Tiny LED WiFi Companion Cube

Created by Charlyn G

https://learn.adafruit.com/tiny-led-wifi-cube

Last updated on 2024-06-03 03:40:33 PM EDT

Table of Contents

Overview 3

- Prerequisite Reading
- Parts

3D Printing 6

Circuit Diagram 8

Setup AdafruitIO and custom webpage 9

- Send test messages
- Custom webpage

Prepare parts and initial soldering 13

- Prepare the power circuit
- Prepare the QT Py
- Solder the battery wires

Solder and assemble LEDs 19

CircuitPython Setup 27

- Additional files

CircuitPython Code 28

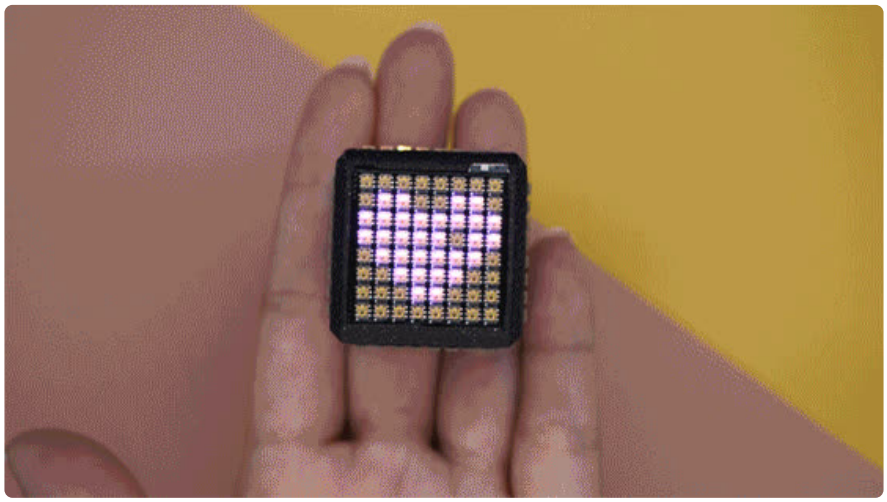
- Code.py
- Cube.py
- Testing before final assembly

Final assembly 36

Resources 43

Overview

Getting a message on a screen is great, but getting them on a cube is BETTER. Build an adorable message cube to accompany you on life's adventures and deliver words from the internet via AdafruitIO!



This cube is powered by CircuitPython on a fancy ESP32-S3 QT Py! It has a LIS3DH accelerometer sensor board attached via a STEMMA QT connector, which means it's easy to change the sensor anytime you need new cube superpowers. The QT Py BFF charger add-on makes this cube rechargeable via the USB-C port of the QT Py.

These six Dotstar LED matrices pack the cutest punch all into this tiny package! Ready to display 8x8 pixel art, a random wise quote, or anything else you might want.

Will it have the longest battery life? No. Will it be super fun to carry around? ABSOLUTELY.

Note: This tutorial is labeled intermediate because there's a bit of fine detail soldering and desoldering involved to be able to stuff everything inside the cube, and we'll be dealing with live battery wires. Use caution!

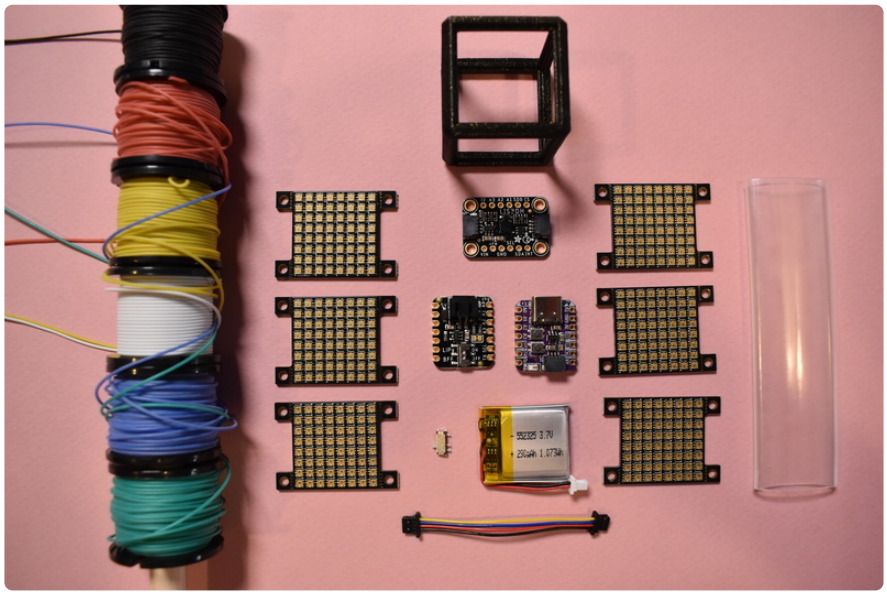
Prerequisite Reading

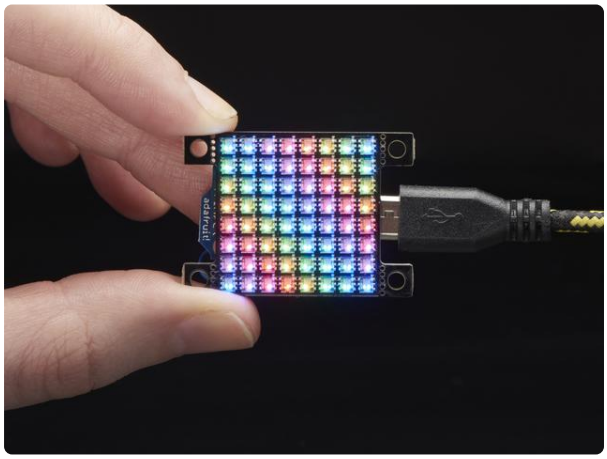
You might want to brush up on some soldering skills before tackling this project!

- Adafruit Guide To Excellent Soldering (https://adafru.it/drl)
- Collin's Lab: Soldering (https://adafru.it/dyT)

Let's go make ourselves a glowy cube!

Parts

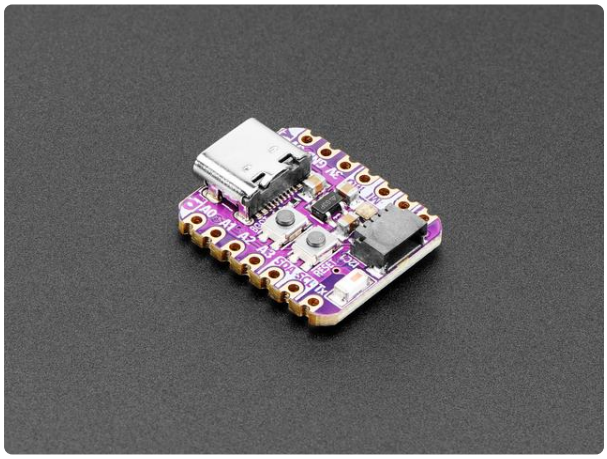




Adafruit DotStar High Density 8x8 Grid - 64 RGB LED Pixel Matrix

Do not eat this LED grid just because it is so colorful and bite-sized! This is the tiniest little LED grid we could make, with 64 full RGB color pixels in a square that is only...

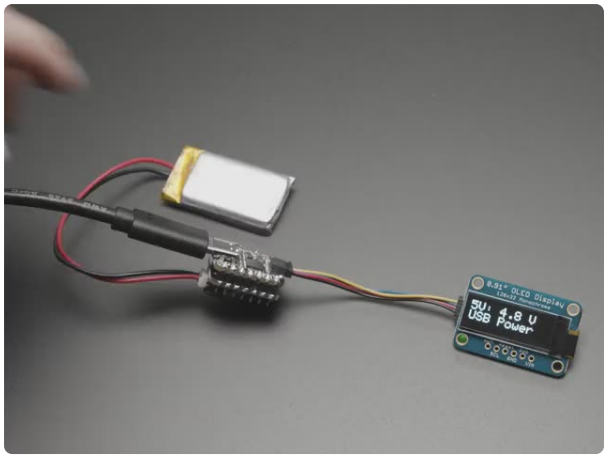
https://www.adafruit.com/product/3444



Adafruit QT Py ESP32-S3 WiFi Dev Board with STEMMA QT

The ESP32-S3 has arrived in QT Py format - and what a great way to get started with this powerful new chip from Espressif! With dual 240 MHz cores, WiFi and BLE support, and native...

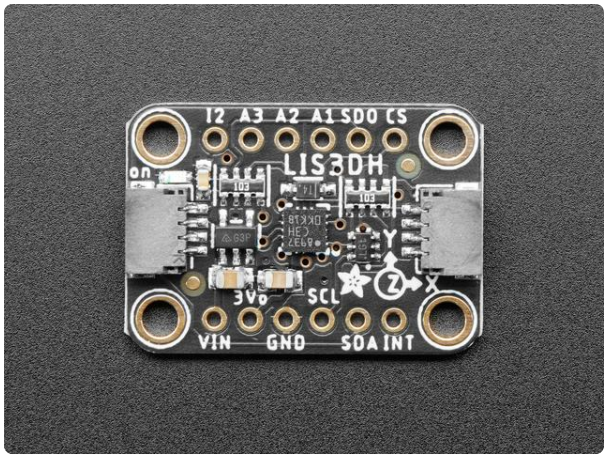
https://www.adafruit.com/product/5426



Adafruit LiIon or LiPoly Charger BFF Add-On for QT Py

Is your QT Py all alone, lacking a friend to travel the wide world with? When you were a kid you may have learned...

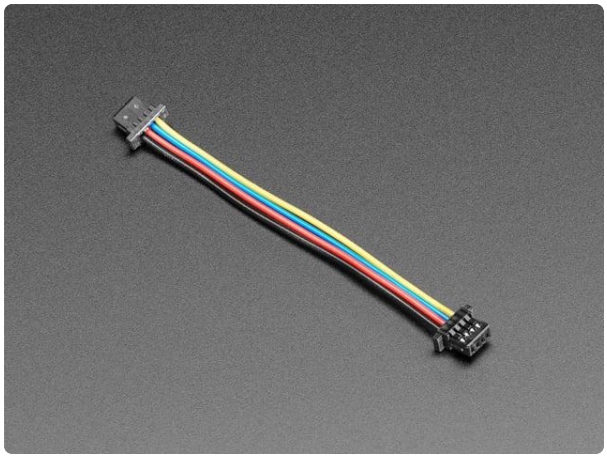
https://www.adafruit.com/product/5397



Adafruit LIS3DH Triple-Axis Accelerometer (+-2g/4g/8g/16g)

The LIS3DH is a very popular low power triple-axis accelerometer. It's low-cost, but has just about every 'extra' you'd want in...

https://www.adafruit.com/product/2809



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

4 x Different colors of 30AWG silicone wires — https://www.adafruit.com/product/3164

The best wires hands down. You'll want at least 4 colors

---

1 x 290mah Lithium-Ion battery from TinyCircuits — https://www.digikey.com/en/products/detail/tinycircuits/ASR00007/7404517

This is the PERFECT size to fit inside this cube!

---

1 x Micro slide switch — https://amzn.to/3em2f0E

A tiny switch for a tiny cube

You'll need approximately two widths: 1/2" and 3/4"

1 x Clear heat shrink tube set https://amzn.to/3RwRIxW

You'll need approximately two widths: 1/2" and 3/4"

---

1 x Kapton tape

A heat-resistant tape to withstand the soldering heat

---

1 x Sandpaper or sanding stick

Any medium-grit will do, something like 100 or 180 grit will do

---

1 x PETG Filament

For printing the frame. A material with similar heat resistance might do.

---

1 x Popsicle stick

(or similar non-conductive pokey thing)

You will also need the following tools:

- Flush cutters
- Hot air gun
- Third hand

3D Printing

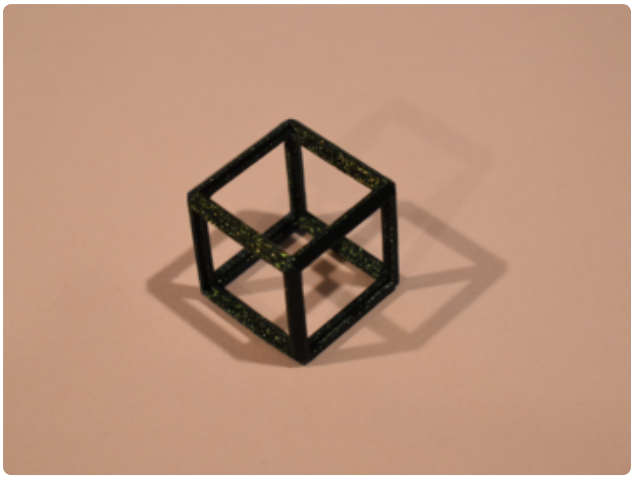
Download the print files using the buttons below. The parameterized model is included in a `.f3d` file in case you want to adjust and customize the model yourself!

Download from Printables

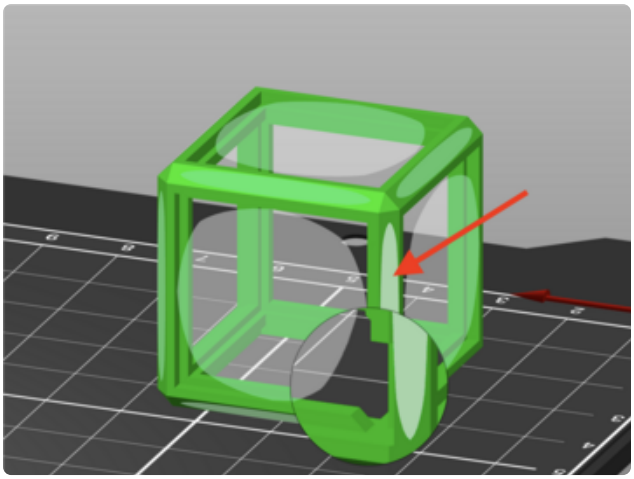
https://adafru.it/10Na

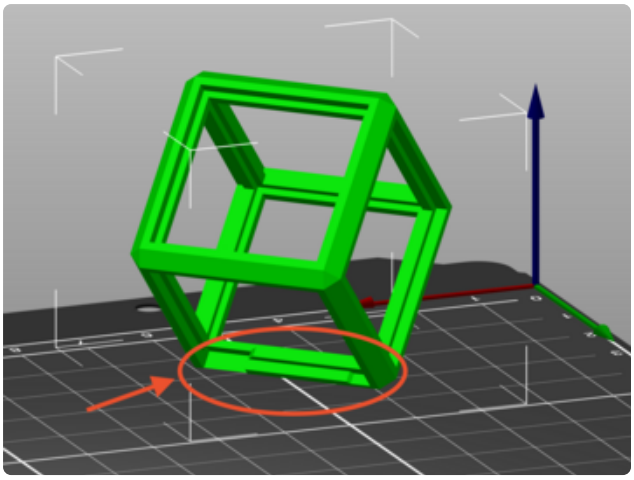
Download as a zip file

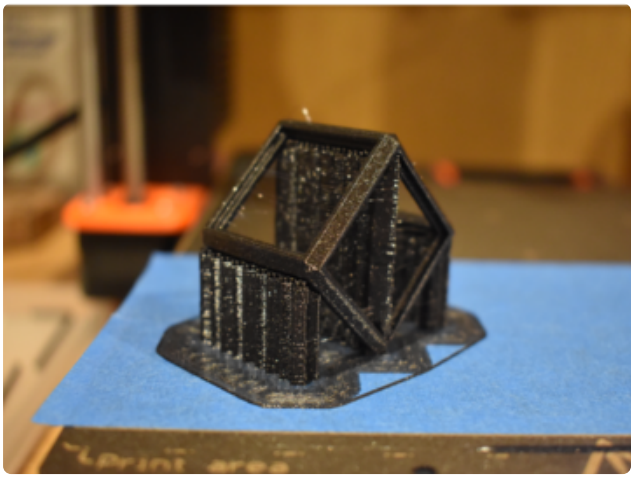
https://adafru.it/10Nb



All six of the LED matrices will be press-fitted onto this 3D printed frame. It is required to print this using PETG filament or a similar material that can withstand more heat than PLA to avoid warping issues, since the matrices themselves tend to become quite warm especially when charging the battery.







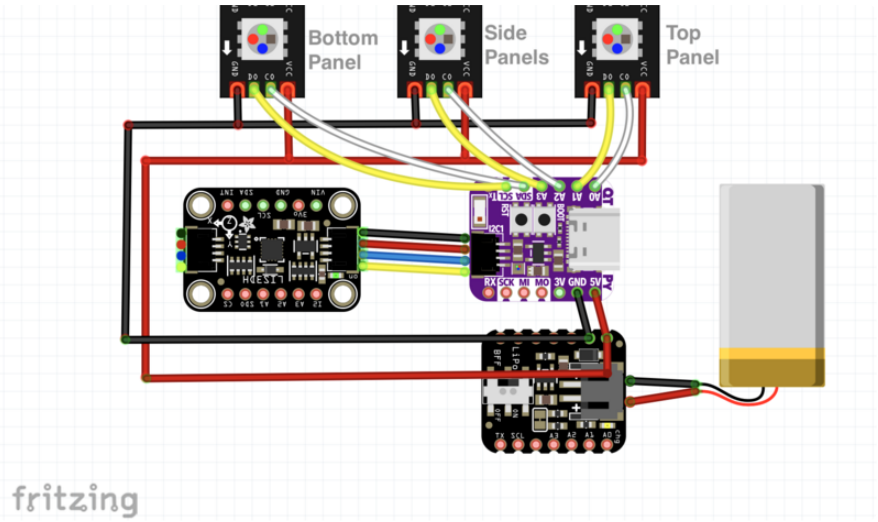
To reduce the amount of support material needed, it's recommended to print this cube in the orientation shown in these photos. There is a cutout on the frame where the switch will go, and that edge of the frame should be face down on the bed.

The last photo illustrates the support material generated for this print using the Prusa Slicer. Here are the most relevant settings used for the Prusament PETG filament (some adjustments might need to be made for your own printer and filament):

Layer height: 0.20mm
Brim width: 5mm
Supports: Everywhere
Fill density: 15%
Fill pattern: Gyroid
Extruder temp: 250C

Circuit Diagram

Here is a circuit diagram that can be used to reference connections while soldering. This is not meant to be an accurate representation of how the wires or placement of the parts will look like -- the components are laid out this way for clarity. It is simply meant to illustrate how to connect each component together.



A couple of notes:

- The LIS3DH accelerometer board will be connected to the QT Py via the STEMMA QT ports.
- The "side panels" will be a series of 4 chained Dotstar matrices. The top and bottom panel will be a single Dotstar matrix.

Here is a tabular version of the connections between the QT Py and the LED panels, in case it's useful:

QT Py	Top panel	Side panels	Bottom panel
5V	5V	5V	5V
GND	GND	GND	GND
A0	CIN	-	-
A1	DIN	-	-
A2	-	CIN	-
A3	-	DIN	-
A4	-	-	CIN
A5	-	-	DIN

Setup AdafruitIO and custom webpage

If you don't yet have an AdafruitIO account, you can use these two buttons to find out how to create an account and how to create a feed:

How to create an AdafruitIO account

https://adafru.it/Ef8

How to create a AdafruitIO Feed

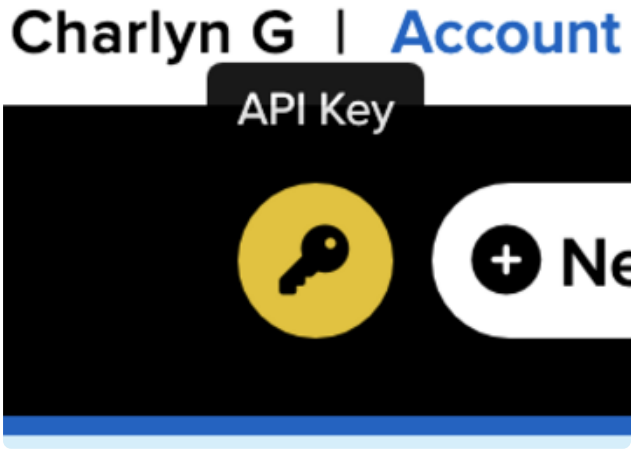
https://adafru.it/f5k

The AdafruitIO Feed you create will be the source of the messages that the cube will fetch. You can put a message into this feed, and the cube will automatically be able to display it after a short delay. There are two Feeds you have to create:

1. `cube-words` - this feed will contain text messages that will be scrolling across the sides fo the cube
2. `cube-pixels` - this feed will contain string messages which will be parsed into an array of pixel coordinates that will be turned on or off. Messages to this feed, if formatted correctly, can show pixel art at the top of the cube with a specified color.

Once you get your feed created (use the guide above for details about how to do that), you can publish some test messages.

Send test messages

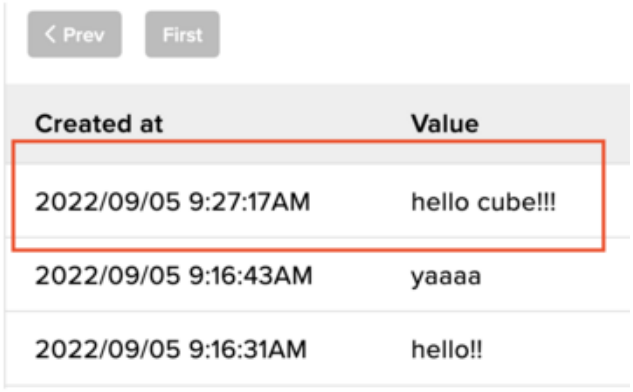


First, grab your AdafruitIO username and key from https://io.adafruit.com/ (https://adafru.it/fsU). Click the yellow button that says "API key" on the navigation bar, which should open a popup that contains these details.

Copy and paste this curl command to a terminal to send a test message to your `cube-words` feed that you just created, making sure to replace the placeholders YOUR_USERNAME and YOUR_KEY with your own details.

```
curl https://io.adafruit.com/api/v2/YOUR_USERNAME/feeds/cube-words/data \
    -H "Content-Type: application/json" \
```

```
    -H "X-AIO-Key: YOUR_KEY" \
    -d '{"value": "hello, cube!!!"}'
```



You can check your feed page to verify that the message did get published into your feed. Go to https://io.adafruit.com/ (https://adafru.it/fsU), click on Feeds and click into `cube-words`. You should see the message "hello cube!!!" published there.

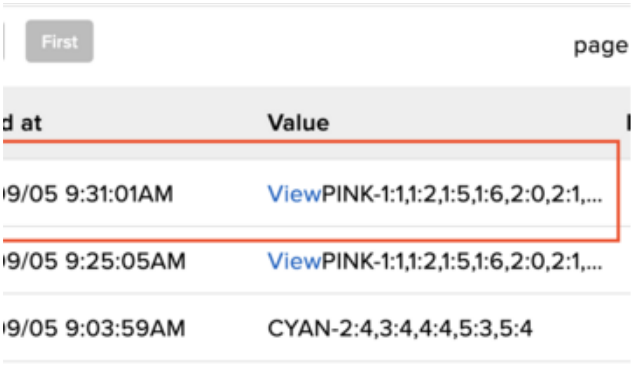
Then, you can send a test message to the `cube-pixels` feed. This message needs to be a specific format for the code to interpret it properly. The format is as follows:

`COLOR-x:y,x:y,x:y...`

Where `COLOR` is an enum that is either `PURPLE, AMBER, JADE, CYAN, BLUE, GOLD`, or `PINK`, and `x:y` is the x and y coordinate of a pixel that needs to be turned on with the specified color.

This is not a standard format, it's something very specific to the cube, so I've provided a sample message here. Remember to replace the `YOUR_` placeholders with your details.

```
curl https://io.adafruit.com/api/v2/YOUR_USERNAME/feeds/cube-pixels/data \
    -H "Content-Type: application/json" \
    -H "X-AIO-Key: YOUR_KEY" \
    -d '{"value":
"PINK-1:1,1:2,1:5,1:6,2:0,2:1,2:2,2:3,2:4,2:5,2:6,2:7,3:0,3:1,3:2,3:3,3:4,3:6,3:7,4:1,4:2,4:3,
```



Again, verify that you've published the message correctly in the `cube-pixels` feed. Go to https://io.adafruit.com/ (https://adafru.it/fsU), click on Feeds and click into `cube-pixels`. You should see the message published there.

When you successfully fetch a message from this feed using your cube, you'll see an adorable pixel heart <3

Custom webpage

To make this message publishing easy and fun, I've created a webpage that will allow you to draw pixel art and select a color, which will publish a well-formatted message into your `cube-pixels` feed. There is also a text box to send a word message to the `cube-words` feed. The details about how exactly I made this page is outside the scope of this tutorial, but you can modify it to publish to your own feed:

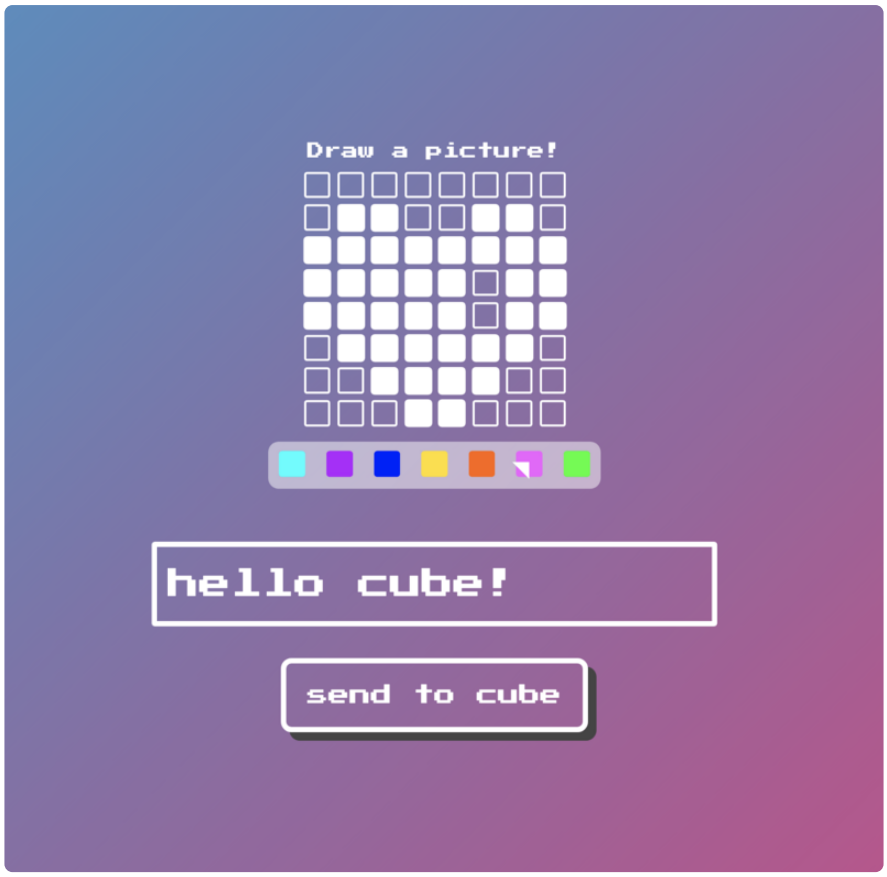
1. At the top of the file **script.js**, you'll find three variables, and each of them have placeholders that start with `YOUR_`. Replace these placeholders with your own AdafruitIO username and key. Save the file.
2. Open the file **index.html** in a modern browser, preferably Chrome.

You can now draw something and it will send a properly formatted string to your `cube-pixels` feed, and will send anything you type in the text box to the `cube-words` feed. Download the zip file below to start modifying and playing with these files!

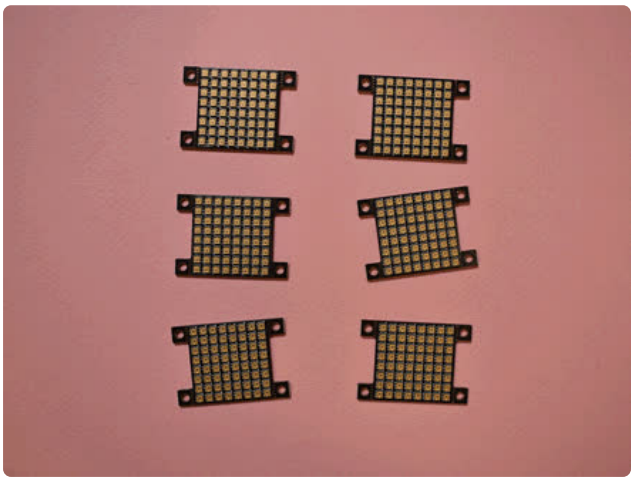
cube-webpage.zip

https://adafru.it/10Nc

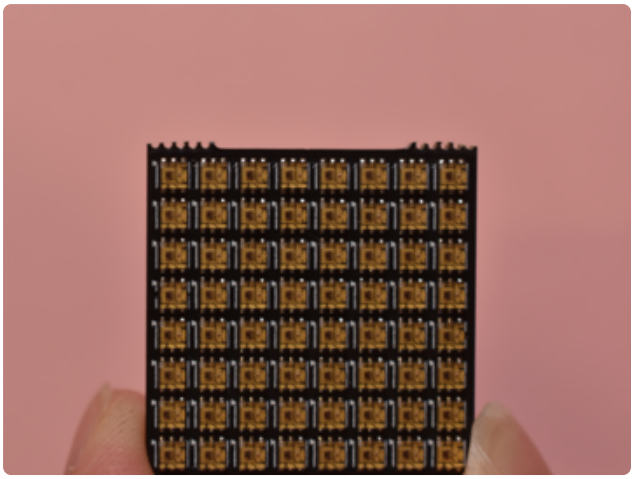
At the end of this tutorial, you should be able to see the latest messages you've sent to your feeds glowingly displayed on your cube!



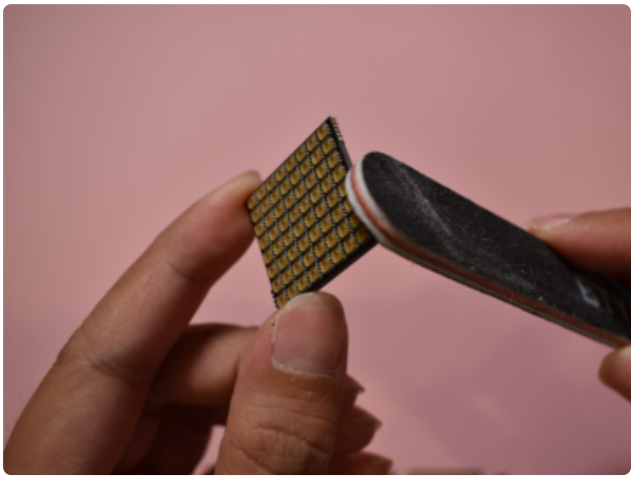
Prepare parts and initial soldering



Use a flush cutter to carefully cut the tabs off the Dotstar matrices. This doesn't have to be a clean cut -- you should intentionally leave a lot of material on the grid so that you can sand it to press-fit into the frame

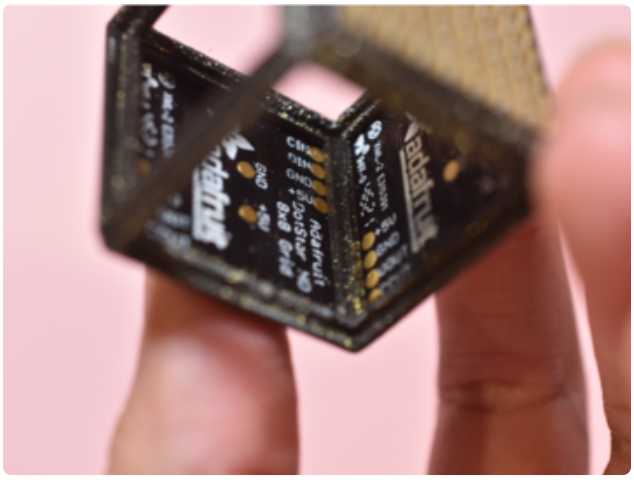


This is approximately how much material should be left from the tabs on the edge of the grid before sanding



Carefully use some sand paper or sanding stick to smooth out these rough edges, but continue to check if the grid will press-fit into the 3d printed frame. You want to keep checking the fit as you sand to avoid over-sanding.

It bears repeating: it is BETTER to sand TOO LITTLE, than too much.



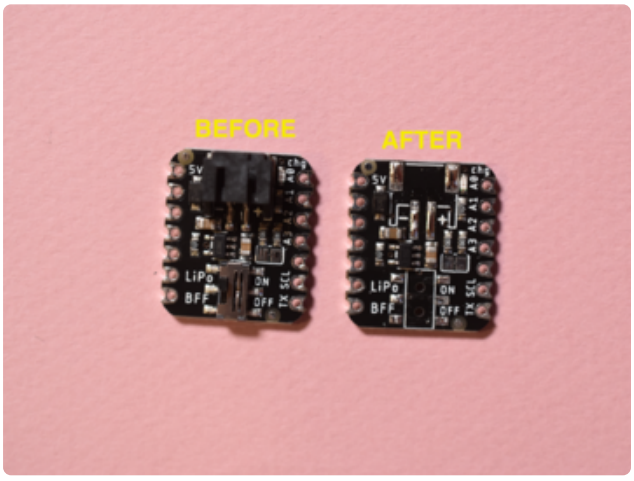
Take your time with this step -- the frame will very slightly expand as the LEDs get warmer, so you want to err towards the grid needing a little bit more force to press fit in.

You'll want to make sure that the switch cutout is oriented at the "top" position, and then orient each piece as shown using the back of the grid as a guide (this will result in the cut tabs being on the north and south of each grid)

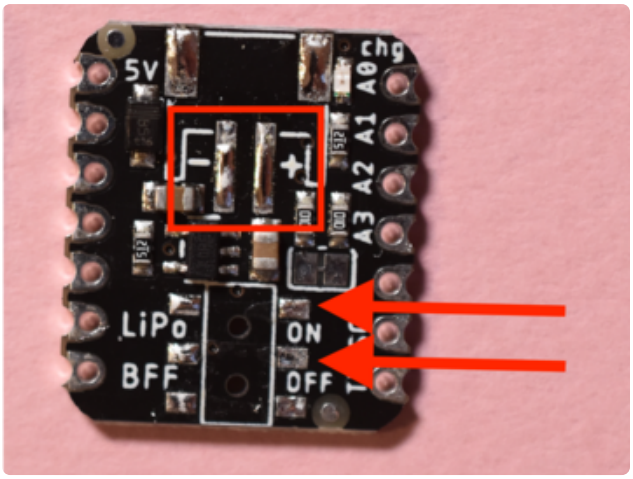
Again, don't sand too much. When in doubt, just move on to the next steps, you can always sand some more in the final assembly steps.

Prepare the power circuit

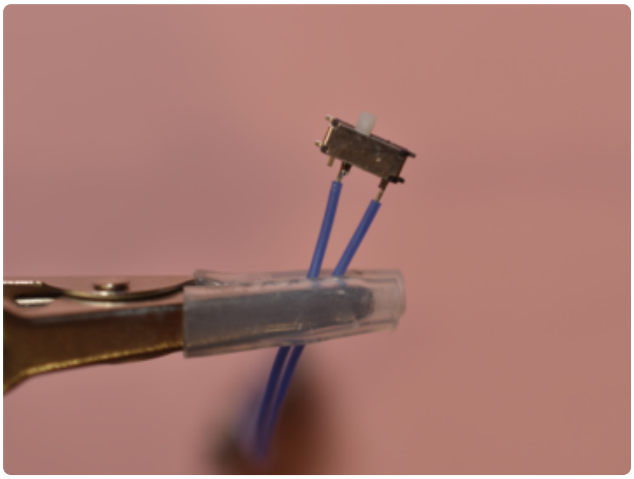
We'll have to deconstruct the QT Py charger add-on so that everything can fit inside the cube.



Desolder the JST connector AND the power switch on the QT Py charger add-on, making sure that the power pads are left intact. The photo here shows what the charger add-on will look like before and after desoldering.

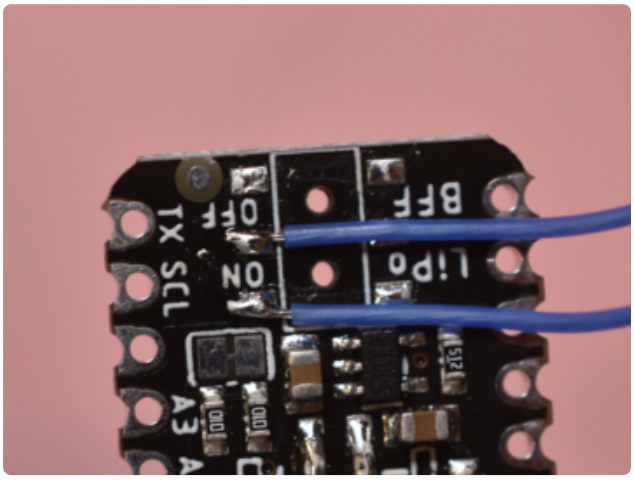


The red arrows are the two pads that we'll need to connect to turn on the circuit, and the pads marked (-) and (+) denoted by the red square are the pads we'll need to connect the battery wires directly to.



Solder two wires to the tiny power switch, one onto the middle pole and another on either side poles.

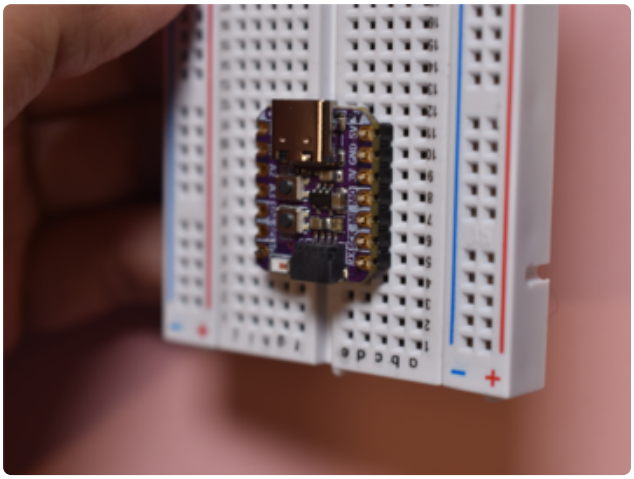
It will help if you tin both the switch poles and the wires with some solder before you solder them together.

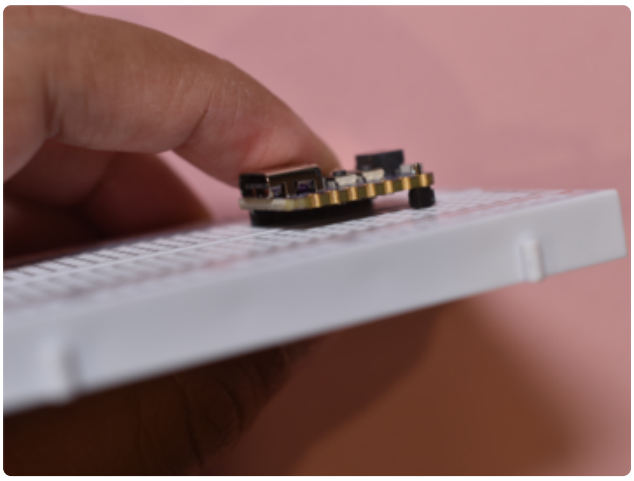


Solder each wire from the switch onto the on/off pads on the charger add-on. This switch will now act as the power switch for the entire circuit.

Now that you have a power circuit without a battery attached, we can move on to connecting the charger add-on to the QT Py itself.

Prepare the QT Py

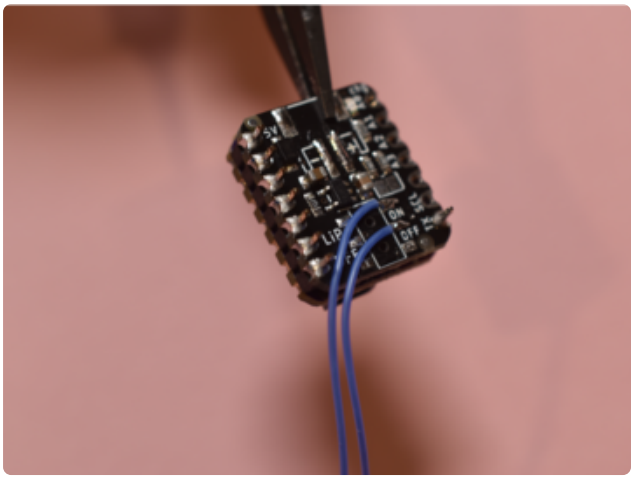


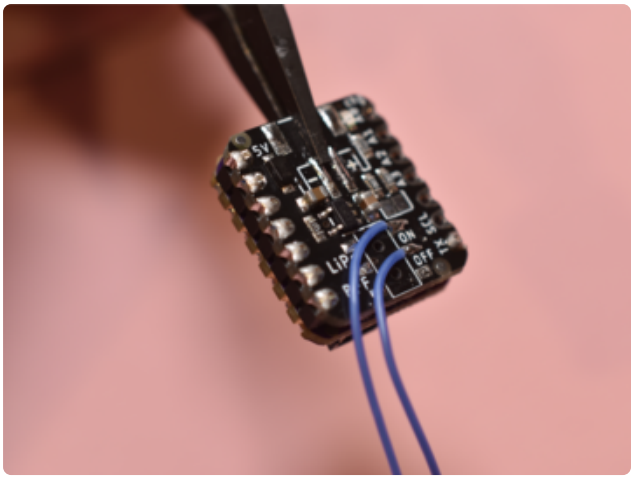


Insert 7 pin headers into some breadboard, and align the right side of the QT Py onto these headers (the side with the 5V and GND pins). Then, insert a single pin header into the breadboard on the other side of the QT Py, and align the TX pin of the QT Py onto this single pin. Refer to the photos to see this a bit more clearly. There should be seven pins on one side, and one pin on the other side.

Solder these headers onto the QT Py!

Note that you could also choose to fully solder 7 pins on each side of the QT Py, it might just be a bit more challenging to solder the wires connecting to the LED grid later on.



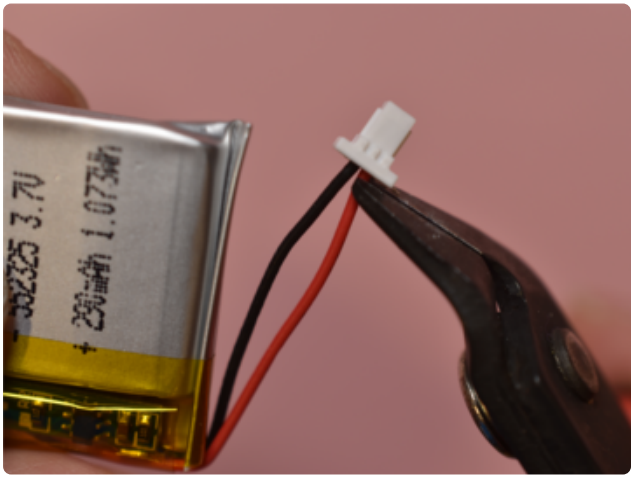


Now that you have all the headers soldered onto the QT Py, you can now align the BFF charger add-on to the back of the QT Py. Ensure that the 5V pin is aligned with the 5V pin of the QT Py on the other side.

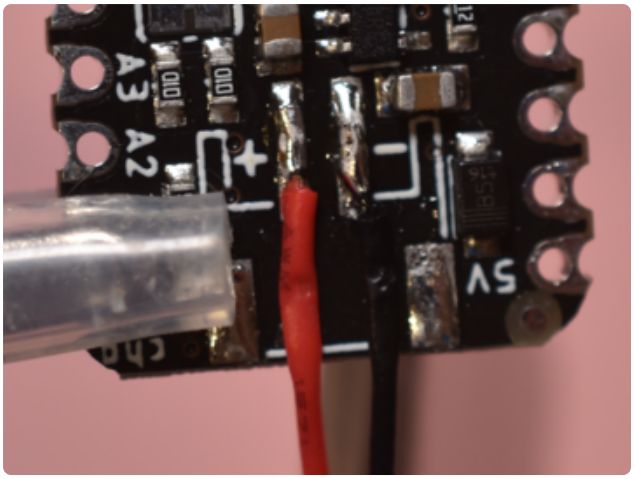
Cut off all excess headers as short as you can with a flush cutter.

Solder the battery wires

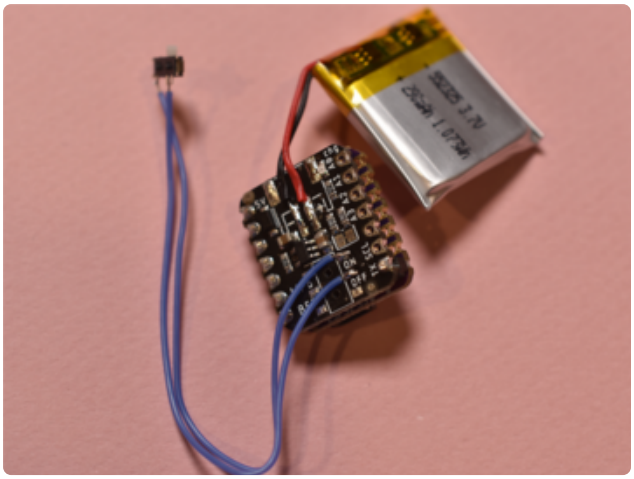
Warning: the next steps will require you to cut the wires directly connected to a battery. DO NOT cut both wires at the same time. Doing so might cause a short or a spark. Cut each wire one at a time, and make sure to keep these wires apart when the conductive material is exposed.

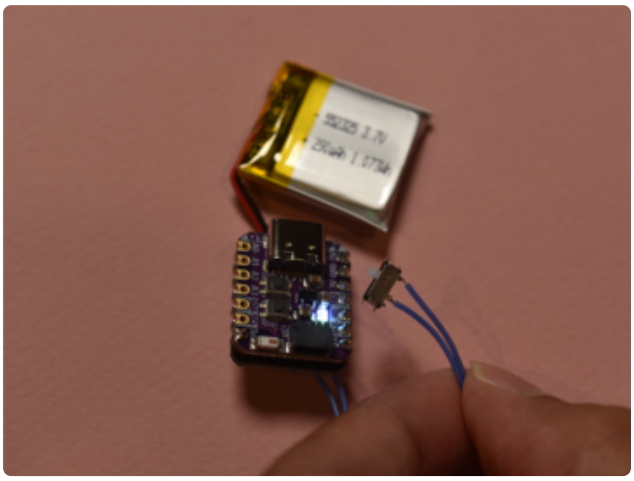


Using your trusty flush cutters, cut off **one** wire as close to the connector as you can to get the max amount of wire length. Then, cut the other wire off. Do NOT cut both wires at the same time.



Use a wire stripper to expose the wires, and use some solder to tin each wire. Then, with the help of a third hand, carefully solder the red wire to the positive (+) pad and the black wire to the negative (-) pad.





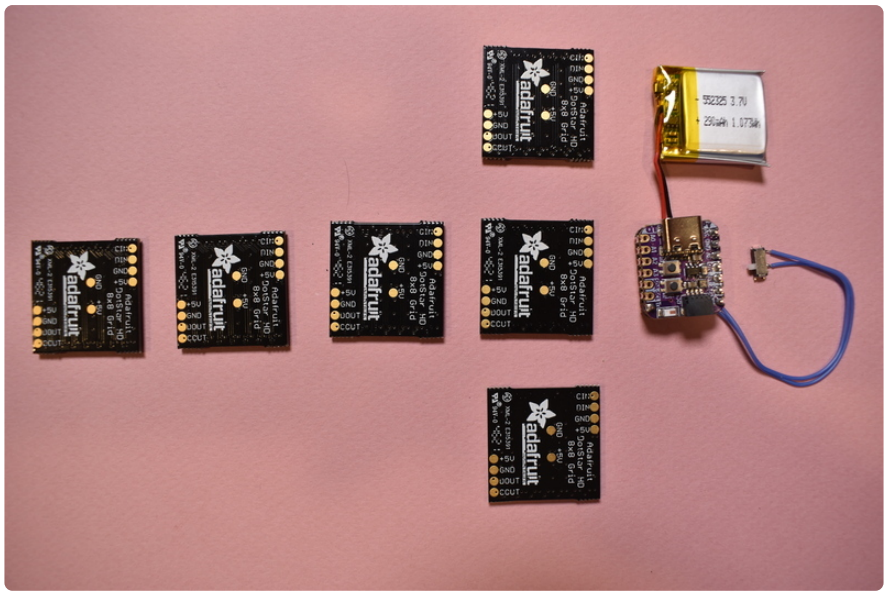
Now, you should have an assembly of the QT Py, the charger add on, the battery and the switch which should look like the photo here.

Try to turn on the switch. You should see the LED on the QT Py light up if you haven't modified the factory code onboard.

Great job! Admire your work for a second before moving on

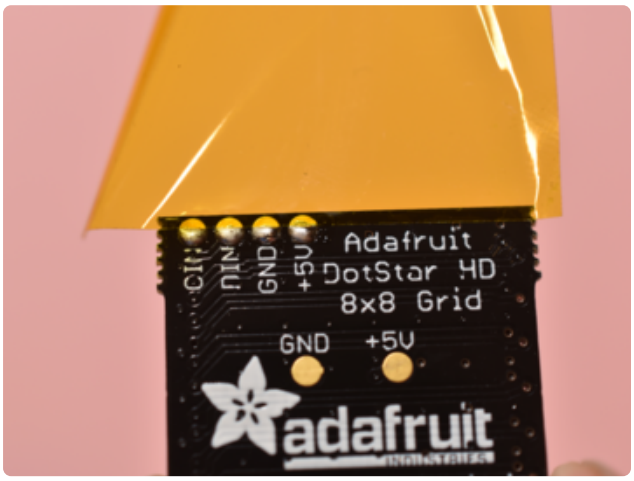
Solder and assemble LEDs

At this point, you'll have 6 LED matrices that will press fit onto the frame, and an assembled QT Py + power circuit, like shown below:

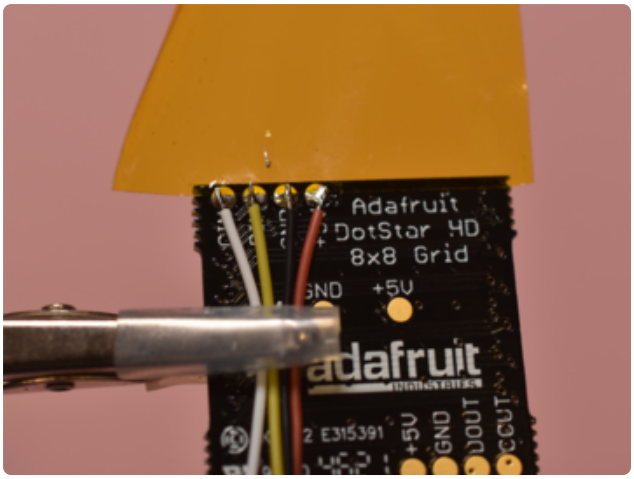


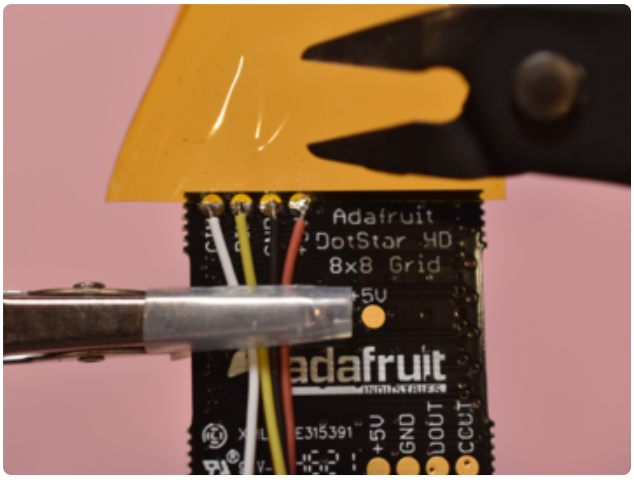
The plan is to connect 4 matrices together to form one long matrix display that will wrap around all 4 sides of the cube. Then, we'll have one matrix be the "top" matrix and another one will be the "bottom" matrix, each separately connected to the QT Py as separate displays.

Use some kapton tape to cover about 40% of the circle pads of the matrix. This will ensure that we can press-fit the matrices flush into the 3d printed frame.



Tin these pads with some solder, and verify that the area under the kapton tape did not get solder on it. A tiny bit is ok, as long as the tape is still mostly flat.





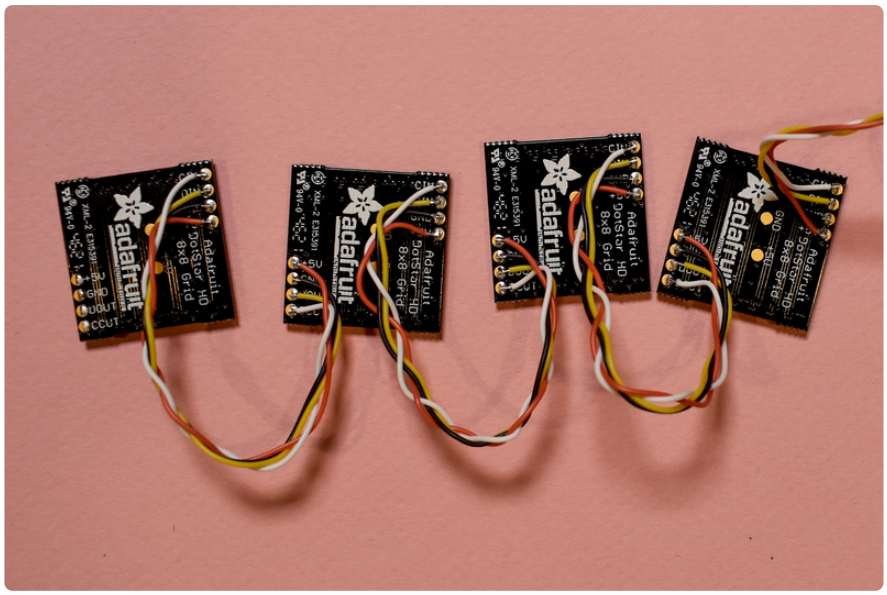
Cut about 10cm of 4 different colored wires, and solder a different color onto each pad. Stick to these colors to help you chain the 4 matrices properly. These are the colors used here:

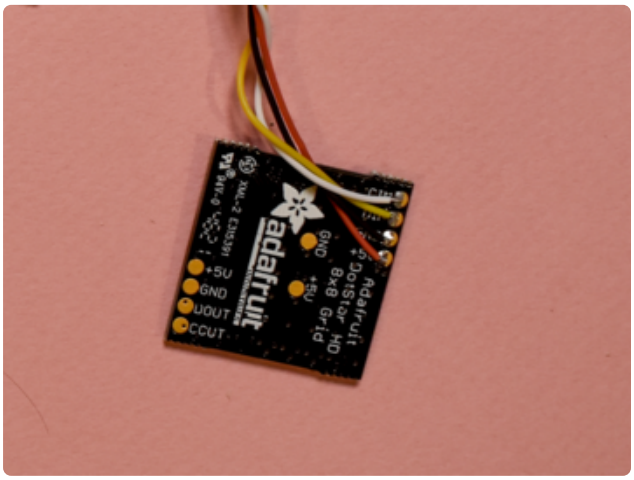
White for CIN/COUT
Yellow for DIN/DOUT
Black for GND
Red for 5V
Use the flush cutters to cut off any excess wires that might be over the taped area. Again, doing this will ensure that the matrix will sit flush in the 3D printed frame.

Do the above steps for all 4 matrices, and make sure to connect the matrices to each other via the correct pad connections as follows:

- CIN to COUT
- DIN to DOUT
- GND to GND
- 5V to 5V

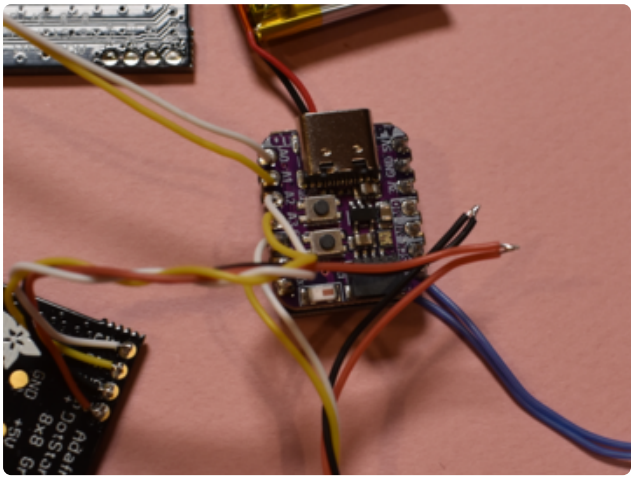
You can optionally choose to braid the wires together for a cleaner look. When you've chained 4 matrices together, it should look something like this:





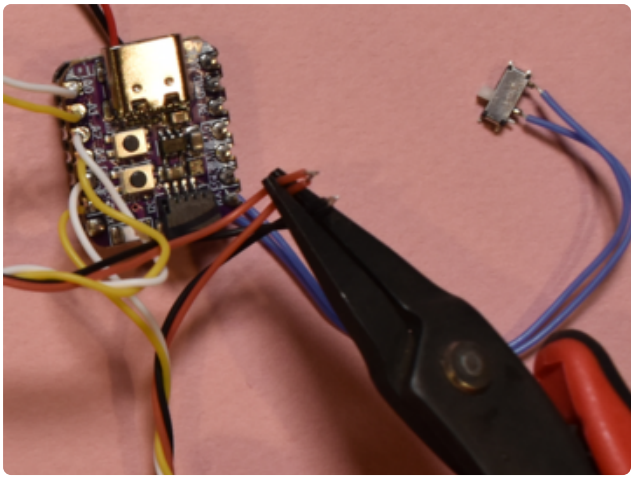
Take one more matrix, and solder the same colors of wires to each pad, using the same kapton tape technique discussed above.

This will be the bottom matrix.

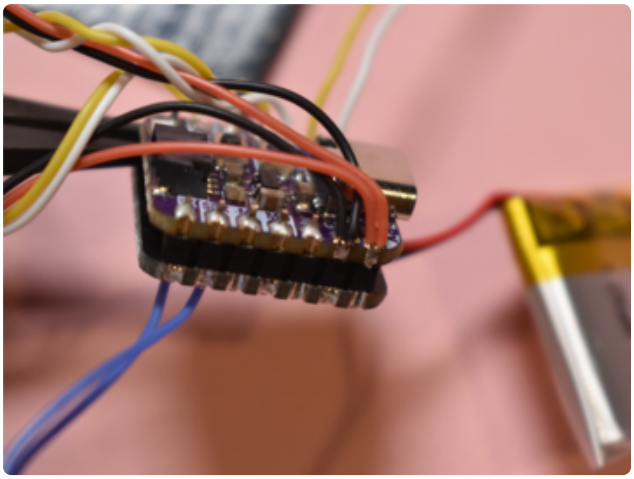


Now it's time to connect the CIN and DIN wires of the "side panels" and the bottom matrix into the QT Py. Solder the wires as follows:

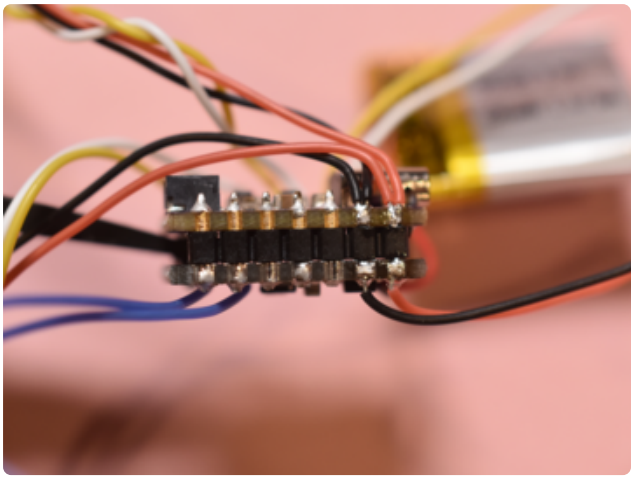
Side panels CIN (white) to A2
Side panels DIN (yellow) to A3
Bottom matrix CIN (white) to A4
Bottom matrix DIN (yellow) to A5
Afterwards, connect about 10 cm of white wire to A0 and connect the same length of yellow wire to A1. Do not connect these wires to anything yet, these will connect to the "top" matrix later.



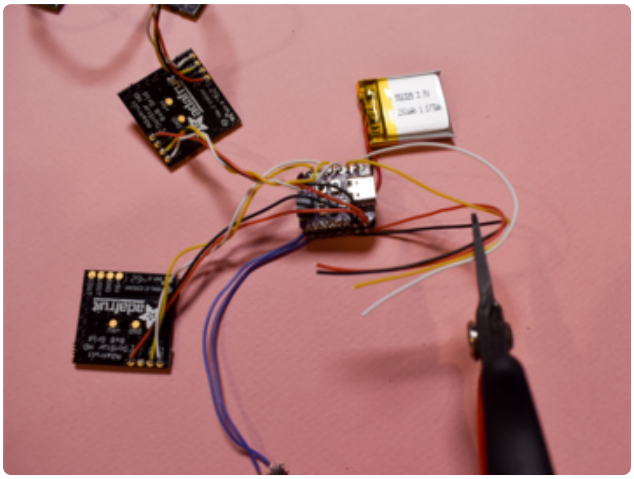
Combine both the GND (black) and 5V (red) wires for the side panels and the bottom matrix as shown, we'll solder these next.



The QT Py has castellated edges, meaning that it's actually possible to solder on the edge of the board. We'll take advantage of this -- solder the combined red 5V wires of both the side panels and bottom matrix to the 5V edge of the pin, and then do the same for the black GND wires.

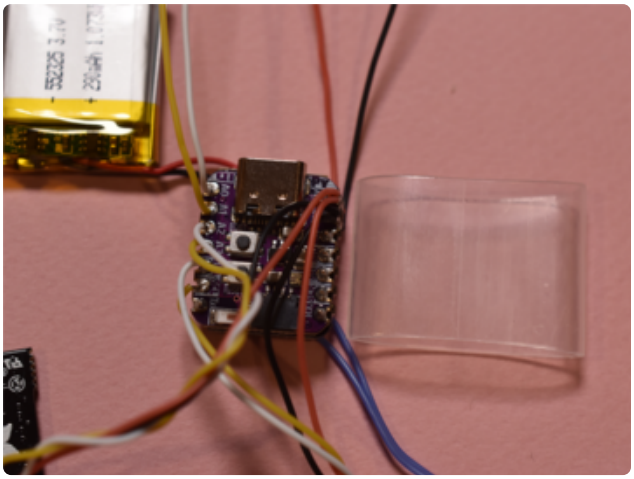


And then cut separate pieces of 10 cm each of red and black wires, and solder to the edge of the castellated pads on the charger add-on. These two wires will be connected to the "top" matrix later on.

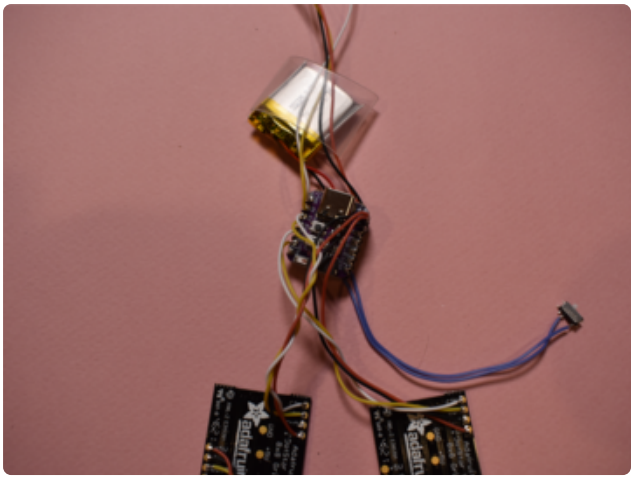


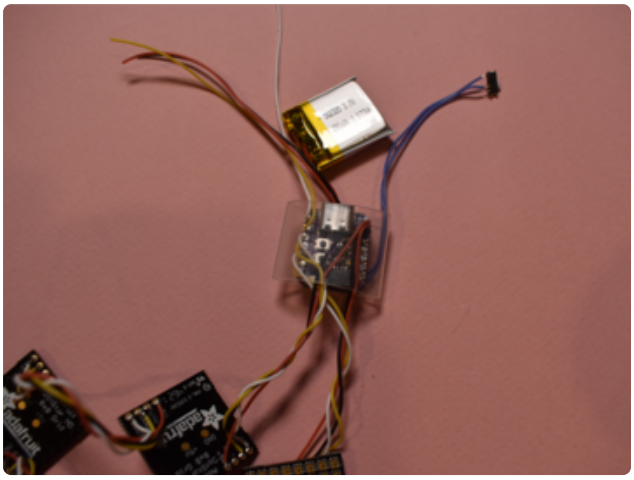
Now, you'll have all 4 side panels and the bottom matrix connected to the QT Py. You'll also have these 4 wires sticking out of the QT Py which will connect to the top matrix later.

Before we can connect these wires to the top matrix, we'll need to first put some heat shrink around the QT Py to protect it while inside the cube

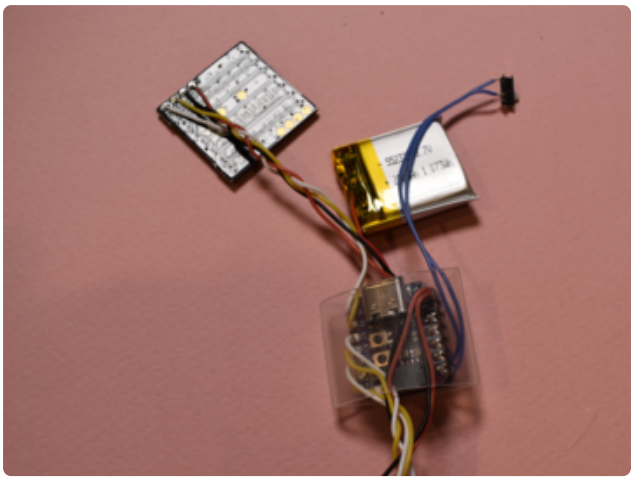


Measure some heat shrink (about 3/4" width) to roughly match the height of the QT Py. When heat is applied, the width will shrink but the height will barely change.

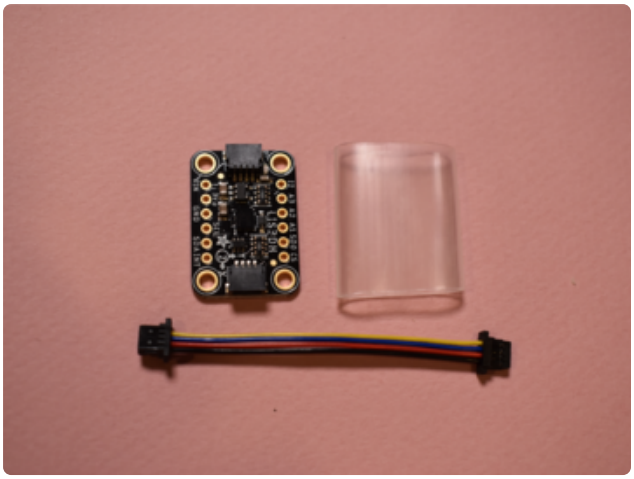


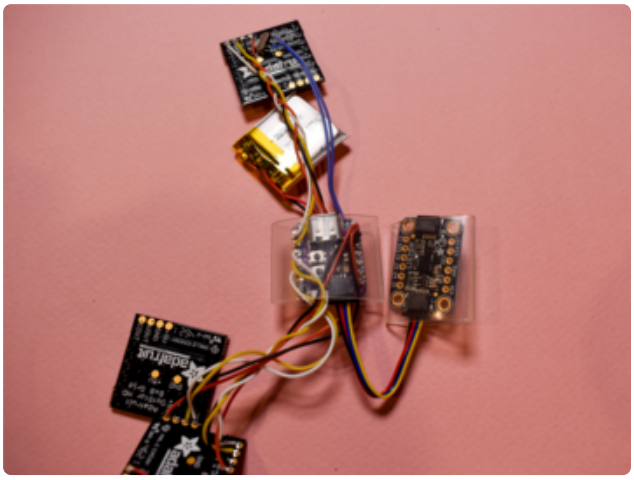


Thread the **loose wires**, the **battery**, and the **switch** through the heat shrink, and then position the heat shrink around the QT Py.



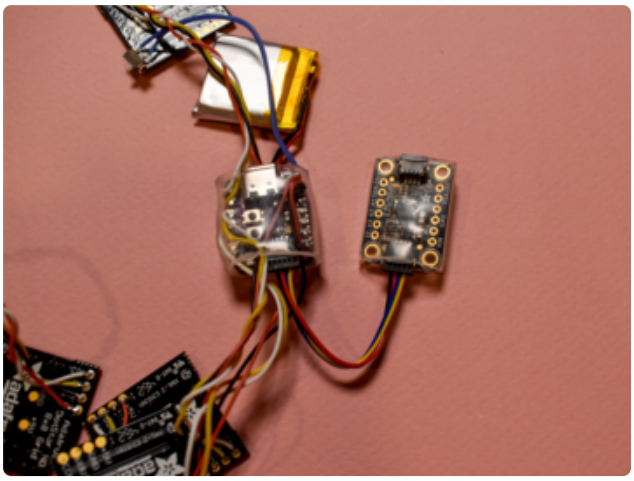
At this point, you can now attach the top matrix to the loose wires, using the same kapton tape technique we used for the other matrices.





Then, take the LIS3DH accelerometer board and measure some heat shrink (about 1/4" width) to match the height of the board.

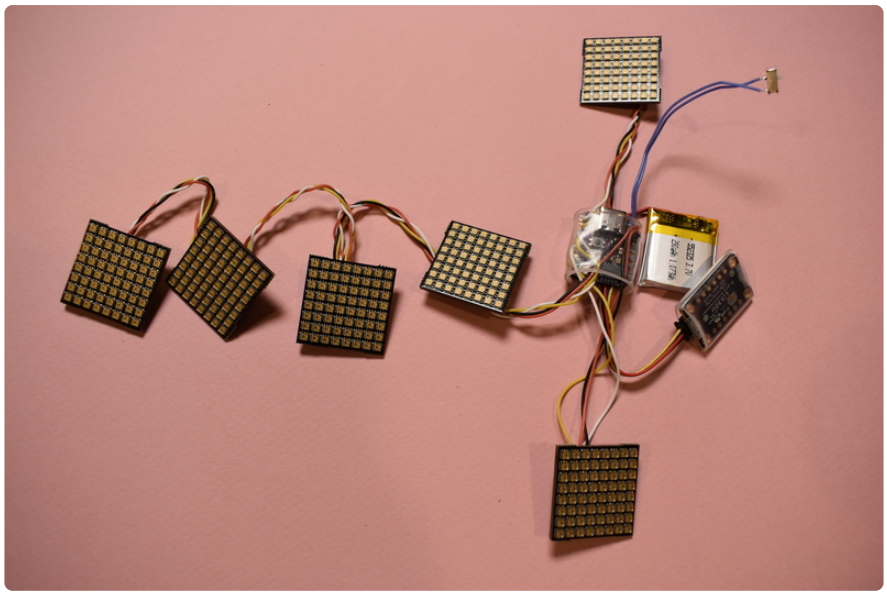
Attach the STEMMA QT wires between the QT Py and the sensor board.



Arrange the wires on the QT Py such that they are not on top of the two onboard buttons. Use a hot air gun set to about 110C and carefully apply heat to shrink the heat shrink wrap.

Now your electronics are well protected and ready to be stuffed into a tiny cube!

At this point, you'll be ready to upload some code onto the QT Py to test that everything is connected properly. The entire assembly should look similar to this:



CircuitPython Setup

This cube is powered by CircuitPython! If you're new to CircuitPython, follow this quick guide to get your board up and running:

CircuitPython on ESP32-S3

https://adafru.it/10Nd

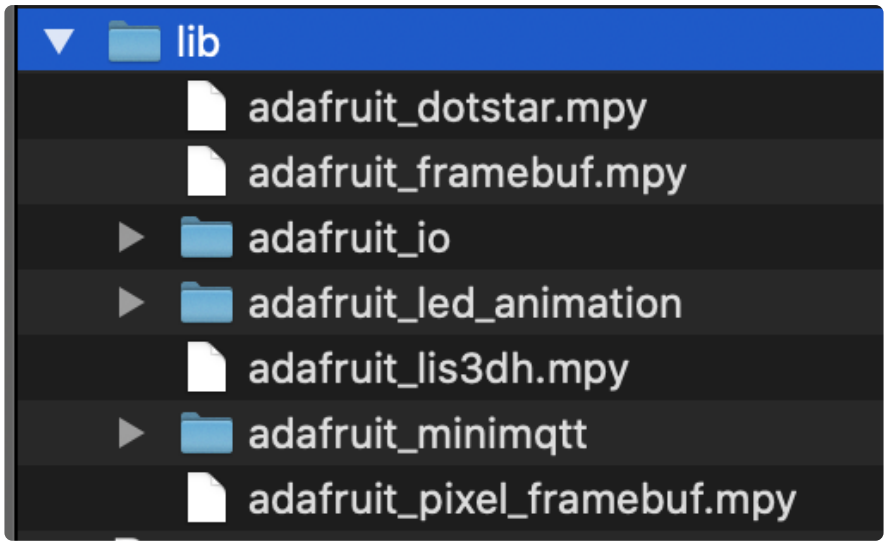
Once you've got the board setup with CircuitPython, you'll need a few libraries to get the code to work.

When on the next page, click the "Download Project Bundle" button in the code window to get the code and latest library files.

Copy over the following files and folders into your **lib** folder in the **CIRCUITPY** drive:

- **adafruit_dotstar**
- **adafruit_framebuf**
- **adafruit_io**
- **adafruit_led_animation**
- **adafruit_lis3dh**
- **adafruit_minimqtt**
- **adafruit_pixel_framebuf**

Your **lib** folder should look like this:



Additional files

You'll also need two more files: the font file to display text and a **secrets.py** file to store your WiFi SSID and password plus your AdafruitIO API key.

Click the button below to download the font file, and unzip it into your **CIRCUITPY** drive:

font5x8.bin.zip

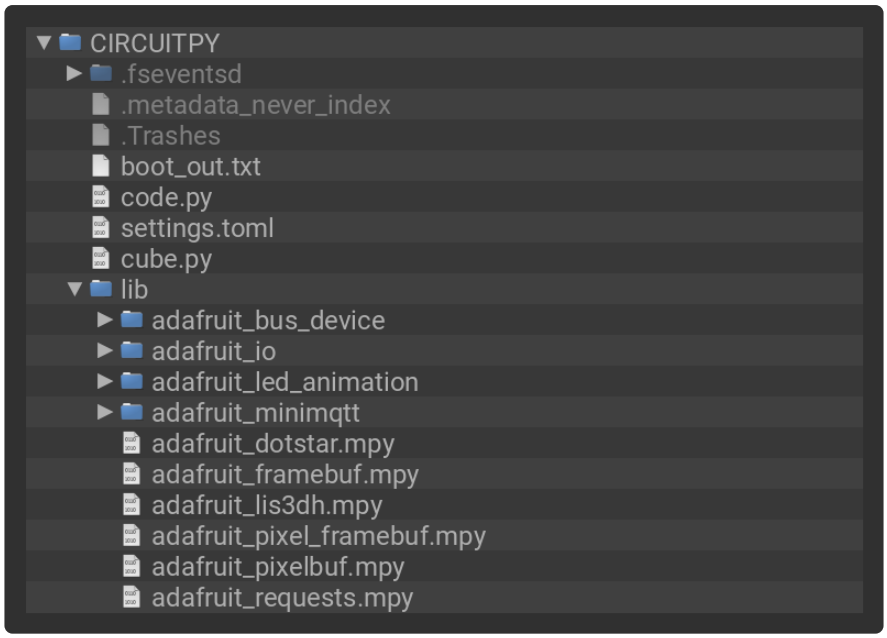
https://adafru.it/10Ne

Finally, create a new file called **secrets.py** following the template below (replace the `YOUR_` variables with your own details) and save it in the **CIRCUITPY** root folder as well:

```
secrets = {
'ssid': 'YOUR_WIFI_SSID',
'password': 'YOUR_WIFI_PASSWORD',
'aio_username': 'YOUR_USERNAME',
'aio_key': 'YOUR_KEY',
}
```

CircuitPython Code

Now we're ready to tackle some code! There are two files that power this cube, and we can briefly talk through each.



Code.py

code.py is where the main loop lives. This is where we handle the accelerometer data and the network requests to fetch new data from AdafruitIO. This file is where we initialize a Cube class that will take care of the bulk of the logic for displaying stuff on the cube. In addition, there are two main functions defined and used in this file.

1. `update_data()` will make a call to AdafruitIO every time the scrolling word has finished one loop. This cadence was chosen to keep the scrolling animation relatively smooth and to reduce the amount of network calls. This function will update some global variables that are then passed into the `cube.update()` function
2. `orientation()` will do some basic math and logic to detect the orientation of the cube in space. The resulting orientation is used to determine which `Cube` function to activate.

```
# SPDX-FileCopyrightText: 2022 Charlyn Gonda for Adafruit Industries
#
# SPDX-License-Identifier: MIT
from secrets import secrets
import ssl
import busio
import board
import adafruit_lis3dh
import wifi
import socketpool
import adafruit_requests

from adafruit_led_animation.color import (
    PURPLE, AMBER, JADE, CYAN, BLUE, GOLD, PINK)
from adafruit_io.adafruit_io import IO_HTTP

from cube import Cube
```

```
# Specify pins
top_cin = board.A0
top_din = board.A1
side_panels_cin = board.A2
side_panels_din = board.A3
bottom_cin = board.A4
bottom_din = board.A5

# Initialize cube with pins
cube = Cube(top_cin,
            top_din,
            side_panels_cin,
            side_panels_din,
            bottom_cin,
            bottom_din)

# Initial display to indicate the cube is on
cube.waiting_mode()

# Setup for Accelerometer
i2c = busio.I2C(board.SCL1, board.SDA1)
lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c)

connected = False
while not connected:
    try:
        wifi.radio.connect(secrets["ssid"], secrets["password"])
        print("Connected to %s!" % secrets["ssid"])
        print("My IP address is", wifi.radio.ipv4_address)
        connected = True
    # pylint: disable=broad-except
    except Exception as error:
        print(error)
        connected = False


# Setup for http requests
pool = socketpool.SocketPool(wifi.radio)
REQUESTS = adafruit_requests.Session(pool, ssl.create_default_context())
IO = IO_HTTP(secrets["aio_username"], secrets["aio_key"], REQUESTS)

# Data for top pixels, will be updated by update_data()
TOP_PIXELS_ON = []
TOP_PIXELS_COLOR = CYAN
TOP_PIXELS_COLOR_MAP = {
    "PURPLE": PURPLE,
    "AMBER": AMBER,
    "JADE": JADE,
    "CYAN": CYAN,
    "BLUE": BLUE,
    "GOLD": GOLD,
    "PINK": PINK,
}
# Data for scrolling word, will be updated by update_data()
CUBE_WORD = "... ..."


def update_data():
    # pylint: disable=global-statement
    global CUBE_WORD, TOP_PIXELS_ON, TOP_PIXELS_COLOR
    if connected:
        print("Updating data from Adafruit IO")
        try:
            quote_feed = IO.get_feed('cube-words')
            quotes_data = IO.receive_data(quote_feed["key"])
            CUBE_WORD = quotes_data["value"]

            pixel_feed = IO.get_feed('cube-pixels')
            pixel_data = IO.receive_data(pixel_feed["key"])
```

```
            color, pixels_list = pixel_data["value"].split("-")
            TOP_PIXELS_ON = pixels_list.split(",")
            TOP_PIXELS_COLOR = TOP_PIXELS_COLOR_MAP[color]
        # pylint: disable=broad-except
        except Exception as update_error:
            print(update_error)


orientations = [
    "UP",
    "DOWN",
    "RIGHT",
    "LEFT",
    "FRONT",
    "BACK"
]

# pylint: disable=inconsistent-return-statements


def orientation(curr_x, curr_y, curr_z):
    absX = abs(curr_x)
    absY = abs(curr_y)
    absZ = abs(curr_z)

    if absX > absY and absX > absZ:
        if x >= 0:
            return orientations[1]  # up

        return orientations[0]  # down

    if absZ > absY and absZ > absX:  # when "down" is "up"
        if z >= 0:
            return orientations[2]  # left

        return orientations[3]  # right

    if absY > absX and absY > absZ:
        if y >= 0:
            return orientations[4]  # front

        return orientations[5]  # back


upside_down = False
while True:
    x, y, z = lis3dh.acceleration
    oriented = orientation(x, y, z)

    # clear cube when on one side
    # this orientation can be used while charging
    if oriented == orientations[5]:  # "back" side
        cube.clear_cube(True)
        continue

    if oriented == orientations[1]:
        upside_down = True
    else:
        upside_down = False

    if not upside_down:
        if cube.done_scrolling:
            update_data()

        cube.update(CUBE_WORD, TOP_PIXELS_COLOR, TOP_PIXELS_ON)
        cube.scroll_word_and_update_top()

    else:
        cube.upside_down_mode()
```

Cube.py

cube.py contains a class called `Cube` that is responsible for all the cube display logic and keeping track of the cube's various states. It has 5 main functions that are used inside **code.py**:

1. `update()` is a convenience function to update both the word that scrolls through the cube and what pixels should be turned "on" for the top matrix, along what color those top pixels should be
2. `scroll_word_and_update_top()` will continuously scroll through the given word and will show the pixel art on the top of the cube
3. `clear_cube()` can clear the sides of the cube, with the ability to clear the top of the cube if you set `clearTop=True`
4. `upside_down_mode()` will display a specific cube animation, and this is triggered when the cube is upside-down
5. `waiting_mode()` just shows two pixels lit up on the top matrix, just to indicate that the cube is on when it first boots up.

When you assemble your cube, and you find that the word scroll orientation is not as you expected, you can try to flip the initialization of `self.pixel_framebuf_sides` inside the `__init__` function of `Cube` to `reverse_y=False, reverse_x=True` to see if that will yield a better orientation.

Otherwise, there should be little to no modification needed for this code, but definitely feel free to modify!

```
# SPDX-FileCopyrightText: 2022 Charlyn Gonda for Adafruit Industries
#
# SPDX-License-Identifier: MIT
import time
import adafruit_dotstar

from adafruit_led_animation.animation.rainbowchase import RainbowChase
from adafruit_led_animation.color import AMBER, JADE, CYAN, GOLD, PINK
from adafruit_pixel_framebuf import PixelFramebuffer


class Cube():
    def __init__(
            self,
            top_cin,
            top_din,
            side_panels_cin,
            side_panels_din,
            bottom_cin,
            bottom_din):

        # static numbers
        self.num_pixels = 64*4
        self.num_pixels_topbottom = 64
        self.pixel_width = 8*4
        self.pixel_height = 8
```

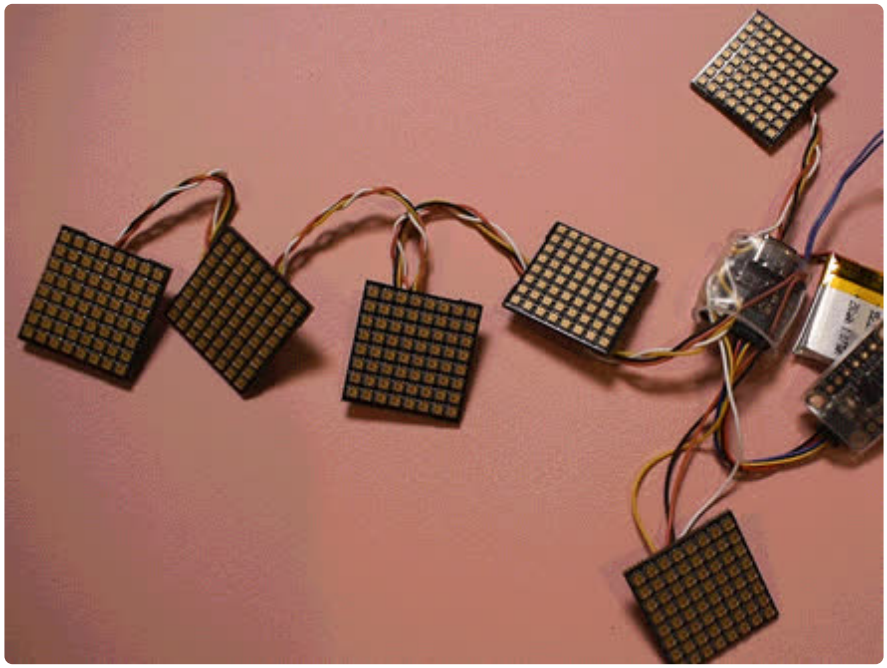
```
        # top pixels
        top_pixels = adafruit_dotstar.DotStar(
            top_cin, top_din, self.num_pixels_topbottom,
            brightness=0.03, auto_write=False)
        self.pixel_framebuf_top = PixelFramebuffer(
            top_pixels,
            self.pixel_height,
            self.pixel_height,
            rotation=1,
            alternating=False,
        )

        # side pixels
        self.side_pixels = adafruit_dotstar.DotStar(
            side_panels_cin, side_panels_din, self.num_pixels,
            brightness=0.03, auto_write=False)
        self.pixel_framebuf_sides = PixelFramebuffer(
            self.side_pixels,
            self.pixel_height,
            self.pixel_width,
            rotation=1,
            alternating=False,
            reverse_y=True,
            reverse_x=False
        )
        self.rainbow_sides = RainbowChase(
            self.side_pixels, speed=0.1, size=3, spacing=6)

        # bottom pixels
        pixels_bottom = adafruit_dotstar.DotStar(
            bottom_cin, bottom_din, self.num_pixels_topbottom,
            brightness=0.03, auto_write=False)
        self.pixel_framebuf_bottom = PixelFramebuffer(
            pixels_bottom,
            self.pixel_height,
            self.pixel_height,
            rotation=0,
            alternating=False,
            reverse_y=False,
            reverse_x=True
        )

        # scrolling word state vars
        self.last_color_time = -1
        self.color_wait = 1
        self.word = ''
        self.total_scroll_len = 0
        self.scroll_x_pos = -self.pixel_width
        self.color_idx = 0
        self.color_list = [AMBER, JADE, CYAN, PINK, GOLD]
        self.done_scrolling = False

        # whether or not the cube is already clear
        self.clear = False

        # top pixel vars
        self.top_pixel_coords = []
        self.top_pixel_color = CYAN

        # upside down mode
        self.bottom_last = -1
        self.bottom_wait = 1
        self.bottom_squares = [[0, 0, 8, 8], [
            1, 1, 6, 6], [2, 2, 4, 4], [3, 3, 2, 2]]
        self.bottom_squares_idx = 0

    def update(self, word, color, coords):
        self.word = word
```

```
        self.total_scroll_len = (len(self.word) * 5) + len(self.word)
        self.top_pixel_coords = coords
        self.top_pixel_color = color

    def scroll_word_and_update_top(self):
        if self.scroll_x_pos >= self.total_scroll_len:
            self.scroll_x_pos = -self.pixel_width
            self.clear_cube()
            self.done_scrolling = True
        else:
            self.done_scrolling = False
            self.clear = False

        self.__scroll_framebuf(self.word, self.scroll_x_pos, 0)
        self.__display_top_pixels()
        self.scroll_x_pos = self.scroll_x_pos + 1

    def clear_cube(self, clear_top=False):
        if not self.clear:
            self.pixel_framebuf_sides.fill(0)
            self.pixel_framebuf_sides.display()
            self.side_pixels.fill(0)
            self.side_pixels.show()
            self.pixel_framebuf_bottom.fill(0)
            self.pixel_framebuf_bottom.display()
            if clear_top:
                self.pixel_framebuf_top.fill(0)
                self.pixel_framebuf_top.display()
            self.clear = True

    def upside_down_mode(self):
        self.clear_cube(True)
        self.rainbow_sides.animate()
        now = time.monotonic()
        self.__bottom_square_animation(now)

    def waiting_mode(self):
        self.pixel_framebuf_top.pixel(3, 3, CYAN)
        self.pixel_framebuf_top.pixel(4, 4, PINK)
        self.pixel_framebuf_top.display()

    def __bottom_square_animation(self, now):
        self.pixel_framebuf_bottom.fill(0)
        color_int = self._rgb_to_int(CYAN)
        if now > self.bottom_last + self.bottom_wait:
            self.__coord_wrap()
            self.bottom_last = now
            x, y, w, h = self.bottom_squares[self.bottom_squares_idx]
            self.pixel_framebuf_bottom.rect(x, y, w, h, color_int)
            self.pixel_framebuf_bottom.display()

    def __coord_wrap(self):
        self.bottom_squares_idx = self.bottom_squares_idx + 1
        if self.bottom_squares_idx >= len(self.bottom_squares):
            self.bottom_squares_idx = 0

    def __display_top_pixels(self):
        self.pixel_framebuf_top.fill(0)

        for coord in self.top_pixel_coords:
            x, y = coord.split(":")
            self.pixel_framebuf_top.pixel(int(x), int(y), self.top_pixel_color)
        self.pixel_framebuf_top.display()

    def __scroll_framebuf(self, word, shift_x, shift_y):
        self.pixel_framebuf_sides.fill(0)

        color = self.__next_color()
        color_int = self._rgb_to_int(color)
```

```
        # negate x so that the word can be shown from left to right
        self.pixel_framebuf_sides.text(word, -shift_x, shift_y, color_int)
        self.pixel_framebuf_sides.display()

    def __next_color(self):
        if self.color_idx >= len(self.color_list):
            self.color_idx = 0

        result = self.color_list[self.color_idx]
        now = time.monotonic()
        if now >= self.last_color_time + self.color_wait:
            self.color_idx = self.color_idx + 1
            self.last_color_time = now

        return result

    @staticmethod
    def _rgb_to_int(rgb):
        return rgb[0] << 16 | rgb[1] << 8 | rgb[2]
```

Testing before final assembly

It will be a good idea to make sure that all the soldering and wiring we've done in the previous step went correctly. Upload both **code.py** and **cube.py** into your **CIRCUITPY** drive, and you should see this brief animation to verify that everything is working:



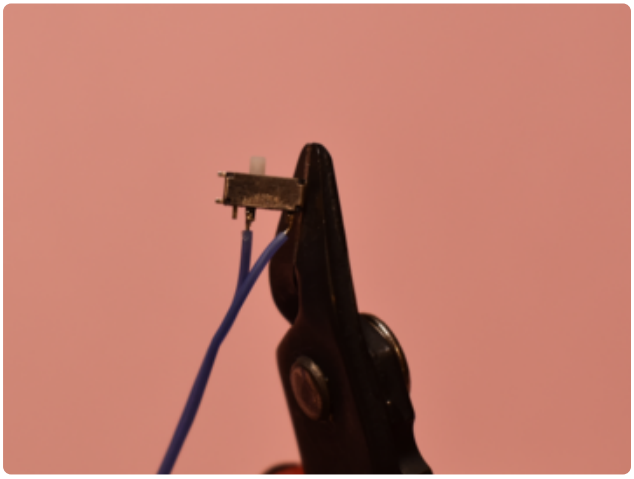
If you're seeing stuff on the matrices, that means you're good to go! You might even wiggle the accelerometer a bit, it should trigger the "upside down" animation which can help to make sure that the bottom matrix is also good to go.

Now we can move on to final assembly! Take a pause here and admire your work, you're almost done.

Final assembly

We're ready to stuff everything inside the cube! You may end up needing to sand the LED grids a bit more at this stage if they haven't quite been able to fit into the frame yet.

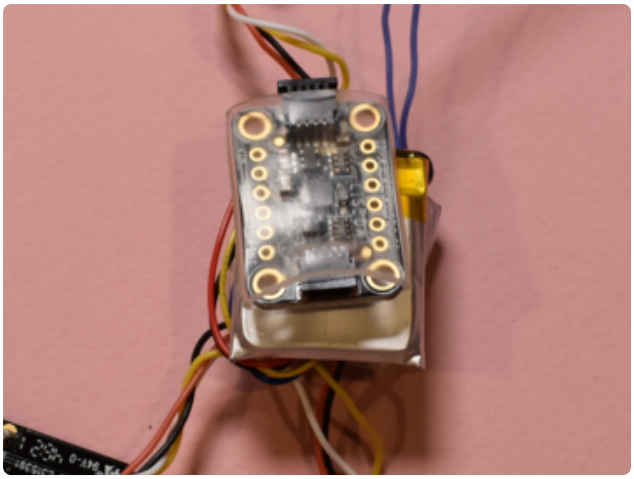
Use eye protection when cutting parts

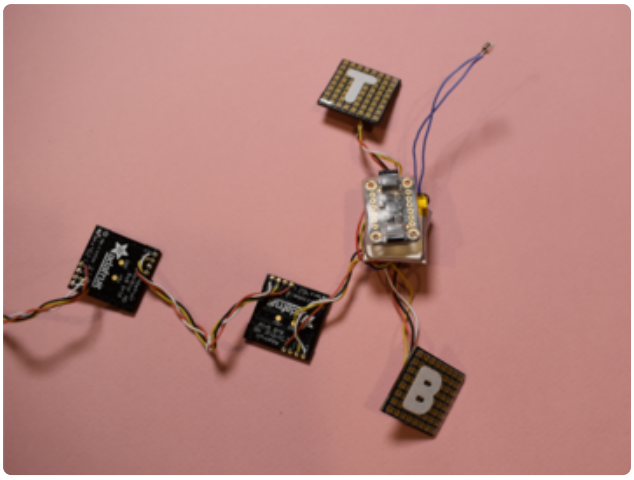


First, use a flush cutter to cut off the side tabs and the back knobs on the tiny switch. This will ensure that the switch will fit into the notch in the frame.

The "top" side of the cube should be the one with the switch notch.

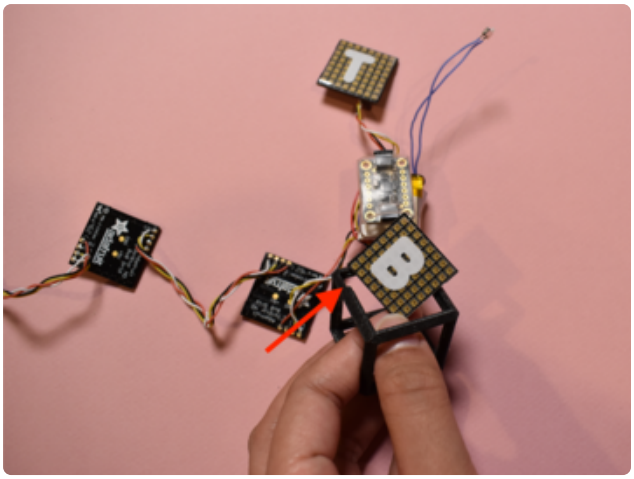
The idea is that all the grid panels will have to go through this top side before going through to their final side.



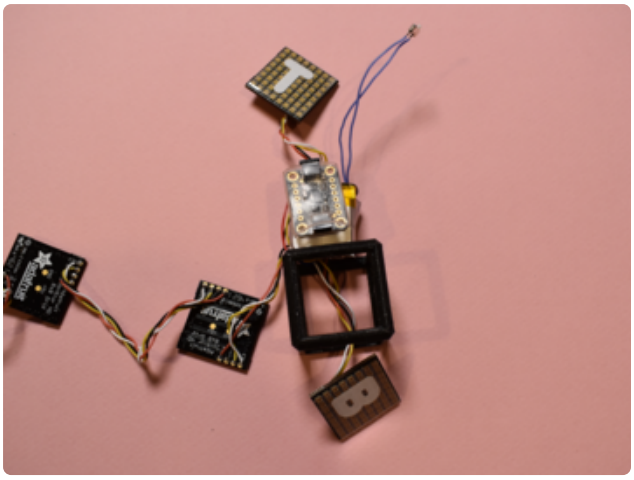


Make a stack with the QT Py at the bottom, the battery in the middle and the accelerometer on top.

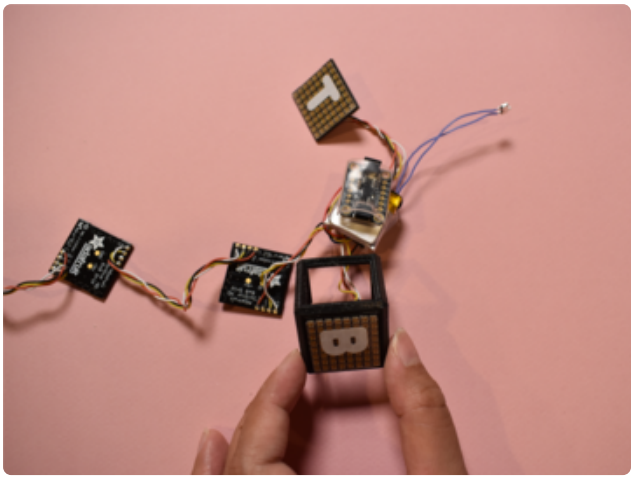
The LED grids for top and bottom have been labeled "T" for top and "B" for bottom in these next photos to help keep us oriented.



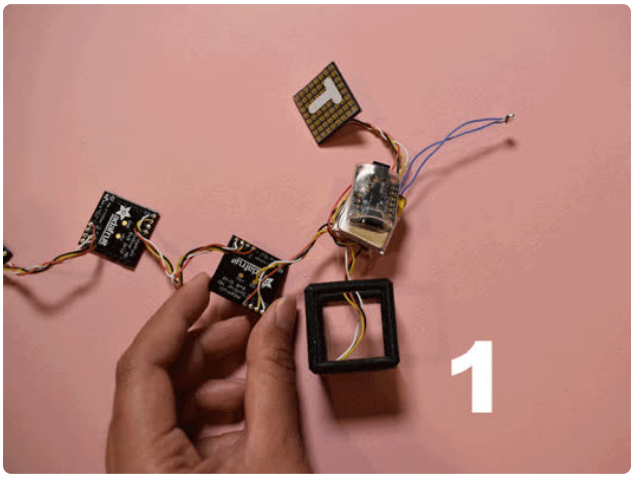
First, thread the bottom grid through the top side (notice where the notch is in the photo, indicated by the arrow).



Then, thread the bottom grid through to the bottom side of the cube (see the last photo).

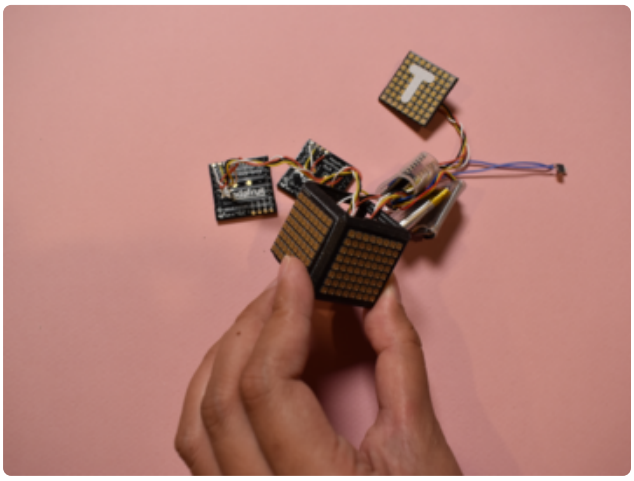


You can now press-fit the bottom grid to the bottom side. One side done, five more to go!

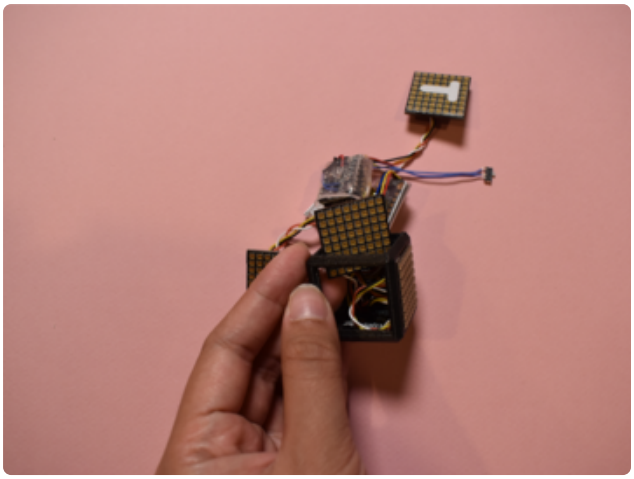


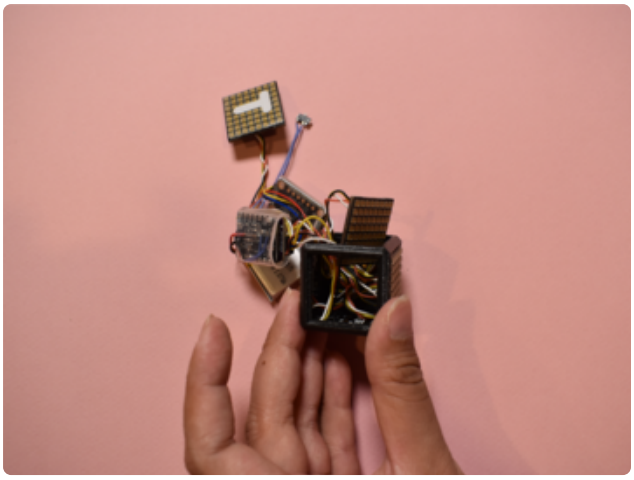
Take the first side panel, the one closest to the QT Py. Thread it through the top side first, then through one of the sides (it doesn't matter which). Here's an animated gif to better illustrate this.

You should orient this panel so that the cut tabs are on the top and bottom sides of the panel.

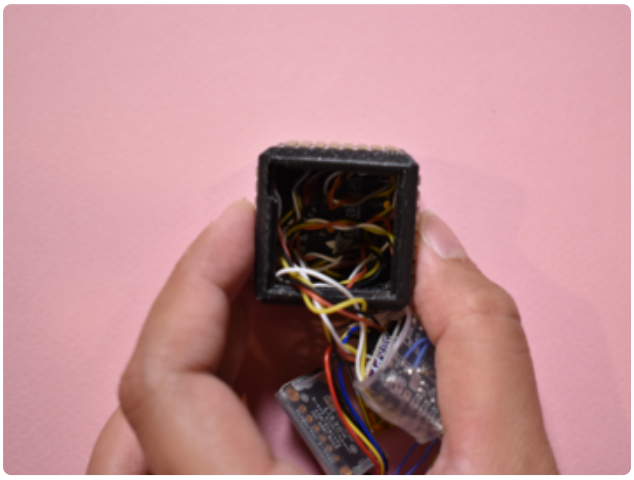


Take the next panel and thread it through the top first, then pick another side right next to the first panel. Press-fit this second panel into the frame in the same orientation as the first panel. You should now have just two sides left open, and the top side open.

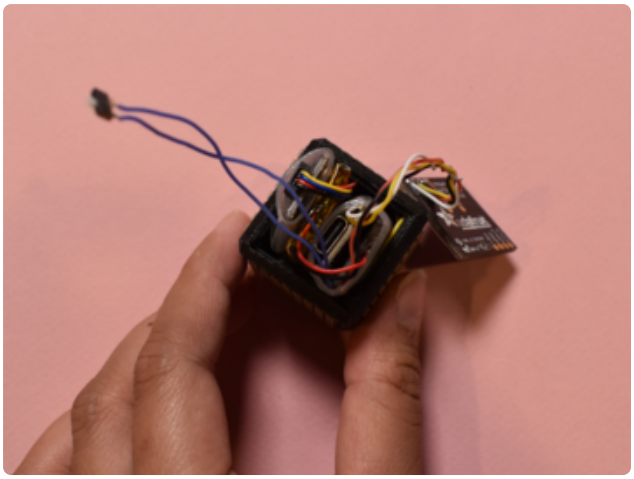




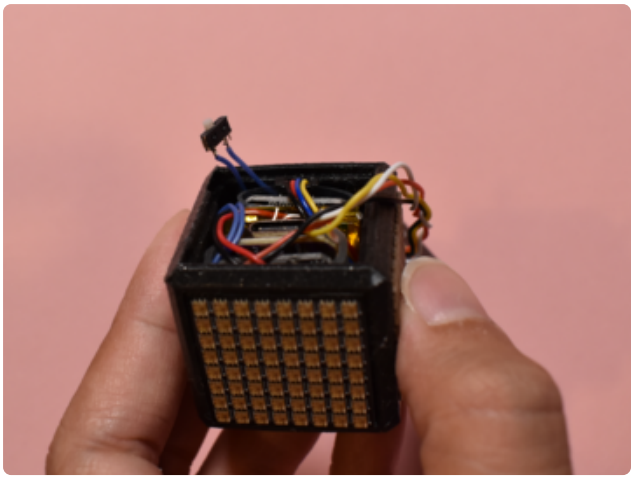
Do the same for the final two panels: first threading through the top as usual, then threading through the sides, in the same orientation as the first two panels. Press-fit these final side panels into the frame.



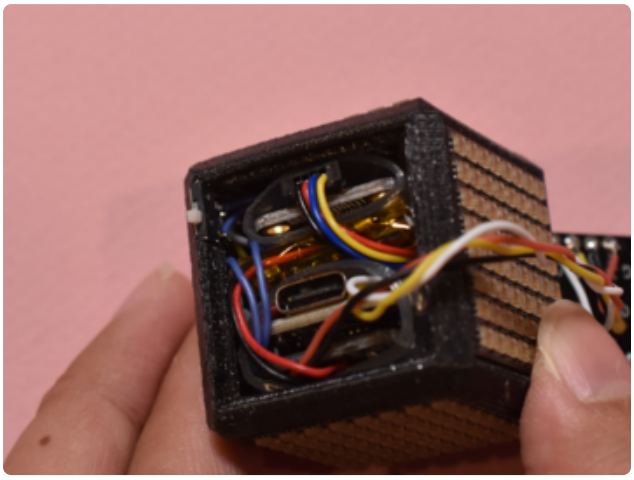
Now you should have just the top side left open. Don't worry, there should be enough space in there to stuff the rest of the electronics in. Use a wooden popsicle stick or something similarly non-conductive to gently push the wires towards the sides of the cube.



Carefully insert the stack of electronics inside the cube. You might find it easier to pop one or two panels out while you're doing this.

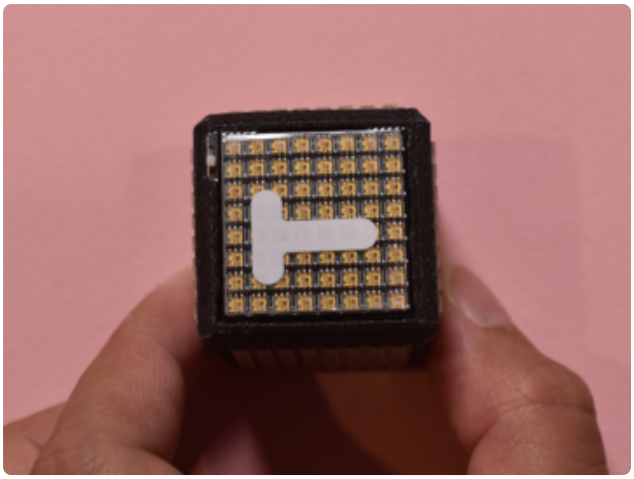


There should be enough room around the electronics to push the excess lengths of the switch wires in. Position the switch next to the notch in the frame.

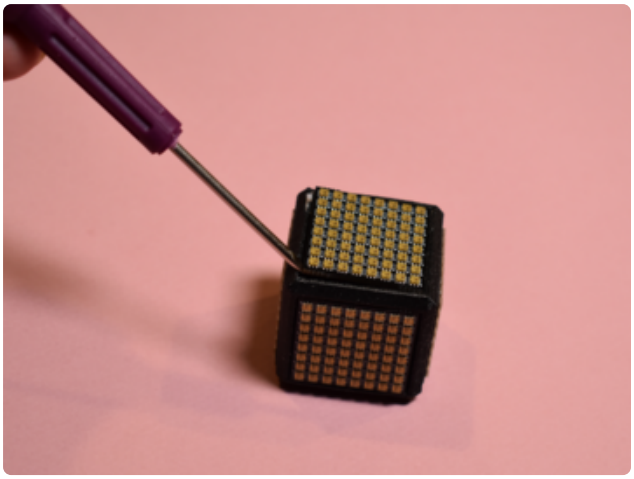


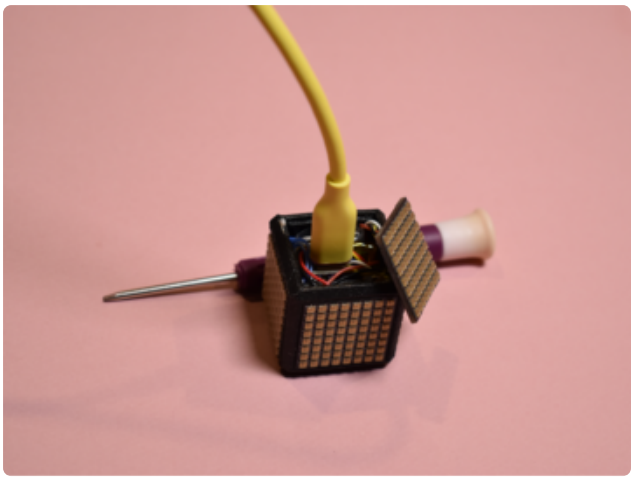
Fit the switch into the notch, silver side facing towards the frame. There should be no need to force it in. If it doesn't fit, check that the side tabs have been cut off enough, and also make sure that the little back knobs have also been cut off.

The body of the switch should be pretty much level with the frame.



Finally, press fit the top grid onto the top side of the frame. You may have to smush in the excess wire connected to the top grid into the cube before you can successfully press-fit the grid on.





There is another little notch on the same side as the switch notch that can fit a small flathead screwdriver. This makes it easy to remove the top panel to access the USB-C port for charging and re-programming the cube!

That's it! Congratulations, you've made your very own glowy companion cube. You can easily switch out the sensor board in this project if you want to program this adorable cube to do other things. With WiFi and even Bluetooth available on the ESP32-S3 chip, you'll just have to think outside the box. :D I hope you enjoy your happy little companion cube.

Resources

3D models for cube

https://adafru.it/10Rd

Fritzing circuit

https://adafru.it/10Re

The zip file below contains the webpage that makes it easy to publish a message to the two feeds mentioned in the guide. It is also linked in the guide itself, but adding here in case it needs to be changed.

cube-webpage.zip

https://adafru.it/10Rf