# PyPortal Roku Remote

Created by Eva Herrada

https://learn.adafruit.com/pyportal-roku-remote

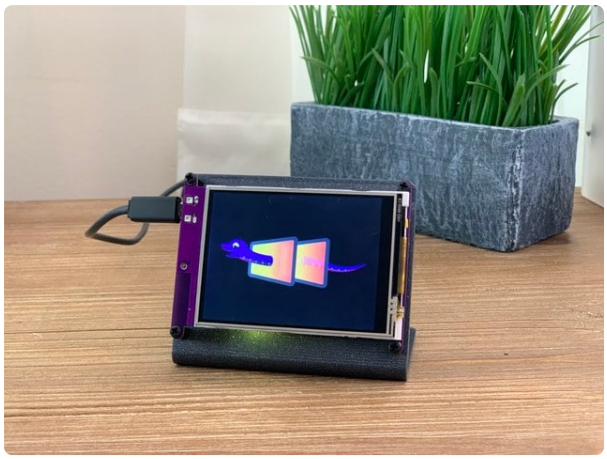
Last updated on 2024-06-03 03:09:38 PM EDT

# Table of Contents

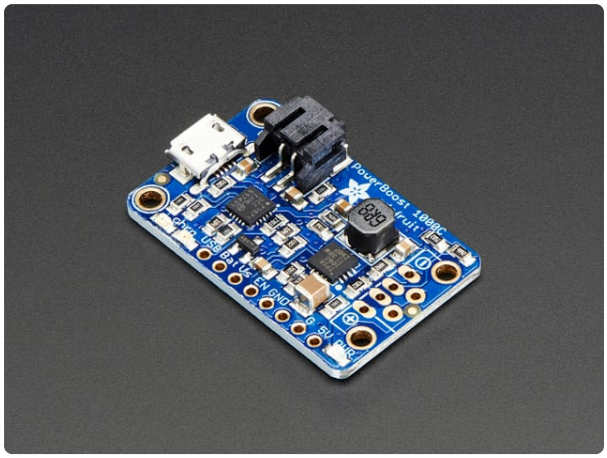
# Overview

This project uses a PyPortal to make a customizable remote that works over WiFi to control a Roku media player. It uses the Roku External Control Protocol (https://adafru.it/MbY) and takes advantage of the PyPortal's touchscreen for a very easy to use input method.

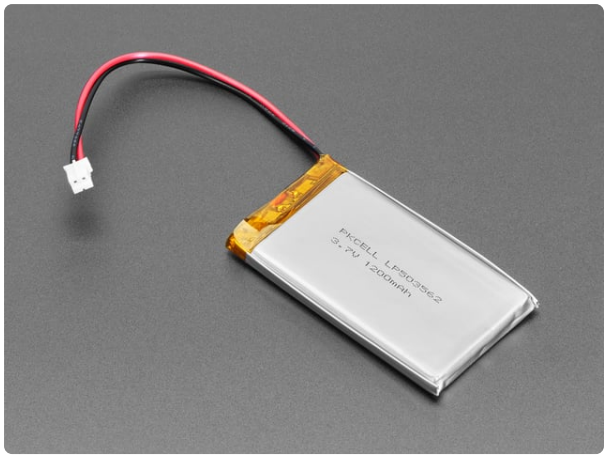
## Parts



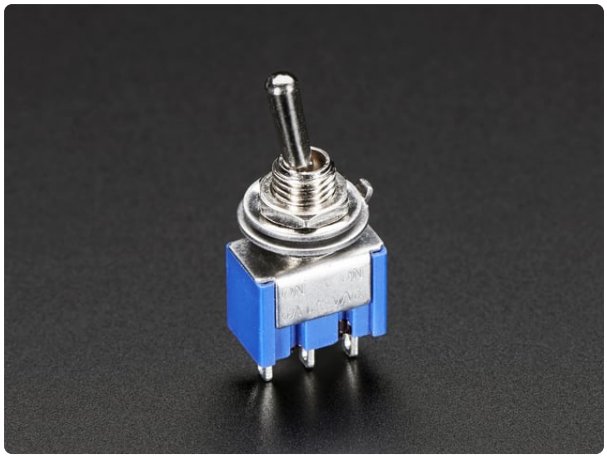
[Adafruit PyPortal - CircuitPython Powered Internet Display]
PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...
https://www.adafruit.com/product/4116



[PowerBoost 1000 Charger - Rechargeable 5V Lipo USB Boost @ 1A]
PowerBoost 1000C is the perfect power supply for your portable project! With a built-in load-sharing battery charger circuit, you'll be able to keep your power-hungry...
https://www.adafruit.com/product/2465



[Lithium Ion Polymer Battery - 3.7v 1200mAh]
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/258



Mini Panel Mount SPDT Toggle Switch
This or that, one or the other, perhaps or perhaps not! So hard to make decisions these days without feeling like you're just going back and forth constantly. Deciding whether or...
https://www.adafruit.com/product/3221

USB cable - USB A to Micro-B
This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...
https://www.adafruit.com/product/592

## Additional Items

You will also need a Roku. Just about any version should work, including a smart tv with a built-in one.

If you want to make the case, you also need a 3D printer. If you aren't making the 3D printed case, you only need the PyPortal and a LiPo. Keep in mind that the PyPortal **does not have a 2-pin JST** but can be powered via the 3-pin JST. If you don't make the case, you will still need to either make or buy an adapter. Only the pins with the red and black cables are necessary to power the PyPortal, so you should be able to strip and solder the cables together very easily.

---

# 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** "flash" drive to iterate.

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

## Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)

**Download the latest version of CircuitPython for the PyPortal via CircuitPython.org**

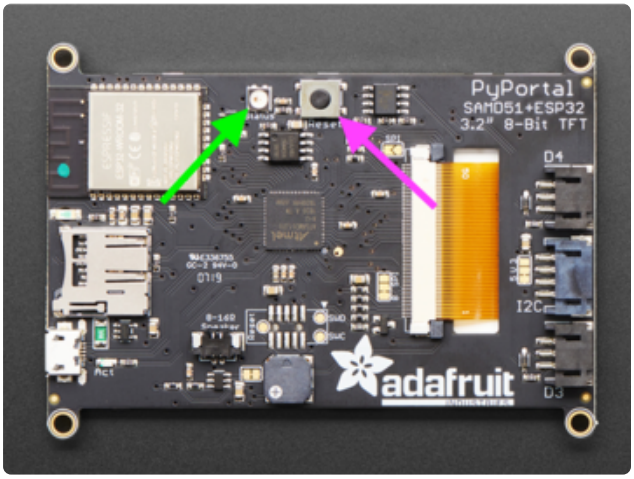
https://adafru.it/Egk

**Download the latest version of CircuitPython for the PyPortal Pynt via CircuitPython.org**

https://adafru.it/HFd


**Click the link above to download the latest version of CircuitPython for the PyPortal.**

Download and save it to your desktop (or wherever is handy).

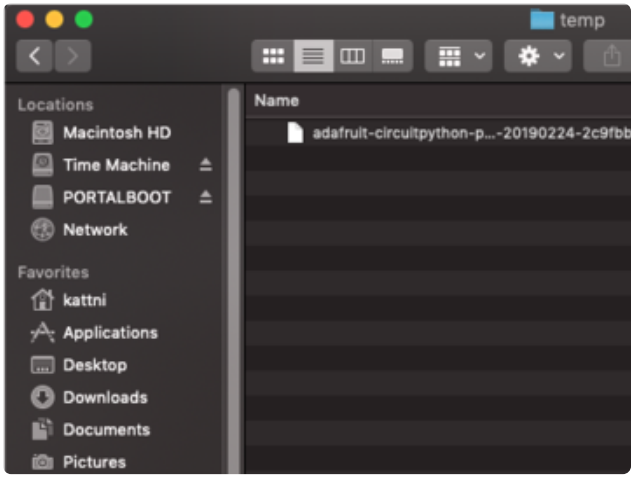



Plug your PyPortal into your computer using a known-good USB cable.

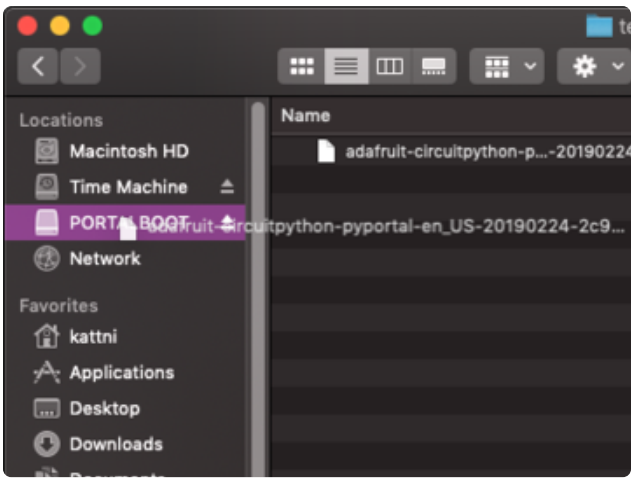
**A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.**

Double-click the **Reset** button on the top in the middle (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

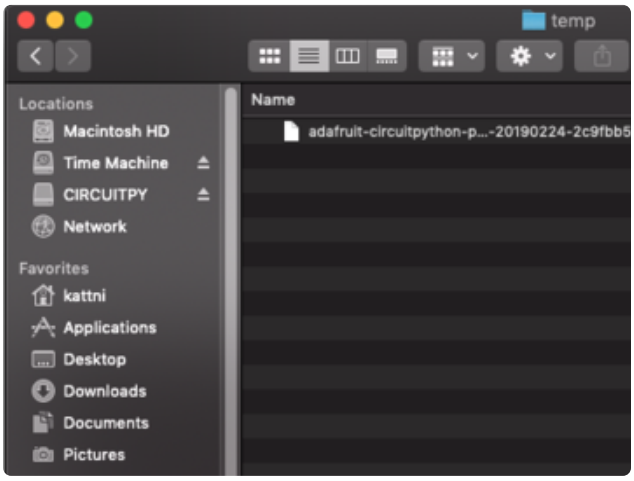
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!




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




Drag the **adafruit-circuitpython-pyportal-<whatever>.uf2** file to **PORTALBOOT.**




The LED will flash. Then, the **PORTALBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

If you haven't added any code to your board, the only file that will be present is **boot_out.txt**. This is absolutely normal! It's time for you to add your **code.py** and get started!

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

## PyPortal Default Files

Click below to download a zip of the files that shipped on the PyPortal or PyPortal Pynt.

**PyPortal Default Files**

https://adafru.it/UF-

**PyPortal Pynt Default Files**

https://adafru.it/UGa

---

# PyPortal CircuitPython Setup

To use all the amazing features of your PyPortal with CircuitPython, you must first install a number of libraries. This page covers that process.

# Adafruit CircuitPython Bundle

Download the Adafruit CircuitPython Library Bundle. You can find the latest release here:

**Latest Adafruit CircuitPython Library Bundle**

https://adafru.it/ENC

Download the **adafruit-circuitpython-bundle-*.x-mpy-*.zip** bundle zip file where ***.x MATCHES THE VERSION OF CIRCUITPYTHON YOU INSTALLED**, and unzip a folder of the same name. Inside you'll find a **lib** folder. You have two options:

- You can add the **lib** folder to your **CIRCUITPY** drive. This will ensure you have all the drivers. But it will take a bunch of space on the 8 MB disk
- Add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into **CIRCUITPY/lib** now!

- **adafruit_esp32spi** - This is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- **adafruit_requests** - This library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- adafruit_connection_manager - used by adafruit_requests.
- **adafruit_pyportal** - This is our friendly wrapper library that does a lot of our projects, displays graphics and text, fetches data from the internet. Nearly all of our projects depend on it!
- **adafruit_portalbase** - This library is the base library that adafruit_pyportal library is built on top of.
- **adafruit_touchscreen** - a library for reading touches from the resistive touchscreen. Handles all the analog noodling, rotation and calibration for you.
- **adafruit_io** - this library helps connect the PyPortal to our free datalogging and viewing service
- **adafruit_imageload** - an image display helper, required for any graphics!
- **adafruit_display_text** - not surprisingly, it displays text on the screen
- **adafruit_bitmap_font** - we have fancy font support, and its easy to make new fonts. This library reads and parses font files.
- **adafruit_slideshow** - for making image slideshows - handy for quick display of graphics and sound
- **neopixel** - for controlling the onboard neopixel
- **adafruit_adt7410** - library to read the temperature from the on-board Analog Devices ADT7410 precision temperature sensor (not necessary for Titano or Pynt)
- **adafruit_bus_device** - low level support for I2C/SPI
- **adafruit_fakerequests** - This library allows you to create fake HTTP requests by using local files.

# 3D Printed Case

3D Printed Case (https://adafru.it/Eel)

# CircuitPython Setup and Code

Click on **Download Project Bundle** button below to download the **code.py** and other files. Connect the PyPortal to your computer via a known good data+power USB cable. The PyPortal should show up in your operating system file explorer as a drive called **CIRCUITPY**.

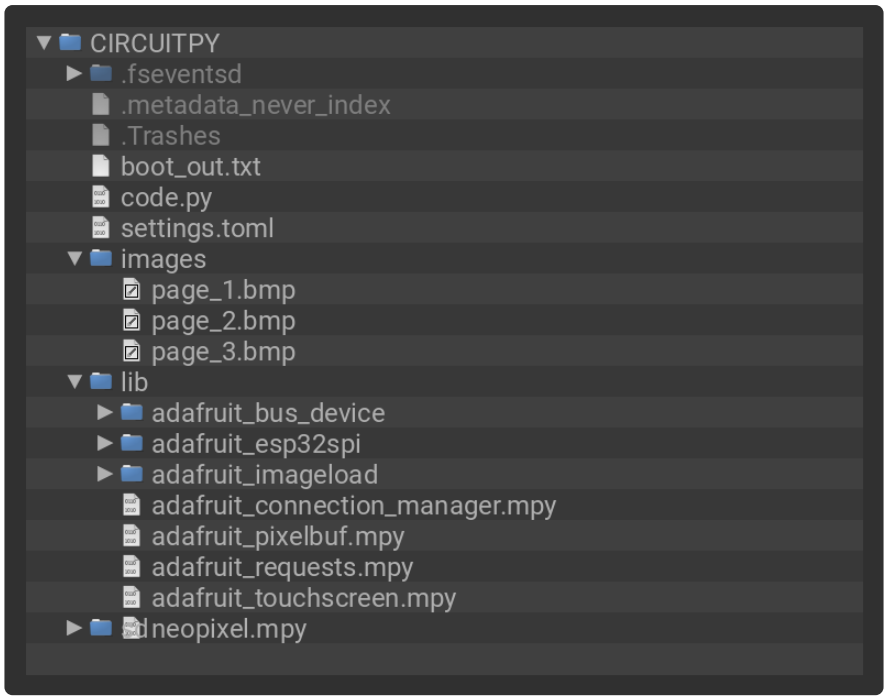
```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import math
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import displayio
import adafruit_touchscreen
import adafruit_imageload

# Set up the touchscreen
ts = adafruit_touchscreen.Touchscreen(
    board.TOUCH_XL,
    board.TOUCH_XR,
    board.TOUCH_YD,
    board.TOUCH_YU,
    calibration=((5200, 59000), (5800, 57000)),
    size=(320, 240),
)

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)

wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

# Set the ip of your Roku here
ip = "192.168.1.3"

"""
Possible keypress key values to send the Roku
```

```
Home
Rev
Fwd
Play
Select
Left
Right
Down
Up
Back
InstantReplay
Info
Backspace
Search
Enter
FindRemote

Keypress key values that only work on smart TVs with built-in Rokus
VolumeDown
VolumeMute
VolumeUp
PowerOff
ChannelUp
ChannelDown
"""


def getchannels():
    """ Gets the channels installed on the device. Also useful because it
        verifies that the PyPortal can see the Roku"""
    try:
        print("Getting channels. Usually takes around 10 seconds...", end="")
        response = wifi.get("http://{}:8060/query/apps".format(ip))
        channel_dict = {}
        for i in response.text.split("<app")[2:]:
            a = i.split("=")
            chan_id = a[1].split('"')[1]
            name = a[-1].split(">")[1].split("<")[0]
            channel_dict[name] = chan_id
        response.close()
        return channel_dict
    except (ValueError, RuntimeError, ConnectionError, OSError) as e:
        print("Failed to get data\n", e)
        wifi.reset()
        return None
    response = None
    return None


def sendkey(key):
    """ Sends a key to the Roku """
    try:
        print("Sending key: {}...".format(key), end="")
        response = wifi.post("http://{}:8060/keypress/{}".format(ip, key))
        if response:
            response.close()
            print("OK")
    except (ValueError, RuntimeError, ConnectionError, OSError) as e:
        print("Failed to get data\n", e)
        wifi.reset()
        return
    response = None


def sendletter(letter):
    """ Sends a letter to the Roku, not used in this guide """
    try:
        print("Sending letter: {}...".format(letter), end="")
        response = wifi.post("http://{}:8060/keypress/lit_{}".format(ip, letter))
```

```
        if response:
            response.close()
            print("OK")
    except (ValueError, RuntimeError, ConnectionError, OSError) as e:
        print("Failed to get data\n", e)
        wifi.reset()
        return
    response = None


def openchannel(channel):
    """ Tells the Roku to open the channel with the corresponding channel id """
    try:
        print("Opening channel: {}...".format(channel), end="")
        response = wifi.post("http://{}:8060/launch/{}".format(ip, channel))
        if response:
            response.close()
            print("OK")
        response = None
    except (ValueError, RuntimeError, ConnectionError, OSError) as e:
        print("Probably worked, but got error\n", e)
        wifi.reset()
    response = None


def switchpage(tup):
    """ Used to switch to a different page """
    p_num = tup[0]
    tile_grid = tup[1]
    new_page = pages[p_num - 1]
    new_page_vals = pages_vals[p_num - 1]
    my_display_group[-1] = tile_grid
    return new_page, new_page_vals


# Verifies the Roku and Pyportal are connected and visible
channels = getchannels()

my_display_group = displayio.Group()

image_1, palette_1 = adafruit_imageload.load(
    "images/page_1.bmp", bitmap=displayio.Bitmap, palette=displayio.Palette
)
tile_grid_1 = displayio.TileGrid(image_1, pixel_shader=palette_1)
my_display_group.append(tile_grid_1)

image_2, palette_2 = adafruit_imageload.load(
    "images/page_2.bmp", bitmap=displayio.Bitmap, palette=displayio.Palette
)
tile_grid_2 = displayio.TileGrid(image_2, pixel_shader=palette_2)

image_3, palette_3 = adafruit_imageload.load(
    "images/page_3.bmp", bitmap=displayio.Bitmap, palette=displayio.Palette
)
tile_grid_3 = displayio.TileGrid(image_3, pixel_shader=palette_3)

# fmt: off
# Page 1
page_1 = [sendkey, sendkey, sendkey,
          sendkey, sendkey, sendkey,
          sendkey, sendkey, sendkey,
          sendkey, sendkey, switchpage]

page_1_vals = ["Back", "Up", "Home",
               "Left", "Select", "Right",
               "Search", "Down", "Info",
               "Rev", "Play", (2, tile_grid_2)]

# Page 2
```

```
page_2 = [openchannel, openchannel, openchannel,
          openchannel, openchannel, openchannel,
          openchannel, openchannel, openchannel,
          switchpage, sendkey, switchpage]

page_2_vals = [14362, 2285, 13,
               12, 8378, 837,
               38820, 47389, 7767,
               (1, tile_grid_1), "Home", (3, tile_grid_3)]

page_3 = [None, None, None,
          sendkey, None, sendkey,
          sendkey, sendkey, sendkey,
          switchpage, sendkey, sendkey]

page_3_vals = [None, None, None,
               "Search", None, "Info",
               "Rev", "Play", "Fwd",
               (2, tile_grid_2), "Back", "Home"]
# fmt: on

pages = [page_1, page_2, page_3]
pages_vals = [page_1_vals, page_2_vals, page_3_vals]

page = page_1
page_vals = page_1_vals

board.DISPLAY.root_group = my_display_group
print("READY")


last_index = 0
while True:
    p = ts.touch_point
    if p:
        x = math.floor(p[0] / 80)
        y = abs(math.floor(p[1] / 80) - 2)
        index = 3 * x + y
        # Used to prevent the touchscreen sending incorrect results
        if last_index == index:
            if page[index]:
                # pylint: disable=comparison-with-callable
                if page[index] == switchpage:
                    page, page_vals = switchpage(page_vals[index])
                else:
                    page[index](page_vals[index])
                time.sleep(0.1)

        last_index = index
```

Copy all the files from the zip file to the PyPortal **CIRCUITPY** drive. The files on **CIRCUITPY** should be similar to the directory listing below.




---

# Code Run Through

The first few lines of **code.py** just import all the libraries we'll be using.

```
import time
import math
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import displayio
import adafruit_touchscreen
import adafruit_imageload
```

Then, the touchscreen is set up and calibrated, creating the `ts` object we'll be using to find where the user is pressing.

```
ts = adafruit_touchscreen.Touchscreen(
    board.TOUCH_XL,
    board.TOUCH_XR,
    board.TOUCH_YD,
    board.TOUCH_YU,
    calibration=((5200, 59000), (5800, 57000)),
    size=(320, 240),
)
```

Now, we set up the WiFi connection. Make sure you have the latest version of CircuitPython installed since this will definitely not work on older versions.

```
try:
    from secrets import secrets
```

```
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)

wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)
```

Now we declare the IP of the Roku. You can find this by going to **Settings -> Network -> About**. In my case it was `192.168.1.3`, but yours will probably be different. The keypress commands you can send to the Roku are listed out. See the 'Keypress key values' section of the Roku ECP documentation (https://adafru.it/MbY) for more info about these.

```
# Set the ip of your Roku here
ip = "192.168.1.3"

"""
Possible keypress key values to send the Roku
Home
Rev
Fwd
Play
Select
Left
Right
Down
Up
Back
InstantReplay
Info
Backspace
Search
Enter
FindRemote

Keypress key values that only work on smart TVs with built-in Rokus
VolumeDown
VolumeMute
VolumeUp
PowerOff
ChannelUp
ChannelDown
"""
```

This is the first of four helper functions. This function gets the XML with the installed channels and their IDs. See the 'General ECP Commands' section of the Roku ECP documentation (https://adafru.it/MbY) for more info about this.

```
def getchannels():
    """ Gets the channels installed on the device. Also useful because it
        verifies that the PyPortal can see the Roku"""
```

```
    try:
        print("Getting channels. Usually takes around 10 seconds...", end="")
        response = wifi.get("http://{}:8060/query/apps".format(ip))
        channel_dict = {}
        for i in response.text.split("&lt;app")[2:]:
            a = i.split("=")
            chan_id = a[1].split('"')[1]
            name = a[-1].split("&gt;")[1].split("&lt;")[0]
            channel_dict[name] = chan_id
        response.close()
        return channel_dict
    except (ValueError, RuntimeError) as e:
        print("Failed to get data\n", e)
        wifi.reset()
        return None
    response = None
    return None
```

This function sends keypresses to the Roku. It sends commands for stuff like navigation and media control. It takes a string as input. See the 'General ECP Commands' section of the Roku ECP documentation (https://adafru.it/MbY) for more info about this.

```
def sendkey(key):
    """ Sends a key to the Roku """
    try:
        print("Sending key: {}...".format(key), end="")
        response = wifi.post("http://{}:8060/keypress/{}".format(ip, key))
        if response:
            response.close()
            print("OK")
    except (ValueError, RuntimeError) as e:
        print("Failed to get data\n", e)
        wifi.reset()
        return
    response = None
```

This function isn't actually used, but if it were, it could be used to send characters when the on-screen keyboard is active. It takes a character as input. See the 'General ECP Commands' section of the Roku ECP documentation (https://adafru.it/MbY) for more info about this.

```
def sendletter(letter):
    """ Sends a letter to the Roku, not used in this guide """
    try:
        print("Sending letter: {}...".format(letter), end="")
        response = wifi.post("http://{}:8060/keypress/lit_{}".format(ip, letter))
        if response:
            response.close()
            print("OK")
    except (ValueError, RuntimeError) as e:
        print("Failed to get data\n", e)
        wifi.reset()
        return
    response = None
```

Openchannel sends the Roku the ID associated with the channel you'd like to open, and the Roku then opens said channel. It takes an integer as an input. See the

'General ECP Commands' section of the Roku ECP documentation (https://adafru.it/MbY) for more info about this.

```
def openchannel(channel):
    """ Tells the Roku to open the channel with the corresponding channel id """
    try:
        print("Opening channel: {}...".format(channel), end="")
        response = wifi.post("http://{}:8060/launch/{}".format(ip, channel))
        if response:
            response.close()
            print("OK")
        response = None
    except (ValueError, RuntimeError):
        print("Probably worked")
        wifi.reset()
    response = None
```

This function takes in a tuple of the page you would like to go to and the tile grid of the bitmap for that page and then switches you to that page. It wasn't possible to pass in two values since they were stored in a list, which is why it takes a tuple. It returns two new lists that are used to determine which functions run when a button gets pressed and what values get sent to the functions.

```
def switchpage(tup):
    """ Used to switch to a different page """
    p_num = tup[0]
    tile_grid = tup[1]
    new_page = pages[p_num - 1]
    new_page_vals = pages_vals[p_num - 1]
    my_display_group[-1] = tile_grid
    return new_page, new_page_vals
```

This line isn't really necessary, but it does serve a useful purpose. It verifies that everything is still connected to WiFi and you can communicate with the Roku. Additionally, the first communication you make with the Roku using the ECP takes a very long time, so this takes care of that so the remote is ready to go as soon as the images show up on the screen.

```
# Verifies the Roku and Pyportal are connected and visible
channels = getchannels()
```

Now, we set up displayio. We load 3 different bitmaps and then create a TileGrid object for each of them.

```
my_display_group = displayio.Group()

image_1, palette_1 = adafruit_imageload.load(
    "images/page_1.bmp", bitmap=displayio.Bitmap, palette=displayio.Palette
)
tile_grid_1 = displayio.TileGrid(image_1, pixel_shader=palette_1)
my_display_group.append(tile_grid_1)

image_2, palette_2 = adafruit_imageload.load(
    "images/page_2.bmp", bitmap=displayio.Bitmap, palette=displayio.Palette
```

```
)
tile_grid_2 = displayio.TileGrid(image_2, pixel_shader=palette_2)

image_3, palette_3 = adafruit_imageload.load(
    "images/page_3.bmp", bitmap=displayio.Bitmap, palette=displayio.Palette
)
tile_grid_3 = displayio.TileGrid(image_3, pixel_shader=palette_3)
```

If you want to change what buttons do what, this is the place to change it. The lists are organized so every value is in the same row and column visually as it is on the display. `page_x` stores what functions are run when that part of the screen is pressed and `page_x_vals` stores the values to be passed to those functions when they are run.

For example, say I wanted to open Netflix when I pressed in the upper right-hand corner on the first page, I'd change the third value of `page_1` to `openchannel` and change the third value of `page_1_vals` to `12`.

```
page_1 = [sendkey, sendkey, sendkey,
          sendkey, sendkey, sendkey,
          sendkey, sendkey, sendkey,
          sendkey, sendkey, switchpage]

page_1_vals = ["Back", "Up", "Home",
               "Left", "Select", "Right",
               "Search", "Down", "Info",
               "Rev", "Play", (2, tile_grid_2)]

# Page 2
page_2 = [openchannel, openchannel, openchannel,
          openchannel, openchannel, openchannel,
          openchannel, openchannel, openchannel,
          switchpage, sendkey, switchpage]

page_2_vals = [14362, 2285, 13,
               12, 8378, 837,
               38820, 47389, 7767,
               (1, tile_grid_1), "Home", (3, tile_grid_3)]

page_3 = [None, None, None,
          sendkey, None, sendkey,
          sendkey, sendkey, sendkey,
          switchpage, sendkey, sendkey]

page_3_vals = [None, None, None,
               "Search", None, "Info",
               "Rev", "Play", "Fwd",
               (2, tile_grid_2), "Back", "Home"]
```

The first two lists in the following lines of code contain the lists discussed in the previous section. This just makes it easier to switch between the sets of commands based on what page is active.

The next two lines set `page` to `page_1` and `page_vals` to `page_1_vals`. This makes the first page open by default. These variables are used to store the function and value lists for the active page.

After that, the display starts showing the tile grid, and we print ready. The latter statement is really only useful for when you have a serial terminal open.

```
pages = [page_1, page_2, page_3]
pages_vals = [page_1_vals, page_2_vals, page_3_vals]

page = page_1
page_vals = page_1_vals

board.DISPLAY.root_group = my_display_group
print("READY")
```

## Main loop

```
last_index = 0
while True:
    p = ts.touch_point
    if p:
        x = math.floor(p[0] / 80)
        y = abs(math.floor(p[1] / 80) - 2)
        index = 3 * x + y
        # Used to prevent the touchscreen sending incorrect results
        if last_index == index:
            if page[index]:
                # pylint: disable=comparison-with-callable
                if page[index] == switchpage:
                    page, page_vals = switchpage(page_vals[index])
                else:
                    page[index](page_vals[index])
                time.sleep(0.1)

        last_index = index
```

Right before the main loop starts, `last_index` is set to 0. `last_index` is used so that you need to press on the button for at least two loops for it to activate. This barely affects responsiveness and drastically decreases the chance of an incorrect read.

Then, the loop is started, and we define `p` as where the Touchscreen object, `ts` is detecting a press.

```
last_index = 0
while True:
    p = ts.touch_point
```

If `p` is `None`, so if there aren't any detected presses, the loop simply starts again.

However, if `p` is not `None`, so if the board is detecting the touch screen is being pressed, it continues to the next section of the loop.

To make things easier on myself, I simply converted the touch screen x and y values ranging from 0-320 and 0-240, respectively, to values ranging from 0-3 in the x-direction and 0-2 in the y-direction. I reverse y by taking the absolute value of the value ranging from 0-2 and subtracting 2.

Now, `index` is created. The `3*x` is how many rows up to go, since each row has 3 values in it, and the `y` is how many columns over to go.

```
if p:
    x = math.floor(p[0] / 80)
    y = abs(math.floor(p[1] / 80) - 2)
    index = 3 * x + y
```

The last section of the loop runs the commands from the list. I've added line numbers so it will be easier to explain.

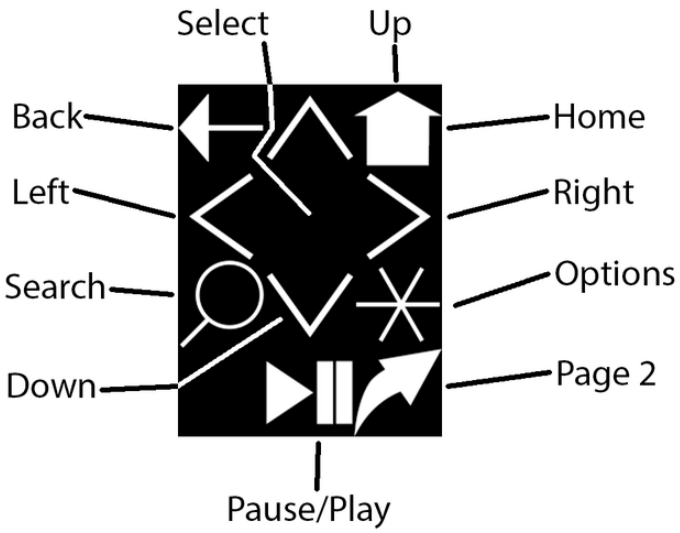
line 1 just verifies that the current button has been pressed twice without a different one being pressed. If that is true, we then check if that button even has a function connected to it. If it doesn't, the loop restarts. Then, on line 4, we see if the user is intending to switch the page, since that function returns values that need to be assigned to variables, unlike the other functions. If that is not the case, then we simply run the function in `page[index]` and pass it `page_vals[index]`. If a button has been pressed, we wait 0.1 seconds so you don't accidentally press it more than intended, and then set `last_index` to the current `index`.

```
if last_index == index:
    if page[index]:
        # pylint: disable=comparison-with-callable
        if page[index] == switchpage:
            page, page_vals = switchpage(page_vals[index])
        else:
            page[index](page_vals[index])
        time.sleep(0.1)

    last_index = index
```

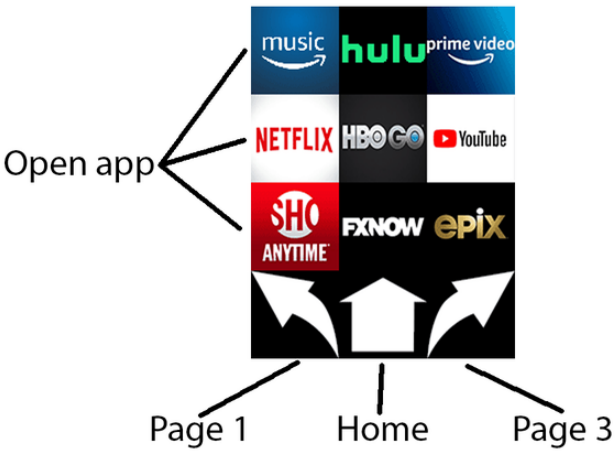
# UI & Customization

## UI

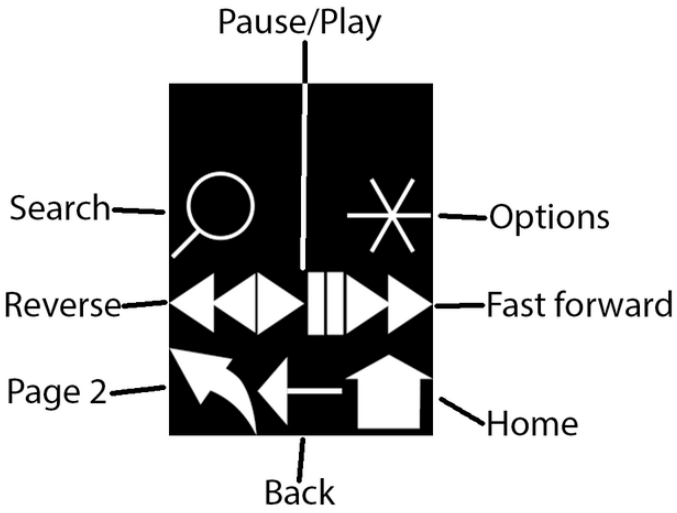
Page 1:




Page 2:



Page 3:




## Customization

The rest of this section is intended for advanced users and assumes a decent amount of knowledge of their photo editing application of choice. Please read this learn guide page (https://adafru.it/MbZ) about indexing bitmaps in GIMP before continuing.

I'll go through the process I used to make the bitmaps in case you'd also like to do that. I've included all of the images I used for this in different formats in the project zip.

First, I used the ECP to get the list of all the apps installed on the device. The lines below were run on Linux. Powershell also has `curl`, although it's just an alias of `Invoke-RestMethod`, so you might need to use a slightly different syntax.

```
$ curl http://192.168.1.3:8060/query/apps
```

In my case, that command returned this XML:

```
dherrada@dherrada-MS-7C37:~$ curl http://192.168.1.3:8060/query/apps
&lt;?xml version="1.0" encoding="UTF-8" ?&gt;
&lt;apps&gt;
    &lt;app id="31012" type="menu" version="1.9.33"&gt;FandangoNOW Movies &amp;
TV&lt;/app&gt;
    &lt;app id="12" type="appl" version="4.2.81176083"&gt;Netflix&lt;/app&gt;
    &lt;app id="2285" type="appl" version="6.32.3"&gt;Hulu&lt;/app&gt;
    &lt;app id="8378" type="appl" version="4.12.4"&gt;HBO GO&lt;/app&gt;
    &lt;app id="38820" type="appl" version="2.15.2"&gt;Showtime Anytime&lt;/app&gt;
    &lt;app id="7767" type="appl" version="131.0.950"&gt;EPIX&lt;/app&gt;
    &lt;app id="47389" type="appl" version="4.6.158"&gt;FXNOW&lt;/app&gt;
    &lt;app id="65067" type="appl" version="3.9.27"&gt;STARZ&lt;/app&gt;
```

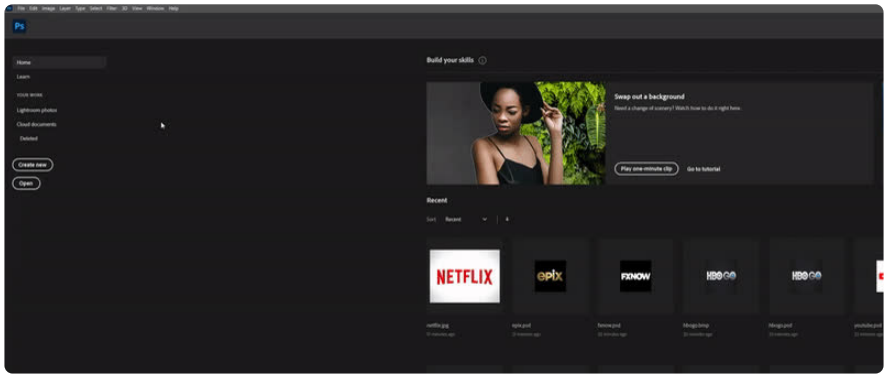
```
    &lt;app id="13" type="appl" version="11.4.2020060412"&gt;Prime Video&lt;/app&gt;
    &lt;app id="63344" type="appl" version="3.4.140"&gt;Comedy Central&lt;/app&gt;
    &lt;app id="68669" type="appl" version="7.11.0"&gt;NBC&lt;/app&gt;
    &lt;app id="73376" type="appl" version="4.6.158"&gt;ABC&lt;/app&gt;
    &lt;app id="552944" type="appl" version="1.2.55"&gt;Roku Tips &amp; Tricks&lt;/
app&gt;
    &lt;app id="65978" type="appl" version="2.10.2002191329"&gt;CNNgo&lt;/app&gt;
    &lt;app id="12716" type="appl" version="2.10.0"&gt;AMC&lt;/app&gt;
    &lt;app id="151908" type="appl" version="2.9.42"&gt;The Roku Channel&lt;/app&gt;
    &lt;app id="46041" type="appl" version="6.13.248"&gt;Sling TV&lt;/app&gt;
    &lt;app id="837" type="appl" version="1.0.80000286"&gt;YouTube&lt;/app&gt;
    &lt;app id="28" type="appl" version="5.3.50"&gt;Pandora&lt;/app&gt;
    &lt;app id="14362" type="appl" version="2.0.39"&gt;Amazon Music&lt;/app&gt;
&lt;/apps&gt;
```

Let's say that we wanted the icon for Netflix. Netflix has an ID of 11, so that's what I'll use to specify I'm looking for Netflix's icon.

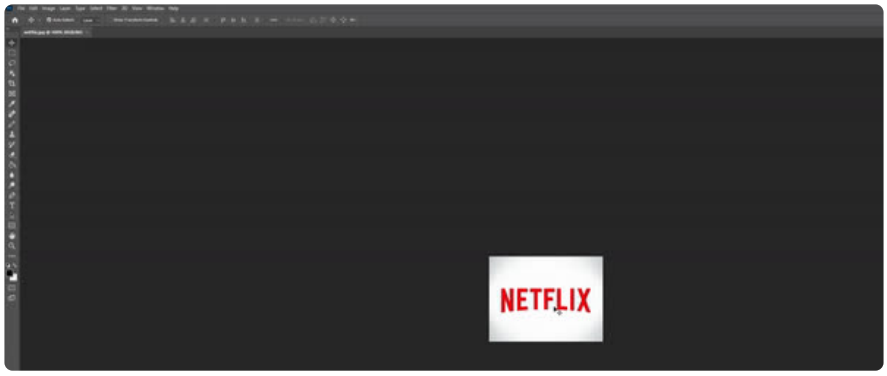
```
$ curl http://192.168.1.3/query/icon/12 --output netflix.jpg
```

I did this for all the app icons I wanted and then transferred them to my Windows partition and opened them in Photoshop, although you can also use GIMP for this.

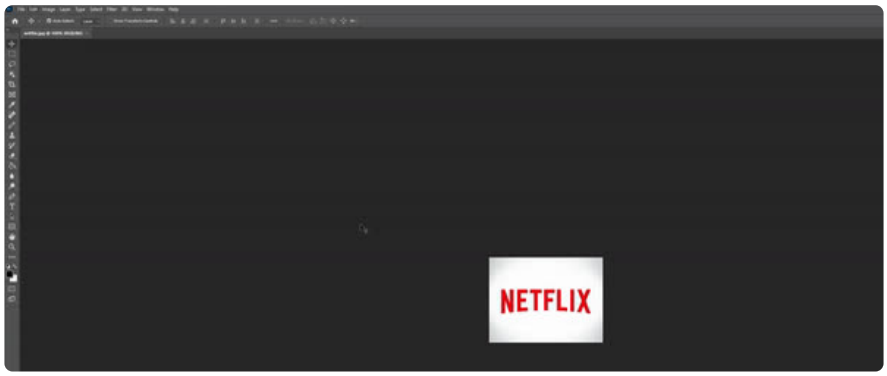
First, open the image.




Then, you may need to try to move the image around, it should give you a pop-up asking to convert it to a normal layer, click 'Convert to Normal Layer.' Make sure to not actually move the image around.

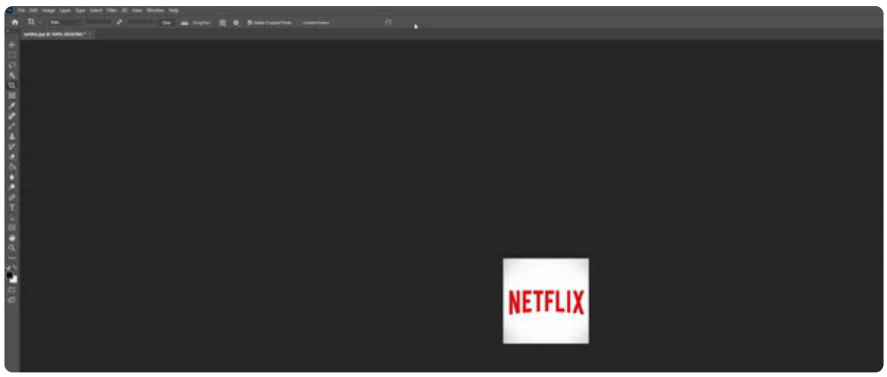


Press **CTRL+A** and click **Edit->Free Transform** or **CTRL+T**. Squish the image on the sides a bit. You'll get a feel for it, but in this case I went for 240 pixels wide.



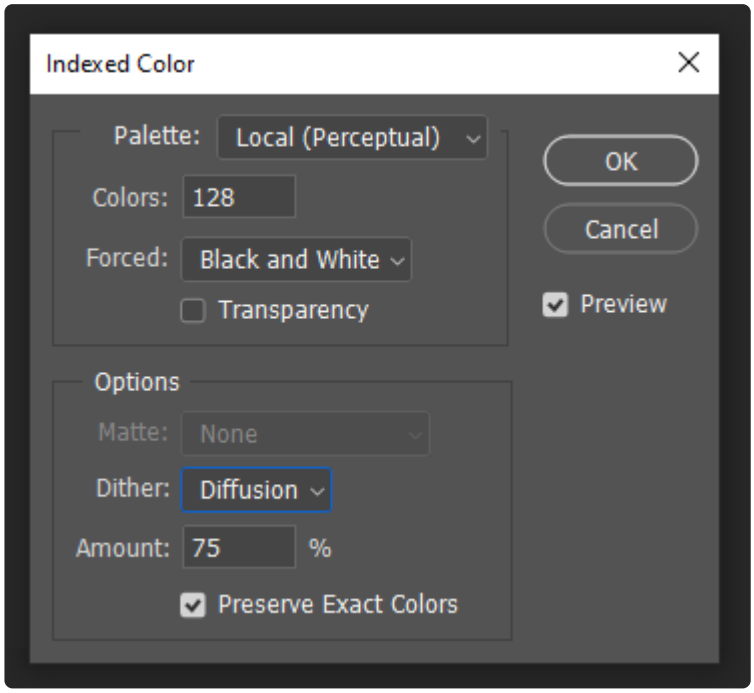

Now, select the crop tool and crop the image to 218 pixels wide.

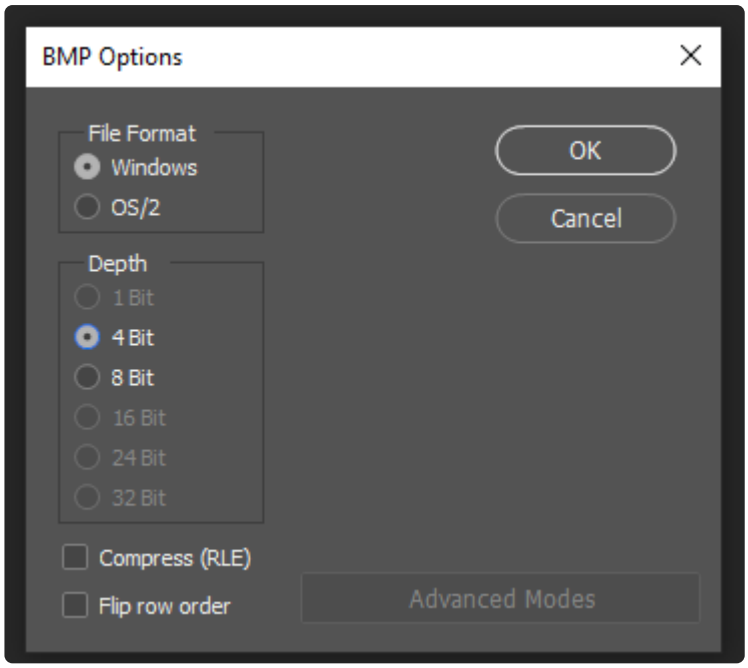

Click **Image->Image Size** and change the width and height to 80 pixels.




Now, do this for every image you want and copy them to your page, rotating them -90 degrees.

After that, convert it to indexed color through I**mage->Mode->Indexed Color** and use the settings shown below. Save the image as a bmp using the smallest bits of color you can (usually 4-bit or 8-bit).






If you're running out of memory once you've put your bitmaps on the device, then they're probably too big. I'd recommend converting them to a smaller bit depth here. (https://adafru.it/Mb-) On images that are just black and white, it usually looks ok to just use 2 colors, which lets you have a very small file.