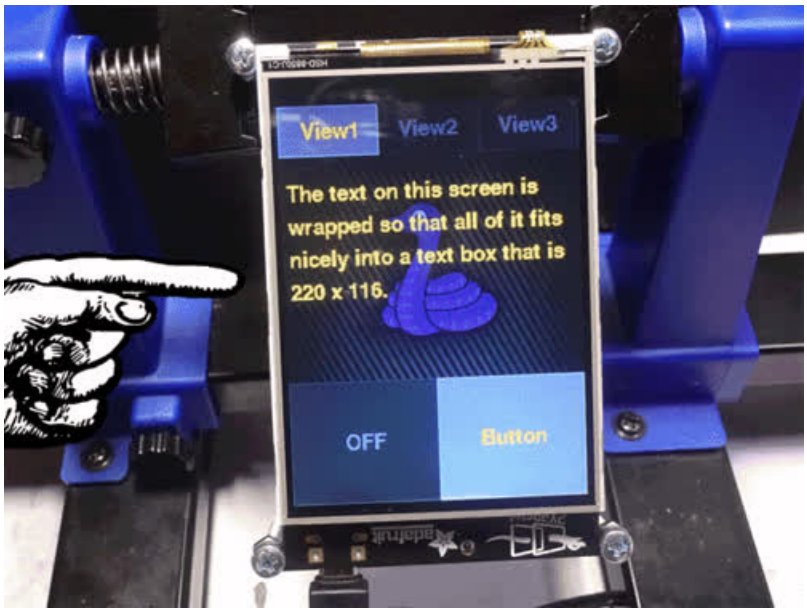
adafruit learning system

Making a PyPortal User Interface with DisplayIO

Created by Richard Albritton



Last updated on 2020-06-09 01:38:44 PM EDT

Overview

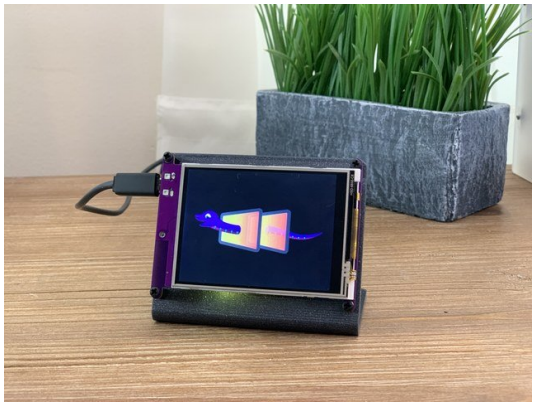
The PyPortal is the perfect device to interact with internet of things (IoT) projects. Perfect when you need something that's a bit more than a few buttons on a microcontroller, but a computer or smartphone app would be too much. So you need some sort of simple user interface that can handle buttons, lots of text, icons, and even a few views.

In this guide we will cover a few steps that will make it easy to build a small User Interface in CircuitPython (https://adafru.it/EFq), the easy to use Python implementation for microcontrollers and single board computers (SBC). You may do things like:

- How to change the display orientation.
- Using Groups like windows that can be turned on or off.
- Creating image placeholders that let you easily change the background and icons.
- Display dynamic text with word wrapping.
- Create buttons for navigation and in unique windows.

This process is known to work with the PyPortal and PyPortal Pynt. The PyPortal Titano uses a different display hardware.

Parts



Adafruit PyPortal - CircuitPython Powered Internet Display

OUT OF STOCK

Out Of Stock

USB cable - USB A to Micro-B

$2.95

IN STOCK

Add To Cart

Loading CircuitPython

Mu Editor

The Mu editor works really well with the PyPortal and it is the recommended choice for editing CircuitPython code.

https://adafru.it/HA8

https://adafru.it/HA8

Updating Your PyPortal

This project was setup and tested using CircuitPython version 5 or higher. You will want to update your PyPortal and Libraries to match the version you are using.

There are separate CircuitPython versions for PyPortal, Pyportal Titano, and PyPortal Pynt. Use the version specific to your board.

Use the blue button to see if you need to install or update the CircuitPython installation for your display.

https://adafru.it/EnM

https://adafru.it/EnM

CircuitPython Library Installation

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx) matching your version of CircuitPython. PyPortal requires at least CircuitPython version 5.

https://adafru.it/ENC

https://adafru.it/ENC

CircuitPython Library files used in this tutorial

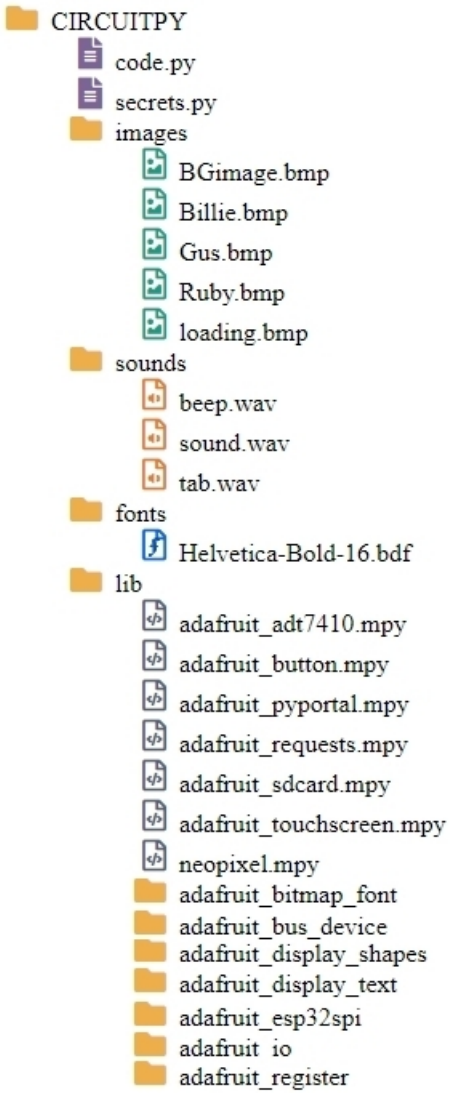
Plug your PyPortal into your computer via a known, good USB data + power cable (not the kind that comes with USB battery packs, those are power only). A new flash drive should show up in your computer's file explorer/finder named CIRCUITPY. If it's not there, check your cable and that you installed CircuitPython correctly earlier.

The following libraries are used here. Copy the corresponding file from the library bundle to your CIRCUITPY drive in a subdirectory named lib. Create this subdirectory, if necessary, then copy these files/directories:

- **adafruit_adt7410.mpy**
- **adafruit_bitmap_font**
- **adafruit_bus_device**
- **adafruit_button.mpy**
- **adafruit_display_shapes**
- **adafruit_display_text**
- **adafruit_esp32spi**

- adafruit_io
- adafruit_pyportal.mpy
- adafruit_register
- adafruit_requests.mpy
- adafruit_sdcard.mpy
- adafruit_touchscreen.mpy
- neopixel.mpy

Before continuing make sure your board's **lib** folder has the following files and folders copied over. Note that the non-library files will be retrieved in the page called "The Full Code".

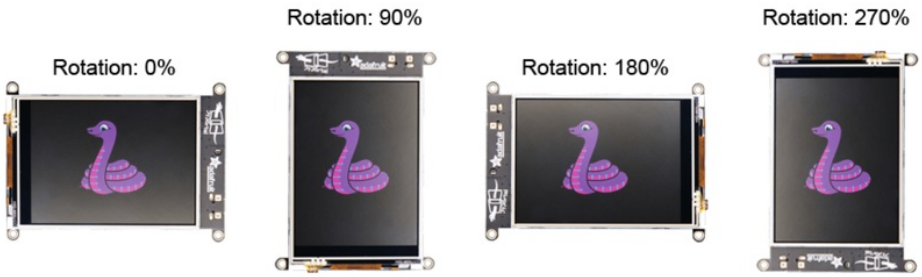


UI Elements

There are many parts that are common in a good user interface, and the PyPortal is powerful enough to do some cool stuff. So let's go over a few things that you may want to think about doing with your UI.

Orientation

Chances are that your UI layout will be greatly affected by what orientation your PyPortal display is. If you have looked around at most of the guides for the PyPortal you may have noticed that most of them are using the default horizontal orientation with the USB plug on the right side. Well, that is not a limitation and this guide will show you how to easily use your display in any orientation.



Windows and Layers

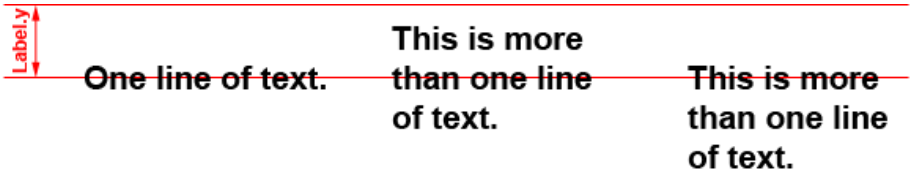
The PyPortal can do quite a lot and at some point you may want to take full advantage of what it can do, and have multiple display windows to show more information and options. Using the CircuitPython Displayio library will allow you to use groups like layers that can be updated, hidden, and shown on the fly.

Backgrounds and icons

Who wants a simple colored background when you can use a full color image that can be changed at any time. Why not mix in some icons that can be switched out using code. This guide will show you an easy way to set up image placeholders that will make it easy.

Dynamic Text updates

If you are connected to an API or MQTT, chances are that you are sending text-based information to be displayed on the PyPortal. We will talk about a few ways to format that text so that the words wrap into a text box.



All the Buttons

Navigation buttons are a key part to a good UI. You will learn how to use them not only for display navigation, but also

as elements on a particular layers. We will also go over how to make buttons that simulate toggle groups, momentary, and latching switches.

Beeps and Boops

Sure, you can see that a button was pressed on the screen. You may also add a satisfying sound to button presses as well. It is super easy to do, and takes your UI to the next level.

Display

Setting up the Display

We will be using the `board.DISPLAY` option so that we have a lot of control over the way our UI is rendered, like screen orientation as well as the ability to turn elements of our UI on or off.

To use `board.DISPLAY`, you will need the following code in your CircuitPython program.

```
display = board.DISPLAY
```

For this project we are using `board.DISPLAY` so that all of the critical display settings are loaded for the CircuitPython device we are using. In this case, we are loading the configuration setup for the PyPortal into `displayio.Display()`.

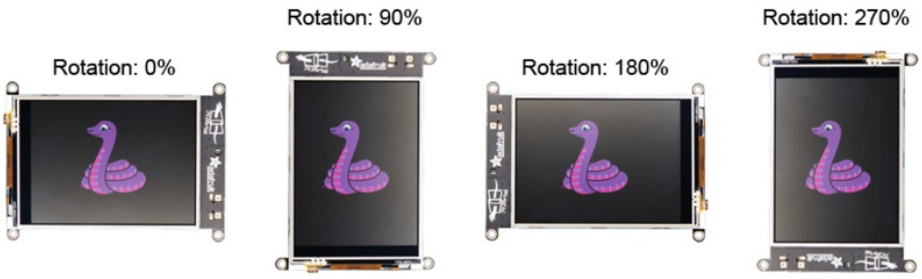
Aside from the main display setup we have a lot of options that we can now set. The two main options we will look at are `rotation` and `backlight`.

Screen Rotation

This will let you change the orientation of the display in increments of 90°. So you can also do a user interface in portrait or landscape mode.

To do this, use the following code.

```
display = board.DISPLAY
display.rotation=270
```



Rendering the Display

Eventually, we will want to have the screen display the content, and to do that we will use the following command:

`board.DISPLAY.show(splash)`

This will set a Group named **splash** as the source of information to be displayed. We will get into Groups on the next section.

The Backlight

You can also adjust the brightness of the display backlight using `board.DISPLAY`. This simple function will allow you to change the brightness easily.

```
# Backlight function
# Value between 0 and 1 where 0 is OFF, 0.5 is 50% and 1 is 100% brightness.
def set_backlight(val):
    val = max(0, min(1.0, val))
    board.DISPLAY.auto_brightness = False
    board.DISPLAY.brightness = val
```

Just use `set_backlight(0.3)` to set the backlight to 30% brightness.

If you want to see more stuff that you can do with `displayio.Display()` have a look at the Display section of CircuitPython docs.

https://adafru.it/EFy

https://adafru.it/EFy

The Touchscreen

The PyPortal has a built-in recessive touchscreen that we will use for button navigation in our UI. The following code will set up the basic touchscreen.

```
# Touchscreen setup
# ------Rotate 270:
screen_width = 240
screen_height = 320
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_XL, board.TOUCH_XR,
                                      board.TOUCH_YD, board.TOUCH_YU,
                                      calibration=((5200, 59000), (5800, 57000)),
                                      size=(screen_width, screen_height))
```

Rotating the Touchscreen

One issue that we will run into if we rotate the Display for the PyPortal is that this will change the orientation of the display, but the touchscreen coordinates remain the same. Since we will be using the touch screen to detect button presses this is a big issue. Thankfully, we can change the way the touchscreen reads its coordinates by moving the `x1_pin`, `x2_pin`, `y1_pin`, and `y2_pin` around in addition to switching the size x and y to go from horizontal to vertical format.

To save you some time, here are the touchscreen settings you will want to use if the screen is rotated 0°, 90°, 180°, or 270°

```
# -------Rotate 0:
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_XL, board.TOUCH_XR,
                                      board.TOUCH_YD, board.TOUCH_YU,
                                      calibration=((5200, 59000), (5800, 57000)),
                                      size=(320, 240))

# -------Rotate 90:
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_YU, board.TOUCH_YD,
                                      board.TOUCH_XL, board.TOUCH_XR,
                                      calibration=((5200, 59000), (5800, 57000)),
                                      size=(240, 320))

# ------Rotate 180:
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_XR, board.TOUCH_XL,
                                      board.TOUCH_YU, board.TOUCH_YD,
                                      calibration=((5200, 59000), (5800, 57000)),
                                      size=(320, 240))

# ------Rotate 270:
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_YD, board.TOUCH_YU,
                                      board.TOUCH_XR, board.TOUCH_XL,
                                      calibration=((5200, 59000), (5800, 57000)),
                                      size=(240, 320))
```

Just replace the touchscreen settings with whatever ones you would like to use.

Here is the code you would need to set the PyPortal up in 270° orientation.

```
pyportal = PyPortal()
display = board.DISPLAY
display.rotation = 270

# Backlight function
# Value between 0 and 1 where 0 is OFF, 0.5 is 50% and 1 is 100% brightness.
def set_backlight(val):
    val = max(0, min(1.0, val))
    board.DISPLAY.auto_brightness = False
    board.DISPLAY.brightness = val

# Set the Backlight
set_backlight(0.3)

# Touchscreen setup
# ------Rotate 270:
screen_width = 240
screen_height = 320
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_YD, board.TOUCH_YU,
                                      board.TOUCH_XR, board.TOUCH_XL,
                                      calibration=((5200, 59000),
                                                   (5800, 57000)),
                                      size=(screen_width, screen_height))
```

Note that we have added PyPortal to the project even though we are not using it for Display, there are some other things that we will use it for later.

For more information about Touchscreen, click the button below.

https://adafru.it/lsA

https://adafru.it/lsA

Groups

Groups allow you to manage the elements that are rendered on the display. You can think of them like layers that can be removed or displayed. You can even have a Group inside of another Group and that is exactly what we are going to do but first we need to set a few things up.

Define a Group

To define a display group we want to use the `displayio.Group()` function. This function requires you to set a `max_size` that puts a limit to how many objects that your group can hold. So if I know that I will only be using two buttons and a text box for one group, I would use `displayio.Group(max_size=3)`. If I then added a fourth object to that group, I would get an error message telling me that the Group is already full. So you may want to add a few more spots that you need for your group

Here is an example of some groups that we would want to setup for our UI.

```
# ------------- Display Groups ------------- #
splash = displayio.Group(max_size=10) # The Main Display Group
view1 = displayio.Group(max_size=15, x=0, y=40) # Group for View 1 objects
view2 = displayio.Group(max_size=15, x=0, y=40) # Group for View 2 objects
view3 = displayio.Group(max_size=15, x=0, y=40) # Group for View 3 objects
```

This sets up one Main display group called **splash** and three other groups that will be used to display selectable tab views. Notice how we have set the view groups to be moved down 40 pixels. This is so that these groups will start their display coordinates from x=0, y=0 just underneath where our Tab buttons will be rendered. It will just make the layout of those pages a little easier to manage.

Adding to a Group

Now that we have a few groups, we will want to assign them to our splash group so that they can be part of the display image. This is done by using `.append()` to add you object to the list of objects in the group.

Here is how it looks to add our three view groups to our main group:

```
splash.append(view1)
splash.append(view2)
splash.append(view3)
```

Later, we will be adding objects like buttons, text, and images to these Groups.

Other helpful functions

here are a few other helpful functions that we will be using to manage our groups.

- **pop()** - This is basically used to delete a Group. We will use this later for switching Icon images.
    - ***group.pop()*** - This deletes the first element in group.
    - ***group.pop(2)*** - This deletes the third element in group.
    - ***view3.pop(-2)*** - This deletes the second to last element in the **view3** Group.
- **remove(i)** - This is used to remove a Group or Object from a Group. It will not delete the Object or Group, but it can be used to temporarily remove that element for the display.

- *group.remove(object)* - This will remove the element **object** from group
- *splash.remove(view2)* - This will remove the **view2** Group from the **splash** Group

For CircuitPython 5 and greater, there is a new function called `hidden` and it can be used to hide a Group or Object in a Group.

- **hidden** = **bool** - Set this to `True` to hide all elements of a group.
  - *splash[4].hidden = True* - This will hide the 5th Group or Object in the **splash** Group.
  - *splash[4].hidden = False* - This will show the 5th Group or Object in the **splash** Group.

While this function will hide something from view, it will not always bring it back unless the display is refreshed. For this reason, we will be using another method for showing and hiding Groups.

Hide and Show a Group

The biggest reason we want to use Groups is that we can use them like Layers to show and hide making it possible to have multiple screen views. For this example we are going to use buttons that will let us switch between **view1**, **view2**, and **view3**. This will be done by showing one of those pages while hiding the other two. To make this easy, and because we will be doing it a lot, we will make a function for hiding a group and another for showing a group.

```
# ------------- Layer Functions ------------- #
# Hide a given Group by removing it from the main display Group.
def hideLayer(i):
    try:
        splash.remove(i)
    except ValueError:
        pass
# Show a given Group by adding it to the main display Group.
def showLayer(i):
    try:
        splash.append(i)
    except ValueError:
        pass
```

We will go over how this function gets used in the Usage section.

For more information on Groups, check out the link below for the CircuitPython documentation.

https://adafru.it/EFx

https://adafru.it/EFx

Images

The Basics of displaying an image

To display an image you first need an image that is in Bitmap format and 24-bits or less. To keep things organized you will want to keep your Bitmaps in a folder called **images** on your **CIRCUITPY** drive.

Displaying an Image

Loading an image takes a few steps involving `open()`, `displayio.OnDiskBitmap()`, `displayio.TileGrid()`, `displayio.Group()`, and `board.DISPLAY.show()`. So here is what it looks like for loading a small image icon:

```
group = displayio.Group(max_size=1)
group.x = 100
group.y = 120

image_file = open("/images/Gus.bmp", "rb")
image = displayio.OnDiskBitmap(image_file)
image_sprite = displayio.TileGrid(image, pixel_shader=displayio.ColorConverter())

group.append(image_sprite)
board.DISPLAY.show(group)

while True:
    pass
```

So here is what happened there.

1. The image can not be displayed unless it is in a Group so we set one up first.
2. Then we setup the file using `open()` in **read binary**, ***"rb"***, mode.
3. Next, we can load the file using `displayio.OnDiskBitmap()`
4. Move the now loaded file data into the TileGrid.
5. Add the newly made sprite into our Group.
6. Then we can display the group.

The TileGrid basically holds all of the image data so it can be displayed. It is mostly used for sprites in games because it can break up and rearrange an image that has already been loaded into the memory. We are simply going to use TileGrid as a container for our image though.

Removing an Image

So now that we loaded an image we also need to know how to upload it so we can replace it with another image. This is done using `group.pop()` to delete the image from the group so that a new one can be loaded.

```
group.pop()
```

So all together you can see that there are a few steps involved with displaying images. Now let us go over how we will be using this process to display and change a few images in our UI.

How we will use Images for the UI

The Startup Screen

It may take a few seconds to load all of the elements of your UI so why not start with a Loading screen?

To do this we will be using the set_background() function from the PyPortal library. This will let us easily load a full screen image onto the PyPortal while making it easy to replace that image later with our UI.

```
# Display an image until the loop starts
pyportal.set_background('/images/loading.bmp')
```

That is it and the image will be displayed. However, the reason we will not use this for the rest of our code is because this function is just for loading one image into the background and we can not add it to the groups that we will be using later.

The Background Image

We will be using another fullscreen image as the background for our user interface but since that will have other elements loaded on top of it we will want to use Groups.

So lets set up a Group to hold the background then add it to our main display Group **splash**.

```
bg_group = displayio.Group(max_size=1)
splash.append(bg_group)
```

Notice that `bg_group` can only hold 1 object because we set `max_size` to `1`. This is because we only ever want one image for our background at a time. Later we will see how that one image could be changed at any time without tanking our memory.

Other Image Display Groups

Now we need to make another Group to hold an Icon image that we will be changing via the UI. This one will also set the x and y position and add it to one of our Views rather than the main display Group.

```
icon_group = displayio.Group(max_size=1)
icon_group.x = 180
icon_group.y = 120
icon_group.scale = 1
view2.append(icon_group)
```

Just like `bg_group` this will only hold one object so we do not bog down the system memory by loading to many images.

Adding this Group to the **view2** Group rather than **splash** will make it so that the Icon can be visible or not visible just by showing or hiding the **view2** Group without affecting the rest of the display.

> *The `icon_group.scale = 1` part of the code is not needed unless you wanted to make the image bigger. Although it just makes the pixels of the image bigger so a scale of 2 would make each pixel 2x2 and so on. So images become more blocky and it cannot be used to make an image smaller than a scale of 1.*

Set Image Function

So do you remember from the top of this page how many steps it took to load and unload an image? Well to make it easy for us to deal with switching images, we will be using a function to do all the load and unload BMP image stuff. This will help to keep our memory free as well as making it very easy to switch large and small images in the loop by just using `set_image(Group, filename)`.

The following code will need to be added to your code.py file.

```
# This will handel switching Images and Icons
def set_image(group, filename):
    """Set the image file for a given goup for display.
    This is most useful for Icons or image slideshows.
        :param group: The chosen group
        :param filename: The filename of the chosen image
    """
    print("Set image to ", filename)
    if group:
        group.pop()

    if not filename:
        return  # we're done, no icon desired
    try:
        if image_file:
            image_file.close
    except NameError:
        pass
    image_file = open(filename, "rb")
    image = displayio.OnDiskBitmap(image_file)
    try:
        image_sprite = displayio.TileGrid(image, pixel_shader=displayio.ColorConverter())
    except TypeError:
        image_sprite = displayio.TileGrid(image, pixel_shader=displayio.ColorConverter(), position=
(0,0))
    group.append(image_sprite)
```

We will talk about how to use this function in the Usage section.

If you would like to see more information on loading images with CircuitPython, check out these links:

https://adafru.it/EFt

https://adafru.it/EFt

https://adafru.it/EFw

https://adafru.it/EFw

Text Box

Adding a Font

To display text we first need to load a Bitmap Font from our fonts folder:

```
# Set the font and preload letters
font = bitmap_font.load_font("/fonts/Helvetica-Bold-16.bdf")
font.load_glyphs(b'abcdefghjiklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890- ()')
```

You can have as many fonts as you want so long as you have a Bitmap version of each one.

Next we will preload the most common letters so that we won't have to wait so long for our text to display.

https://adafru.it/EFI

https://adafru.it/EFI

Setting up the text blocks

To display text we will be using Label form the Adafruit Display Text library. This will make it very easy for us to update text in our various groups on the fly.

First we want to set up some standard values like the x an y coordinates for the top left of our text boxes.

```
# Default Label styling:
TABS_X = 5
TABS_Y = 50
```

Next we will set up each of the text boxes that we will be updating text on in our many Groups.

Here is what that looks like:

```
# Text Label Objects
feed1_label = Label(font, text="Text Window 1", color=0xE39300, max_glyphs=200)
feed1_label.x = TABS_X
feed1_label.y = TABS_Y
view1.append(feed1_label)

feed2_label = Label(font, text="Text Window 2", color=0xFFFFFF, max_glyphs=200)
feed2_label.x = TABS_X
feed2_label.y = TABS_Y
view2.append(feed2_label)

sensors_label = Label(font, text="Data View", color=0x03AD31, max_glyphs=200)
sensors_label.x = TABS_X
sensors_label.y = TABS_Y
view3.append(sensors_label)

sensor_data = Label(font, text="Data View", color=0x03AD31, max_glyphs=100)
sensor_data.x = TABS_X+15
sensor_data.y = 170
view3.append(sensor_data)
```

Each Label is declared, moved into position and added to their Group. Note that we have two Labels that will be added to the view3 Group. This is because one of the Labels will be updated rapidly with sensor data and we do not want to redraw all of the text for updates.

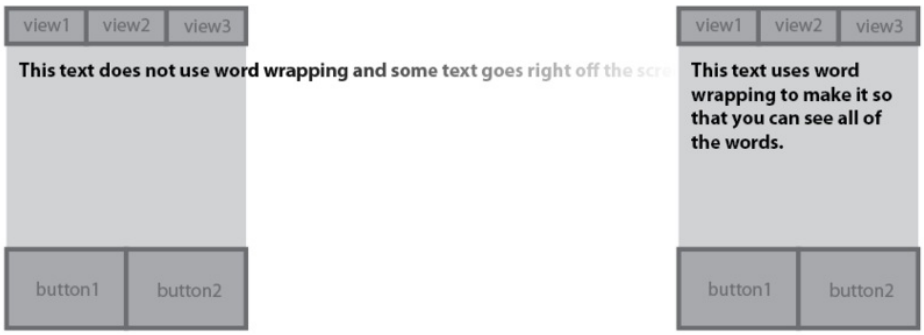
For more information about Label and Display Text, click the link below.

https://adafru.it/IDE

https://adafru.it/IDE

Word-wrap Function

Unfortunately, Label does not support word wrapping yet but we can use the `PyPortal.wrap_nicely()` function to reformat a string and add line breaks to wrap text so we can see it all. Now `PyPortal.wrap_nicely()` will do most of the hard work by breaking our text up into an array of text lines, but we still need to add line breaks at the end of these to make the words wrap and reposition the text so that it is aligned to the top.



Add this function to your code and it will handle updating all of our multiline text.

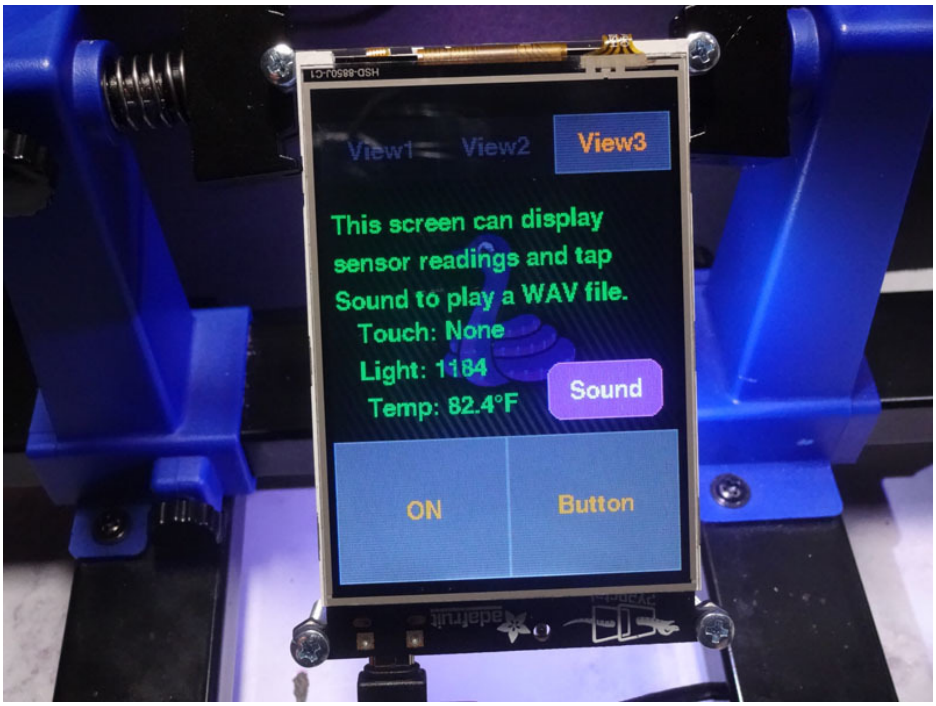
```
# Used to calculate vertical text height for Top Alignment
text_hight = Label(font, text="M", color=0x03AD31, max_glyphs=10)

# return a string with word wrapping using PyPortal.wrap_nicely
def text_box(target, top, max_chars, string):
    text = pyportal.wrap_nicely(string, max_chars)
    new_text = ""
    test = ""
    for w in text:
        new_text += '\n'+w
        test += 'M\n'
    text_hight.text = test
    glyph_box = text_hight.bounding_box
    print(glyph_box[3])
    target.text = "" # Odd things happen without this
    target.y = round(glyph_box[3]/2)+top
    target.text = new_text
```

We will go over how to use this function on the Usage section of the guide.

Format Specification Mini-Language

So let's say you want to display a bit of text that has some sensor reading or other data types in the mix. Using .format() gives you a lot of options to assemble your text so here is an example of how that works.

`sensor_data.text = 'Touch: {}\nLight: {}\nTemp: {}°F'.format(touch, light, tempF)`



https://adafru.it/lbh

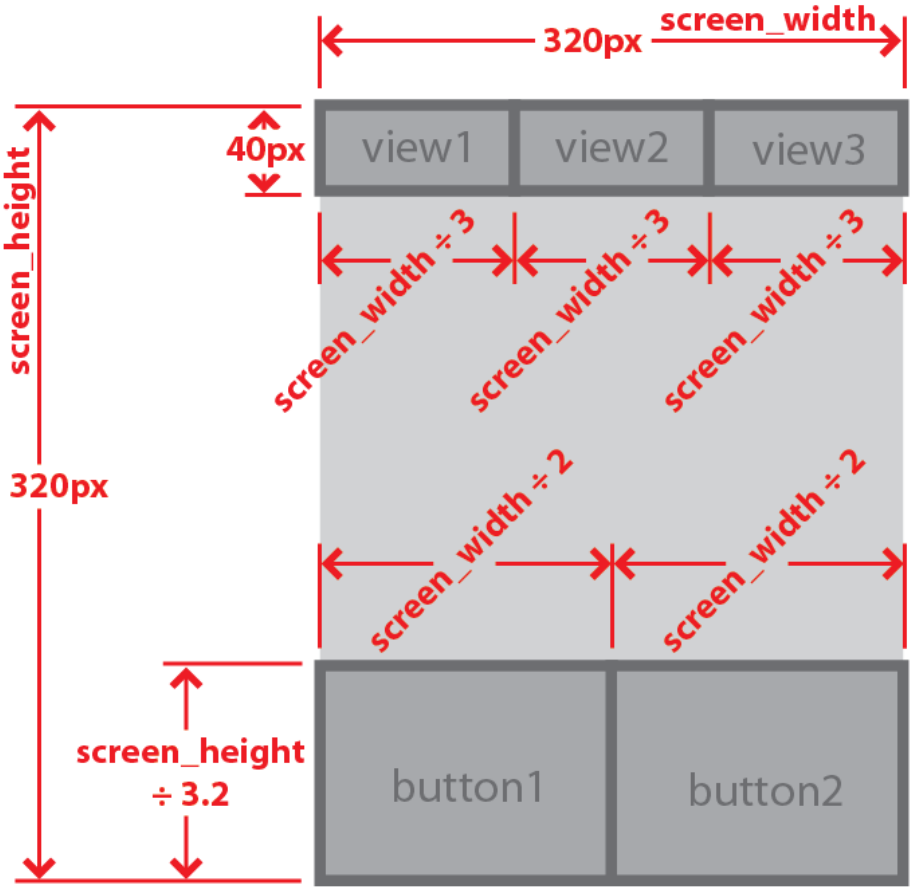
https://adafru.it/lbh

Buttons

Buttons are a group consisting of a Shape, Label, and touchscreen coordinates. So they take a bit to setup, but then they are rather easy to use.

Setup Some Styles

Just like with our text we will set up some variables that will help with the repetitive dimensions we will be using.




```
# ---------- Display Buttons ------------- #
# Default button styling:
BUTTON_HEIGHT = 40
BUTTON_WIDTH = 80

# We want three buttons across the top of the screen
TAPS_HEIGHT = 40
TAPS_WIDTH = int(screen_width/3)
TAPS_Y = 0

# We want two big buttons at the bottom of the screen
BIG_BUTTON_HEIGHT = int(screen_height/3.2)
BIG_BUTTON_WIDTH = int(screen_width/2)
BIG_BUTTON_Y = int(screen_height-BIG_BUTTON_HEIGHT)
```

Setup the Buttons

We will be setting up a few buttons that will always be visible to be used with our main UI. These buttons will be added to a group called `buttons` to make it easy to tell what button was pressed later. They will also be added to our `splash` Group so that they will be displayed with our main Group.

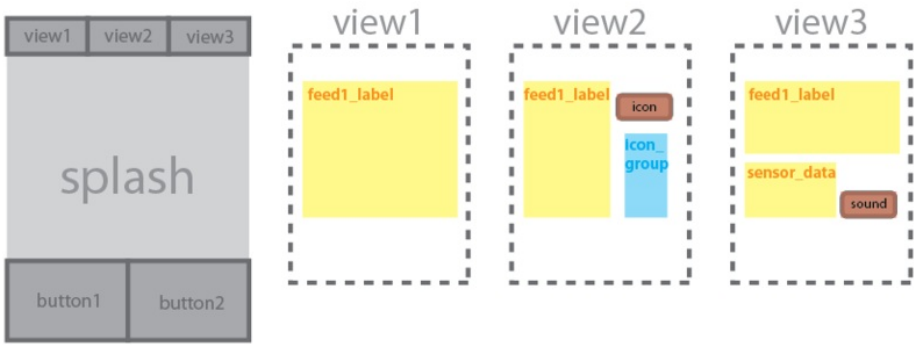
```
# This group will make it easy for us to read a button press later.
buttons = []

# Main User Interface Buttons
button_view1 = Button(x=0, y=0,
                      width=TAPS_WIDTH, height=TAPS_HEIGHT,
                      label="View1", label_font=font, label_color=0xff7e00,
                      fill_color=0x5c5b5c, outline_color=0x767676,
                      selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                      selected_label=0x525252)
buttons.append(button_view1)  # adding this button to the buttons group

button_view2 = Button(x=TAPS_WIDTH, y=0,
                      width=TAPS_WIDTH, height=TAPS_HEIGHT,
                      label="View2", label_font=font, label_color=0xff7e00,
                      fill_color=0x5c5b5c, outline_color=0x767676,
                      selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                      selected_label=0x525252)
buttons.append(button_view2)  # adding this button to the buttons group

button_view3 = Button(x=TAPS_WIDTH*2, y=0,
                      width=TAPS_WIDTH, height=TAPS_HEIGHT,
                      label="View3", label_font=font, label_color=0xff7e00,
                      fill_color=0x5c5b5c, outline_color=0x767676,
                      selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                      selected_label=0x525252)
buttons.append(button_view3)  # adding this button to the buttons group

button_switch = Button(x=0, y=BIG_BUTTON_Y,
                       width=BIG_BUTTON_WIDTH, height=BIG_BUTTON_HEIGHT,
                       label="Switch", label_font=font, label_color=0xff7e00,
                       fill_color=0x5c5b5c, outline_color=0x767676,
                       selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                       selected_label=0x525252)
buttons.append(button_switch)  # adding this button to the buttons group

button_2 = Button(x=BIG_BUTTON_WIDTH, y=BIG_BUTTON_Y,
                  width=BIG_BUTTON_WIDTH, height=BIG_BUTTON_HEIGHT,
                  label="Button", label_font=font, label_color=0xff7e00,
                  fill_color=0x5c5b5c, outline_color=0x767676,
                  selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                  selected_label=0x525252)
buttons.append(button_2)  # adding this button to the buttons group

# Add all of the main buttons to the spalsh Group
for b in buttons:
    splash.append(b.group)
```

You can see that there are many parameters for Button and this example uses them all. You do not have to set a color for the selection options, but the colors will invert when the button is pressed if you do not specify those colors.

Each Button is added to the button group and that that is used to add each Button to the `splash` Group.

Disappearing Buttons

There are two more Buttons that will be on separate view tabs and thus will be hidden at times. We do not want these Buttons to be added to the `splash` Group but we do want one on the `view2` Group and the other on `view3`.



We will still be adding these buttons to the button group though so that we can detect a button press for these just like the others.

```
# Make a button to change the icon image on view2
button_icon = Button(x=150, y=60,
                  width=BUTTON_WIDTH, height=BUTTON_HEIGHT,
                  label="Icon", label_font=font, label_color=0xffffff,
                  fill_color=0x8900ff, outline_color=0xbc55fd,
                  selected_fill=0x5a5a5a, selected_outline=0xff6600,
                  selected_label=0x525252, style=Button.ROUNDRECT)
buttons.append(button_icon) # adding this button to the buttons group

# Add this button to view2 Group
view2.append(button_icon.group)

# Make a button to play a sound on view2
button_sound = Button(x=150, y=170,
                  width=BUTTON_WIDTH, height=BUTTON_HEIGHT,
                  label="Sound", label_font=font, label_color=0xffffff,
                  fill_color=0x8900ff, outline_color=0xbc55fd,
                  selected_fill=0x5a5a5a, selected_outline=0xff6600,
                  selected_label=0x525252, style=Button.ROUNDRECT)
buttons.append(button_sound) # adding this button to the buttons group

# Add this button to view2 Group
view3.append(button_sound.group)
```

Note that these buttons were added to their display Group using `.group`. This is because a Button is a group of other objects.

Later we will get into how we can read these buttons being pressed on the Usage page.

There is a bit more information on Buttons at the link below.

https://adafru.it/FiB

https://adafru.it/FiB

Sounds

The Sound Files

The PyPortal can play WAV files rather easily using the onboard speaker or adding a larger speaker for a bigger sound. There are plenty of places to download WAV files or you can make your own and save them as a WAV.

You will want to place all of your sound files into a folder named sounds on your **CIRCUITPY** drive.

There is already a really good guide on how to make CircuitPython compatible audio files at the link below.

https://adafru.it/BvU

https://adafru.it/BvU

Playing Sound

The PyPortal library makes it very easy to play sounds using the `PyPortal.play_file()` function.

For this example we will be setting up three sound file locations so that we can easily change what sound we want to play. We will mostly be using sounds that relate to buttons being pressed in the UI.

```
# ---------- Sound Effects ------------- #
soundDemo = '/sounds/sound.wav'
soundBeep = '/sounds/beep.wav'
soundTab = '/sounds/tab.wav'

pyportal.play_file(soundDemo)
```

https://adafru.it/IDF

https://adafru.it/IDF

Usage

Showing and Hiding Groups

When we set up the Groups, we added a function to make it easy for use to turn those Groups on or off. To use this function, just call `showLayer()` with the name of the Group that you want to show. So if you want to show the Group **view1**, use the following:

`showLayer(vew1)`

If you want to hide the Group **view2**, use the following:

`hideLayer(view1)`

So now if you only want the Group **view3** to be visible, you would do the following:

```
hideLayer(view1)
hideLayer(view2)
showLayer(view3)
```

Switching Images

The `set_image()` function that we set up earlier takes two parameters:

- **group** - Name of the Group that you want the image to be loaded into.
- **filename** - The path for the BMP image to load.

From there the function will remove the last image from the given Group, load a new BMP into the TileGrid and add it to the given Group.

That is it. We now have an easy way to change images for our UI.

The first image we will load will be into our `bg_group` by running the following code.

```
set_image(bg_group,"/images/BGimage.bmp")
```

Now the `set_image()` can be used to change images in the code loop for things like displaying weather icons, emojis, or even change the entire background image. Later we will use the `set_image()` function to make a button that will change an icon on one of our View tabs.

Updating Text

To use the function that we created for word wrapping, just run `text_box()` with the following parameters.

- **target** - The Label that you want to update with text.
- **top** - How far from the top is this Label, because an update may move the text.
- **string** - The text that you want to wrap in String format.
- **max_chars** - maximum amount of text characters you want in a line of text.

So the following code will let you send String data to a Label so that the text is wrapped and aligned to the top left of

its originally set spot.

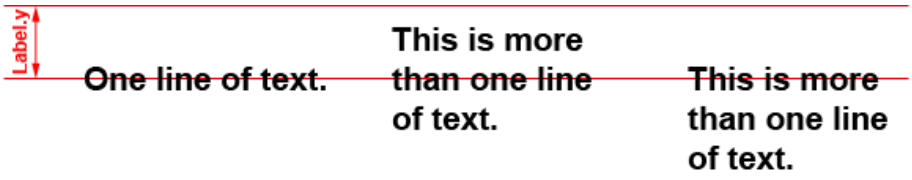
```
text_box(my_lable, 10, 'The text on this screen is wrapped.', 15)
```

The text block that the function returns will look like this:

```
The text on
this screen is
wrapped.
```

Use the top attribute to set the y position of the text box from the top of the group.

Here is how the Label will look with only one line of of text, multiple lines in the same Label.y position, and multiple lines repositioned so the top stays in place.



There is a new update in the works to make it easy to change the text box alignment, but for now this will work.

Reading Buttons in the Loop

Now that we have our buttons we need to read them. To do that we will first see if anything is touching the screen. Then we test the screen's touch coordinates to see if they are inside any of our buttons.

If the touchscreen is touched inside one of our buttons we can execute some code. For the Tab buttons this means switching views. The larger buttons will send commands to the NeoPixel to do something.

```
# ------------- Handle Button Press Detection  ------------- #
    if touch:  # Only do this if the screen is touched
        # loop with buttons using enumerate() to number each button group as i
        for i, b in enumerate(buttons):
            if b.contains(touch):  # Test each button to see if it was pressed
                print('button%d pressed' % i)
                if i == 0 and view_live != 1:
                    # only if button[0] is pressed and view1 is visable
                    switch_view(1)
                    while ts.touch_point:  # for debounce
                        pass
                if i == 1 and view_live != 2:
                    # only if button[1] is pressed and view2 is visable
                    switch_view(2)
                    while ts.touch_point:  # for debounce
                        pass
                if i == 2 and view_live != 3:
                    # only if button[2] is pressed and view3 is visable
                    switch_view(3)
                    while ts.touch_point:  # for debounce
                        pass
                if i == 3:
```

```
if i == 3:
    # Toggle switch button type
    if switch_state == 0:
        switch_state = 1
        b.label = "ON"
        b.selected = False
        pixel.fill(WHITE)
        print("Swich ON")
    else:
        switch_state = 0
        b.label = "OFF"
        b.selected = True
        pixel.fill(BLACK)
        print("Swich OFF")
    while ts.touch_point:  # for debounce
        pass
    print("Swich Pressed")
if i == 4:
    # Momentary button type
    b.selected = True
    print('Button Pressed')
    button_mode = numberUP(button_mode, 5)
    if button_mode == 1:
        pixel.fill(RED)
    elif button_mode == 2:
        pixel.fill(YELLOW)
    elif button_mode == 3:
        pixel.fill(GREEN)
    elif button_mode == 4:
        pixel.fill(BLUE)
    elif button_mode == 5:
        pixel.fill(PURPLE)
    switch_state = 1
    button_switch.label = "ON"
    button_switch.selected = False
    while ts.touch_point:  # for debounce
        pass
    print("Button released")
    b.selected = False
if i == 5 and view_live == 2:
    # button[5] only works if view2 is visable
    b.selected = True
    while ts.touch_point:  # for debounce
        pass
    print("Icon Button Pressed")
    b.selected = False
if i == 6 and view_live == 3:
    # button[6] only works if view3 is visable
    b.selected = True
    while ts.touch_point:  # for debounce
        pass
    print("Sound Button Pressed")
    b.selected = False
```

The last two buttons are ones that we have displayed on **view2** and **view3**. The issue with these buttons is that they are not visible unless the Group that they are a part of is visible. However, this button press detecting code does not care if the button is visible or not. The code just checks to see if the users touch was inside where the button should be. To fix this, we make it so that the button action only takes place if the button is pressed and its view Group is live.

```
if i == 6 and view_live == 3:
    b.selected = True
    while ts.touch_point:  # for debounce
        pass
    print("Sound Button Pressed")
    b.selected = False
```

The Full Code

All together now!

So now that we know how all the components work, here is an example of how this would all go together. This example uses all of these tools to make a UI with three window tabs with elements like buttons and images on them. You can start by using this as a base and customize it to be your own.

You can click **Download: Project Zip** in the code listing to get all the files or click the green button to see the files in the GitHub repository.

Copy the files to the PyPortal **CIRCUITPY** drive in the directories listed at the bottom of the page.

https://adafru.it/IF7

https://adafru.it/IF7

```
import time
import board
import microcontroller
import displayio
import busio
from analogio import AnalogIn
import neopixel
import adafruit_adt7410
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text.label import Label
from adafruit_button import Button
import adafruit_touchscreen
from adafruit_pyportal import PyPortal

# ------------- Inputs and Outputs Setup ------------- #
try:  # attempt to init. the temperature sensor
    i2c_bus = busio.I2C(board.SCL, board.SDA)
    adt = adafruit_adt7410.ADT7410(i2c_bus, address=0x48)
    adt.high_resolution = True
except ValueError:
    # Did not find ADT7410. Probably running on Titano or Pynt
    adt = None

# init. the light sensor
light_sensor = AnalogIn(board.LIGHT)

pixel = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=1)
WHITE = 0xffffff
RED = 0xff0000
YELLOW = 0xffff00
GREEN = 0x00ff00
BLUE = 0x0000ff
PURPLE = 0xff00ff
BLACK = 0x000000

# ---------- Sound Effects ------------- #
soundDemo = '/sounds/sound.wav'
soundBeep = '/sounds/beep.wav'
soundTab = '/sounds/tab.wav'
```

```
soundtab    /sounds/tabtab.
```

```
# ------------- Other Helper Functions------------- #
# Helper for cycling through a number set of 1 to x.
def numberUP(num, max_val):
    num += 1
    if num <= max_val:
        return num
    else:
        return 1

# ------------- Screen Setup ------------- #
pyportal = PyPortal()
display = board.DISPLAY
display.rotation = 270

# Backlight function
# Value between 0 and 1 where 0 is OFF, 0.5 is 50% and 1 is 100% brightness.
def set_backlight(val):
    val = max(0, min(1.0, val))
    board.DISPLAY.auto_brightness = False
    board.DISPLAY.brightness = val

# Set the Backlight
set_backlight(0.3)

# Touchscreen setup
# ------Rotate 270:
screen_width = 240
screen_height = 320
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_YD, board.TOUCH_YU,
                                      board.TOUCH_XR, board.TOUCH_XL,
                                      calibration=((5200, 59000),
                                                   (5800, 57000)),
                                      size=(screen_width, screen_height))


# ------------- Display Groups ------------- #
splash = displayio.Group(max_size=15)  # The Main Display Group
view1 = displayio.Group(max_size=15)  # Group for View 1 objects
view2 = displayio.Group(max_size=15)  # Group for View 2 objects
view3 = displayio.Group(max_size=15)  # Group for View 3 objects

def hideLayer(hide_target):
    try:
        splash.remove(hide_target)
    except ValueError:
        pass

def showLayer(show_target):
    try:
        time.sleep(0.1)
        splash.append(show_target)
    except ValueError:
        pass

# ------------- Setup for Images ------------- #

# Display an image until the loop starts
pyportal.set_background('/images/loading.bmp')
```

```
bg_group = displayio.Group(max_size=1)
splash.append(bg_group)


icon_group = displayio.Group(max_size=1)
icon_group.x = 180
icon_group.y = 120
icon_group.scale = 1
view2.append(icon_group)

# This will handel switching Images and Icons
def set_image(group, filename):
    """Set the image file for a given goup for display.
    This is most useful for Icons or image slideshows.
        :param group: The chosen group
        :param filename: The filename of the chosen image
    """
    print("Set image to ", filename)
    if group:
        group.pop()

    if not filename:
        return  # we're done, no icon desired

    image_file = open(filename, "rb")
    image = displayio.OnDiskBitmap(image_file)
    try:
        image_sprite = displayio.TileGrid(image, pixel_shader=displayio.ColorConverter())
    except TypeError:
        image_sprite = displayio.TileGrid(image, pixel_shader=displayio.ColorConverter(),
                                          position=(0, 0))
    group.append(image_sprite)

set_image(bg_group, "/images/BGimage.bmp")

# ---------- Text Boxes ------------- #
# Set the font and preload letters
font = bitmap_font.load_font("/fonts/Helvetica-Bold-16.bdf")
font.load_glyphs(b'abcdefghjiklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890- ()')

# Default Label styling:
TABS_X = 5
TABS_Y = 50

# Text Label Objects
feed1_label = Label(font, text="Text Wondow 1", color=0xE39300, max_glyphs=200)
feed1_label.x = TABS_X
feed1_label.y = TABS_Y
view1.append(feed1_label)

feed2_label = Label(font, text="Text Wondow 2", color=0xFFFFFF, max_glyphs=200)
feed2_label.x = TABS_X
feed2_label.y = TABS_Y
view2.append(feed2_label)

sensors_label = Label(font, text="Data View", color=0x03AD31, max_glyphs=200)
sensors_label.x = TABS_X
sensors label.v = TABS Y
```

```
view3.append(sensors_label)

sensor_data = Label(font, text="Data View", color=0x03AD31, max_glyphs=100)
sensor_data.x = TABS_X+15
sensor_data.y = 170
view3.append(sensor_data)


text_hight = Label(font, text="M", color=0x03AD31, max_glyphs=10)
# return a reformatted string with word wrapping using PyPortal.wrap_nicely
def text_box(target, top, string, max_chars):
    text = pyportal.wrap_nicely(string, max_chars)
    new_text = ""
    test = ""
    for w in text:
        new_text += '\n'+w
        test += 'M\n'
    text_hight.text = test  # Odd things happen without this
    glyph_box = text_hight.bounding_box
    target.text = ""  # Odd things happen without this
    target.y = int(glyph_box[3]/2)+top
    target.text = new_text

# ---------- Display Buttons ------------- #
# Default button styling:
BUTTON_HEIGHT = 40
BUTTON_WIDTH = 80

# We want three buttons across the top of the screen
TAPS_HEIGHT = 40
TAPS_WIDTH = int(screen_width/3)
TAPS_Y = 0

# We want two big buttons at the bottom of the screen
BIG_BUTTON_HEIGHT = int(screen_height/3.2)
BIG_BUTTON_WIDTH = int(screen_width/2)
BIG_BUTTON_Y = int(screen_height-BIG_BUTTON_HEIGHT)

# This group will make it easy for us to read a button press later.
buttons = []

# Main User Interface Buttons
button_view1 = Button(x=0, y=0,
                      width=TAPS_WIDTH, height=TAPS_HEIGHT,
                      label="View1", label_font=font, label_color=0xff7e00,
                      fill_color=0x5c5b5c, outline_color=0x767676,
                      selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                      selected_label=0x525252)
buttons.append(button_view1)  # adding this button to the buttons group

button_view2 = Button(x=TAPS_WIDTH, y=0,
                      width=TAPS_WIDTH, height=TAPS_HEIGHT,
                      label="View2", label_font=font, label_color=0xff7e00,
                      fill_color=0x5c5b5c, outline_color=0x767676,
                      selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                      selected_label=0x525252)
buttons.append(button_view2)  # adding this button to the buttons group

button_view3 = Button(x=TAPS_WIDTH*2, y=0,
```

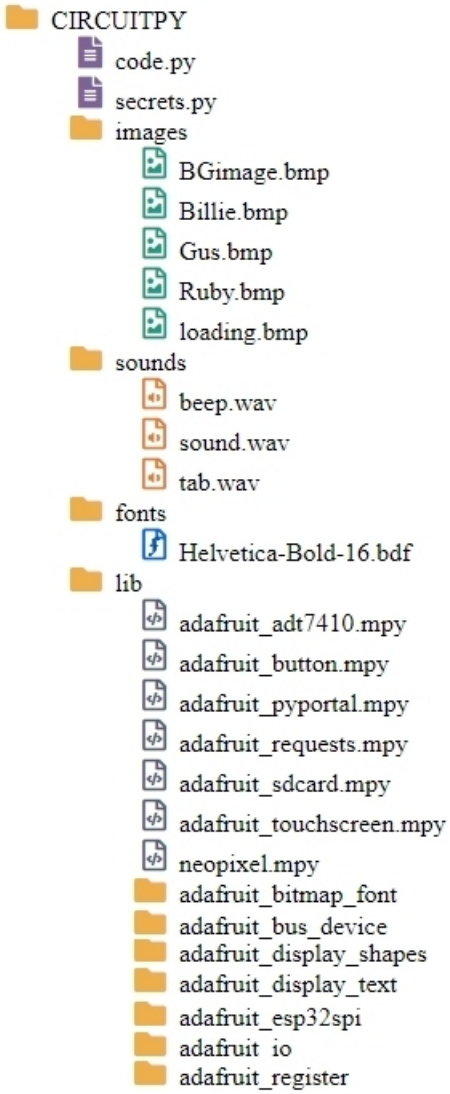
```
                      width=TAPS_WIDTH, height=TAPS_HEIGHT,
                      label="View3", label_font=font, label_color=0xff7e00,
                      fill_color=0x5c5b5c, outline_color=0x767676,
                      selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                      selected_label=0x525252)
buttons.append(button_view3)  # adding this button to the buttons group

button_switch = Button(x=0, y=BIG_BUTTON_Y,
                       width=BIG_BUTTON_WIDTH, height=BIG_BUTTON_HEIGHT,
                       label="Switch", label_font=font, label_color=0xff7e00,
                       fill_color=0x5c5b5c, outline_color=0x767676,
                       selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                       selected_label=0x525252)
buttons.append(button_switch)  # adding this button to the buttons group

button_2 = Button(x=BIG_BUTTON_WIDTH, y=BIG_BUTTON_Y,
                  width=BIG_BUTTON_WIDTH, height=BIG_BUTTON_HEIGHT,
                  label="Button", label_font=font, label_color=0xff7e00,
                  fill_color=0x5c5b5c, outline_color=0x767676,
                  selected_fill=0x1a1a1a, selected_outline=0x2e2e2e,
                  selected_label=0x525252)
buttons.append(button_2)  # adding this button to the buttons group

# Add all of the main buttons to the spalsh Group
for b in buttons:
    splash.append(b.group)


# Make a button to change the icon image on view2
button_icon = Button(x=150, y=60,
                     width=BUTTON_WIDTH, height=BUTTON_HEIGHT,
                     label="Icon", label_font=font, label_color=0xffffff,
                     fill_color=0x8900ff, outline_color=0xbc55fd,
                     selected_fill=0x5a5a5a, selected_outline=0xff6600,
                     selected_label=0x525252, style=Button.ROUNDRECT)
buttons.append(button_icon)  # adding this button to the buttons group

# Add this button to view2 Group
view2.append(button_icon.group)

# Make a button to play a sound on view2
button_sound = Button(x=150, y=170,
                      width=BUTTON_WIDTH, height=BUTTON_HEIGHT,
                      label="Sound", label_font=font, label_color=0xffffff,
                      fill_color=0x8900ff, outline_color=0xbc55fd,
                      selected_fill=0x5a5a5a, selected_outline=0xff6600,
                      selected_label=0x525252, style=Button.ROUNDRECT)
buttons.append(button_sound)  # adding this button to the buttons group

# Add this button to view2 Group
view3.append(button_sound.group)

#pylint: disable=global-statement
def switch_view(what_view):
    global view_live
    if what_view == 1:
        hideLayer(view2)
        hideLayer(view3)
        button_view1.selected = False
        button view2.selected = True
```

```
        button_view3.selected = True
        showLayer(view1)
        view_live = 1
        print("View1 On")
    elif what_view == 2:
        # global icon
        hideLayer(view1)
        hideLayer(view3)
        button_view1.selected = True
        button_view2.selected = False
        button_view3.selected = True
        showLayer(view2)
        view_live = 2
        print("View2 On")
    else:
        hideLayer(view1)
        hideLayer(view2)
        button_view1.selected = True
        button_view2.selected = True
        button_view3.selected = False
        showLayer(view3)
        view_live = 3
        print("View3 On")
#pylint: enable=global-statement

# Set veriables and startup states
button_view1.selected = False
button_view2.selected = True
button_view3.selected = True
showLayer(view1)
hideLayer(view2)
hideLayer(view3)

view_live = 1
icon = 1
icon_name = "Ruby"
button_mode = 1
switch_state = 0
button_switch.label = "OFF"
button_switch.selected = True

# Update out Labels with display text.
text_box(feed1_label, TABS_Y,
         "The text on this screen is wrapped so that all of it fits nicely into a \
text box that is ### x ###.", 30)
text_box(feed1_label, TABS_Y,
         'The text on this screen is wrapped so that all of it fits nicely into a \
text box that is {} x {}.'
         .format(feed1_label.bounding_box[2], feed1_label.bounding_box[3]*2), 30)

text_box(feed2_label, TABS_Y, 'Tap on the Icon button to meet a new friend.', 18)

text_box(sensors_label, TABS_Y,
         "This screen can display sensor readings and tap Sound to play a WAV file.", 28)

board.DISPLAY.show(splash)

# ------------- Code Loop ------------- #
while True:
```

```
    touch = ts.touch_point
    light = light_sensor.value

    if adt:  # Only if we have the temperature sensor
        tempC = adt.temperature
    else:  # No temperature sensor
        tempC = microcontroller.cpu.temperature

    tempF = tempC * 1.8 + 32
    sensor_data.text = 'Touch: {}\nLight: {}\n Temp: {:.0f}°F'.format(touch, light, tempF)

    # ------------- Handle Button Press Detection  ------------- #
    if touch:  # Only do this if the screen is touched
        # loop with buttons using enumerate() to number each button group as i
        for i, b in enumerate(buttons):
            if b.contains(touch):  # Test each button to see if it was pressed
                print('button%d pressed' % i)
                if i == 0 and view_live != 1:  # only if view1 is visable
                    pyportal.play_file(soundTab)
                    switch_view(1)
                    while ts.touch_point:
                        pass
                if i == 1 and view_live != 2:  # only if view2 is visable
                    pyportal.play_file(soundTab)
                    switch_view(2)
                    while ts.touch_point:
                        pass
                if i == 2 and view_live != 3:  # only if view3 is visable
                    pyportal.play_file(soundTab)
                    switch_view(3)
                    while ts.touch_point:
                        pass
                if i == 3:
                    pyportal.play_file(soundBeep)
                    # Toggle switch button type
                    if switch_state == 0:
                        switch_state = 1
                        b.label = "ON"
                        b.selected = False
                        pixel.fill(WHITE)
                        print("Swich ON")
                    else:
                        switch_state = 0
                        b.label = "OFF"
                        b.selected = True
                        pixel.fill(BLACK)
                        print("Swich OFF")
                    # for debounce
                    while ts.touch_point:
                        pass
                    print("Swich Pressed")
                if i == 4:
                    pyportal.play_file(soundBeep)
                    # Momentary button type
                    b.selected = True
                    print('Button Pressed')
                    button_mode = numberUP(button_mode, 5)
                    if button_mode == 1:
                        pixel.fill(RED)
                    elif button mode == 2:
```

```
                    pixel.fill(YELLOW)
                elif button_mode == 3:
                    pixel.fill(GREEN)
                elif button_mode == 4:
                    pixel.fill(BLUE)
                elif button_mode == 5:
                    pixel.fill(PURPLE)
                switch_state = 1
                button_switch.label = "ON"
                button_switch.selected = False
                # for debounce
                while ts.touch_point:
                    pass
                print("Button released")
                b.selected = False
            if i == 5 and view_live == 2:  # only if view2 is visable
                pyportal.play_file(soundBeep)
                b.selected = True
                while ts.touch_point:
                    pass
                print("Icon Button Pressed")
                icon = numberUP(icon, 3)
                if icon == 1:
                    icon_name = "Ruby"
                elif icon == 2:
                    icon_name = "Gus"
                elif icon == 3:
                    icon_name = "Billie"
                b.selected = False
                text_box(feed2_label, TABS_Y,
                         "Every time you tap the Icon button the icon image will \
change. Say hi to {}!".format(icon_name), 18)
                set_image(icon_group, "/images/"+icon_name+".bmp")
            if i == 6 and view_live == 3:  # only if view3 is visable
                b.selected = True
                while ts.touch_point:
                    pass
                print("Sound Button Pressed")
                pyportal.play_file(soundDemo)
                b.selected = False
```

When you are finished the **CIRCUITPY** drive should look something like this:


CIRCUITPY
code.py
secrets.py
images
BGimage.bmp
Billie.bmp
Gus.bmp
Ruby.bmp
loading.bmp
sounds
beep.wav
sound.wav
tab.wav
fonts
Helvetica-Bold-16.bdf
lib
adafruit_adt7410.mpy
adafruit_button.mpy
adafruit_pyportal.mpy
adafruit_requests.mpy
adafruit_sdcard.mpy
adafruit_touchscreen.mpy
neopixel.mpy
adafruit_bitmap_font
adafruit_bus_device
adafruit_display_shapes
adafruit_display_text
adafruit_esp32spi
adafruit_io
adafruit_register

Make it your own

Adding MQTT connection to your UI

At some point you may want to connect your UI to a Pub Sub service like MQTT. Here is some example code that uses the miniMQTT and PyPortal libraries to connect with a hosted MQTT server. See is you can add some of these UI elements to the example.

```
import time
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
import adafruit_minimqtt as MQTT
import adafruit_pyportal

pyportal = adafruit_pyportal.PyPortal()

### WiFi ###

# 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

# pylint: disable=protected-access
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(pyportal._esp, secrets, None)

# ------------- MQTT Topic Setup ------------- #
mqtt_topic = "test/topic"

### Code ###
# Define callback methods which are called when events occur
# pylint: disable=unused-argument, redefined-outer-name
def connected(client, userdata, flags, rc):
    # This function will be called when the client is connected
    # successfully to the broker.
    print("Subscribing to %s" % (mqtt_topic))
    client.subscribe(mqtt_topic)


def disconnected(client, userdata, rc):
    # This method is called when the client is disconnected
    print("Disconnected from MQTT Broker!")


def message(client, topic, message):
    """Method callled when a client's subscribed feed has a new
    value.
    :param str topic: The topic of the feed with a new value.
    :param str message: The new value
    """
    print("New message on topic {0}: {1}".format(topic, message))


# Connect to WiFi
```

```
# Connect to WiFi
print("Connecting to WiFi...")
wifi.connect()
print("Connected!")

# Initialize MQTT interface with the esp interface
# pylint: disable=protected-access
MQTT.set_socket(socket, pyportal._esp)

# Set up a MiniMQTT Client
mqtt_client = MQTT.MQTT(
    broker=secrets["broker"],
    username=secrets["user"],
    password=secrets["pass"],
    is_ssl=False,
)

# Setup the callback methods above
mqtt_client.on_connect = connected
mqtt_client.on_disconnect = disconnected
mqtt_client.on_message = message

# Connect the client to the MQTT broker.
mqtt_client.connect()

photocell_val = 0
while True:
    # Poll the message queue
    mqtt_client.loop()

    # Send a new message
    print("Sending photocell value: %d" % photocell_val)
    mqtt_client.publish(mqtt_topic, photocell_val)
    photocell_val += 1
    time.sleep(1)
```