Easy NeoPixel Graphics with the CircuitPython Pixel Framebuf Library
Created by Melissa LeBlanc-Williams
# Guide Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>Parts</td>
<td>3</td>
</tr>
<tr>
<td>Wiring</td>
<td>5</td>
</tr>
<tr>
<td>NeoPixel Matrix Wiring</td>
<td>6</td>
</tr>
<tr>
<td>CircuitPython Wiring</td>
<td>6</td>
</tr>
<tr>
<td>Raspberry Pi Wiring</td>
<td>7</td>
</tr>
<tr>
<td>DotStar Wiring</td>
<td>8</td>
</tr>
<tr>
<td>Import and Setup</td>
<td>10</td>
</tr>
<tr>
<td>CircuitPython Setup</td>
<td>10</td>
</tr>
<tr>
<td>Python Setup</td>
<td>10</td>
</tr>
<tr>
<td>Font File</td>
<td>11</td>
</tr>
<tr>
<td>Import and Object Setup</td>
<td>11</td>
</tr>
<tr>
<td>NeoPixels</td>
<td>11</td>
</tr>
<tr>
<td>DotStar LEDs</td>
<td>12</td>
</tr>
<tr>
<td>Usage</td>
<td>14</td>
</tr>
<tr>
<td>Initialization</td>
<td>14</td>
</tr>
<tr>
<td>Example Initializations</td>
<td>14</td>
</tr>
<tr>
<td>32x8 Panel</td>
<td>14</td>
</tr>
<tr>
<td>16x16 Panel</td>
<td>15</td>
</tr>
<tr>
<td>NeoPixel FeatherWing</td>
<td>15</td>
</tr>
<tr>
<td>DotStar FeatherWing</td>
<td>15</td>
</tr>
<tr>
<td>Drawing Functions</td>
<td>15</td>
</tr>
<tr>
<td>Displaying the Framebuffer</td>
<td>15</td>
</tr>
<tr>
<td>Fill</td>
<td>16</td>
</tr>
<tr>
<td>Pixel Drawing</td>
<td>17</td>
</tr>
<tr>
<td>Line Drawing</td>
<td>18</td>
</tr>
<tr>
<td>Horizontal and Vertical Line Drawing</td>
<td>18</td>
</tr>
<tr>
<td>Rectangle Drawing</td>
<td>19</td>
</tr>
<tr>
<td>Text Drawing</td>
<td>20</td>
</tr>
<tr>
<td>Image Drawing with Python</td>
<td>22</td>
</tr>
<tr>
<td>How It Works</td>
<td>24</td>
</tr>
<tr>
<td>API Documentation</td>
<td>26</td>
</tr>
</tbody>
</table>
Overview

The Pixel Framebuf library makes drawing to a grid of NeoPixels or DotStars as easy as writing to a display. The library has all of the drawing functions of the original framebuffer library, including the ability to draw lines, circles, rectangles, and text.

This library can be used on a wide variety of NeoPixel and DotStar LED layouts including with flexible and non-flexible matrices, NeoPixels strips laid out by hand, or FeatherWings. This library has been tested to work with both CircuitPython Microcontrollers and Blinka on the Raspberry Pi.

One of the huge advantages that using NeoPixel and DotStar LED matrices has over the standard RGB LED matrices is that they only use one or two GPIO pins. The Pixel Framebuf library gives you the simplicity and control to make use of these matrices to display data in a meaningful way in much the same way that you can on RGB matrices.

Parts

Here are a few of many products that this library supports:

Flexible 16x16 NeoPixel RGB LED Matrix
For advanced NeoPixel fans, we now have a bendable, Flexible 16x16 NeoPixel LED Matrix! Control all 256 ultra-bright LEDs using a single microcontroller pin, set each...
$94.95
In Stock
Add to Cart
Adafruit NeoPixel LED Strip Starter Pack - 30 LED meter - Black
You can't bake a cake without flour, sugar, and baking soda? Cream cheese? Muscadet? Ok - to be honest we don't do much cake baking. But we do light up a TON of...

$24.95
In Stock
Add to Cart

NeoPixels take a lot of power, so a 5V 4 Amp or even a 10 Amp power supply is recommended.

5V 4A (4000mA) switching power supply - UL Listed
Need a lot of 5V power? This switching supply gives a clean regulated 5V output at up to 4 Amps (4000mA). 110 or 240 input, so it works in any country. The plugs are "US...

$14.95
In Stock
Add to Cart

5V 10A switching power supply
This is a beefy switching supply, for when you need a lot of power! It can supply 5V DC up to 10 Amps, running from 110V or 220V power (the plug it comes with is for US/Canada/Japan...
Wiring

To make wiring simple, we recommend a couple of different products available in our store. First we recommend a DC Power jack with terminals so it's easy to connect up to an external power supply.

**Female DC Power adapter - 2.1mm jack to screw terminal block**
If you need to connect a DC power wall wart to a board that doesn't have a DC jack - this adapter will come in very handy! There is a 2.1mm DC jack on one end, and a screw terminal...
$2.00
In Stock
Add to Cart

The other product that makes it easy if you have one of our NeoPixel Matrices is a 2-wire JST SM connector so you can use the the data line without cutting anything:

**2-pin JST SM Plug + Receptacle Cable Set**
These 2-wire cables are 16cm long and come as a set. One side has a 2-pin JST SM type connector plug on the end. The other side has a matching 2-pin JST SM type...
$0.75
In Stock
Add to Cart

Likewise, if you have one of our DotStar Matrices, we have a 4-wire version as well:

**4-pin JST SM Plug + Receptacle Cable Set**
These 4-wire cables are 15cm long and come as a set, one side has a JST SM type connector plug on the end. The other side has a matching JST SM type receptacle connector. They are good...
$1.50
In Stock
Add to Cart

Another easy option to connect NeoPixels is to use some Jumper Wires:

**Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)**
Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in a 'strip' of 40 (4 pieces of each of ten rainbow...
$3.95
In Stock
Add to Cart
NeoPixel Matrix Wiring

Wiring the pixels up to either a Microcontroller running CircuitPython or a Raspberry Pi running Blinka is very easy since either one only uses a single GPIO pin. Adafruit matrices come wired with separate power wires and data lines which makes it really easy to connect.

CircuitPython Wiring

It is recommended using a microcontroller with an M4 processor, because it can drive a lot of NeoPixels fast and doesn't cost much more than an M0 board.

To connect it, you will need to wire it as follows:

- Connect the **red wire** to **+5V** on the power supply.
- Connect the **black wire** to **Ground** on the power supply
- Connect the **white DIN wire** of the female JST connector to **D6** on the Microcontroller
- Connect the **black wire** of the female JST connector to **Ground** on the Microcontroller
Raspberry Pi Wiring

Raspberry Pi boards are powerful enough to drive lots of NeoPixels, but you will need to connect it to GPIO10, GPIO12, GPIO18 or GPIO21 to work! We recommend GPIO18, which is Pin 12 on the 40-pin header.

To connect it, you will need to wire it as follows:

- Connect the red wire to +5V on the power supply.
- Connect the black wire to Ground on the power supply
- Connect the white wire of the JST connector to GPIO18 (Pin 12) on the Microcontroller
- Connect the black wire of the JST connector to Ground (Pin 6) on the Microcontroller
For more information on wiring other configurations, be sure to check out our Adafruit NeoPixel Überguide (https://adafru.it/dhw).

DotStar Wiring

Wiring the DotStar matrix up will only take up two GPIO pins. Adafruit matrices come wired with separate power wires and a connector with the power, data, and clock lines in a single connector, which makes them easy to chain together. For a very in-depth guide on DotStars and wiring them up in various configurations, be sure to checkout the Adafruit DotStar LEDs (https://adafru.it/zbD) guide.
Import and Setup

The Pixel Framebuf library is designed to make adding text and graphics super simple. It is built on top of the very capable LED Animation and framebuf libraries and because of the flexibility provided by those libraries, it works on a wide variety of NeoPixel and DotMatrix displays and layouts.

CircuitPython Setup

To get the required libraries for this guide, download the latest CircuitPython library bundle from circuitpython.org.

Open the downloaded zip and find the following folder and files within the `lib` folder:

- `adafruit_pixel_framebuf.mpy`
- `adafruit_framebuf.mpy`
- `adafruit_led_animation`

If you are using NeoPixels, you will also need:

- `neopixel.mpy`

Or if you are using DotStar LEDs, you will need:

- `adafruit_dotstar.mpy`

Drag this folder and files to the `lib` folder on your CIRCUITPY drive.

Python Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

Run NeoPixels on the Raspberry Pi requires the `sudo` command, so you will need to make sure you have the necessary libraries installed using `sudo pip3`.

Since Pip does a great job of handling dependencies, there's only a few commands you will need to run
to get this set up:

- `sudo pip3 install adafruit-circuitpython-pixel-framebuf`

If you are running NeoPixels, you will also need the NeoPixel library:

- `sudo pip3 install adafruit-circuitpython-neopixel`

If you are running DotStars, you will also need the DotStar library:

- `sudo pip3 install adafruit-circuitpython-dotstar`

You will also need the Pillow library if you want to make use of the `image()` function:

- `sudo pip3 install Pillow`

Raspberry Pi requires a hardware PWM pin to drive the NeoPixels. We recommend D18, so you will need to change that in the code below if you any place you see D6.

Font File

If you want to display text, you will also need to include the font file. This font is included in the adafruit_framebuf library examples.

```
https://adafruit.it/Gfb
```

Place the font file in the same folder as you code.

Import and Object Setup

This library should work with either NeoPixels or DotStar LEDs. We are going to start by showing you how to use them with NeoPixels.

NeoPixels

The first thing you will need to do to run this library is to import the necessary libraries. An example of import and setup for the NeoPixel FeatherWing is as follows:
import board
import neopixel
from adafruit_pixel_framebuf import PixelFramebuffer

pixel_pin = board.D6
pixel_width = 8
pixel_height = 4

pixels = neopixel.NeoPixel(
    pixel_pin, pixel_width * pixel_height, brightness=0.1,
    auto_write=False,
)

First you import board and neopixel. Next import the PixelFramebuffer module. On certain NeoPixel displays, you may also need to import VERTICAL as well, which we'll cover in the Initialization section.

Next you'll want to set a few convenience variables including the pixel_pin, which is the pin that the NeoPixel data line is connected to. The pixel_width and pixel_height are there to help make the code more readable and correspond to the number of pixels wide and number of pixels high.

Finally, create the pixel object. You'll want to make sure auto_write is set to False so you don't see every single time an individual pixel is changed, which can be quite slow.

This guide will use NeoPixels for all the examples, but the Pixel Framebuf library works equally well with DotStar LEDs.

DotStar LEDs

The only change you would need to make to use this library with DotStar LEDs would be to initialize the DotStar library instead of the NeoPixel library. An example of import and setup for the DotStar FeatherWing is as follows:

import board
import adafruit_dotstar
from adafruit_pixel_framebuf import PixelFramebuffer

pixel_pin = board.D6
pixel_width = 12
pixel_height = 6

pixels = adafruit_dotstar.DotStar(
    board.D13, board.D11, pixel_width * pixel_height, brightness=0.3,
    auto_write=False,
)
This example imports the necessary modules and assigns the appropriate pins and number of pixels to use 72 DotStar LEDs connected to D13 and D11.
Usage

Using the Pixel Framebuf library is easy. The most complicated part of it is the initialization, because it can vary so widely with each specific setup, so start with that and then go over each of the functions available to you.

Initialization

For initialization, we'll go over the different options and what the specific parameters mean so you can best determine which options to change for your setup.

- **pixels** - (required) The NeoPixels or DotStars object that you initialized
- **width** - (required) The width of the framebuf in pixels.
- **height** - (required) The height of the framebuf in pixels.
- **orientation** - The direction that the pixels are physically laid out in either columns or rows. The value is `HORIZONTAL` by default and you will need to import `VERTICAL` if you want to set the value to `VERTICAL`.

- **alternating** - If the pixels zigzag, you will want to set this to `True`. If they are laid out in the same direction, you will want to set this to `False`. The default value is `True`.

- **reverse_x** - Set this to `True` if you want to reverse the direction of all pixels horizontally.

- **reverse_y** - Set this to `True` if you want to reverse the direction of all pixels vertically.

- **top** - An optional tuple representing the top left pixel.

- **bottom** - An optional tuple representing the bottom right pixel.

- **rotation** - A value between 0 and 3 representing the initial rotation of the Framebuffer. This can also be set later with the `rotation` property.

Example Initializations

Here are the initializations for a few of the panels we have available. In these examples, it is assumed that you have done the necessary imports and setup as described on the Import and Setup page and that your NeoPixels or DotStars are named `pixels`.

32x8 Panel
pixel_framebuf = PixelFramebuffer(
    pixels,
    32,
    8,
    orientation=VERTICAL,
    rotation=2
)

16x16 Panel

pixel_framebuf = PixelFramebuffer(
    pixels,
    16,
    16,
    reverse_x=True,
)

NeoPixel FeatherWing

pixel_framebuf = PixelFramebuffer(
    pixels,
    8,
    4,
    alternating=False,
)

DotStar FeatherWing

pixel_framebuf = PixelFramebuffer(
    pixels,
    12,
    6,
    alternating=False,
)

Drawing Functions

All drawing functions are based on coordinates starting from the upper left-hand side as the starting point.

Displaying the Framebuffer

Each time you want to display the current Framebuffer, you will need to call the `display()` function. This function takes no parameters and displays whatever is currently in memory. We will show you how to use it in each of the drawing function example code snippets.
Fill

To fill in the entire Framebuffer with a specific color, you can use the `fill()` function. For instance, to fill it with Blue (`0x0000FF`), you would use the following command:

```python
pixel_framebuf.fill(0x0000FF)
pixel_framebuf.display()
```

To turn all the pixels off, you can set them to `0x000000`.

```python
pixel_framebuf.fill(0x000000)
pixel_framebuf.display()
```
Pixel Drawing

To draw a pixel, you just need to use the `pixel()` function and provide the coordinates and color. For instance, if you wanted to draw a set the pixel at (4, 6) to red, you would use the following command:

```python
pixel_framebuf.pixel(4, 6, 0xFF0000)
pixel_framebuf.display()
```
If you wanted to get the pixel's value, you would retrieve it like this:

```python
color = pixel_framebuf.pixel(4, 6)
```

**Line Drawing**

To draw a line, you just need to use the `line()` function and provide the starting coordinates, ending coordinates, and color. For instance, if you wanted to draw a line from (0, 0) to (7, 9) in red (`0xFF0000`), you would use the following command:

```python
pixel_framebuf.line(0, 0, 7, 9, 0xFF0000)
pixel_framebuf.display()
```

**Horizontal and Vertical Line Drawing**

To draw horizontal or vertical, you can use the `hline()` and `vline()` functions. These are optimized for fast drawing because they don't need to calculate a slope. You just need to pass the x and y coordinates along with the length of the line in pixels and the color you wish to draw.

To use these functions, to draw a 5 pixel red line starting at (2, 3), you would use the following:

```python
pixel_framebuf.hline(2, 3, 5, 0xFF0000)
pixel_framebuf.vline(2, 3, 5, 0xFF0000)
pixel_framebuf.display()
```
Rectangle Drawing

To draw a non-filled Rectangle, you will want to use the `rect()` function. You provide it starting coordinates, width, height, and line color. To draw a red rectangle at (2, 2) with a width of 8 and a height of 12, you would use:

```python
pixel_framebuf.rect(2, 2, 8, 12, 0xFF0000)
pixel_framebuf.display()
```
To draw a filled rectangle, you would do the same thing with the `fill_rect()` function:

```python
pixel_framebuf.fill_rect(2, 2, 8, 12, 0xFF0000)
pixel_framebuf.display()
```

Text Drawing

To draw text, you would use the `text()` function, provide the string, upper left coordinates of the text, and the color. For instance, to draw "Hi" at (3, 4) in green (`0x00FF00`), you would use the following:

```python
pixel_framebuf.text("Hi", 3, 4, 0x00FF00)
pixel_framebuf.display()
```
Text can go off the screen and simply redrawing the text in different positions on top of the same background can produce an animated text effect.
Image Drawing with Python

If you are using CPython on Linux, you can also make use of the Pillow library and the image function to draw images to the display. The only limitations are the image needs to be the exact same size as the Framebuffer and needs to be in RGB mode.

We're going to go over an example for running this. First download the `blinka_16x16.png` image and upload it to the folder that you plan to run the script in.

```
https://adafruit.it/Nxb
```

Next download the `pixel_framebuf_pillow_image.py` example, which is available in the Library repo:
import board
import neopixel
from PIL import Image
from adafruit_pixel_framebuf import PixelFramebuffer

pixel_pin = board.D18
pixel_width = 16
pixel_height = 16

pixels = neopixel.NeoPixel(
pixel_pin,
pixel_width * pixel_height,
brightness=0.1,
auto_write=False,
)

pixel_framebuf = PixelFramebuffer(
pixels,
pixel_width,
pixel_height,
reverse_x=True,
)

# Make a black background in RGBA Mode
image = Image.new("RGBA", (pixel_width, pixel_height))

# Open the icon
icon = Image.open("blinka_16x16.png")

# Alpha blend the icon onto the background
image.alpha_composite(icon)

# Convert the image to RGB and display it
pixel_framebuf.image(image.convert("RGB"))
pixel_framebuf.display()

Run the code.

- sudo python3 pixel_framebuf_pillow_image.py

You should see an output similar to below:
How It Works

Below describes what is going on. You may recognize much of it from the previous section.

First start with imports. In addition to the usual imports, also import `Image` from the Pillow library.

```python
import board
import neopixel
from PIL import Image
from adafruit_pixel_framebuf import PixelFramebuffer
```

After that, set up the object, except it's using D18 in this case. You can find more details about this earlier in this guide.

```python
pixel_pin = board.D18
pixel_width = 16
pixel_height = 16

pixels = neopixel.NeoPixel(
    pixel_pin, pixel_width * pixel_height, brightness=0.1, auto_write=False,
)

pixel_framebuf = PixelFramebuffer(pixels, pixel_width, pixel_height, reverse_x=True)
```

Next use Pillow to create a background and `alpha_composite()` the image onto it. The reason to do it this way is because the adafruit_framebuf library requires the image to be in RGB format and if you directly you use `convert()` in Pillow, it will convert all transparencies to white. It's easier to see with a darker
background.

Start off by creating a new image in **RGBA** mode. **RGBA** mode is required to use the `alpha_composite()` function. The default color is black, so you don't need to specify a color to use by default, although you certainly could.

```python
image = Image.new("RGBA", (pixel_width, pixel_height))
```

Next open the icon file.

```python
icon = Image.open("blinka_16x16.png")
```

Next use the `alpha_composite()` function to blend the icon onto the black background.

```python
image.alpha_composite(icon)
```

Finally, convert the combined image to RGB right before passing it to the `image()` function and then display it.

```python
pixel_framebuf.image(image.convert("RGB"))
pixel_framebuf.display()
```

That's it. It should display onto the NeoPixels.