Theme Park Wait Time Display

Created by Ruiz Brothers

https://learn.adafruit.com/park-wait-time

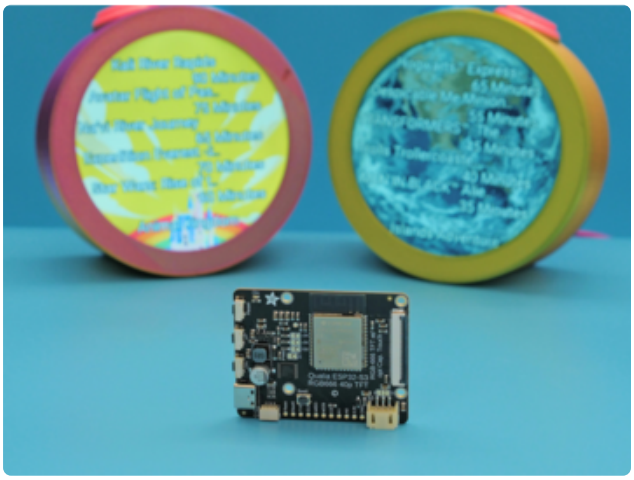
Last updated on 2024-11-18 01:00:46 PM EST

Table of Contents

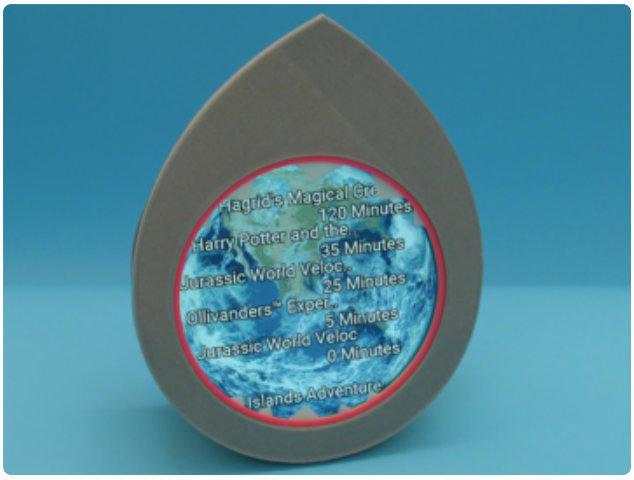
Overview

This project syncs with queue-times.com theme park queue lines to show the LIVE wait time of rides in any given moment!

This handy time display project will let you gauge crowd levels, even if you live thousands of miles away and didn't even have a trip planned!



The 4" round display is powered by the Qualia S3. CircuitPython code accesses queue-times.com API and displays the wait times on the screen, overlaid on top of a custom background image.

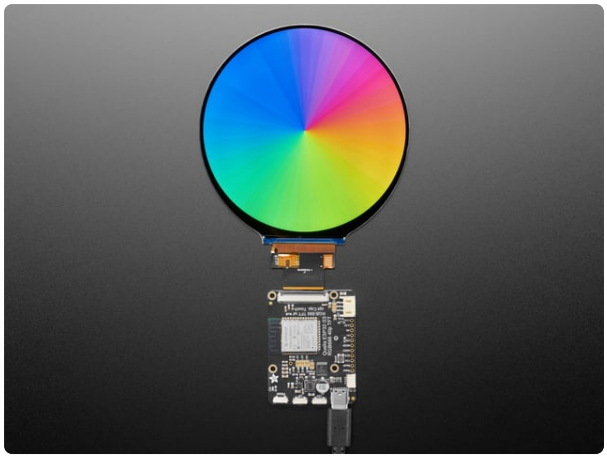


An arcade button lets you switch between different parks and show the top five rides with their wait times.

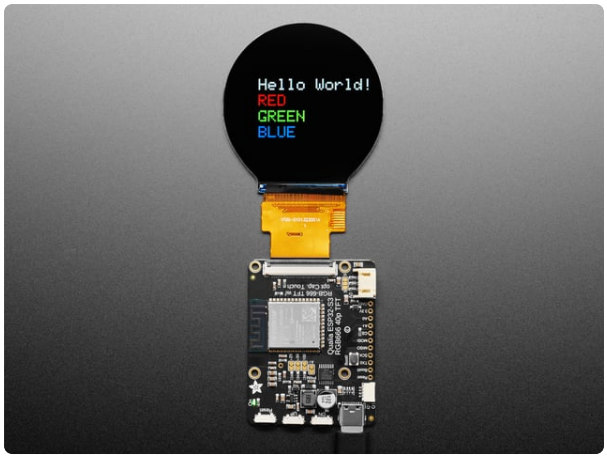
The 3D printed case houses the components, you can even customize the face plate to your preferred theme park!

Parts, Tools & Components

You'll need just a few parts to build this project. If you don't have access to a 3D printer, you can send the files to a service or check with your local hackerspace or library

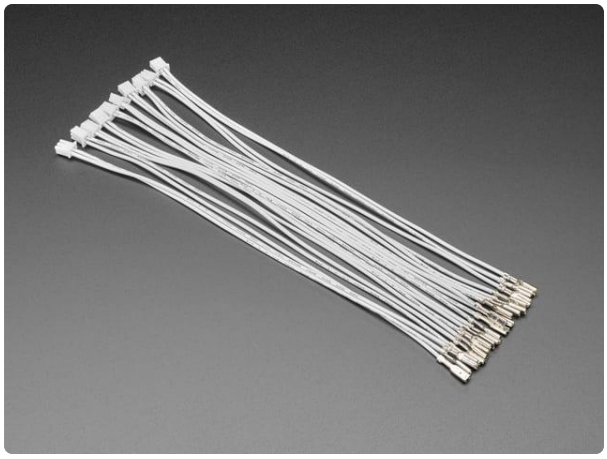


[Round RGB TTL TFT Display - 4" 720x720 - No Touchscreen]
This is a screen for advanced hackers who like the look of a nice, round TFT screen with tons of pixels. This massive 4" diagonal-sized display has 720x720 16-bit full-color...
https://www.adafruit.com/product/5793



[Adafruit Qualia ESP32-S3 for TTL RGB-666 Displays]
There's a few things everyone loves: ice cream, kittens, and honkin' large TFT screens. We're no strangers to small TFT's -
https://www.adafruit.com/product/5800

[Arcade Button - 30mm Translucent Clear]
A button is a button, and a switch is a switch, but these translucent arcade buttons are in a class of their own. They're the same size as common arcade controls (often referred to...
https://www.adafruit.com/product/471

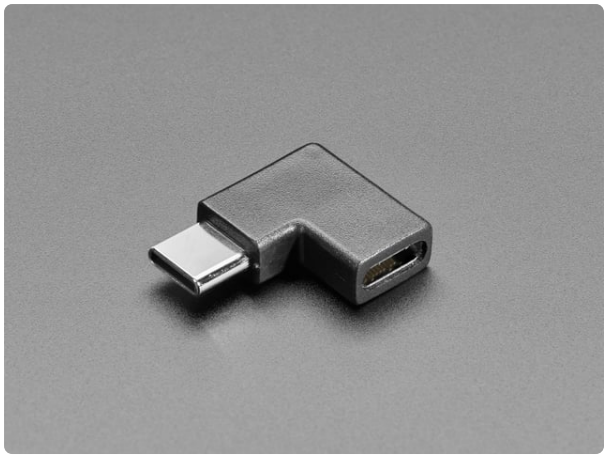


[Arcade Button Quick-Connect Wire Pairs - 0.11" (10 pack)]
Quick connector wire sets will make wiring up our arcade-style or metal buttons quicky-quick. Each wire comes as a 'pair' with two 0.11" quick-connects pre-crimped onto...
https://www.adafruit.com/product/1152

STEMMA JST PH 2mm 3-Pin to Male Header Cable - 200mm

This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" male header plugs on the end. We're carrying these to match up with our...

https://www.adafruit.com/product/3893



Right Angle USB Type C Adapter - USB 3.1 Gen 4 Compatible

As technology changes and adapts, so does Adafruit, and speaking of adapting, this right angle adapter is USB C...

https://www.adafruit.com/product/4432

Pink and Purple Woven USB A to USB C Cable - 1 meter long

This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also made for USB C for our modernized breakout boards, Feathers, and...

https://www.adafruit.com/product/5153

4 x M2.5mm Screws

https://amzn.to/3XKJc4v

M2.5mm Screws

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.

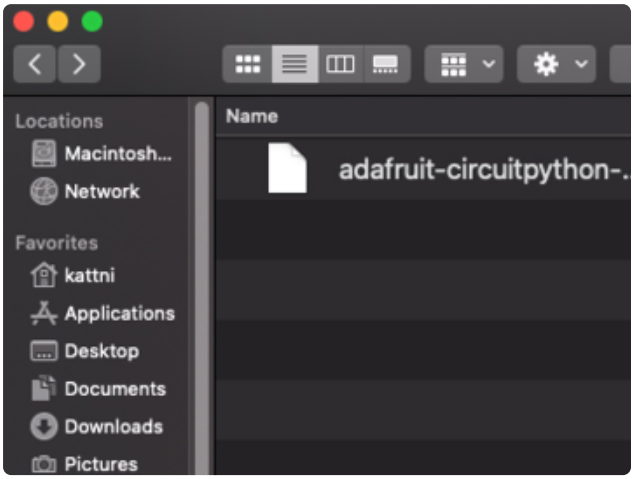
CircuitPython Quickstart

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

This microcontroller requires the latest **unstable (development)** release of CircuitPython. Click below to visit the downloads page on **circuitpython.org** for your board. Then, Browse **S3** under **Absolute Newest**.

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

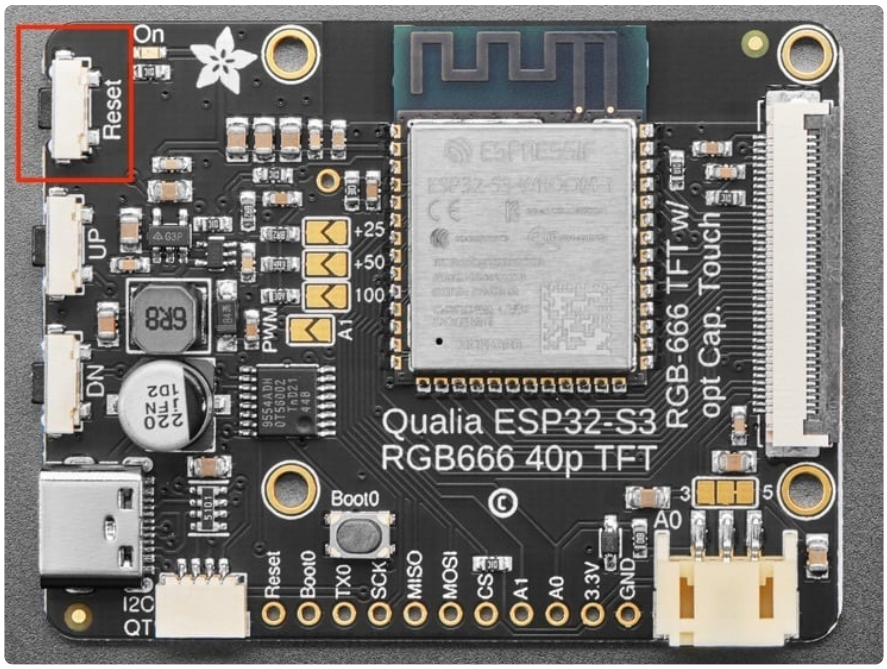
https://adafru.it/191D



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

Save it wherever is convenient for you.

The Qualia S3 does not have a RGB status LED



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

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

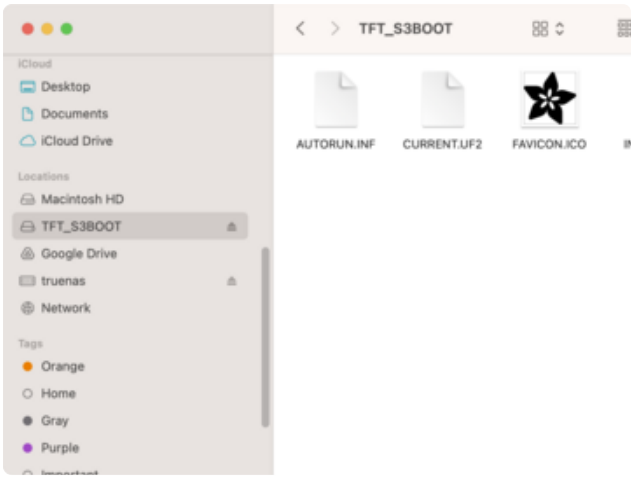
> This board does not have a Neopixel, so you will need to just double tap the reset button.

For this board, tap reset and wait about a half a second and then tap reset again.

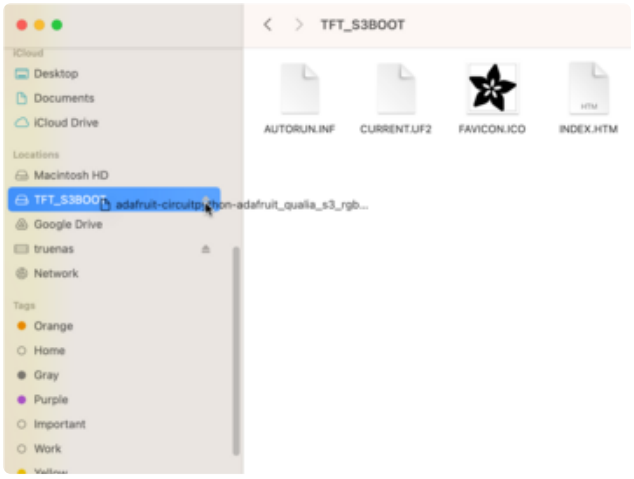
> Some boards may not have a UF2 bootloader installed. If double-clicking does not work, follow the instructions on the "Install UF2 Bootloader" page in this guide.

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

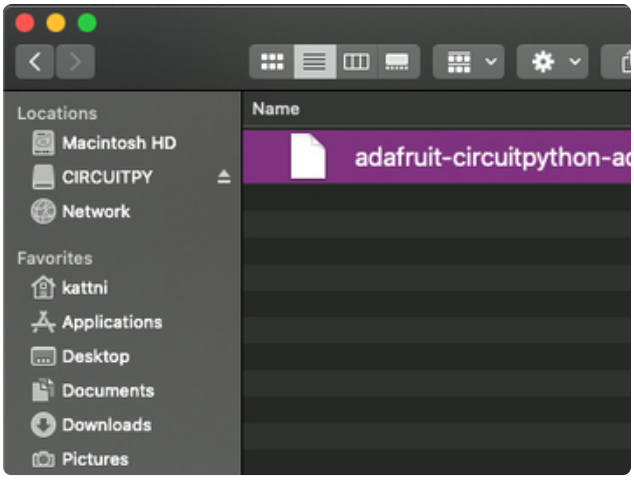
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **TFT_S3BOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **TFT_S3BOOT**.



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

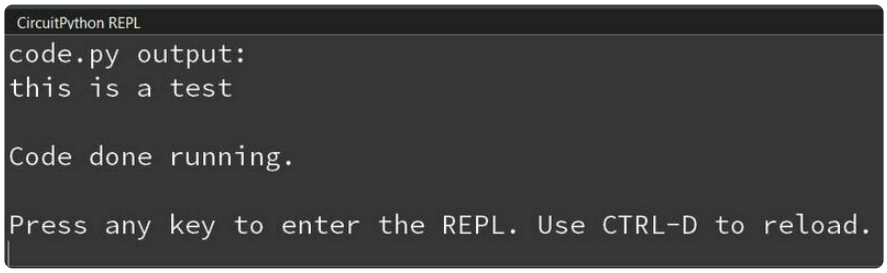
When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

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 the Display

Once you've finished setting up your Qualia S3 with CircuitPython, you can access the code 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: 2024 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import ssl
import time
import microcontroller
import wifi
import socketpool
import adafruit_requests
import board
import displayio
import keypad
from adafruit_ticks import ticks_ms, ticks_add, ticks_diff
from adafruit_display_text import outlined_label
from adafruit_bitmap_font import bitmap_font
from adafruit_qualia.graphics import Graphics, Displays

urls = [{'name': "Epcot",
         'url': "https://queue-times.com/en-US/parks/5/queue_times.json"},
        {'name': "Magic Kingdom",
         'url': "https://queue-times.com/en-US/parks/6/queue_times.json"},
        {'name': "Hollywood Studios",
         'url': "https://queue-times.com/en-US/parks/7/queue_times.json"},
        {'name': "Animal Kingdom",
         'url': "https://queue-times.com/en-US/parks/8/queue_times.json"},
        ]
bitmap = displayio.OnDiskBitmap("/park-bg.bmp")

key = keypad.Keys((board.A0,), value_when_pressed=False, pull=True)

wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}")

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

graphics = Graphics(Displays.ROUND40, default_bg=None, auto_refresh=True)

grid = displayio.TileGrid(bitmap, pixel_shader=bitmap.pixel_shader)
group = displayio.Group()
group.append(grid)
```

```
font = bitmap_font.load_font("/Roboto-Regular-47.pcf")
ride_text = []
wait_text = []
for i in range(5):
    text_ride = outlined_label.OutlinedLabel(font, text=" ",
    outline_color=0x000000,
    outline_size=3,
    padding_left=4,
    padding_right=4,
    padding_top=4,
    padding_bottom=4)
    ride_text.append(text_ride)
    text_wait = outlined_label.OutlinedLabel(font, text=" ",
    outline_color=0x000000,
    outline_size=3,
    padding_left=4,
    padding_right=4,
    padding_top=4,
    padding_bottom=4)
    wait_text.append(text_wait)
    group.append(text_ride)
    group.append(text_wait)

text_park = outlined_label.OutlinedLabel(font, text=urls[0]['name'],
    outline_color = 0x000000,
    outline_size=3,
    padding_left=4,
    padding_right=4,
    padding_top=4,
    padding_bottom=4)
text_park.x = (graphics.display.width - text_park.width) // 2
text_park.y = graphics.display.height - (text_park.height * 2)
group.append(text_park)

def center(g, b):
    # center the image
    g.x -= (b.width - graphics.display.width) // 2
    g.y -= (b.height - graphics.display.height) // 2

center(grid, bitmap)

graphics.display.root_group = group

def sort_rides(data):
    y = 30
    x = [135, 55, 15, 25, 45]
    all_rides = []
    for land in data['lands']:
        all_rides.extend(land['rides'])
    sorted_rides = sorted(all_rides, key=lambda x: x['wait_time'], reverse=True)
    for ride in sorted_rides:
        r = sorted_rides.index(ride)
        if r > 4:
            break
        #print(wait_text[r])
        ride_text[r].text = f"{ride['name']:.20}"
        if len(ride['name']) > 20:
            ride_text[r].text = ride_text[r].text + ".."
        ride_text[r].x = x[r]
        ride_text[r].y = y + 70
        wait_text[r].text = f"{ride['wait_time']} Minutes"
        wait_text[r].x = 400
        wait_text[r].y = ride_text[r].y + wait_text[r].height + 20
        y += wait_text[r].height * 2 + 30

clock_timer = 5 * 60 * 1000
clock_clock = ticks_ms()
park_index = 0
update = True
```

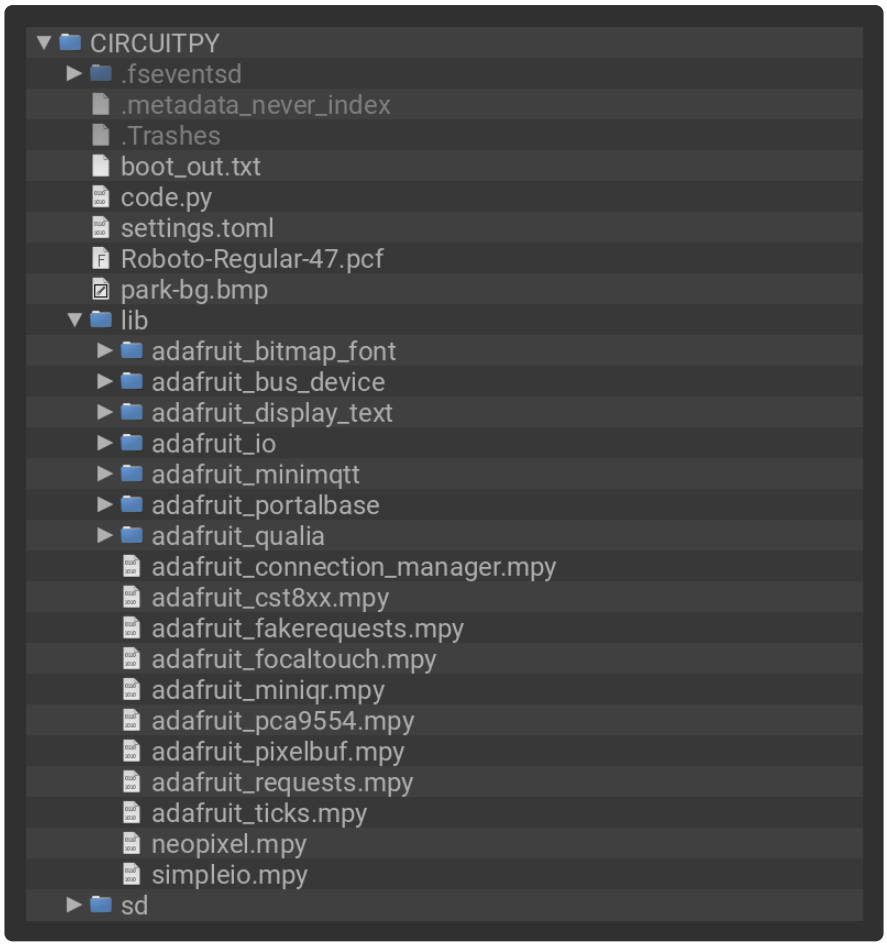
```
while True:
    try:
        event = key.events.get()
        if event:
            if event.pressed:
                print("updating display")
                park_index = (park_index + 1) % len(urls)
                text_park.text=urls[park_index]['name']
                text_park.x = (graphics.display.width - text_park.width) // 2
                text_park.y = graphics.display.height - (text_park.height * 2)
                update = True
        if ticks_diff(ticks_ms(), clock_clock) >= clock_timer or update:
            response = requests.get(urls[park_index]['url'])
            #  packs the response into a JSON
            response_data = response.json()
            sort_rides(response_data)
            update = False
            clock_clock = ticks_add(clock_clock, clock_timer)
    except Exception as error: # pylint: disable=broad-except
        print(f"error! {error} resetting..")
        time.sleep(5)
        microcontroller.reset()
```

Upload the Code and Libraries to the Qualia S3

After downloading the Project Bundle, plug your Qualia S3 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 Qualia S3's **CIRCUITPY** drive.

- **lib** folder
- **code.py**
- **park-bg.bmp**
- **Roboto-Regular-47.pcf**

Your Qualia S3 **CIRCUITPY** drive should look like this after copying the **lib** folder, **park-bg.bmp** file, **Roboto-Regular-47.pcf** 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/19ap) earlier in this guide. You'll need to include your `CIRCUITPY_WIFI_SSID` and `CIRCUITPY_WIFI_PASSWORD`.

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

How the CircuitPython Code Works

At the top of the code is a dictionary with the name of the park and its Queue Times API JSON feed. The text in the `name` entry will be shown on the display. You can update this dictionary with the names of the parks that you want to track. The code will dynamically adjust to a different number of feeds.

```
urls = [{'name': "Epcot", 'url': "https://queue-times.com/en-US/parks/5/
queue_times.json"},
        {'name': "Magic Kingdom", 'url': "https://queue-times.com/en-US/parks/6/
queue_times.json"},
        {'name': "Hollywood Studios", 'url': "https://queue-times.com/en-US/parks/7/
```

```
queue_times.json"},
        {'name': "Animal Kingdom", 'url': "https://queue-times.com/en-US/parks/8/
queue_times.json"},
        ]
```

Bitmap, Keypad and WiFi

The background image is loaded in as an `OnDiskBitmap` and the button that changes the park view is instantiated as a `Keypad` object on pin **A0**. A WiFi connection is established using your SSID and password entries from **settings.toml**.

```
bitmap = displayio.OnDiskBitmap("/park-bg.bmp")

key = keypad.Keys((board.A0,), value_when_pressed=False, pull=True)

wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}")

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

Graphics

Ten different labels are used to display the ride name and ride wait time. These are created with a `for` loop and are added to the display `Group`. The ride text elements are added to the `ride_text` list and the wait time text elements are added to the `wait_text` list.

One text label is created for the park name that is stored in the `urls` dictionary. This label is located at the bottom of the display.

```
graphics = Graphics(Displays.ROUND40, default_bg=None, auto_refresh=True)

grid = displayio.TileGrid(bitmap, pixel_shader=bitmap.pixel_shader)
group = displayio.Group()
group.append(grid)

font = bitmap_font.load_font("/Roboto-Regular-47.pcf")
ride_text = []
wait_text = []
for i in range(5):
    text_ride = outlined_label.OutlinedLabel(font, text=" ",
    outline_color=0x000000,
    outline_size=3,
    padding_left=4,
    padding_right=4,
    padding_top=4,
    padding_bottom=4)
    ride_text.append(text_ride)
    text_wait = outlined_label.OutlinedLabel(font, text=" ",
    outline_color=0x000000,
    outline_size=3,
    padding_left=4,
    padding_right=4,
    padding_top=4,
    padding_bottom=4)
    wait_text.append(text_wait)
```

```
    group.append(text_ride)
    group.append(text_wait)

text_park = outlined_label.OutlinedLabel(font, text=urls[0]['name'],
    outline_color = 0x000000,
    outline_size=3,
    padding_left=4,
    padding_right=4,
    padding_top=4,
    padding_bottom=4)
text_park.x = (graphics.display.width - text_park.width) // 2
text_park.y = graphics.display.height - (text_park.height * 2)
group.append(text_park)
```

Sorting

A function is used in the loop that sorts the returned list of rides by their wait time. The five rides with the longest wait times are shown on the display by updating the text elements in `ride_text` and `wait_text`.

```
def sort_rides(data):
    y = 30
    x = [135, 55, 15, 25, 45]
    all_rides = []
    for land in data['lands']:
        all_rides.extend(land['rides'])
    sorted_rides = sorted(all_rides, key=lambda x: x['wait_time'], reverse=True)
    for ride in sorted_rides:
        r = sorted_rides.index(ride)
        if r &gt; 4:
            break
        #print(wait_text[r])
        ride_text[r].text = f"{ride['name']:.20}"
        if len(ride['name']) &gt; 20:
            ride_text[r].text = ride_text[r].text + ".."
        ride_text[r].x = x[r]
        ride_text[r].y = y + 70
        wait_text[r].text = f"{ride['wait_time']} Minutes"
        wait_text[r].x = 400
        wait_text[r].y = ride_text[r].y + wait_text[r].height + 20
        y += wait_text[r].height * 2 + 30
```

Times and States

A `ticks` timer, set for five minutes, is used to keep time in the loop. `park_index` tracks which park is selected from the `urls` dictionary and the `update` state tracks whether the JSON feed should be polled.

```
clock_timer = 5 * 60 * 1000
clock_clock = ticks_ms()
park_index = 0
update = True
```

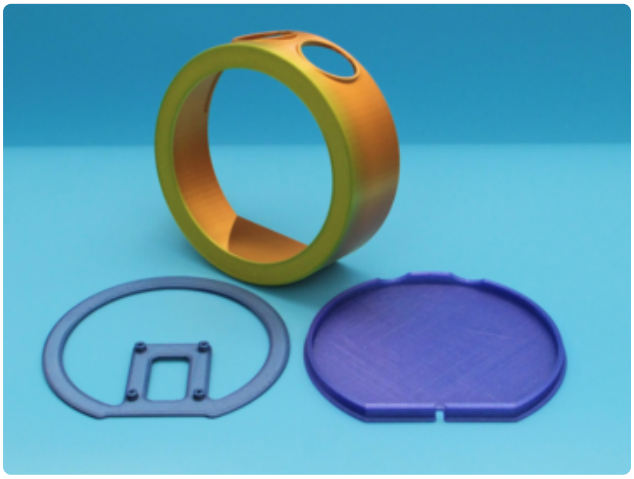
The Loop

In the loop, if the button is pressed then the `park_index` increases by one, resulting in the park text updating on the display and `update` being set to `True`. If `update` is `True` or if the `ticks` timer runs out, the JSON feed for the selected park is fetched.

The `sort_rides()` function sorts the JSON feed by ride wait time and updates the text elements for the top five rides and their wait times.

```
while True:
    try:
        event = key.events.get()
        if event:
            if event.pressed:
                print("updating display")
                park_index = (park_index + 1) % len(urls)
                text_park.text=urls[park_index]['name']
                text_park.x = (graphics.display.width - text_park.width) // 2
                text_park.y = graphics.display.height - (text_park.height * 2)
                update = True
        if ticks_diff(ticks_ms(), clock_clock) &gt;= clock_timer or update:
            response = requests.get(urls[park_index]['url'])
            #  packs the response into a JSON
            response_data = response.json()
            sort_rides(response_data)
            update = False
            clock_clock = ticks_add(clock_clock, clock_timer)
    except Exception as error:
        print(f"error! {error} resetting..")
        time.sleep(5)
        microcontroller.reset()
```

3D Printing



3D Printed Parts

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

Parts were tested using PLA filament.

Original design source files may be downloaded using the links below

Edit Design

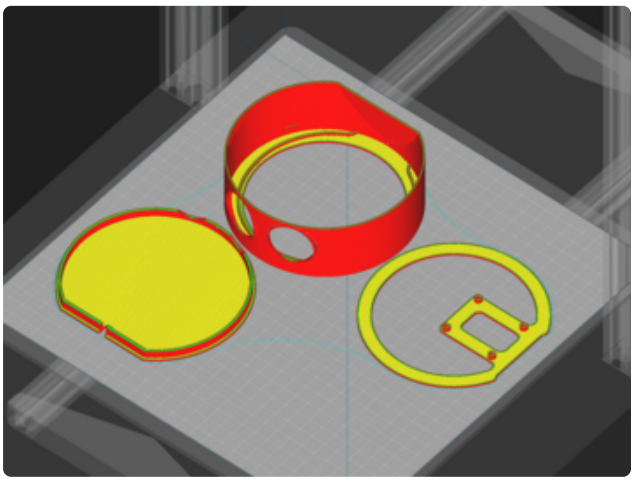
https://adafru.it/1a5h

Download STLs

https://adafru.it/1a5k

Download STEP file

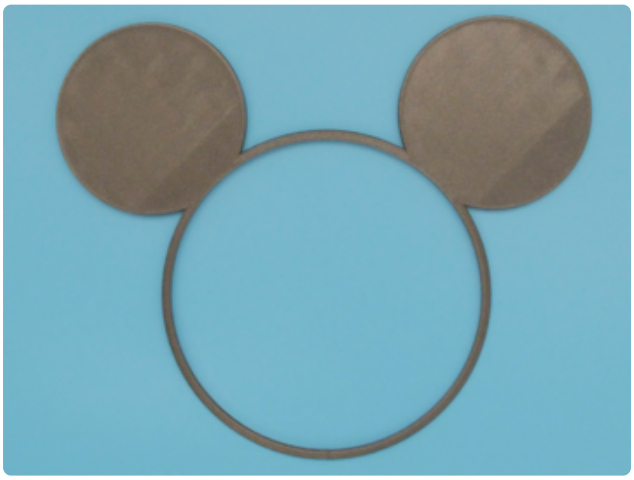
https://adafru.it/1a5l

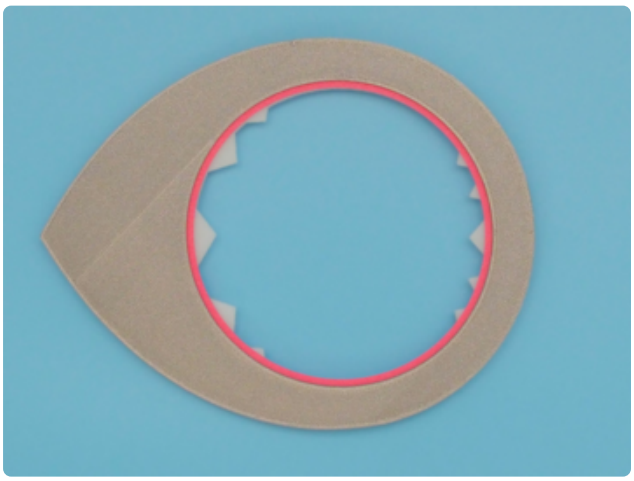


Slice with Settings for PLA material

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

PLA filament 200c extruder
0.25 layer height
20% gyroid infill
60mm/s print speed
60°C heated bed

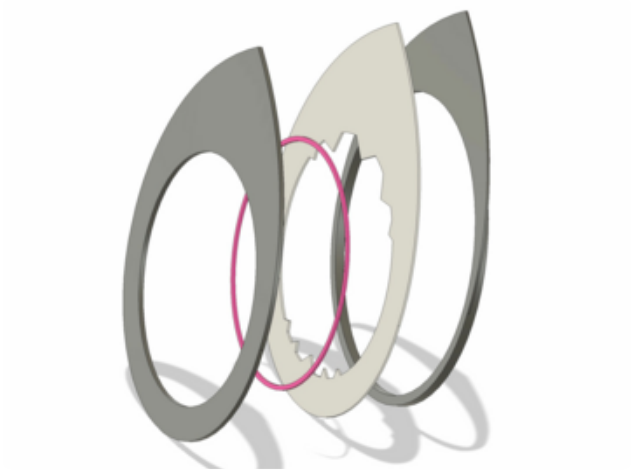


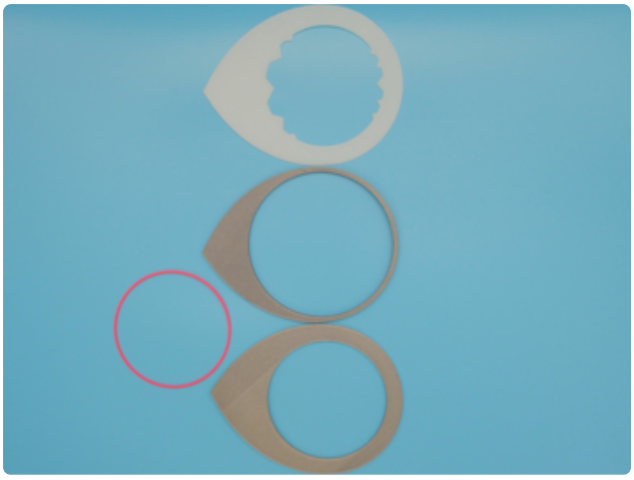


Face plates

You can customize the front face plates for the case to match for preferred theme part. The design includes mouse ears and a shark!

The face plates press fit onto the front of the case over the display.





Multicolor face plate

The shark face plate is designed to be stacked and glued for single extruder printers.

Theme_Park_Wait_Times_Clock-STEP.zip

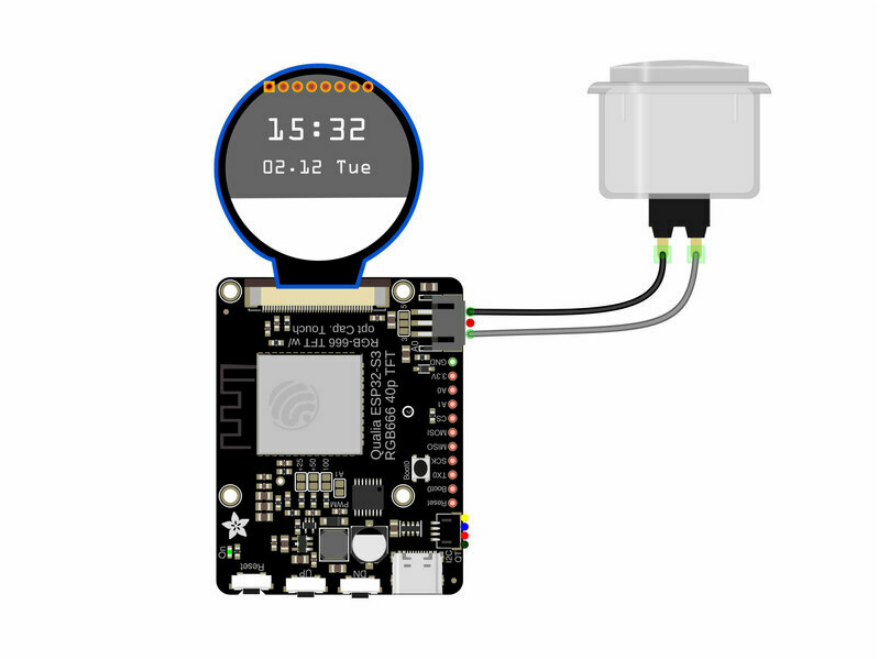
https://adafru.it/1a5m

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).



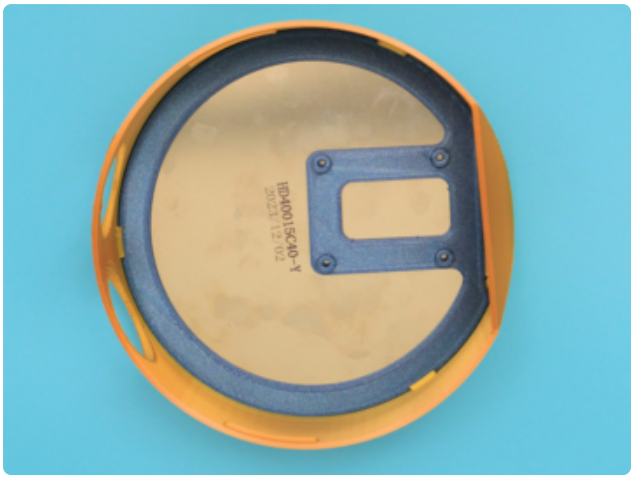
Assemble

Place screen

The 4" round display is placed inside the case. Gently press fit the display into the inner walls.

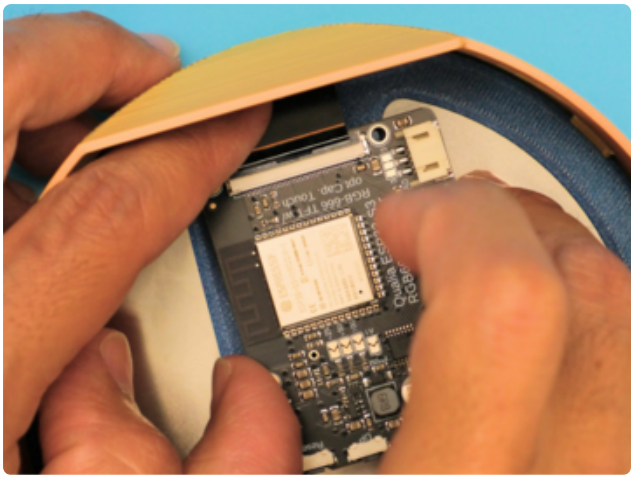
Snap fit frame

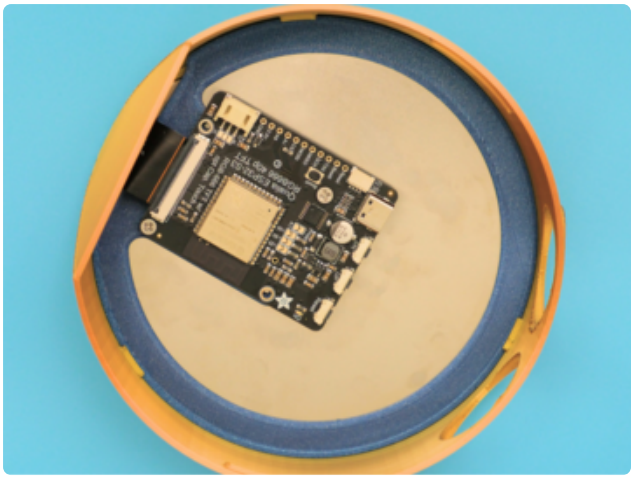
Place the frame on top of the display. Three nubs around the inside of the case hold the frame in place. Gently bend the frame to fit over the nubs to snap fit to the case.



Adjust frame

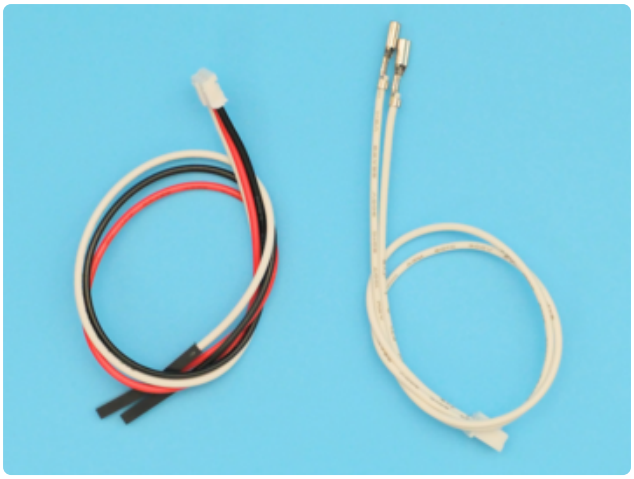
The frame can freely rotate to align the display ribbon cable to the Qualia board.





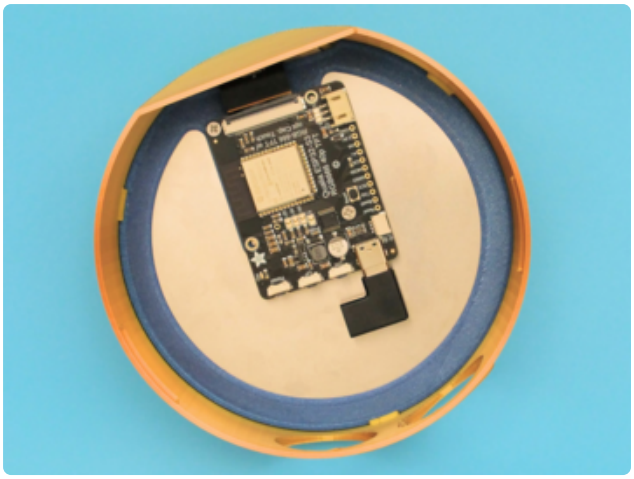
Attach ribbon cable

Open the display latch on the Qualia board. Place the Qualia board over the stand-offs. Use M2.5x5mm screws to mount the board.



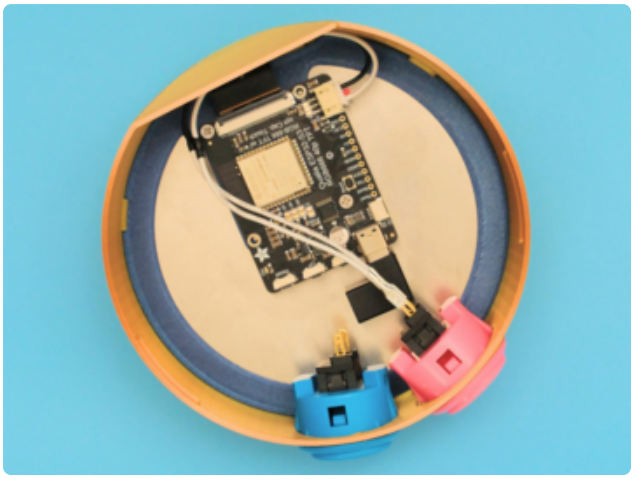
Assemble wires

Solder quick connect wires to a cable with a JST-3PH connector.



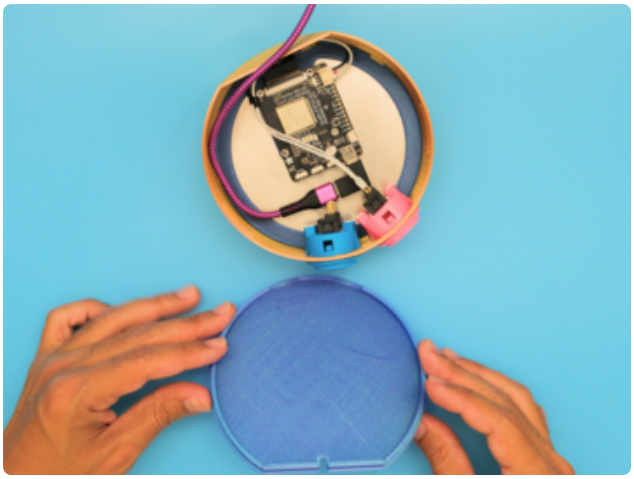
USB right angle

Plug in a USB C right angle adapter to easily connect a USB cable.



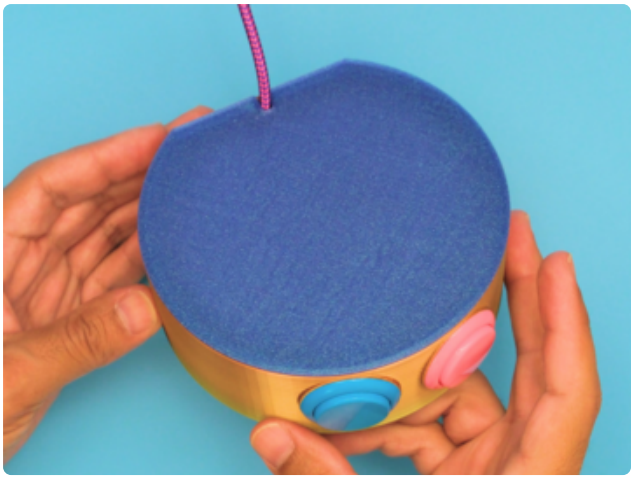
Arcade buttons

Press fit arcade buttons into to the case. Connect the quick connects to the terminals on the arcade buttons.



USB Cable

Plug in the USB cable to the Qualia board.



Snap fit lid

The USB cable fits into the cutout on the lid part. Align the lid to the snap fit nubs on the case and press fit to the case.

Complete