



Touch Deck: DIY Customizable TFT Control Pad

Created by John Park

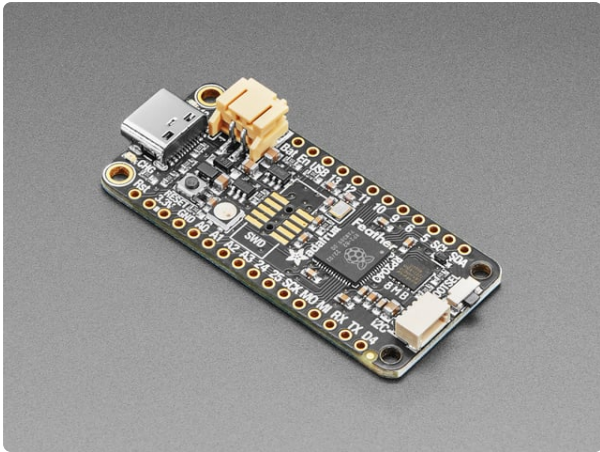


<https://learn.adafruit.com/touch-deck-diy-tft-customized-control-pad>

Last updated on 2026-02-03 10:56:30 PM UTC

Table of Contents

Overview	3
<ul style="list-style-type: none">• Parts and Materials• Other Parts• Optional	
Install CircuitPython	6
<ul style="list-style-type: none">• CircuitPython Quickstart• Safe Mode• Flash Resetting UF2	
Installing Libraries	10
Installing the Mu Editor	10
<ul style="list-style-type: none">• Download and Install Mu• Starting Up Mu• Using Mu	
Code the Touch Deck	13
<ul style="list-style-type: none">• Text Editor• CircuitPython Code• Project Files• Testing• How it Works• Customizing the Layers File• Full Layers File	
Assemble the Touch Deck	25
<ul style="list-style-type: none">• Feather + Wing• Case Top• Top Screws• Standoffs• Assemble• USB Plug• Use the Touch Deck• Customize	



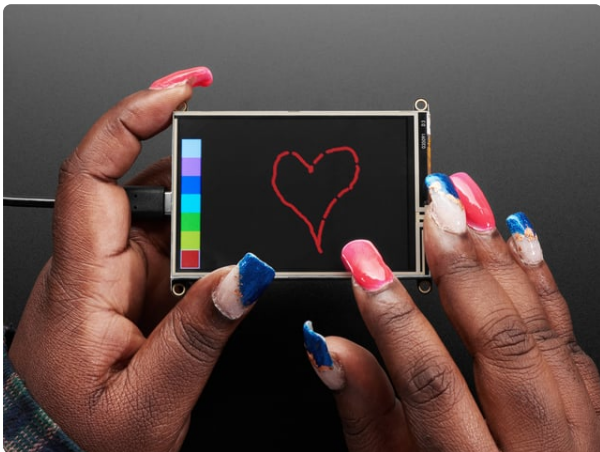
Adafruit Feather RP2040

A new chip means a new Feather, and the Raspberry Pi RP2040 is no exception. When we saw this chip we thought "this chip is going to be awesome when we give it the Feather..."

<https://www.adafruit.com/product/4884>



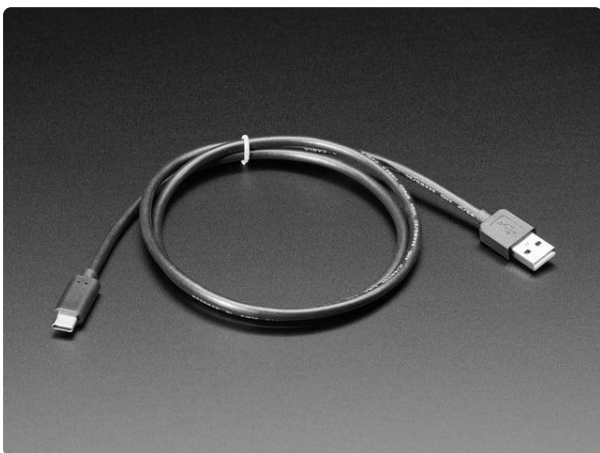
should be able to use Feather M4 and Feather nRF52840 boards with this ect as well, but currently it's been tested only on the Feather RP2040.



Adafruit TFT FeatherWing - 3.5" 480x320 Touchscreen for Feathers

Spice up your Feather project with a beautiful 3.5" touchscreen display shield with built in microSD card socket. This TFT display is 3.5" diagonal with a bright 6 white-LED...

<https://www.adafruit.com/product/3651>



USB Type A to Type C Cable - approx 1 meter / 3 ft long

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

<https://www.adafruit.com/product/4474>

or



[USB C to USB C Cable - USB 3.1 Gen 4 with E-Mark - 1 meter long](https://www.adafruit.com/product/4199)

As technology changes and adapts, so does Adafruit! Rather than the regular USB A, this cable has USB C to USB C plugs! USB C is the latest...

<https://www.adafruit.com/product/4199>

Other Parts

You'll need access to a 3D Printer + filament to print the Touch Deck case. For the case, you'll also need some fasteners and rubber feet.



[Black Nylon Machine Screw and Stand-off Set – M2.5 Thread](https://www.adafruit.com/product/3299)

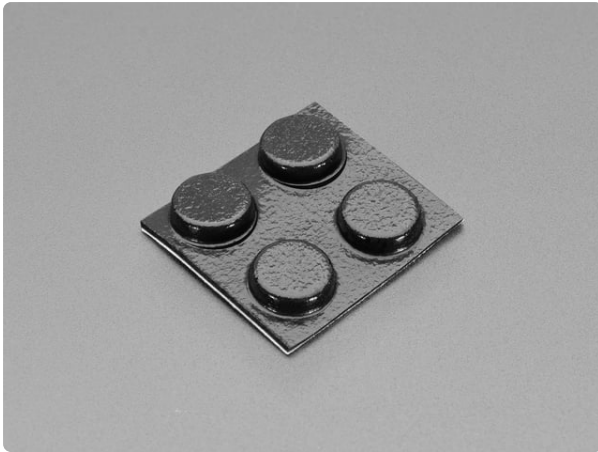
Totalling 380 pieces, this M2.5 Screw Set is a must-have for your workstation. You'll have enough screws, nuts, and hex standoffs to fuel your maker...

<https://www.adafruit.com/product/3299>



M2.5 x 16mm Screws

You'll need four longer screws for the top of the case. I [used these \(https://adafru.it/Ria\)](https://adafru.it/Ria) socket head screws for that stylish look, but any style you like will work.

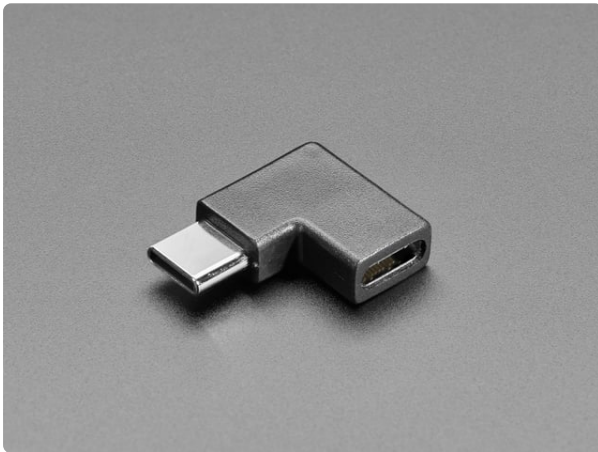


[Little Rubber Bumper Feet - Pack of 4](https://www.adafruit.com/product/550)

Keep your electronics from going barefoot, give them little rubber feet! These small sticky bumpers are our favorite accessory for any electronic kit or device. They are sticky, but...

<https://www.adafruit.com/product/550>

Optional



[Right Angle USB Type C Adapter - USB 3.1 Gen 4 Compatible](https://www.adafruit.com/product/4432)

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>

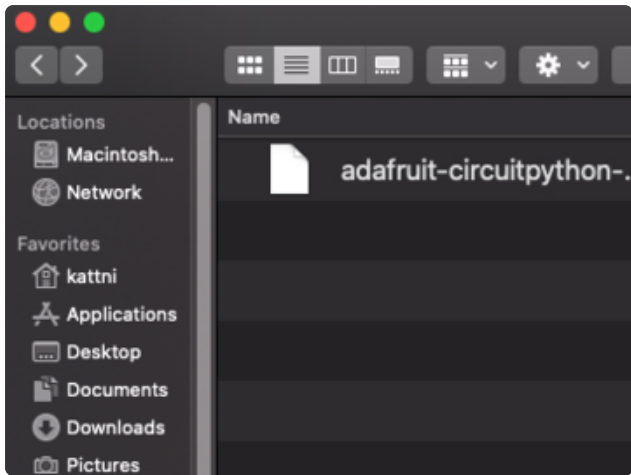
Install CircuitPython

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

CircuitPython Quickstart

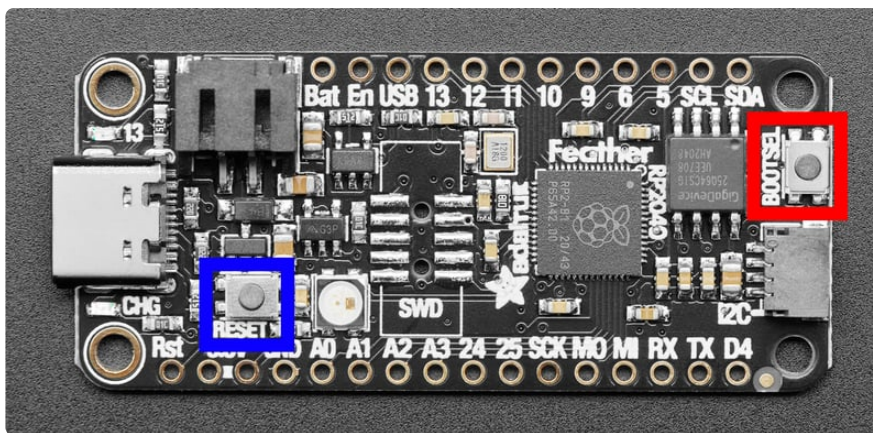
Follow this step-by-step to quickly get CircuitPython running on your board.

<https://adafru.it/R1D>



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

Save it wherever is convenient for you.

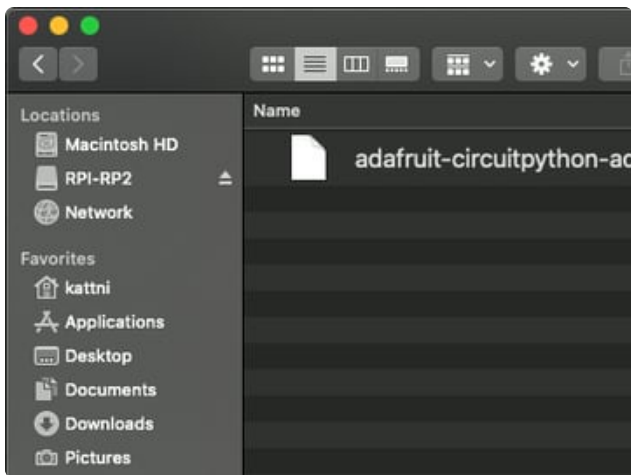


To enter the bootloader, hold down the **BOOT/BOOTSEL** button (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset** button (highlighted in red or blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

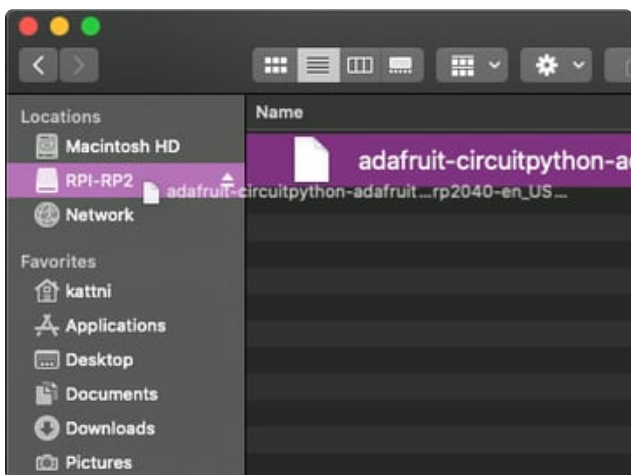
If the drive does not appear, release all the buttons, and then repeat the process above.

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

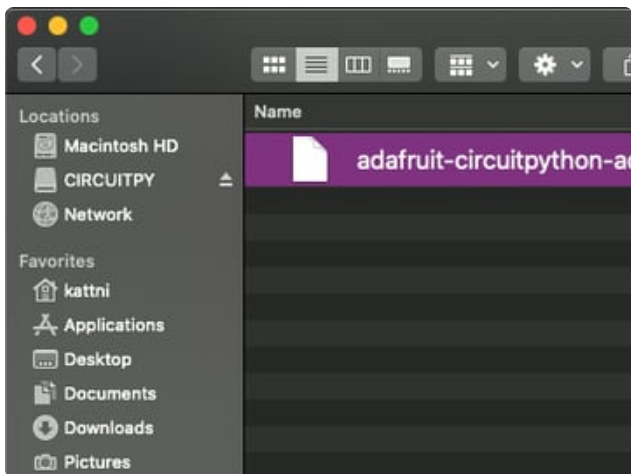
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



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



Drag the `adafruit_circuitpython_etc.uf2` file to **RPI-RP2**.



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

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

Safe Mode

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

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

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

Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

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

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

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

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

Flash Resetting UF2

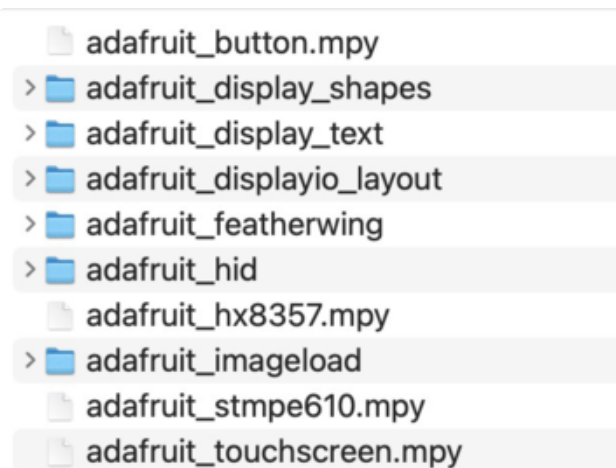
If your board ever gets into a really weird state and CIRCUITPY doesn't show up as a disk drive after installing CircuitPython, try loading this 'nuke' UF2 to RPI-RP2. which will do a 'deep clean' on your Flash Memory. **You will lose all the files on the board**, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

<https://adafru.it/RLE>

Installing Libraries

Alongside the core CircuitPython libraries (which are baked into CircuitPython), you'll also add a number of libraries to make the Touch Deck work.

Download the [library bundle \(https://adafru.it/ENC\)](https://adafru.it/ENC) here.



Copy the following library files and folders from the bundle to the **lib** folder on your **CIRCUITPY** drive:

adafruit_button
adafruit_display_shapes
adafruit_display_text
adafruit_displayio_layout
adafruit_featherwing
adafruit_hid
adafruit_hx8357
adafruit_imageload
adafruit_stmpe610
adafruit_touchscreen

Installing the Mu Editor

Mu is a simple code editor that works with the Adafruit CircuitPython boards. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial

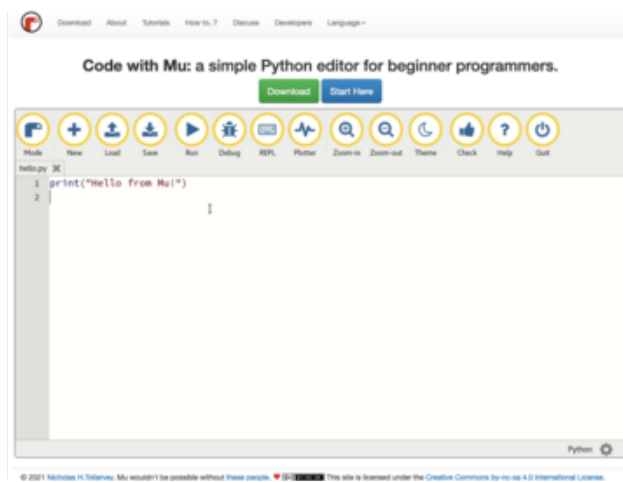
console is built right in so you get immediate feedback from your board's serial output!



Mu is our recommended editor - please use it (unless you are an experienced user with a favorite editor already!). While it has been announced end of life in 2026, it still works fine.

You are free to use whatever text editor you wish along with a terminal program to connect to the CircuitPython REPL. Thonny is one such editor.

Download and Install Mu



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

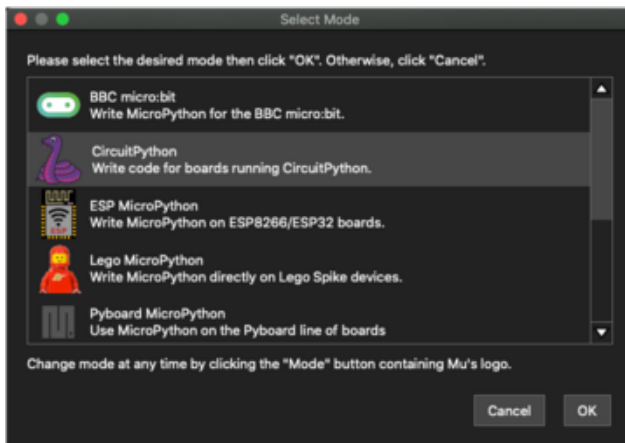
Click the **Download** link for downloads and installation instructions.

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



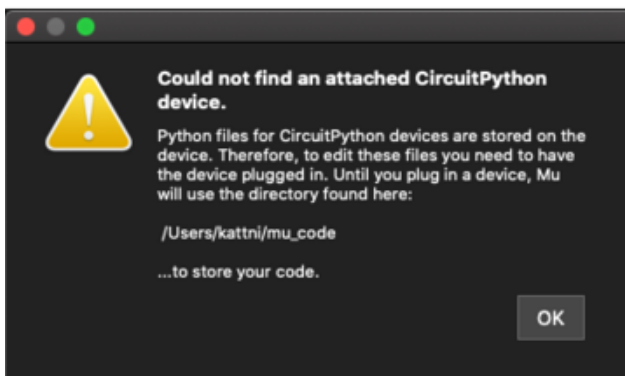
Windows users: due to the nature of MSI installers, please remove old versions of Mu before installing the latest version.

Starting Up Mu



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

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



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

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

Using Mu

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



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

Code the Touch Deck

Text Editor

Adafruit recommends using the Mu editor for using your CircuitPython code with the Feather. You can get more info in [this guide \(https://adafru.it/ANO\)](https://adafru.it/ANO).

Alternatively, you can use any text editor that saves files.

CircuitPython Code

Copy the code below and paste it into Mu. Then, save it to your Feather RP2040's CIRCUITPY drive as **code.py**.

```
boot_out.txt
code.py
> lib
> touch_deck_icons
touch_deck_layers.py
```

You'll need to download the .zip file in the GitHub link below and decompress the file to get all of the **touch_deck_icons** directory that's full of sample icons, as well as the **touch_deck_layers.py** file. Copy both of these to your Feather's CIRCUITPY drive.

Project Files

Download the project files by clicking the Download Project Bundle link below and copy them onto your **CIRCUITPY** drive.

```
# SPDX-FileCopyrightText: 2020 Tim C, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
This version runs on Feather RP2040 with a 3.5" FeatherWing
"""

import time
import displayio
import terminalio
from adafruit_display_text import bitmap_label
from adafruit_displayio_layout.layouts.grid_layout import GridLayout
from adafruit_displayio_layout.widgets.icon_widget import IconWidget
from adafruit_featherwing import tft_featherwing_35
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS
from adafruit_hid.consumer_control import ConsumerControl
from touch_deck_layers import (
    touch_deck_config,
    KEY,
    STRING,
    MEDIA,
    KEY_PRESS,
    KEY_RELEASE,
    CHANGE_LAYER,
)

# seems to help the touchscreen not get stuck with chip not found
time.sleep(3)

# display and touchscreen initialization
displayio.release_displays()
tft_featherwing = tft_featherwing_35.TFTFeatherWing35()
display = tft_featherwing.display
touchscreen = tft_featherwing.touchscreen

# HID setup
kbd = Keyboard(usb_hid.devices)
cc = ConsumerControl(usb_hid.devices)
kbd_layout = KeyboardLayoutUS(kbd)

# variables to enforce timeout between icon presses
COOLDOWN_TIME = 0.5
LAST_PRESS_TIME = -1

# 'mock' icon indexes for the layer buttons
# used for debouncing
PREV_LAYER_INDEX = -1
NEXT_LAYER_INDEX = -2
HOME_LAYER_INDEX = -3

# start on first layer
current_layer = 0

# Make the main_group to hold everything
main_group = displayio.Group()
display.root_group = main_group

# loading screen
loading_group = displayio.Group()
```

```

# black background, screen size minus side buttons
loading_background = displayio.Bitmap(
    (display.width - 40) // 20, display.height // 20, 1
)
loading_palette = displayio.Palette(1)
loading_palette[0] = 0x0

# scaled group to match screen size minus side buttons
loading_background_scale_group = displayio.Group(scale=20)
loading_background_tilegrid = displayio.TileGrid(
    loading_background, pixel_shader=loading_palette
)
loading_background_scale_group.append(loading_background_tilegrid)

# loading screen label
loading_label = bitmap_label.Label(terminalio.FONT, text="Loading...", scale=3)
loading_label.anchor_point = (0.5, 0.5)
loading_label.anchored_position = (display.width // 2, display.height // 2)

# append background and label to the group
loading_group.append(loading_background_scale_group)
loading_group.append(loading_label)

# GridLayout to hold the icons
# size and location can be adjusted to fit
# different sized screens.
layout = GridLayout(
    x=20,
    y=20,
    width=420,
    height=290,
    grid_size=(4, 3),
    cell_padding=6,
)

# list that holds the IconWidget objects for each icon.
_icons = []

# list that holds indexes of currently pressed icons and layer buttons
# used for debouncing
_pressed_icons = []

# layer label at the top of the screen
layer_label = bitmap_label.Label(terminalio.FONT)
layer_label.anchor_point = (0.5, 0.0)
layer_label.anchored_position = (display.width // 2, 4)
main_group.append(layer_label)

# right side layer buttons
next_layer_btn = IconWidget("", "touch_deck_icons/layer_next.bmp", on_disk=True)
next_layer_btn.x = display.width - 40
next_layer_btn.y = display.height - 100
next_layer_btn.resize = (40, 100)
main_group.append(next_layer_btn)

prev_layer_btn = IconWidget("", "touch_deck_icons/layer_prev.bmp", on_disk=True)
prev_layer_btn.x = display.width - 40
prev_layer_btn.y = 110
prev_layer_btn.resize = (40, 100)
main_group.append(prev_layer_btn)

home_layer_btn = IconWidget("", "touch_deck_icons/layer_home.bmp", on_disk=True)
home_layer_btn.x = display.width - 40
home_layer_btn.y = 0
home_layer_btn.resize = (40, 100)
main_group.append(home_layer_btn)

```

```

# helper method to load icons for an index by its index in the
# list of layers
def load_layer(layer_index):
    # show the loading screen
    main_group.append(loading_group)
    time.sleep(0.05)

    # resets icon lists to empty
    global _icons
    _icons = []
    layout._cell_content_list = []

    # remove previous layer icons from the layout
    while len(layout) > 0:
        layout.pop()

    # set the layer labeled at the top of the screen
    layer_label.text = touch_deck_config["layers"][layer_index]["name"]

    # loop over each shortcut and it's index
    for i, shortcut in enumerate(touch_deck_config["layers"][layer_index]
["shortcuts"]):
        # create an icon for the current shortcut
        _new_icon = IconWidget(shortcut["label"], shortcut["icon"], on_disk=True)

        # add it to the list of icons
        _icons.append(_new_icon)

        # add it to the grid layout
        # calculate it's position from the index
        layout.add_content(_new_icon, grid_position=(i % 4, i // 4), cell_size=(1,
1))

    # hide the loading screen
    time.sleep(0.05)
    main_group.pop()

# append the grid layout to the main_group
# so it gets shown on the display
main_group.append(layout)

# load the first layer to start
load_layer(current_layer)

# main loop
while True:
    if touchscreen.touched:
        # loop over all data in touchscreen buffer
        while not touchscreen.buffer_empty:
            touches = touchscreen.touches
            # loop over all points touched
            for point in touches:
                if point:
                    # current time, used for timeout between icon presses
                    _now = time.monotonic()

                    # if the timeout has passed
                    if _now - LAST_PRESS_TIME > COOLDOWN_TIME:
                        # print(point)

                        # map the observed minimum and maximum touch values
                        # to the screen size
                        y = point["y"] - 250
                        x = 4096 - point["x"] - 250
                        y = y * display.width // (3820 - 250)
                        x = x * display.height // (3820 - 250)

                        # touch data is 90 degrees rotated

```

```

# flip x, and y here to account for that
p = (y, x)
# print(p)

# Next layer button pressed
if (
    next_layer_btn.contains(p)
    and NEXT_LAYER_INDEX not in _pressed_icons
):

    # increment layer
    current_layer += 1
    # wrap back to beginning from end
    if current_layer >= len(touch_deck_config["layers"]):
        current_layer = 0
    # load the new layer
    load_layer(current_layer)

    # save current time to check for timeout
    LAST_PRESS_TIME = _now

    # append this index to pressed icons for debouncing
    _pressed_icons.append(NEXT_LAYER_INDEX)

# home layer button pressed
if (
    home_layer_btn.contains(p)
    and HOME_LAYER_INDEX not in _pressed_icons
):
    # 0 index is home layer
    current_layer = 0
    # load the home layer
    load_layer(current_layer)

    # save current time to check for timeout
    LAST_PRESS_TIME = _now

    # append this index to pressed icons for debouncing
    _pressed_icons.append(HOME_LAYER_INDEX)

# Previous layer button pressed
if (
    prev_layer_btn.contains(p)
    and PREV_LAYER_INDEX not in _pressed_icons
):

    # decrement layer
    current_layer -= 1
    # wrap back to end from beginning
    if current_layer < 0:
        current_layer = len(touch_deck_config["layers"]) - 1

    # load the new layer
    load_layer(current_layer)

    # save current time to check for timeout
    LAST_PRESS_TIME = _now

    # append this index to pressed icons for debouncing
    _pressed_icons.append(PREV_LAYER_INDEX)

# loop over current layer icons and their indexes
for index, icon_shortcut in enumerate(_icons):
    # if this icon was pressed
    if icon_shortcut.contains(p):
        # debounce logic, check that it wasn't already
        pressed

        if index not in _pressed_icons:
            # print("pressed {}".format(index))

```

```

# get actions for this icon from config object
_cur_actions = touch_deck_config["layers"][
    current_layer
][["shortcuts"][index]["actions"]]

# tuple means it's a single action
if isinstance(_cur_actions, tuple):
    # put it in a list by itself
    _cur_actions = [_cur_actions]

# loop over the actions
for _action in _cur_actions:
    # HID keyboard keys
    if _action[0] == KEY:
        kbd.press(*_action[1])
        kbd.release(*_action[1])

    # String to write from layout
    elif _action[0] == STRING:
        kbd_layout.write(_action[1])

    # Consumer control code
    elif _action[0] == MEDIA:
        cc.send(_action[1])

    # Key press
    elif _action[0] == KEY_PRESS:
        kbd.press(*_action[1])

    # Key release
    elif _action[0] == KEY_RELEASE:
        kbd.release(*_action[1])

    # Change Layer
    elif _action[0] == CHANGE_LAYER:
        if isinstance(
            _action[1], int
        ) and 0 <= _action[1] < len(
            touch_deck_config["layers"]
        ):

            current_layer = _action[1]
            load_layer(_action[1])

    # if there are multiple actions
    if len(_cur_actions) > 1:
        # small sleep to make sure
        # OS can respond to previous action
        time.sleep(0.2)

# save current time to check for timeout
LAST_PRESS_TIME = _now
# append this index to pressed icons for
debouncing
    _pressed_icons.append(index)
else: # screen not touched

    # empty the pressed icons list
    _pressed_icons.clear()

```

Testing

Once you've pasted the project files onto your device, it will automatically reboot and begin running the touch deck app using the build-in example layers. Open up a text editor and click into it to give it focus. Press the "Play" icon at the top left of the screen. If everything working properly you should see "k" get typed into the text editor. If the touch deck app does not launch for you try opening the Mu Serial Console and check to see if there are any errors printed in there.

How it Works

Hardware Driver

This project is made to work with the `adafruit_featherwing` [library](https://adafru.it/Dbf) (<https://adafru.it/Dbf>) contains a helper class that we use to initialize the touch overlay and display. Once we've done that, we're ready to start drawing things on the screen and reacting to user touches.

Graphical Interface

There are two main components used in the interface: `GridLayout` and `IconWidget`. Both of them are in the `adafruit_displayio_layout` [library](https://adafru.it/QGE) (<https://adafru.it/QGE>). The `IconWidget` component contains an image, and optionally some text beneath it. All of the shortcut icons in the touch deck are `IconWidget` objects. The layer buttons on the right side are also `IconWidgets`, with a differently shaped image and no text. `GridLayout` is the widget we use to distribute our `IconWidgets` into a 4x3 grid. It does all the mathematical heavy lifting to size and position all of the widgets that we add to it.

Shortcut Sending

Whenever you press one of the touch deck icons, the code uses the `adafruit_hid` [library](https://adafru.it/xid) (<https://adafru.it/xid>) to send the keys, consumer codes, or strings specified by your layers configuration file.

Configuration

The code reads a configuration dictionary object from the file `touch_deck_layers.py`. Within this file, you can define the layers of icons that you'd like to use, and set the keys and other shortcuts they'll send when you touch them.

Customizing the Layers File

Your layers file is where you define what icons you want to use and what actions they should do when you press them. These are grouped into lists known as layers. Each layer can hold up to 12 icons. Each layer contains a `name` that will get shown at the top of the screen.

Each item in the layer has three properties: `label`, `icon`, and `actions`. The `icon` is a string filepath pointing to the bmp image that should be shown. All of the icons in this project are in the `touch_deck_icons/` directory. The `label` property is the text that will be shown under the icon image. Lastly, the `actions` are the things to do when you press on the icon -- keys to press, or strings to write.

There are five types of actions that can be used on an icon. Each icon can have a single action, or a list of actions that will get run in sequence.

KEY

This will press and release a single key. It can also contain modifier keys that will be held down while the selected key is pressed.

```
{
  "label": "Play",
  "icon": "touch_deck_icons/pr_play.bmp",
  "actions": (KEY, [Keycode.K]),
},
{
  "label": "Fast",
  "icon": "touch_deck_icons/pr_fast.bmp",
  "actions": (KEY, [Keycode.RIGHT_SHIFT, Keycode.PERIOD]),
}
```

In the example above, there is a "Play" shortcut that presses the "k" key, and a "Fast" shortcut that holds shift while pressing the period key, which would make ">" -- which is the fast forward shortcut for YouTube.

MEDIA

This action type will send the specified consumer control codes.

```
{
  "label": "Vol +",
  "icon": "touch_deck_icons/pr_volup.bmp",
  "actions": (MEDIA, ConsumerControlCode.VOLUME_INCREMENT),
},
```

This shortcut will press the **volume up** key.

STRING

The STRING action type will write a string using the US Keyboard Layout. You can use this to type words.

```
{
  "label": "Blinka",
  "icon": "touch_deck_icons/af_blinka.bmp",
  "actions": (STRING, ":blinka:"),
}
```

This shortcut will type the string `":blinka:"` which gets interpreted by Discord as an emoji and gets replaced by a Blinka icon on the Adafruit Discord server.

KEY_PRESS and KEY_RELEASE

The final 2 action types are meant to be used together. These allow you to hold a key down while pressing several other keys. These are helpful for PC shortcuts like "hold alt and type 1234". `KEY_PRESS` will press and hold a key down. Then you can do some other actions. And finally `KEY_RELEASE` will release the pressed key.

`KEY_RELEASE` will release the pressed key.

```
{
  "label": "Test (L)",
  "icon": "touch_deck_icons/test48_icon.bmp",
  "actions": [
    (KEY_PRESS, [Keycode.SHIFT]),
    (KEY, [Keycode.B]),
    (KEY, [Keycode.L]),
    (KEY, [Keycode.I]),
    (KEY, [Keycode.N]),
    (KEY, [Keycode.K]),
    (KEY, [Keycode.A]),
    (KEY_RELEASE, [Keycode.SHIFT])
  ],
},
```

This example holds the shift key down, and then presses the B, L, I, N, K, A keys in succession.

Full Layers File

```

# SPDX-FileCopyrightText: 2021 foamyguy for Adafruit Industries
#
# SPDX-License-Identifier: MIT

from adafruit_hid.keycode import Keycode
from adafruit_hid.consumer_control_code import ConsumerControlCode

MEDIA = 1
KEY = 2
STRING = 3
KEY_PRESS = 4
KEY_RELEASE = 5
CHANGE_LAYER = 6

touch_deck_config = {
    "layers": [
        {
            "name": "Youtube Controls",
            "shortcuts": [
                {
                    "label": "Play",
                    "icon": "touch_deck_icons/pr_play.bmp",
                    "actions": (KEY, [Keycode.K]),
                },
                {
                    "label": "Pause",
                    "icon": "touch_deck_icons/pr_pause.bmp",
                    "actions": (KEY, [Keycode.K]),
                },
                {
                    "label": "Rewind",
                    "icon": "touch_deck_icons/pr_rewind.bmp",
                    "actions": (KEY, [Keycode.LEFT_ARROW]),
                },
                {
                    "label": "FastForward",
                    "icon": "touch_deck_icons/pr_ffwd.bmp",
                    "actions": (KEY, [Keycode.RIGHT_ARROW]),
                },
                {
                    "label": "Previous",
                    "icon": "touch_deck_icons/pr_previous.bmp",
                    "actions": (KEY, [Keycode.RIGHT_SHIFT, Keycode.P]),
                },
                {
                    "label": "Next",
                    "icon": "touch_deck_icons/pr_next.bmp",
                    "actions": (KEY, [Keycode.RIGHT_SHIFT, Keycode.N]),
                },
                {
                    "label": "Vol -",
                    "icon": "touch_deck_icons/pr_voldown.bmp",
                    "actions": (MEDIA, ConsumerControlCode.VOLUME_DECREMENT),
                },
                {
                    "label": "Vol +",
                    "icon": "touch_deck_icons/pr_volup.bmp",
                    "actions": (MEDIA, ConsumerControlCode.VOLUME_INCREMENT),
                },
                {
                    "label": "Fullscreen",
                    "icon": "touch_deck_icons/pr_fullscreen.bmp",
                    "actions": (KEY, [Keycode.F]),
                },
                {
                    "label": "Slow",
                    "icon": "touch_deck_icons/pr_slow.bmp",
                    "actions": (KEY, [Keycode.RIGHT_SHIFT, Keycode.COMMA]),
                }
            ]
        }
    ]
}

```

```

    },
    {
      "label": "Fast",
      "icon": "touch_deck_icons/pr_fast.bmp",
      "actions": (KEY, [Keycode.RIGHT_SHIFT, Keycode.PERIOD]),
    },
    {
      "label": "Mute",
      "icon": "touch_deck_icons/pr_mute.bmp",
      "actions": (KEY, [Keycode.M]),
    },
  ],
},
{
  "name": "Discord",
  "shortcuts": [
    {
      "label": "Blinka",
      "icon": "touch_deck_icons/af_blinka.bmp",
      "actions": (STRING, ":blinka:"),
    },
    {
      "label": "Adabot",
      "icon": "touch_deck_icons/af_adabot.bmp",
      "actions": (STRING, ":adabot:"),
    },
    {
      "label": "Billie",
      "icon": "touch_deck_icons/af_billie.bmp",
      "actions": (STRING, ":billie:"),
    },
    {
      "label": "Cappy",
      "icon": "touch_deck_icons/af_cappy.bmp",
      "actions": (STRING, ":cappy:"),
    },
    {
      "label": "Connie",
      "icon": "touch_deck_icons/af_connie.bmp",
      "actions": (STRING, ":connie:"),
    },
    {
      "label": "Gus",
      "icon": "touch_deck_icons/af_gus.bmp",
      "actions": (STRING, ":gus:"),
    },
    {
      "label": "Hans",
      "icon": "touch_deck_icons/af_hans.bmp",
      "actions": (STRING, ":hans:"),
    },
    {
      "label": "Mho",
      "icon": "touch_deck_icons/af_mho.bmp",
      "actions": (STRING, ":mho:"),
    },
    {
      "label": "Minerva",
      "icon": "touch_deck_icons/af_minerva.bmp",
      "actions": (STRING, ":minerva:"),
    },
    {
      "label": "NeoTrellis",
      "icon": "touch_deck_icons/af_neotrellis.bmp",
      "actions": (STRING, ":neotrellis:"),
    },
    {
      "label": "Ruby",
      "icon": "touch_deck_icons/af_ruby.bmp",
    }
  ]
}

```

```

    "actions": (STRING, ":ruby:"),
  },
  {
    "label": "Sparky",
    "icon": "touch_deck_icons/af_sparky.bmp",
    "actions": (STRING, ":sparky:"),
  },
],
},
{
  "name": "Symbols",
  "shortcuts": [
    {
      "label": "Infinity", # ∞
      "icon": "touch_deck_icons/sy_infinity.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.FIVE]),
    },
    {
      "label": "Degree", # °
      "icon": "touch_deck_icons/sy_degree.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.ZERO]),
    },
    {
      "label": "Pi", # π
      "icon": "touch_deck_icons/sy_pi.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.P]),
    },
    {
      "label": "Sigma", # Σ
      "icon": "touch_deck_icons/sy_sigma.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.W]),
    },
    {
      "label": "Partial diff", #
      "icon": "touch_deck_icons/sy_pdiff.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.D]),
    },
    {
      "label": "Increment", # Δ
      "icon": "touch_deck_icons/sy_increment.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.J]),
    },
    {
      "label": "Omega", # Ω
      "icon": "touch_deck_icons/sy_omega.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.Z]),
    },
    {
      "label": "Mu", # μ
      "icon": "touch_deck_icons/sy_micro.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.M]),
    },
    {
      "label": "Rad 0", # ∅
      "icon": "touch_deck_icons/sy_rado.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.SHIFT, Keycode.O]),
    },
    {
      "label": "Square root", # √
      "icon": "touch_deck_icons/sy_sqrtrt.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.V]),
    },
    {
      "label": "Approx", # ≈
      "icon": "touch_deck_icons/sy_approx.bmp",
      "actions": (KEY, [Keycode.ALT, Keycode.X]),
    },
    {
      "label": "Plus minus", # ±

```

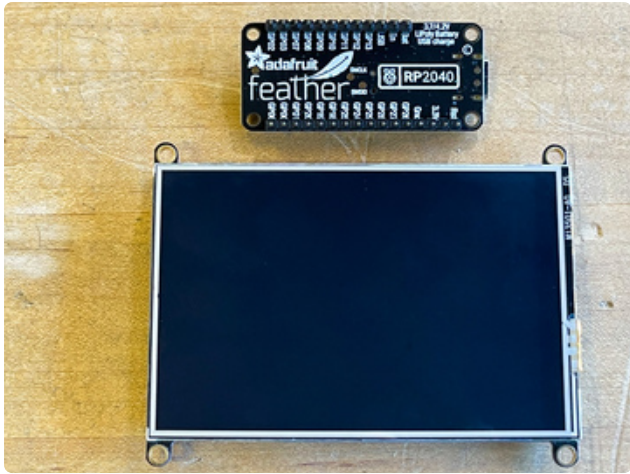
```
        "icon": "touch_deck_icons/sy_plusminus.bmp",  
        "actions": (KEY, [Keycode.ALT, Keycode.SHIFT, Keycode.EQUALS]),  
    },  
},  
],  
}
```

Assemble the Touch Deck



To build the case as shown here, first 3D print the base and top models:

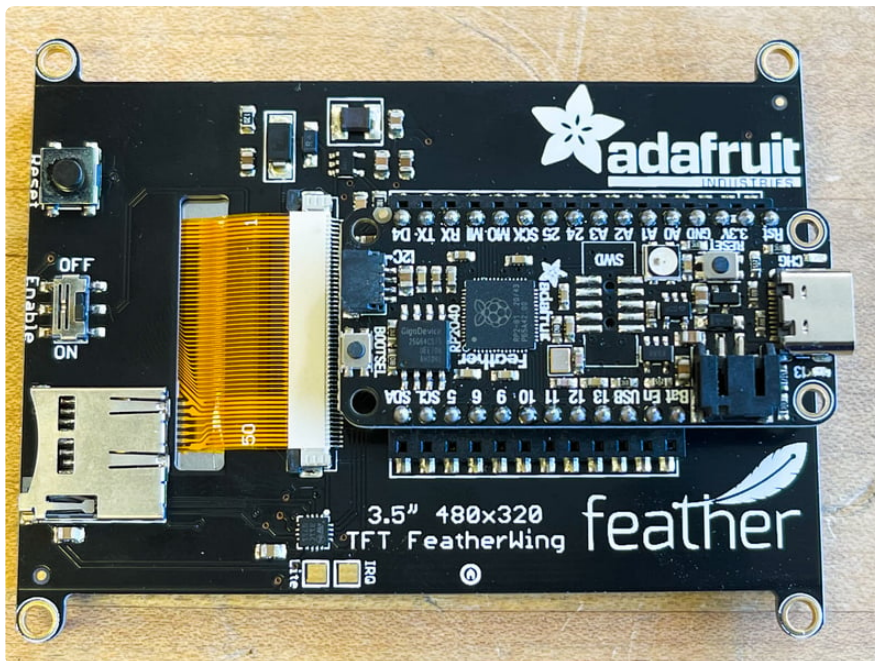
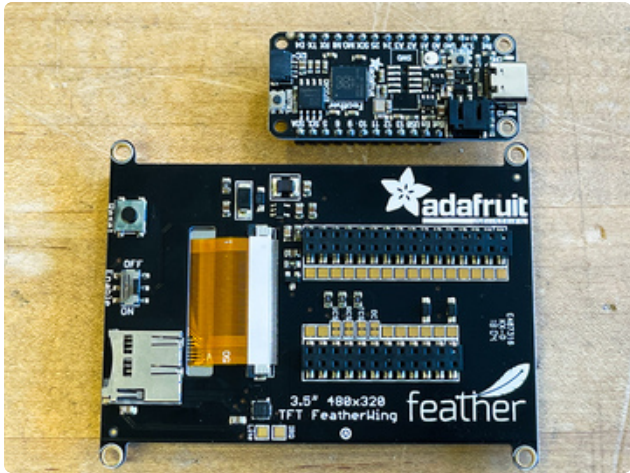
<https://adafru.it/Rhc>



Feather + Wing

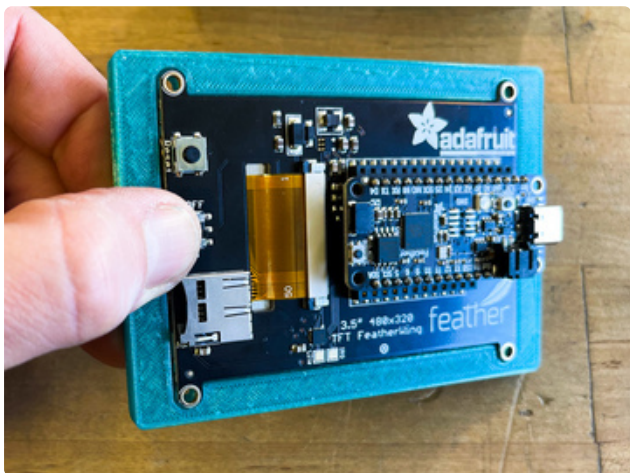
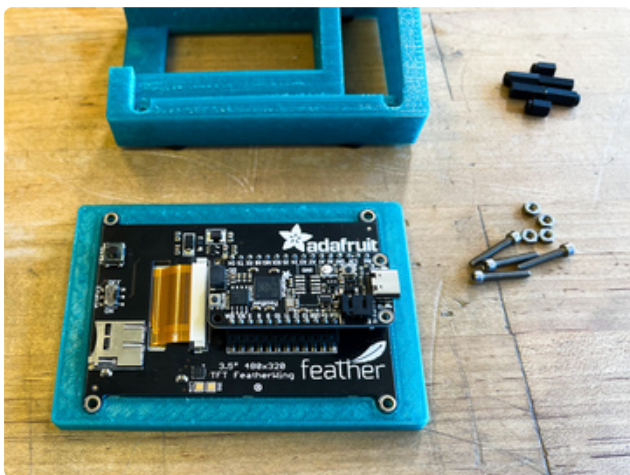
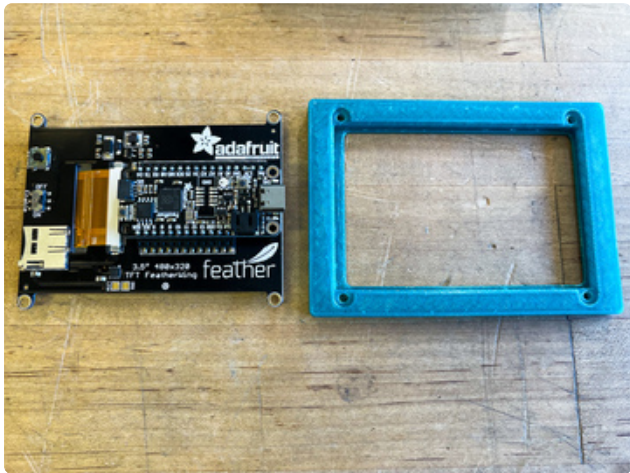
Solder header pins under the Feather board as shown so that you can plug it into the TFT FeatherWing.

[See this guide section \(https://adafru.it/Rhd\)](https://adafru.it/Rhd) on "Soldering Plain Headers" for more info.





FeatherWing has an ENABLE switch which will disable both Feather and g - so make sure it is turned ON while you're using it



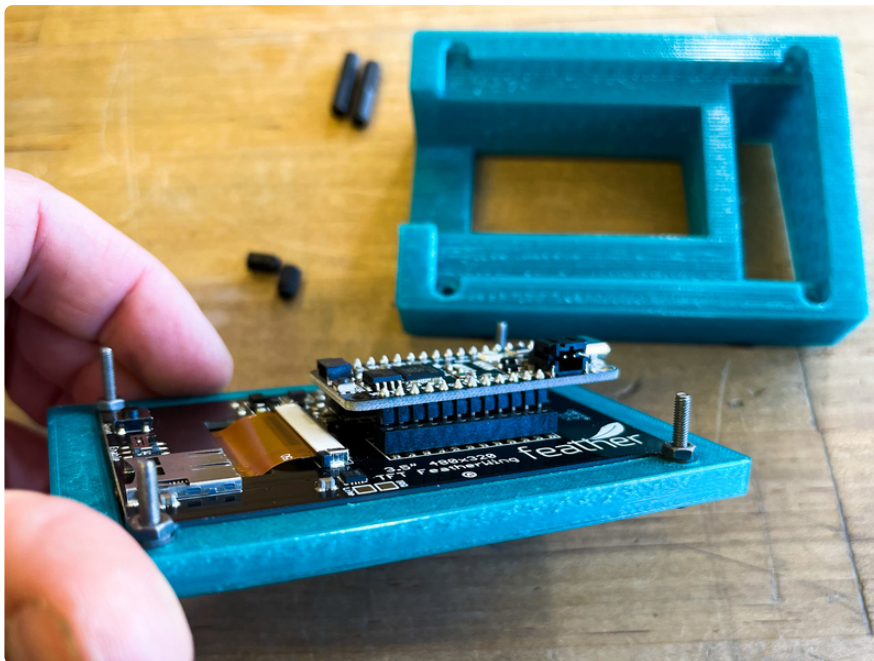
Case Top

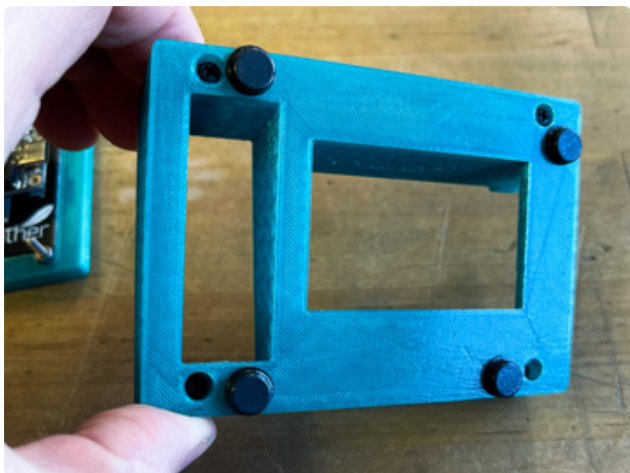
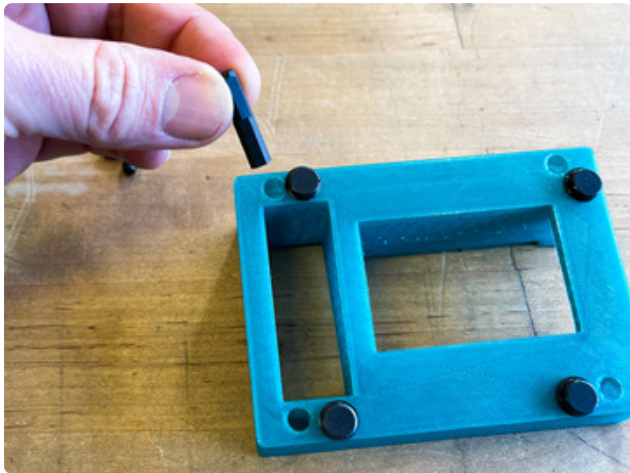
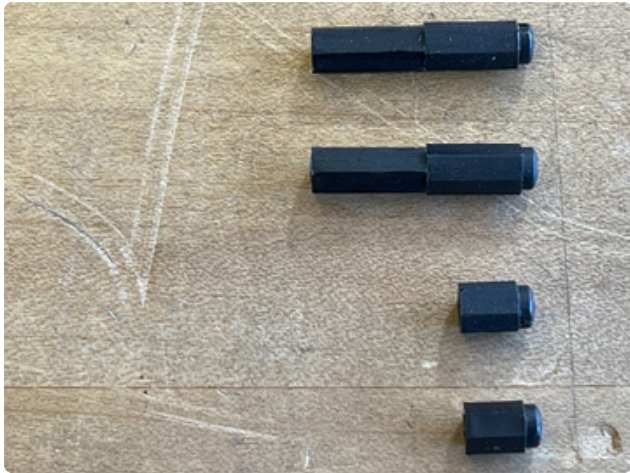
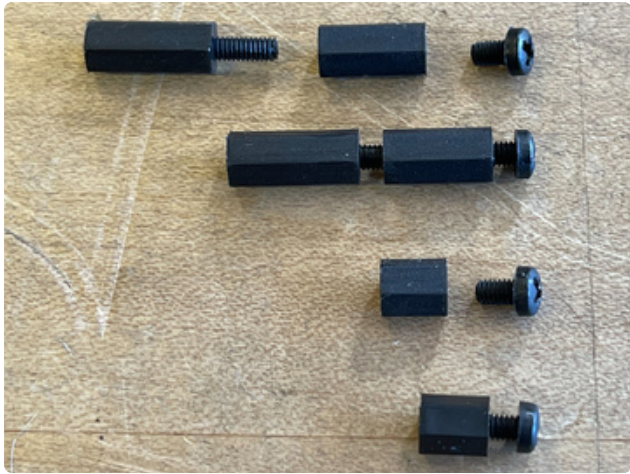
The TFT fits into the case top snugly. Note that the bezel has one slightly wider side which should be on the right side of the display to cover the blank area of the screen assembly.



Top Screws

Insert the four M2.5 x 16mm long screws from the top of the case, going through the mounting holes of the TFT. Thread the four nuts onto the screws, getting them finger tight.





Standoffs

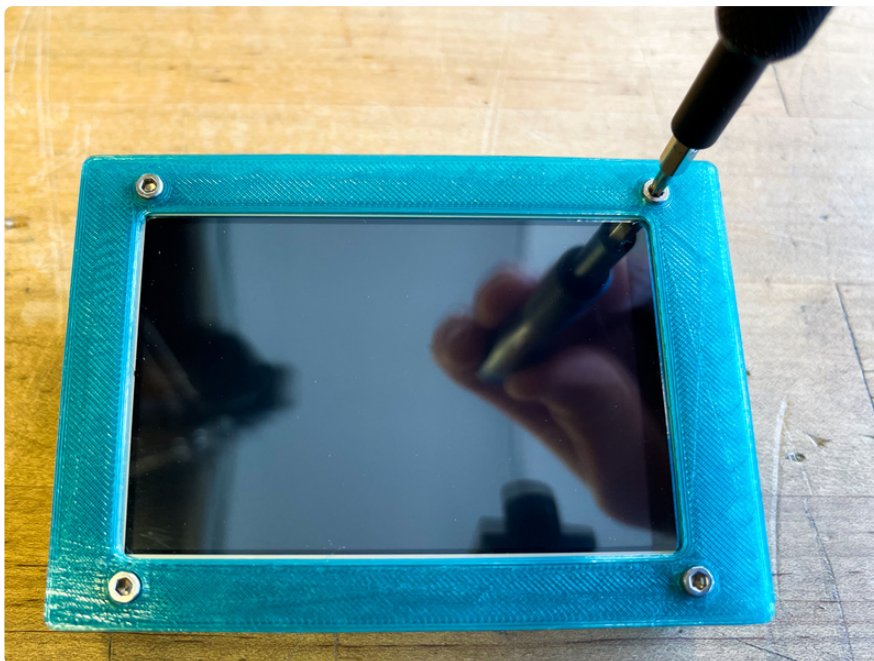
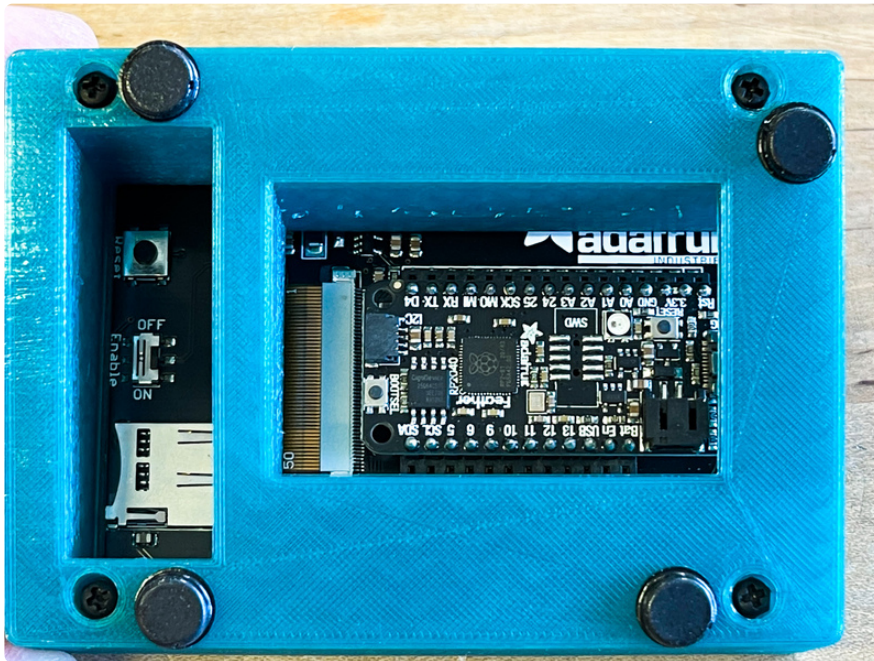
Assemble four sets of standoffs and screws into two different lengths as shown. These will be screwed onto the long screws from underneath the base.

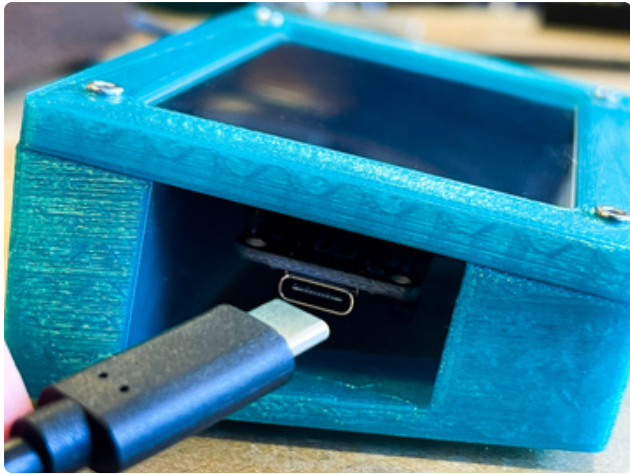
Insert the two longer standoffs into the deeper holes, and the two shorter standoffs into the shallower holes.

Assemble

Push the case top and screen/Feather into the base and then screw the standoffs onto the screws.

You can see here some suggested placements for the four rubber bumper feet. These help prevent sliding.





USB Plug

You can plug your Touch Deck into your computer directly with a USB-C cable, or get a snazzy right angle adapter!





Use the Touch Deck

The Touch Deck launches with your default layer of icons. Use any of them now, or pick a different layer with the Layer Next or Layer Back buttons.

Press the Layer Home button to return to your default layer at any time.



Customize

The Touch Deck download from the previous page includes the icons and layer file for three different pages: YouTube Controller, Discord Adafruit Emoji, and Greek/Math Symbols.

You can create your own custom icons and layers, such as this Wirecast streaming setup that includes eight camera switching controls plus four app launcher/switcher buttons.