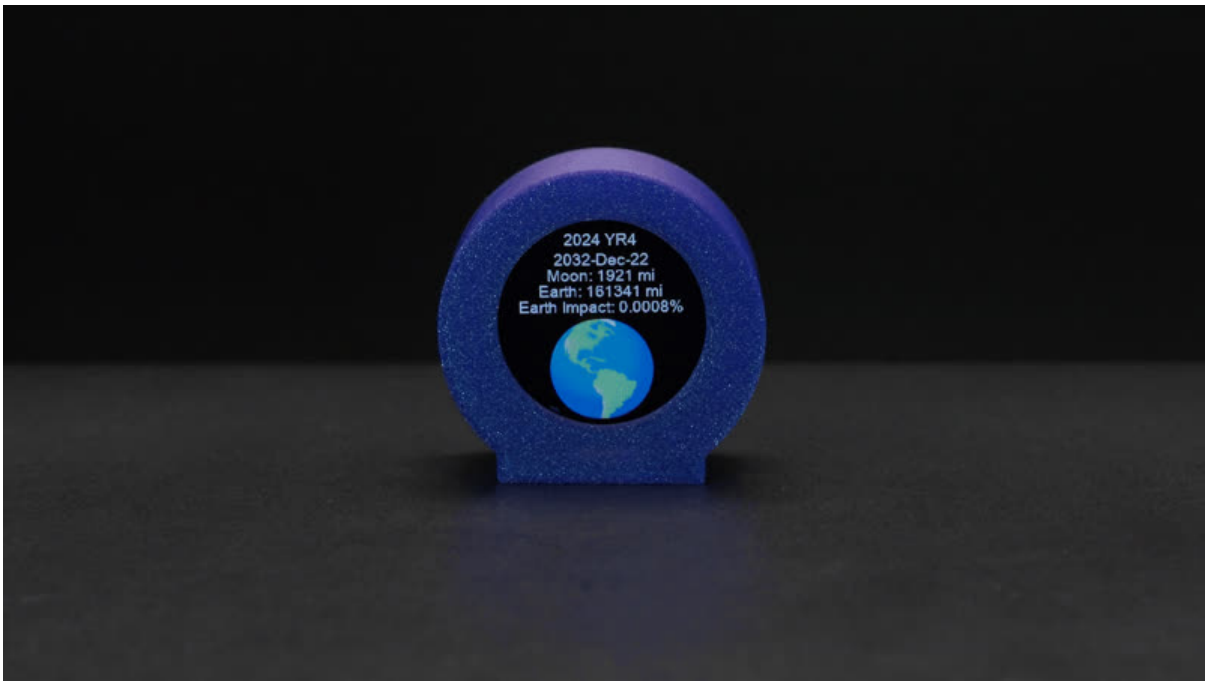
Asteroid Tracker

Created by Ruiz Brothers



https://learn.adafruit.com/asteroid-tracker

Last updated on 2025-04-14 09:23:35 AM EDT

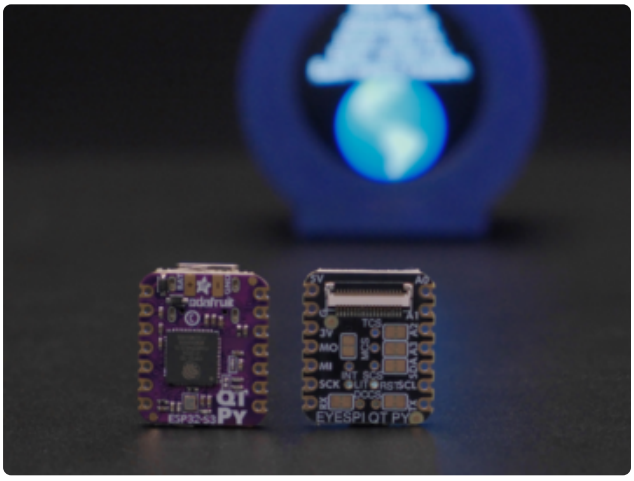
Table of Contents

Overview

Build an internet-connected Asteroid Tracker. This tracker displays the asteroid name, date, predicted distance from the moon, predicted distance from Earth and impact percentage.

This tracker is set to display the information for Asteroid 2024 YR4, an asteroid which at one time had a 3.1% chance of striking Earth.

The shift of the predicted path of Asteroid 2024 YR4 in the past few weeks shows it's much less likely to hit Earth but there is a small chance of a moon impact!

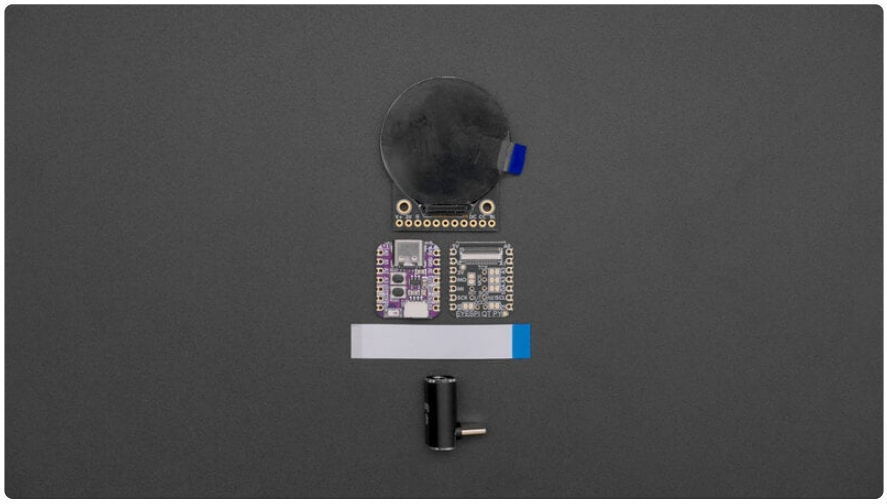


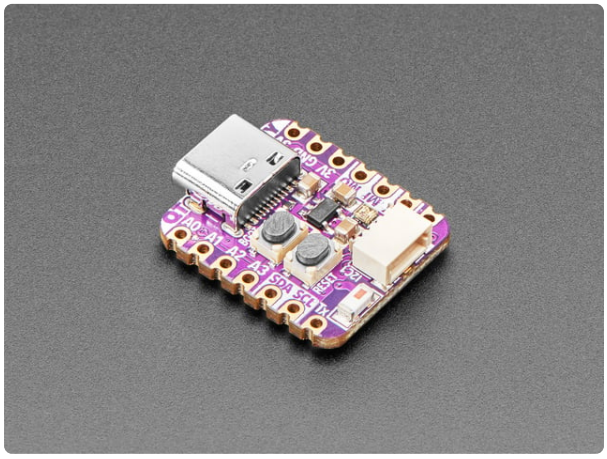
Powered by CircuitPython, this project runs on an Adafruit QT Py ESP32-S3 board connected to a 1.28" 240x240 round TFT LCD display. The display connects to the QT Py with an EYESPI BFF.

The QT Py with 4 MB flash / 2 MB PSRAM avoids memory issues with the bitmaps and the JSON data used by the code.

This project uses the NASA Sentry API to display tracking information, which doesn't need an API key, making it easier to set up!

Parts

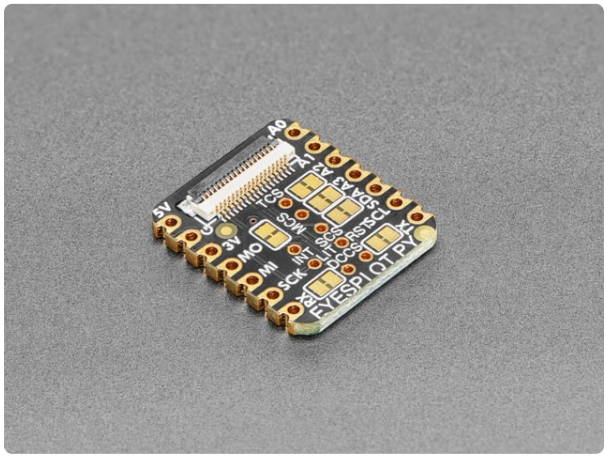




[Adafruit QT Py S3 with 2MB PSRAM WiFi Dev Board with STEMMA QT]

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...

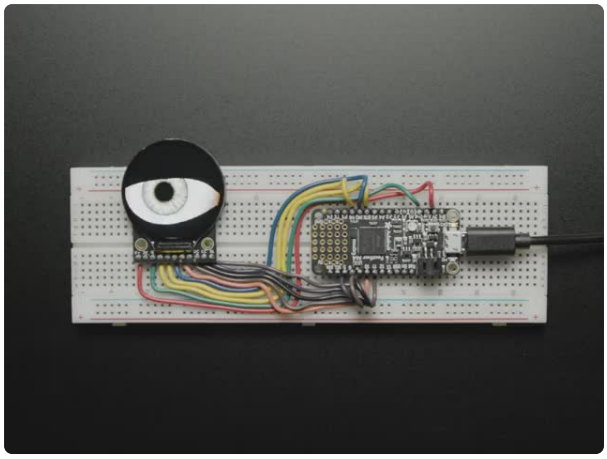
https://www.adafruit.com/product/5700



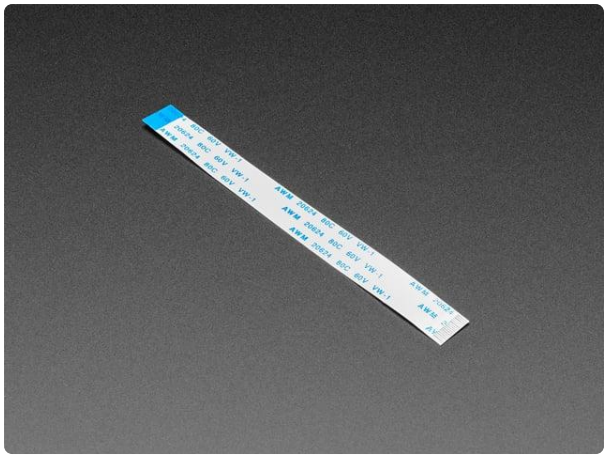
[Adafruit EYESPI BFF for QT Py or Xiao - 18 Pin FPC Connector]

Our QT Py boards are a great way to make very small microcontroller projects that pack a ton of power - and now we have a way for you to add a small, colorful, and bright display to...

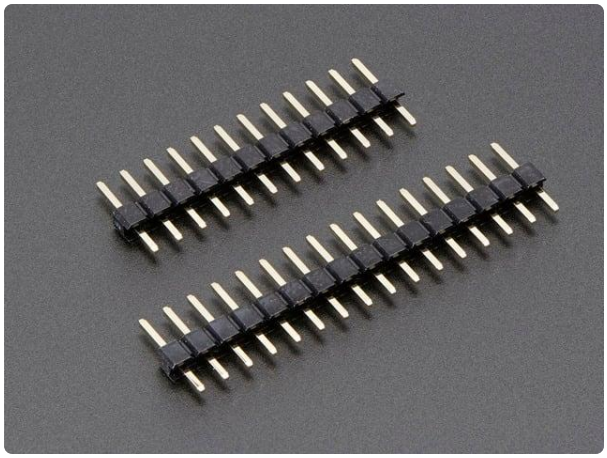
https://www.adafruit.com/product/5772



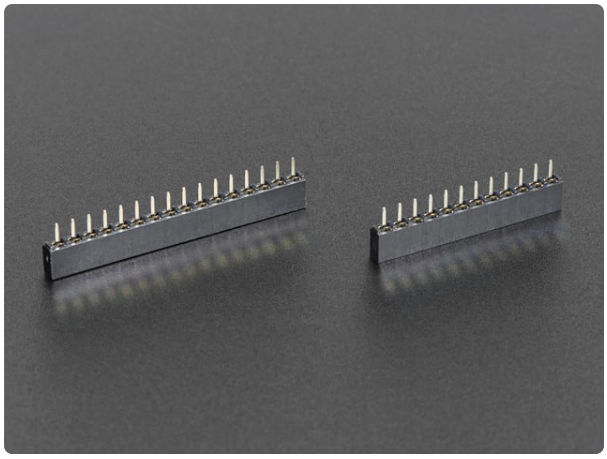
[Adafruit 1.28" 240x240 Round TFT LCD Display with MicroSD]
'Round these parts we enjoy unusually-shaped displays. And this one certainly fits the description - it's a 1.28" diagonal TFT that comes in a round shape and contains a...
https://www.adafruit.com/product/6178



[EYESPI Cable - 18 Pin 100mm long Flex PCB (FPC) A-B type]
Connect this to that when a 18-pin FPC connector is needed. This 25 cm long cable is made of a flexible PCB. It's A-B style which means that pin one on one side will match...
https://www.adafruit.com/product/5239



[Short Feather Male Headers - 12-pin and 16-pin Male Header Set]
These two Short Male Headers alone are, well, lonely. But pair them with any of our
https://www.adafruit.com/product/3002



[Short Headers Kit for Feather - 12-pin + 16-pin Female Headers]
These two Short Female Headers alone are, well, lonely. But pair them with any of our
https://www.adafruit.com/product/2940

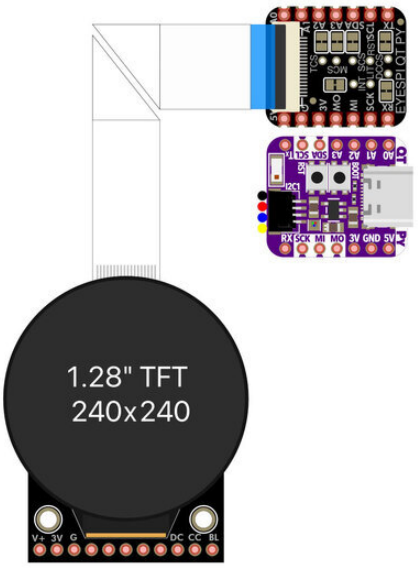
1 x USB C 90 Degree Adapter USB C 90 Degree Adapter	https://www.amazon.com/dp/B0BNYGSDPX
4 x M2.5x5mm Machine Screws M2.5-0.45x5mm Flat Countersunk Head Machine Screws	https://www.amazon.com/dp/B0DCNTY8SX

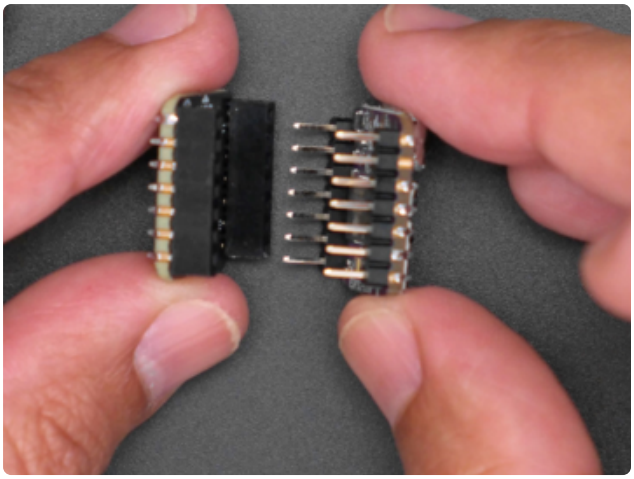
Circuit Diagram

The diagram below provides a general visual reference for wiring of the components once you get to the **Assembly** page. This diagram was created using the software package Fritzing (https://adafru.it/oEP).

Adafruit Library for Fritzing

Adafruit uses the Adafruit's Fritzing parts library to create circuit diagrams for projects. You can download the library or just grab individual parts. Get the library and parts from GitHub - Adafruit Fritzing Parts (https://adafru.it/AYZ).





Header Connections

The EYE SPI board connects to the QT Py via short header pins and short header sockets.

The header pins are soldered to the QT Py.

The socket headers are soldered to the EYE SPI board.

Install CircuitPython

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

CircuitPython Quickstart

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

There are two versions of this board: one with 8MB Flash/No PSRAM and one with 4MB Flash/2MB PSRAM. Each version has their own UF2 build for CircuitPython. There isn't an easy way to identify which version of the board you have by looking at the board silk. If you aren't sure which version you have, try either build to see which one works.

There are two versions of this board: one with 8MB Flash/No PSRAM and one with 4MB Flash/2MB PSRAM.

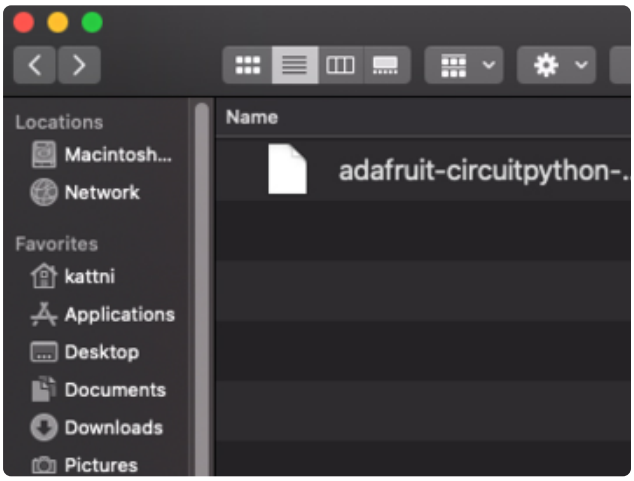
Download the latest version of CircuitPython for the 8MB/No PSRAM version of this board via circuitpython.org

https://adafru.it/-CL

Download the latest version of CircuitPython for the 4MB/2MB

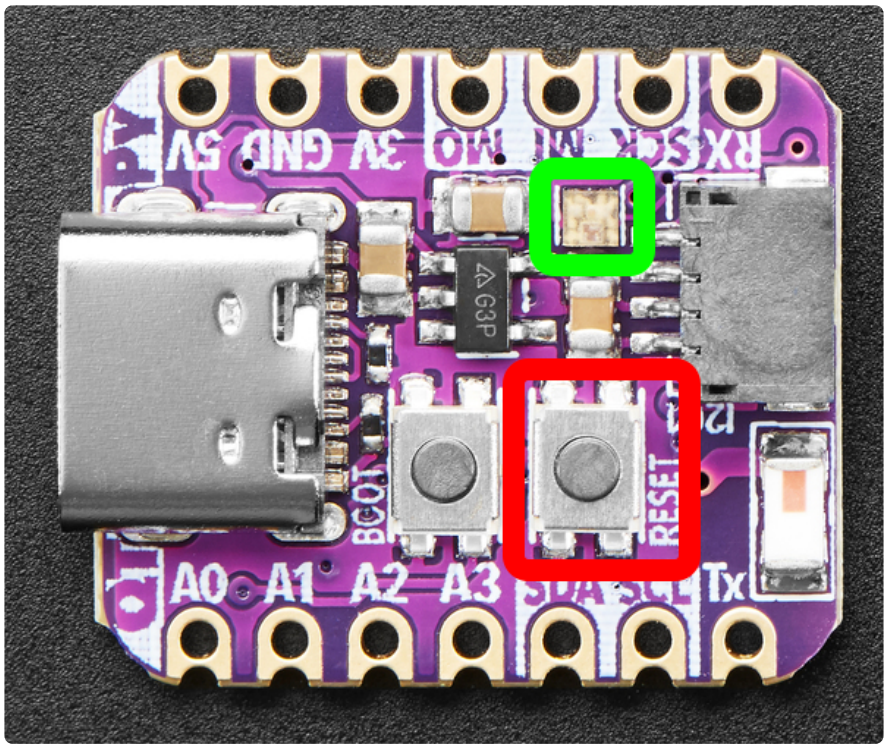
PSRAM version of this board via circuitpython.org

https://adafru.it/18fo



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

Save it wherever is convenient for you.



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

Click the **reset** button once (highlighted in red above), and then click it again when you see the **RGB status LED(s)** (highlighted in green above) turn purple (approximately half a second later). Sometimes it helps to think of it as a "slow double-click" of the reset button.

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

On some very old versions of the UF2 bootloader, the status LED turns red instead of purple.

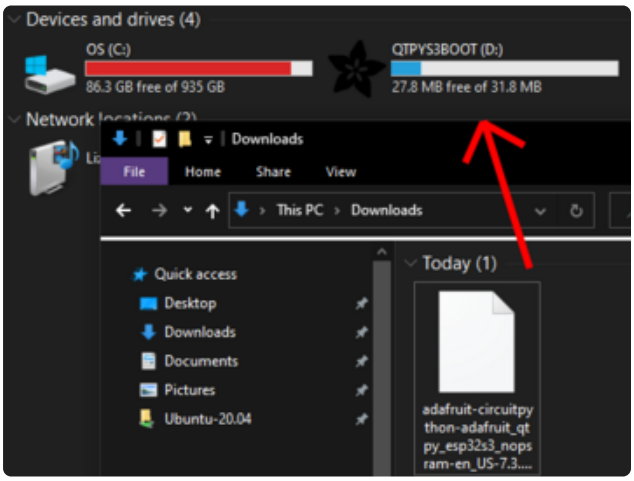
For this board, tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again. The second tap needs to happen while the LED is still purple.

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

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

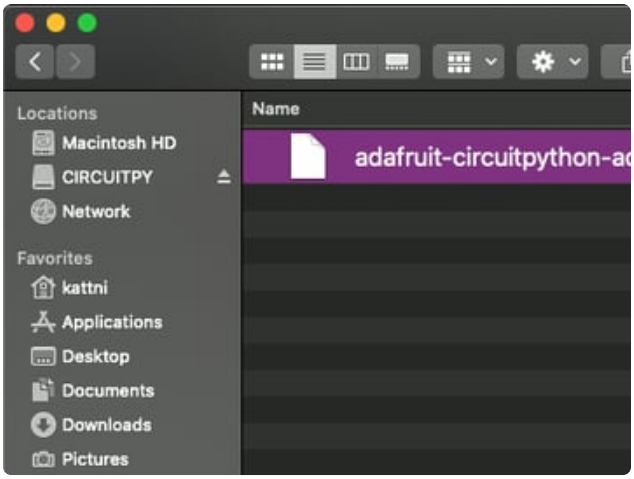
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**

> If after several tries, and verifying your USB cable is data-ready, you still cannot get to the bootloader, it is possible that the bootloader is missing or damaged. Check out the Factory Reset page for details on resolving this issue.



You will see a new disk drive appear called **QTPYS3BOOT**.

Drag the **adafruit_circuitpython_etc.uf2** file to **QTPYS3BOOT**.



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

That's it!

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of CircuitPython 8 (https://adafru.it/Em8), there is support for a **settings.toml** file. This is a file that is stored on your **CIRCUITPY** drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT services. It is designed to separate your sensitive information from your **code.py** file so you are able to share your code without sharing your credentials.

CircuitPython previously used a **secrets.py** file for this purpose. The **settings.toml** file is quite similar.

> Your settings.toml file should be stored in the main directory of your CIRCUITPY drive. It should not be in a folder.

CircuitPython **settings.toml** File

This section will provide a couple of examples of what your **settings.toml** file should look like, specifically for CircuitPython WiFi projects in general.

The most minimal **settings.toml** file must contain your WiFi SSID and password, as that is the minimum required to connect to WiFi. Copy this example, paste it into your **settings.toml**, and update:

- `your_wifi_ssid`
- `your_wifi_password`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
```

Many CircuitPython network-connected projects on the Adafruit Learn System involve using Adafruit IO. For these projects, you must also include your Adafruit IO username and key. Copy the following example, paste it into your settings.toml file, and update:

- `your_wifi_ssid`
- `your_wifi_password`
- `your_aio_username`
- `your_aio_key`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_AIO_USERNAME = "your_aio_username"
ADAFRUIT_AIO_KEY = "your_aio_key"
```

Some projects use different variable names for the entries in the **settings.toml** file. For example, a project might use `ADAFRUIT_AIO_ID` in the place of `ADAFRUIT_AIO_USERNAME`. **If you run into connectivity issues, one of the first things to check is that the names in the settings.toml file match the names in the code.**

> Not every project uses the same variable name for each entry in the settings.toml file! Always verify it matches the code.

settings.toml File Tips

Here is an example **settings.toml** file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

In a **settings.toml** file, it's important to keep these factors in mind:

- Strings are wrapped in double quotes; ex: `"your-string-here"`

- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
    - Floats, octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in **.toml** files
    - Example: `\U0001f44d` for (thumbs up emoji) and `\u20ac` for € (EUR sign)
- Unicode emoji, and non-ASCII characters, stand for themselves as long as you're careful to save in "UTF-8 without BOM" format

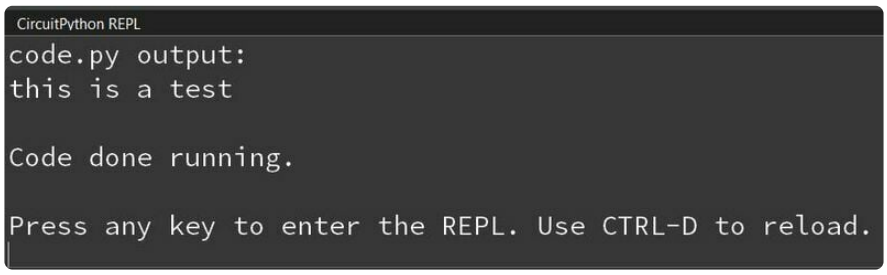
When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

Accessing Your **settings.toml** Information in **code.py**

In your **code.py** file, you'll need to `import` the `os` library to access the **settings.toml** file. Your settings are accessed with the `os.getenv()` function. You'll pass your settings entry to the function to import it into the **code.py** file.

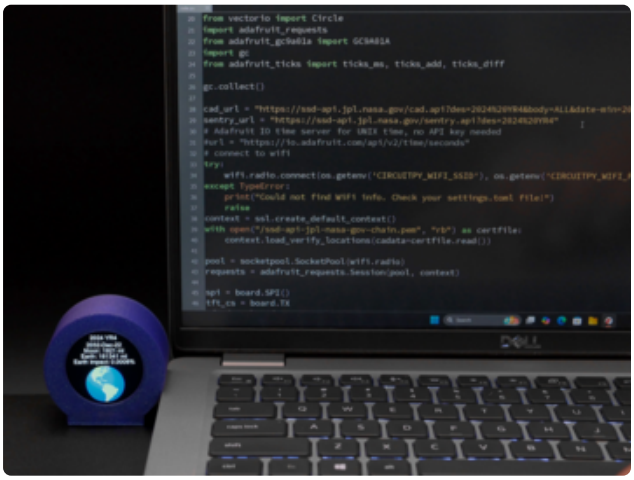
```
import os

print(os.getenv("test_variable"))
```



In the upcoming CircuitPython WiFi examples, you'll see how the **settings.toml** file is used for connecting to your SSID and accessing your API keys.

Code



Once you've finished setting up your QT Py ESP32-S3 with CircuitPython, you can access the code, images, font and necessary libraries by downloading the Project Bundle.

To do this, click on the **Download Project Bundle** button in the window below. It will download to your computer as a zipped folder.

```
# SPDX-FileCopyrightText: 2025 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import time
import ssl
import board
import wifi
import socketpool
import microcontroller
import displayio
from adafruit_display_text.bitmap_label import Label
from adafruit_bitmap_font import bitmap_font
import adafruit_imageload
from fourwire import FourWire
import adafruit_requests
from adafruit_gc9a01a import GC9A01A
from adafruit_ticks import ticks_ms, ticks_add, ticks_diff

cad_url = ("https://ssd-api.jpl.nasa.gov/cad.api?"
           "des=2024%20YR4&body=ALL&"
           "date-min=2030-01-01&date-max=2060-01-01")
sentry_url = "https://ssd-api.jpl.nasa.gov/sentry.api?des=2024%20YR4"
# connect to wifi
try:
    wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
os.getenv('CIRCUITPY_WIFI_PASSWORD'))
except TypeError:
    print("Could not find WiFi info. Check your settings.toml file!")
    raise
context = ssl.create_default_context()
with open("/ssd-api-jpl-nasa-gov-chain.pem", "rb") as certfile:
    context.load_verify_locations(cadata=certfile.read())

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

spi = board.SPI()
tft_cs = board.TX
tft_dc = board.RX
tft_reset = None

displayio.release_displays()

display_bus = FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=tft_reset)
```

```
display = GC9A01A(display_bus, width=240, height=240, auto_refresh=False)

main_group = displayio.Group()
display.root_group = main_group

bitmap_bg, palette_bg = adafruit_imageload.load("/earth_bg.bmp",
                                        bitmap=displayio.Bitmap,
                                        palette=displayio.Palette)

grid_bg = displayio.TileGrid(bitmap_bg, pixel_shader=palette_bg)
main_group.append(grid_bg)

font = bitmap_font.load_font('/Arial-14.bdf')
name_area = Label(font, text="2024 YR4", color=0xFFFFFF, background_color=0x000000)
name_area.anchored_position = (display.width / 2, 0)
name_area.anchor_point = (0.5, 0.0)

date_area = Label(font, text="2032-12-22", color=0xFFFFFF,
background_color=0x000000)
date_area.anchored_position = (display.width / 2, name_area.height+10)
date_area.anchor_point = (0.5, 0.0)

moon_area = Label(font, text="Moon: ", color=0xFFFFFF, background_color=0x000000)
moon_area.anchored_position = (display.width / 2, name_area.height+10 +
date_area.height+5)
moon_area.anchor_point = (0.5, 0.0)

earth_area = Label(font, text="Earth: ", color=0xFFFFFF, background_color=0x000000)
earth_area.anchored_position = (display.width / 2, name_area.height+10 +
                                                   moon_area.height+5 +
                                                   date_area.height + 5)
earth_area.anchor_point = (0.5, 0.0)

impact_area = Label(font, text="Earth Impact: 0.0000%", color=0xFFFFFF,
background_color=0x000000)
impact_area.anchored_position = (display.width / 2, name_area.height+10 +
                                                    moon_area.height+5 +
                                                    earth_area.height + 5 +
                                                    date_area.height + 5)
impact_area.anchor_point = (0.5, 0.0)
main_group.append(impact_area)
main_group.append(earth_area)
main_group.append(moon_area)
main_group.append(date_area)
main_group.append(name_area)

bit_asteroid, pal_asteroid = adafruit_imageload.load("/asteroid.bmp",
                                        bitmap=displayio.Bitmap,
                                        palette=displayio.Palette)

asteroid = displayio.TileGrid(bit_asteroid, pixel_shader=pal_asteroid,
                              x = 25, y=100)
pal_asteroid.make_transparent(0)
main_group.append(asteroid)

def diagonal_travel(bitmap_object, start_x=-59, start_y=-59, end_x=240, end_y=240,
delay=0.01):
    # Set initial position
    bitmap_object.x = start_x
    bitmap_object.y = start_y

    # Calculate total movement distance
    distance_x = end_x - start_x
    distance_y = end_y - start_y

    # Calculate number of steps (use the larger distance)
    steps = max(abs(distance_x), abs(distance_y)) // 1

    # Calculate step size for each axis to maintain diagonal movement
```

```
    step_x = distance_x / steps
    step_y = distance_y / steps

    # Animate the movement
    for i in range(steps + 1):
        # Update position
        bitmap_object.x = int(start_x + (step_x * i))
        bitmap_object.y = int(start_y + (step_y * i))
        display.refresh()
        # Pause to control animation speed
        time.sleep(delay)

def au_to_miles(au):
    # 1 AU = 92,955,807 miles
    miles_per_au = 92955807

    return au * miles_per_au

timer_clock = ticks_ms()
timer = 3600 * 1000
first_run = True

while True:
    try:
        if first_run or ticks_diff(ticks_ms(), timer_clock) >= timer:
            sentry_response = requests.get(sentry_url)
            sentry_json = sentry_response.json()
            impact = sentry_json['summary']['ip']
            sentry_response.close()
            overall_ip = float(impact) * 100
            cad_response = requests.get(cad_url)
            cad_json = cad_response.json()
            earth_distance = au_to_miles(float(cad_json['data'][0][4]))
            earth_area.text = f"{cad_json['data'][0][10]}: {int(earth_distance)} mi"
            moon_distance = au_to_miles(float(cad_json['data'][1][4]))
            moon_area.text = f"{cad_json['data'][1][10]}: {int(moon_distance)} mi"
            date = cad_json['data'][0][3]
            date = date.split()
            date_area.text = f"{date[0]}"
            cad_response.close()
            impact_area.text = f"Earth Impact: {overall_ip:.4f}%"
            display.refresh()
            timer_clock = ticks_add(timer_clock, timer)
        diagonal_travel(asteroid, start_x=-45, start_y=300, end_x=300, end_y=-45)
        time.sleep(0.1)
    # pylint: disable=broad-except
    except Exception as e:
        print("Error:\n", str(e))
        print("Resetting microcontroller in 10 seconds")
        time.sleep(10)
        microcontroller.reset()
```

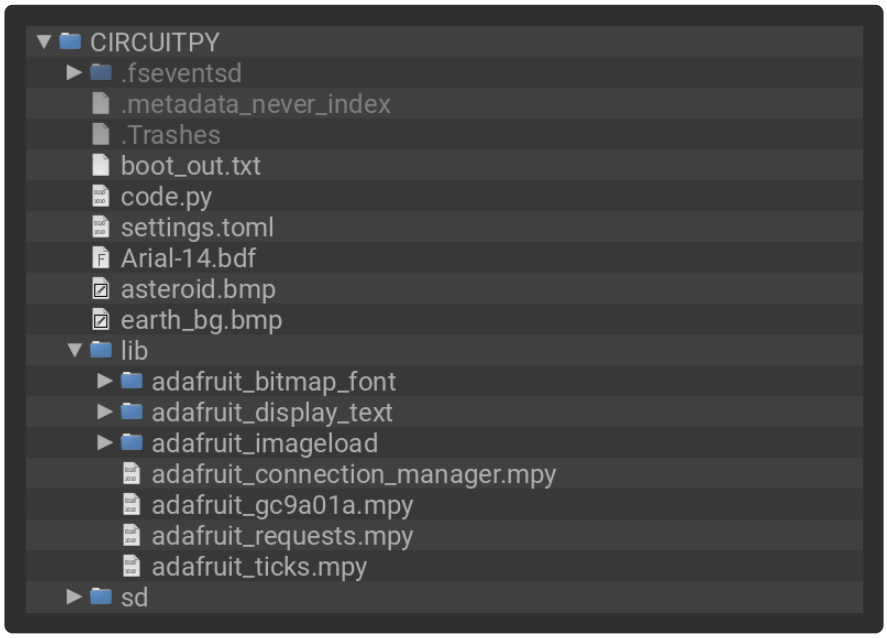
Upload the Code and Libraries to the QT Py

After downloading the Project Bundle, plug your QT Py into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called **CIRCUITPY**. Unzip the folder and copy the following items to the QT Py's **CIRCUITPY** drive.

- **lib** folder
- **code.py**

- **asteroid.bmp**
- **earth_bg.bmp**
- **Arial-14.bdf**

Your QT Py ESP32-S3 **CIRCUITPY** drive should look like this after copying the **lib** folder, image files, font file and the **code.py** file.



Add Your **settings.toml** File

As of CircuitPython 8.0.0, there is support for Environment Variables (https://adafru.it/11wE). Environment variables are stored in a **settings.toml** file. Similar to **secrets.py**, the **settings.toml** file separates your sensitive information from your main **code.py** file. Add your **settings.toml** file as described in the Create Your settings.toml File page (https://adafru.it/1afW) earlier in this guide. You'll need to include values for your `CIRCUITPY_WIFI_SSID` and `CIRCUITPY_WIFI_PASSWORD`.

```
CIRCUITPY_WIFI_SSID = "your-ssid-here"
CIRCUITPY_WIFI_PASSWORD = "your-ssid-password-here"
```

How the Code Works

The asteroid tracking information is available via two NASA JPL API's: the Sentry API (https://adafru.it/1ahh) that tracks objects that pose a threat to Earth and the Close Approach API (https://adafru.it/1ahi) that tracks objects that will pass near the planets in the solar system and our moon.

Both URLs are setup to track the 2024 YR4 asteroid (https://adafru.it/1ahj) that is estimated to make a close approach to the moon and Earth in 2032.

```
cad_url = ("https://ssd-api.jpl.nasa.gov/cad.api?"
           "des=2024%20YR4&amp;body=ALL&amp;"
           "date-min=2030-01-01&amp;date-max=2060-01-01")
sentry_url = "https://ssd-api.jpl.nasa.gov/sentry.api?des=2024%20YR4"
# connect to wifi
try:
    wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
os.getenv('CIRCUITPY_WIFI_PASSWORD'))
except TypeError:
    print("Could not find WiFi info. Check your settings.toml file!")
    raise
context = ssl.create_default_context()
with open("/ssd-api-jpl-nasa-gov-chain.pem", "rb") as certfile:
    context.load_verify_locations(cadata=certfile.read())

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

Graphics

The round display is connected via SPI with the EYESPI BFF cable. The background bitmap image positions the Earth in the bottom half of the screen. Five text objects are created to display the asteroid name, estimated date, the distance it will be from the moon, the distance it will be from Earth and the percentage of a likely impact with Earth. The second bitmap is a small asteroid. Its background color is made transparent with its palette (`pal_asteroid.make_transparent(0)`).

```
spi = board.SPI()
tft_cs = board.TX
tft_dc = board.RX
tft_reset = None

displayio.release_displays()

display_bus = FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=tft_reset)
display = GC9A01A(display_bus, width=240, height=240, auto_refresh=False)

main_group = displayio.Group()
display.root_group = main_group

bitmap_bg, palette_bg = adafruit_imageload.load("/earth_bg.bmp",
                                        bitmap=displayio.Bitmap,
                                        palette=displayio.Palette)

grid_bg = displayio.TileGrid(bitmap_bg, pixel_shader=palette_bg)
main_group.append(grid_bg)

font = bitmap_font.load_font('/Arial-14.bdf')
name_area = Label(font, text="2024 YR4", color=0xFFFFFF, background_color=0x000000)
name_area.anchored_position = (display.width / 2, 0)
name_area.anchor_point = (0.5, 0.0)

date_area = Label(font, text="2032-12-22", color=0xFFFFFF,
background_color=0x000000)
date_area.anchored_position = (display.width / 2, name_area.height+10)
date_area.anchor_point = (0.5, 0.0)

moon_area = Label(font, text="Moon: ", color=0xFFFFFF, background_color=0x000000)
moon_area.anchored_position = (display.width / 2, name_area.height+10 +
date_area.height+5)
moon_area.anchor_point = (0.5, 0.0)
```

```
earth_area = Label(font, text="Earth: ", color=0xFFFFFF, background_color=0x000000)
earth_area.anchored_position = (display.width / 2, name_area.height+10 +
                                                   moon_area.height+5 +
                                                   date_area.height + 5)
earth_area.anchor_point = (0.5, 0.0)

impact_area = Label(font, text="Earth Impact: 0.0000%", color=0xFFFFFF,
background_color=0x000000)
impact_area.anchored_position = (display.width / 2, name_area.height+10 +
                                                    moon_area.height+5 +
                                                    earth_area.height + 5 +
                                                    date_area.height + 5)
impact_area.anchor_point = (0.5, 0.0)
main_group.append(impact_area)
main_group.append(earth_area)
main_group.append(moon_area)
main_group.append(date_area)
main_group.append(name_area)

bit_asteroid, pal_asteroid = adafruit_imageload.load("/asteroid.bmp",
                                          bitmap=displayio.Bitmap,
                                          palette=displayio.Palette)

asteroid = displayio.TileGrid(bit_asteroid, pixel_shader=pal_asteroid,
                              x = 25, y=100)
pal_asteroid.make_transparent(0)
main_group.append(asteroid)
```

Diagonally Across

The asteroid bitmap moves diagonally across the display every few seconds with the help of the `diagonal_travel()` function. You can change the starting and ending `x` and `y` coordinates to affect the direction of the bitmap.

```
def diagonal_travel(bitmap_object, start_x=-59, start_y=-59, end_x=240, end_y=240,
delay=0.01):
    # Set initial position
    bitmap_object.x = start_x
    bitmap_object.y = start_y

    # Calculate total movement distance
    distance_x = end_x - start_x
    distance_y = end_y - start_y

    # Calculate number of steps (use the larger distance)
    steps = max(abs(distance_x), abs(distance_y)) // 1

    # Calculate step size for each axis to maintain diagonal movement
    step_x = distance_x / steps
    step_y = distance_y / steps

    # Animate the movement
    for i in range(steps + 1):
        # Update position
        bitmap_object.x = int(start_x + (step_x * i))
        bitmap_object.y = int(start_y + (step_y * i))
        display.refresh()
        # Pause to control animation speed
        time.sleep(delay)
```

Convert Astronomical Units

Another helper function is `au_to_miles()` this converts astronomical units to miles. The API results are delivered in au.

```
def au_to_miles(au):
    # 1 AU = 92,955,807 miles
    miles_per_au = 92955807

    return au * miles_per_au
```

The Loop

In the loop, `ticks` are used to keep track of time. Every hour, requests are made to the two API's. The retrieved data is used to update the text elements. Outside of `ticks`, the `diagonal_travel()` function is used to animate the asteroid bitmap across the screen. All of this is wrapped in a `try`/`except` to reboot the QT Py if there are any connection issues or errors.

```
while True:
    try:
        if first_run or ticks_diff(ticks_ms(), timer_clock) &gt;= timer:
            sentry_response = requests.get(sentry_url)
            sentry_json = sentry_response.json()
            impact = sentry_json['summary']['ip']
            sentry_response.close()
            overall_ip = float(impact) * 100
            cad_response = requests.get(cad_url)
            cad_json = cad_response.json()
            earth_distance = au_to_miles(float(cad_json['data'][0][4]))
            earth_area.text = f"{cad_json['data'][0][10]}: {int(earth_distance)} mi"
            moon_distance = au_to_miles(float(cad_json['data'][1][4]))
            moon_area.text = f"{cad_json['data'][1][10]}: {int(moon_distance)} mi"
            date = cad_json['data'][0][3]
            date = date.split()
            date_area.text = f"{date[0]}"
            cad_response.close()
            impact_area.text = f"Earth Impact: {overall_ip:.4f}%"
            display.refresh()
            timer_clock = ticks_add(timer_clock, timer)
        diagonal_travel(asteroid, start_x=-45, start_y=300, end_x=300, end_y=-45)
        time.sleep(0.1)
    # pylint: disable=broad-except
    except Exception as e:
        print("Error:\n", str(e))
        print("Resetting microcontroller in 10 seconds")
        time.sleep(10)
        microcontroller.reset()
```

3D Printing

3D Printed Parts

STL files for 3D printing will need to be oriented for printing using either FDM or SLS machines.

Parts were printed with PLA filament. The black is generic. The white used is specified in Parts on the Overview page.

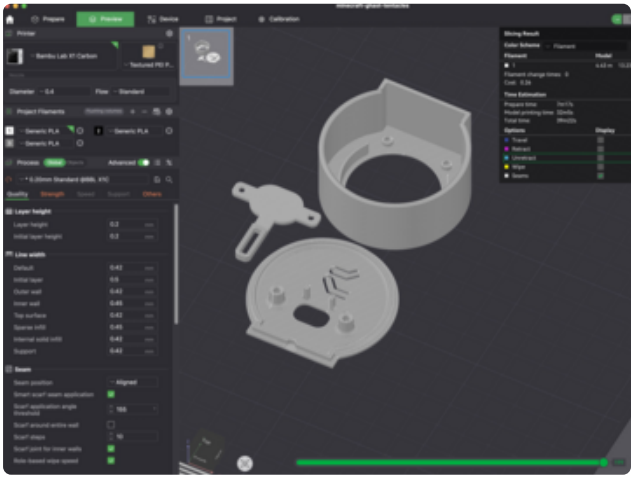
Original design source files may be downloaded using the links below.

Download Asteroid STLs

https://adafru.it/1ahk

Edit Design

https://adafru.it/1ahl

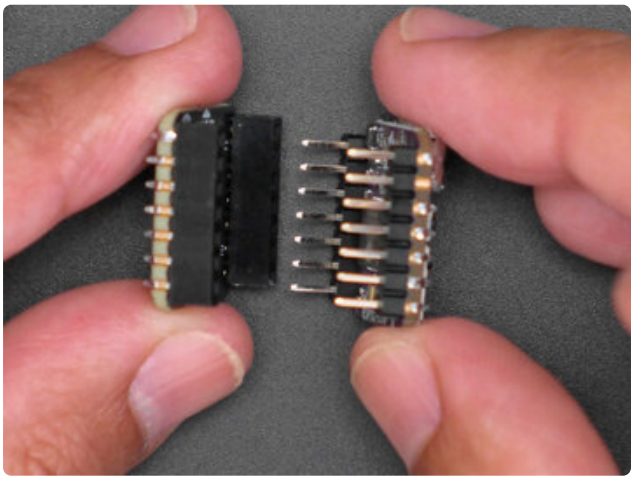


Slice with settings for PLA material

The parts were sliced using BambuStudio using the slice settings below.

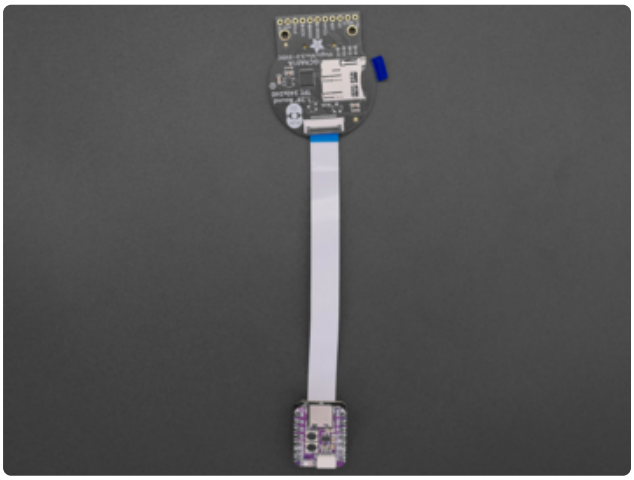
PLA filament 220c extruder
0.2 layer height
10% gyroid infill
200mm/s print speed
60c heated bed

Assemble



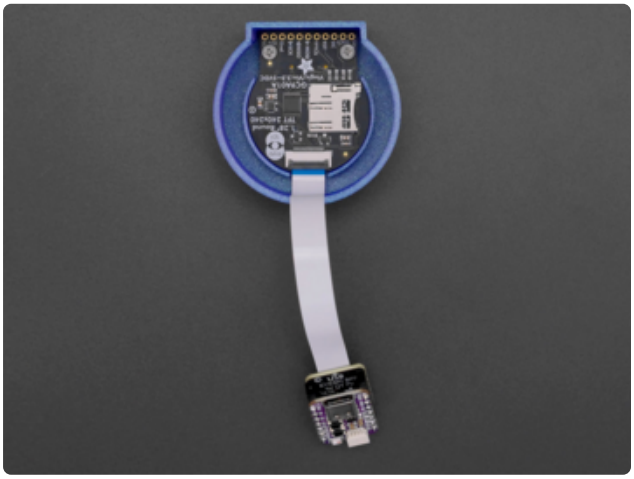
Join Headers

Solder short headers to the QT Py and EYESPI board to keep the board stack compact.



Connect Ribbon cable

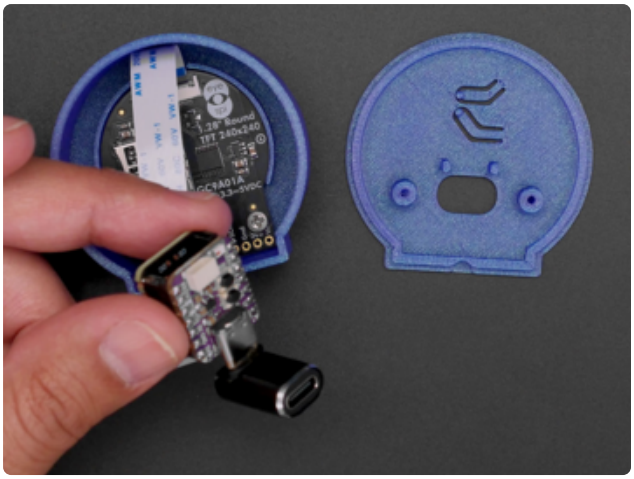
Use a 100mm long Flex PCB cable to join the two boards. Carefully lift the cable hinge and insert the cable with the blue backing facing down to make a connection to each board.

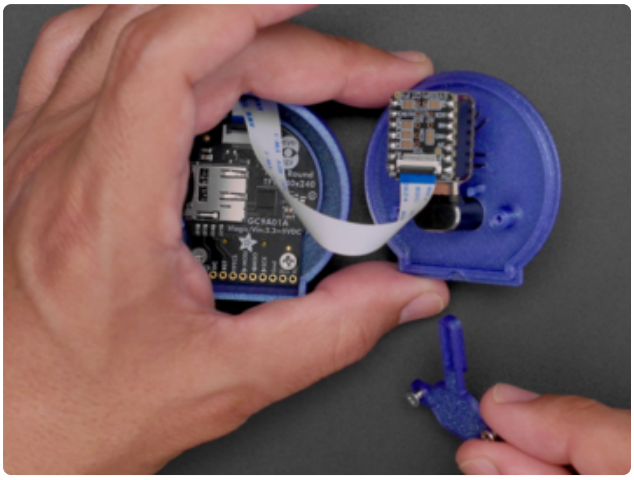


Place TFT display

Align the screw holes on the display to the standoffs inside the case and fasten with two M2.5x5mm screws.

Gently fold the ribbon cable along the display cutout inside the case.





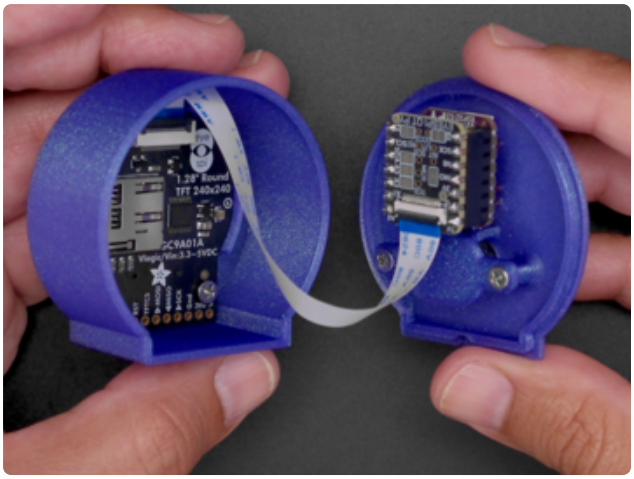
USB C Adapter

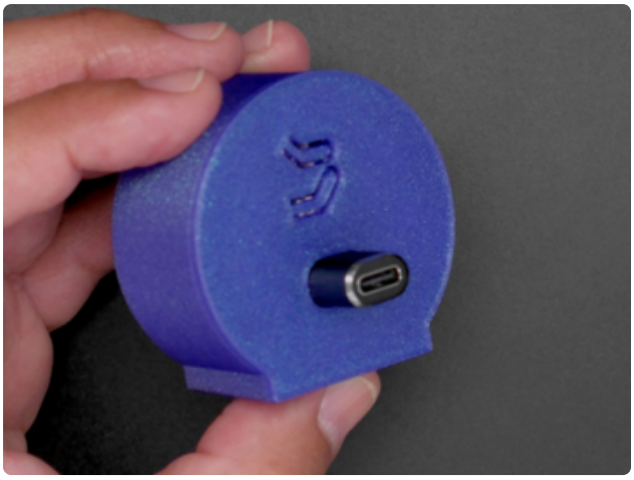
Connect the USB C adapter with the connector facing the QT Py buttons.

The adapter press fits through the cutout on the lid.

The printed USB holder plate is inserted between the QT Py and EYESPI board with the extruded rectangle facing down.

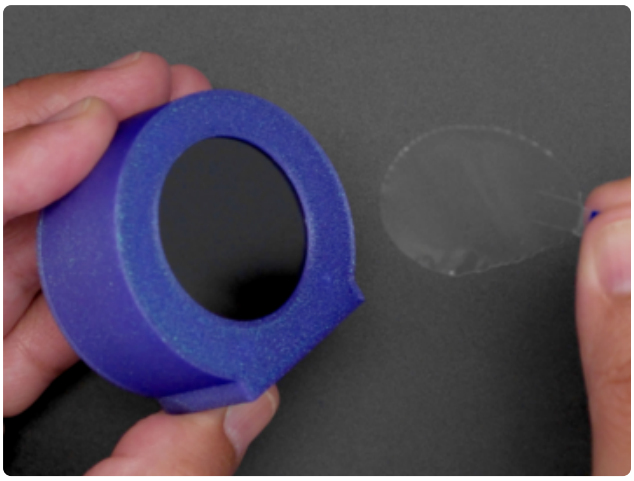
Secure the printed holder plate with two M2.5x5mm screws.





Align lid

Carefully fold the ribbon cable into the case. Align the cylindrical part of the case to the lid. Apply pressure to the square portion of the lid to press fit the two pieces together.



Remove protective film

Peel the protective film from the display if you haven't already!


2024 YR4
2032-Dec-22
Moon: 1921 mi
Earth: 161341 mi
Earth Impact: 0.0008%