Adafruit MacroPad RP2040

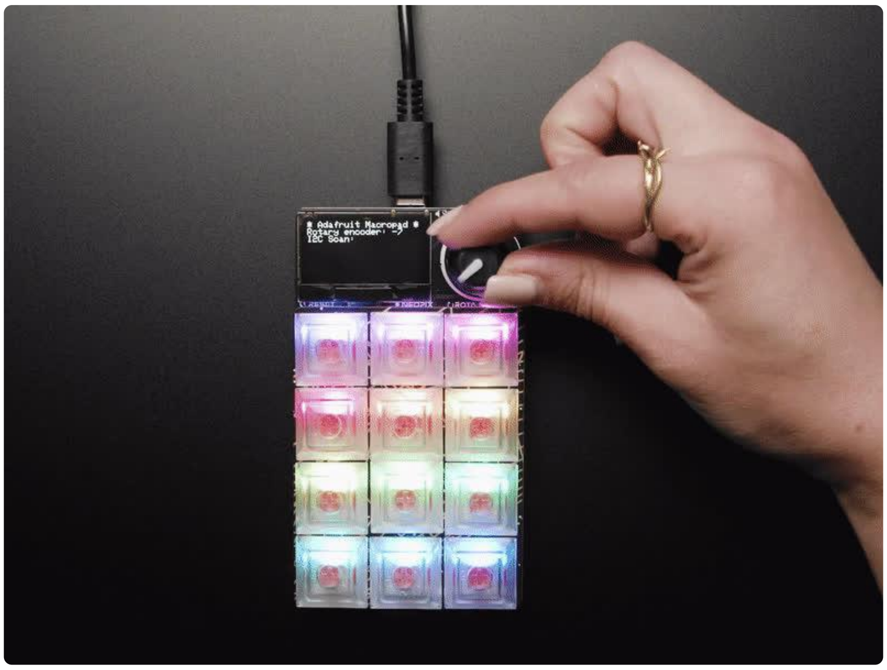
Created by Kattni Rembor

https://learn.adafruit.com/adafruit-macropad-rp2040

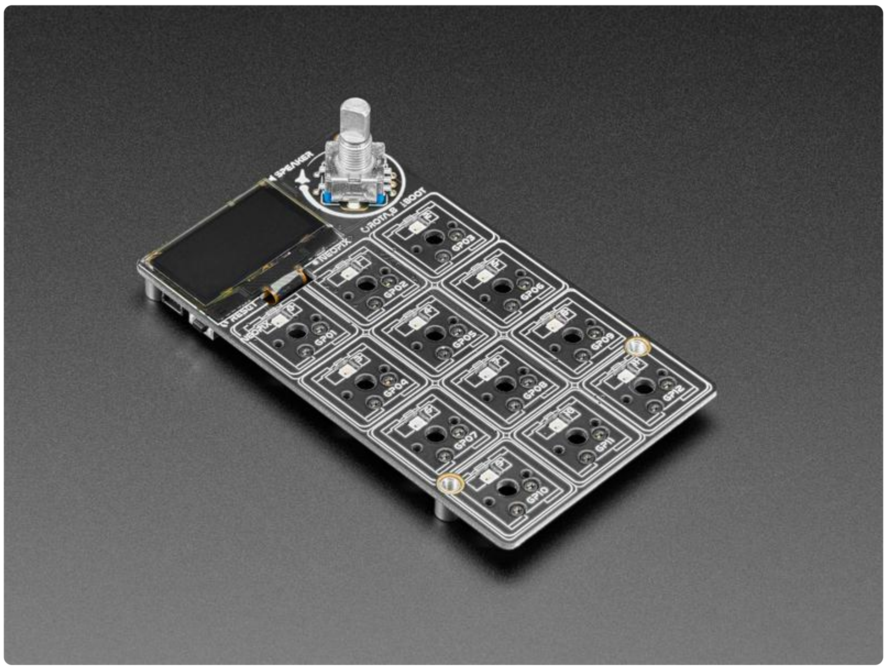
Last updated on 2023-05-04 02:49:16 PM EDT

Table of Contents

Overview



Strap yourself in, we're launching in T-minus 10 seconds...Destination? A new Class M planet called MACROPAD! M here, stands for Microcontroller because this 3x4 keyboard controller features the newest technology from the Raspberry Pi sector: say hello to the RP2040. It's speedy little microcontroller with lots of GPIO pins and a 64 times more RAM than the Apollo Guidance Computer. We added 8 MB of flash memory for plenty of storage.



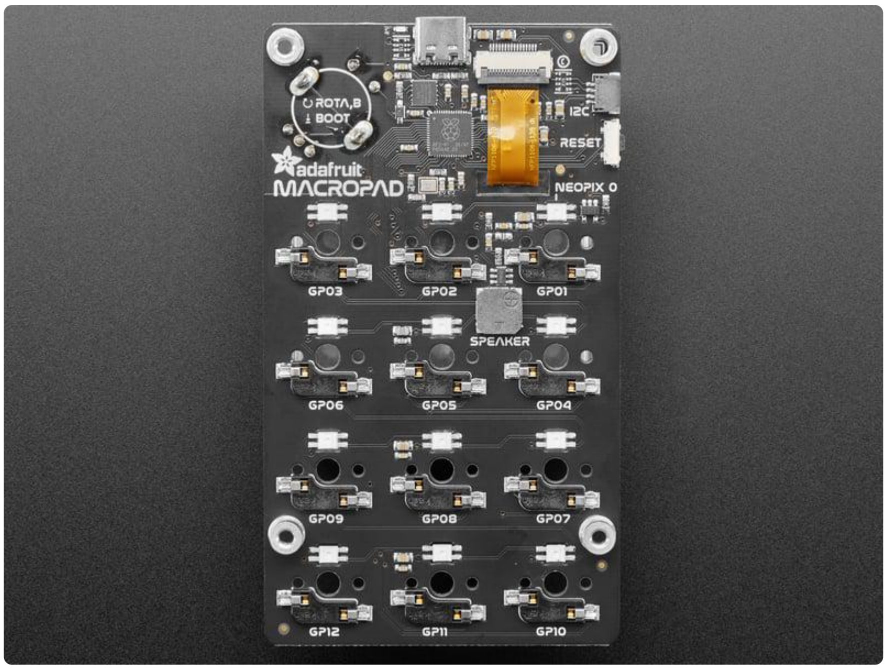
Get ready to upgrade your desk's mission control station with a CircuitPython or Arduino powered Macropad - complete with 12 buttons, OLED display, speaker and rotary encoder. Customize it for your spacecraft to help guide you through the great reaches of the unknown. (Or just have it type out your favorite emojis.)

Each of the 12 sockets can accept a Cherry MX-compatible key switch. No soldering required, just snap it in! Use any key switch you like - but we recommend ones with slots that will allow the matching twelve NeoPixels underneath to shine through.

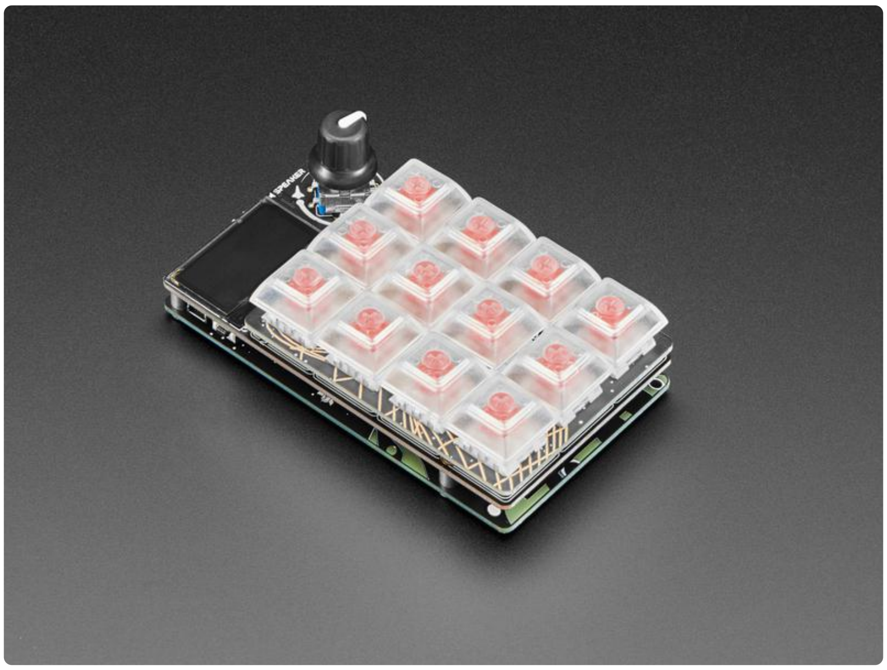
```
* Adafruit Macropad *
Rotary encoder: 0
I2C Scan:
```

This space-ship is also fitted with a 128x64 monochome OLED for a crisp heads-up display that can be used in Arduino or CircuitPython to display keymaps, stats, computer performance, etc. There's also a rotary encoder with push-button soldered in. Twist and turn it or push to change volume or monitor brightness or scroll: whatever you like! A tiny speaker can give audio feedback or play fun bleepy tunes.

Want to add more hardware? No worries - a STEMMA QT port on the side lets you connect any I2C add-on peripherals from the massive STEMMA QT / Qwiic family of plug in boards ().



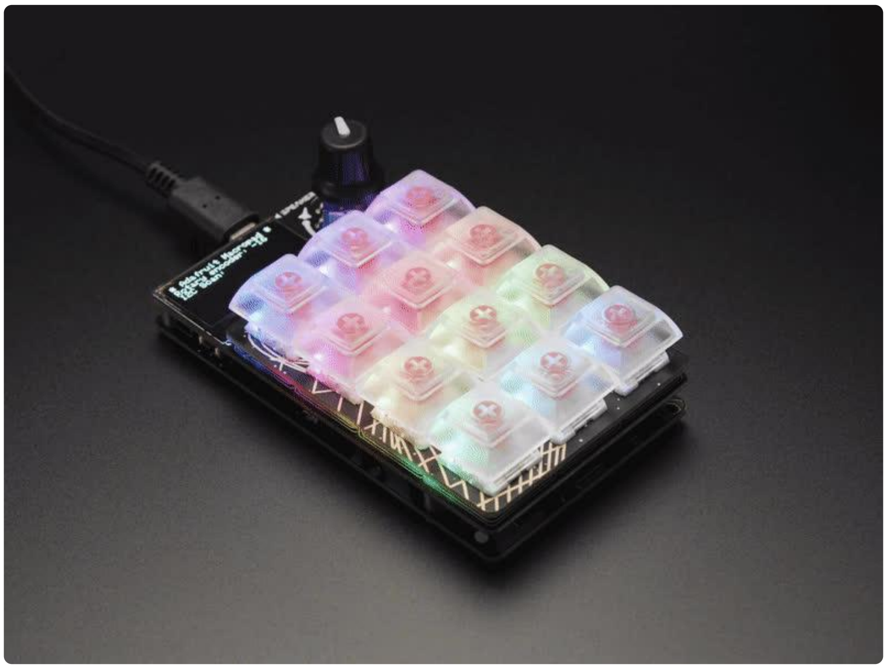
The RP2040 is now supported in QMK! You can use it with the MacroPad.



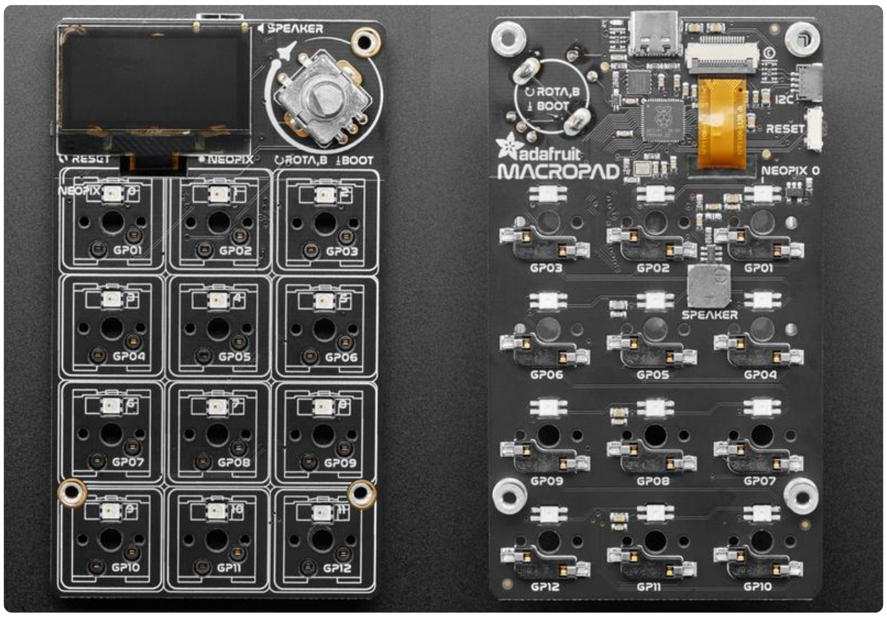
TL;DR?

- Raspberry Pi RP2040 Chip + 8MB Flash memory - Dual core Cortex M0+ at ~130MHz with 264KB of RAM. Runs CircuitPython, Arduino or MicroPython with ease and lots of space for development code and files
- USB C Connector for Power/Data - of course this can act as an HID device but also can be MIDI, UART, etc.
- 3x4 Mechanical key switch sockets - accepts any Cherry MX-compatible switches. Individually tied to GPIO pins (not matrix wired)

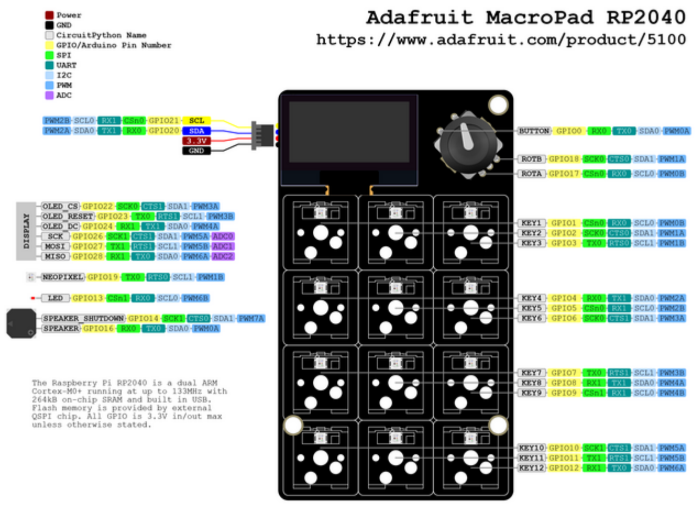
- One NeoPixel RGB LED per switch, on north side
- Rotary encoder, 20 detents per rotation, with push-switch on GPIO pin. Push switch is also used for entering bootloader mode when held down on power-up or reset.
- 128x64 SH1106 Monochrome OLED display - On high speed hardware SPI port for quick updates
- 8mm Speaker/Buzzer - With Class D amplifier and RC filter, can be used to make simple beeps and sounds effects.
- STEMMA QT Connector - Allows adding any I2C sensors/displays/devices with plug-and-play cables.
- Reset button - On the side, for quick restarting of code
- Four M3 mounting bosses - Make custom enclosures easily



Pinouts



The MacroPad RP2040 is full of macropad deliciousness. It has some great features beyond the keys. Time for tour!

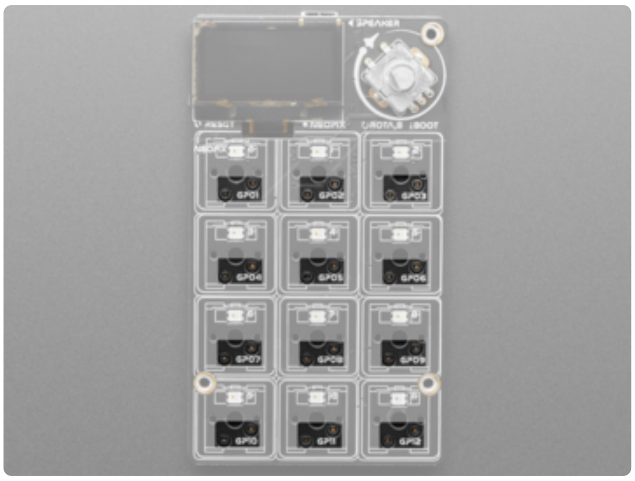


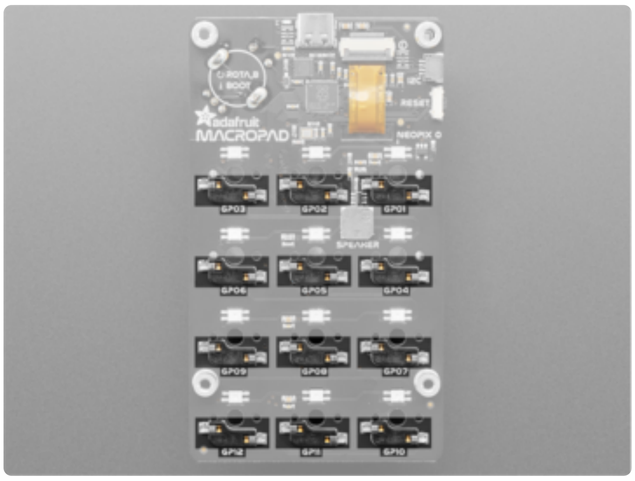
Adafruit MacroPad RP2040
https://www.adafruit.com/product/5100
🟥 Power
⬛ GND
⬜ CircuitPython Name
🟨 GPIO/Arduino Pin Number
🟩 SPI
🟩 UART
🟦 I2C
🟦 PWM
🟪 ADC
PWM2B SCL0 RX1 CSn0 GPIO21 SCL
PWM2A SDA0 TX1 RX0 GPIO20 SDA
3.3V
GND
DISPLAY
OLED_CS GPIO22 SCK0 CTS1 SDA1 PWM3A
OLED_RESET GPIO23 TX0 RTS1 SCL1 PWM3B
OLED_DC GPIO24 RX1 TX1 SDA0 PWM4A
SCK GPIO26 SCK1 CTS1 SDA1 PWM5A ADC0
MOSI GPIO27 TX1 RTS1 SCL1 PWM5B ADC1
MISO GPIO28 RX1 TX0 SDA0 PWM6A ADC2
NEOPIXEL GPIO19 TX0 RTS0 SCL1 PWM1B
LED GPIO13 CSn1 RX0 SCL0 PWM6B
SPEAKER_SHUTDOWN GPIO14 SCK1 CTS0 SDA1 PWM7A
SPEAKER GPIO16 RX0 TX0 SDA0 PWM0A
BUTTON GPIO0 RX0 TX0 SDA0 PWM0A
ROTB GPIO18 SCK0 CTS0 SDA1 PWM1A
ROTA GPIO17 CSn0 RX0 SCL0 PWM0B
KEY1 GPIO1 CSn0 RX0 SCL0 PWM0B
KEY2 GPIO2 SCK0 CTS0 SDA1 PWM1A
KEY3 GPIO3 TX0 RTS0 SCL1 PWM1B
KEY4 GPIO4 RX0 TX1 SDA0 PWM2A
KEY5 GPIO5 CSn0 RX1 SCL0 PWM2B
KEY6 GPIO6 SCK0 CTS1 SDA1 PWM3A
KEY7 GPIO7 TX0 RTS1 SCL1 PWM3B
KEY8 GPIO8 RX1 TX1 SDA0 PWM4A
KEY9 GPIO9 CSn1 RX1 SCL0 PWM4B
KEY10 GPIO10 SCK1 CTS1 SDA1 PWM5A
KEY11 GPIO11 TX1 RTS1 SCL1 PWM5B
KEY12 GPIO12 RX1 TX0 SDA0 PWM6A

The Raspberry Pi RP2040 is a dual ARM Cortex-M0+ running at up to 133MHz with 264kB on-chip SRAM and built in USB. Flash memory is provided by external QSPI chip. All GPIO is 3.3V in/out max unless otherwise stated.

PDF of the pinouts image above is available here ().

Key Switch Sockets





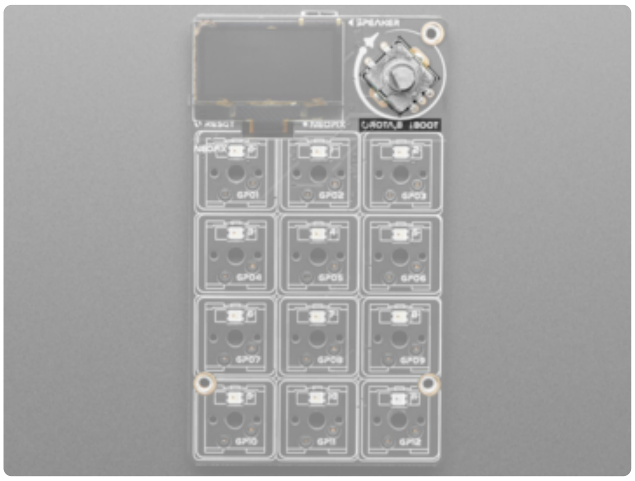
On the MacroPad, laid out in a 3x4 grid, are the Cherry-MX compatible key switch sockets. They are mounted on the back of the board so the socket points through to the front. The key switch sockets are individually tied to GPIO pins (i.e. not matrix-wired).

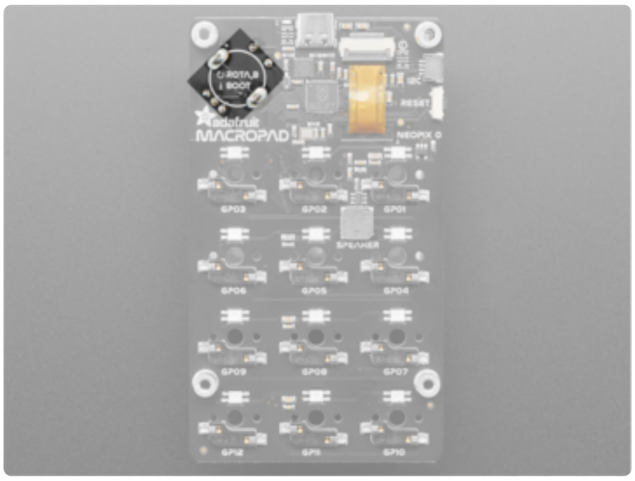
Simply press any compatible key switch into the socket from the top of the board. You can add a dab of glue to keep the switch in place; hot glue or a dot of epoxy will work.

The sockets are available in CircuitPython as `board.KEY1` through `board.KEY12`. In Arduino they are pins 1 through 12. Pressing a key grounds the pin, so set a pull-up on each pin.

Rotary Encoder / BOOT Button

We snuck the BOOT button in as the button switch in the rotary encoder. Press the rotary encoder to engage the BOOT button!



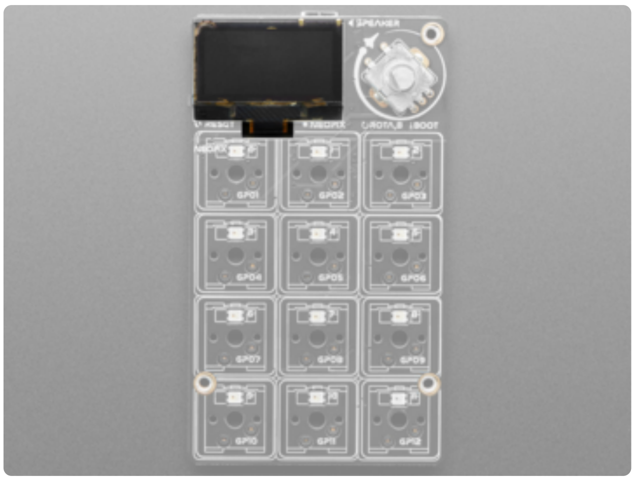


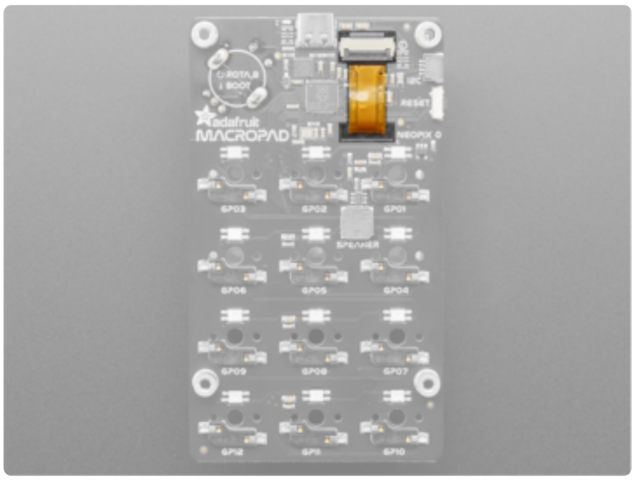
On the top right corner of the board (when viewed from the top), is the rotary encoder / BOOT button. The rotary encode has 20 detents per rotation. The BOOT button is required to enter the bootloader (needed for both CircuitPython and Arduino), and is also available as a user input in code. To use the button, simply press the rotary encoder down.

The rotary encoder is available in CircuitPython at `board.ROTA` or `board.ENCODER_A` and `board.ROTB` or `board.ENCODER_B`. It is available in Arduino as `PIN_ROTA` and `PIN_ROTB`.

The BOOT button is available in CircuitPython at `board.ENCODER_SWITCH` or `board.BUTTON`. It is available in Arduino at `PIN_SWITCH`. Pressing the button grounds the pin.

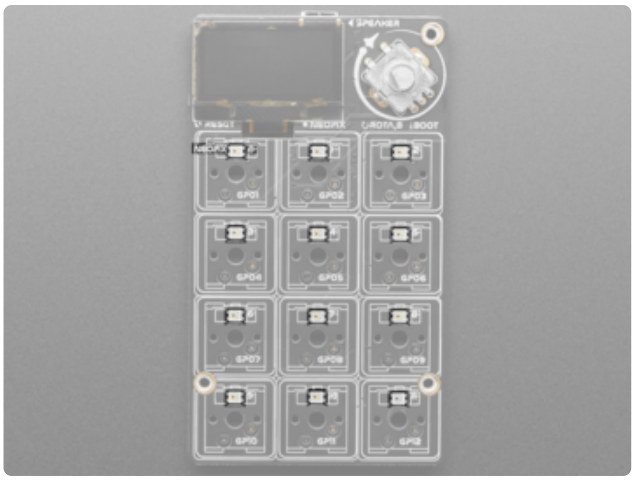
OLED Display

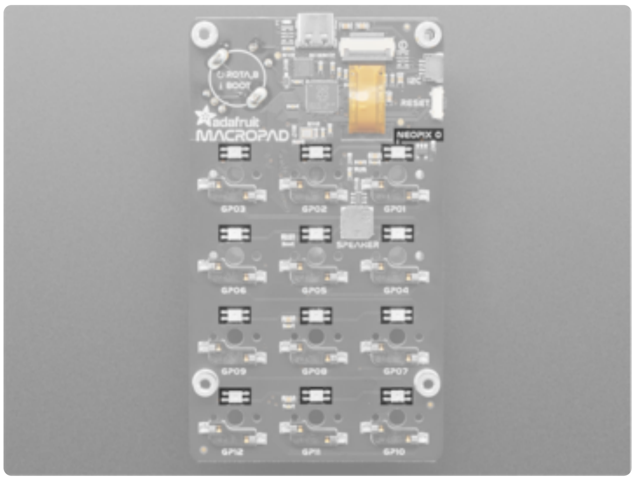




On the top left corner of the board (when viewed from the top), is the 128x64 SH1106 Monochrome OLED display. The ribbon cable for the display goes through a hole in the board to the back, where it is inserted into the display connector. This OLED is on high-speed hardware SPI to ensure quick updates.

NeoPixel LEDs

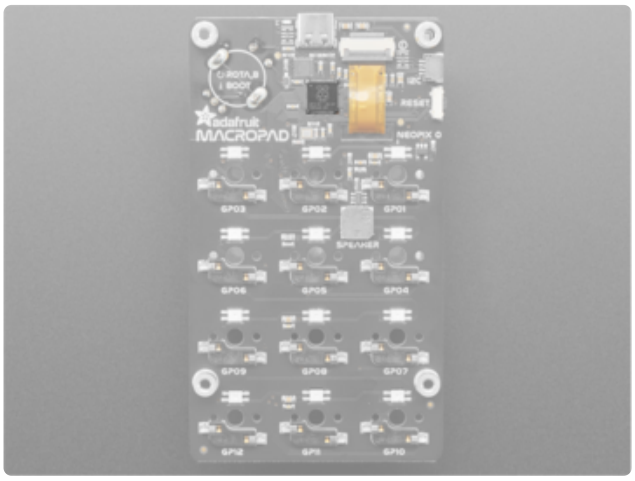




Above each set of key switch sockets, are RGB NeoPixel LEDs laid out in the same 3x4 grid. These reverse-mount LEDs are mounted to the back of the board to shine through to the front (to allow for the key switches to sit flush against the front of the board!).

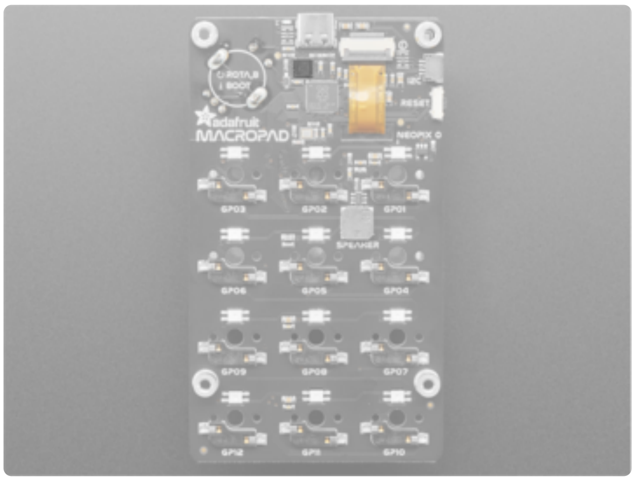
The NeoPixel LEDs are available in CircuitPython as `board.NEOPIXEL`. They are available in Arduino as `PIN_NEOPIXEL`.

RP2040 Microcontroller



The large square on the back of the board, at the top-center, is the RP2040 microcontroller. This is the brain of the board.

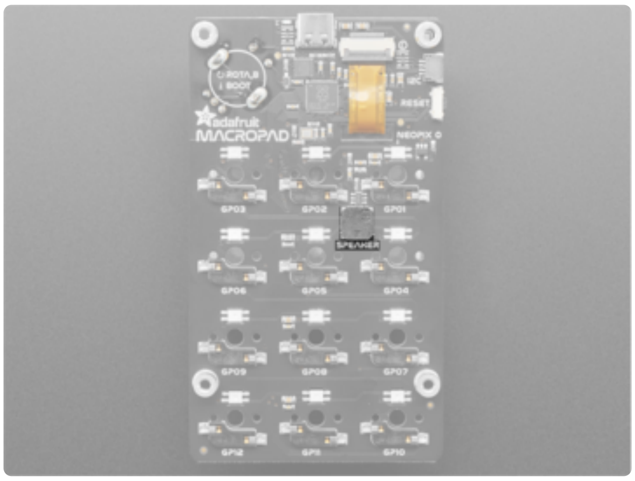
QSPI Flash



The little square above the RP2040 microcontroller is the 8MB QSPI flash.

QSPI is neat because it allows you to have 4 data in/out lines instead of just SPI's single line in and single line out. This means that QSPI is at least 4 times faster. But in reality is at least 10x faster because you can clock the QSPI peripheral much faster than a plain SPI peripheral. In CircuitPython, the QSPI flash is used natively by the interpreter and is read-only to user code, instead the flash just shows up as the writable disk drive!

Speaker

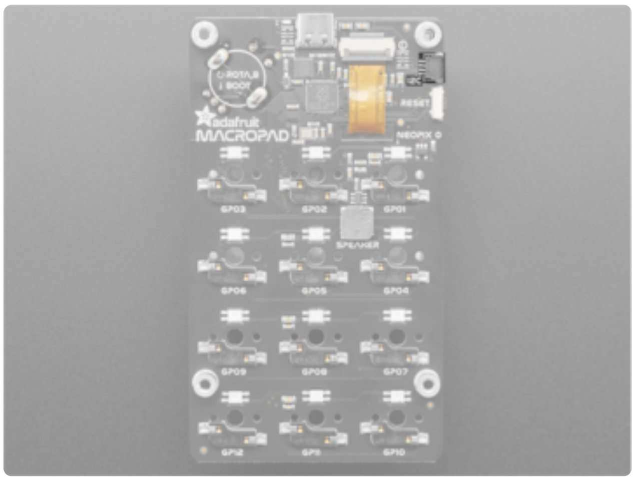


The grey square a bit to the right of the center of the board on the back is the 8mm speaker/buzzer. With a Class D amplifier and RC filter, it can be used to make simple beeps and sounds effects.

The speaker is available in CircuitPython at `board.SPEAKER`. It is available in Arduino at `PIN_SPEAKER`.

The speaker must be enabled to work in code. The speaker enable pin in CircuitPython is `board.SPEAKER_ENABLE`. In Arduino, it is `PIN_SPEAKER_ENABLE`. Set the pin high to enable the speaker.

STEMMA QT Connector

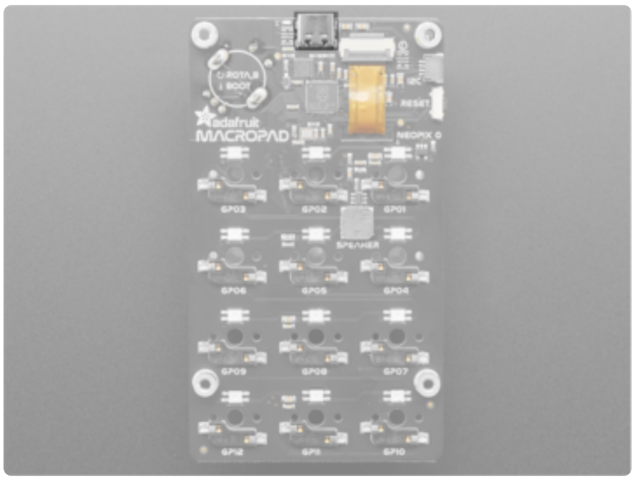


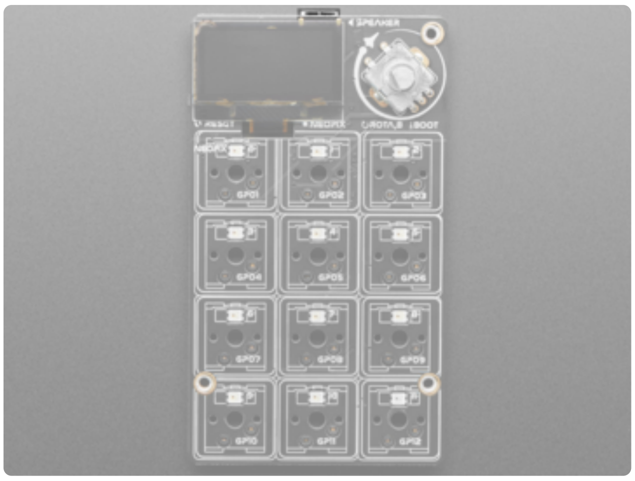
In the top right corner of the back of the board, below the mounting boss, is the STEMMA QT () connector. This Qwiic-compatible I2C connector is designed to make it super simple to connect up STEMMA QT sensors and breakouts ().

The I2C SCL and SDA pins on the connector are connected to 10k pull-up resistors, so they will be at 3.3V when quiescent.

In CircuitPython, you can use the STEMMA connector with `board.SCL` and `board.SDA`, or `board.STEMMA_I2C()`.

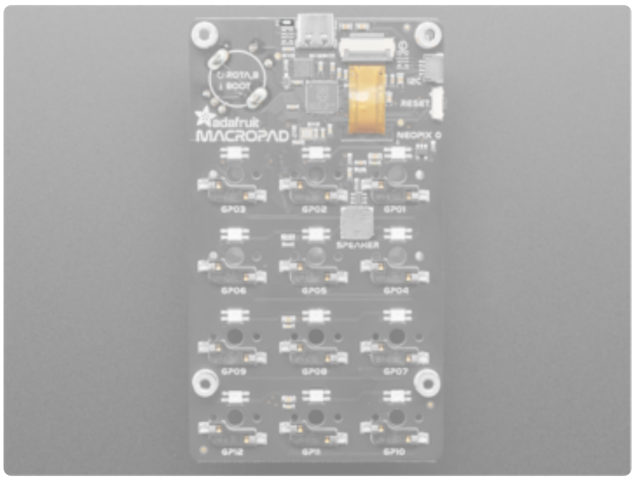
USB C Connector





At the top-center of the board, visible from both sides, is the USB Type C connector. This connector is used both for transferring data from your computer (e.g. updating your CircuitPython code.py file, or uploading an Arduino sketch) and powering the board.

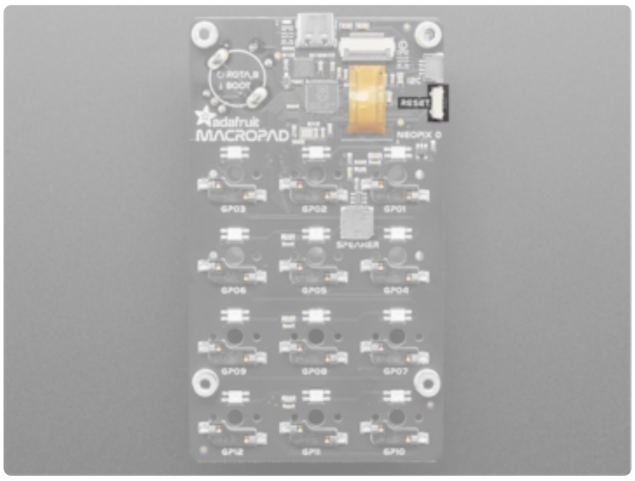
Red LED

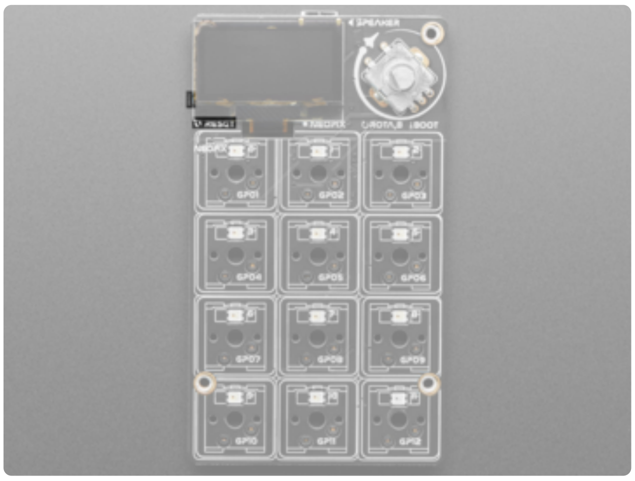


On the top edge of the back of the board, to the right of the USB Type C connector is the red LED. You can control this in your code.

The red LED is available in CircuitPython at `board.LED`. It is available in Arduino at `PIN_LED`.

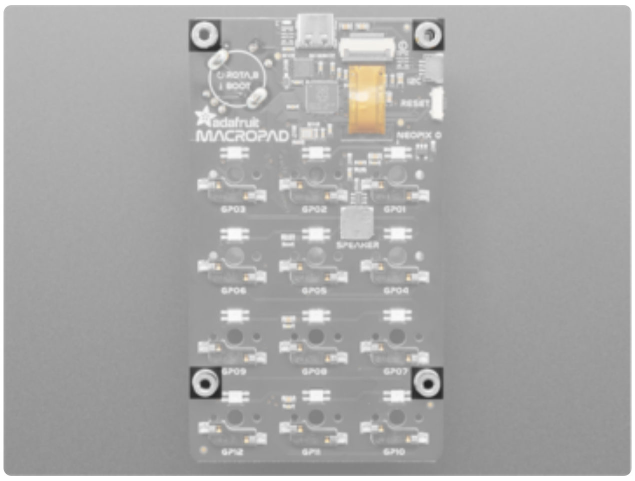
Reset Button





On the right edge of the back of the board (visible from the front), below the STEMMA QT connector, is the reset button. Tap once to reset the board. When combined with the boot button, the reset button allows the board to enter the bootloader.

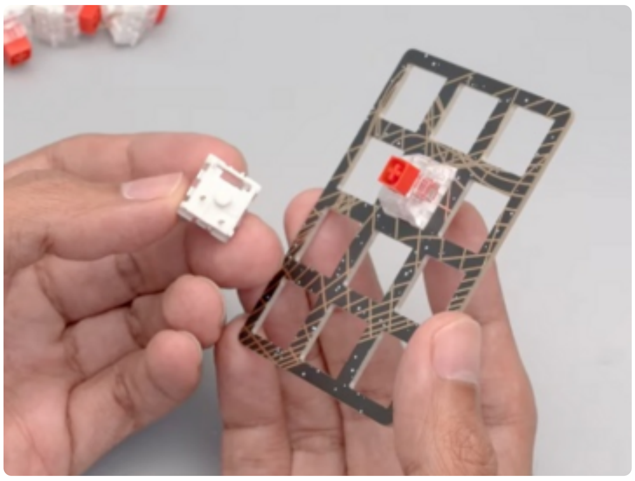
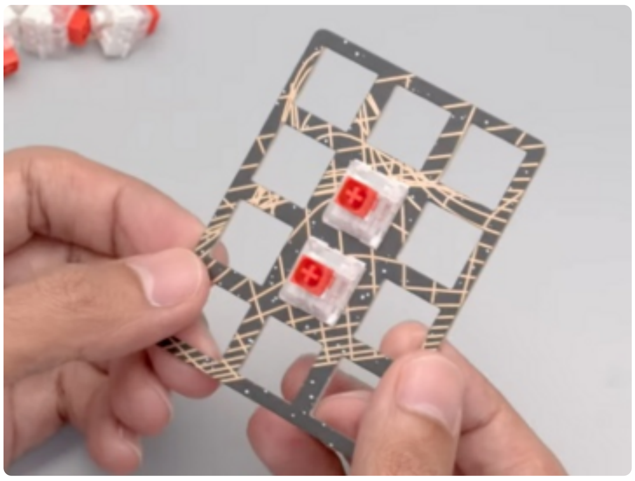
Mounting Bosses



Arranged in the top two corners of the board, and towards the bottom two corners are four mounting bosses to allow for using the MacroPad enclosure kit () or designing your own enclosures.

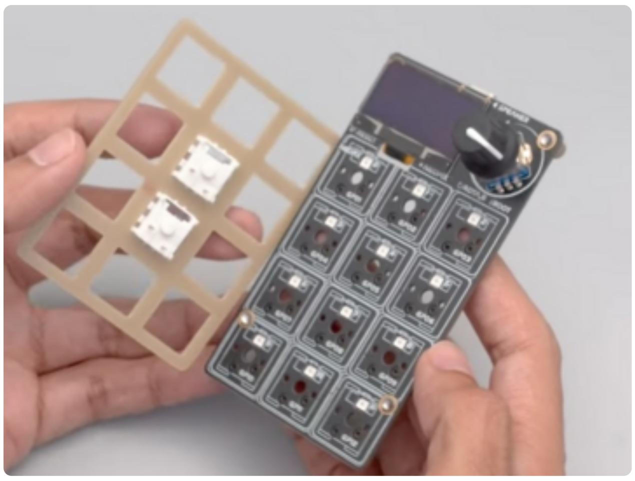
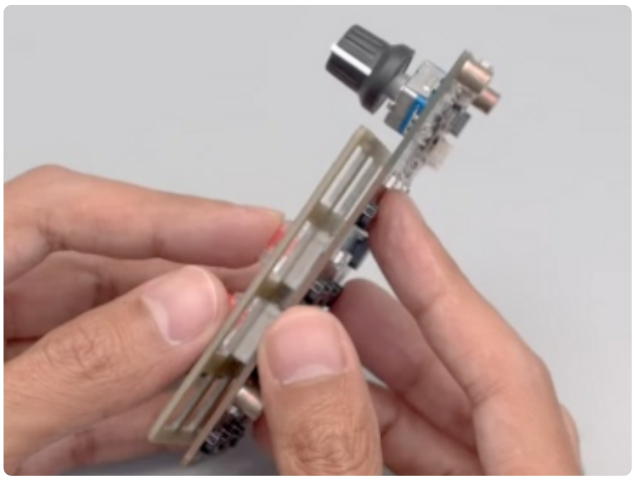
Macropad Assembly

The Macropad features hot-swap sockets for the switches -- gone are the days of having to commit to one type of switch and solder it down! Now, you can plug in your Cherry MX red keyswitches, use them for a while, get bored, decide its time to test out some lubed, filmed, re-sprung Invyr Holy Pandas, and swap them just like that!

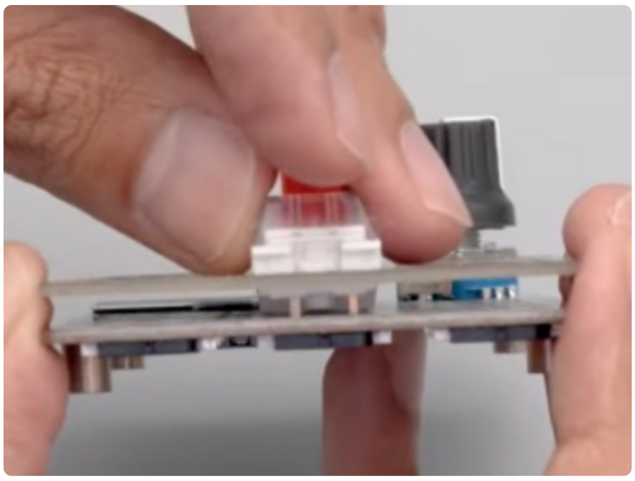
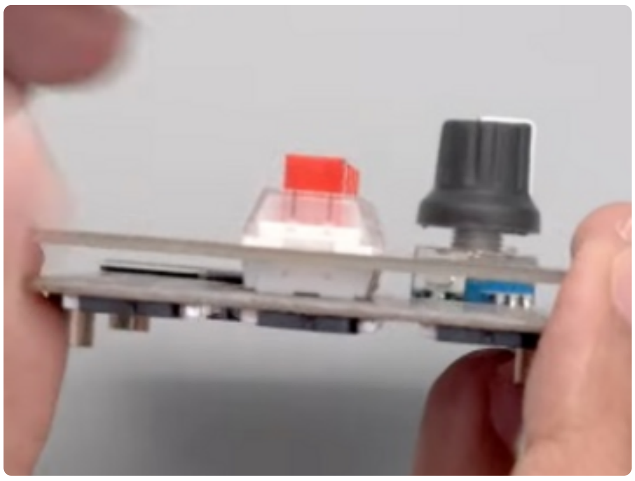
Switches into Plate

First, insert a couple pf keyswitches through the keyswitch plate. The plate mechanically connects the switches to each other, which lends some nice lateral stability to the keys.

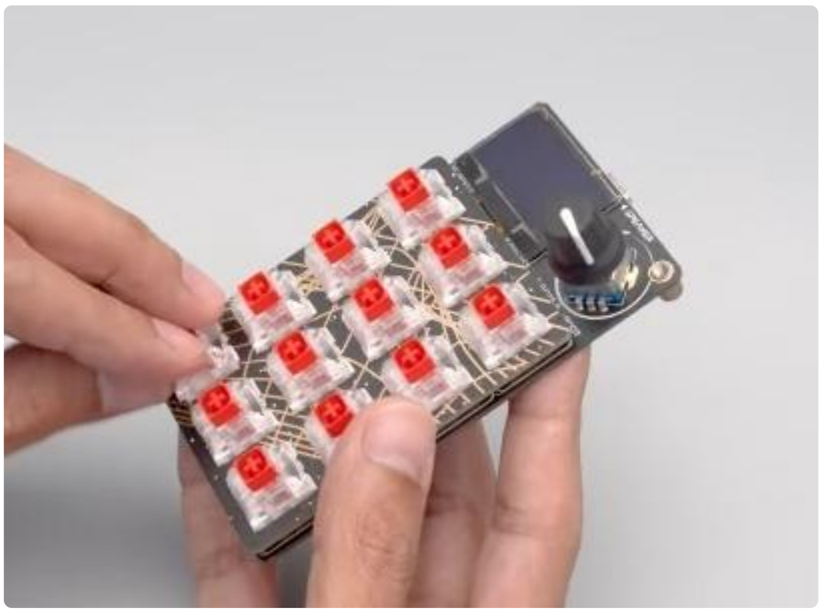
Connect to Board

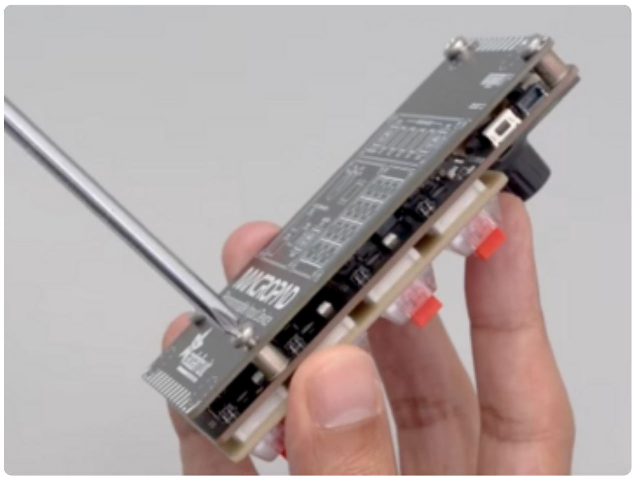
Carefully press the two switches into the switch sockets, being very careful to align the legs so none bend!

Add Switches

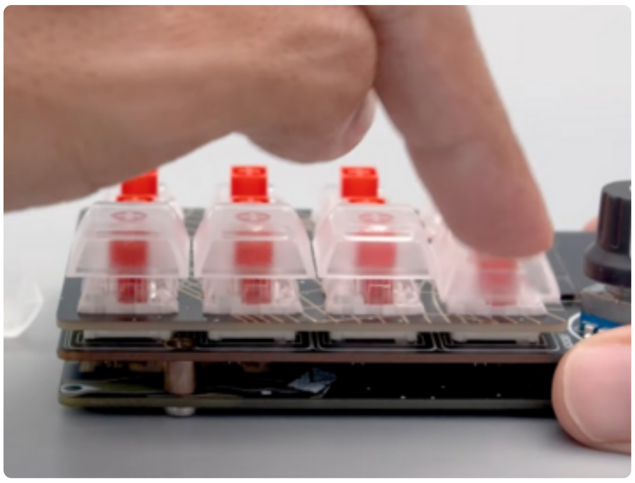
Continue adding switches, being mindful of their orientation.





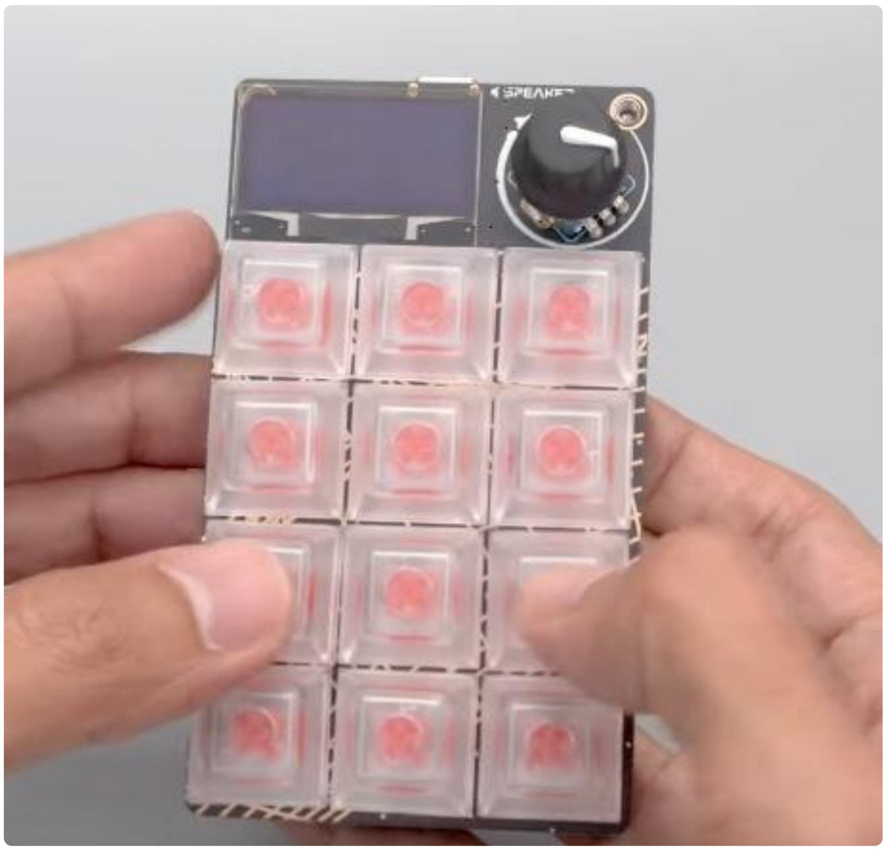
Backplate

You can add the optional backplate using four M3 x 6mm screws.



Keycaps

Now, you can add your keycaps! simply press them onto the keyswitch stems until they are fully seated.



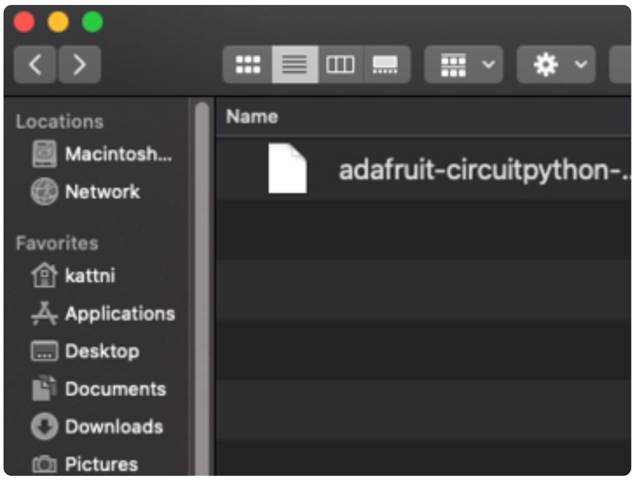
CircuitPython

CircuitPython () is a derivative of MicroPython () 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

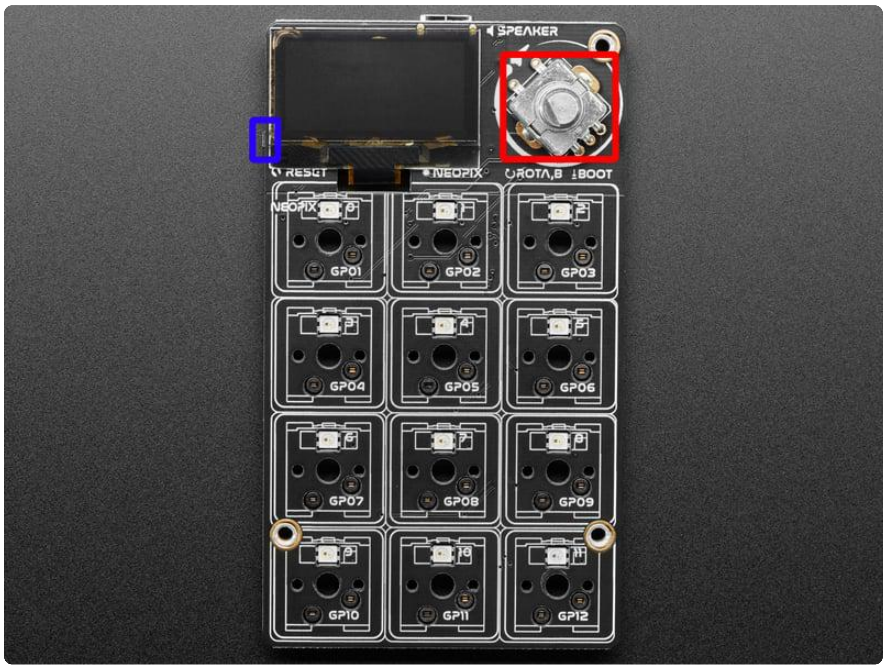
Follow this step-by-step to quickly get CircuitPython running on your board.

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



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

Save it wherever is convenient for you.



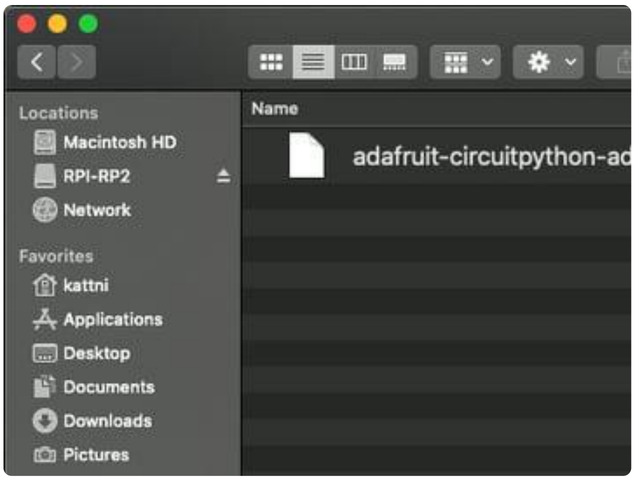
The BOOT button is the button switch in the rotary encoder! To engage the BOOT button, simply press down on the rotary encoder.

To enter the bootloader, hold down the BOOT/BOOTSEL button (highlighted in red above), and while continuing to hold it (don't let go!), press and release the reset button (highlighted in blue above). Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!

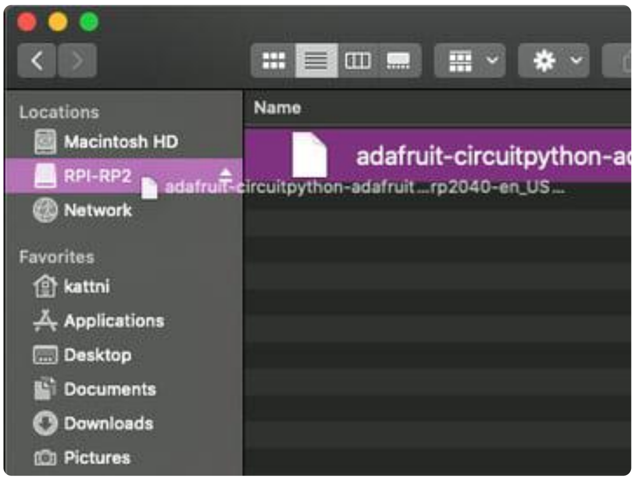
If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

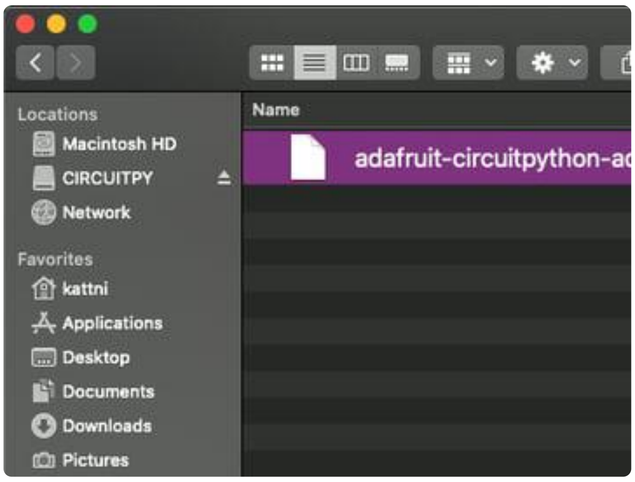
A lot of people end up using charge-only USB cables and it is very frustrating! Make sure you have a USB cable you know is good for data sync.



You will see a new disk drive appear called RPI-RP2.



Drag the adafruit_circuitpython_etc.uf2 file to RPI-RP2.



The RPI-RP2 drive will disappear and a new disk drive called CIRCUITPY will appear.

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

Safe Mode

You want to edit your code.py or modify the files on your CIRCUITPY drive, but find that you can't. Perhaps your board has gotten into a state where CIRCUITPY is read-only. You may have turned off the CIRCUITPY drive altogether. Whatever the reason, 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

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, 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

If you successfully enter safe mode on CircuitPython, 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.

Flash Resetting UF2

If your board ever gets into a really weird state and doesn't even show up as a disk drive when installing CircuitPython, try loading this 'nuke' UF2 which will do a 'deep clean' on your Flash Memory. You will lose all the files on the board, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

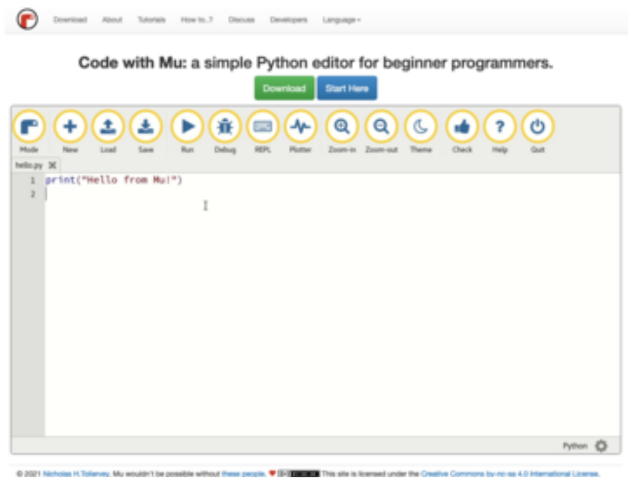
Download flash erasing "nuke" UF2

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


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

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

Download Mu from https://codewith.mu ().

Click the Download link for downloads and installation instructions.

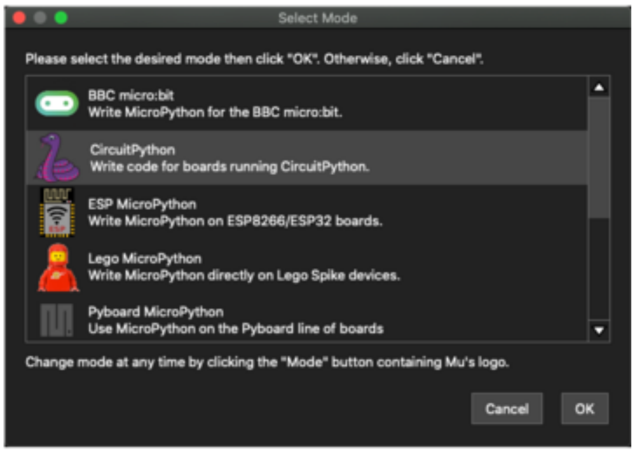
Click Start Here to find a wealth of other information, including extensive tutorials and and how-to's.

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

Ubuntu users: Mu currently (checked May 4, 2022) does not install properly on Ubuntu 22.04. See https://github.com/mu-editor/mu/issues to track this issue.

See https://learn.adafruit.com/welcome-to-circuitpython/recommended-editors and https://learn.adafruit.com/welcome-to-circuitpython/pycharm-and-circuitpython for other editors to use.

Starting Up Mu



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

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

CircuitPython
Write code for boards running CircuitPython.

ESP MicroPython
Write MicroPython on ESP8266/ESP32 boards.

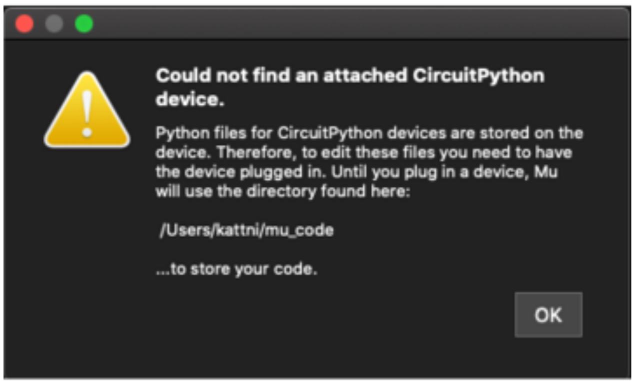
Lego MicroPython
Write MicroPython directly on Lego Spike devices.

Pyboard MicroPython
Use MicroPython on the Pyboard line of boards

Change mode at any time by clicking the "Mode" button containing Mu's logo.

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

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



Could not find an attached CircuitPython device.

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

/Users/kattni/mu_code

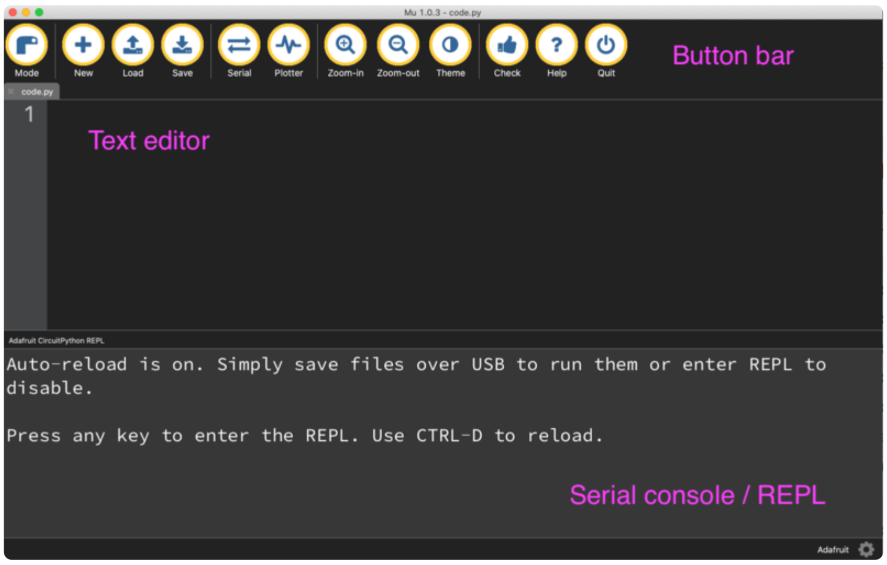
...to store your code.

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.



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

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

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

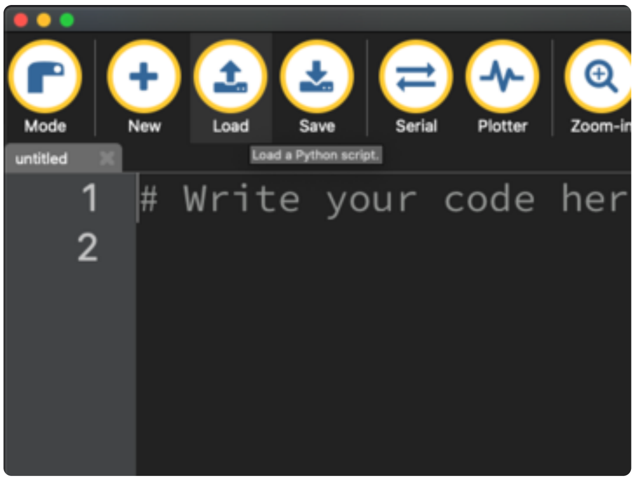
Creating and Editing Code

One of the best things about CircuitPython is how simple it is to get code up and running. 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 () 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 is not a problem on MacOS.)

Creating Code



```
# Write your code her
```

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.

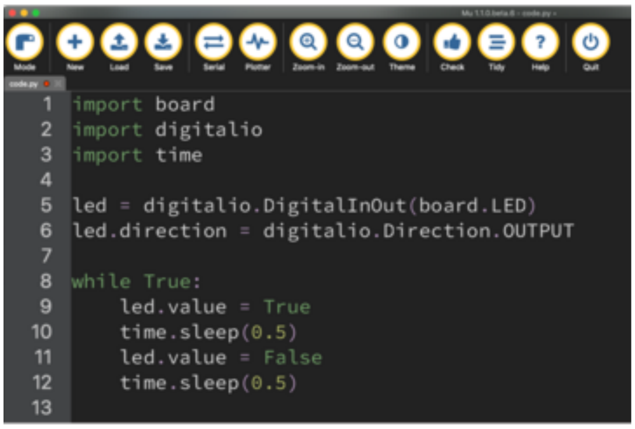
Copy and paste the following code into your editor:

```
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 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 or the Trinkeys!

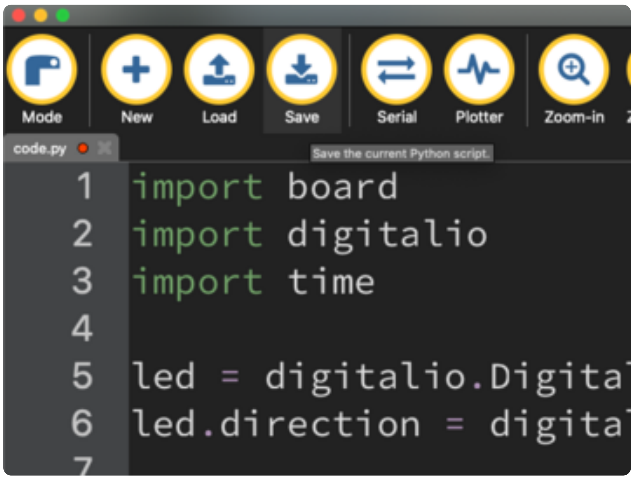
If you're using a KB2040, QT Py or a Trinkey, please download the NeoPixel blink example ().

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.



```
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)
```

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.



```
import board
import digitalio
import time

led = digitalio.Digita
led.direction = digita
```

Save the code.py file on your CIRCUITPY drive.

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

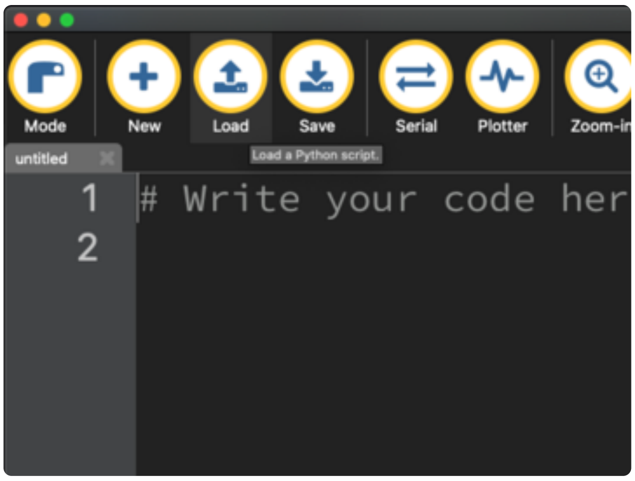
Congratulations, you've just run your first CircuitPython program!

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, and the Trinkey series, you will find only an RGB NeoPixel LED.

Editing Code



```
# Write your code her
```

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 () 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 () page of every board guide to get your board up and running again.

Back to Editing Code...

Now! Let's try editing the program you added to your board. Open your code.py file into your editor. 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.tx t, 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.

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!`

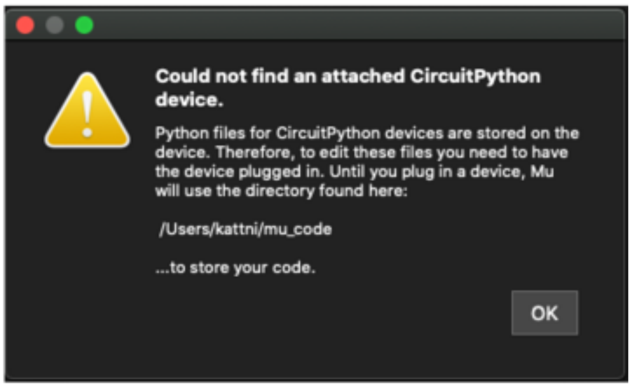
However, these print statements need somewhere to display. That's where the serial console comes in!

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.



Could not find an attached CircuitPython device.

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

/Users/kattni/mu_code

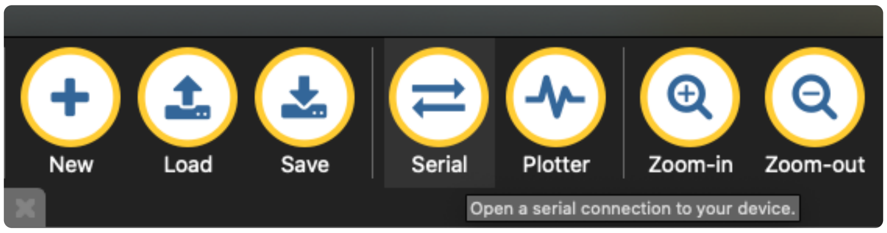
...to store your code.

First, make sure your CircuitPython board is plugged in.

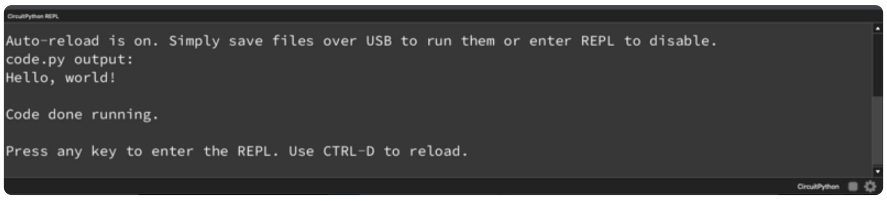
If you open Mu without a board plugged in, you may encounter the error seen here, letting you know no CircuitPython board was found and indicating where your code will be stored until you plug in a board.

If you are using Windows 7, make sure you installed the drivers ().

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.



```
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.
```

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

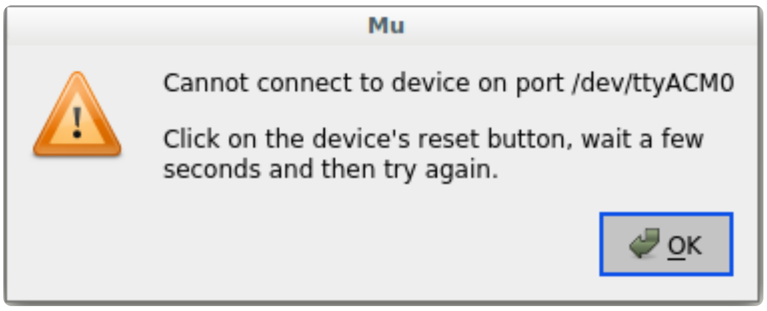
If you're on Linux, and are seeing multi-second delays connecting to the serial console, or are seeing "AT" and other gibberish when you connect, then the `modemmanager` service might be interfering. Just remove it; it doesn't have much use unless you're still using dial-up modems.

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 S erial button, you need to add yourself to a user group to have permission to connect to the serial console.



Mu

Cannot connect to device on port /dev/ttyACM0

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

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

```
sudo adduser $USER dialout
```

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

Using Something Else?

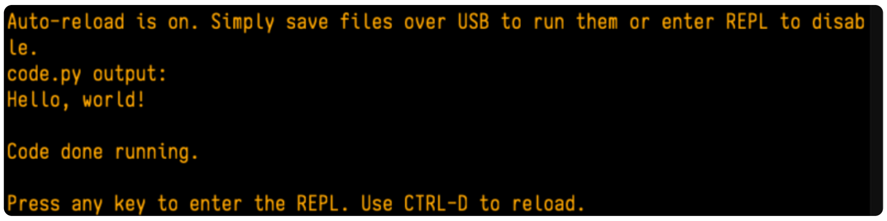
If you're not using Mu to edit, are using or if for some reason you are not a fan of its built in serial console, you can run the serial console from a separate program.

Windows requires you to download a terminal program. Check out the Advanced Serial Console on Windows page for more details. ()

MacOS has Terminal built in, though there are other options available for download. Check the Advanced Serial Console on Mac page for more details. ()

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. ()

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 disab
le.
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.

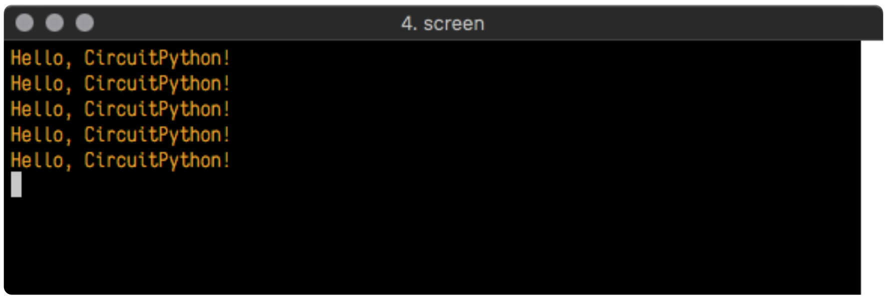
Open your code.py file into your editor, and include a `print` statement. You can print anything you like! Just include your phrase between the quotation marks inside the parentheses. For example:

```
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.

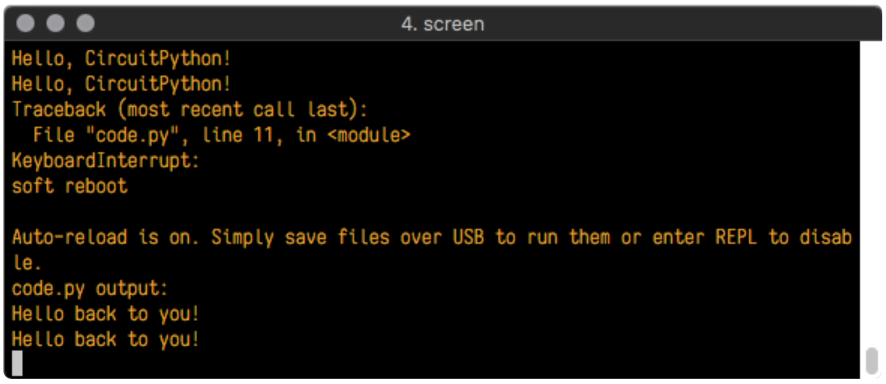


```
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
```

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

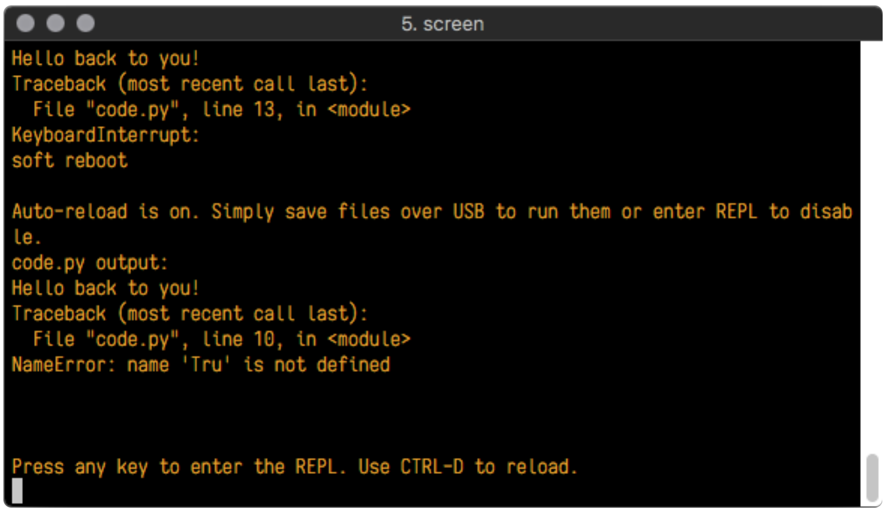
The `Traceback (most recent call last):` is telling you the last thing your board was doing before you saved your file. This is normal behavior and will happen every time the board resets. This is really handy for troubleshooting. Let's introduce an error so you can see how it is used.

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!

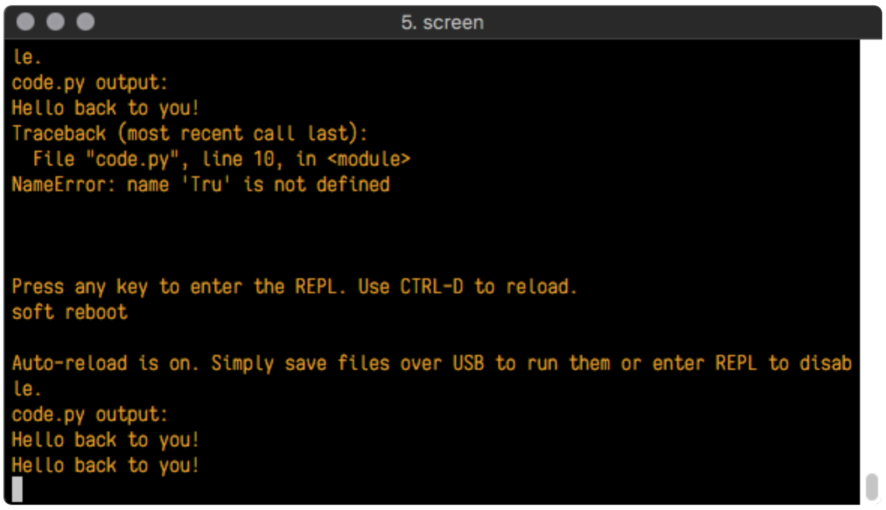
Usually when you run into errors, it's not because you introduced them on purpose. You may have 200 lines of code, and have no idea where your error could be hiding. This is where the serial console can help. Let's take a look!



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

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



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

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

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



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



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

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

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

The serial console will display any output generated by your code. Some sensors, such as a humidity sensor or a thermistor, receive data and you can use print statements to display that information. You can also use print statements for troubleshooting, 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

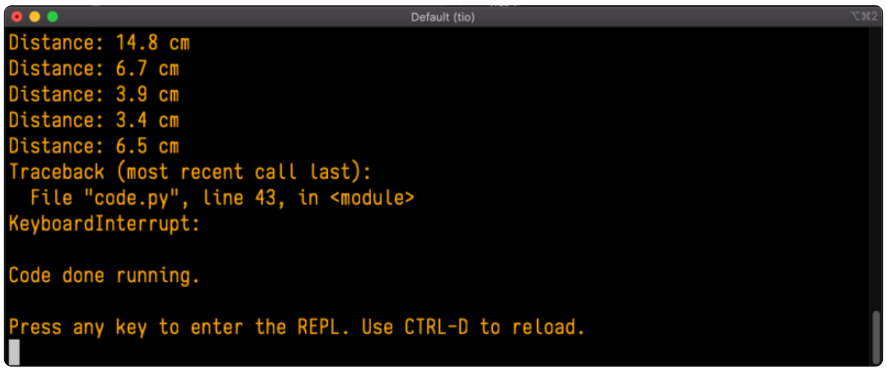
The other feature of the serial connection is the Read-Evaluate-Print-Loop, or REPL. The REPL allows you to enter individual lines of code and have them run immediately. It's really handy if you're running into trouble with a particular program and can't figure out why. It's interactive so it's great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

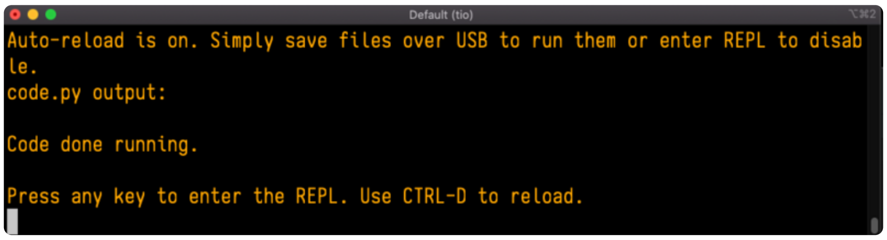
If there is code running, in this case code measuring distance, it will stop and you'll see `Press any key to enter the REPL. Use CTRL-D to reload.` Follow those instructions, and press any key on your keyboard.

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.



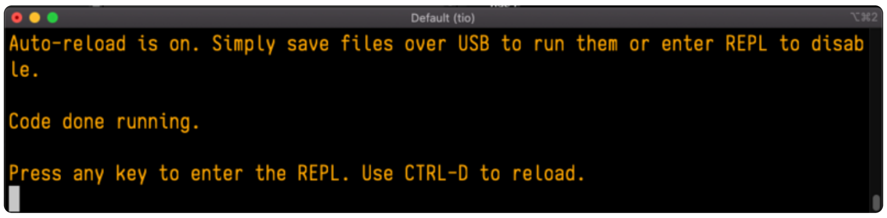
```
Distance: 14.8 cm
Distance: 6.7 cm
Distance: 3.9 cm
Distance: 3.4 cm
Distance: 6.5 cm
Traceback (most recent call last):
  File "code.py", line 43, in <module>
KeyboardInterrupt:

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.



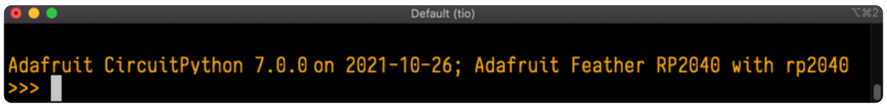
```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.
code.py output:

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.



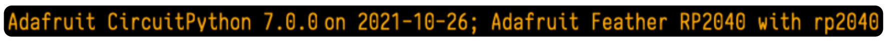
```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.

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!


```
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.

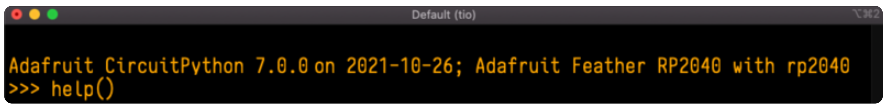
This is followed by the CircuitPython prompt.

```
>>>
```

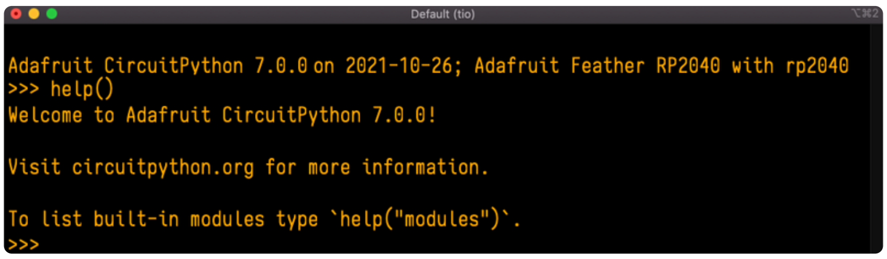
Interacting with the REPL

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.

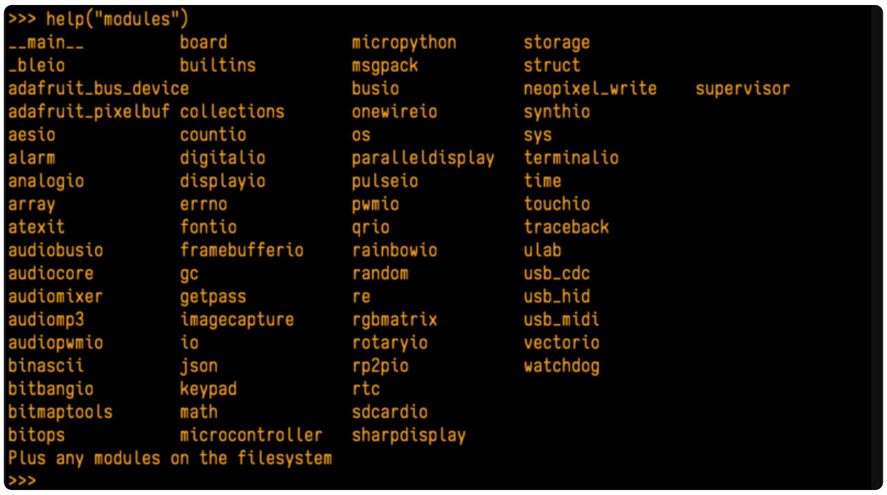

```
Adafruit CircuitPython 7.0.0 on 2021-10-26; Adafruit Feather RP2040 with rp2040
>>> help()
```

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!

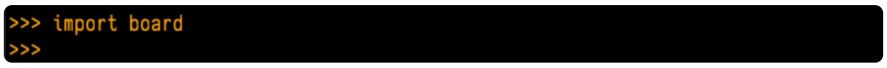
Type `help("modules")` into the REPL next to the prompt, and press enter.



```
>>> 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
>>>
```

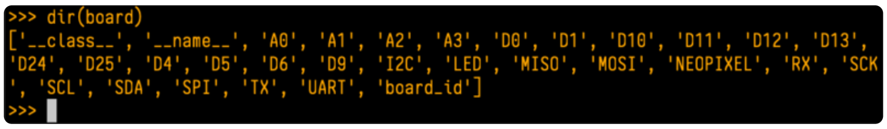
This is a list of all the core modules built into CircuitPython, including `board`. Remember, `board` contains all of the pins on the board that you can use in your code. From the REPL, you are able to see that list!

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']
>>> 
```

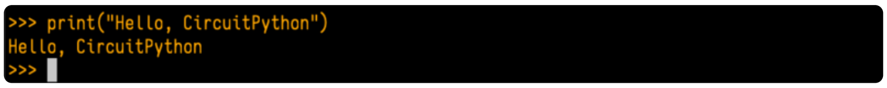
This is a list of all of the pins on your board that are available for you to use in your code. Each board's list will differ slightly depending on the number of pins available. Do you see `LED`? That's the pin you used to blink the red LED!

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.

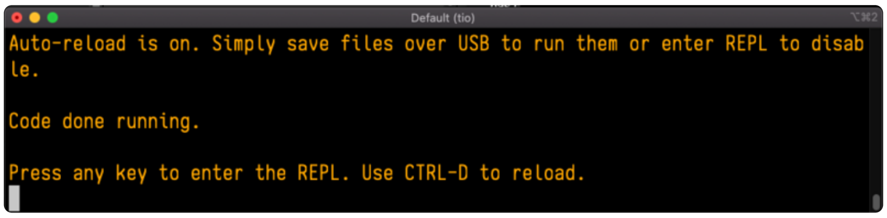
Try typing more into the REPL to see what happens!

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 CT RL+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!

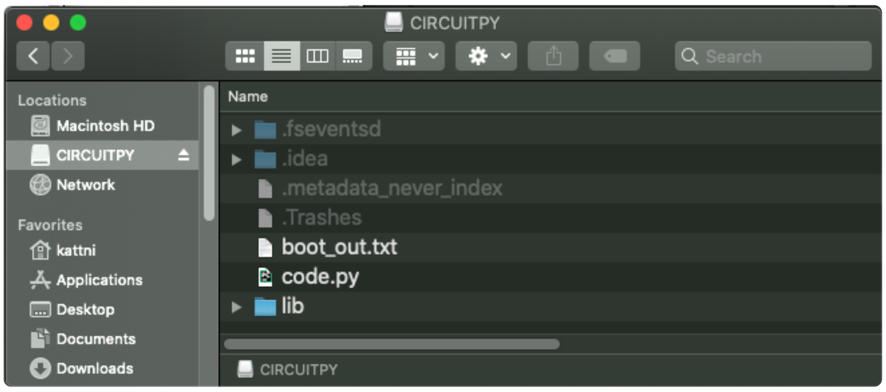

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.

Code done running.

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 () 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.

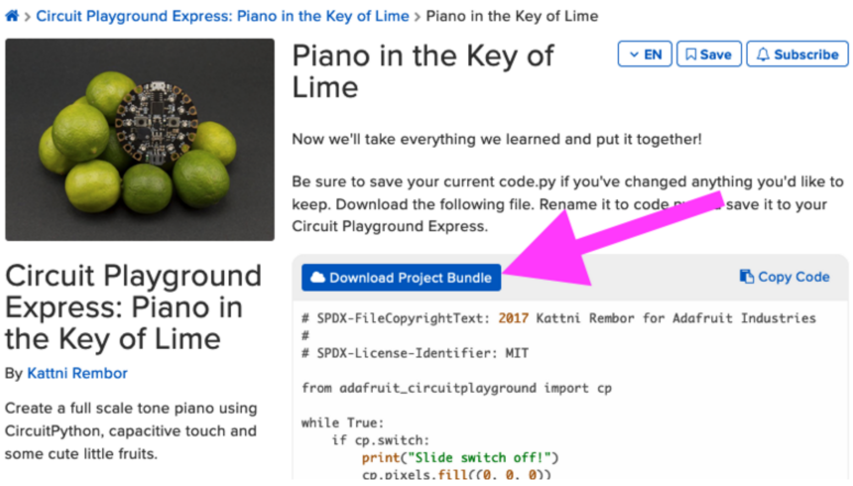
Either way, as you start to explore CircuitPython, you'll want to know how to get libraries on board.

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

Circuit Playground Express: Piano in the Key of Lime

By Kattni Rembor

Create a full scale tone piano using CircuitPython, capacitive touch and some cute little fruits.

Piano in the Key of Lime

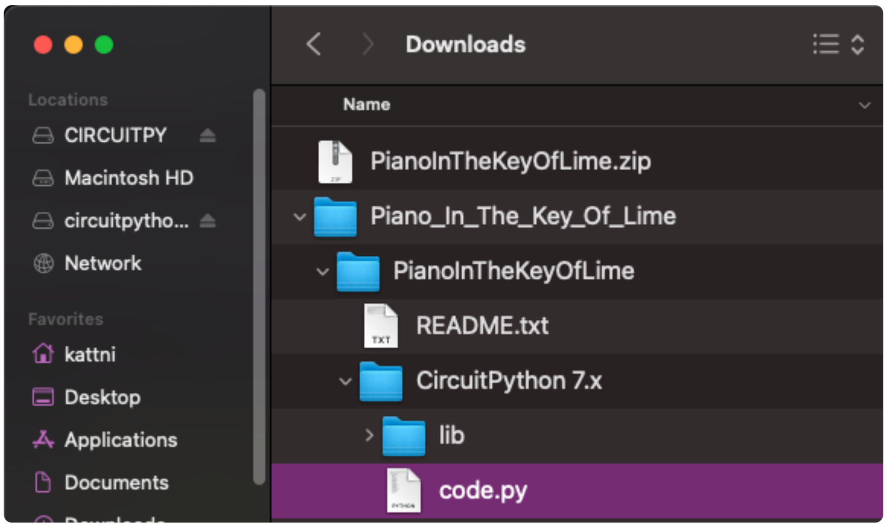
Now we'll take everything we learned and put it together!

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 save it to your Circuit Playground Express.

```
# 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

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

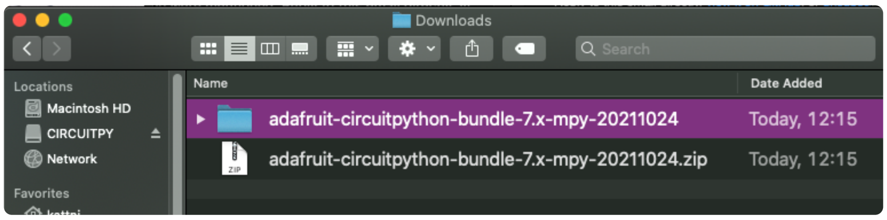
You can download the latest CircuitPython Community Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

Click for the latest CircuitPython Community Library Bundle release

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.

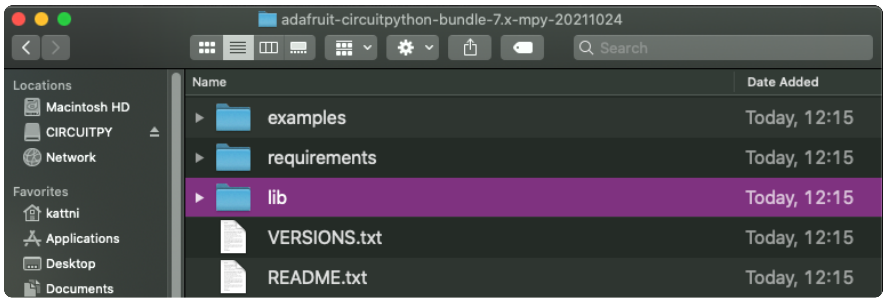
Understanding the Bundle

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.



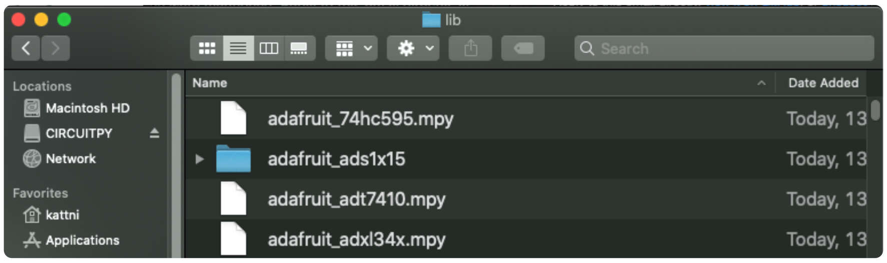
Name	Date Added
adafruit-circuitpython-bundle-7.x-mpy-20211024	Today, 12:15
adafruit-circuitpython-bundle-7.x-mpy-20211024.zip	Today, 12:15

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.



Name	Date Added
examples	Today, 12:15
requirements	Today, 12:15
lib	Today, 12:15
VERSIONS.txt	Today, 12:15
README.txt	Today, 12:15

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

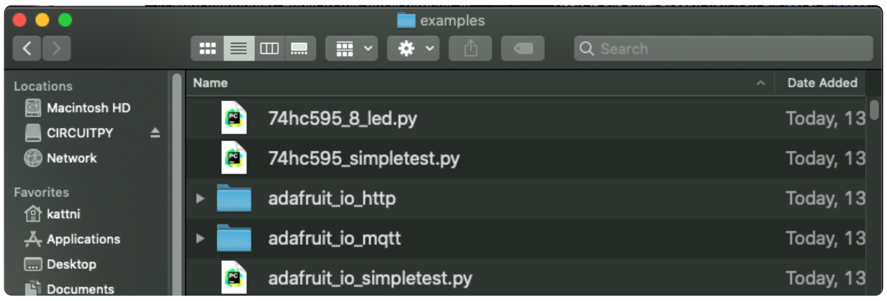


Name	Date Added
adafruit_74hc595.mpy	Today, 13
adafruit_ads1x15	Today, 13
adafruit_adt7410.mpy	Today, 13
adafruit_adxl34x.mpy	Today, 13

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.

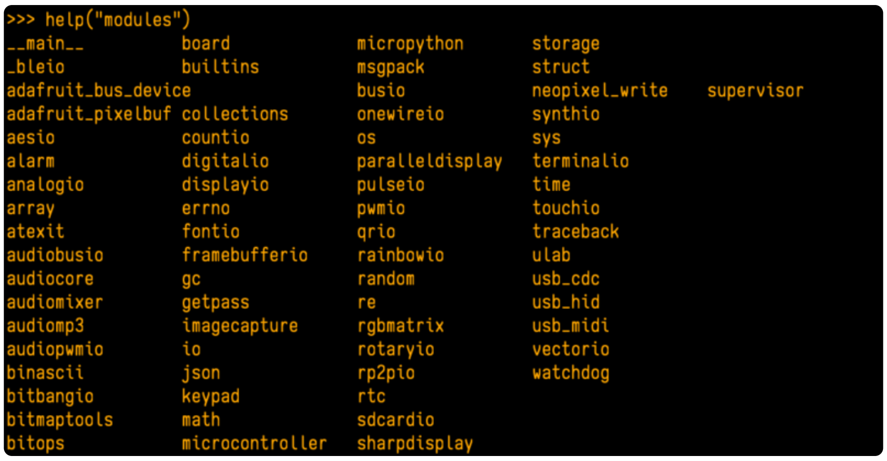
Here is an example import list for you to work with in this section. There is no setup or other code shown here, as the purpose of this section involves only the import list.

```
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 () on The REPL page () 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                 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
```

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 CI RCUITPY 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

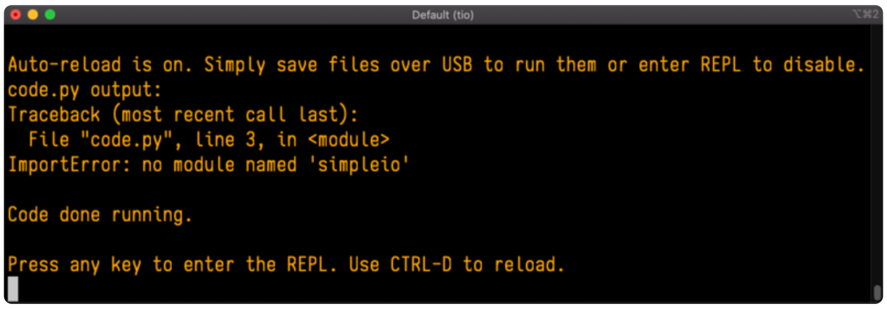
If you choose to load libraries as you need them, or you're starting fresh with an existing example, you may end up with code that tries to use a library you haven't yet loaded. This section will demonstrate what happens when you try to utilise a library that you don't have loaded on your board, and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

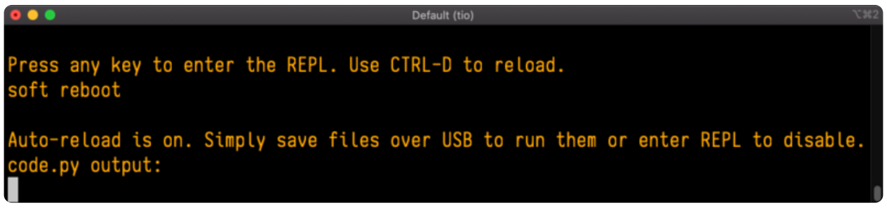


```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Traceback (most recent call last):
  File "code.py", line 3, in <module>
ImportError: no module named 'simpleio'

Code done running.

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

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.



```
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 ().

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 () 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 (). 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 () for a full list of functionality

---

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 ()
CPC = Circuit Playground Classic () (does not run CircuitPython)
CPX = Circuit Playground Express ()
CPB = Circuit Playground Bluefruit ()

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 6.x or earlier. Where can I find compatible libraries?

We are no longer building or supporting the CircuitPython 6.x or earlier library bundles. We highly encourage you to update CircuitPython to the latest version () and use the current version of the libraries (). However, if for some reason you cannot update, here are the last available library bundles for older versions:

- 2.x bundle ()
- 3.x bundle ()
- 4.x bundle ()
- 5.x bundle ()
- 6.x bundle ()

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 () 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 ().

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

The nRF52840 and nRF52833 boards 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.

ESP32-C3 and ESP32-S3 boards currently provide an incomplete () BLE implementation. Your program can act as a central, and connect to a peripheral. You can advertise, but you cannot create services. You cannot advertise anonymously. Pairing and bonding are not supported.

The ESP32 could provide a similar implementation, but it is not yet available. 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 () or other NINA-FW-based co-processors. Some boards have this coprocessor on board, such as the PyPortal (). 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 ()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 () 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! ()

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 () 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 (). Builds are available for Windows, macOS, x64 Linux, and Raspberry Pi Linux. Choose the latest `mpy-cross` whose version matches the version of CircuitPython you are using.

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

How do I check how much memory I have free?

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 ()!

As of CircuitPython 8.x we have started to support ESP32 and ESP32-C3 and have added a WiFi workflow for wireless coding! ()

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 AVRs 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. ().

You need to download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then download the latest bundle ().

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 5.x or earlier. Where can I find compatible libraries?

Adafruit is no longer building or supporting the CircuitPython 5.x or earlier library bundles. You are highly encourged to update CircuitPython to the latest version () and use the current version of the libraries (). However, if for some reason you cannot update, links to the previous bundles are available in the FAQ ().

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 ()installed. The Feather M0 Basic, Feather M0 Adalogger, and similar boards use a regular Arduino-compatible bootloader, which does not show a boardna meBOOT drive.

MakeCode

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

MacOS

DriveDx and its accompanything SAT SMART Driver can interfere with seeing the BOOT drive. See this forum post () for how to fix the problem.

Windows 10

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

Windows 7 or 8.1

To use a CircuitPython-compatible board with Windows 7 or 8.1, you must install a driver. Installation instructions are available here ().

It is recommended () that you upgrade to Windows 10 if possible; an upgrade is probably still free for you. Check here ().

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

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 boar dnameBOOT drive.

Let us know in the Adafruit support forums () 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.

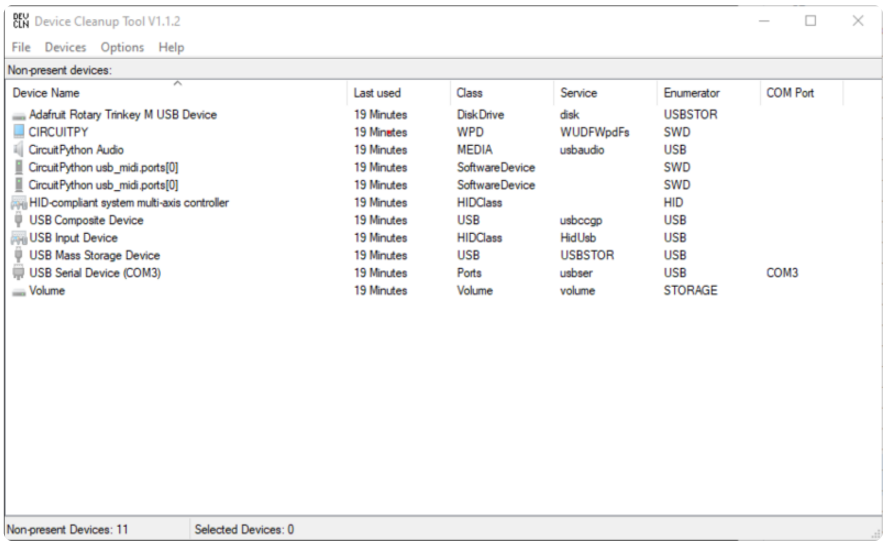
Norton anti-virus can interfere with CIRCUITPY. A user has reported this problem on Windows 7. The user turned off both Smart Firewall and Auto Protect, and CIRCUITPY then appeared.

Sophos Endpoint security software can cause CIRCUITPY to disappear () and the BOOT drive to reappear. It is not clear what causes this behavior.

Device Errors or Problems on Windows

Windows can become confused about USB device installations. This is particularly true of Windows 7 and 8.1. It is recommended () that you upgrade to Windows 10 if possible; an upgrade is probably still free for you: see this link ().

If not, try cleaning up your USB devices. Use Uwe Sieber's Device Cleanup Tool () (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.



Non-present devices:

Device Name	Last used	Class	Service	Enumerator	COM Port
Adafruit Rotary Trinkey M USB Device	19 Minutes	DiskDrive	disk	USBSTOR	
CIRCUITPY	19 Minutes	WPD	WUDFWpdFs	SWD	
CircuitPython Audio	19 Minutes	MEDIA	usbaudio	USB	
CircuitPython usb_midi.ports[0]	19 Minutes	SoftwareDevice		SWD	
CircuitPython usb_midi.ports[0]	19 Minutes	SoftwareDevice		SWD	
HID-compliant system multi-axis controller	19 Minutes	HIDClass		HID	
USB Composite Device	19 Minutes	USB	usbccgp	USB	
USB Input Device	19 Minutes	HIDClass	HidUsb	USB	
USB Mass Storage Device	19 Minutes	USB	USBSTOR	USB	
USB Serial Device (COM3)	19 Minutes	Ports	usbser	USB	COM3
Volume	19 Minutes	Volume	volume	STORAGE	

Non-present Devices: 11   Selected Devices: 0

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

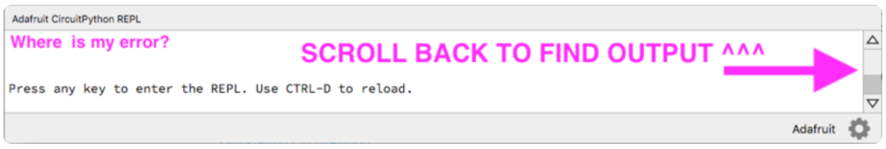
There are times when the serial console will accurately not display anything, such as, when no code is currently running, or when code with no serial output is already running before you open the console. However, if you find yourself in a situation where you feel it should be displaying something like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



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

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

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 " ()Acronis Managed Machine Service Mini" ().

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.disable_autoreload()
```

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.

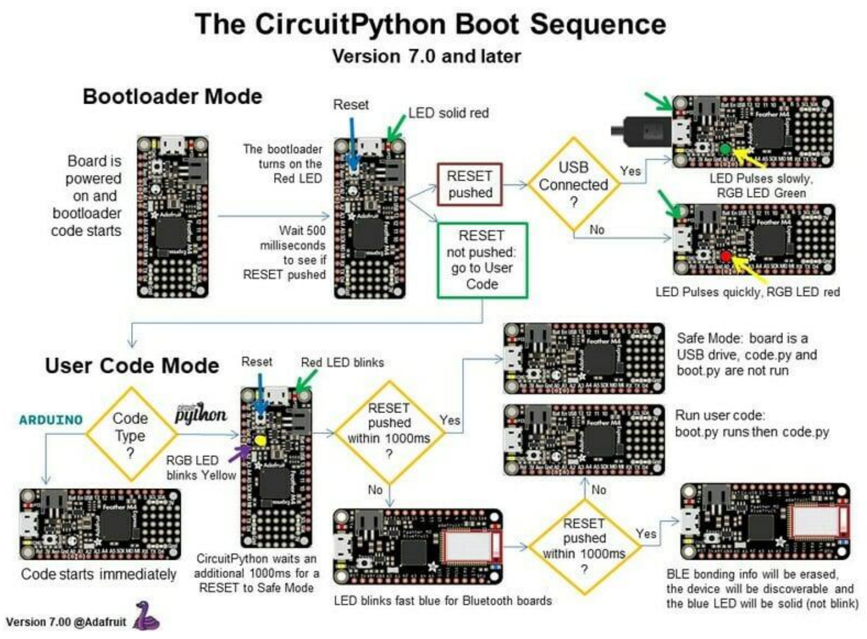
On start up, the LED will blink YELLOW multiple times for 1 second. Pressing the RESET button (or on Espressif, the BOOT button) during this time will restart the board and then enter safe mode. On Bluetooth capable boards, after the yellow blinks, there will be a set of faster blue blinks. Pressing 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.


The CircuitPython Boot Sequence
Version 7.0 and later
Bootloader Mode
Board is powered on and bootloader code starts
Reset
LED solid red
The bootloader turns on the Red LED
Wait 500 milliseconds to see if RESET pushed
RESET pushed
RESET not pushed: go to User Code
USB Connected ?
Yes
No
LED Pulses slowly, RGB LED Green
LED Pulses quickly, RGB LED red
User Code Mode
ARDUINO
Code Type ?
python
Reset
Red LED blinks
RGB LED blinks Yellow
Code starts immediately
CircuitPython waits an additional 1000ms for a RESET to Safe Mode
RESET pushed within 1000ms ?
Yes
No
Safe Mode: board is a USB drive, code.py and boot.py are not run
Run user code: boot.py runs then code.py
LED blinks fast blue for Bluetooth boards
RESET pushed within 1000ms ?
Yes
No
BLE bonding info will be erased, the device will be discoverable and the blue LED will be solid (not blink)
Version 7.00 @Adafruit

CircuitPython 6.3.0 and earlier

Here's what the colors and blinking mean:

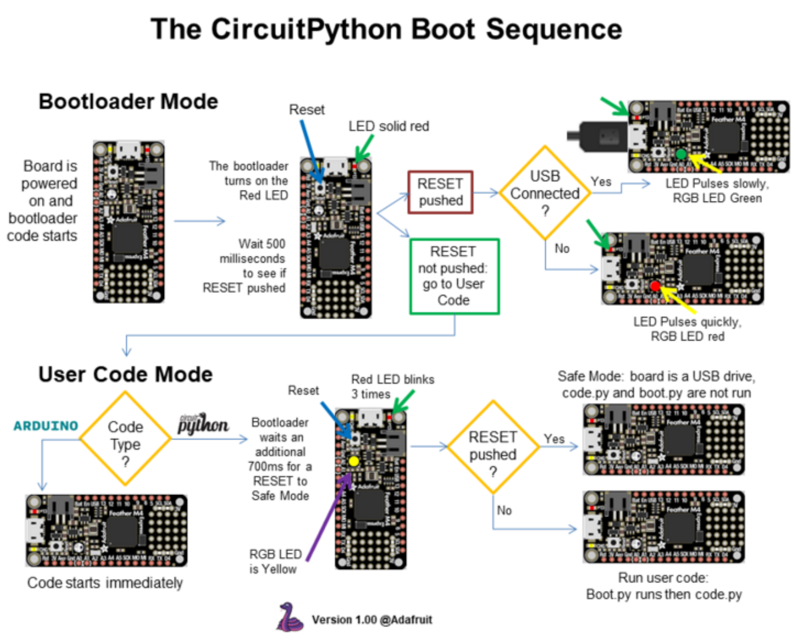
- steady GREEN: code.py (or code.txt, main.py, or main.txt) is running
- pulsing GREEN: code.py (etc.) has finished or does not exist
- steady YELLOW at start up: (4.0.0-alpha.5 and newer) CircuitPython is waiting for a reset to indicate that it should start in safe mode
- pulsing YELLOW: Circuit Python 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
Bootloader Mode
Board is powered on and bootloader code starts
Reset
LED solid red
The bootloader turns on the Red LED
Wait 500 milliseconds to see if RESET pushed
RESET pushed
RESET not pushed: go to User Code
USB Connected ?
Yes
No
LED Pulses slowly, RGB LED Green
LED Pulses quickly, RGB LED red
User Code Mode
ARDUINO
Code Type ?
python
Code starts immediately
Reset
Red LED blinks 3 times
Bootloader waits an additional 700ms for a RESET to Safe Mode
RGB LED is Yellow
RESET pushed ?
Yes
No
Safe Mode: board is a USB drive, code.py and boot.py are not run
Run user code: Boot.py runs then code.py
Version 1.00 @Adafruit

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 ().

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_NAM E. 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

To enter safe mode when using CircuitPython 7.x, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED will blink yellow during that time. If you press reset during that 1000ms, 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

To enter safe mode when using CircuitPython 6.x, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 700ms. On some boards, the onboard status LED (highlighted in green above) will turn solid yellow during this time. If you press reset during that 700ms, 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 () to do this.

1. Connect to the CircuitPython REPL () using Mu or a terminal program.
2. Type the following into the REPL:

```
&gt;&gt;&gt; import storage
&gt;&gt;&gt; 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

Feather M0 Express

Feather M4 Express

Metro M0 Express

Metro M4 Express QSPI Eraser

Trellis M4 Express (QSPI)

Grand Central M4 Express (QSPI)

PyPortal M4 Express (QSPI)

Circuit Playground Bluefruit (QSPI)

Monster M4SK (QSPI)

PyBadge/PyGamer QSPI Eraser.UF2

CLUE_Flash_Erase.UF2

Matrix_Portal_M4_(QSPI).UF2

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 () .uf2 file to the boardnameBOOT drive.

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

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

If you haven't already downloaded the latest release of CircuitPython for your board, check out the installation page (). You'll also need to 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 Trinkey 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

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

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

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

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

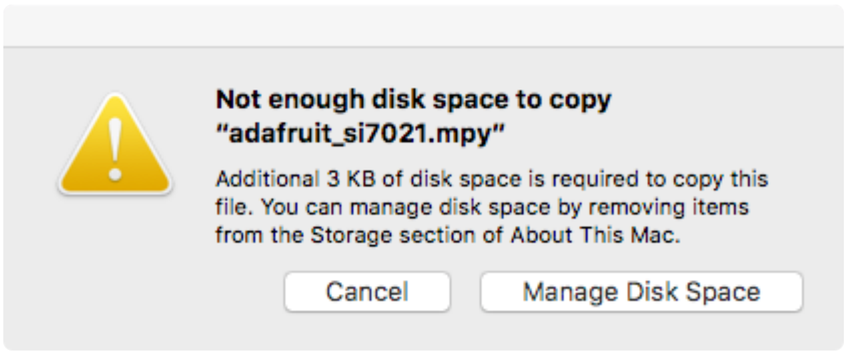
Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that do not have a UF2 bootloader include the Feather M0 Basic Proto, Feather Adalogger, or the Arduino Zero.

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` (), 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.


Not enough disk space to copy "adafruit_si7021.mpy"

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

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 () 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:

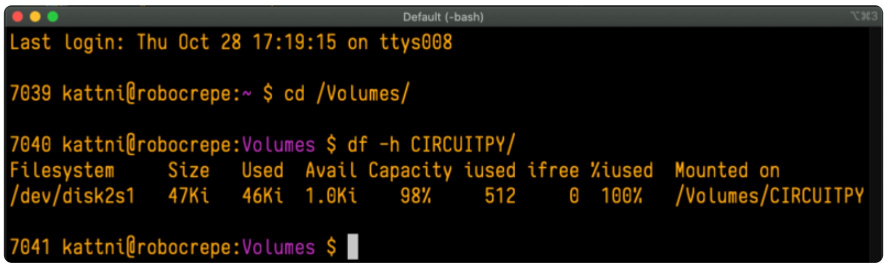
```
cp -rX folder_to_copy /Volumes/CIRCUITPY
```

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/
```

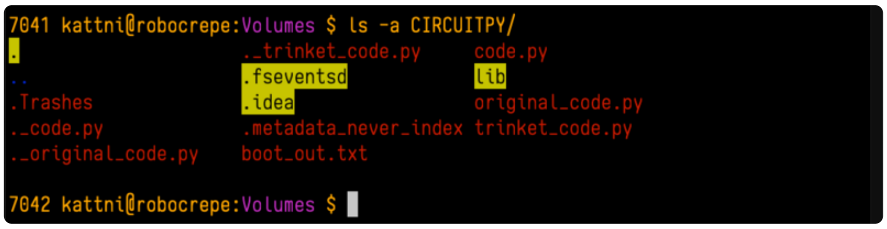
Other MacOS Space-Saving Tips

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.



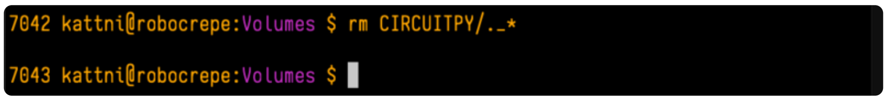
```
Last login: Thu Oct 28 17:19:15 on ttys008

7039 kattni@robocrepe:~ $ cd /Volumes/

7040 kattni@robocrepe:Volumes $ df -h CIRCUITPY/
Filesystem     Size   Used  Avail Capacity iused ifree %iused  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!



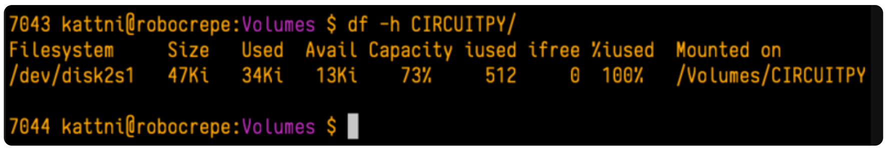
```
7041 kattni@robocrepe:Volumes $ ls -a CIRCUITPY/
.                     ._trinket_code.py      code.py
..                    .fseventsd             lib
.Trashes              .idea                  original_code.py
._code.py             .metadata_never_index  trinket_code.py
._original_code.py    boot_out.txt

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 %iused  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.

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

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

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

Refer to the diagrams above for boot sequence details.

"Uninstalling" CircuitPython

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

Maybe you tried CircuitPython and want to go back to MakeCode or Arduino? Not a problem. You can always remove or reinstall CircuitPython whenever you want! Heck, you can change your mind every day!

There is nothing to uninstall. CircuitPython is "just another program" that is loaded onto your board. You simply load another program (Arduino or MakeCode) and it will overwrite CircuitPython.

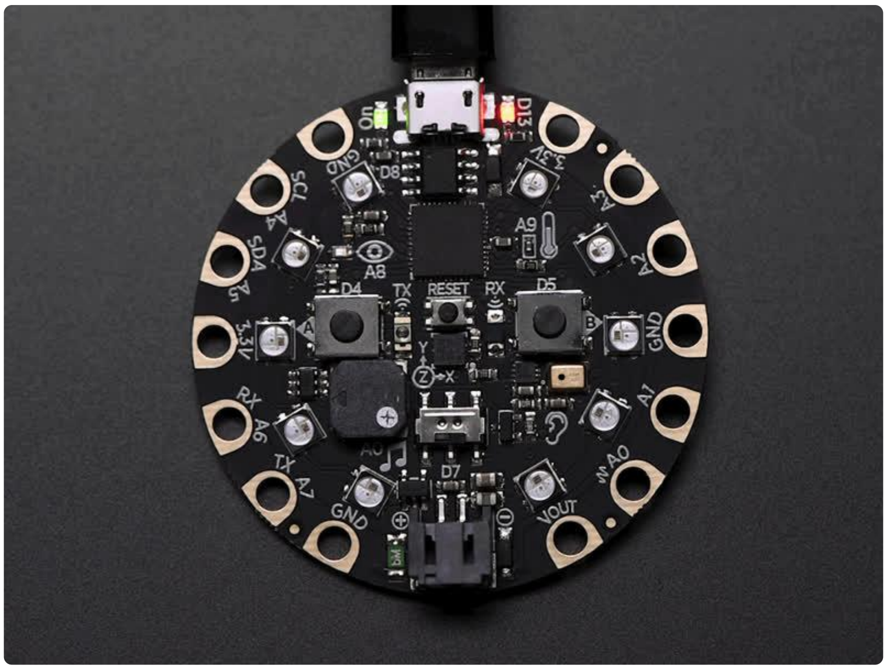
Backup Your Code

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

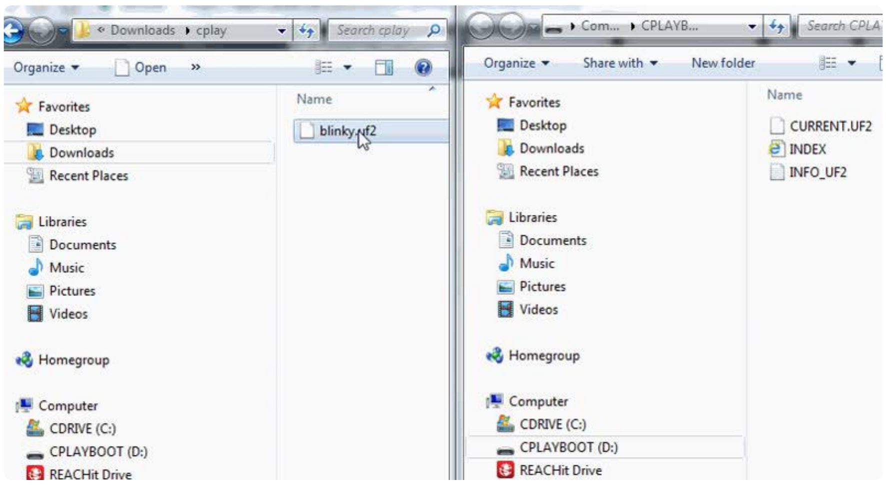
Moving Circuit Playground Express to MakeCode

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

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



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



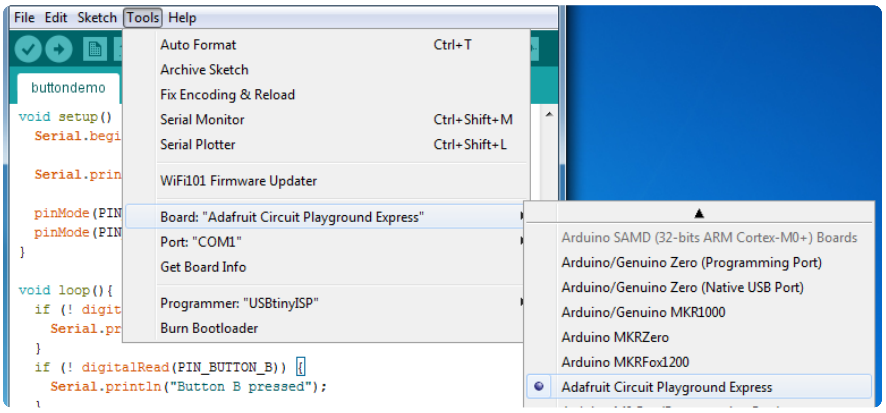
Your MakeCode is now running and CircuitPython has been removed. Going forward you only have to single click the reset button to get to CPLAYBOOT. This is an idiosyncrasy of MakeCode.

Moving to Arduino

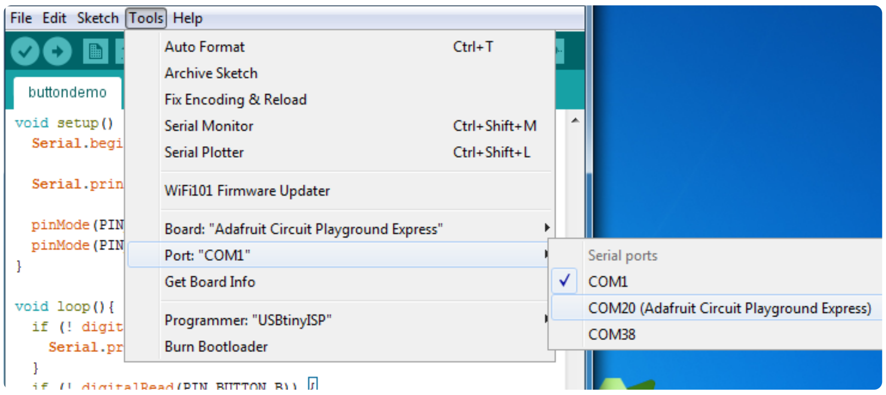
If you want to use Arduino instead, you just use the Arduino IDE to load an Arduino program. Here's an example of uploading a simple "Blink" Arduino program, but you don't have to use this particular program.

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

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


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

Select the correct matching Port:



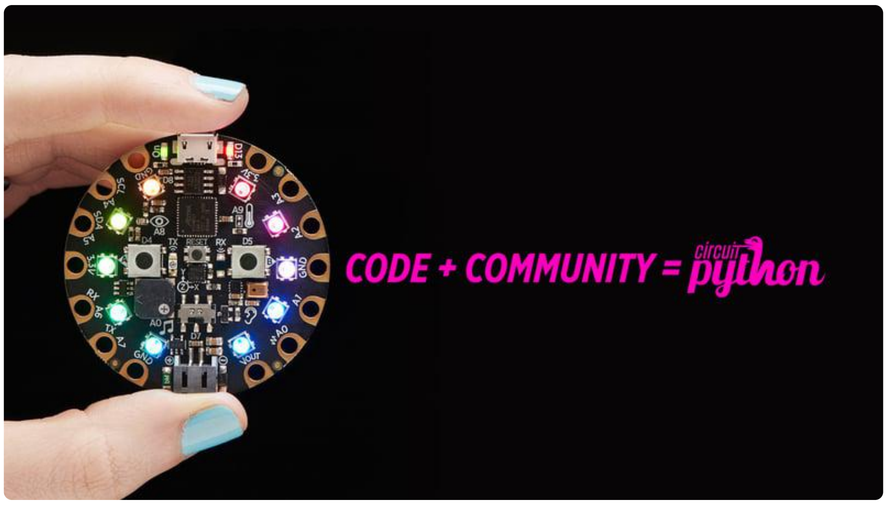
Create a new simple Blink sketch example:

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

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Make sure the LED(s) are still green, then click Upload to upload Blink. Once it has uploaded successfully, the serial Port will change so re-select the new Port!

Once Blink is uploaded you should no longer need to double-click to enter bootloader mode. Arduino will automatically reset when you upload.

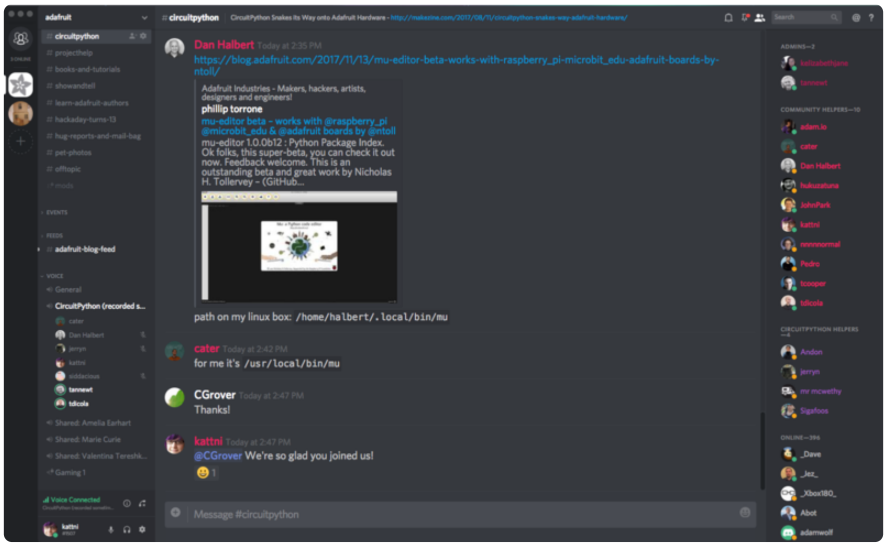
Welcome to the Community!



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


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

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

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

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

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

CGrover Today at 2:47 PM
Thanks!

kattni Today at 2:47 PM
@CGrover We're so glad you joined us!
😀 1

Message #circuitpython

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.

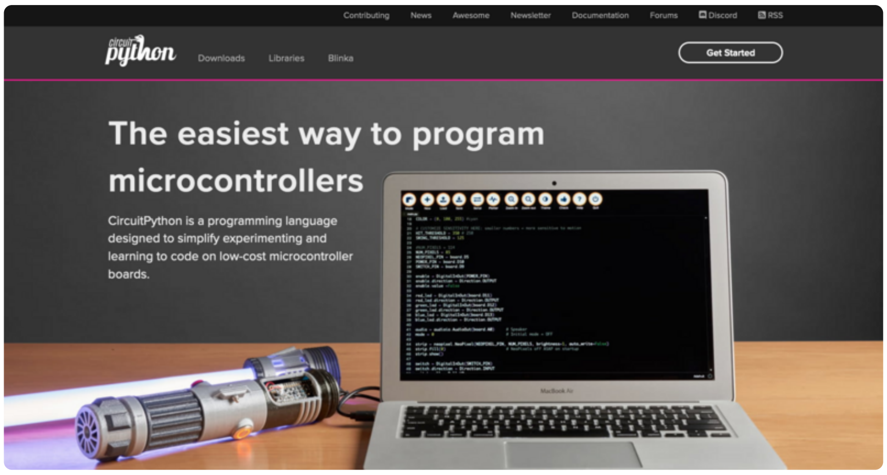
The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. Your contributions are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

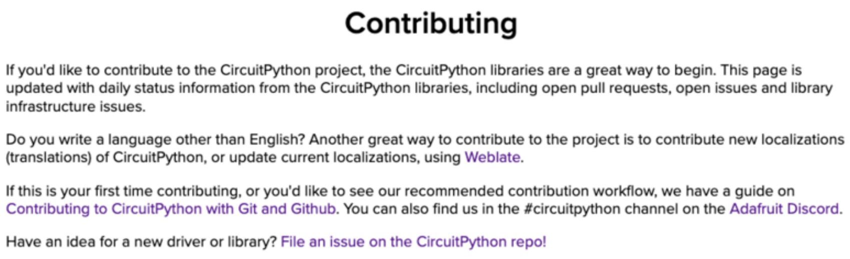
CircuitPython.org



The easiest way to program microcontrollers

CircuitPython is a programming language designed to simplify experimenting and learning to code on low-cost microcontroller boards.

Beyond the Adafruit Learn System, which you are viewing right now, the best place to find information about CircuitPython is circuitpython.org (). Everything you need to get started with your new microcontroller and beyond is available. You can do things like download CircuitPython for your microcontroller () or download the latest CircuitPython Library bundle (), or check out which single board computers support Blinka (). 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 ().



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 (). 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:

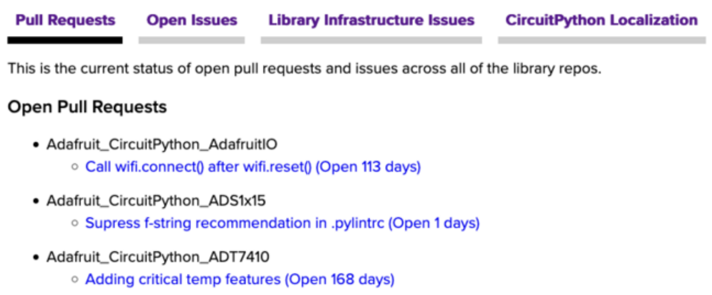
Current Status for Tue, Nov 02, 2021

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.



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

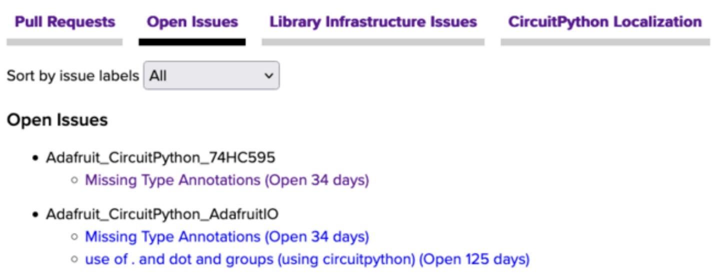
Open Pull Requests

- 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.


Pull Requests | Open Issues | Library Infrastructure Issues | CircuitPython Localization

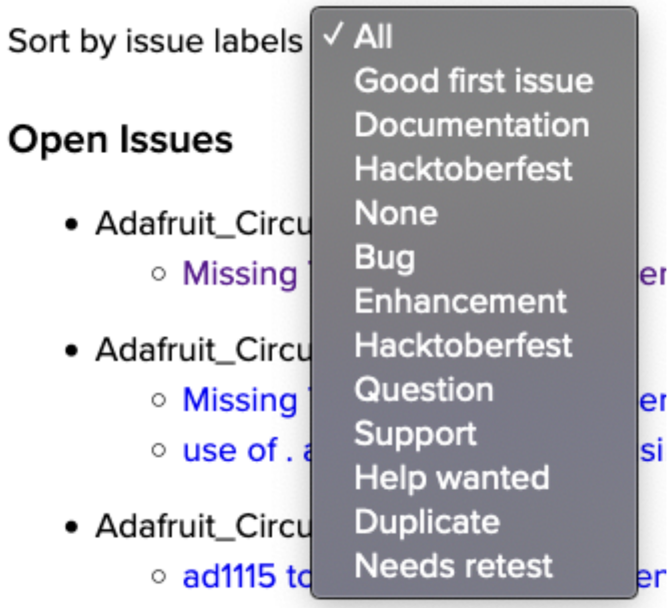
Sort by issue labels: All

Open Issues

- Adafruit_CircuitPython_74HC595
  - Missing Type Annotations (Open 34 days)
- Adafruit_CircuitPython_AdafruitIO
  - Missing Type Annotations (Open 34 days)
  - use of . and dot and groups (using circuitpython) (Open 125 days)

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.



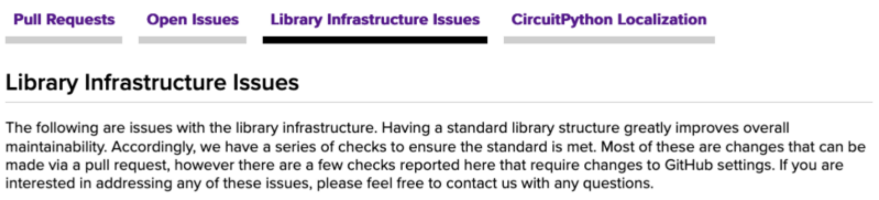
If you're new to everything, new to contributing to open source, or new to contributing to the CircuitPython project, you can choose "Good first issue". Issues with that label are well defined, with a finite scope, and are intended to be easy for someone new to figure out.

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 () 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.



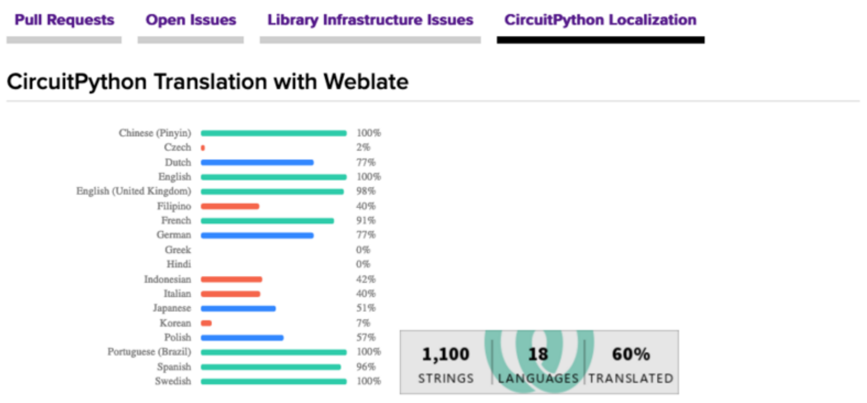
Library Infrastructure Issues

The following are issues with the library infrastructure. Having a standard library structure greatly improves overall maintainability. Accordingly, we have a series of checks to ensure the standard is met. Most of these are changes that can be made via a pull request, however there are a few checks reported here that require changes to GitHub settings. If you are interested in addressing any of these issues, please feel free to contact us with any questions.

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.



CircuitPython Translation with Weblate

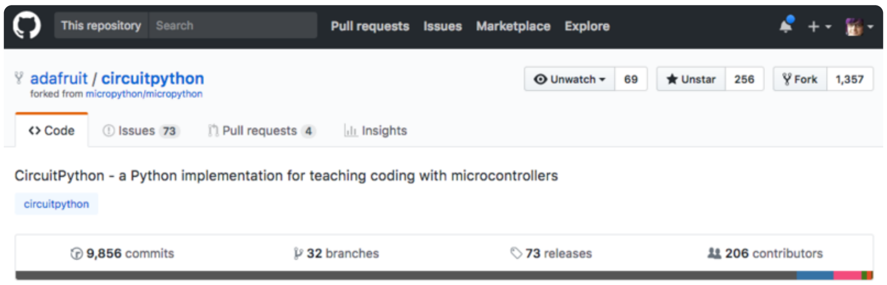
Chinese (Pinyin)	100%
Czech	2%
Dutch	77%
English	100%
English (United Kingdom)	98%
Filipino	40%
French	91%
German	77%
Greek	0%
Hindi	0%
Indonesian	42%
Italian	40%
Japanese	51%
Korean	7%
Polish	57%
Portuguese (Brazil)	100%
Spanish	96%
Swedish	100%

1,100	18	60%
STRINGS	LANGUAGES	TRANSLATED

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 () is an excellent place to start!

Adafruit GitHub



adafruit / circuitpython
forked from micropython/micropython

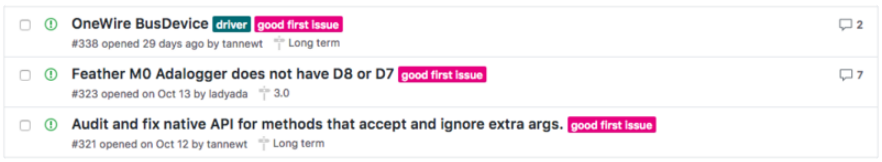
CircuitPython - a Python implementation for teaching coding with microcontrollers

circuitpython

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

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

If you're new to GitHub or programming in general, there are great opportunities for you. For the CircuitPython core, head over to the CircuitPython repository on GitHub, click on "Issues ()", and you'll find a list that includes issues labeled "good first issue ()" . For the libraries, head over to the Contributing page Issues list (), and use the drop down menu to search for "good first issue ()". 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 ().



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

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

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

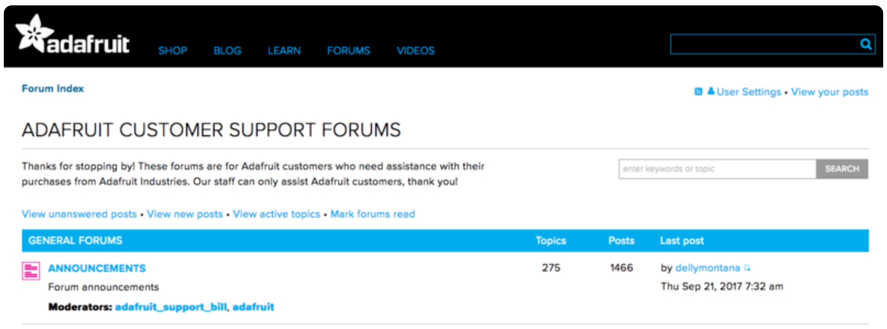
Already experienced and looking for a challenge? Checkout the rest of either issues list and you'll find plenty of ways to contribute. You'll find all sorts of things, from new driver requests, to library bugs, to core module updates. There's plenty of opportunities for everyone at any level!

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 (). 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


adafruit
SHOP
BLOG
LEARN
FORUMS
VIDEOS
Forum Index
User Settings • View your posts
enter keywords or topic
SEARCH
View unanswered posts • View new posts • View active topics • Mark forums read

ADAFRUIT CUSTOMER SUPPORT FORUMS

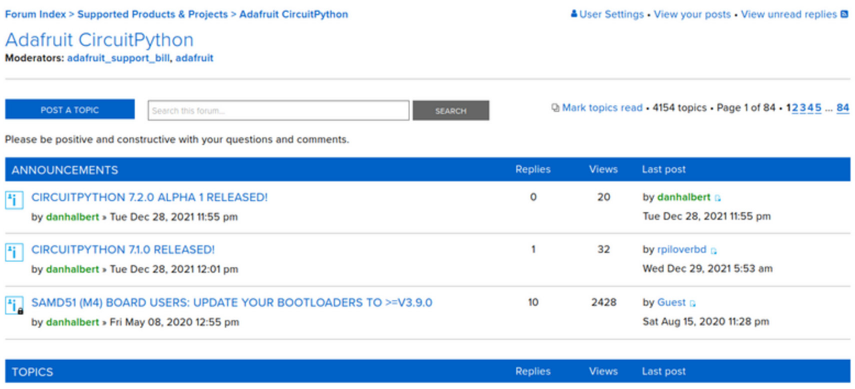
Thanks for stopping by! These forums are for Adafruit customers who need assistance with their purchases from Adafruit Industries. Our staff can only assist Adafruit customers, thank you!

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

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

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

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


Forum Index > Supported Products & Projects > Adafruit CircuitPython

User Settings • View your posts • View unread replies

Adafruit CircuitPython

Moderators: adafruit_support_bill, adafruit

POST A TOPIC | Search this forum... | SEARCH

Mark topics read • 4154 topics • Page 1 of 84 • 1 2 3 4 5 ... 84

Please be positive and constructive with your questions and comments.

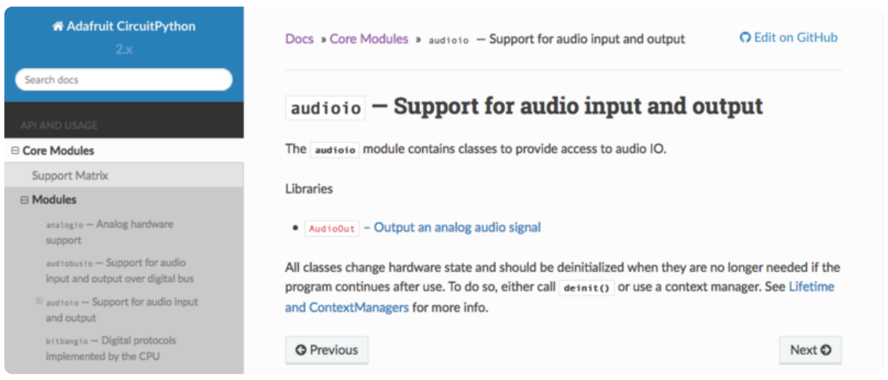
ANNOUNCEMENTS	Replies	Views	Last post
CIRCUITPYTHON 7.2.0 ALPHA 1 RELEASED! by danhalbert » Tue Dec 28, 2021 11:55 pm	0	20	by danhalbert Tue Dec 28, 2021 11:55 pm
CIRCUITPYTHON 7.1.0 RELEASED! by danhalbert » Tue Dec 28, 2021 12:01 pm	1	32	by rpiloverbd Wed Dec 29, 2021 5:53 am
SAMD51 (M4) BOARD USERS: UPDATE YOUR BOOTLOADERS TO >=V3.9.0 by danhalbert » Fri May 08, 2020 12:55 pm	10	2428	by Guest Sat Aug 15, 2020 11:28 pm

TOPICS	Replies	Views	Last post

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


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

`audioio` — Support for audio input and output

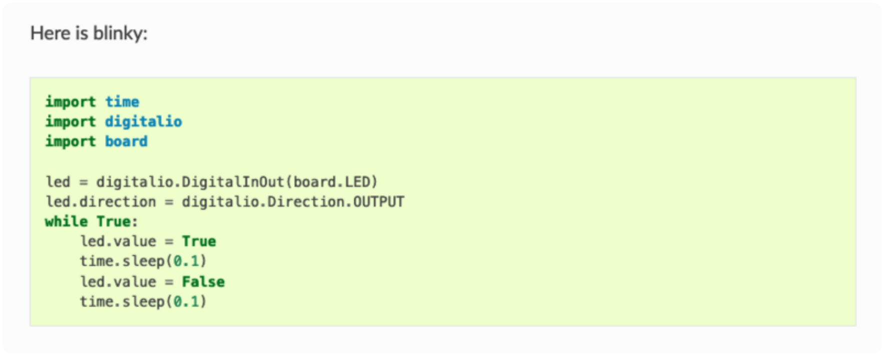
The `audioio` module contains classes to provide access to audio IO.

Libraries

- `AudioOut` – Output an analog audio signal

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

Read the Docs () is a 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 () 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)
```

CircuitPython Essentials

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

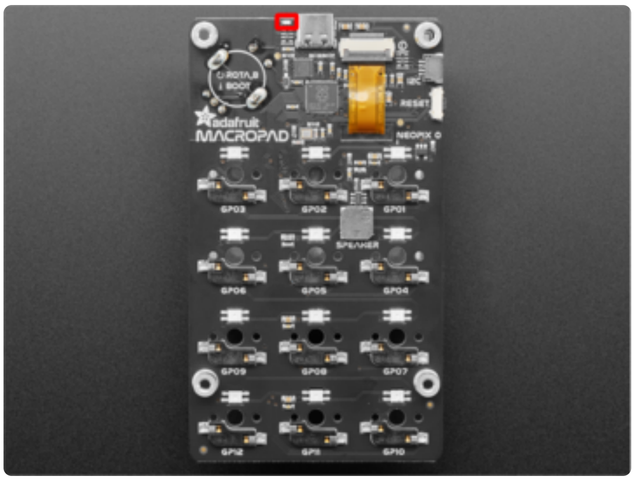
There are a number of core modules built into CircuitPython, which can be used along side the many CircuitPython libraries available. The following pages demonstrate some of these modules. Each page presents a different concept including a code example with an explanation. All of the examples are designed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

Time to get started learning the CircuitPython essentials!

Blink

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

LED Location

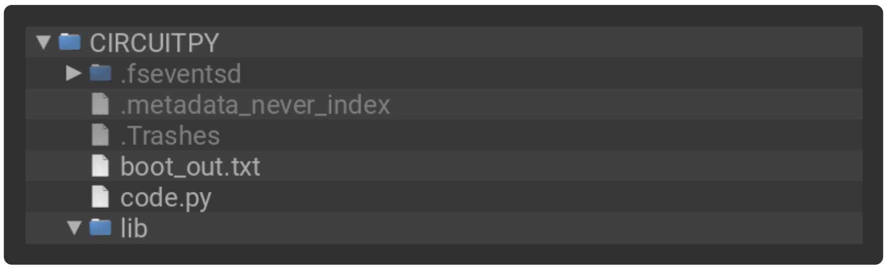


The red LED is located on the top edge of the back of the board, to the left of the USB connector.

Blinking an LED

In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the directory CircuitPython_Templates/blink/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

Your CIRCUITPY drive should now look similar to the following image:



```
▼ CIRCUITPY
    ▶ .fseventsd
      .metadata_never_index
      .Trashes
      boot_out.txt
      code.py
    ▼ lib
```

```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""CircuitPython Blink Example - the CircuitPython 'Hello, World!'"""
import time
import board
import digitalio

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

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

The built-in LED begins blinking!

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

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

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

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

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

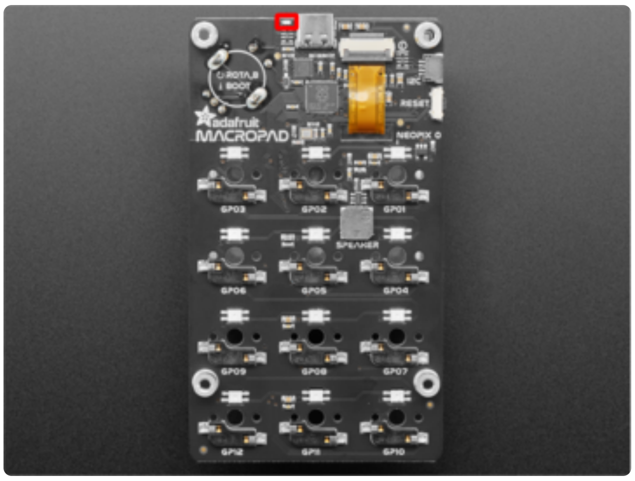
With only a small update, you can control the blink speed. The blink speed is controlled by the amount of time you tell the code to wait before moving on using `time.sleep()`. The example uses `0.5`, which is one half of one second. Try increasing or decreasing these values to see how the blinking changes.

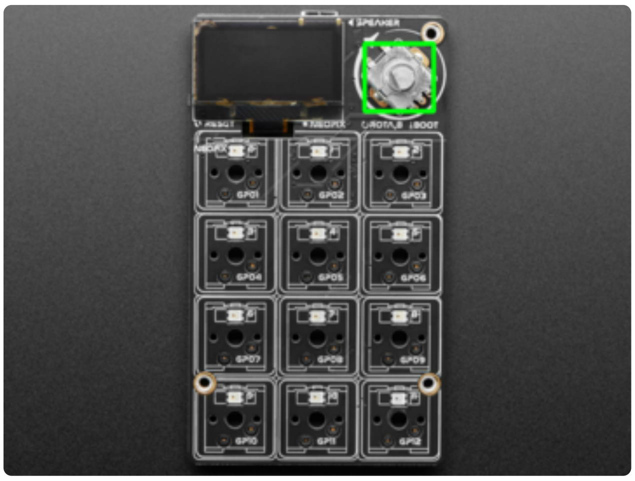
That's all there is to blinking an LED using CircuitPython!

Digital Input

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

LED and Button





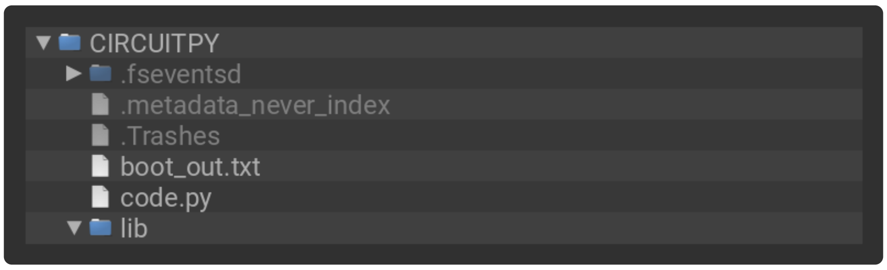
The red LED (highlighted in red) is located on the top edge of the back of the board, to the left of the USB connector.

The button (highlighted in green) is located in the rotary encoder - to use the button, simply press down on the rotary encoder.

Controlling the LED with a Button

In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the directory Adafruit_MacroPad/Digital_Input/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

Your CIRCUITPY drive should now look similar to the following image:



```
▼ CIRCUITPY
  ▶ .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
  ▼ lib
```

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

"""CircuitPython Digital Input example for MacroPad"""
import board
import digitalio

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

button = digitalio.DigitalInOut(board.BUTTON)
button.switch_to_input(pull=digitalio.Pull.UP)

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

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

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

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

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

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

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

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

Keypad

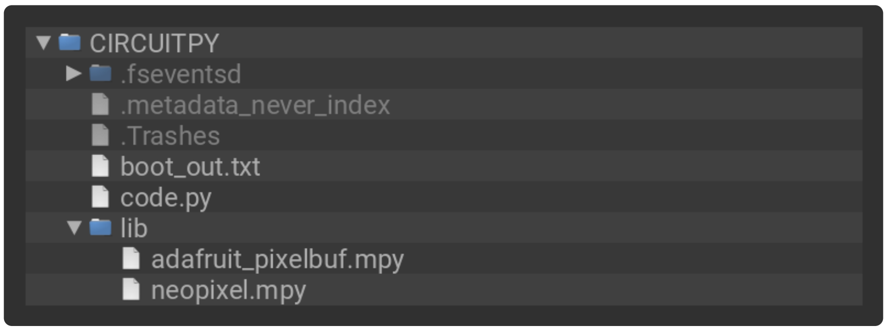
To use the keypad module, you must be running at least CircuitPython 7.0.0-alpha.4!

Using the keys on the Adafruit MacroPad in CircuitPython is super simple, thanks to the `keypad` module. This module allows you to print the key number, and read key press and releases. The `rotaryio` module allows you to read the rotation of the rotary encoder, and `digitalio` allows you to read the rotary encoder button switch presses. All of these modules are built into CircuitPython, meaning to use them, you do not need to load any separate libraries onto your MacroPad.

However, the following example involves the NeoPixel LEDs, which do require a separate library - Adafruit CircuitPython NeoPixel.

Save the following to your CIRCUITPY drive as code.py.

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


```
CIRCUITPY
    .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    lib
        adafruit_pixelbuf.mpy
        neopixel.mpy
```

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

"""Keypad and rotary encoder example for Adafruit MacroPad"""
import board
import digitalio
import rotaryio
import neopixel
import keypad
from rainbowio import colorwheel


key_pins = (board.KEY1, board.KEY2, board.KEY3, board.KEY4, board.KEY5, board.KEY6,
            board.KEY7, board.KEY8, board.KEY9, board.KEY10, board.KEY11,
board.KEY12)
keys = keypad.Keys(key_pins, value_when_pressed=False, pull=True)
```

```
encoder = rotaryio.IncrementalEncoder(board.ROTA, board.ROTB)
button = digitalio.DigitalInOut(board.BUTTON)
button.switch_to_input(pull=digitalio.Pull.UP)

pixels = neopixel.NeoPixel(board.NEOPIXEL, 12, brightness=0.2)

last_position = None
while True:
    if not button.value:
        pixels.brightness = 1.0
    else:
        pixels.brightness = 0.2

    position = encoder.position
    if last_position is None or position != last_position:
        print("Rotary:", position)
    last_position = position

    color_value = (position * 2) % 255

    event = keys.events.get()
    if event:
        print(event)
        if event.pressed:
            pixels[event.key_number] = colorwheel(color_value)
        else:
            pixels[event.key_number] = 0
```

Now try pressing any of the keys to see a message printed out. The corresponding NeoPixel will light up. To change the color of the NeoPixel and see a value printed out, rotate the rotary encoder. To temporarily increase the brightness of the NeoPixel, press down on the rotary encoder.

Note that the key numbers start at 0, so the printed key numbers are 0-11. The CircuitPython pin names are KEY1 - KEY12. KEY1 is key number 0, KEY2 is key number 1, etc, through KEY12 being key number 11.

MacroPad CircuitPython Library

The Adafruit MacroPad has a number of great features, all of which work great with CircuitPython. The Adafruit CircuitPython MacroPad () library wraps all of those features into one place to make it super simple to get started using CircuitPython with your Adafruit MacroPad.

This section provides a few examples of using the MacroPad CircuitPython library to read key presses, the rotary encoder values and rotary encoder switch state, and to send HID and MIDI commands.

The MacroPad library is easy to use. Simply import it, and then create an instance of it in your code. To do this, you include the following two lines at the beginning of your program.

```
from adafruit_macropad import MacroPad

macropad = MacroPad()
```

Then, you're ready to access all the features of the library using `macropad`. Each of the examples will show you how to access different features of the library.

First, you'll want to install the MacroPad library and its dependencies.

MacroPad Library Installation

To use the MacroPad library, you'll need to install it and a few other CircuitPython libraries on your CIRCUITPY drive.

There are two ways to get the necessary libraries onto your CIRCUITPY drive. You can click the Download Project Bundle button at the top of each example, open the 7.x folder within, and copy the code.py file and the lib folder to your CIRCUITPY drive.

Alternatively, you can follow the instructions below.

Download the latest Adafruit CircuitPython Bundle that matches the version of CircuitPython you're using.

Download the latest CircuitPython Library Bundle

Extract the zip and navigate to the lib folder found within. Drag the necessary libraries from the zip lib folder to the lib folder on your CIRCUITPY drive.

At a minimum, the following libraries are required to use the MacroPad CircuitPython library. Drag the following files and folders to the lib folder on your CIRCUITPY drive:

- adafruit_macropad.mpy - A helper library for using the features of the Adafruit MacroPad.
- adafruit_debouncer.mpy - A helper library for debouncing pins. Used to provide a debounced instance of the rotary encoder switch.

- adafruit_simple_text_display.mpy - A helper library for easily displaying lines of text on a display.
- neopixel.mpy - A CircuitPython driver for NeoPixel LEDs.
- adafruit_display_text/ - A library to display text using `displayio`. Used for the text display functionality of the MacroPad library that allows you easily display lines of text on the built-in display.
- adafruit_hid/ - CircuitPython USB HID drivers.
- adafruit_midi/ - A CircuitPython helper for encoding/decoding MIDI packets over a MIDI or UART connection
- adafruit_ticks.mpy - A helper to work with intervals and deadlines in milliseconds

There is an example included that uses a library that is not required for the MacroPad library to work, but provides a convenient way to layout text in grid. The following library is recommended as well:

- adafruit_displayio_layout - A library that includes a grid layout helper.

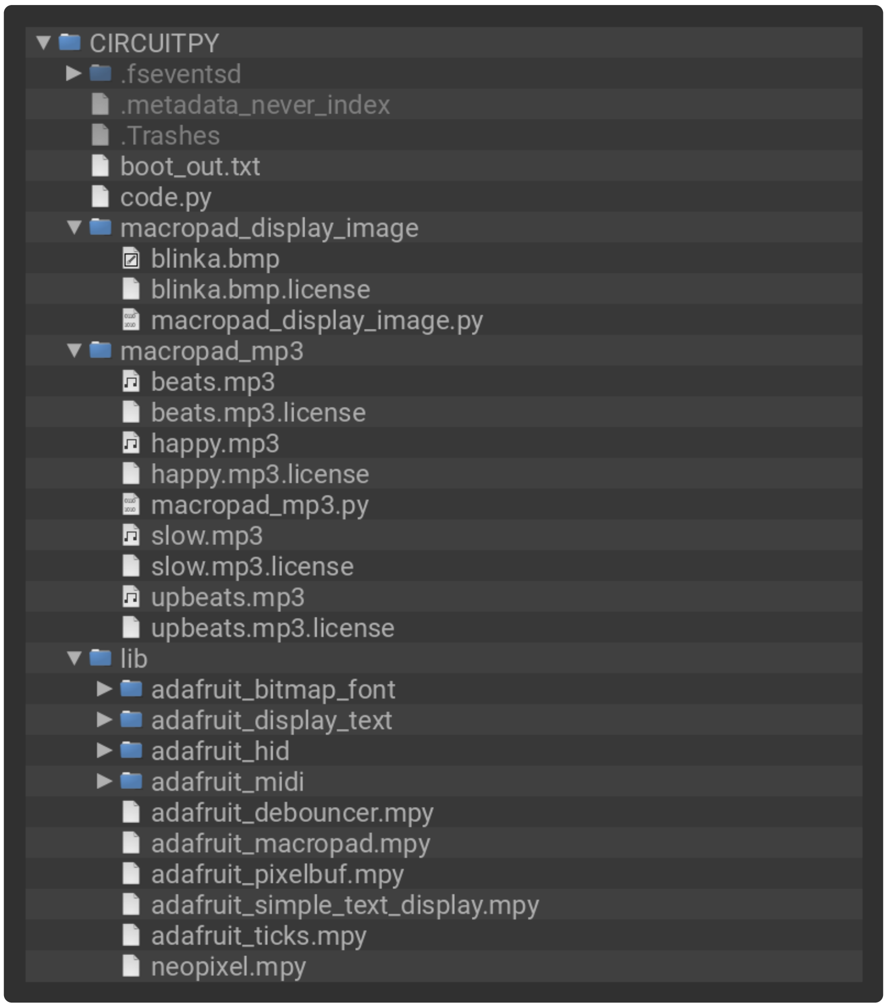
MacroPad Basics

The Adafruit MacroPad RP2040 features a 3x4 key pad with NeoPixel LEDs, and a rotary encoder with push switch. This example reads the key presses, the relative position of the rotary encoder and the state of the rotary encoder switch, and prints the information to the serial console.

Update your code.py to the following.

Click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the folder that matches your CircuitPython version, and copy the entire lib folder and the code.p y file to your CIRCUITPY drive.

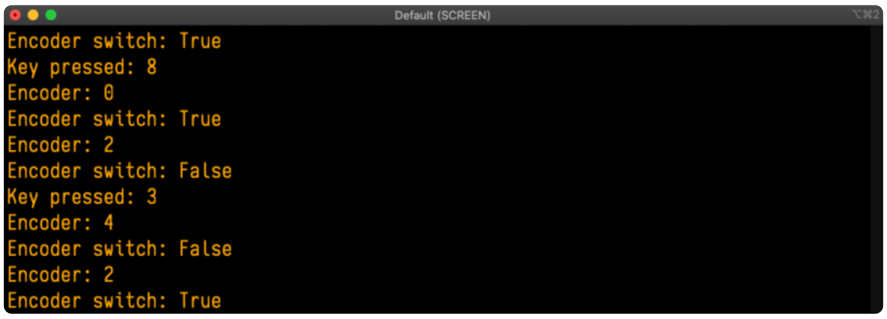
Your CIRCUITPY drive should now look similar to the following image:



```
CIRCUITPY
    .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    macropad_display_image
        blinka.bmp
        blinka.bmp.license
        macropad_display_image.py
    macropad_mp3
        beats.mp3
        beats.mp3.license
        happy.mp3
        happy.mp3.license
        macropad_mp3.py
        slow.mp3
        slow.mp3.license
        upbeats.mp3
        upbeats.mp3.license
    lib
        adafruit_bitmap_font
        adafruit_display_text
        adafruit_hid
        adafruit_midi
        adafruit_debouncer.mpy
        adafruit_macropad.mpy
        adafruit_pixelbuf.mpy
        adafruit_simple_text_display.mpy
        adafruit_ticks.mpy
        neopixel.mpy
```

```
# SPDX-FileCopyrightText: Copyright (c) 2021 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
Simpletest demo for MacroPad. Prints the key pressed, the relative position of the
rotary
encoder, and the state of the rotary encoder switch to the serial console.
"""
import time
from adafruit_macropad import MacroPad

macropad = MacroPad()

while True:
    key_event = macropad.keys.events.get()
    if key_event and key_event.pressed:
        print("Key pressed: {}".format(key_event.key_number))
    print("Encoder: {}".format(macropad.encoder))
    print("Encoder switch: {}".format(macropad.encoder_switch))
    time.sleep(0.4)
```

Now, connect to the serial console (). Try pressing keys, rotating the rotary encoder, and pressing the rotary encoder switch to see the results printed out.



```
Encoder switch: True
Key pressed: 8
Encoder: 0
Encoder switch: True
Encoder: 2
Encoder switch: False
Key pressed: 3
Encoder: 4
Encoder switch: False
Encoder: 2
Encoder switch: True
```

To use the MacroPad library, you need to import it and instantiate it with the following code:

```
from adafruit_macropad import MacroPad

macropad = MacroPad()
```

Once, instantiated as `macropad`, you have access to all the features of the MacroPad library. To use the features of the library, you include `macropad.feature_name` in your code. This example uses the following features:

- `keys` - The keys on the MacroPad. Uses events to track key number and state, e.g. pressed or released. You must fetch the events using `keys.events.get()` and then the events are available for usage in your code. Each event has three properties: `key_number`, `pressed`, and `released`.
- `encoder` - The rotary encoder relative rotation position. Always begins at 0 when the code is run, so the value returned is relative to the initial location.
- `encoder_switch` - The rotary encoder switch. Returns `True` when pressed.

Therefore, to read the rotary encoder, you would include `macropad.encoder` in your code.

In this example, you first import `time`, then the MacroPad library, and instantiate the library as `macropad`.

Inside the loop, the first thing you do is setup to look for the key press by creating the `key_event` variable and assigning it to `macropad.keys.events.get()`. Then, you check to see if there is a `key_event` (i.e. a key being pressed) and if it is a key being pressed (`key_event.pressed`). Then, if so, print the key number (`key_event.key_number`) being pressed to the serial console.

Then, you print to the serial console the relative position of the rotary encoder (with `macropad.encoder`) and the state of the encoder switch (with `macropad.encoder_switch`).

Finally, you include a `time.sleep(0.4)` to print the information every 0.4 seconds to keep the serial console results readable.

That's all there is to reading the key presses, rotary encoder relative position, and rotary encoder switch state on the Adafruit MacroPad using the CircuitPython MacroPad library!

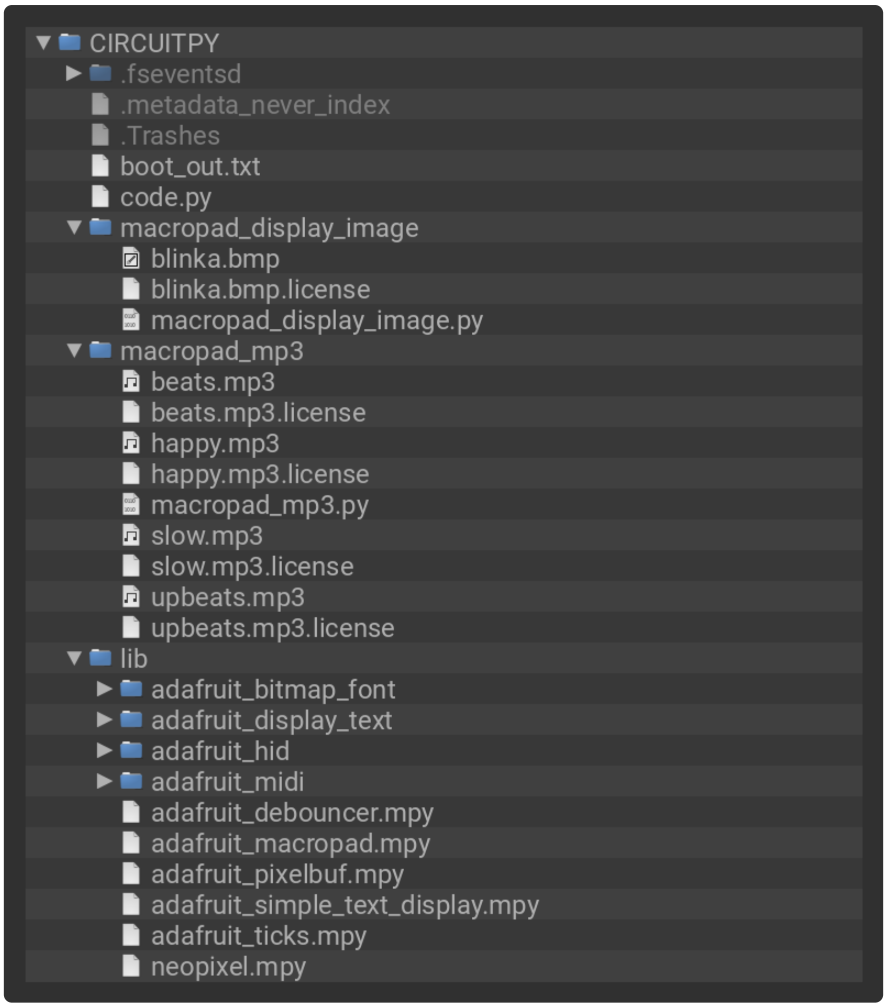
MacroPad Display Text

The Adafruit MacroPad RP2040 features a 3x4 key pad with NeoPixel LEDs, a rotary encoder with push switch, and a display. This example reads the key presses, the relative position of the rotary encoder and the state of the rotary encoder switch, and displays the information on the display.

Update your code.py to the following.

Click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the folder that matches your CircuitPython version, and copy the entire lib folder and the code.p y file to your CIRCUITPY drive.

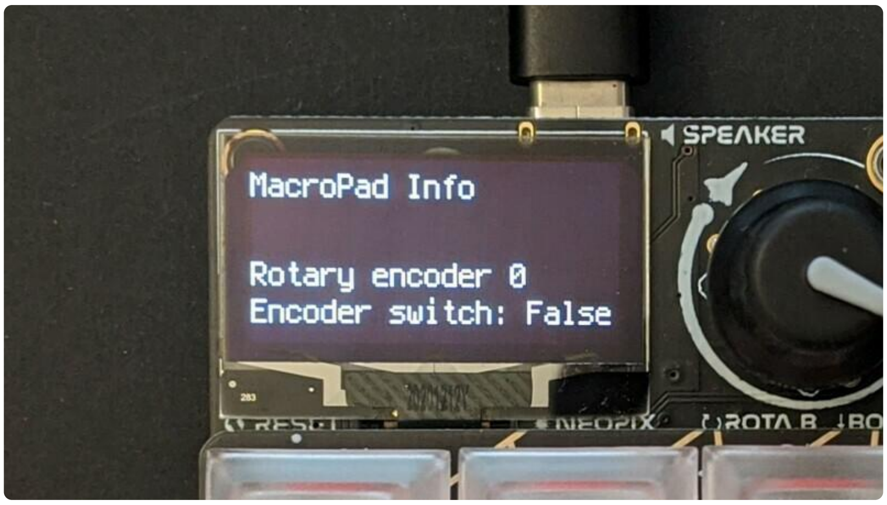
Your CIRCUITPY drive should now look similar to the following image:



```
CIRCUITPY
    .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    macropad_display_image
        blinka.bmp
        blinka.bmp.license
        macropad_display_image.py
    macropad_mp3
        beats.mp3
        beats.mp3.license
        happy.mp3
        happy.mp3.license
        macropad_mp3.py
        slow.mp3
        slow.mp3.license
        upbeats.mp3
        upbeats.mp3.license
    lib
        adafruit_bitmap_font
        adafruit_display_text
        adafruit_hid
        adafruit_midi
        adafruit_debouncer.mpy
        adafruit_macropad.mpy
        adafruit_pixelbuf.mpy
        adafruit_simple_text_display.mpy
        adafruit_ticks.mpy
        neopixel.mpy
```

```
# SPDX-FileCopyrightText: Copyright (c) 2021 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
Simpletest demo for MacroPad. Displays the key pressed, the relative position of
the rotary
encoder, and the state of the rotary encoder switch to the built-in display. Note
that the key
pressed line does not appear until a key is pressed.
"""
from adafruit_macropad import MacroPad

macropad = MacroPad()

text_lines = macropad.display_text(title="MacroPad Info")

while True:
    key_event = macropad.keys.events.get()
    if key_event and key_event.pressed:
        text_lines[0].text = "Key {} pressed!".format(key_event.key_number)
    text_lines[1].text = "Rotary encoder {}".format(macropad.encoder)
    text_lines[2].text = "Encoder switch: {}".format(macropad.encoder_switch)
    text_lines.show()
```

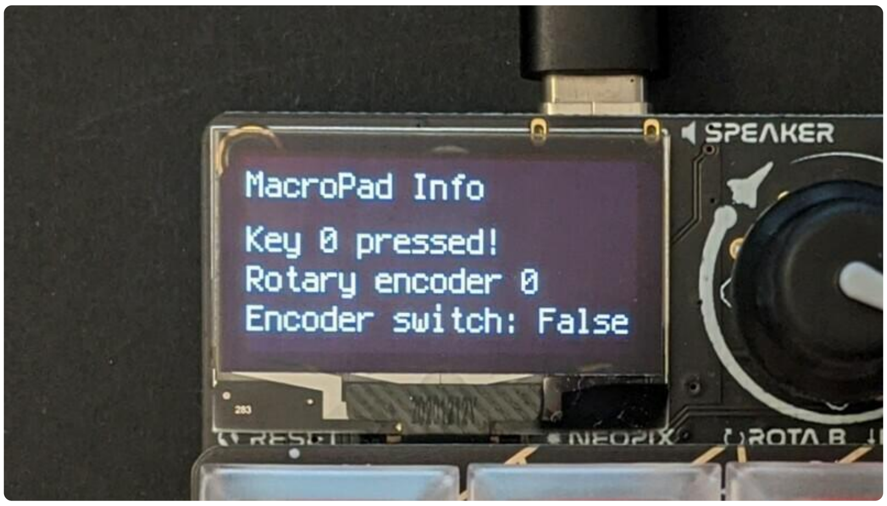
Now, check out the display!



If you rotate the rotary encoder, the number will change. It doesn't matter where the rotary encoder is, it will begin at 0. The number provided is a relative position.

If you press the rotary encoder switch down, it will display `True`.

Note that the key press line does not show up initially. Try pressing a key!



To use the `display_text` feature of the MacroPad library, you need to instantiate it by assigning it to a variable, e.g. `text_lines = macropad.display_text(title="MacroPad Info")`. Once created, the title cannot be updated. Note that if you want to be able to dynamically update the title, simply instantiate it without a tittle (`text_lines = macropad.display_text()`), and treat the first line of text as the title, which can be dynamically updated.

Once you've instantiated it, you can create lines of text below the title with dynamic information in them, such as the key number being pressed or the relative position of the rotary encoder. To do this, you use the `text_lines` object, and provide it a line number and a string to display. Remember, Python begins counting at 0. For example, to display a line of text with the rotary encoder relative position below the title, you would include `text_lines[0].text = "Rotary encoder {}".format(macropad.encoder)` in your code. To include a second line of code, you would use `text_lines[1].text =` and provide a string to display.

This feature uses the Simple Text Display library; for advanced usage check out the Simple Text Display documentation ().

In your code, first, you import the MacroPad library, and then instantiate it.

Then, you create a `text_lines` variable, initialise the `display_text` feature by assigning `text_lines = macropad.display_text()`, and, inside the parentheses, provide it the title as a string, e.g. `title="MacroPad Info"`.

Inside the loop, the first thing you do is setup to look for the key press by creating the `key_event` variable and assigning it to `macropad.keys.events.get()`. Then, you check to see if there is a `key_event` (i.e. a key being pressed) and if it is a key being pressed (`key_event.pressed`). Then, if so, if so, update the first line of text to appear on the display showing which key number (`key_event.key_number`) was pressed.

Next, you display two more lines of text - one for the rotary encoder relative position and one for the rotary encoder switch state. Each of these updates when you rotate the rotary encoder or press on the rotary encoder switch.

Finally, you call `text_lines.show()` to make the lines of text show up on the display.

That's all there is to displaying lines of text on the built-in display of the Adafruit MacroPad using the CircuitPython MacroPad library!

MacroPad Display Image

The Adafruit MacroPad comes with a built in display. The MacroPad library makes it super simple to display a CircuitPython-compatible bitmap image on the display. To learn more about how to create a CircuitPython-compatible bitmap, check out this guide () - the difference here is, the MacroPad display is monochrome, so you'll want a black and white image.

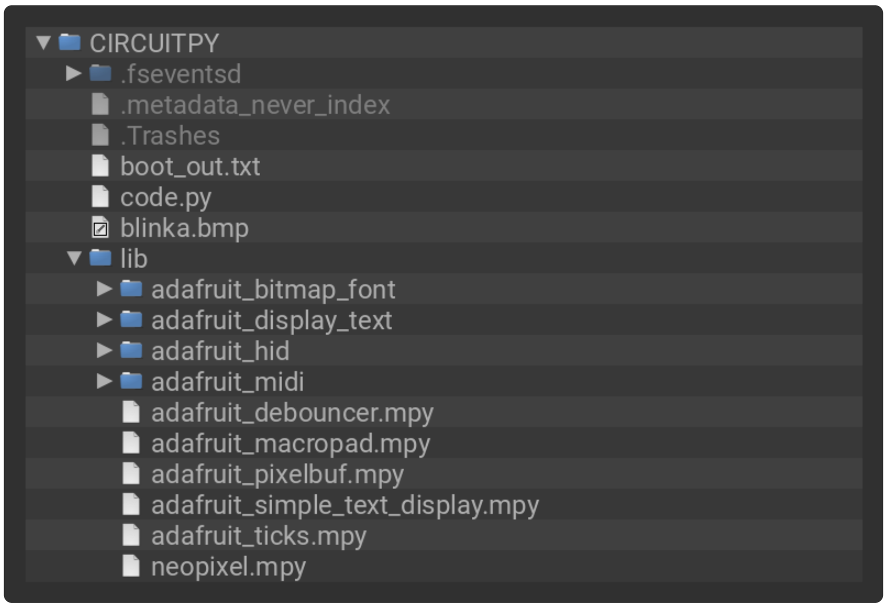
You can easily update the code to use any compatible bitmap you'd like, but for this example, use the image included below.

Installing Project Code

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your CIRCUITPY drive. Then you need to update code.py with the example script.

Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the directory examples/macropad_display_image/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

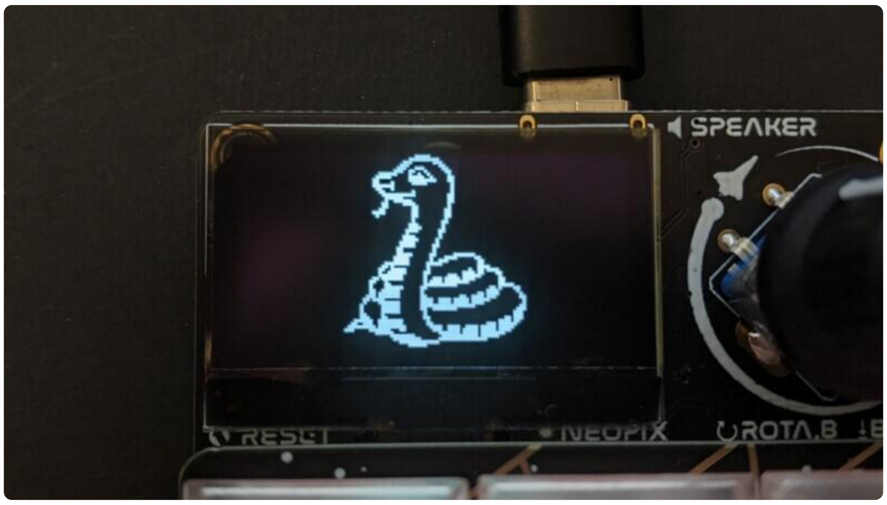
Your CIRCUITPY drive should now look similar to the following image:



```
▼ CIRCUITPY
  ▶ .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    blinka.bmp
  ▼ lib
    ▶ adafruit_bitmap_font
    ▶ adafruit_display_text
    ▶ adafruit_hid
    ▶ adafruit_midi
      adafruit_debouncer.mpy
      adafruit_macropad.mpy
      adafruit_pixelbuf.mpy
      adafruit_simple_text_display.mpy
      adafruit_ticks.mpy
      neopixel.mpy
```

```python
# SPDX-FileCopyrightText: Copyright (c) 2021 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
MacroPad display image demo. Displays a bitmap image on the built-in display.
"""
from adafruit_macropad import MacroPad

macropad = MacroPad()

macropad.display_image("blinka.bmp")

while True:
    pass
```

Check out the display!



To use the `display_image` feature of the MacroPad library, you import and instantiate the library as usual, and then include `macropad.display_image("image_name.bmp")` in your code, where `image_name` is the name of your bitmap image.

In this example, you import and instantiate the MacroPad library.

Then, you include `macropad.display_image("blinka.bmp")`.

Inside the loop, you simply include a `pass`.

That's all there is to displaying a CircuitPython-compatible bitmap image on the built-in display of the Adafruit MacroPad using the CircuitPython MacroPad library!

MacroPad Rotation

The Adafruit MacroPad RP2040 features a 3x4 key pad with NeoPixel LEDs, and a display. This example reads key presses, lights up NeoPixel associated the key pressed, and prints to the display which key is pressed.

The keys are numbered 0 through 11 (remember Python begins counting at 0), beginning at the top left of the keypad, and ending at the lower right, numbered along each row reading left to right. The NeoPixels are numbered the same way, left to right, top to bottom. The image below shows the standard key and pixel numbering.

The MacroPad lends itself to many different projects (), most of them using the MacroPad as shown above. What if you'd rather use it in a different orientation? Perhaps a 90 degree rotation to have a 4x3 keypad with the rotary encoder and display on the left. What about a 180 degree rotation to put the display and the rotary encoder on the bottom. Or a 270 degree rotation to have the rotary encoder on the bottom right below the display. This type of rotation involves remapping the keys and the NeoPixels and rotating the display to match. Sound like a lot of work? It is. And the MacroPad library does all the work for you!

The MacroPad library makes it simple to rotate your MacroPad. When you instantiate the library after import, instead of simply doing `macropad = MacroPad()`, you include a `rotation`. The supported rotation options are:

- `0` - This is the default. This is the MacroPad in a standard orientation with the USB connector pointing upward.
- `90` - This is the MacroPad with the USB connector pointing to the left.
- `180` - This is the MacroPad with the USB connector pointing towards the bottom.
- `270` - This is the MacroPad with the USB connector pointing to the right.

Any other rotation value will cause an error.

For example, to rotate the MacroPad 90 degrees, you would import and instantiate the library as follows.

```
from adafruit_macropad import MacroPad

macropad = MacroPad(rotation=90)
```

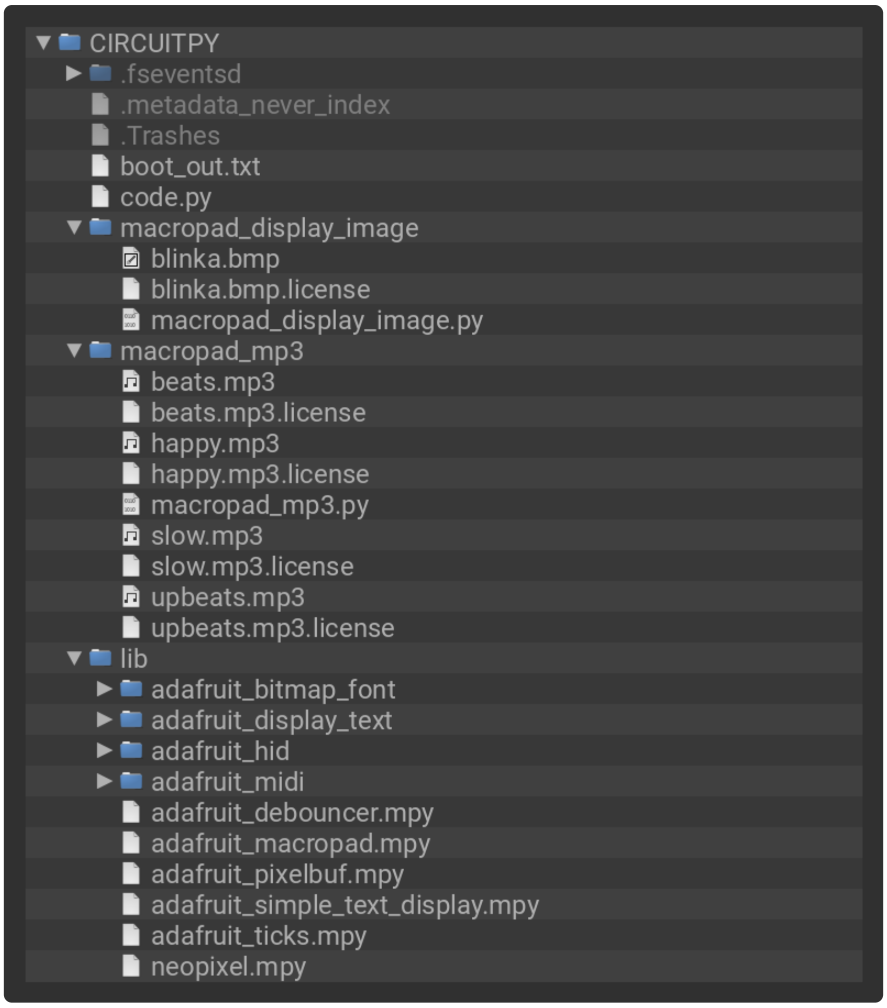
This rotates the display 90 degrees and remaps the keys and NeoPixels to match. The rest of your code will now read the keys and NeoPixels, left to right, top to bottom beginning at the key closest to the rotary encoder. What does that look like? Check out the following example.

Rotation Example

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your CIRCUITPY drive. Then you need to update code.py with the example script.

Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the directory examples/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your CIRCUITPY drive.

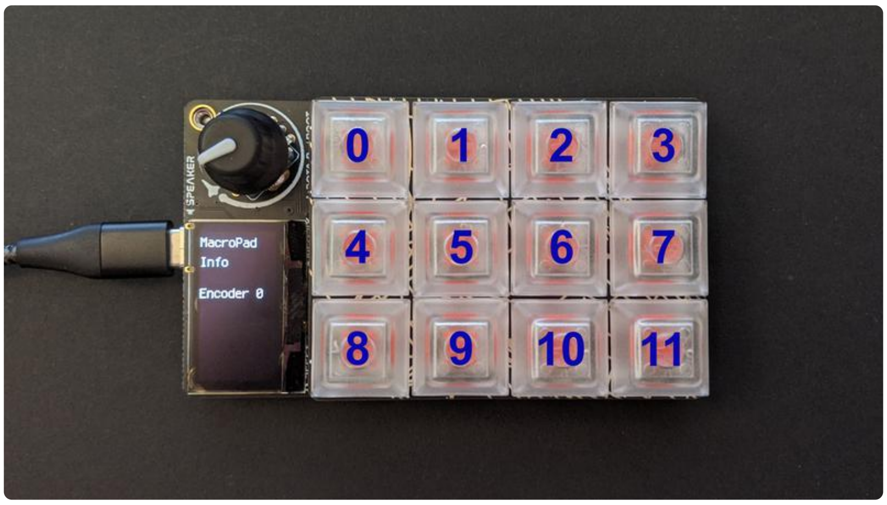
Your CIRCUITPY drive should now look similar to the following image:



```
CIRCUITPY
  .fseventsd
  .metadata_never_index
  .Trashes
  boot_out.txt
  code.py
  macropad_display_image
    blinka.bmp
    blinka.bmp.license
    macropad_display_image.py
  macropad_mp3
    beats.mp3
    beats.mp3.license
    happy.mp3
    happy.mp3.license
    macropad_mp3.py
    slow.mp3
    slow.mp3.license
    upbeats.mp3
    upbeats.mp3.license
  lib
    adafruit_bitmap_font
    adafruit_display_text
    adafruit_hid
    adafruit_midi
    adafruit_debouncer.mpy
    adafruit_macropad.mpy
    adafruit_pixelbuf.mpy
    adafruit_simple_text_display.mpy
    adafruit_ticks.mpy
    neopixel.mpy
```

```
# SPDX-FileCopyrightText: Copyright (c) 2021 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
MacroPad rotation demo. Rotates the display 90 degrees and remaps the NeoPixels and
keys to match.
Lights up the associated pixel when the key is pressed. Displays the key number
pressed and the
rotary encoder relative position on the display.
"""
from rainbowio import colorwheel
from adafruit_macropad import MacroPad

macropad = MacroPad(rotation=90)

text_lines = macropad.display_text(title="MacroPad \nInfo")

while True:
    key_event = macropad.keys.events.get()
    if key_event:
        if key_event.pressed:
            text_lines[1].text = "Key {}!".format(key_event.key_number)
            macropad.pixels[key_event.key_number] = colorwheel(
                int(255 / 12) * key_event.key_number
            )
        else:
            macropad.pixels.fill((0, 0, 0))
    text_lines[2].text = "Encoder {}".format(macropad.encoder)
    text_lines.show()
```

Now, rotate your MacroPad 90 degrees, and try pressing the keys and rotate the rotary encoder! The 90 degree orientation is when the rotary encoder is in the upper left corner. The key and NeoPixels are numbered starting on the key closest to the rotary encoder.



This example rotates the MacroPad 90 degrees. Remember, when the MacroPad is in a sideways orientation, the display is 64x128 pixels (instead of the standard 128x64 pixels). Any text displayed needs to be shortened to fit on the display. So, this example puts the title on two lines to shorten it up and uses shorter strings than the Display Text example ().

In your code, you first import `colorwheel` from `rainbowio`.

Next, you import and instantiate the MacroPad library, specifying the 90 degree rotation with `rotation=90`.

Then, you create a `text_lines` variable, initialise the `display_text` feature by assigning `text_lines = macropad.display_text()`, and, inside the parentheses, provide it the title as a string, e.g. `title="MacroPad\nInfo"`. Note the `\n` which puts `Info` on a separate line.

Inside the loop, the first thing you do is setup to look for the key press by creating the `key_event` variable and assigning it to `macropad.keys.events.get()`. Then, you check to see if there is a `key_event` (i.e. a key being pressed). If it is a key being pressed (`key_event.pressed`), you print to the display the key number pressed and light up the key using the `key_number` to generate a `colorwheel()` value. Otherwise when the key is released, turn off the LEDs.

Then, you print the relative encoder value to the display.

Finally, you call `text_lines.show()` to make the lines of text show up on the display.

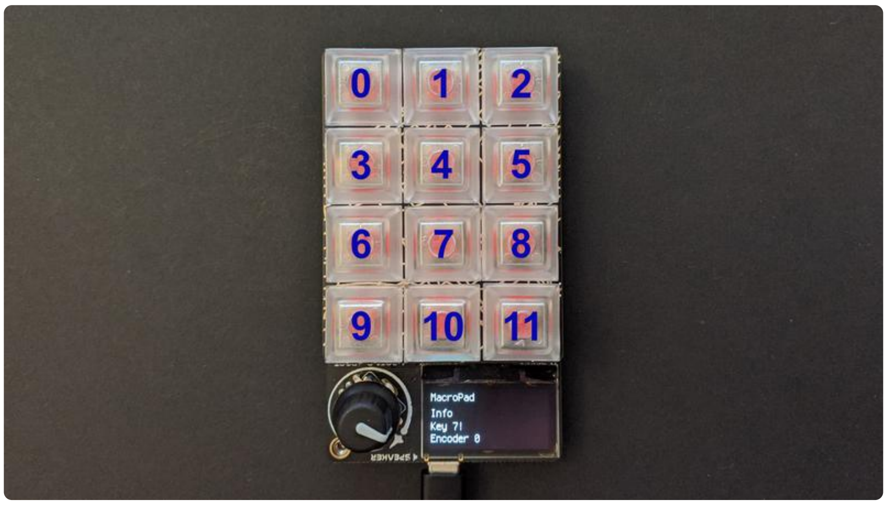
Other Rotations

Simply updating the `rotation=` in the MacroPad library instantiation allows you to view the other orientation options.

For example, to rotate this example to 180 degrees, you would change the line of code from `macropad = MacroPad(rotation=90)` to the following:

```
macropad = MacroPad(rotation=180)
```

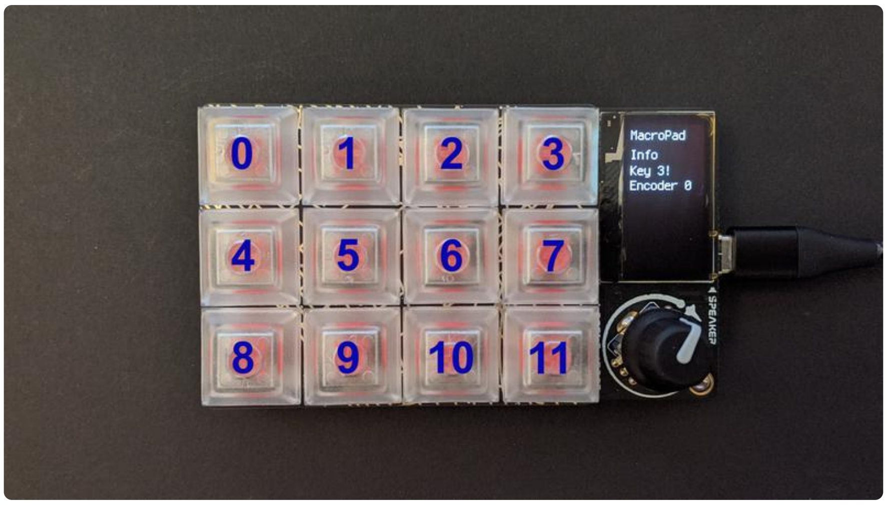
Rotate the MacroPad so the USB connector is pointed downwards and try pressing the keys to see the key number printed to the display.



To rotate this example to 270 degrees, you would change the line of code from `macropad = MacroPad(rotation=180)` to the following:

```
macropad = MacroPad(rotation=180)
```

Rotate the MacroPad so the USB connector is pointed to the left and try pressing the keys to see the key number printed to the display.



Once rotated, you can include any other features of the MacroPad library using the matching key and NeoPixel map with the MacroPad oriented in whatever way you like!

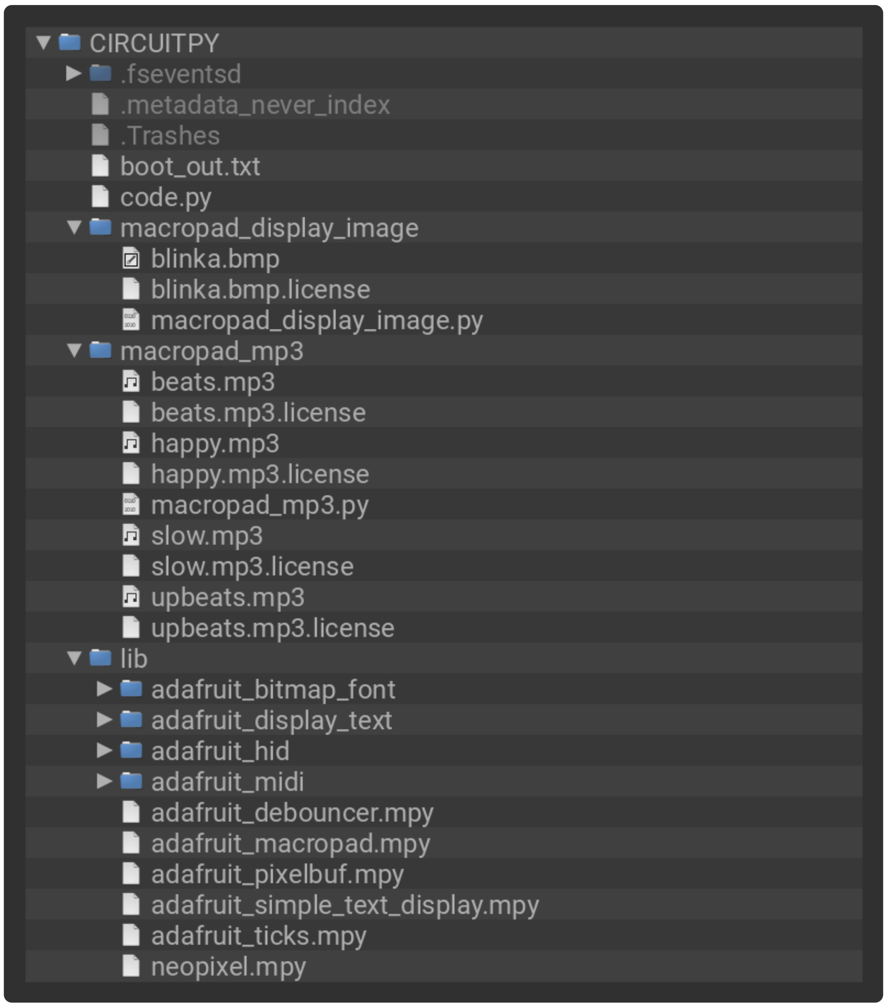
That's all there is to rotating the MacroPad using the CircuitPython MacroPad library!

MacroPad Tone

The Adafruit MacroPad RP2040 features a 3x4 key pad with NeoPixel LEDs, and a small buzzer/speaker This example plays a different tone for each key pressed, and lights up each key a different color while pressed.

Update your code.py to the following.

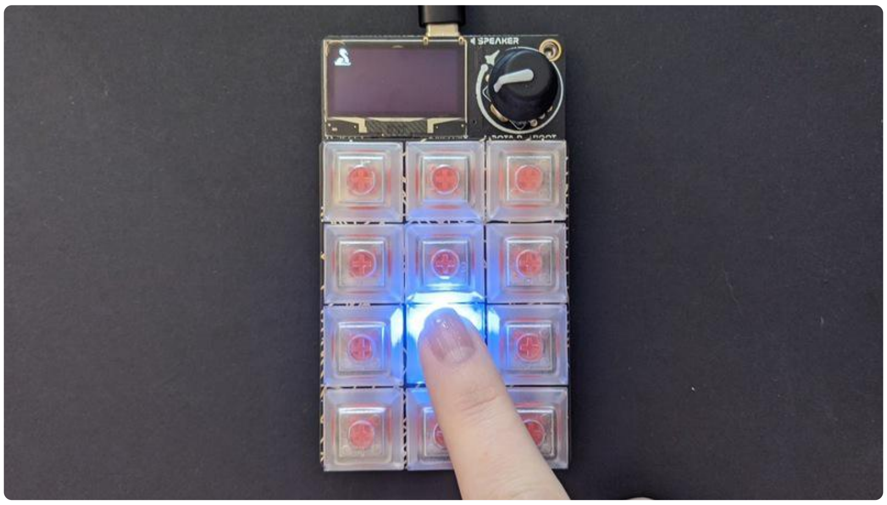
Click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the folder that matches your CircuitPython version, and copy the entire lib folder and the code.p y file to your CIRCUITPY drive.



```
CIRCUITPY
    .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    macropad_display_image
        blinka.bmp
        blinka.bmp.license
        macropad_display_image.py
    macropad_mp3
        beats.mp3
        beats.mp3.license
        happy.mp3
        happy.mp3.license
        macropad_mp3.py
        slow.mp3
        slow.mp3.license
        upbeats.mp3
        upbeats.mp3.license
    lib
        adafruit_bitmap_font
        adafruit_display_text
        adafruit_hid
        adafruit_midi
        adafruit_debouncer.mpy
        adafruit_macropad.mpy
        adafruit_pixelbuf.mpy
        adafruit_simple_text_display.mpy
        adafruit_ticks.mpy
        neopixel.mpy
```

```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
MacroPad tone demo. Plays a different tone for each key pressed and lights up each
key a different
color while the key is pressed.
"""
from rainbowio import colorwheel
from adafruit_macropad import MacroPad

macropad = MacroPad()

tones = [196, 220, 246, 262, 294, 330, 349, 392, 440, 494, 523, 587]

while True:
    key_event = macropad.keys.events.get()

    if key_event:
        if key_event.pressed:
            macropad.pixels[key_event.key_number] = colorwheel(
                int(255 / 12) * key_event.key_number
            )
            macropad.start_tone(tones[key_event.key_number])

        else:
            macropad.pixels.fill((0, 0, 0))
            macropad.stop_tone()
```

Now, press any key! It plays a unique tone and lights up a different color of the rainbow while pressed.



The MacroPad library includes the ability to play tones, either for a specified duration, for example, 0.5 seconds, or until you tell it to stop in your code, for example, the duration of a key press.

The common thing required for playing a tone, regardless of which method you choose, is to specify the tone frequency in Hz as an integer (meaning a whole number without a decimal). For example, to play a "middle C" tone, you would specify `262`.

To play that tone for a specified duration of 0.5 seconds, you would include `macropad.play_tone(262, 0.5)` in your code.

This example uses the `start_tone` and `stop_tone` features of the MacroPad library. The first, `start_tone`, requires only a tone frequency in Hz. However, when you call it in your code, it will play until you tell it to stop. So, if you call `start_tone` without calling `stop_tone()`, it will continue to play indefinitely! So, you always want to call `stop_tone()` somewhere after `start_tone`.

In this example, you import `colorwheel` from `rainbowio`, and you import and instantiate the MacroPad library.

Next, you create a list of 12 tone frequencies in Hz. You must include 12 for it to work properly, as there are 12 keys. You can customise the tones easily by changing the numbers.

Inside the loop, the first thing you do is setup to look for the key press by creating the `key_event` variable and assigning it to `macropad.keys.events.get()`. Then, you check to see if there is a `key_event` (i.e. a key being pressed). If it is a key being pressed (`key_event.pressed`), you light up the key using the `key_number` to generate a `colorwheel()` value, and you start playing the tone from the list that is the same number in the list as the key number being pressed. Remember that Python begins counting from 0. So if you press the first key, the first tone will be played, 196Hz. If you press the fifth key, the fifth tone in the list will be played, 294Hz.

Otherwise, as soon as the key is no longer being pressed (i.e. released), you turn off all the LEDs, and stop playing the tone.

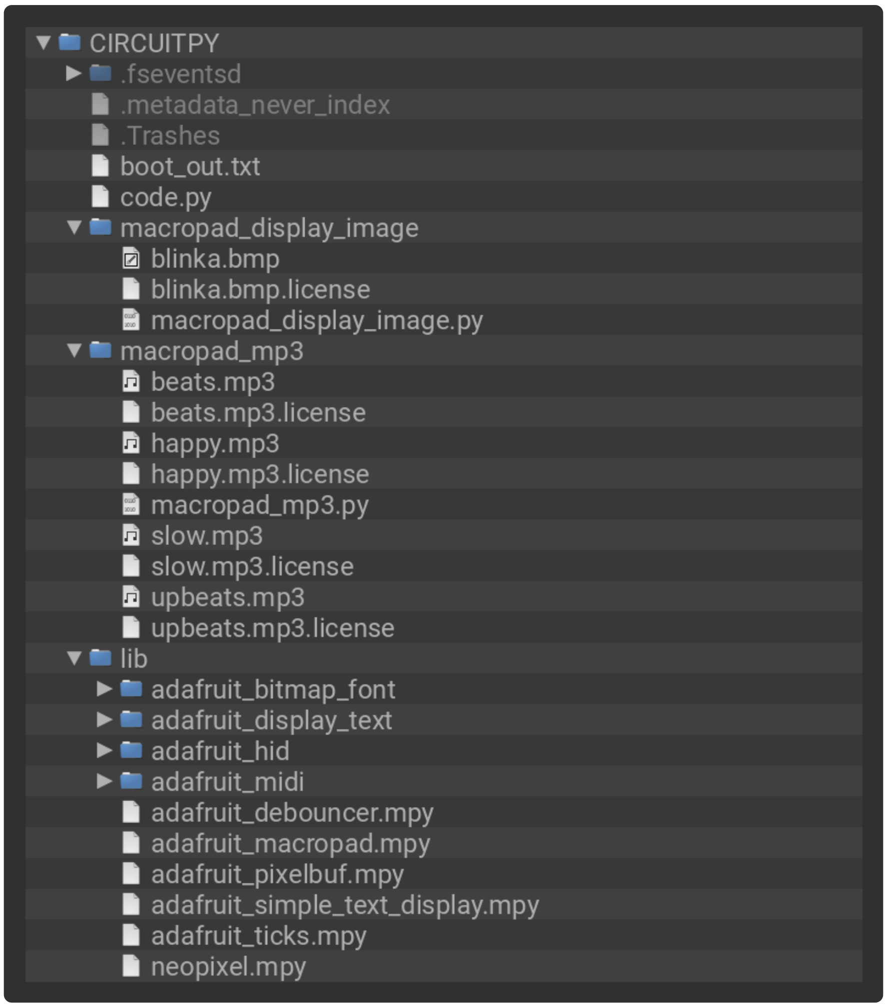
That's all there is to playing a tone using the CircuitPython MacroPad library!

MacroPad Keyboard and Mouse

The Adafruit MacroPad RP2040 features a 3x4 key pad and a rotary encoder with push switch. This example uses the first few keys to send different types of HID commands, the rotary encoder switch to send a right mouse click, and the rotary encoder to move the mouse left and right.

Update your code.py to the following.

Click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, open the folder that matches your CircuitPython version, and copy the entire lib folder and the code.p y file to your CIRCUITPY drive.



```
CIRCUITPY
    .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    macropad_display_image
        blinka.bmp
        blinka.bmp.license
        macropad_display_image.py
    macropad_mp3
        beats.mp3
        beats.mp3.license
        happy.mp3
        happy.mp3.license
        macropad_mp3.py
        slow.mp3
        slow.mp3.license
        upbeats.mp3
        upbeats.mp3.license
    lib
        adafruit_bitmap_font
        adafruit_display_text
        adafruit_hid
        adafruit_midi
        adafruit_debouncer.mpy
        adafruit_macropad.mpy
        adafruit_pixelbuf.mpy
        adafruit_simple_text_display.mpy
        adafruit_ticks.mpy
        neopixel.mpy
```

Be aware that this an HID demo, which means it will be sending key commands and moving your mouse! Once you hit save, make sure your cursor is in a text editor of some sort so you can see the results typed out.

```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
MacroPad HID keyboard and mouse demo. The demo sends "a" when the first key is
pressed, a "B" when
the second key is pressed, "Hello, World!" when the third key is pressed, and
decreases the volume
when the fourth key is pressed. It sends a right mouse click when the rotary
encoder switch is
pressed. Finally, it moves the mouse left and right when the rotary encoder is
rotated
counterclockwise and clockwise respectively.
"""
from adafruit_macropad import MacroPad

macropad = MacroPad()

last_position = 0
while True:
    key_event = macropad.keys.events.get()

    if key_event:
        if key_event.pressed:
            if key_event.key_number == 0:
                macropad.keyboard.send(macropad.Keycode.A)
            if key_event.key_number == 1:
                macropad.keyboard.press(macropad.Keycode.SHIFT, macropad.Keycode.B)
                macropad.keyboard.release_all()
            if key_event.key_number == 2:
                macropad.keyboard_layout.write("Hello, World!")
            if key_event.key_number == 3:
                macropad.consumer_control.send(
                    macropad.ConsumerControlCode.VOLUME_DECREMENT
                )

    macropad.encoder_switch_debounced.update()

    if macropad.encoder_switch_debounced.pressed:
        macropad.mouse.click(macropad.Mouse.RIGHT_BUTTON)

    current_position = macropad.encoder

    if macropad.encoder > last_position:
        macropad.mouse.move(x=+5)
        last_position = current_position

    if macropad.encoder < last_position:
        macropad.mouse.move(x=-5)
        last_position = current_position
```

Try pressing the first key, it will type out a lowercase "a". Press the second key, it will type out an uppercase "B". Press the third key, it will type out the string "Hello, World!". Press the fourth key, it will decrease the volume. Press down on the rotary encoder switch to send a right mouse click. Rotate the rotary encoder clockwise to move the mouse cursor to the right, and counterclockwise to move it to the left.

The MacroPad library wraps in the ability to send HID commands to your computer, such as key presses, strings, consumer control commands (e.g. volume increase or decrease), and mouse clicks and movement. The library also makes it simple to read key presses and the rotary encoder position, and makes a debounced version of the rotary encoder switch available.

In your code, first, you import the MacroPad library, and then instantiate it.

Then you create a `last_position` variable to track the rotary encoder position, and set it to `0`.

Inside the loop, the first thing you do is setup to look for the key press by creating the `key_event` variable and assigning it to `macropad.keys.events.get()`. Then, you check to see if there is a `key_event` (i.e. a key being pressed) and if it is a key being pressed (`key_event.pressed`). Then do the following based on key number. Remember, key 0 is the first key - Python begins counting at 0.

- If key 0 is pressed, send the letter "a".

- If key 1 is pressed, send "shift+b" to send "B".
- If key 2 is pressed, send the string "Hello, World!"
- If key 3 is pressed, decrease the volume.

Next, you include `macropad.encoder_switch_debounced.update()` in your code to continually check the state of the switch. This line is required to use the debounced encoder switch. Once you include the `update()` line, you can check for two states:

- `macropad.encoder_switch_debounced.pressed` - True when the switch is pressed. Sends only one press event per switch press.
- `macropad.encoder_switch_debounced.released` - True when the switch is released. Sends only one release event per switch release.

In this case, you check to see if the rotary encoder is pressed, and if so, send a right mouse click. Using the debounced version of the rotary encoder ensures that the mouse click is only sent one time per switch press.

Then, you set `current_position = macropad.encoder` so you have the current position of the rotary encoder to work with.

Next, you check to see if the encoder position is greater than the `last_position` (which starts at 0), which is to say, has it been rotated clockwise. If so, move the mouse to the right, and set the last position equal to the current position so you can begin tracking again.

Finally, you do the same again, except this time you're checking whether the encoder position is less than the last position, which is to say, has it been rotated counterclockwise. If so, move the mouse to the left, and set `last_position = current_position`.

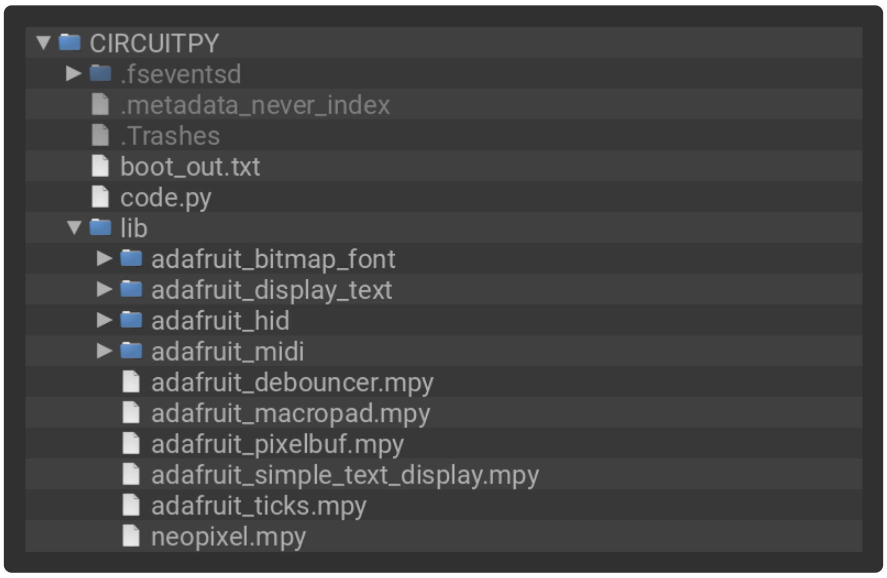
That's all there is to using the Adafruit MacroPad to send HID commends with the CircuitPython MacroPad library!

Macropad MIDI

This code shows how you can send USB MIDI messages using the Macropad.

Save the following to your CIRCUITPY drive as code.py.

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



```
CIRCUITPY
    .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
    lib
        adafruit_bitmap_font
        adafruit_display_text
        adafruit_hid
        adafruit_midi
        adafruit_debouncer.mpy
        adafruit_macropad.mpy
        adafruit_pixelbuf.mpy
        adafruit_simple_text_display.mpy
        adafruit_ticks.mpy
        neopixel.mpy
```

```
# SPDX-FileCopyrightText: 2022 John Park for Adafruit Industries
# SPDX-License-Identifier: MIT
# Macropad MIDI Tester
# Play MIDI notes with keys
# Click encoder to switch modes
# Turn encoder to adjust CC, ProgramChange, or PitchBend
from adafruit_macropad import MacroPad
from rainbowio import colorwheel

CC_NUM = 74  # select your CC number

macropad = MacroPad(rotation=180)  # create the macropad object, rotate orientation
macropad.display.auto_refresh = False  # avoid lag

# --- Pixel setup --- #
key_color = colorwheel(120)  # fill with cyan to start
macropad.pixels.brightness = 0.1
macropad.pixels.fill(key_color)

# --- MIDI variables ---
mode = 0
mode_text = ["Patch", ("CC #%s" % (CC_NUM)), "Pitch Bend"]
midi_values = [0, 16, 8]  # bank, cc value, pitch
# Chromatic scale starting with C3 as bottom left keyswitch (or use any notes you
like)
midi_notes = [
            57, 58, 59,
            54, 55, 56,
            51, 52, 53,
            48, 49, 50
            ]

# --- Display text setup ---
text_lines = macropad.display_text("Macropad MIDI Tester")
text_lines[0].text = "Mode: Patch {}".format(midi_values[0]+1)  # Patch display
offset by 1
text_lines[1].text = "Press knob for modes"
text_lines.show()
```

```
last_knob_pos = macropad.encoder  # store knob position state


while True:
    while macropad.keys.events:  # check for key press or release
        key_event = macropad.keys.events.get()
        if key_event:
            if key_event.pressed:
                key = key_event.key_number
                macropad.midi.send(macropad.NoteOn(midi_notes[key], 120))  # send
midi noteon
                macropad.pixels[key] = colorwheel(90)  # light up green
                text_lines[1].text = "NoteOn:{}".format(midi_notes[key])

            if key_event.released:
                key = key_event.key_number
                macropad.midi.send(macropad.NoteOff(midi_notes[key], 0))
                macropad.pixels[key] = key_color  # return to color set by encoder
bank value
                text_lines[1].text = "NoteOff:{}".format(midi_notes[key])

    macropad.encoder_switch_debounced.update()  # check the knob switch for press
or release
    if macropad.encoder_switch_debounced.pressed:
        mode = (mode+1) % 3
        if mode == 0:
            text_lines[0].text = ("Mode: %s %d" % (mode_text[mode],
midi_values[mode]+1))
        elif mode == 1:
            text_lines[0].text = ("Mode: %s %d" % (mode_text[mode],
int(midi_values[mode]*4.1)))
        else:
            text_lines[0].text = ("Mode: %s %d" % (mode_text[mode],
midi_values[mode]-8))
        macropad.red_led = macropad.encoder_switch
        text_lines[1].text = " "  # clear the note line

    if macropad.encoder_switch_debounced.released:
        macropad.red_led = macropad.encoder_switch

    if last_knob_pos is not macropad.encoder:  # knob has been turned
        knob_pos = macropad.encoder  # read encoder
        knob_delta = knob_pos - last_knob_pos  # compute knob_delta since last read
        last_knob_pos = knob_pos  # save new reading

        if mode == 0:  # ProgramChange
            midi_values[mode] = min(max(midi_values[mode] + knob_delta, 0), 127)  #
delta + minmax
            macropad.midi.send(macropad.ProgramChange(midi_values[mode]))  # midi
send ProgramChange
            key_color = colorwheel(midi_values[mode]+120)  # change key_color as
patches change
            macropad.pixels.fill(key_color)
            text_lines[0].text = ("Mode: %s %d" % (mode_text[mode],
midi_values[mode]+1))

        if mode == 1:  # CC
            midi_values[mode] = min(max(midi_values[mode] + knob_delta, 0), 31)  #
scale the value
            macropad.midi.send(macropad.ControlChange(CC_NUM,
int(midi_values[mode]*4.1)))
            text_lines[0].text = ("Mode: %s %d" % (mode_text[mode],
int(midi_values[mode]*4.1)))

        if mode == 2:  # PitchBend
            midi_values[mode] = min(max(midi_values[mode] + knob_delta, 0), 15)  #
smaller range
            macropad.midi.send(macropad.PitchBend((midi_values[mode]*1024)))  #
```

```
range * mult = 16384
            text_lines[0].text = ("Mode: %s %d" % (mode_text[mode],
midi_values[mode]-8))

        last_knob_pos = macropad.encoder

    macropad.display.refresh()
```

To test it out, load up a software synthesizer on your computer or iOS/Android device, and press the Macropad keys to play notes. You can find a good list of software to try here (), but this will work with any MIDI capable software synth.

Modes

Click the knob to cycle among three modes:

- Patch select mode
- CC mode
- Pitch bend mode

In Patch select mode you can turn the knob to switch synth patch presets, and test out different sounds built into your synth.

In CC mode the knob sends ControlChange messages from 0-127 on CC #74 by default. This is often used to change filter frequency, but you can customize this in your software.

In Pitch Bend mode, turn the knob left and right to bend the pitch!

If you'd like to learn more about how it works, you can watch an in-depth look at things here:

MacroPad Library Docs

MacroPad Library Docs ()

Arduino IDE Setup

The Arduino Philhower core () provides support for RP2040 microcontroller boards. This page covers getting your Arduino IDE set up to include your board.

Arduino IDE Download

The first thing you will need to do is to download the latest release of the Arduino IDE. The Philhower core requires version 1.8 or higher.

Arduino IDE Download

Download and install it to your computer.

Once installed, open the Arduino IDE.

Adding the Philhower Board Manager URL

In the Arduino IDE, and navigate to the Preferences window. You can access it through File > Preferences on Windows or Linux, or Arduino > Preferences on OS X.

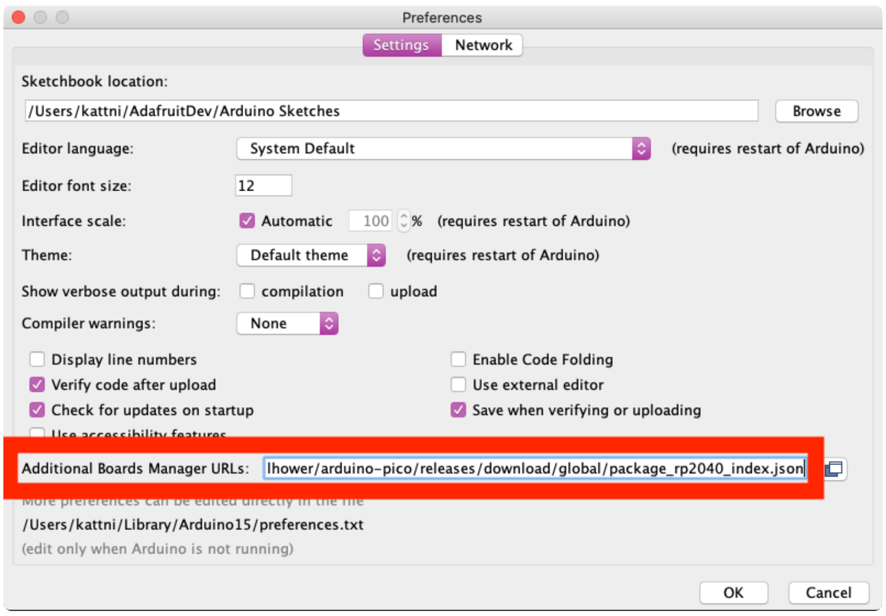
The Preferences window will open.

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://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json`

Add the URL to the the Additional Boards Manager URLs field (highlighted in red below).


Preferences

Settings | Network

Sketchbook location:
/Users/kattni/AdafruitDev/Arduino Sketches | Browse

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

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

Theme: Default theme (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☐ Display line numbers
☑ Verify code after upload
☑ Check for updates on startup
☐ Use accessibility features
☐ Enable Code Folding
☐ Use external editor
☑ Save when verifying or uploading

Additional Boards Manager URLs: lhower/arduino-pico/releases/download/global/package_rp2040_index.json

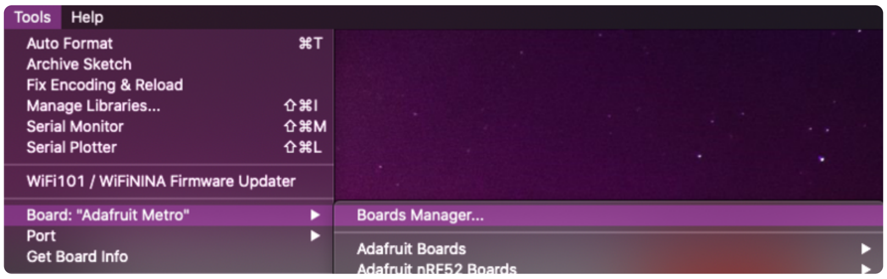
More preferences can be edited directly in the file
/Users/kattni/Library/Arduino15/preferences.txt
(edit only when Arduino is not running)

OK | Cancel

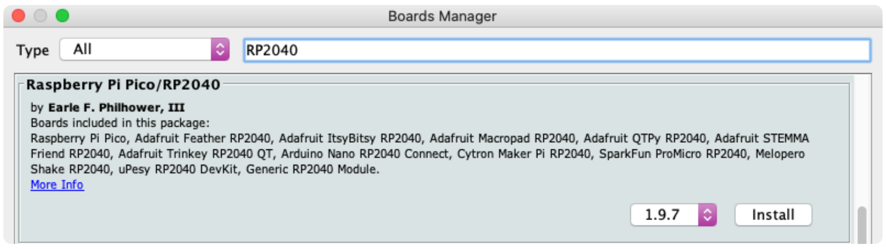
Click OK to save and close Preferences.

Add Board Support Package

In the Arduino IDE, click on Tools > Board > Boards Manager. If you have previously selected a board, the Board menu item may have a board name after it.



In the Boards Manager, search for RP2040. Scroll down to the Raspberry Pi Pico/RP2040 by Earle F Philhower, III entry. Click Install to install it.


Boards Manager

Type All | RP2040

Raspberry Pi Pico/RP2040
by **Earle F. Philhower, III**
Boards included in this package:
Raspberry Pi Pico, Adafruit Feather RP2040, Adafruit ItsyBitsy RP2040, Adafruit Macropad RP2040, Adafruit QTPy RP2040, Adafruit STEMMA Friend RP2040, Adafruit Trinkey RP2040 QT, Arduino Nano RP2040 Connect, Cytron Maker Pi RP2040, SparkFun ProMicro RP2040, Melopero Shake RP2040, uPesy RP2040 DevKit, Generic RP2040 Module.
More Info

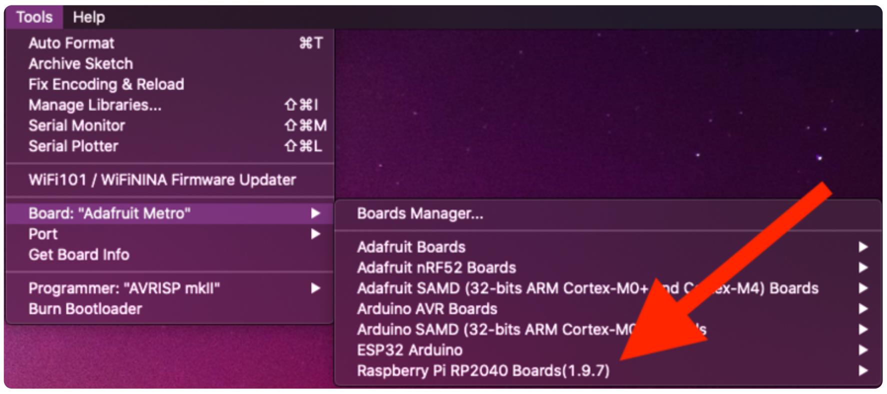
1.9.7 | Install

Installing a new board package can take a few minutes. Don't click Cancel!

Once installation is complete, click Close to close the Boards Manager.

Choose Your Board

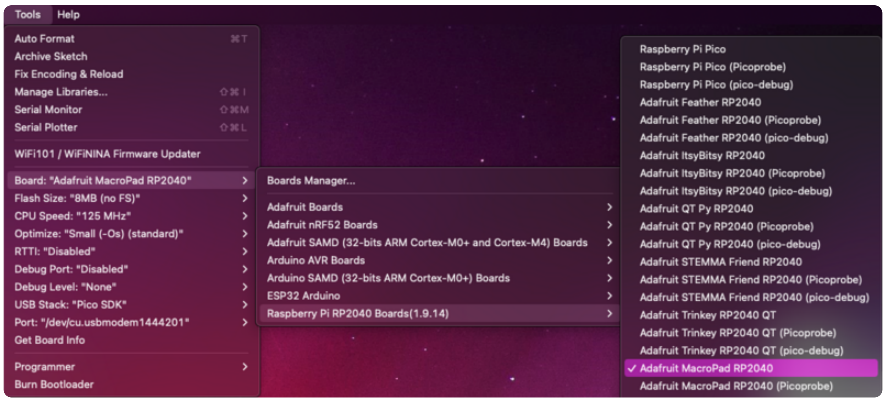
In the Tools > Boards menu, you should now see Raspberry Pi RP2040 Boards (possibly followed by a version number).



Board: "Adafruit Metro"
Programmer: "AVRISP mkII"

Navigate to the Raspberry Pi RP2040 Boards menu. You will see the available boards listed.

Navigate to the Raspberry Pi RP2040 Boards menu and choose Adafruit MacroPad RP2040.


Tools
Help
Auto Format ⌘T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... ⇧⌘I
Serial Monitor ⇧⌘M
Serial Plotter ⇧⌘L
WiFi101 / WiFiNINA Firmware Updater
Get Board Info
Programmer
Burn Bootloader
Boards Manager...
Adafruit Boards
Adafruit nRF52 Boards
Adafruit SAMD (32-bits ARM Cortex-M0+ and Cortex-M4) Boards
Arduino AVR Boards
Arduino SAMD (32-bits ARM Cortex-M0+) Boards
ESP32 Arduino
Raspberry Pi RP2040 Boards(1.9.14)
Raspberry Pi Pico
Raspberry Pi Pico (Picoprobe)
Raspberry Pi Pico (pico-debug)
Adafruit Feather RP2040
Adafruit Feather RP2040 (Picoprobe)
Adafruit Feather RP2040 (pico-debug)
Adafruit ItsyBitsy RP2040
Adafruit ItsyBitsy RP2040 (Picoprobe)
Adafruit ItsyBitsy RP2040 (pico-debug)
Adafruit QT Py RP2040
Adafruit QT Py RP2040 (Picoprobe)
Adafruit QT Py RP2040 (pico-debug)
Adafruit STEMMA Friend RP2040
Adafruit STEMMA Friend RP2040 (Picoprobe)
Adafruit STEMMA Friend RP2040 (pico-debug)
Adafruit Trinkey RP2040 QT
Adafruit Trinkey RP2040 QT (Picoprobe)
Adafruit Trinkey RP2040 QT (pico-debug)
✓ Adafruit MacroPad RP2040
Adafruit MacroPad RP2040 (Picoprobe)

Board: "Adafruit MacroPad RP2040"
Flash Size: "8MB (no FS)"
CPU Speed: "125 MHz"
Optimize: "Small (-Os) (standard)"
RTTI: "Disabled"
Debug Port: "Disabled"
Debug Level: "None"
USB Stack: "Pico SDK"
Port: "/dev/cu.usbmodem1444201"

If there is no serial Port available in the dropdown, or an invalid one appears - don't worry about it! The RP2040 does not actually use a serial port to upload, so its OK if it does not appear if in manual bootload mode. You will see a serial port appear after uploading your first sketch

Now you're ready to begin using Arduino with your RP2040 board!

Arduino Usage

Now that you've set up the Arduino IDE with the Philhower RP2040 Arduino core, you're ready to start using Arduino with your RP2040.

RP2040 Arduino Pins

There is no pin remapping for Arduino on the RP2040. Therefore, the pin names on the top of the board are not the pin names used for Arduino. The Arduino pin names are the RP2040 GPIO pin names.

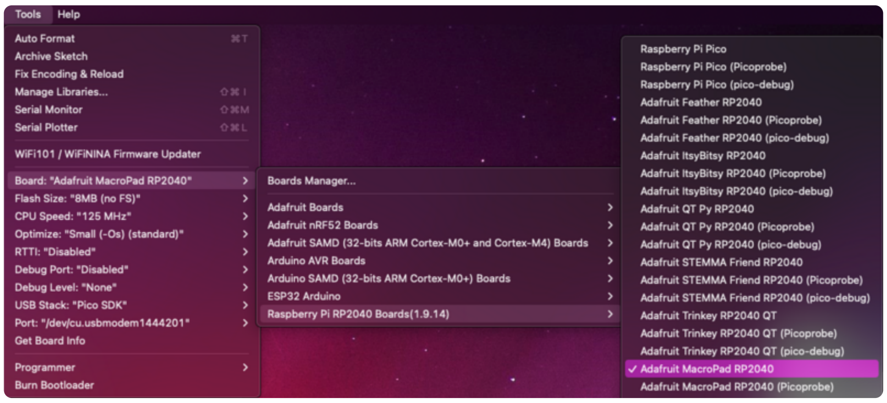
To find the Arduino pin name, check the PrettyPins diagram found on the Pinouts page. Each GPIO pin in the diagram has a GPIOx pin name listed, where x is the pin number. The Arduino pin name is the number following GPIO. For example, GPIO1 would be Arduino pin `1`.

Choose Your Board

Navigate to the Tools > Boards > Raspberry Pi RP2040 Boards menu. The Raspberry PI RP2040 Boards menu name may be followed by a version number.

Once the menu has expanded, you will see three different versions the MacroPad available: Adafruit MacroPad RP2040, Adafruit MacroPad RP2040 (Picoprobe), and Adafruit MacroPad RP2040 (pico-debug). Unless you are specifically familiar with the other two, always choose Adafruit MacroPad RP2040.

Choose Adafruit MacroPad RP2040 from the menu.

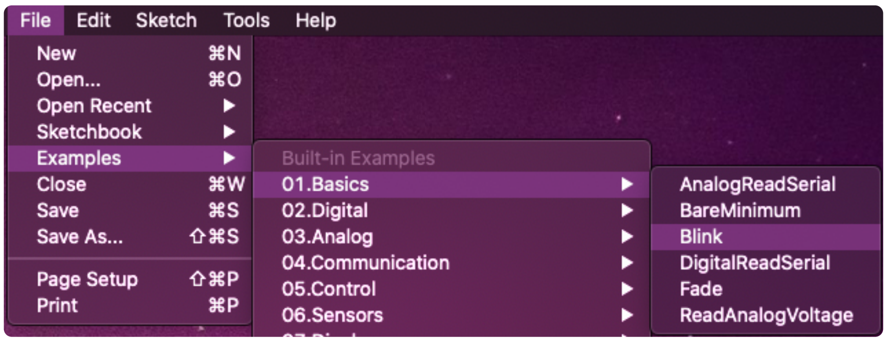

Tools
Help
Auto Format
Archive Sketch
Fix Encoding & Reload
Manage Libraries...
Serial Monitor
Serial Plotter
WiFi101 / WiFiNINA Firmware Updater
Get Board Info
Programmer
Burn Bootloader
Boards Manager...
Adafruit Boards
Adafruit nRF52 Boards
Adafruit SAMD (32-bits ARM Cortex-M0+ and Cortex-M4) Boards
Arduino AVR Boards
Arduino SAMD (32-bits ARM Cortex-M0+) Boards
ESP32 Arduino
Raspberry Pi RP2040 Boards(1.9.14)
Raspberry Pi Pico
Raspberry Pi Pico (Picoprobe)
Raspberry Pi Pico (pico-debug)
Adafruit Feather RP2040
Adafruit Feather RP2040 (Picoprobe)
Adafruit Feather RP2040 (pico-debug)
Adafruit ItsyBitsy RP2040
Adafruit ItsyBitsy RP2040 (Picoprobe)
Adafruit ItsyBitsy RP2040 (pico-debug)
Adafruit QT Py RP2040
Adafruit QT Py RP2040 (Picoprobe)
Adafruit QT Py RP2040 (pico-debug)
Adafruit STEMMA Friend RP2040
Adafruit STEMMA Friend RP2040 (Picoprobe)
Adafruit STEMMA Friend RP2040 (pico-debug)
Adafruit Trinkey RP2040 QT
Adafruit Trinkey RP2040 QT (Picoprobe)
Adafruit Trinkey RP2040 QT (pico-debug)
✓ Adafruit MacroPad RP2040
Adafruit MacroPad RP2040 (Picoprobe)

Board: "Adafruit MacroPad RP2040"
Flash Size: "8MB (no FS)"
CPU Speed: "125 MHz"
Optimize: "Small (-Os) (standard)"
RTTI: "Disabled"
Debug Port: "Disabled"
Debug Level: "None"
USB Stack: "Pico SDK"
Port: "/dev/cu.usbmodem1444201"

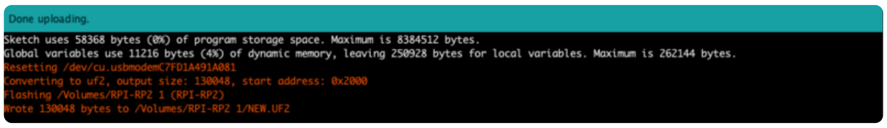
Load the Blink Sketch

Begin by plugging in your board to your computer, and wait a moment for it to be recognised by the OS. It will create a COM/serial port that you can now select from the Tools > Port menu dropdown.

Open the Blink sketch by clicking through File > Examples > 01.Basics > Blink.



Click Upload. A successful upload will result in text similar to the following.



```
Done uploading.
Sketch uses 58368 bytes (0%) of program storage space. Maximum is 8384512 bytes.
Global variables use 11216 bytes (4%) of dynamic memory, leaving 250928 bytes for local variables. Maximum is 262144 bytes.
Resetting /dev/cu.usbmodemC7FD1A491A081
Converting to uf2, output size: 130048, start address: 0x2000
Flashing /Volumes/RPI-RP2 1 (RPI-RP2)
Wrote 130048 bytes to /Volumes/RPI-RP2 1/NEW.UF2
```

Once complete, the little red LED will begin blinking once every second! Try changing up the `delay()` timing to change the rate at which the LED blinks.

Manually Enter the Bootloader

If you get into a state with the bootloader where you can no longer upload a sketch, or you have uploaded code that crashes and doesn't auto-reboot into the bootloader, you may have to manually enter the bootloader.

To enter the bootloader, hold down the boot button (by pressing the rotary encoder!), and while continuing to hold it (don't let go!), press and release the reset button. Conti nue to hold the boot button until the RPI-RP2 drive appears!

Once the RPI-RP2 drive shows up, your board is in bootloader mode. There will not be a port available in bootloader mode, this is expected. Click Upload on your sketch to try again.

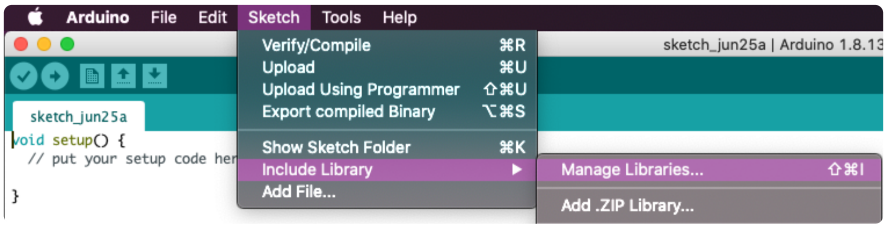
MacroPad Arduino Example

Once you have Arduino setup on your MacroPad, you're ready to continue with the following example. First, you'll need to install a few libraries.

Required Libraries

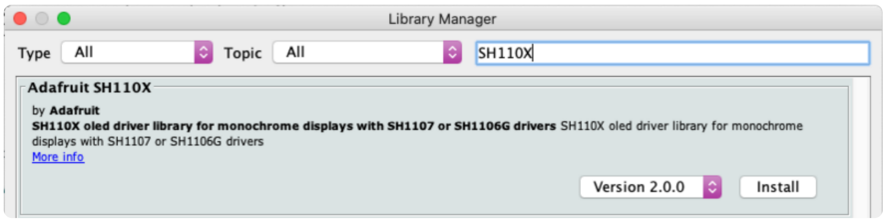
You'll need to install Adafruit SH110X, Adafruit NeoPixel, and RotaryEncoder.

Open the Arduino Library Manager:



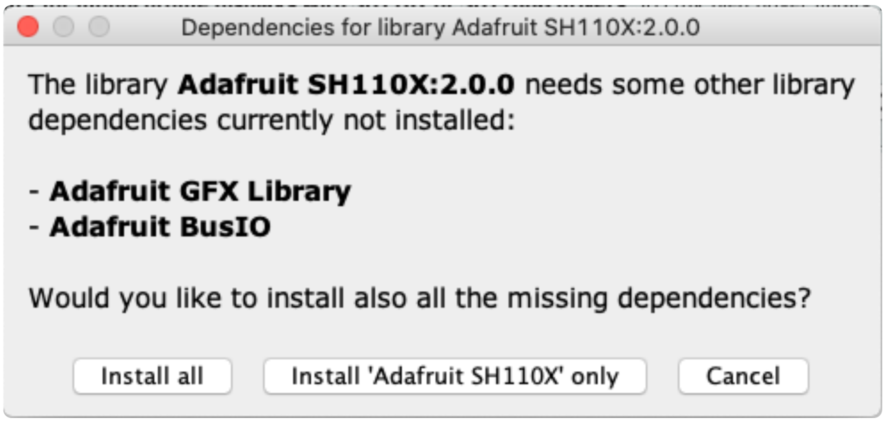
```
void setup() {
  // put your setup code her
}
```

Search for SH110X, and install Adafruit SH110X.



Adafruit SH110X
by Adafruit
SH110X oled driver library for monochrome displays with SH1107 or SH1106G drivers SH110X oled driver library for monochrome displays with SH1107 or SH1106G drivers
More info

When asked to install any dependencies, choose Install all.



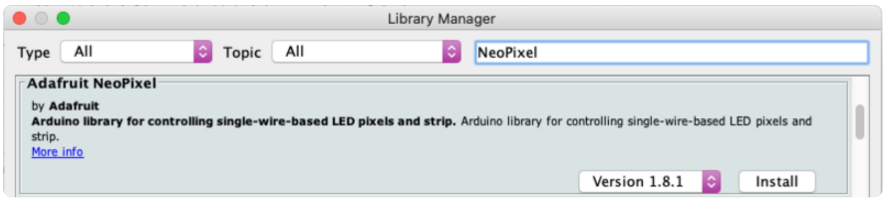
Dependencies for library Adafruit SH110X:2.0.0

The library **Adafruit SH110X:2.0.0** needs some other library dependencies currently not installed:

- **Adafruit GFX Library**
- **Adafruit BusIO**

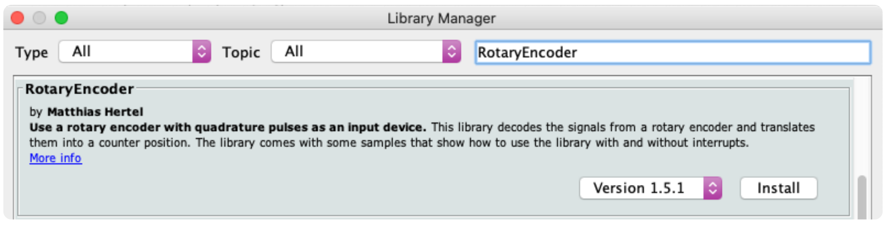
Would you like to install also all the missing dependencies?

Search for NeoPixel and install Adafruit NeoPixel, being sure to double check the name.



Adafruit NeoPixel
by **Adafruit**
Arduino library for controlling single-wire-based LED pixels and strip. Arduino library for controlling single-wire-based LED pixels and strip.
More info
Version 1.8.1 Install

Search for RotaryEncoder and install RotaryEncoder.



RotaryEncoder
by **Matthias Hertel**
Use a rotary encoder with quadrature pulses as an input device. This library decodes the signals from a rotary encoder and translates them into a counter position. The library comes with some samples that show how to use the library with and without interrupts.
More info
Version 1.5.1 Install

Example Code

Here is a precompiled UF2 of the demo code which can be copied to the MacroPad's RPI-RP2 bootloader folder. Click the green button below to download.

MacroPad_Demo.uf2

The code listing for the demo sketch is below to allow recompiling or changing using the Arduinio IDE.

If this is the first Arduino sketch you've loaded on your MacroPad, you may need to manually put it into the bootloader by holding the boot button, pressing reset, and continuing to hold the boot button until the RPI-RP2 drive appears.

```
#include &lt;Adafruit_SH110X.h&gt;
#include &lt;Adafruit_NeoPixel.h&gt;
#include &lt;RotaryEncoder.h&gt;
#include &lt;Wire.h&gt;

// Create the neopixel strip with the built in definitions NUM_NEOPIXEL and
PIN_NEOPIXEL
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUM_NEOPIXEL, PIN_NEOPIXEL, NEO_GRB +
NEO_KHZ800);

// Create the OLED display
Adafruit_SH1106G display = Adafruit_SH1106G(128, 64, &amp;SPI1, OLED_DC, OLED_RST,
OLED_CS);

// Create the rotary encoder
RotaryEncoder encoder(PIN_ROTA, PIN_ROTB, RotaryEncoder::LatchMode::FOUR3);
void checkPosition() {  encoder.tick(); } // just call tick() to check the state.
// our encoder position state
int encoder_pos = 0;

void setup() {
  Serial.begin(115200);
  //while (!Serial) { delay(10); }     // wait till serial port is opened
  delay(100);  // RP2040 delay is not a bad idea

  Serial.println("Adafruit Macropad with RP2040");

  // start pixels!
  pixels.begin();
  pixels.setBrightness(255);
  pixels.show(); // Initialize all pixels to 'off'

  // Start OLED
  display.begin(0, true); // we dont use the i2c address but we will reset!
  display.display();

  // set all mechanical keys to inputs
  for (uint8_t i=0; i&lt;=12; i++) {
    pinMode(i, INPUT_PULLUP);
  }



  // set rotary encoder inputs and interrupts
  pinMode(PIN_ROTA, INPUT_PULLUP);
  pinMode(PIN_ROTB, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(PIN_ROTA), checkPosition, CHANGE);
  attachInterrupt(digitalPinToInterrupt(PIN_ROTB), checkPosition, CHANGE);

  // We will use I2C for scanning the Stemma QT port
  Wire.begin();

  // text display tests
  display.setTextSize(1);
  display.setTextWrap(false);
  display.setTextColor(SH110X_WHITE, SH110X_BLACK); // white text, black background

  // Enable speaker
  pinMode(PIN_SPEAKER_ENABLE, OUTPUT);
  digitalWrite(PIN_SPEAKER_ENABLE, HIGH);
  // Play some tones
  pinMode(PIN_SPEAKER, OUTPUT);
  digitalWrite(PIN_SPEAKER, LOW);
  tone(PIN_SPEAKER, 988, 100);  // tone1 - B5
  delay(100);
  tone(PIN_SPEAKER, 1319, 200); // tone2 - E6
  delay(200);
}

uint8_t j = 0;
bool i2c_found[128] = {false};

void loop() {
  display.clearDisplay();
  display.setCursor(0,0);
  display.println("* Adafruit Macropad *");

  encoder.tick();          // check the encoder
  int newPos = encoder.getPosition();
  if (encoder_pos != newPos) {
    Serial.print("Encoder:");
    Serial.print(newPos);
    Serial.print(" Direction:");
    Serial.println((int)(encoder.getDirection()));
    encoder_pos = newPos;
  }
  display.setCursor(0, 8);
  display.print("Rotary encoder: ");
  display.print(encoder_pos);

  // Scanning takes a while so we don't do it all the time
  if ((j &amp; 0x3F) == 0) {
    Serial.println("Scanning I2C: ");
    Serial.print("Found I2C address 0x");
    for (uint8_t address = 0; address &lt;= 0x7F; address++) {
      Wire.beginTransmission(address);
      i2c_found[address] = (Wire.endTransmission () == 0);
      if (i2c_found[address]) {
        Serial.print("0x");
        Serial.print(address, HEX);
        Serial.print(", ");
      }
    }
    Serial.println();
  }

  display.setCursor(0, 16);
  display.print("I2C Scan: ");
  for (uint8_t address=0; address &lt;= 0x7F; address++) {
    if (!i2c_found[address]) continue;
    display.print("0x");
    display.print(address, HEX);
    display.print(" ");
  }

  // check encoder press
  display.setCursor(0, 24);
  if (!digitalRead(PIN_SWITCH)) {
    Serial.println("Encoder button");
    display.print("Encoder pressed ");
    pixels.setBrightness(255);     // bright!
  } else {
    pixels.setBrightness(80);
  }

  for(int i=0; i&lt; pixels.numPixels(); i++) {
    pixels.setPixelColor(i, Wheel(((i * 256 / pixels.numPixels()) + j) &amp; 255));
  }

  for (int i=1; i&lt;=12; i++) {
    if (!digitalRead(i)) { // switch pressed!
      Serial.print("Switch "); Serial.println(i);
      pixels.setPixelColor(i-1, 0xFFFFFF);  // make white
      // move the text into a 3x4 grid
      display.setCursor(((i-1) % 3)*48, 32 + ((i-1)/3)*8);
      display.print("KEY");
      display.print(i);
    }
  }

  // show neopixels, incredment swirl
  pixels.show();
  j++;

  // display oled
  display.display();
}




// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  if(WheelPos &lt; 85) {
   return pixels.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  } else if(WheelPos &lt; 170) {
   WheelPos -= 85;
   return pixels.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  } else {
   WheelPos -= 170;
   return pixels.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  }
}
```

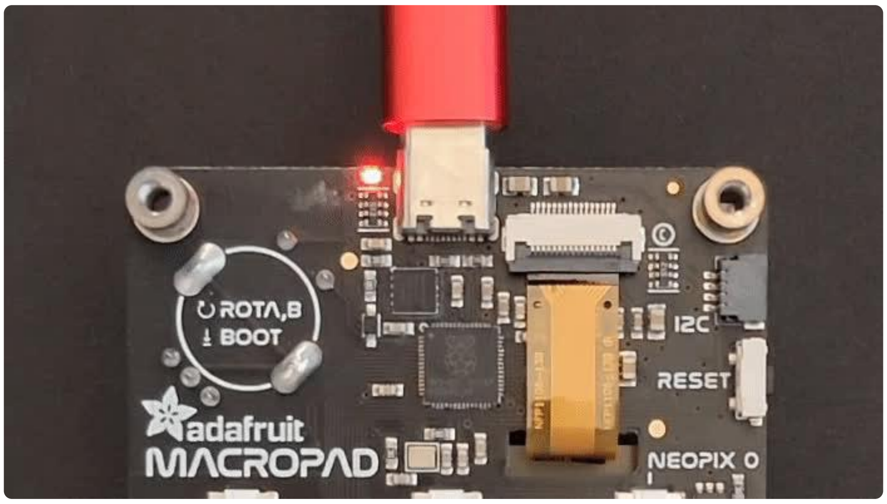
The NeoPixel LEDs light up in a rainbow. Try pressing each key to see a message on the display, and the corresponding pixel turn white. Rotate the rotary encoder to see the value change on the display. Press the rotary encoder to see the NeoPixels get brighter. If you have an I2C device attached via the STEMMA QT port, you'll see the address printed as well.

That's all there is to using the MacroPad with Arduino!

Blink

The first and most basic program you can upload to your Arduino is the classic Blink sketch. This takes something on the board and makes it, well, blink! On and off. It's a great way to make sure everything is working and you're uploading your sketch to the right board and right configuration.

When all else fails, you can always come back to Blink!



Pre-Flight Check: Get Arduino IDE & Hardware Set Up

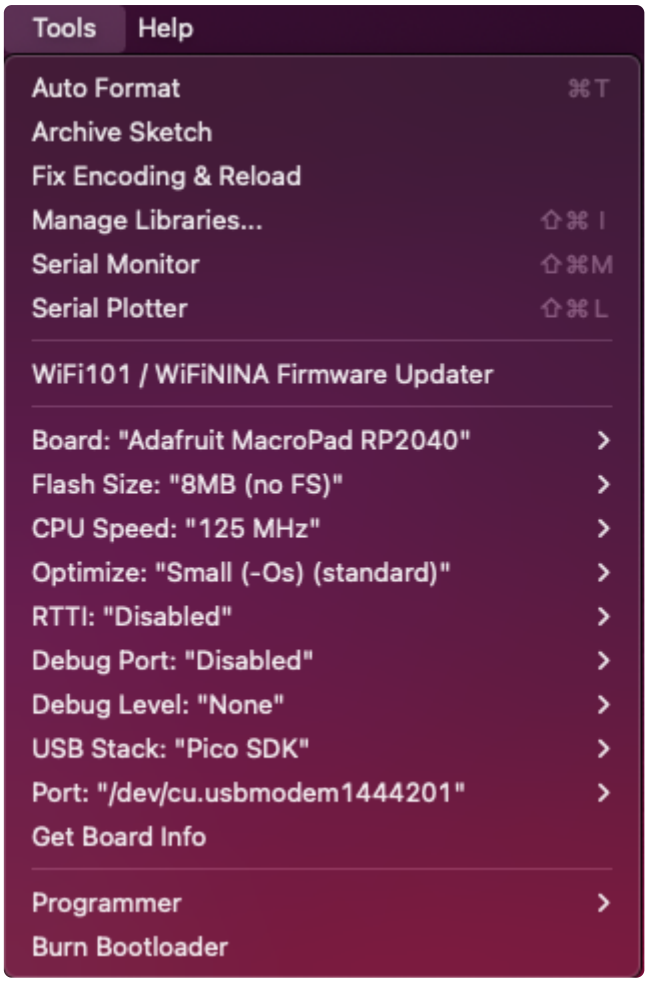
This lesson assumes you have Arduino IDE set up. This is a generalized checklist, some elements may not apply to your hardware. If you haven't yet, check the previous steps in the guide to make sure you:

- Install the very latest Arduino IDE for Desktop (not all boards are supported by the Web IDE so we don't recommend it).
- Install any board support packages (BSP) required for your hardware. Some boards are built in defaults on the IDE, but lots are not! You may need to install plug-in support which is called the BSP.
- Get a Data/Sync USB cable for connecting your hardware. A significant amount of problems folks have stem from not having a USB cable with data pins. Yes, these cursed cables roam the land, making your life hard. If you find a USB cable that doesn't work for data/sync, throw it away immediately! There is no need to keep it around, cables are very inexpensive these days.
- Install any drivers required - If you have a board with a FTDI or CP210x chip, you may need to get separate drivers. If your board has native USB, it probably doesn't need anything. After installing, reboot to make sure the driver sinks in.
- Connect the board to your computer. If your board has a power LED, make sure its lit. Is there a power switch? Make sure its turned On!

Start up Arduino IDE and Select Board/Port

OK now you are prepared! Open the Arduino IDE on your computer. Now you have to tell the IDE what board you are using, and how you want to connect to it.

In the IDE find the Tools menu. You will use this to select the board. If you switch boards, you must switch the selection! So always double-check before you upload code in a new session.


Tools Help

Auto Format ⌘T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... ⇧⌘I
Serial Monitor ⇧⌘M
Serial Plotter ⇧⌘L

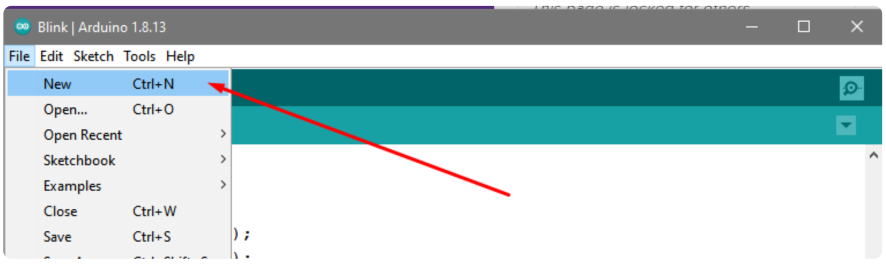
WiFi101 / WiFiNINA Firmware Updater

Board: "Adafruit MacroPad RP2040"
Flash Size: "8MB (no FS)"
CPU Speed: "125 MHz"
Optimize: "Small (-Os) (standard)"
RTTI: "Disabled"
Debug Port: "Disabled"
Debug Level: "None"
USB Stack: "Pico SDK"
Port: "/dev/cu.usbmodem1444201"
Get Board Info

Programmer
Burn Bootloader

New Blink Sketch

OK lets make a new blink sketch! From the File menu, select New



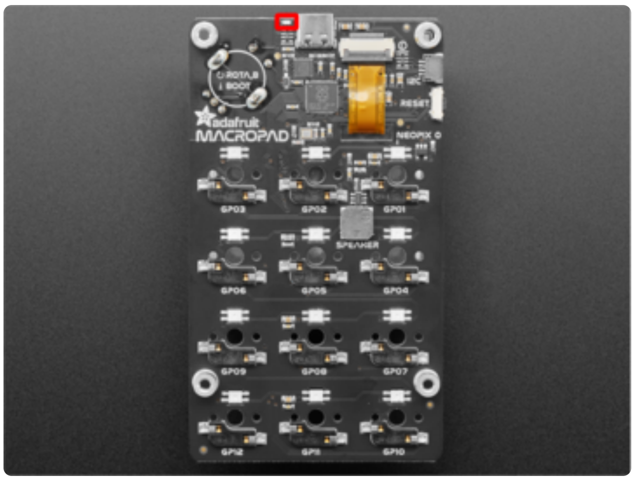
Then in the new window, copy and paste this text:

```
int led = LED_BUILTIN;

void setup() {
  // Some boards work best if we also make a serial connection
  Serial.begin(115200);

  // set LED to be an output pin
  pinMode(led, OUTPUT);
}
```

```
void loop() {
  // Say hi!
  Serial.println("Hello!");

  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(500);                // wait for a half second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(500);                // wait for a half second
}
```

Note that in this example, we are not only blinking the LED but also printing to the Serial monitor, think of it as a little bonus to test the serial connection.

One note you'll see is that we reference the LED with the constant `LED_BUILTIN` rather than a number. That's because, historically, the built in LED was on pin 13 for Arduinos. But in the decades since, boards don't always have a pin 13, or maybe it could not be used for an LED. So the LED could have moved to another pin. It's best to use `LED_BUILTIN` so you don't get the pin number confused!



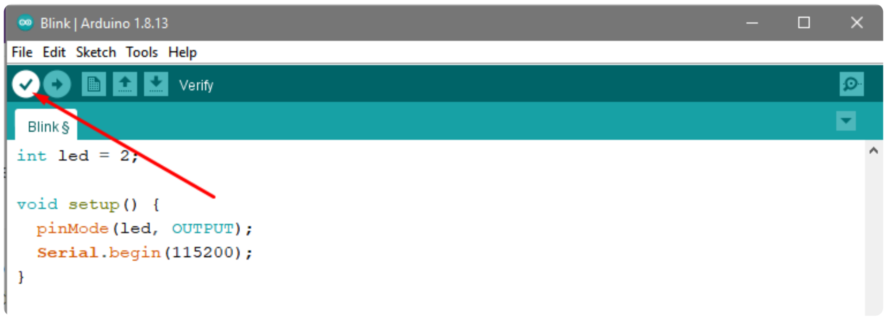
The red LED is located to the left of the USB connector.

The LED on the MacroPad is on pin #13.

Verify (Compile) Sketch

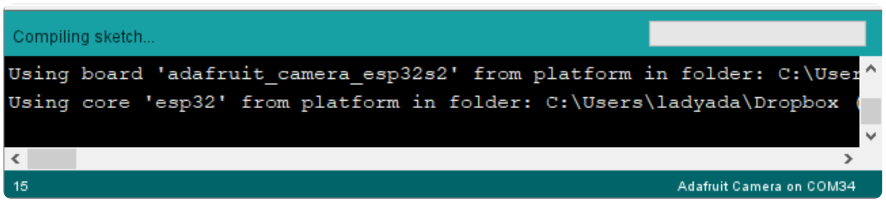
OK now you can click the Verify button to convert the sketch into binary data to be uploaded to the board.

Note that Verifying a sketch is the same as Compiling a sketch - so we will use the words interchangeably



```
int led = 2;

void setup() {
  pinMode(led, OUTPUT);
  Serial.begin(115200);
}
```

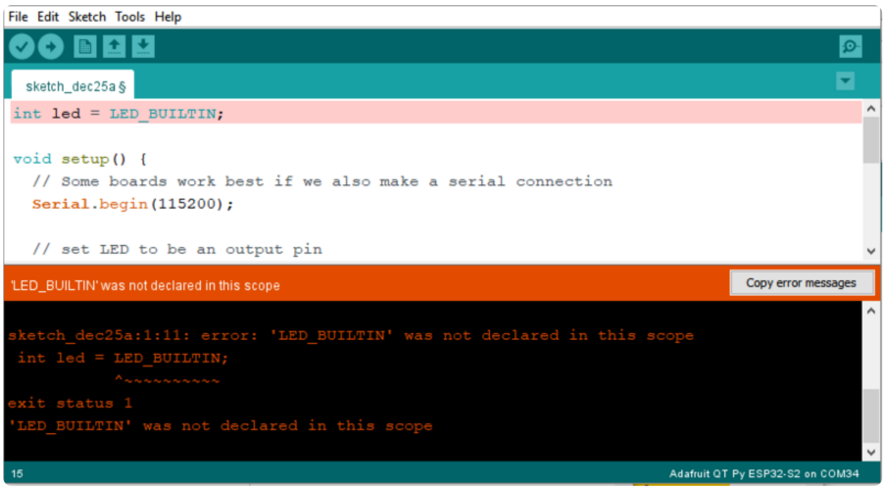
During verification/compilation, the computer will do a bunch of work to collect all the libraries and code and the results will appear in the bottom window of the IDE.



```
Using board 'adafruit_camera_esp32s2' from platform in folder: C:\User
Using core 'esp32' from platform in folder: C:\Users\ladyada\Dropbox
```

If something went wrong with compilation, you will get red warning/error text in the bottom window letting you know what the error was. It will also highlight the line with an error.

For example, here I had the wrong board selected - and the selected board does not have a built in LED!

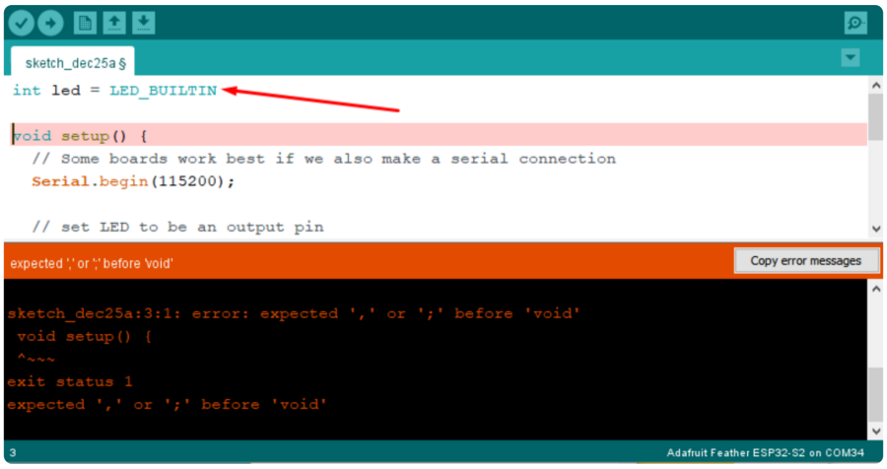


```
int led = LED_BUILTIN;

void setup() {
  // Some boards work best if we also make a serial connection
  Serial.begin(115200);

  // set LED to be an output pin
```

'LED_BUILTIN' was not declared in this scope

```
sketch_dec25a:1:11: error: 'LED_BUILTIN' was not declared in this scope
 int led = LED_BUILTIN;
           ^~~~~~~~~~~
exit status 1
'LED_BUILTIN' was not declared in this scope
```

Here's another common error, in my haste I forgot to add a `;` at the end of a line. The compiler warns me that it's looking for one - note that the error is actually a few lines up!


sketch_dec25a §

```
int led = LED_BUILTIN

void setup() {
  // Some boards work best if we also make a serial connection
  Serial.begin(115200);

  // set LED to be an output pin
```

expected ',' or ';' before 'void'

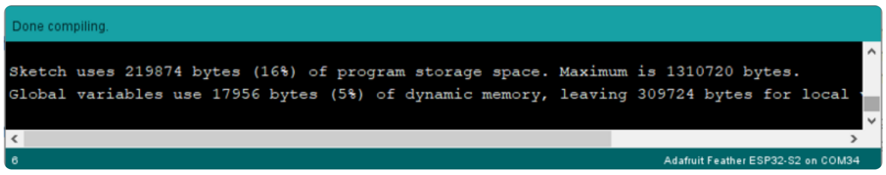
Copy error messages

```
sketch_dec25a:3:1: error: expected ',' or ';' before 'void'
 void setup() {
 ^~~~
exit status 1
expected ',' or ';' before 'void'
```

3 Adafruit Feather ESP32-S2 on COM34

Turning on detailed compilation warnings and output can be very helpful sometimes - Its in Preferences under "Show Verbose Output During:" and check the Compilation button. If you ever need to get help from others, be sure to do this and then provide all the text that is output. It can assist in nailing down what happened!

On success you will see something like this white text output and the message Done compiling. in the message area.

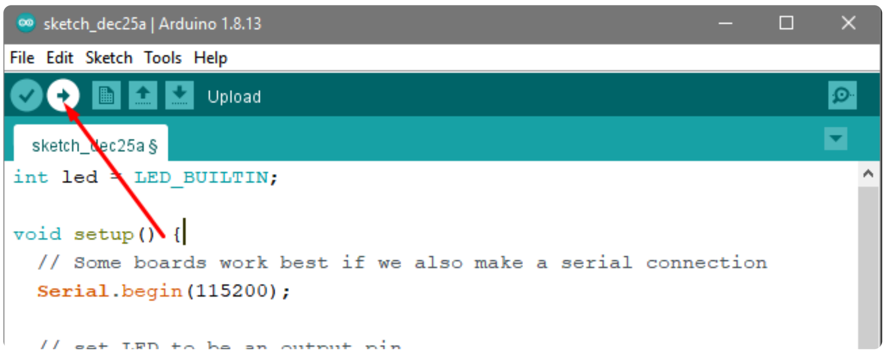

Done compiling.

```
Sketch uses 219874 bytes (16%) of program storage space. Maximum is 1310720 bytes.
Global variables use 17956 bytes (5%) of dynamic memory, leaving 309724 bytes for local
```

6 Adafruit Feather ESP32-S2 on COM34

Upload Sketch

Once the code is verified/compiling cleanly you can upload it to your board. Click the Upload button.

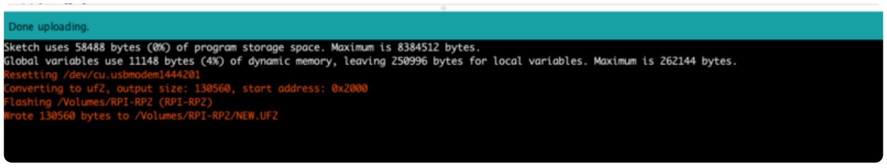


```
int led = LED_BUILTIN;

void setup() {
  // Some boards work best if we also make a serial connection
  Serial.begin(115200);

  // set LED to be an output pin
```

The IDE will try to compile the sketch again for good measure, then it will try to connect to the board and upload a the file.

This is actually one of the hardest parts for beginners because it's where a lot of things can go wrong.

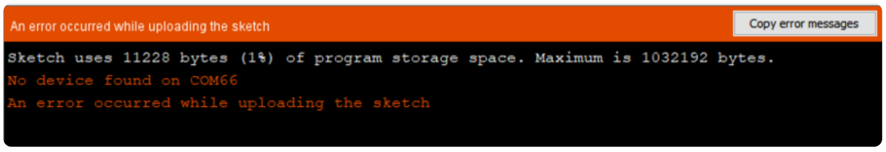
However, lets start with what it looks like on success! Here's what your board upload process looks like when it goes right:


Done uploading.

```
Sketch uses 58488 bytes (0%) of program storage space. Maximum is 8384512 bytes.
Global variables use 11148 bytes (4%) of dynamic memory, leaving 250996 bytes for local variables. Maximum is 262144 bytes.
Resetting /dev/cu.usbmodem1444201
Converting to uf2, output size: 130560, start address: 0x2000
Flashing /Volumes/RPI-RP2 (RPI-RP2)
Wrote 130560 bytes to /Volumes/RPI-RP2/NEW.UF2
```

Often times you will get a warning like this, which is kind of vague:

`No device found on COM66` (or whatever port is selected)
`An error occurred while uploading the sketch`


An error occurred while uploading the sketch

```
Sketch uses 11228 bytes (1%) of program storage space. Maximum is 1032192 bytes.
No device found on COM66
An error occurred while uploading the sketch
```

This could be a few things.

First up, check again that you have the correct board selected! Many electronics boards have very similar names or look, and often times folks grab a board different from what they thought.

If you're positive the right board is selected, we recommend the next step is to put the board into manual bootloading mode.

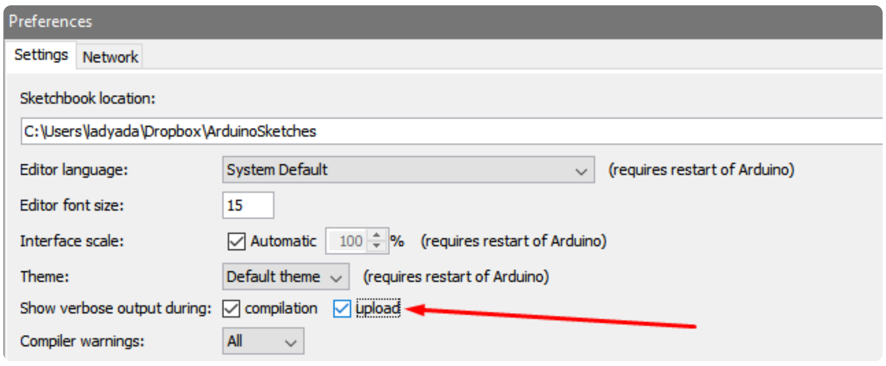
Native USB and manual bootloading

Historically, microcontroller boards contained two chips: the main micro chip (say, ATmega328 or ESP8266 or ESP32) and a separate chip for USB interface that would be used for bootloading (a CH430, FT232, CP210x, etc). With these older designs, the microcontroller is put into a bootloading state for uploading code by the separate chip. It allows for easier uploading but is more expensive as two chips are needed, and also the microcontroller can't act like a keyboard or disk drive.

Modern chips often have 'native' USB - that means that there is no separate chip for USB interface. It's all in one! Great for cost savings, simplicity of design, reduced size and more control. However, it means the chip must be self-aware enough to be able to put itself into bootload/upload mode on its own. That's fine 99% of the time but is very likely you will at some point get the board into an odd state that makes it too confused to bootload.

A lot of beginners have a little freakout the first time this happens, they think the board is ruined or 'bricked' - it's almost certainly not, it is just crashed and/or confused. You may need to perform a little trick to get the board back into a good state, at which point you won't need to manually bootload again.

Before continuing we really, really suggest turning on Verbose Upload messages, it will help in this process because you will be able to see what the IDE is trying to do. It's a checkbox in the Preferences menu.



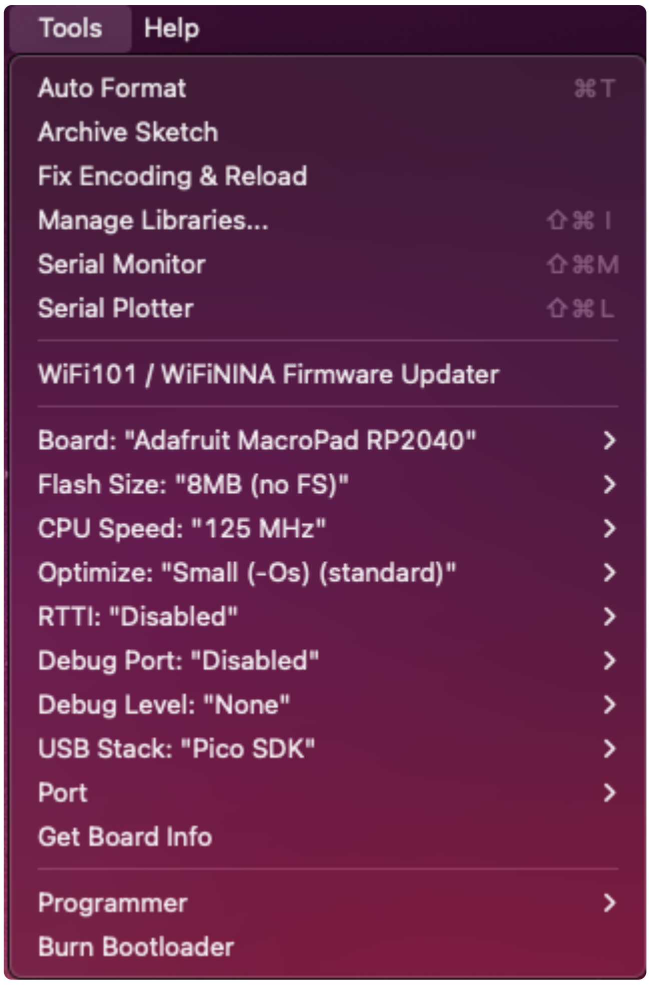
Sketchbook location:
C:\Users\ladyada\Dropbox\ArduinoSketches
Editor language: System Default (requires restart of Arduino)
Editor font size: 15
Interface scale: ☑ Automatic 100 % (requires restart of Arduino)
Theme: Default theme (requires restart of Arduino)
Show verbose output during: ☑ compilation ☑ upload
Compiler warnings: All

Enter Manual Bootload Mode

OK now you know it's probably time to try manual bootloading. No problem! Here is how you do that for this board:

To enter the bootloader, hold down the BOOT button (which is pressing the rotary encoder!), and while continuing to hold it (don't let go!), press and release the reset button. Continue to hold the BOOT button until the RPI-RP2 drive appears!

Once you are in manual bootload mode, go to the Tools menu, and make sure you have selected the bootloader serial port. It is almost certain that the serial port has changed now that the bootloader is enabled


Tools Help

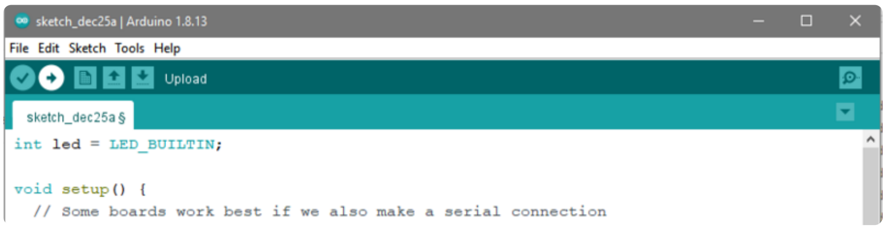
Auto Format ⌘T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... ⇧⌘I
Serial Monitor ⇧⌘M
Serial Plotter ⇧⌘L

WiFi101 / WiFiNINA Firmware Updater

Board: "Adafruit MacroPad RP2040"
Flash Size: "8MB (no FS)"
CPU Speed: "125 MHz"
Optimize: "Small (-Os) (standard)"
RTTI: "Disabled"
Debug Port: "Disabled"
Debug Level: "None"
USB Stack: "Pico SDK"
Port
Get Board Info

Programmer
Burn Bootloader

Now you can try uploading again!


sketch_dec25a | Arduino 1.8.13

File Edit Sketch Tools Help

Upload

sketch_dec25a §

```
int led = LED_BUILTIN;

void setup() {
  // Some boards work best if we also make a serial connection
```

Did you remember to select the new Port in the Tools menu since the bootloader port has changed?

This time, you should have success!

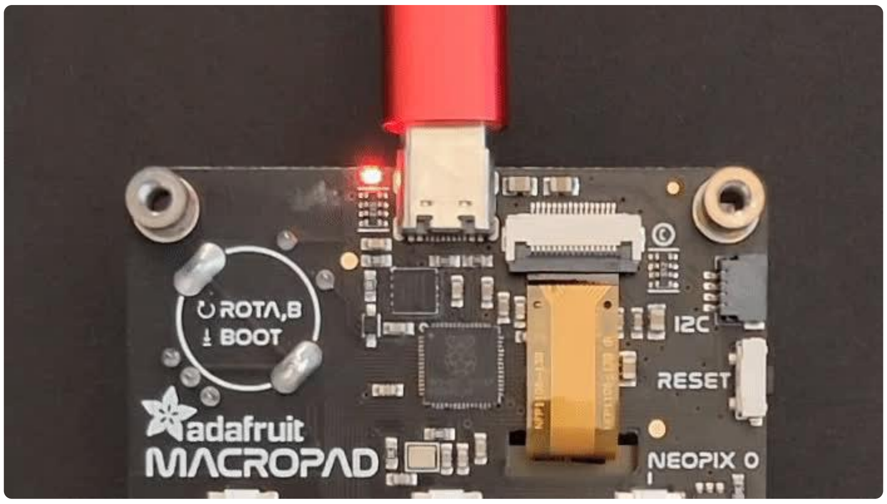
After uploading this way, be sure to click the reset button - it sort of makes sure that the board got a good reset and will come back to life nicely.

After uploading with Manual Bootloader - don't forget to re-select the old Port again

It's also a good idea to try to re-upload the sketch again now that you've performed a manual bootload to get the chip into a good state. It should perform an auto-reset the second time, so you don't have to manually bootload again.

Finally, a Blink!

OK it was a journey but now we're here and you can enjoy your blinking LED. Next up, try to change the delay between blinks and re-upload. It's a good way to make sure your upload process is smooth and practiced.



I2C Scan Test

A lot of sensors, displays, and devices can connect over I2C. I2C is a 2-wire 'bus' that allows multiple devices to all connect on one set of pins so it's very convenient for wiring!

When using your board, you'll probably want to connect up I2C devices, and it can be a little tricky the first time. The best way to debug I2C is go through a checklist and then perform an I2C scan

Common I2C Connectivity Issues

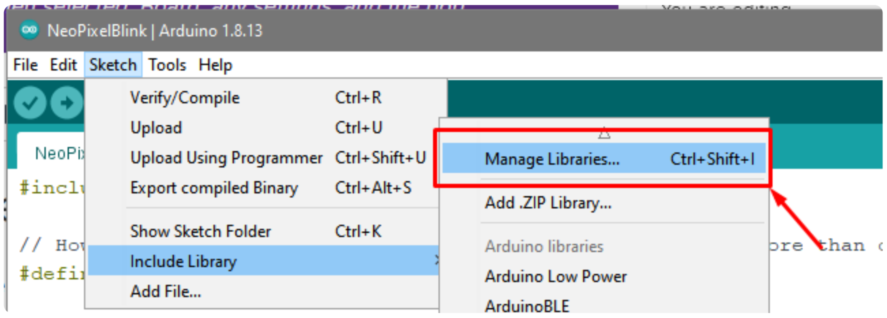
- Have you connected four wires (at a minimum) for each I2C device? Power the device with whatever is the logic level of your microcontroller board (probably 3.3V), then a ground wire, and a SCL clock wire, and and a SDA data wire.
- If you're using a STEMMA QT board - check if the power LED is lit. It's usually a green LED to the left side of the board.

- Does the STEMMA QT/I2C port have switchable power or pullups? To reduce power, some boards have the ability to cut power to I2C devices or the pullup resistors. Check the documentation if you have to do something special to turn on the power or pullups.
- If you are using a DIY I2C device, do you have pullup resistors? Many boards do not have pullup resistors built in and they are required! We suggest any common 2.2K to 10K resistors. You'll need two: one each connects from SDA to positive power, and SCL to positive power. Again, positive power (a.k.a VCC, VDD or V+) is often 3.3V
- Do you have an address collision? You can only have one board per address. So you cannot, say, connect two AHT20's to one I2C port because they have the same address and will interfere. Check the sensor or documentation for the address. Sometimes there are ways to adjust the address.
- Does your board have multiple I2C ports? Historically, boards only came with one. But nowadays you can have two or even three! This can help solve the "hey, but what if I want two devices with the same address" problem: just put one on each bus.
- Are you hot-plugging devices? I2C does not support dynamic re-connection, you cannot connect and disconnect sensors as you please. They should all be connected on boot and not change. (Only exception is if you're using a hot-plug assistant but that'll cost you ()).
- Are you keeping the total bus length reasonable? I2C was designed for maybe 6" max length. We like to push that with plug-n-play cables, but really please keep them as short as possible! (Only exception is if you're using an active bus extender ()).

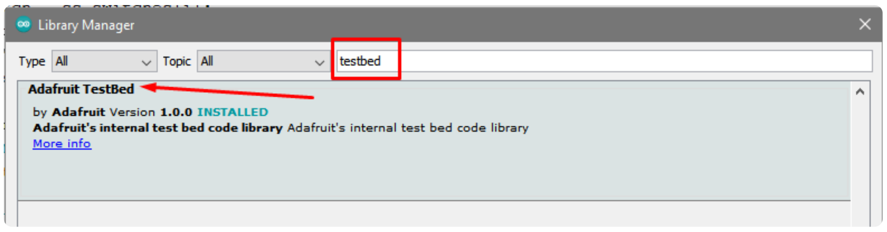
Perform an I2C scan!

Install TestBed Library

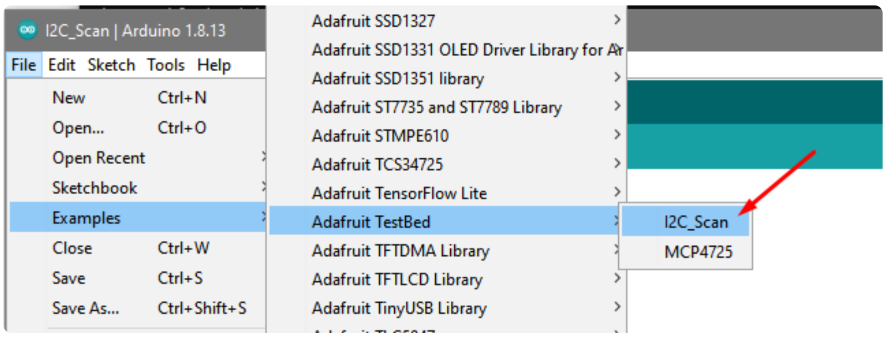
To scan I2C, the Adafruit TestBed library is used. This library and example just makes the scan a little easier to run because it takes care of some of the basics. You will need to add support by installing the library. Good news: it is very easy to do it. Go to the Arduino Library Manager.


NeoPixelBlink | Arduino 1.8.13
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Manage Libraries... Ctrl+Shift+I
Add .ZIP Library...
Arduino libraries
Arduino Low Power
ArduinoBLE

Search for TestBed and install the Adafruit TestBed library


Library Manager
Type All
Topic All
testbed
Adafruit TestBed
by Adafruit Version 1.0.0 INSTALLED
Adafruit's internal test bed code library Adafruit's internal test bed code library
More info

Now open up the I2C Scan example

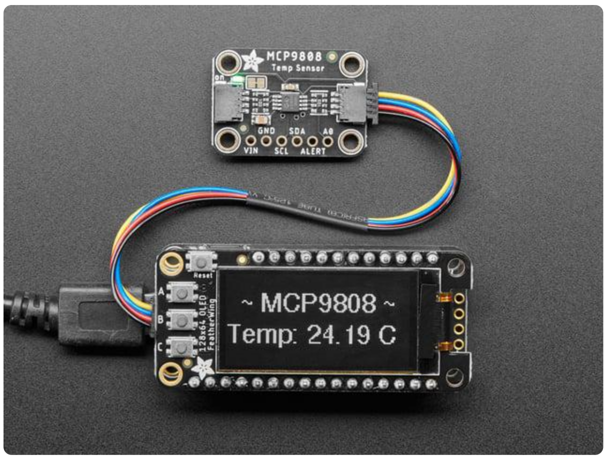


```
#include <Adafruit_TestBed.h>
extern Adafruit_TestBed TB;

#define DEFAULT_I2C_PORT &Wire

// Some boards have TWO I2C ports, how nifty. We should scan both
#if defined(ARDUINO_ARCH_RP2040) \
    || defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) \
    || defined(ARDUINO_ADAFRUIT_QTPY_ESP32S3_NOPSRAM) \
    || defined(ARDUINO_ADAFRUIT_QTPY_ESP32S3) \
    || defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO) \
    || defined(ARDUINO_SAM_DUE)
  #define SECONDARY_I2C_PORT &Wire1
#endif

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

  // Wait for Serial port to open
  while (!Serial) {
    delay(10);
  }
  delay(500);
  Serial.println("Adafruit I2C Scanner");

#if defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) || \
    defined(ARDUINO_ADAFRUIT_QTPY_ESP32S3_NOPSRAM) || \
    defined(ARDUINO_ADAFRUIT_QTPY_ESP32S3) || \
    defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
  // ESP32 is kinda odd in that secondary ports must be manually
  // assigned their pins with setPins()!
  Wire1.setPins(SDA1, SCL1);
#endif

#if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2)
  // turn on the I2C power by setting pin to opposite of 'rest state'
  pinMode(PIN_I2C_POWER, INPUT);
  delay(1);
  bool polarity = digitalRead(PIN_I2C_POWER);
  pinMode(PIN_I2C_POWER, OUTPUT);
  digitalWrite(PIN_I2C_POWER, !polarity);
#endif

#if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2_TFT)
  pinMode(TFT_I2C_POWER, OUTPUT);
  digitalWrite(TFT_I2C_POWER, HIGH);
#endif

#if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2_REVTFT)
  pinMode(TFT_I2C_POWER, OUTPUT);
  digitalWrite(TFT_I2C_POWER, HIGH);
#endif

#if defined(ADAFRUIT_FEATHER_ESP32_V2)
  // Turn on the I2C power by pulling pin HIGH.
  pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_I2C_POWER, HIGH);
#endif
}

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

  Serial.print("Default port (Wire) ");
  TB.theWire = DEFAULT_I2C_PORT;
  TB.printI2CBusScan();

#if defined(SECONDARY_I2C_PORT)
  Serial.print("Secondary port (Wire1) ");
  TB.theWire = SECONDARY_I2C_PORT;
  TB.printI2CBusScan();
#endif

  delay(3000); // wait 3 seconds
}
```

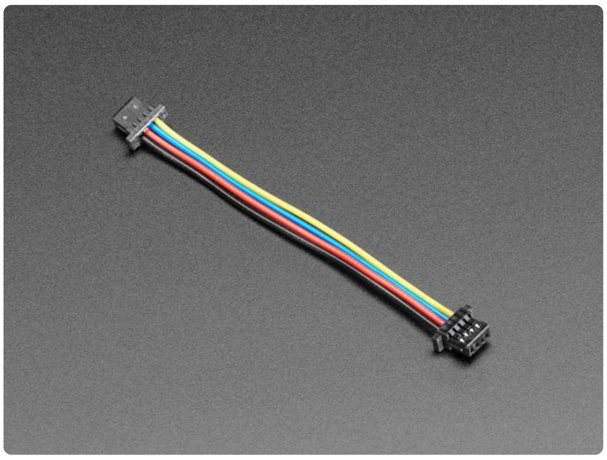
Wire up I2C device

While the examples here will be using the Adafruit MCP9808 (), a high accuracy temperature sensor, the overall process is the same for just about any I2C sensor or device.

The first thing you'll want to do is get the sensor connected so your board has I2C to talk to.



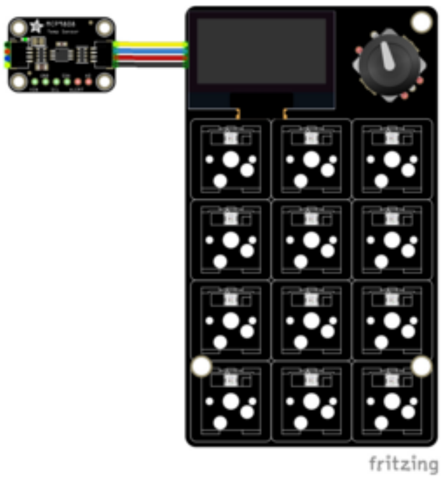
Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout
The MCP9808 digital temperature sensor is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25°C over the sensor's -40°C to...
https://www.adafruit.com/product/5027



STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long
This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...
https://www.adafruit.com/product/4399

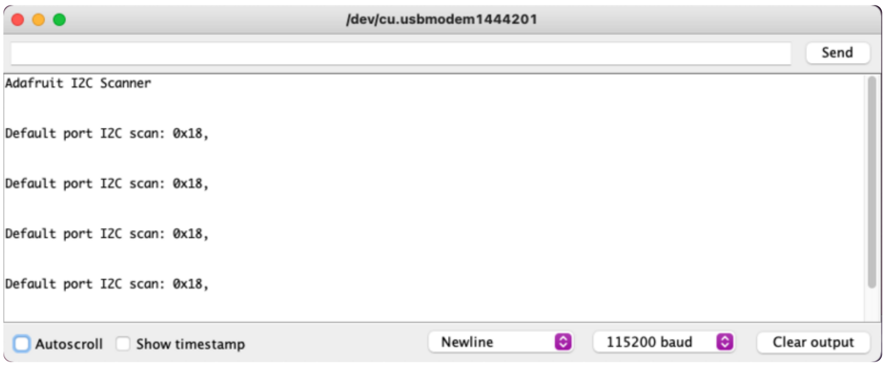
Wiring the MCP9808

The MCP9808 comes with a STEMMA QT connector, which makes wiring it up quite simple and solder-free.



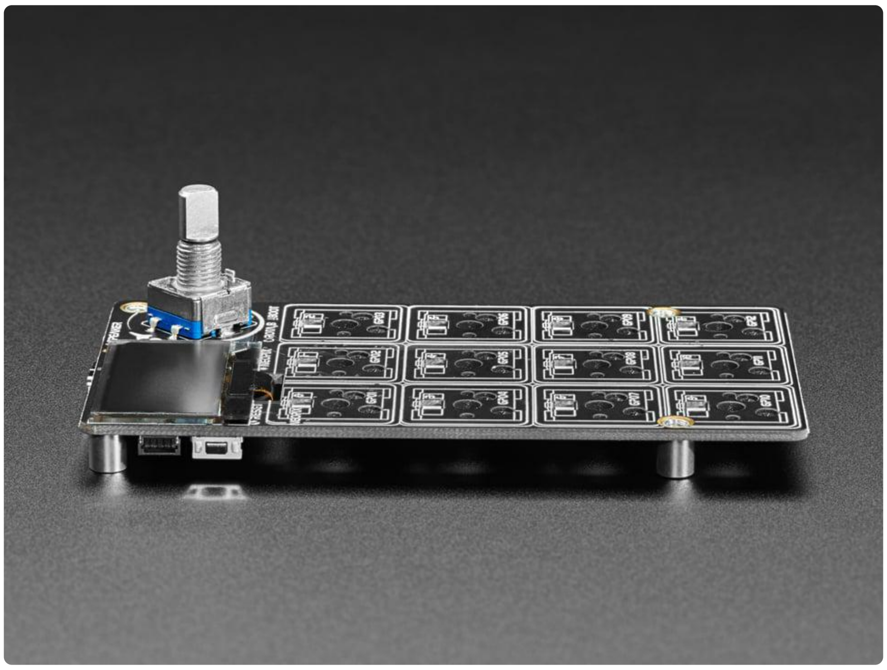
Simply connect the STEMMA QT cable from the STEMMA QT port on your board to the STEMMA QT port on the MCP9808.

Now upload the scanning sketch to your microcontroller and open the serial port to see the output. You should see something like this:



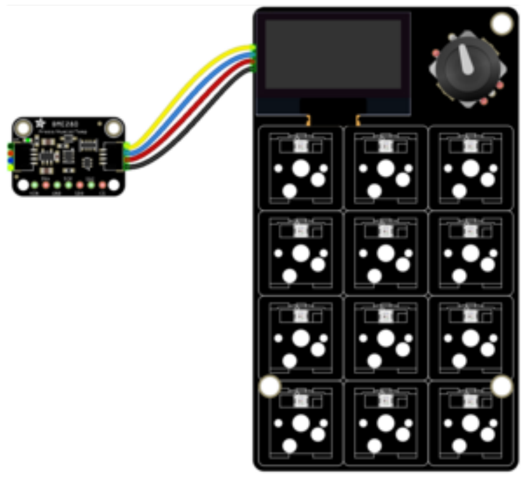
```
Adafruit I2C Scanner


Default port I2C scan: 0x18,


Default port I2C scan: 0x18,


Default port I2C scan: 0x18,


Default port I2C scan: 0x18,
```

Attaching External Hardware



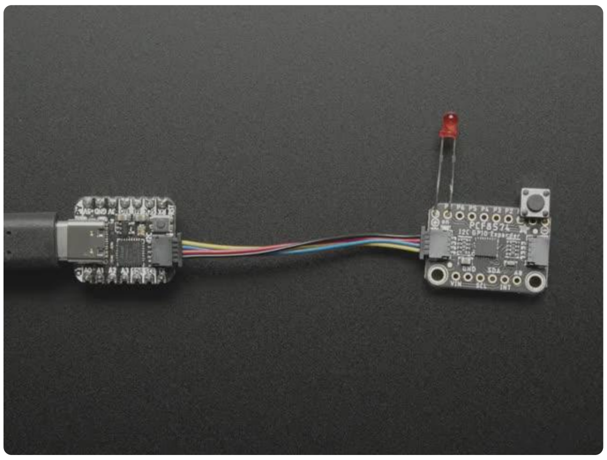
The MacroPad is packed with onboard hardware, but what if you want to use it with e ven more hardware? The answer is to use the STEMMA QT port, located on the back left-hand side of the MacroPad underneath the OLED screen.

I2C Devices



The STEMMA QT port is best for use with I2C devices. You can use any of the Qwiic-compatible I2C breakout boards () available by plugging them into this port and coding them with either the available CircuitPython or Arduino drivers and libraries.

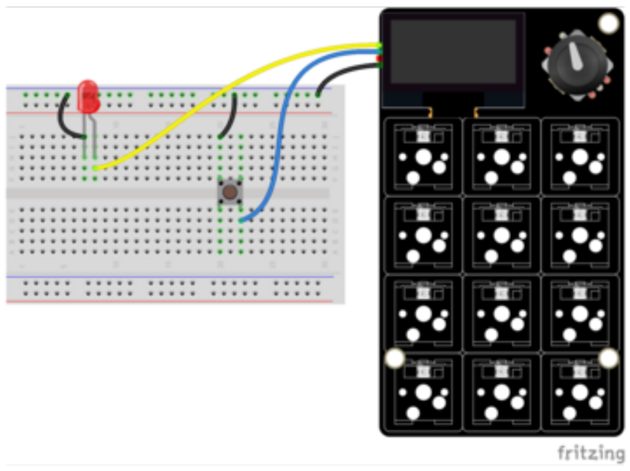
An I2C to GPIO Expander board may be added for more discrete GPIO pin availability if more than two GPIO are needed. An example:



Adafruit PCF8574 I2C GPIO Expander Breakout
Expand your project possibilities, with the Adafruit PCF8574 GPIO Expander Breakout - an affordable 8 channel I2C expander.GPIO expanders...
https://www.adafruit.com/product/5545

STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long
This 4-wire cable is a little over 100mm / 4" long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of...
https://www.adafruit.com/product/4210

SCL and SDA as GPIO



You can use either of the SCL and SDA pins from the STEMMA QT port as discreet GPIO, as long as they are not sharing a bus.

For example, you could connect a button's input to SDA and an LED's anode to SCL, with the STEMMA port's GND connecting to both components. In CircuitPython, these pins are accessed with `board.SDA` and `board.SCL`. In Arduino, SDA is pin `20` and SCL is pin `21`.

It is not possible to add BLE or WiFi at this time to the MacroPad.

Downloads

Files

- RP2040 Datasheet ()
- SH1106 Monochrome OLED datasheet ()
- EagleCAD PCB files on GitHub ()
- 3D models on GitHub ()
- Fritzing object in the Adafruit Fritzing Library ()
- PDF PrettyPins pinout image ()
- SVG PrettyPins pinout image ()

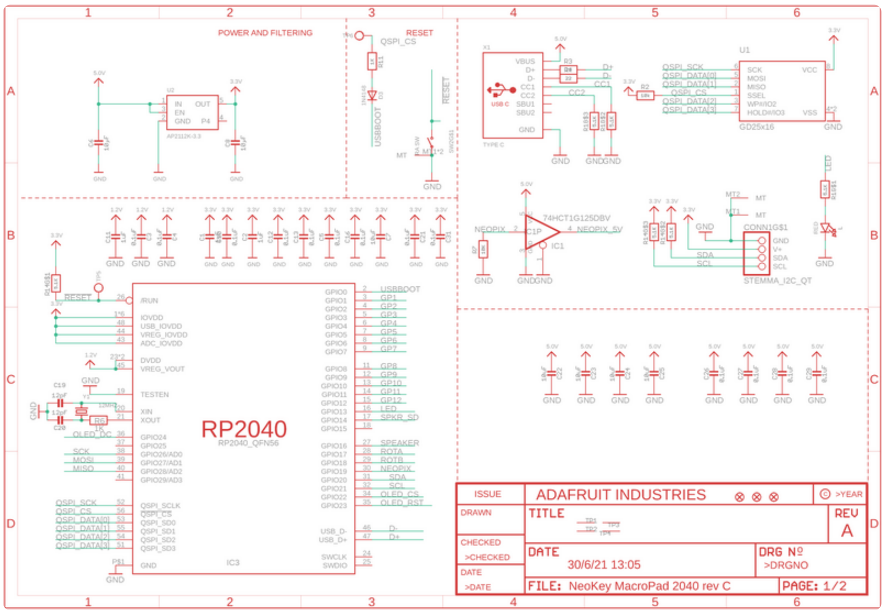
Original MacroPad Demo

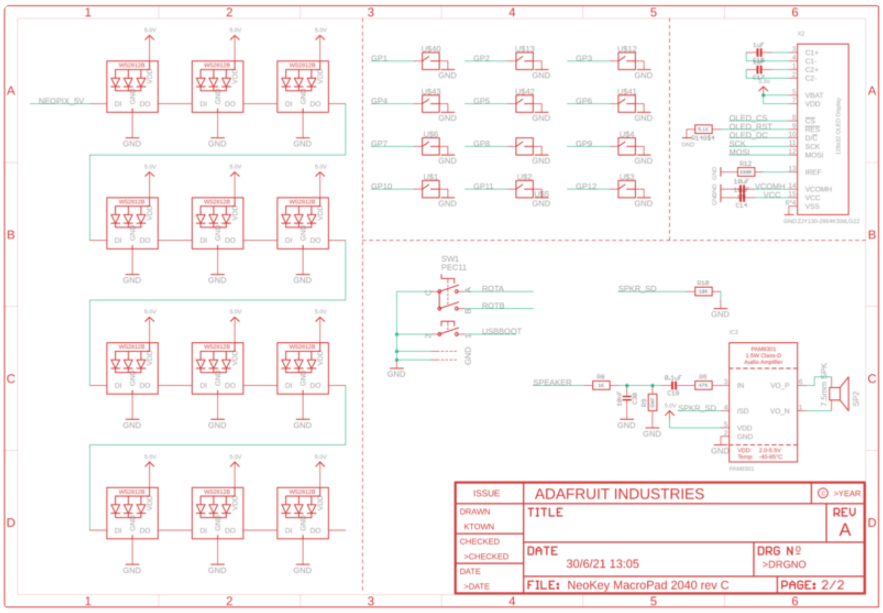
The code that shipped on the MacroPad is the example included in the Arduino section () of this guide.

Alternatively, you can download the following UF2 file and load it onto your MacroPad.

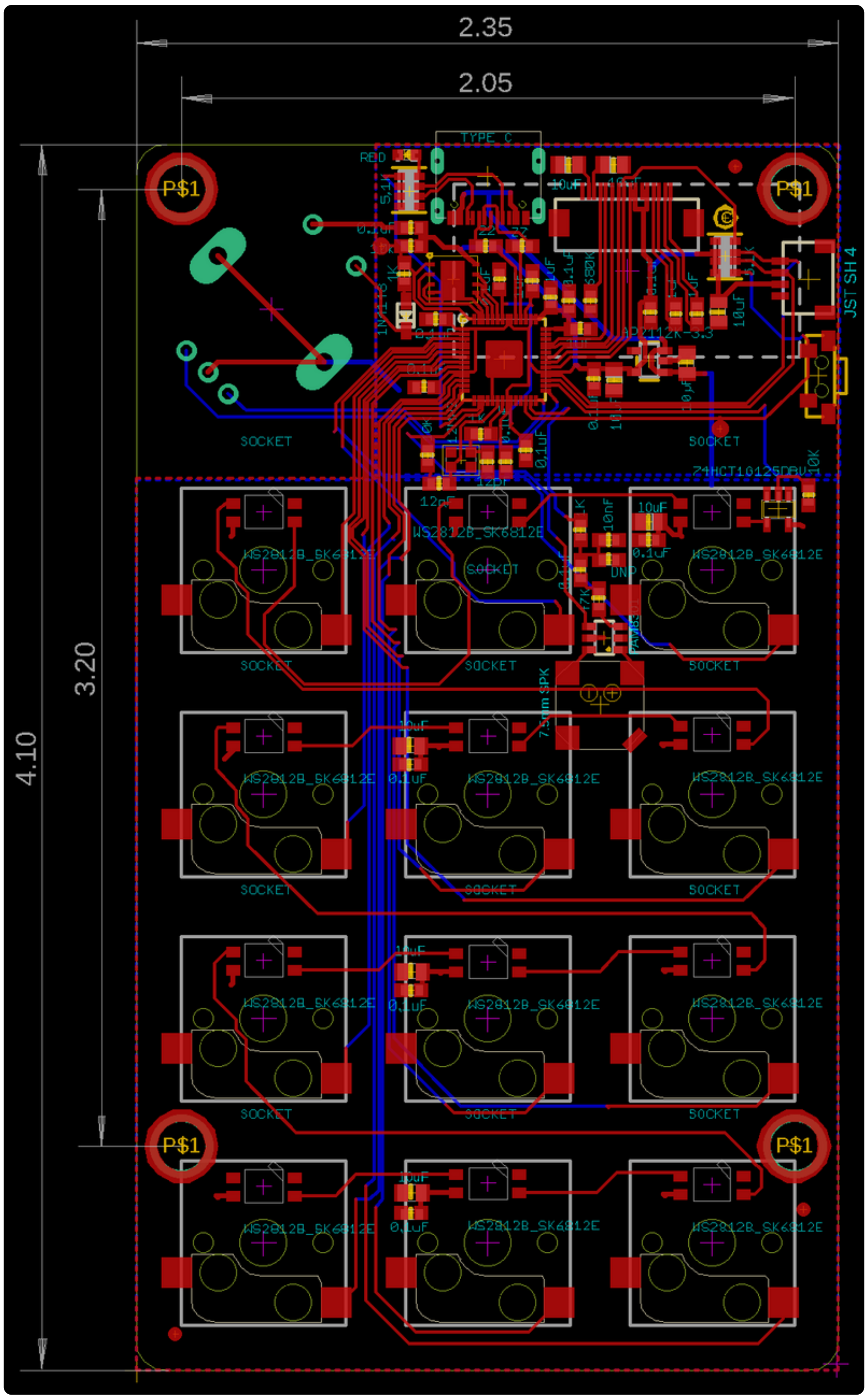
macropad_demo.uf2

Schematic





Fab Print


2.35
2.05
3.20
4.10
P$1
TYPE C
RED
SOCKET
WS2812B_SK6812E
JST SH 4
AP2112K-3.3
74HCT1G125DBV
7.5mm SPK
10uF
0.1uF
12pF
10nF
DNP
10K

3D Model