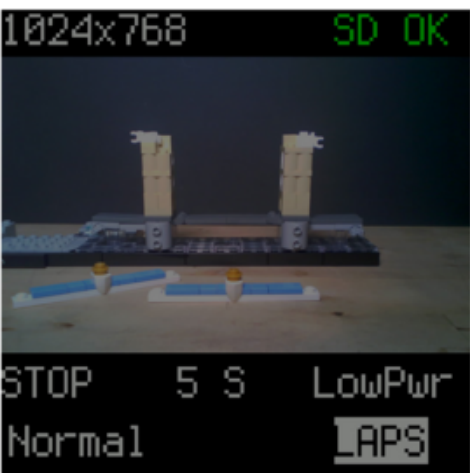
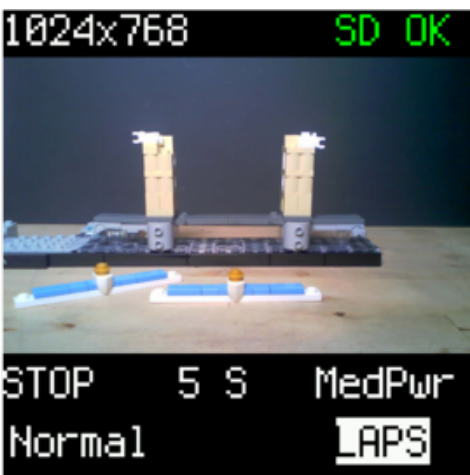
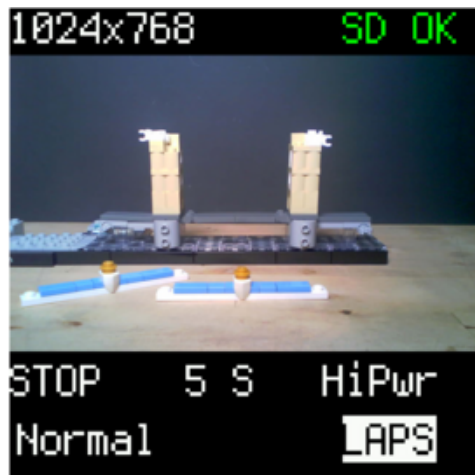
No, we won't be hitting the track -- we just abbreviated "timelapse" to "LAPS" in the menu system.

To switch to **timelapse** mode, you'll use the right or left button to get to the **mode** menu, then press up or down to switch to the **LAPS** selection.



Intervals

You'll see a new menu set show up for adjusting the timelapse settings. Press the **right** button until the timelapse interval item is highlighted. You can then press up or down to increase or decrease the preset intervals, from **5 seconds** up to an hour between shots.



Power

Press the **Select** button to choose the High, Medium, or Low power timelapse modes. These are helpful if you are running off of the MEMENTO battery --

high power mode shows a live preview
medium power mode only updates the display once per shot based on the selected time interval

low power mode dims the display as well as limiting updates to once per shot

Note: in timelapse mode you can still select your other settings, such as resolution, effects, and LED color and brightness as usual

Focus

You may wish to long-press the shutter button to focus on your subject when shooting a timelapse. All of the other automatic camera settings, such as exposure/gain, will

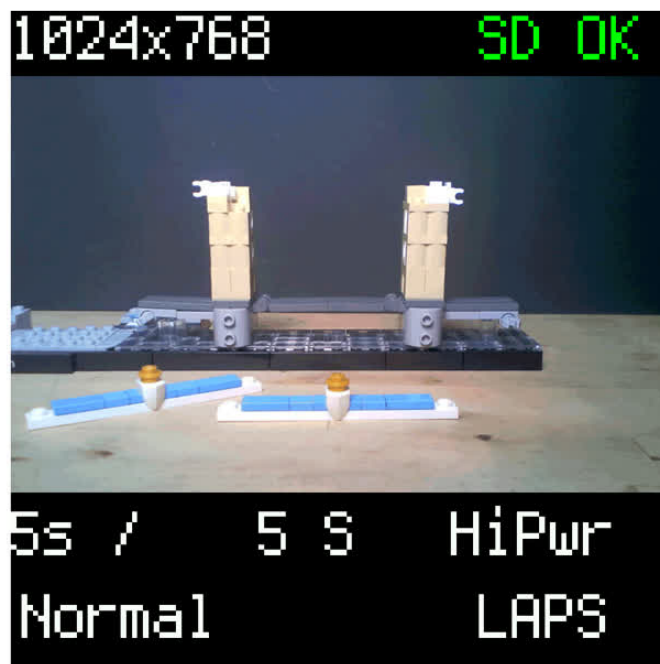
lock onto their first frame settings, to prevent flickering in the final timelapse movie you'll create from your frames.

Start/Stop

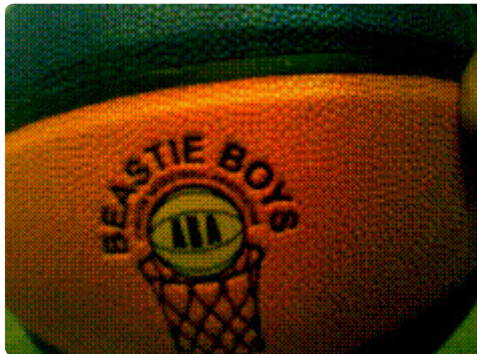
Once you've got your settings dialed in, point the MEMENTO at your subject (a tripod case or other mounting is helpful to prevent accidental bumps) and then press the MEMENTO **OK** button.

The on-screen display will show a countdown timer and then snap a photo. This repeats until you press **OK** again to stop it (or until you run out of SD card space or battery!) So you can capture the clouds moving across the sky all day long, or an hour long house-of-cards building session, it's up to you!

After shooting your frames, bring the files into your computer and convert them into an animation using your favorite editing software, such as Photoshop, or a GIF creation web page. [This page \(https://adafru.it/19ev\)](https://adafru.it/19ev) shows you how to do this step-by-step using ezgif.com/maker



Animated GIF Creation



You can create short animated GIFs with the MEMENTO, perfect for sharing in text chats!

The Fancy Camera example will record a 15 frame, 240x176 pixel GIF* of about two seconds in duration each time you press and release the shutter button.

CircuitPython

Be sure you've [installed CircuitPython on your MEMENTO \(https://adafru.it/19ja\)](https://adafru.it/19ja) before proceeding.

GIF Mode

To enter **GIF** mode, press the right button three times, you'll see the mode text highlight. By default it will be in **JPEG** mode.

Press the up button to cycle modes and choose **GIF** mode.

GIF mode is locked at 240x176 pixels, so you won't see the resolution selection option that is present in JPEG mode

Make a GIF

Now, simply press and release the shutter button to record a two-second GIF animation. The GIF is saved automatically to the SD card. There is no on-camera preview, so you'll need to put the SD card in your computer's SD card reader to view it.

GIF Code

Here's the code snippet from the Fancy Camera code used for GIF creation -- you can use this to incorporate GIF recording into your own custom code:

```
if pycam.mode_text == "GIF":
    try:
        f = pycam.open_next_image("gif")
    except RuntimeError as e:
        pycam.display_message("Error\nNo SD Card", color=0xFF0000)
        time.sleep(0.5)
        continue
    i = 0
    ft = []
    pycam._mode_label.text = "RECORDING" # pylint: disable=protected-access

    pycam.display.refresh()
    with gifio.GifWriter(
        f,
        pycam.camera.width,
        pycam.camera.height,
        displayio.Colorspace.RGB565_SWAPPED,
        dither=True,
    ) as g:
        t00 = t0 = time.monotonic()
        while (i < 15) or not pycam.shutter_button.value:
            i += 1
            _gifframe = pycam.continuous_capture()
            g.add_frame(_gifframe, 0.12)
            pycam.blit(_gifframe)
            t1 = time.monotonic()
            ft.append(1 / (t1 - t0))
            print(end=".")
            t0 = t1
        pycam._mode_label.text = "GIF" # pylint: disable=protected-access
        print(f"\nfinal size {f.tell()} for {i} frames")
        print(f"average framerate {i/(t1-t00)}fps")
        print(f"best {max(ft)} worst {min(ft)} std. deviation {np.std(ft)}")
        f.close()
    pycam.display.refresh()
```

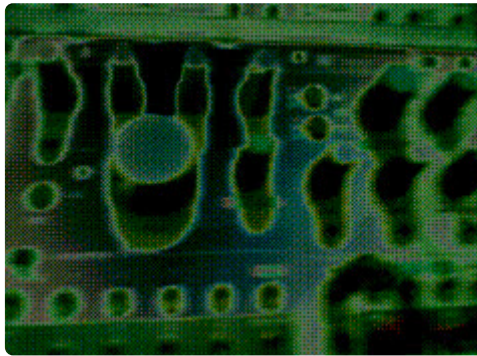
Effects

The default GIF is created without effects applied, but you can get creative by picking built-in effects before shooting a GIF.

Press the right button to highlight the effect menu item, by default it is the **Normal** effect. Press up/down to cycle between the effects:

- Invert
- B&W
- Reddish
- Greenish
- Blueish
- Sepia
- Solarize

Then, shoot your GIF by pressing the shutter button as usual. Here are some examples:



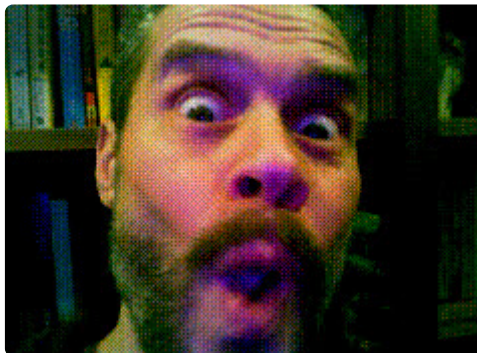
NeoPixel Lighting

You can also add some NeoPixel lighting to your GIFs if you have the MEMENTO Camera Enclosure kit attached and connected over the JST cable.

Press the right button until **LED LV** is highlighted, then press up to pick a pre-set lighting level.

Press the right button again to highlight **LED CLR**, then press the up button so pick a pre-set color.

Here's an example of a weird guy with some magenta lighting highlights that look particularly odd inside his mouth. Freaky.



Post Processing

You can also use GIF editing tools (online or applications) to further edit your GIFs. Here's an example of creating a looping version of the basketball animation by doubling and re-ordering the frames:



* Pronounced **GHIF** as in. "Good grief, give me a break, go get some peanut butter for your graphics if you wanna say 'JIF'."

Don't @ me bro.

Stop Motion

You can use the MEMENTO to create simple stop motion animations. By taking a series of still pictures and stitching them together, you can create an animation!

Persistence of Vision

The key principal to understand for any kind of animation, including stop motion, is persistence of vision. Our brains are very good at filling in the blanks and imagining continuous motion when we see a series of still images displayed in quick succession.

This is how motion pictures work, as well as mechanical illusions, such as zoetropes and flipbooks. Traditional animation methods such as hand drawn animation, and puppet- or clay-based stop motion animation work the same way. Create a single frame by drawing a pose, or posing a figure, and then shoot a frame of the image

onto film, or, more likely these days, a digital photograph. Then, create a new pose, shoot a second frame, and repeat this on and on.

When you then rapidly review those images you photographed, suddenly your subject starts moving and comes to life!



For more in-depth info on creating stop motion animation, [check out this guide \(https://adafru.it/18es\)](https://adafru.it/18es).

CircuitPython

Be sure you've [installed CircuitPython on your MEMENTO \(https://adafru.it/19ja\)](https://adafru.it/19ja) before proceeding.

Onion Skinning

One feature of the MEMENTO camera that's helpful for stop motion is the onion skinning, a.k.a. ghosting feature. This is an overlay you'll see on the screen of your previously shot frame while you're composing your current frame. It lets you see where your character was so you can gauge how far to move it.

In the Fancy Camera application on the MEMENTO, press the right button to highlight the mode item and then press up to change from **JPEG**, **GIF**, or **GBOY** to **STOP** mode.

At first the viewer will appear as usual, but after you shoot one frame you'll then see a semi-transparent overlay of the previously saved image on top of the live view.

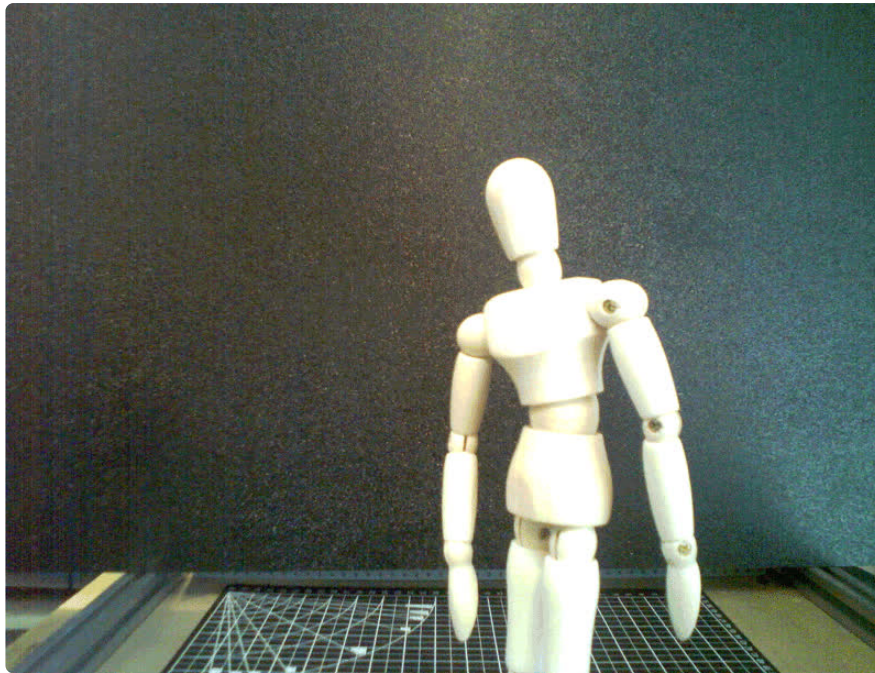


Wave

Try creating a hand waving animation with a mannequin or other figure. You can try moving straight ahead with the poses, referencing the previous pose overlay to see where you were.

You can take multiple photos at one pose to hold the pose and to slow down the action.

After shooting your frames, bring the files into your computer and convert them into an animation using your favorite editing software, such as Photoshop, or a GIF creation web page. [This page \(https://adafru.it/19ev\)](https://adafru.it/19ev) shows you how to do this step-by-step using ezgif.com/maker

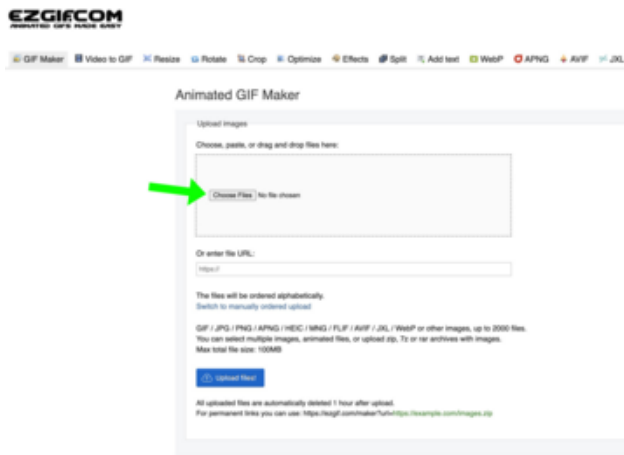


Frames to GIFs



When you shoot stop motion frames or timelapse sessions you'll end up with a sequence of .jpg images on your SD card. That's rad and all, but how do you stitch them into a single GIF animation so everyone can enjoy them easily when you share them in text messages or on social media?

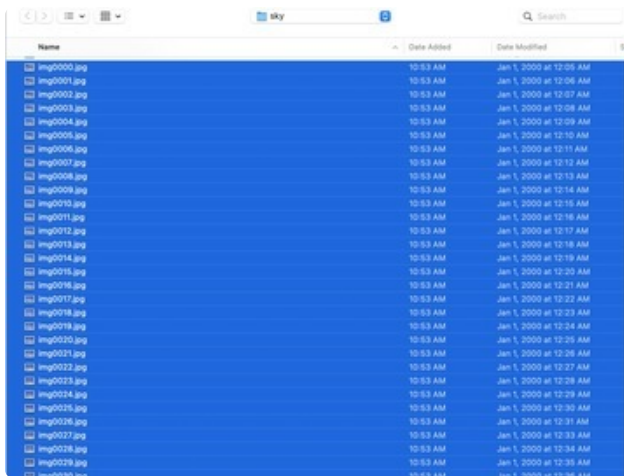
One simple way is to use ezgif.com (<https://adafru.it/18et>) It allows you to upload a sequence of images, adjust things like timing, crop, and size, and then convert them into a single animated GIF you can download.



GIF Maker

Go to <https://ezgif.com/maker> (<https://adafru.it/18et>) in your web browser.

Click on the Choose Files button to open your file browser.



Select Files

Use your file browser to select the sequence of .jpg frames on your MEMENTO SD card that you've put in your computer's SD card reader, then click **Open** in the file browser window.

You can hover over the Choose Files button to see the list of images.



If you'd like to try out this process using some existing frames, the .zip file below contains the original sky timelapse photos taken with the MEMENTO camera.

skylapse.zip

<https://adafru.it/19ew>

Animated GIF Maker

Upload Images

Choose, paste, or drag and drop files here:

Choose Files

No file chosen

Or enter file URL:

https://

The files will be ordered alphabetically.
[Switch to manually ordered upload](#)

GIF / JPG / PNG / APNG / HEIC / MNG / FLIF / AVIF / JXL / WebP or other images, up to 2000 files.
You can select multiple images, animated files, or upload zip, 7z or rar archives with images.
Max total file size: 100MB

Upload files

All uploaded files are automatically deleted 1 hour after upload.
For permanent links you can use: <https://ezgif.com/maker?url=https://example.com/images.zip>

https://

The files will be ordered alphabetically.
[Switch to manually ordered upload](#)

GIF / JPG / PNG / APNG / HEIC / MNG / FLIF / AVIF / JXL / WebP or other images, up to 20 images, up to 20
You can select multiple images, animated files, or upload zip, 7z or rar archives with images
Max total file size: 100MB

Uploading files...

All uploaded files are automatically deleted 1 hour after upload.
For permanent links you can use: <https://ezgif.com/maker?url=https://example.com/images.zip>

Upload Files

Click the **Upload files** button to upload the image sequence to the ezgif server in the sky. The button will gray out and display "Uploading files" while it is transferring images.

Animated GIF Maker

(Drag and drop frames to change their order)



Frame Order, Delay

Once your frames have been uploaded the thumbnail view will appear. Here you can re-order frames by drag-dropping them, adjust delay timing, skip, and copy frames.

Toggle a range of frames:
 From: 1 To: 5 [Skip](#) [Enable](#)

GIF options:
 Delay time: 20 (in 1/100 of second, changing this value will reset delay for all frames)
 Loop count: (Empty - loop forever)
☐ use global colormap (Use the same set of colors for all frames to reduce file size)

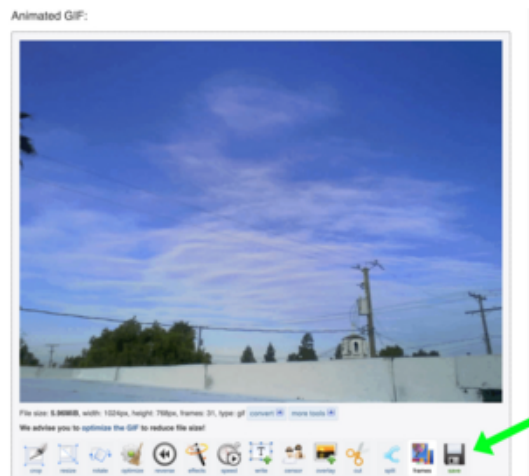
Effects:
☐ crossfade frames
☐ don't stack frames (Remove the frame when it's time to display next one, use for images with transparent background)

[Make a GIF!](#)

Animated GIF:

Make a GIF

When you're ready to have the GIF created, click the **Make a GIF** button.



Save Your GIF

Once the GIF is ready it will play back for you in the browser window to review. You can make changes at this point, just check out the icons at the bottom, they're very cool and pretty self-explanatory.

When you want to download your GIF, click the **save** button with the floppy disc icon. This will download to your system/ browser default download location.



Arduino IDE Setup

The ESP32-S2/S3 bootloader does not have USB serial support for Windows 7 or 8. (See <https://github.com/espressif/arduino-esp32/issues/5994>) please update to version 10 which is supported by espressif! Alternatively you can try this community-crafted Windows 7 driver (<https://github.com/kutukvpavel/Esp32-Win7-VCP-drivers>)

Install Arduino IDE

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

Arduino IDE Download

<https://adafru.it/f1P>

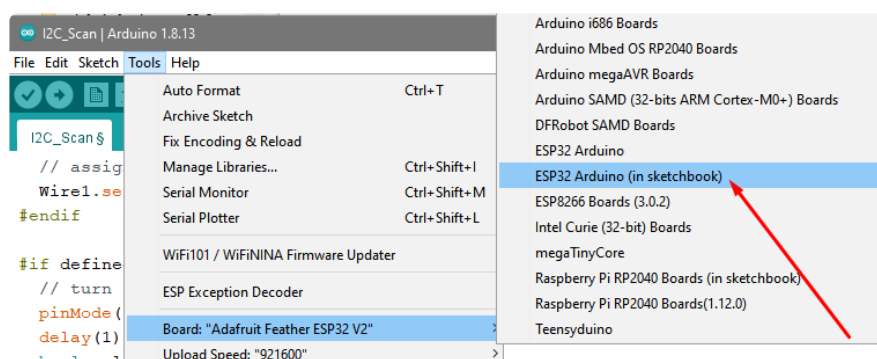
Install ESP32 Board Support Package from GitHub

For this board, we recommend you don't use 'release' version of Espressif's board support package because the current release doesn't include board support.

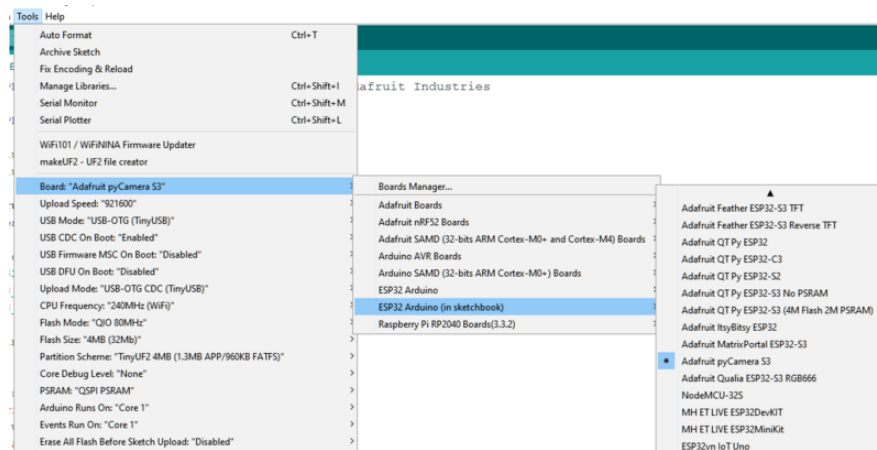
Instead we will install the "very latest" [by following these instructions \(https://adafru.it/YYB\)](https://adafru.it/YYB) (scroll down for Mac and Linux as well)

Basically, install by **git clone**-ing the Espressif ESP32 board support to get the very latest version of the code.

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



Look for the board called Adafruit pyCamera S3.



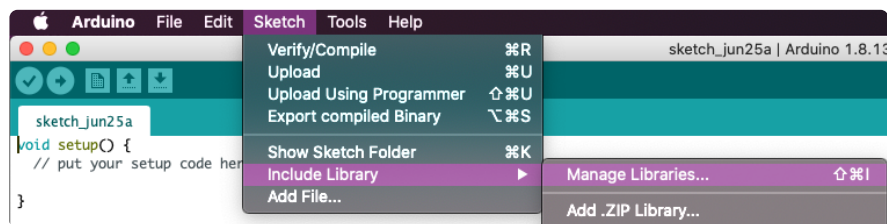
The upload speed can be changed: faster speed makes uploads take less time but sometimes can cause upload issues. **921600** should work fine, but if you're having issues, you can drop down lower.

Arduino MEMENTO Library Installation and Starter Projects

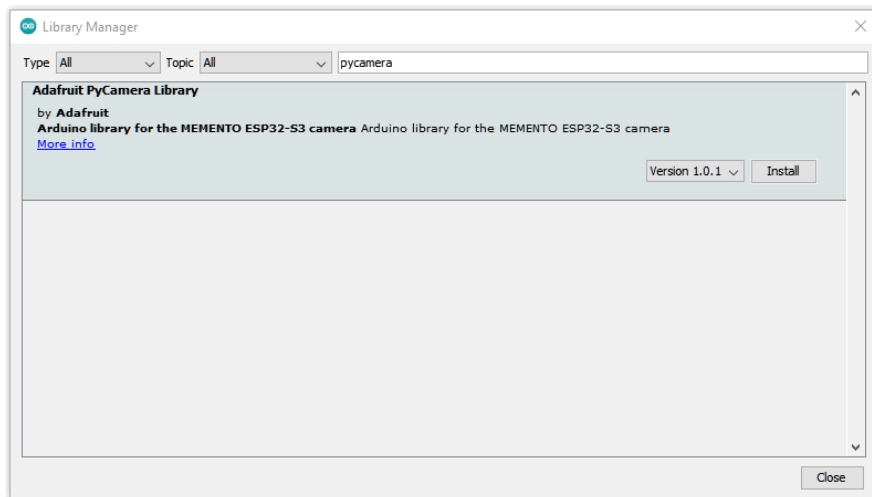
Using the MEMENTO with Arduino involves installing the [Adafruit_PyCamera](https://adafru.it/18e5) (<https://adafru.it/18e5>) library and running the provided example code.

Library Installation

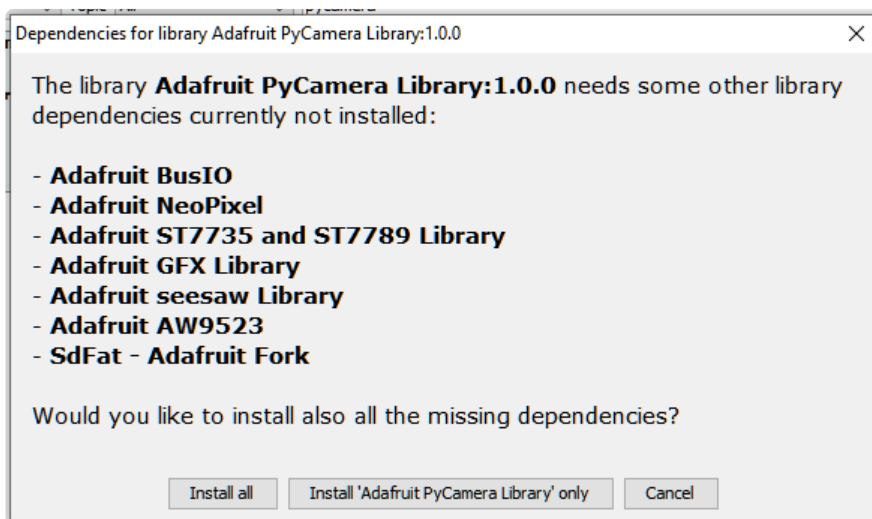
You can install the **Adafruit_PyCamera** library using the Library Manager in the Arduino IDE.



Click the **Manage Libraries ...** menu item, search for **Adafruit_PyCamera**, and select the **Adafruit PyCamera Library**:



If asked about dependencies, click "Install all".



If the "Dependencies" window does not come up, then you already have the dependencies installed.

If the dependencies are already installed, you must make sure you update them through the Arduino Library Manager before loading the example!

The following examples will use this library to access all of the hardware features on the MEMENTO.

PyCamera Library Test

This example is included in the Adafruit PyCamera Arduino library. It is the fully featured Factory Demo that ships on the MEMENTO board. You can either drag and

drop the example UF2 file to the MEMENTO or compile and upload the code with the Arduino IDE.

memento_factory_test.ino.uf2

<https://adafru.it/18eg>

Factory Demo Code

```
// SPDX-FileCopyrightText: 2023 Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include "Adafruit_PyCamera.h"
#include <Arduino.h>

Adafruit_PyCamera pycamera;
framesize_t validSizes[] = {FRAME_SIZE_QVGA, FRAME_SIZE_QVGA, FRAME_SIZE_HVGA,
                             FRAME_SIZE_VGA,  FRAME_SIZE_SVGA, FRAME_SIZE_XGA,
                             FRAME_SIZE_HD,   FRAME_SIZE_SXGA, FRAME_SIZE_UXGA,
                             FRAME_SIZE_QXGA,  FRAME_SIZE_QSXGA};

// A collection of possible ring light colors
uint32_t ringlightcolors_RGBW[] = {0x00000000, 0x00FF0000, 0x00FFFF00,
                                     0x0000FF00, 0x0000FFFF, 0x000000FF,
                                     0x00FF00FF, 0xFF000000};

uint8_t ringlight_i = 0;
uint8_t ringlightBrightness = 100;

#define IRQ 3

void setup() {
  Serial.begin(115200);
  // while (!Serial) yield();
  delay(100);

  if (!pycamera.begin()) {
    Serial.println("Failed to initialize pyCamera interface");
    while (1)
      yield();
  }
  Serial.println("pyCamera hardware initialized!");

  pinMode(IRQ, INPUT_PULLUP);
  attachInterrupt(
    IRQ, [] { Serial.println("IRQ!"); }, FALLING);
}

void loop() {
  static uint8_t loopn = 0;
  pycamera.setNeopixel(pycamera.Wheel(loopn));
  loopn += 8;

  pycamera.readButtons();
  // Serial.printf("Buttons: 0x%08X\n\r", pycamera.readButtons());

  // pycamera.timestamp();
  pycamera.captureFrame();

  // once the frame is captured we can draw onto the framebuffer
  if (pycamera.justPressed(AWEXP_SD_DET)) {
    Serial.println(F("SD Card removed"));
    pycamera.endSD();
    pycamera.fb->setCursor(0, 32);
  }
}
```

```

    pycamera.fb->setTextSize(2);
    pycamera.fb->setTextColor(pycamera.color565(255, 0, 0));
    pycamera.fb->print(F("SD Card removed"));
    delay(200);
}
if (pycamera.justReleased(AWEXP_SD_DET)) {
    Serial.println(F("SD Card inserted!"));
    pycamera.initSD();
    pycamera.fb->setCursor(0, 32);
    pycamera.fb->setTextSize(2);
    pycamera.fb->setTextColor(pycamera.color565(255, 0, 0));
    pycamera.fb->print(F("SD Card inserted"));
    delay(200);
}

float A0_voltage = analogRead(A0) / 4096.0 * 3.3;
if (loopn == 0) {
    Serial.printf("A0 = %0.1f V, Battery = %0.1f V\n\r", A0_voltage,
        pycamera.readBatteryVoltage());
}
pycamera.fb->setCursor(0, 0);
pycamera.fb->setTextSize(2);
pycamera.fb->setTextColor(pycamera.color565(255, 255, 255));
pycamera.fb->print("A0 = ");
pycamera.fb->print(A0_voltage, 1);
pycamera.fb->print("V\nBattery = ");
pycamera.fb->print(pycamera.readBatteryVoltage(), 1);
pycamera.fb->print(" V");

// print the camera frame size
pycamera.fb->setCursor(0, 200);
pycamera.fb->setTextSize(2);
pycamera.fb->setTextColor(pycamera.color565(255, 255, 255));
pycamera.fb->print("Size:");
switch (pycamera.photoSize) {
case FRAMESIZE_QQVGA:
    pycamera.fb->print("160x120");
    break;
case FRAMESIZE_QVGA:
    pycamera.fb->print("320x240");
    break;
case FRAMESIZE_HVGA:
    pycamera.fb->print("480x320");
    break;
case FRAMESIZE_VGA:
    pycamera.fb->print("640x480");
    break;
case FRAMESIZE_SVGA:
    pycamera.fb->print("800x600");
    break;
case FRAMESIZE_XGA:
    pycamera.fb->print("1024x768");
    break;
case FRAMESIZE_HD:
    pycamera.fb->print("1280x720");
    break;
case FRAMESIZE_SXGA:
    pycamera.fb->print("1280x1024");
    break;
case FRAMESIZE_UXGA:
    pycamera.fb->print("1600x1200");
    break;
case FRAMESIZE_QXGA:
    pycamera.fb->print("2048x1536");
    break;
case FRAMESIZE_QSXGA:
    pycamera.fb->print("2560x1920");
    break;
default:

```



```

    pycamera.fb->print("Unknown");
    break;
}

float x_ms2, y_ms2, z_ms2;
if (pycamera.readAccelData(&x_ms2, &y_ms2, &z_ms2)) {
    // Serial.printf("X=%0.2f, Y=%0.2f, Z=%0.2f\n\r", x_ms2, y_ms2, z_ms2);
    pycamera.fb->setCursor(0, 220);
    pycamera.fb->setTextSize(2);
    pycamera.fb->setTextColor(pycamera.color565(255, 255, 255));
    pycamera.fb->print("3D: ");
    pycamera.fb->print(x_ms2, 1);
    pycamera.fb->print(", ");
    pycamera.fb->print(y_ms2, 1);
    pycamera.fb->print(", ");
    pycamera.fb->print(z_ms2, 1);
}

pycamera.blitFrame();

if (pycamera.justPressed(AWEXP_BUTTON_UP)) {
    Serial.println("Up!");
    for (int i = 0; i < sizeof(validSizes) / sizeof(framesize_t) - 1; ++i) {
        if (pycamera.photoSize == validSizes[i]) {
            pycamera.photoSize = validSizes[i + 1];
            break;
        }
    }
}
if (pycamera.justPressed(AWEXP_BUTTON_DOWN)) {
    Serial.println("Down!");
    for (int i = sizeof(validSizes) / sizeof(framesize_t) - 1; i > 0; --i) {
        if (pycamera.photoSize == validSizes[i]) {
            pycamera.photoSize = validSizes[i - 1];
            break;
        }
    }
}

if (pycamera.justPressed(AWEXP_BUTTON_RIGHT)) {
    pycamera.specialEffect = (pycamera.specialEffect + 1) % 7;
    pycamera.setSpecialEffect(pycamera.specialEffect);
    Serial.printf("set effect: %d\n\r", pycamera.specialEffect);
}
if (pycamera.justPressed(AWEXP_BUTTON_LEFT)) {
    pycamera.specialEffect = (pycamera.specialEffect + 6) % 7;
    pycamera.setSpecialEffect(pycamera.specialEffect);
    Serial.printf("set effect: %d\n\r", pycamera.specialEffect);
}

if (pycamera.justPressed(AWEXP_BUTTON_OK)) {
    // iterate through all the ring light colors
    ringlight_i =
        (ringlight_i + 1) % (sizeof(ringlightcolors_RGBW) / sizeof(uint32_t));
    pycamera.setRing(ringlightcolors_RGBW[ringlight_i]);
    Serial.printf("set ringlight: 0x%08X\n\r",
        (unsigned int)ringlightcolors_RGBW[ringlight_i]);
}
if (pycamera.justPressed(AWEXP_BUTTON_SEL)) {
    // iterate through brightness levels, incrementing 25 at a time
    if (ringlightBrightness >= 250)
        ringlightBrightness = 0;
    else
        ringlightBrightness += 50;
    pycamera.ring.setBrightness(ringlightBrightness);
    pycamera.setRing(ringlightcolors_RGBW[ringlight_i]);
    Serial.printf("set ringlight brightness: %d\n\r", ringlightBrightness);
}

```

```

if (pycamera.justPressed(SHUTTER_BUTTON)) {
  Serial.println("Snap!");
  if (pycamera.takePhoto("IMAGE", pycamera.photoSize)) {
    pycamera.fb->setCursor(120, 100);
    pycamera.fb->setTextSize(2);
    pycamera.fb->setTextColor(pycamera.color565(255, 255, 255));
    pycamera.fb->print("Snap!");
    pycamera.speaker_tone(100, 50); // tone1 - B5
    // pycamera.blitFrame();
  }
}
delay(100);
}

```

After uploading the sketch, you will need to press the reset button on MEMENTO to run the new code.

Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 115200 baud. On the MEMENTO display, you'll see a preview of what the camera module is seeing. You'll be able to change the resolution and filter for the camera. If you insert a microSD card, you'll be able to take pictures using the **BOOT** button.



Basic Camera Example

This example is a basic camera script for the MEMENTO. With it you can insert a microSD card and take a photo at 800x640 resolution by pressing the **BOOT** button.

You can either drag and drop the example UF2 file to the MEMENTO or compile and upload the code with the Arduino IDE.

espressif.esp32.adafruit_camera_esp
Arduino_Basic_Camera.ino.uf2

<https://adafru.it/18eh>

```
// SPDX-FileCopyrightText: 2023 Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include "Adafruit_PyCamera.h"
#include <Arduino.h>

Adafruit_PyCamera pycamera;
framesize_t validSizes[] = {FRAMESIZE_QQVGA, FRAMESIZE_QVGA, FRAMESIZE_HVGA,
                             FRAMESIZE_VGA,   FRAMESIZE_SVGA, FRAMESIZE_XGA,
                             FRAMESIZE_HD,    FRAMESIZE_SXGA, FRAMESIZE_UXGA,
                             FRAMESIZE_QXGA,   FRAMESIZE_QSXGA};

void setup() {
  Serial.begin(115200);
  // while (!Serial) yield();
  // delay(1000);

  // Serial.setDebugOutput(true);
  Serial.println("PyCamera Basic Example");
  if (!pycamera.begin()) {
    Serial.println("Failed to initialize PyCamera interface");
    while (1)
      yield();
  }
  Serial.println("PyCamera hardware initialized!");

  pycamera.photoSize = FRAMESIZE_SVGA;
}

void loop() {
  pycamera.readButtons();
  // Serial.printf("Buttons: 0x%08X\n\r", pycamera.readButtons());

  // pycamera.timestamp();
  pycamera.captureFrame();

  // once the frame is captured we can draw onto the framebuffer
  if (pycamera.justPressed(AWEXP_SD_DET)) {
    Serial.println(F("SD Card removed"));
    pycamera.endSD();
    pycamera.fb->setCursor(0, 32);
    pycamera.fb->setTextSize(2);
    pycamera.fb->setTextColor(pycamera.color565(255, 0, 0));
    pycamera.fb->print(F("SD Card removed"));
    delay(200);
  }
  if (pycamera.justReleased(AWEXP_SD_DET)) {
    Serial.println(F("SD Card inserted!"));
    pycamera.initSD();
    pycamera.fb->setCursor(0, 32);
    pycamera.fb->setTextSize(2);
    pycamera.fb->setTextColor(pycamera.color565(255, 0, 0));
    pycamera.fb->print(F("SD Card inserted"));
    delay(200);
  }
}
```

```

pycamera.blitFrame();

if (pycamera.justPressed(SHUTTER_BUTTON)) {
  Serial.println("Snap!");
  if (pycamera.takePhoto("IMAGE", pycamera.photoSize)) {
    pycamera.fb->setCursor(120, 100);
    pycamera.fb->setTextSize(2);
    pycamera.fb->setTextColor(pycamera.color565(255, 255, 255));
    pycamera.fb->print("Snap!");
    pycamera.speaker_tone(100, 50); // tone1 - B5
    // pycamera.blitFrame();
  }
}

delay(100);
}

```

After uploading the sketch, you will need to press the reset button on MEMENTO to run the new code.

Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 115200 baud. On the MEMENTO display, you'll see a preview of what the camera module is seeing. If you insert a microSD card, you'll be able to take pictures using the **BOOT** button. This is a basic example that you can build on with your own code.

Usage with PlatformIO

This page is for advanced users only. For beginner users, we recommend using either CircuitPython or Arduino IDE.

The PyCamera Arduino library brings in a considerable number of dependencies and takes a looong time to compile using Arduino IDE.

If you're working on an advanced MEMENTO project that requires additional dependencies, such as a component from the ESP-IDF component registry, you'll want to compile the PyCamera Arduino project using PlatformIO rather than Arduino IDE.

How fast is it? Very! Using PlatformIO compiles the PyCamera Library Test example in ~4.10 seconds.

Installation

Follow this page's instructions to install PlatformIO and Visual Studio Code (the IDE of choice for using PlatformIO).

Download PlatformIO

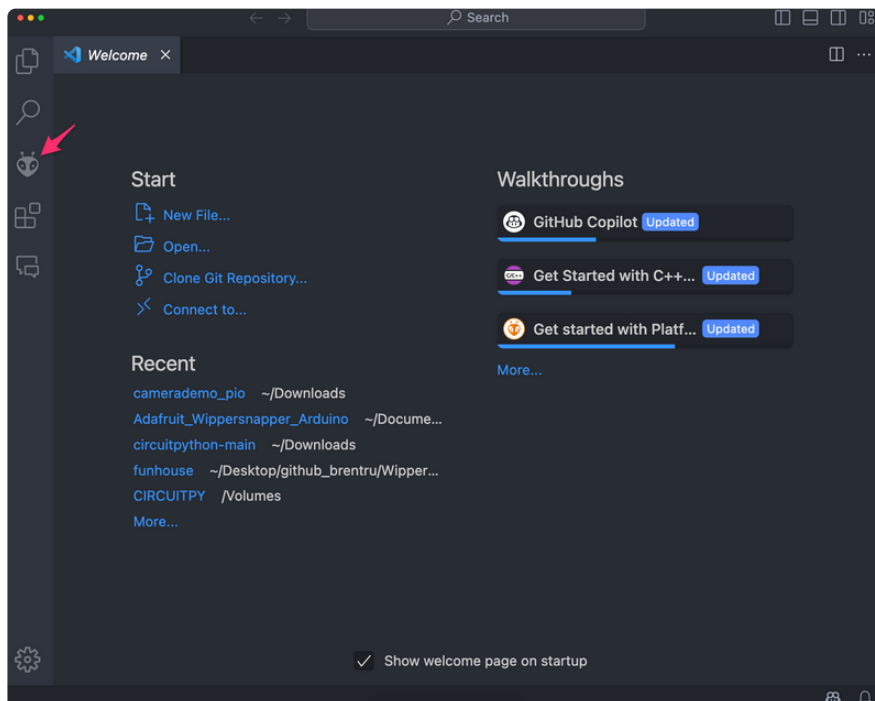
<https://adafru.it/19cu>

The ZIP file below includes a pre-configured workspace for using PlatformIO.
Download and unzip this file. Then, save it somewhere safe, like your desktop.

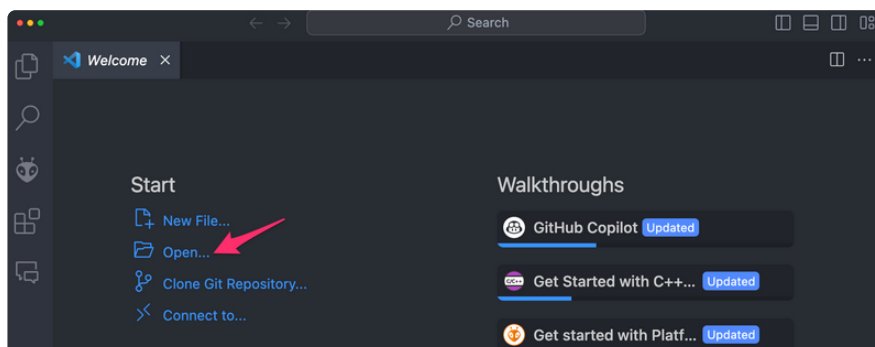
memento_platformio.zip

<https://adafru.it/19cv>

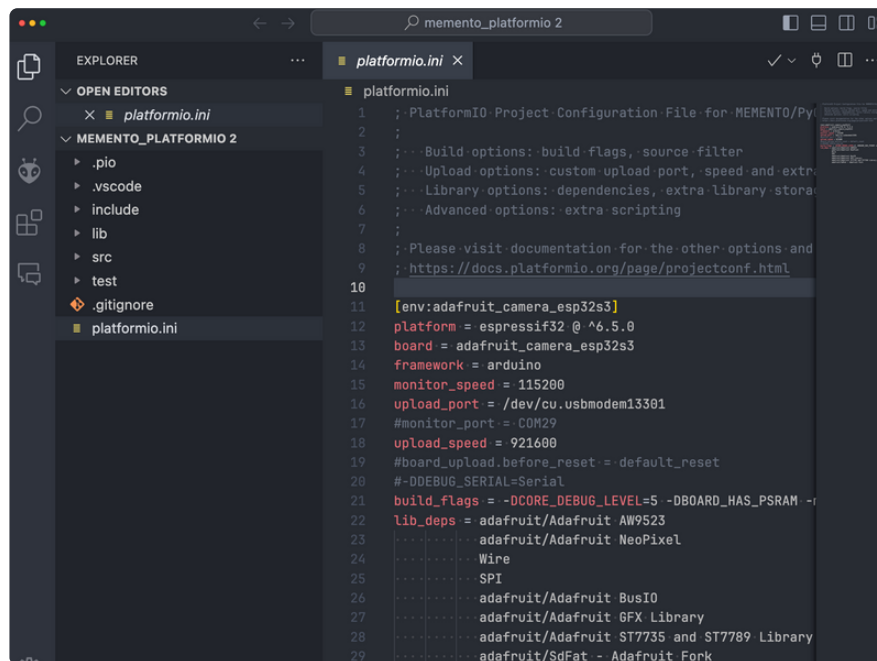
Open Visual Studio Code (VSCode). To ensure you have installed the PlatformIO extension properly, **look for the alien symbol in your VSCode sidebar.**



Underneath Start, click Open...



Navigate to the folder created when you unzipped the zip file. Then, Click Open to open the workspace.



A large amount of configuration files and directories will appear in your VSCode instance.

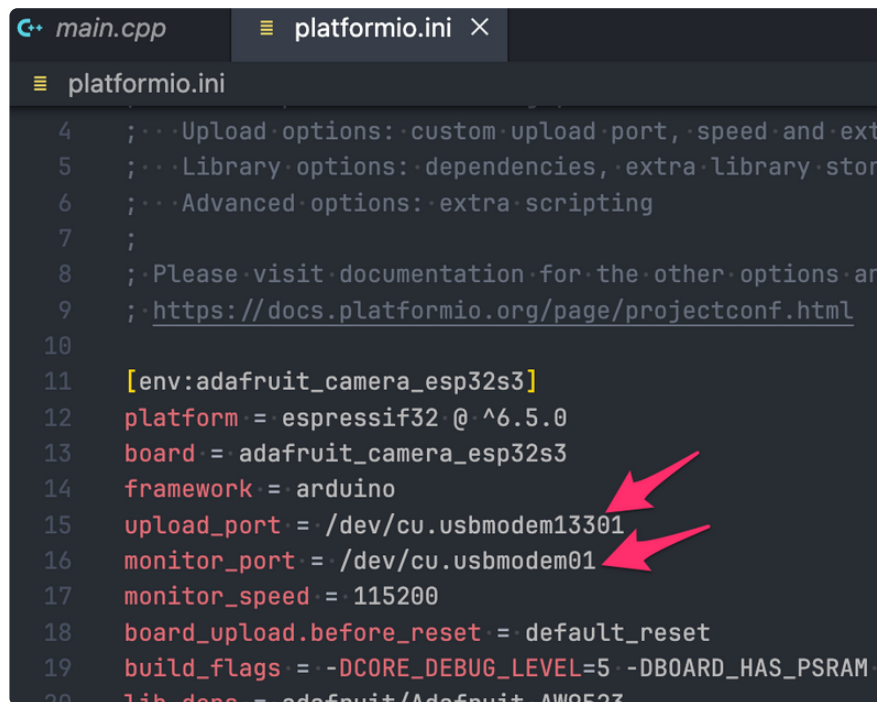
To compile this code, we are only going to discuss the following files and directories:

- **platformio.ini** - This is the project configuration file used to build the demo code. More [documentation about this file is located here \(https://adafru.it/19cw\)](https://adafru.it/19cw).
- **lib** directory - This directory is intended for project-specific (private) libraries. PlatformIO will compile them to static libraries and link them into executable files.
 - For our project, the specific library within this directory is the [Adafruit_PyCamera \(https://adafru.it/19cx\)](https://adafru.it/19cx) library.
- **src** directory - The directory where the source code of the project is located, **main.cpp**. When code in PlatformIO is built or uploaded, files from this directory are used.

PyCamera Library Test

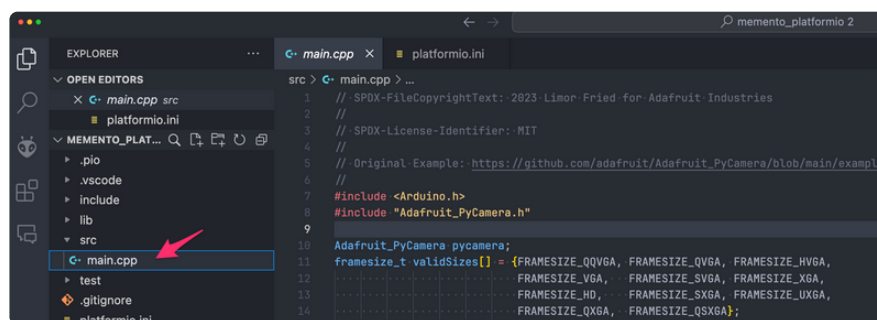
Before the code is compiled, you'll need to make two changes to the platformio.ini file:

- Change `upload_port` to reflect the MEMENTO's upload port.
 - Don't know the desired port? We have steps to find them for [Windows](https://adafru.it/19cy) (<https://adafru.it/19cy>), [MacOS](https://adafru.it/19cz) (<https://adafru.it/19cz>), and [Linux](https://adafru.it/19cA) (<https://adafru.it/19cA>).
- The `monitor_port` is different from the `upload_port`, and will only appear on your computer when you've uploaded the test code. For now, leave this alone.
 - After uploading the test code, change `monitor_port` to reflect the MEMENTO's monitor/serial port.



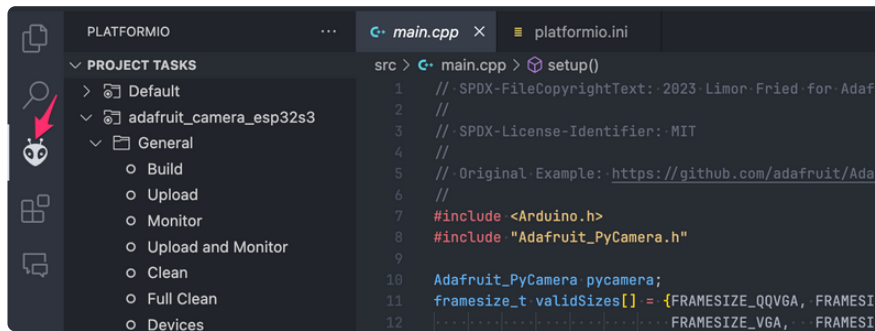
```
main.cpp platformio.ini X
platformio.ini
4 ;...Upload options: custom upload port, speed and extra
5 ;...Library options: dependencies, extra library storage
6 ;...Advanced options: extra scripting
7 ;
8 ;Please visit documentation for the other options and
9 ;https://docs.platformio.org/page/projectconf.html
10
11 [env:adafruit_camera_esp32s3]
12 platform = espressif32 @ ^6.5.0
13 board = adafruit_camera_esp32s3
14 framework = arduino
15 upload_port = /dev/cu.usbmodem13301
16 monitor_port = /dev/cu.usbmodem01
17 monitor_speed = 115200
18 board_upload.before_reset = default_reset
19 build_flags = -DCORE_DEBUG_LEVEL=5 -DBOARD_HAS_PSRAM
20 lib_deps = adafruit/Adafruit_AW9523
```

Navigate to `src/main.cpp` to open the example code.

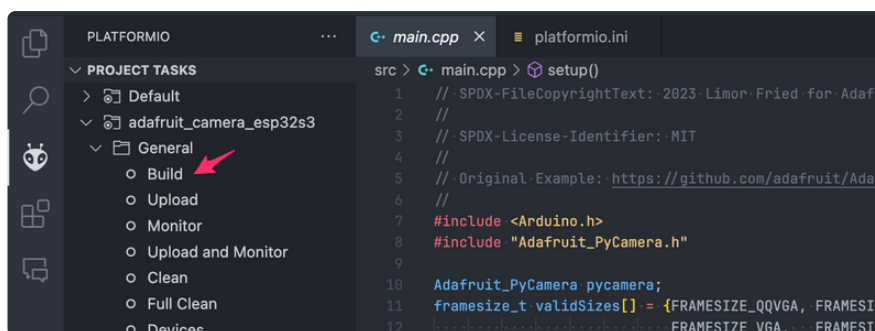


```
EXPLORER
main.cpp X platformio.ini
src > C: main.cpp > ...
1 // SPDX-FileCopyrightText: 2023 Limor Fried for Adafruit Industries
2 //
3 // SPDX-License-Identifier: MIT
4 //
5 // Original Example: https://github.com/adafruit/Adafruit_PyCamera/blob/main/examples
6 //
7 #include <Arduino.h>
8 #include "Adafruit_PyCamera.h"
9
10 Adafruit_PyCamera pycamera;
11 framesize_t validSizes[] = {FRAMESIZE_QQVGA, FRAMESIZE_QVGA, FRAMESIZE_HVGA,
12 FRAMESIZE_VGA, FRAMESIZE_SVGA, FRAMESIZE_XGA,
13 FRAMESIZE_HD, FRAMESIZE_SXGA, FRAMESIZE_UXGA,
14 FRAMESIZE_QXGA, FRAMESIZE_QSXGA};
```

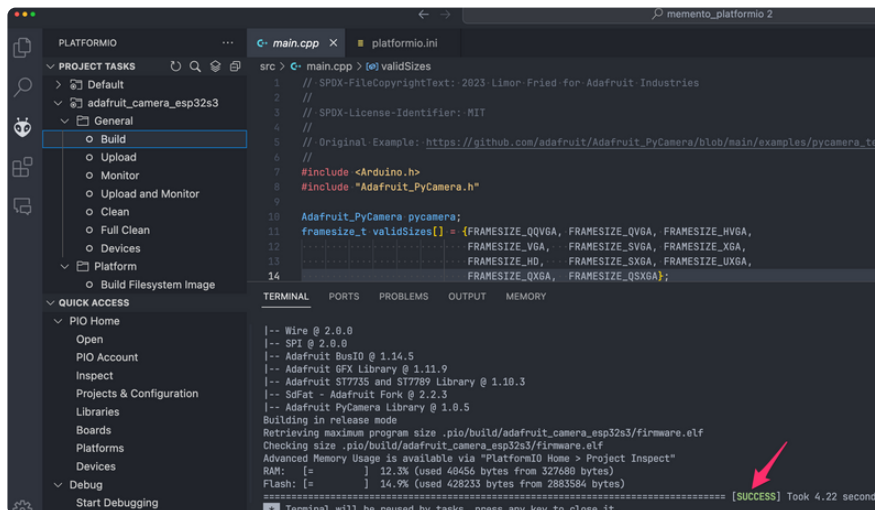

With this file open, click the **Alien symbol** on the **VSCode sidebar** to open the PlatformIO Project Explorer.



Underneath PlatformIO's Project Tasks, click **Build**.

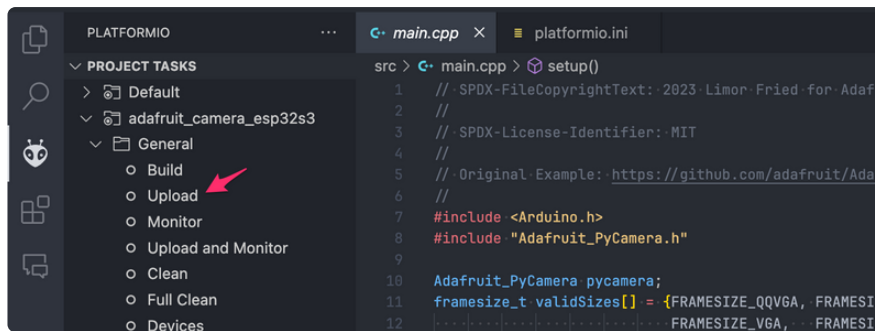


Once the build task is completed, the terminal will show **SUCCESS** along with the time it took to compile the project.

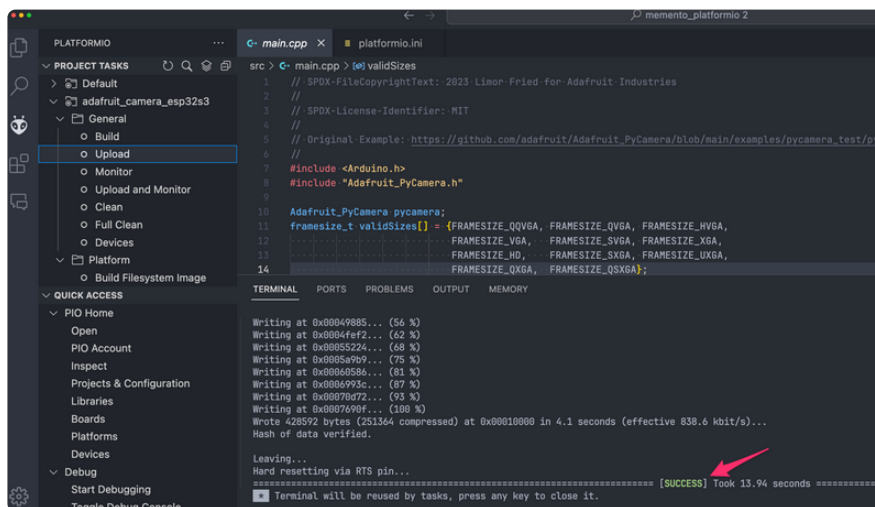


Before uploading this project to the board, [put the board into ROM Bootloader Mode \(https://adafru.it/19cB\)](https://adafru.it/19cB).

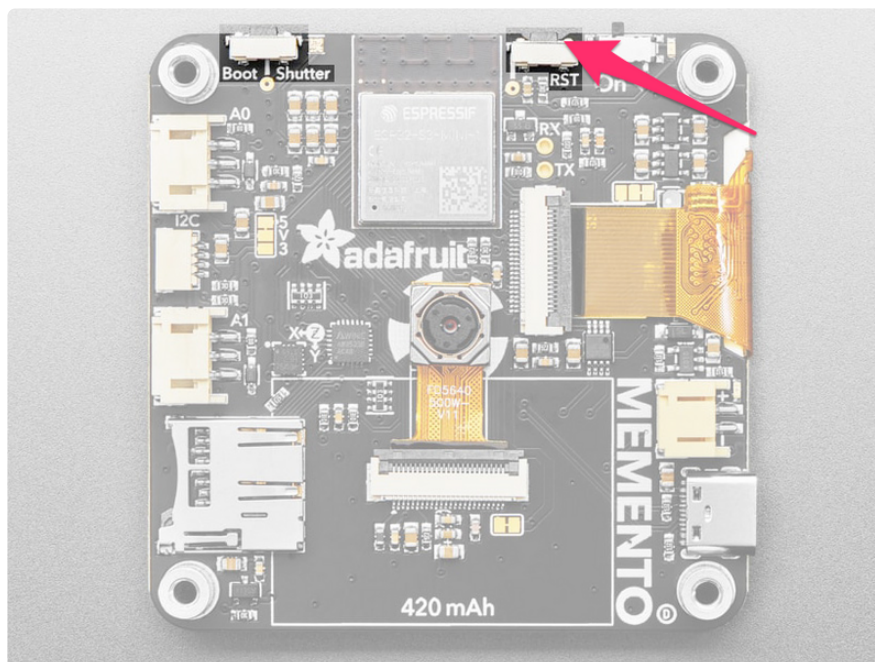
From the PlatformIO Project Tasks menu, click **Upload**.



Once the upload completes, the terminal should look like the following screenshot and show **SUCCESS**.



Press the RST (Reset) button on the MEMENTO to run the uploaded code.



After the board resets, you'll see a preview of what the camera module is seeing on the MEMENTO display. You'll be able to change the resolution and filter for the

camera. If you insert a microSD card, you'll be able to take pictures using the BOOT button.



Factory Reset

The MEMENTO board ships running a PyCamera demo. It's lovely, but you probably had other plans for the board. As you start working with your board, you may want to return to the original code to begin again, or you may find your board gets into a bad state. Either way, this page has you covered.

You're probably used to seeing the **CAMERABOOT** drive when loading CircuitPython or Arduino. The **CAMERABOOT** drive is part of the UF2 bootloader, and allows you to drag and drop files, such as CircuitPython. However, on the ESP32-S3 the UF2 bootloader can become damaged.

Factory Reset Firmware UF2

If you have a bootloader still installed - which means you can double-click to get the **CAMERABOOT** drive to appear, then you can simply drag this UF2 file over to the **CAMERABOOT** drive.

To enter bootloader mode, plug in the board into a USB cable with data/sync capability. Press the reset button once, wait till the RGB LED turns purple, then press the reset button again. Then drag this file over:

**Adafruit MEMENTO Factory Reset
UF2**

Your board is now back to its factory-shipped state! You can now begin again with your plans for your board.

Factory Reset and Bootloader Repair

What if you tried double-tapping the reset button, and you still can't get into the UF2 bootloader? Whether your board shipped without the UF2 bootloader, or something damaged it, this section has you covered.

There is no bootloader protection for the UF2 bootloader. That means it is possible to erase or damage the UF2 bootloader, especially if you upload an Arduino sketch to an ESP32-S2/S3 board that doesn't "know" there's a bootloader it should not overwrite!

It turns out, however, the ESP32-S2/S3 comes with a second bootloader: the ROM bootloader. Thanks to the ROM bootloader, you don't have to worry about damaging the UF2 bootloader. 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 from the ROM bootloader.

Completing a factory reset will erase your board's firmware which is also used for storing CircuitPython/Arduino/Files! Be sure to back up your data first.

There are two ways to do a factory reset and bootloader repair. The first is using WebSerial through a Chromium-based browser, and the second is using `esptool` via command line. **We highly recommend using WebSerial through Chrome/Chromium.**

The next section walks you through the prerequisite steps needed for both methods.

Download .bin and Enter Bootloader

Step 1. Download the factory-reset-and-bootloader.bin file

Save the following file wherever is convenient for you. You will need to access it from the WebSerial ESPTool.

Note that this file is approximately 3MB. This is not because the bootloader is 3MB, it is because the bootloader is near the end of the available flash. Most of the file is empty but its easier to program if you use a combined file.

Click to download MEMENTO
Factory Reset .BIN

<https://adafru.it/18ej>

Step 2. Enter ROM bootloader mode

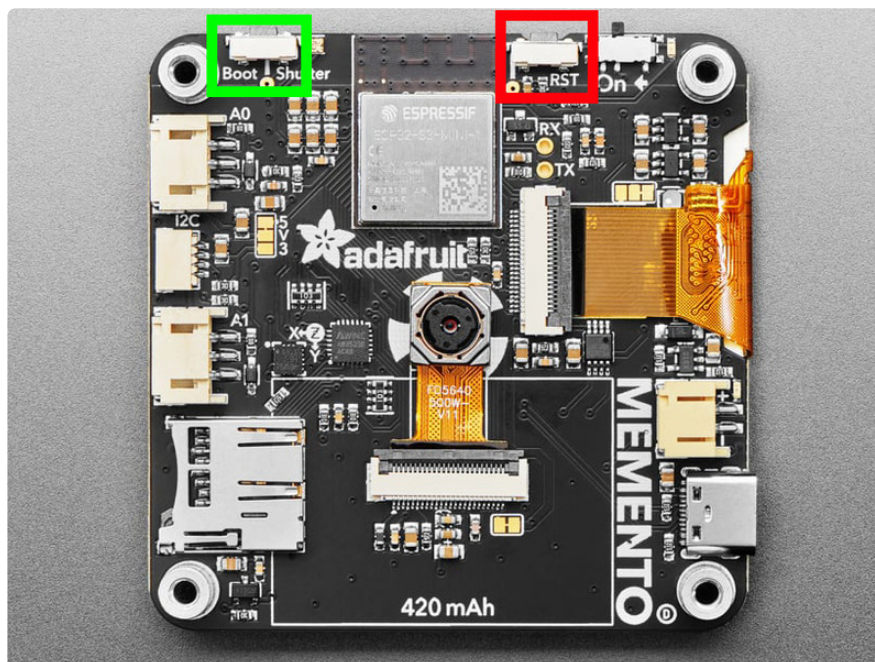
Entering the ROM bootloader is easy. Complete the following steps.

Before you start, make sure your ESP32-S2/S3 is plugged into USB port to your computer using a data/sync cable. Charge-only cables will not work!

Turn on the On/Off switch - check that you see the green power light on so you know the board is powered, a prerequisite!

To enter the bootloader:

1. **Press and hold the BOOT/DFU button down.** Don't let go of it yet!
2. **Press and release the Reset button.** You should still have the BOOT/DFU button pressed while you do this.
3. **Now you can release the BOOT/DFU button.**



No USB drive will appear when you've entered the ROM bootloader. This is normal!

Now that you've downloaded the .bin file and entered the bootloader, you're ready to continue with the factory reset and bootloader repair process. The next two sections walk you through using WebSerial and `esptool`.

The WebSerial ESPTool Method

We highly recommend using WebSerial ESPTool method to perform a factory reset and bootloader repair. However, if you'd rather use esptool via command line, you can skip this section.

This method uses the WebSerial ESPTool through Chrome or a Chromium-based browser. The WebSerial ESPTool was designed to be a web-capable option for programming ESP32-S2/S3 boards. It allows you to erase the contents of the microcontroller and program up to four files at different offsets.

You will have to use a Chromium browser (like Chrome, Opera, Edge...) for this to work, Safari and Firefox, etc. are not supported because we need Web Serial and only Chromium is supporting it to the level needed.

Follow the steps to complete the factory reset.

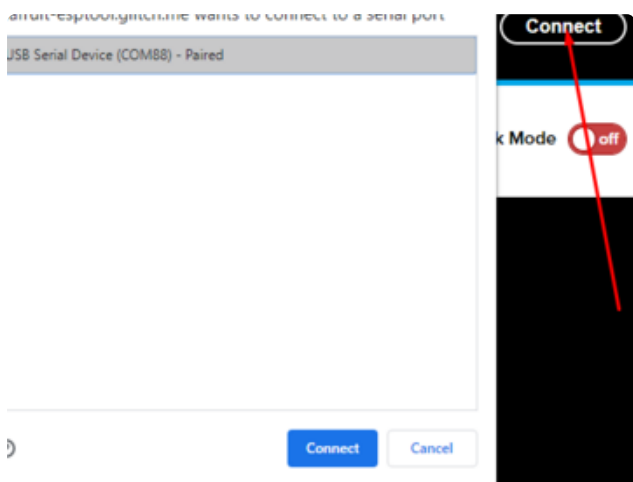
If you're using Chrome 88 or older, see the Older Versions of Chrome section at the end of this page for instructions on enabling Web Serial.

Connect

You should have plugged in **only the ESP32-S2/S3 that you intend to flash**. That way there's no confusion in picking the proper port when it's time!



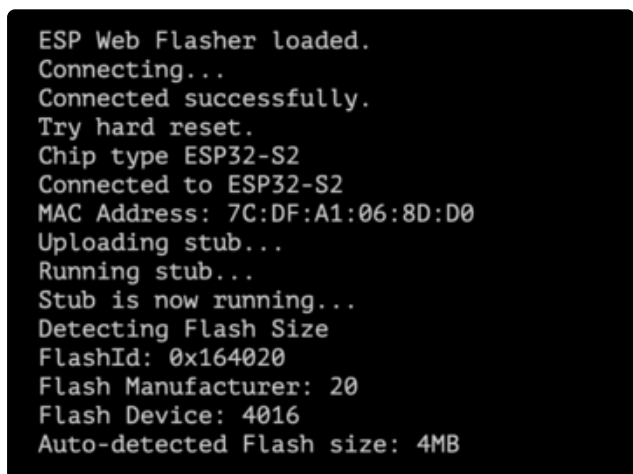
In the **Chrome browser** visit https://adafruit.github.io/Adafruit_WebSerial_ESPTool/ (<https://adafru.it/PMB>). You should see something like the image shown.



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.

Remember, you should remove all other **USB devices** so only the **ESP32-S2/S3** board is attached, that way there's no confusion over multiple ports!

On some systems, such as MacOS, there may be additional system ports that appear in the list.



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 along with other information that was detected.

Offset: 0x

Offset: 0x

Offset: 0x

Offset: 0x

```
Adafruit WebSerial ESPTool loaded.  
Connecting...  
Connected successfully.  
Timed out after 100 milliseconds  
Timed out after 100 milliseconds  
Timed out after 100 milliseconds  
Timed out after 100 milliseconds  
Timed out after 100 milliseconds  
Timed out after 100 milliseconds  
Timed out after 100 milliseconds  
Timed out after 100 milliseconds  
Timed out after 100 milliseconds  
Timed out after 100 milliseconds  
Changed baud rate to 921600  
Connected to ESP32-S2  
MAC Address: E6:85:6D:92:AA:32
```

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

Erase the Contents

This will erase everything on your board! If you have access, and wish to keep any code, now is the time to ensure you've backed up everything.



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.

```
Erasing flash memory. Please wait...  
Finished. Took 15899ms to erase.
```

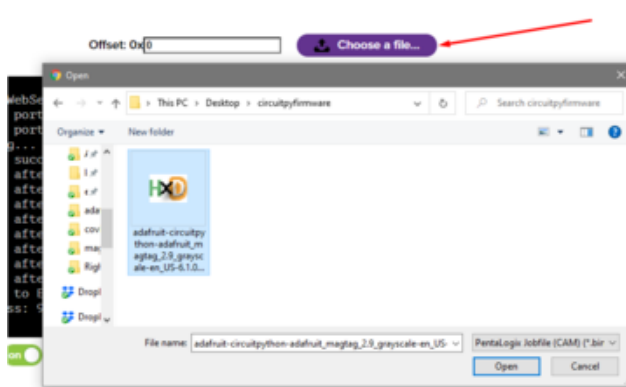
You'll see "Erasing flash memory. Please wait..." This will eventually be followed by "Finished." and the amount of time it took to erase.

Do not disconnect! Immediately continue on to programming the ESP32-S2/S3.

Do not disconnect after erasing! Immediately continue on to the next step!

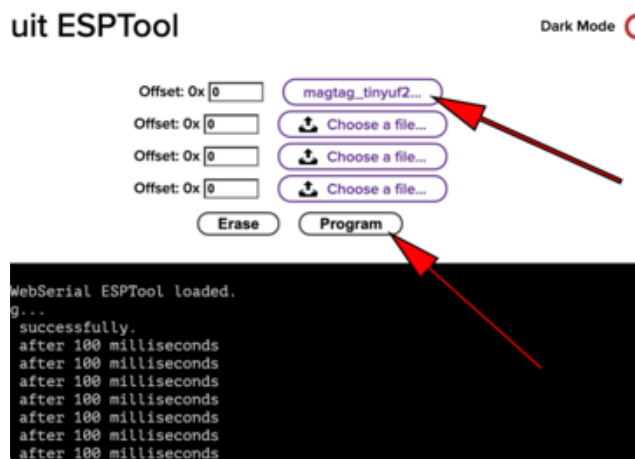
Program the ESP32-S2/S3

Programming the microcontroller can be done with up to four files at different locations, but with the board-specific **factory-reset.bin** file, which you should have downloaded under **Step 1** on this page, you only need to use one file.



Click on the first **Choose a file....** (The tool will only attempt to program buttons with a file and a unique location.) Then, select the ***-factory-reset.bin** file you downloaded in Step 1 that matches your board.

Verify that the **Offset** box next to the file location you used is (0x) 0.



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



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

Once completed, you can skip down to the section titled Reset the Board.

The `esptool` Method (for advanced users)

If you used WebSerial ESPTool, you do not need to complete the steps in this section!

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
```

Make sure you are running esptool v3.0 or higher, which adds ESP32-S2/S3 support.

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,1.9V,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_region,version,get_security_info}
               ...
```

Connect

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/S3.

```
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'.
```

Erase the Flash

Before programming the board, it is a good idea to erase the flash. Run the following command.

```
esptool.py erase_flash
```

You must be connected (by running the command in the previous section) for this command to work as shown.

```
> esptool.py erase_flash
esptool.py v4.7-dev
Found 2 serial ports
Serial port /dev/cu.usbmodem2121101
Connecting...
Detecting chip type... ESP32-S3
Chip is ESP32-S3 (QFN56) (revision v0.1)
Features: WiFi, BLE, Embedded PSRAM 8MB (AP_3v3)
Crystal is 40MHz
MAC: 34:85:18:9b:f1:7c
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 2.3s
Hard resetting via RTS pin...
```

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.

Adjust the bootloader filename accordingly if it differs from tinyuf2_combo.bin.

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'.
```

Once completed, you can continue to the next section.

Reset the board

Now that you've reprogrammed the board, you need to reset it to continue. Click the reset button to launch the new firmware.

The MEMENTO will launch the PyCamera demo. You'll be able to preview the camera view on the TFT, change modes with the buttons and take photos by pressing the **BOOT** button that save to the microSD card.

You've successfully returned your board to a factory reset state!

Older Versions of Chrome

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

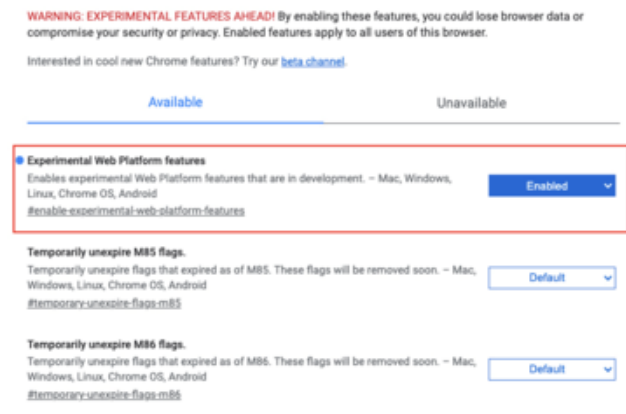
We suggest updating to Chrome 89 or newer, as Web Serial is enabled by default.

If you must continue using an older version of Chrome, follow these steps to enable Web Serial.

Sorry, **Web Serial** is not supported on this device, make sure you're running Chrome 78 or later and have enabled the `enable-experimental-web-platform-features` flag in `chrome://flags`

If you receive an error like the one shown when you visit the WebSerial ESPTool site, you're likely running an older version of Chrome.

You must be using Chrome 78 or later to use Web Serial.



To enable Web Serial in Chrome versions 78 through 88:

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

The Flash an Arduino Sketch Method

This section outlines flashing an Arduino sketch onto your ESP32-S2/S3 board, which automatically installs the UF2 bootloader as well.

Arduino IDE Setup

If you don't already have the Arduino IDE installed, the first thing you will need to do is to download the latest release of the Arduino IDE. ESP32-S2/S3 requires **version 1.8** or higher. Click the link to download the latest.

Arduino IDE Download

<https://adafru.it/Pd5>

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 > Preferences** menu in Windows or Linux, or the **Arduino > Preferences** menu on OS X.

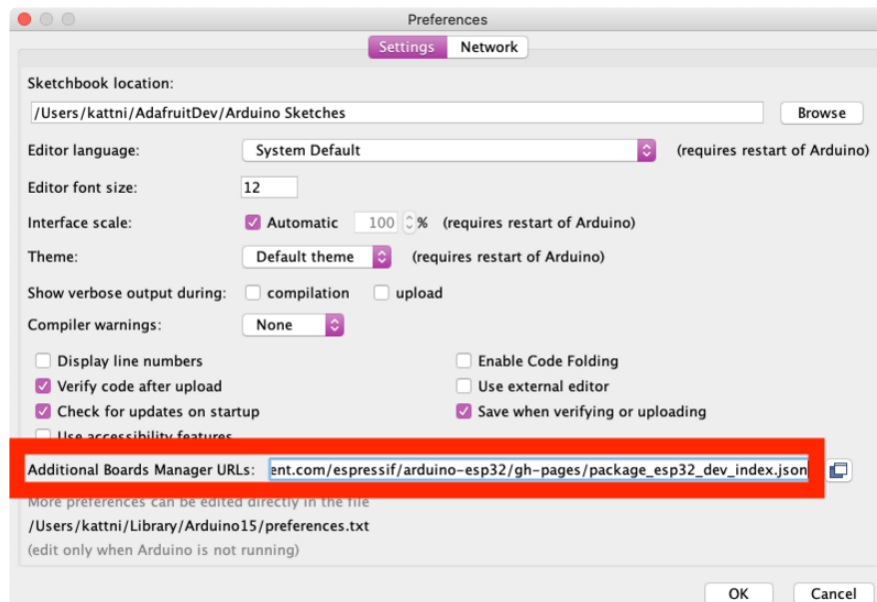
The **Preferences** window will open.

In the **Additional Boards Manager URLs** field, you'll want to add a new URL. The list of URLs is comma separated, and you will only have to add each URL once. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

Copy the following URL.

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

Add the URL to the the **Additional Boards Manager URLs** field (highlighted in red below).



Click **OK** to save and close **Preferences**.

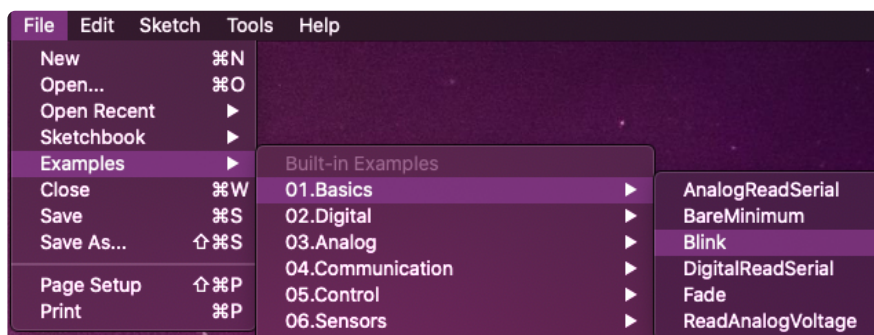
In the **Tools > Boards** menu you should see the **ESP32 Arduino** menu. In the expanded menu, it should contain the ESP32 boards along with all the latest ESP32-S2 boards.

Now that your IDE is setup, you can continue on to loading the sketch.

Load the Blink Sketch

In the **Tools > Boards** menu you should see the **ESP32 Arduino** menu. In the expanded menu, look for the menu option for the **Adafruit pyCamera S3**, and click on it to choose it.

Open the Blink sketch by clicking through **File > Examples > 01.Basics > Blink**.



Once open, click Upload from the sketch window.



Once successfully uploaded, the little red LED will begin blinking once every second. At that point, you can now enter the bootloader.

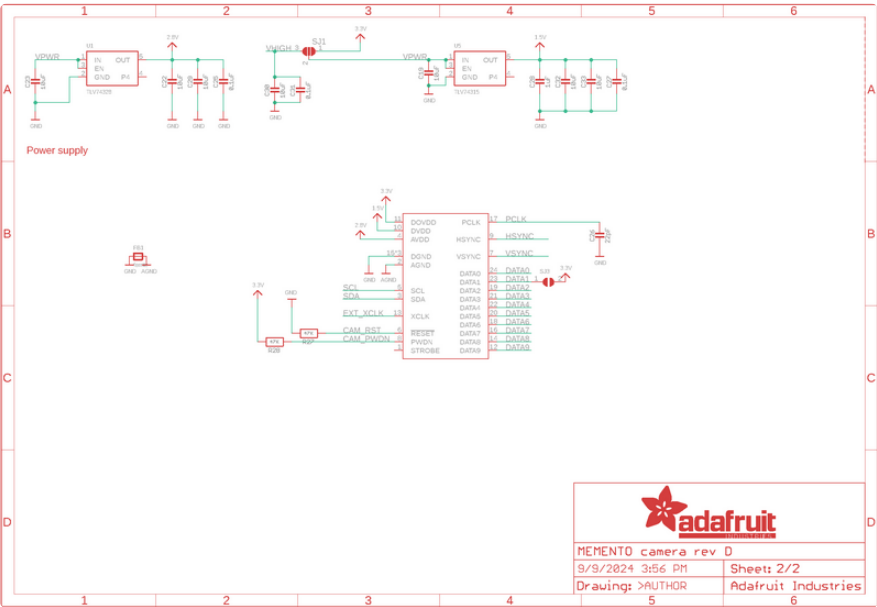
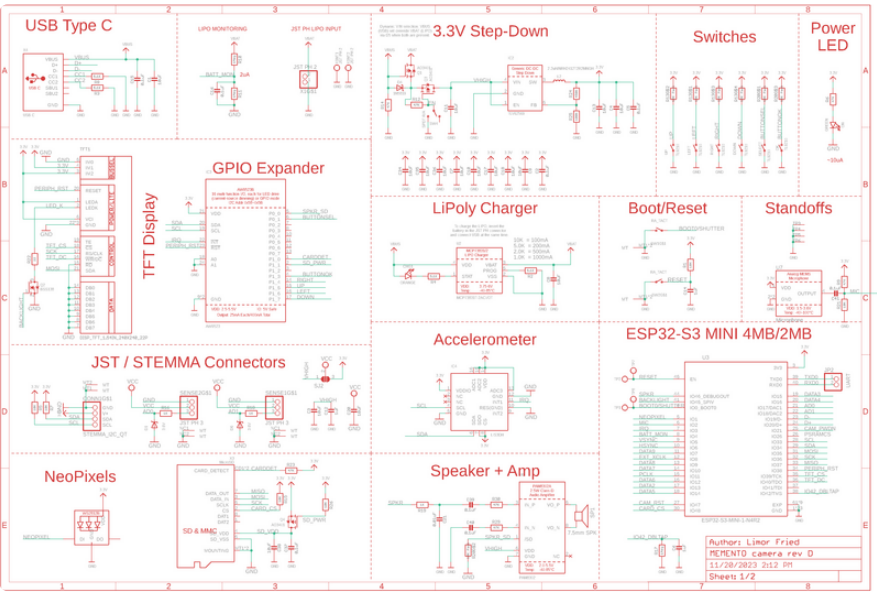
The MEMENTO does not have a red LED, but the NeoPixel pin is defined as LED_BUILTIN to be compatible. As a result, you'll see the NeoPixel blink white after loading this sketch.

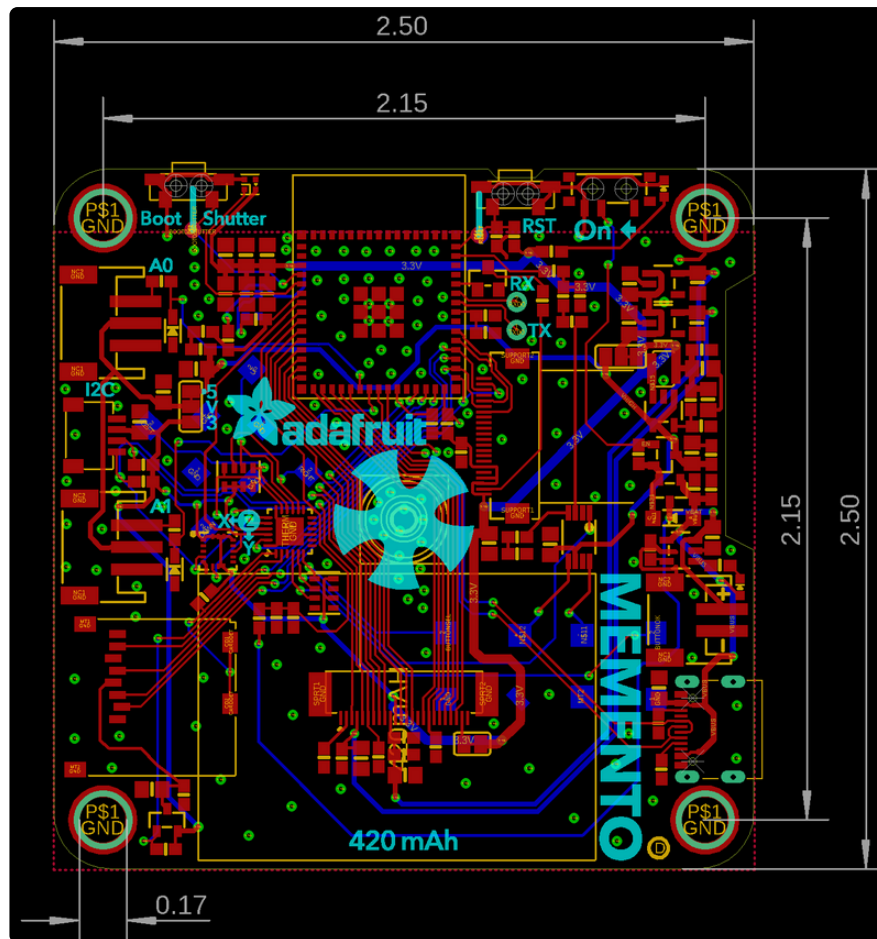
Downloads

Files

- [ESP32-S3 product page with resources \(https://adafru.it/ZAS\)](https://adafru.it/ZAS)
- [ESP32-S3 datasheet \(https://adafru.it/18ek\)](https://adafru.it/18ek)
- [ESP32-S3 Technical Reference \(https://adafru.it/18el\)](https://adafru.it/18el)
- [OV5640 Datasheet \(https://adafru.it/18em\)](https://adafru.it/18em)
- [OV5640 Register Datasheet \(https://adafru.it/18en\)](https://adafru.it/18en)
- [OV5640 Firmware User Guide \(https://adafru.it/18eo\)](https://adafru.it/18eo)
- [EagleCAD PCB files on GitHub \(https://adafru.it/18ep\)](https://adafru.it/18ep)
- [3D Models on GitHub \(https://adafru.it/18eq\)](https://adafru.it/18eq)
- [Fritzing object in the Adafruit Fritzing Library \(https://adafru.it/18er\)](https://adafru.it/18er)
- [Factory Test Firmware Code \(https://adafru.it/19gC\)](https://adafru.it/19gC)

Schematic and Fab Print





3D Model

