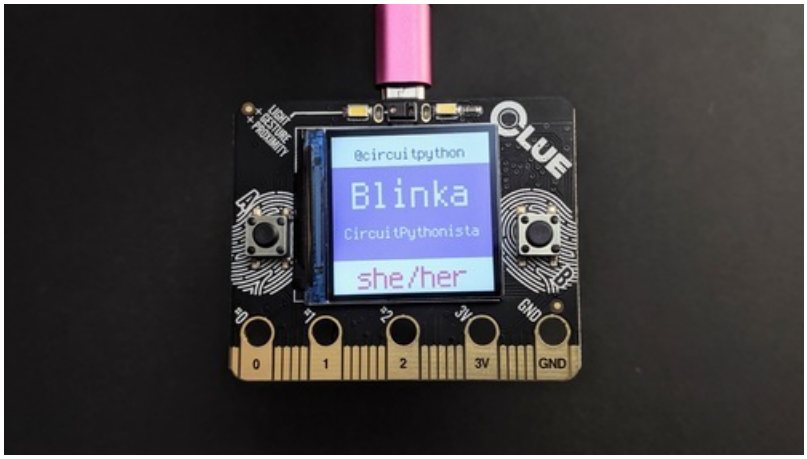
adafruit learning system

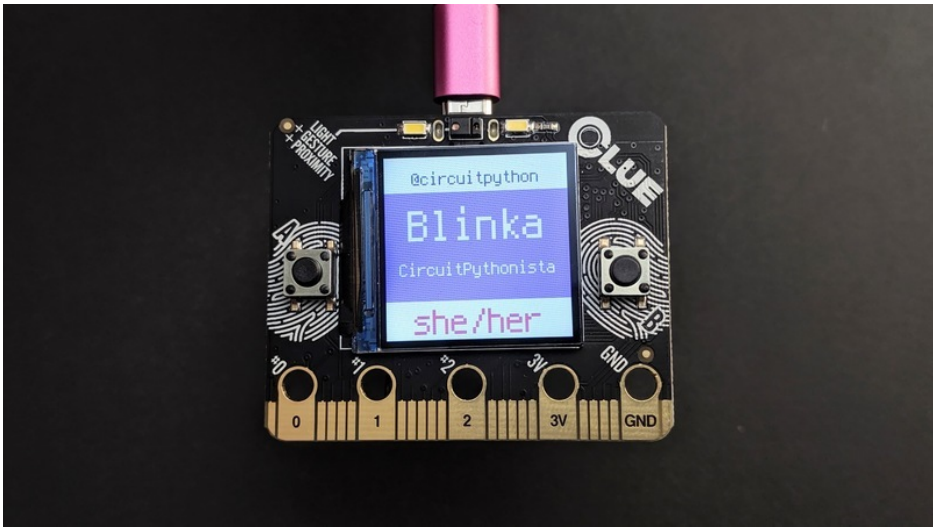
CLUE Custom CircuitPython Badge

Created by Kattni Rembor

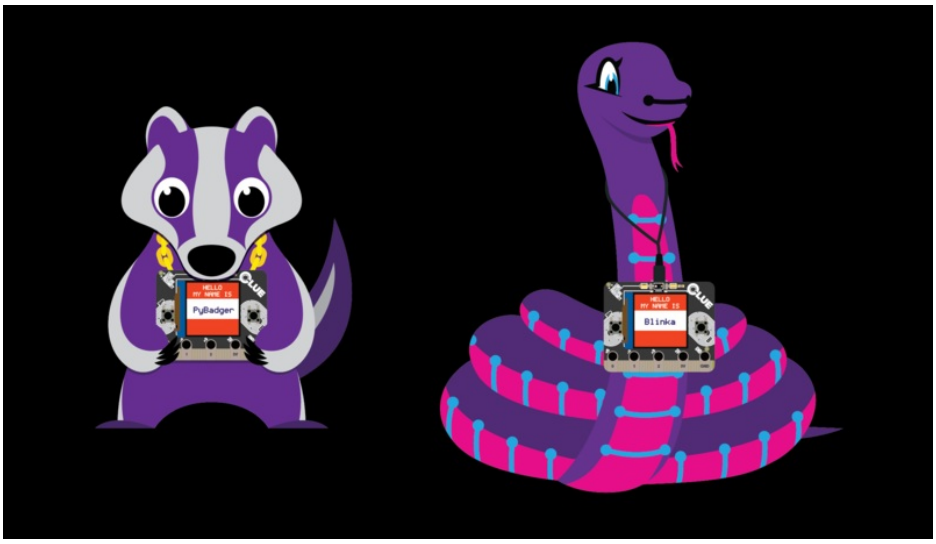


Last updated on 2020-09-11 02:07:53 PM EDT

Overview



It's easy to turn your CLUE into a custom conference or event badge using the Adafruit CircuitPython PyBadger library. The custom badge feature of PyBadger makes it simple to display multiple lines of text over a colored background or image. **This guide will show you how to create a badge displaying your Twitter handle, name, job title and pronouns with a color-block or even an image background!**



CircuitPython's Blinka is off to another conference with her friend PyBadger, and is looking forward to showing off her custom badge on her CLUE. Let's take a look!

Parts

Your browser does not support the video tag. Adafruit CLUE - nRF52840 Express with Bluetooth LE

$39.95
IN STOCK

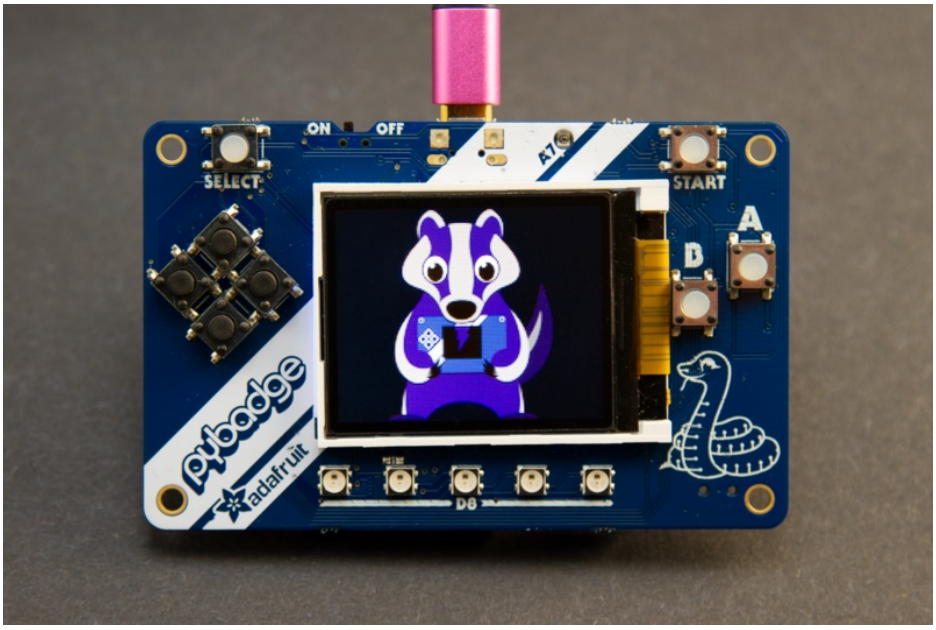
Add To Cart

USB cable - USB A to Micro-B

$2.95
IN STOCK

Add To Cart

Prepare Your Badge



Adafruit CircuitPython PyBadger makes it easy to create an interactive conference or event badge with PyBadge or PyGamer. Blinka is headed to a fun conference, and is excited to create a badge. Let's take a look!

Download PyBadger

Adafruit CircuitPython PyBadger requires the latest CircuitPython and a number of libraries to work.

To download CircuitPython, visit the following link for CLUE, PyBadge or PyGamer, depending on what you're using, and download the latest CircuitPython version for your board.

https://adafru.it/IHF

https://adafru.it/IHF

https://adafru.it/EF4

https://adafru.it/EF4

https://adafru.it/FxM

https://adafru.it/FxM

Next visit the following link and download the library bundle that matches your CircuitPython version.

https://adafru.it/ENC

https://adafru.it/ENC

PyBadger requires the following libraries to work. Download the library bundle and unzip the file. Open the folder, find the **lib** folder within, and open the **lib** folder. Copy the following folders and files to the **lib** folder on your **CIRCUITPY**

drive:

- **adafruit_bitmap_font**
- **adafruit_bus_device**
- **adafruit_display_shapes**
- **adafruit_display_text**
- **adafruit_miniqr.mpy**
- **adafruit_pybadger**
- **neopixel.mpy**

If you are using **PyBadge or PyGamer,** copy the following library:

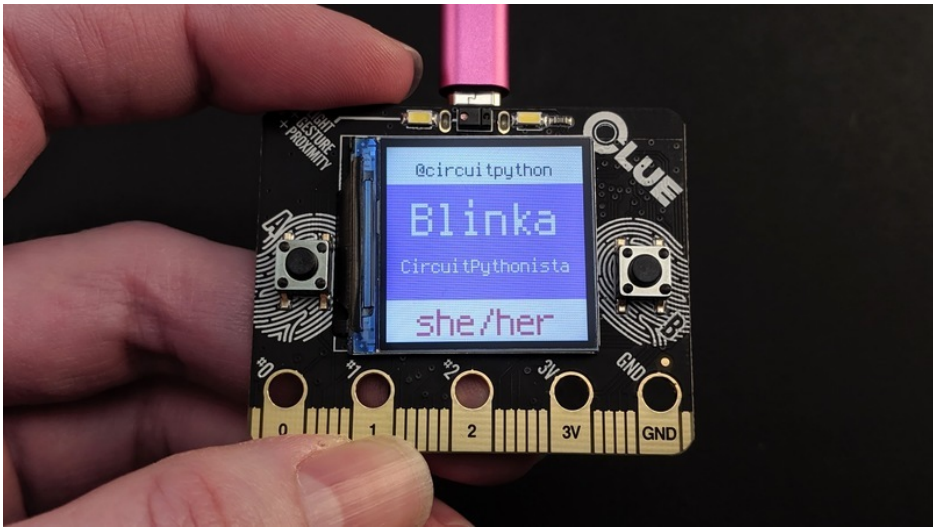
- **adafruit_lis3dh.mpy**

If you are using **CLUE**, copy the following library:

- **adafruit_lsm6ds.mpy**
- **adafruit_register**

Once you have the listed files and folders copied to the lib folder on your **CIRCUITPY** drive, you're ready to go!

CLUE Badge



Now that you have everything installed, we can get started with making a custom badge for CLUE!

Save the following example code as **code.py** on your **CIRCUITPY** drive.

```
"""Custom badge example for Adafruit CLUE."""
from adafruit_pybadger import pybadger

pybadger.badge_background(
    background_color=pybadger.WHITE,
    rectangle_color=pybadger.PURPLE,
    rectangle_drop=0.2,
    rectangle_height=0.6,
)

pybadger.badge_line(
    text="@circuitpython", color=pybadger.BLINKA_PURPLE, scale=2, padding_above=2
)
pybadger.badge_line(text="Blinka", color=pybadger.WHITE, scale=5, padding_above=3)
pybadger.badge_line(
    text="CircuitPythonista", color=pybadger.WHITE, scale=2, padding_above=2
)
pybadger.badge_line(
    text="she/her", color=pybadger.BLINKA_PINK, scale=4, padding_above=4
)

pybadger.show_custom_badge()

while True:
    if pybadger.button.a:
        pybadger.show_qr_code("https://circuitpython.org")

    if pybadger.button.b:
        pybadger.show_custom_badge()
```

Let's take a look at the code.

First, we import the PyBadger library.

```
from adafruit_pybadger import pybadger
```

Then we create the color-block badge background.

We set the background to white. Then we create a purple rectangle that is 60% of the display in height, and is located centered vertically, with the background being 20% of the display above and below the rectangle.

```
pybadger.badge_background(background_color=pybadger.WHITE,
                          rectangle_color=pybadger.PURPLE,
                          rectangle_drop=0.2, rectangle_height=0.6)
```

Next, we'll add the lines of text. We're going to include Blinka's Twitter handle, name, job title and preferred pronoun. You can include any info you like!

```
pybadger.badge_line(text="@circuitpython", color=pybadger.BLINKA_PURPLE, scale=2, padding_above=2)
pybadger.badge_line(text="Blinka", color=pybadger.WHITE, scale=5, padding_above=3)
pybadger.badge_line(text="CircuitPythonista", color=pybadger.WHITE, scale=2, padding_above=2)
pybadger.badge_line(text="she/her", color=pybadger.BLINKA_PINK, scale=4, padding_above=4)
```

Then we call `show_custom_badge()` to display the badge background and badge lines we set up.

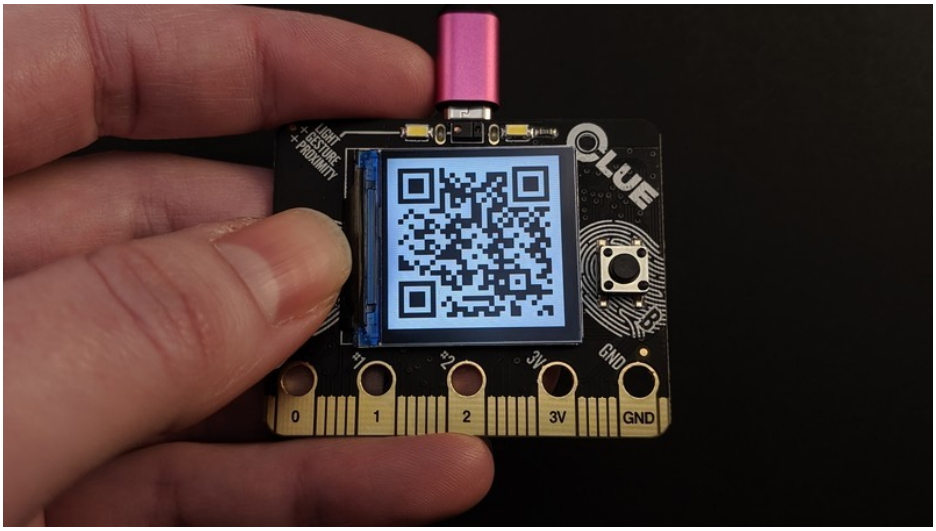
```
pybadger.show_custom_badge()
```

We call it before the loop so it displays on startup and we can display other things inside the loop.

Inside our loop, we're looking for button presses on button A and button B.

First, we check button A. If button A is pressed, we show a customisable QR code.

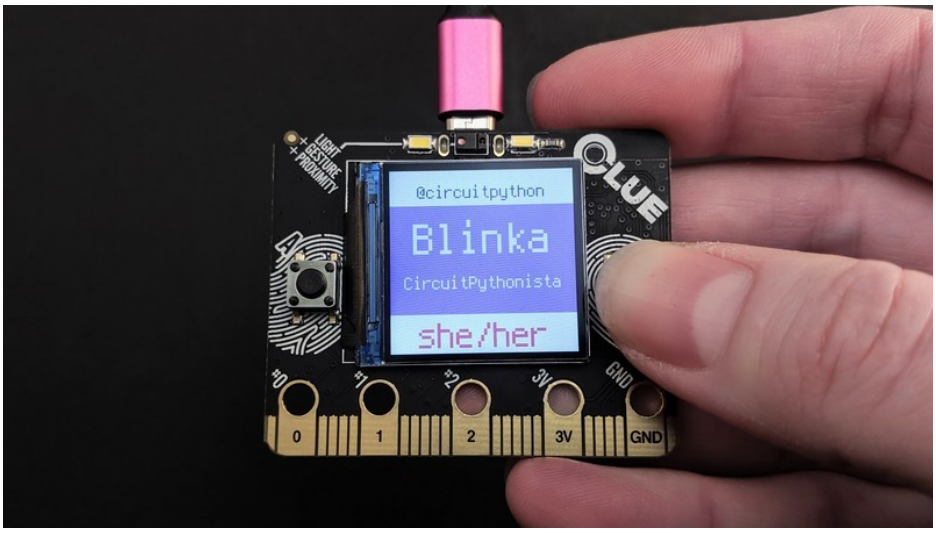
```
while True:
    if pybadger.button.a:
        pybadger.show_qr_code("https://circuitpython.org")
```



To change the target of the QR code, include a URL as a string (e.g. in quotation marks: `"https://circuitpython.org"`).

Finally, we check button B. If button B is pressed, we show the badge again.

```
[...]
    if pybadger.button.b:
        pybadger.show_custom_badge()
```



That's all there is to using creating a custom badge with a color-block background!

Now that you've seen how it works, let's take a more detailed look at the custom badge features of PyBadger.

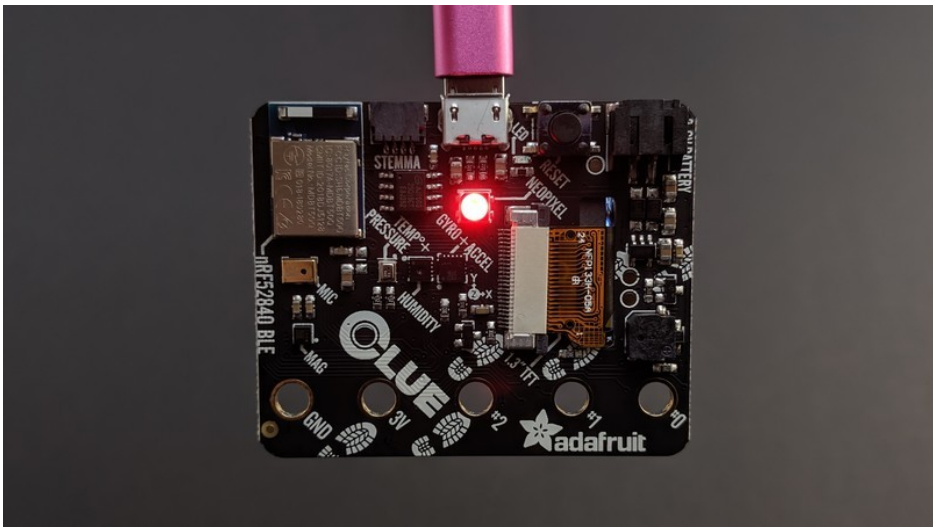
PyBadger Colors

PyBadger includes a set of colors available for use in your code. Each color is equivalent to an `(r, g, b)` value, e.g. `RED` is equal to `(255, 0, 0)`, and `GREEN` is equal to `(0, 255, 0)`. Pybadger colors can take the place of anything that expects an `(r, g, b)` tuple, including controlling the on-board NeoPixel! To use the PyBadger colors, simply include `pybadger.COLOR_NAME` in your code in place of a color tuple.

For example, to turn the NeoPixel on your CLUE red, you could save the following as **code.py**:

```
from adafruit_pybadger import pybadger

while True:
 pybadger.pixels.fill(pybadger.RED)
```



To create the color-block badge background as a white background with a purple rectangle over it, we included the following line in the custom badge code:

```
pybadger.badge_background(background_color=pybadger.WHITE,
                          rectangle_color=pybadger.PURPLE,
                          rectangle_drop=0.2, rectangle_height=0.6)
```

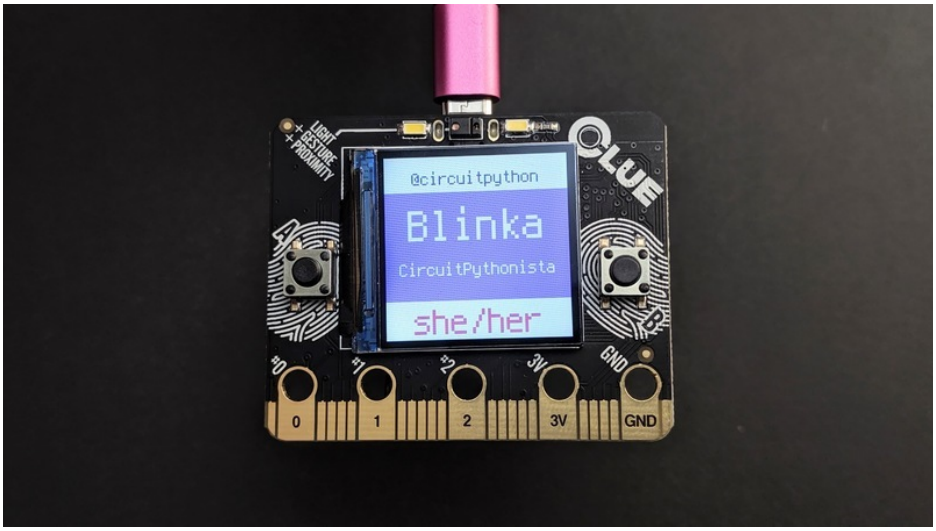
PyBadger makes a variety of colors super simple to use!

The colors available in PyBadger are:

- `RED`
- `YELLOW`
- `ORANGE`
- `GREEN, TEAL`
- `CYAN, BLUE`
- `PURPLE`
- `MAGENTA`
- `WHITE`

- `BLACK`
- `GOLD`
- `PINK`
- `AQUA`
- `JADE`
- `AMBER`
- `VIOLET`
- `SKY`
- `DEEP_PURPLE`
- `PYTHON_YELLOW`
- `PYTHON_BLUE`
- `BLINKA_PURPLE`
- `BLINKA_PINK`

Badge Background



The first thing we did was create the badge background.

```
pybadger.badge_background(background_color=pybadger.WHITE,
                          rectangle_color=pybadger.PURPLE,
                          rectangle_drop=0.2, rectangle_height=0.6)
```

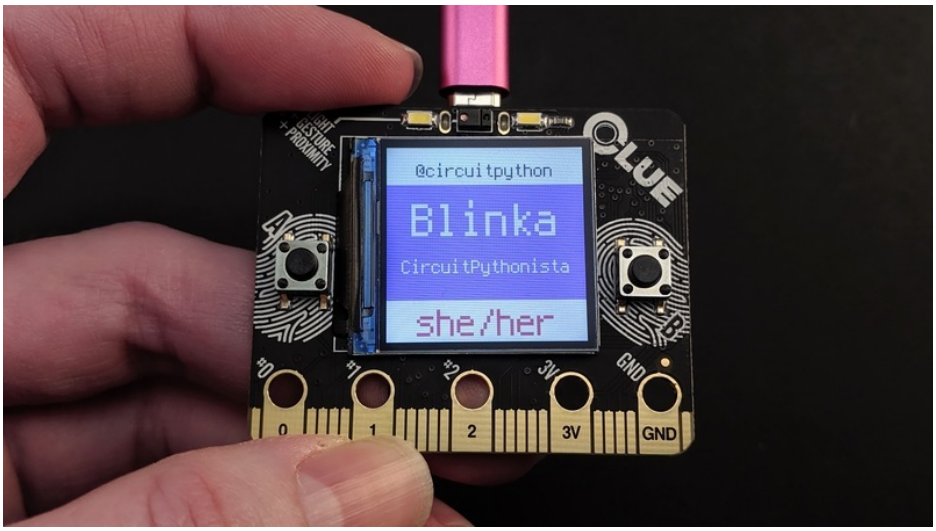
There are four aspects of the badge background you can customise:

- **background_color:** The color that fills the background on the display. Expects a tuple (e.g. `(r, g, b)`) or a `pybadger.COLOR`. Defaults to white. Check out the PyBadger Colors page for a list of available colors.
- **rectangle_color:** The color of a rectangle color-block that displays over the background. Expects a tuple (e.g. `(r, g, b)`) or a `pybadger.COLOR`. Defaults to red. Check out the PyBadger Colors page for a list of available colors.
- **rectangle_drop:** The distance from the top of the display that the rectangle begins. This expects a decimal number between 0 and 1 representing a percentage of the display, e.g. `0.2` means the rectangle will begin at a point 20% of the display height from the top of the display. Defaults to `0.4`.
- **rectangle_height:** The height of the rectangle. This expects a decimal number between 0 and 1 representing a percentage of the display, e.g. `0.4` means the rectangle height will be 40% of the display. Defaults to `0.5`.

The `rectangle_drop` and `rectangle_height` values are used to place the color-block rectangle on the display by using a decimal number (known as a float in Python) between 0 and 1 representing 0% to 100%. You specify the `rectangle_drop` to determine how far from the top the rectangle is located. That percentage of the display is filled with the background color above the rectangle. You specify the `rectangle_height` to determine its height. The remaining percentage of the display is background color shown below the rectangle. In our example, we set `rectangle_height` to `0.2` meaning 20%, and `rectangle_height` to `0.6` meaning 60%. 60% + 20% is 80%, leaving 20% left to display below the rectangle, resulting in it being centered vertically on the display.

If you would rather have the badge background be a single color, set `rectangle_color` to the desired color, `rectangle_drop=0`, and `rectangle_height=1`. This will make the background a single color.

Badge Lines



Next, we added the lines of text.

```
pybadger.badge_line(text="@circuitpython", color=pybadger.BLINKA_PURPLE, scale=2, padding_above=2)
pybadger.badge_line(text="Blinka", color=pybadger.WHITE, scale=5, padding_above=3)
pybadger.badge_line(text="CircuitPythonista", color=pybadger.WHITE, scale=2, padding_above=2)
pybadger.badge_line(text="she/her", color=pybadger.BLINKA_PINK, scale=4, padding_above=4)
```

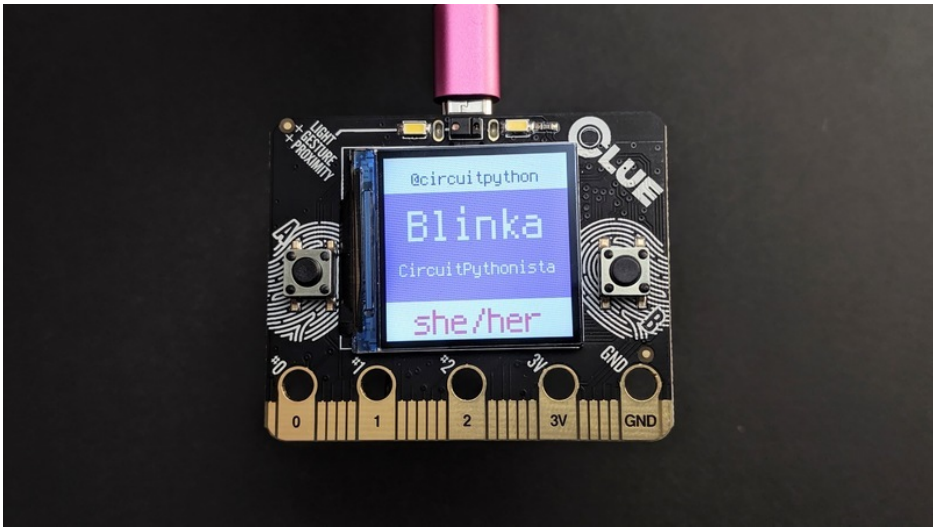
There are six aspects of each line of text you can customise:

- **text:** The text to display. Expects a string (text surrounded by quotation marks, e.g. `"Blinka"`).
- **color:** The color of the text. Expects a tuple (e.g. `(r, g, b)`), or a `pybadger.COLOR`. Check out the PyBadger Colors page for a list of available colors. If you set the text `color` to the same color as the part of the badge background it is displayed over, your text will not show up on the display.
- **scale:** The scale of the text. Expects a whole number 1 or higher. Defaults to `1`. Each increment multiplies the height of the text by the scale value, e.g. a scale of 3 means the text will be 3x higher than a scale of 1.
- **font:** The font used to display the text. Expects a string (e.g. the name of the font in quotation marks: `"Arial-16.bdf"`). Defaults to the built in `terminalio.FONT`. Custom fonts require additional files - check out this guide (https://adafru.it/E7E) for more information.
- **padding_above:** The amount of space to include before the line of text. Expects a whole number. Defaults to `0`. For example, setting `padding_above=1` includes an amount of space before the line equal to the height of the current font at scale 1, and setting `padding_above=2` includes an amount of space before the line equal to twice the height of the current font at scale 1, etc.
- **left_justify:** Set to `True` to left justify the text. Defaults to `False`. Text is centered by default.

Use `scale` to increase the size of the text. Use `padding_above` to space out your lines of text by increasing or decreasing the space above each line of text. Try changing the values to see how it affects the displayed text.

When substituting your own text, you may find it doesn't fit with the current values. It can take some experimenting with different values to get your text sized and placed exactly where you want it. The possibilities are endless!

Show Custom Badge



Finally, you must call `show_custom_badge()` to display the badge background and badge lines we set up.

```
pybadger.show_custom_badge()
```

Following the set up of `badge_background` and `badge_line`, you **MUST** include this line of code! `show_custom_badge()` triggers the background and text lines to display. Without it, nothing will display.

If you have no other code in your `while True:` loop, you should call `show_custom_badge()` inside the loop. For example, if all you wanted to do was show the badge, you could use the following code:

```
from adafruit_pybadger import pybadger

pybadger.badge_background(background_color=pybadger.WHITE,
                          rectangle_color=pybadger.PURPLE,
                          rectangle_drop=0.2, rectangle_height=0.6)

pybadger.badge_line(text="@circuitpython", color=pybadger.BLINKA_PURPLE,
                    scale=2, padding_above=2)
[...]

while True:
 pybadger.show_custom_badge()
```

If you have other code in the loop, you can call `show_custom_badge()` before the loop to display the badge on startup and allow for the code to display different things in the loop.

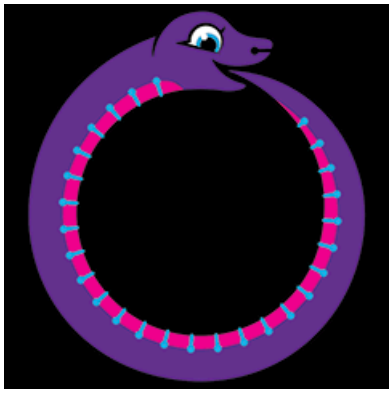
That's all there is to using creating a custom badge with a color-block background!

Image Background

Perhaps the color-block badge background doesn't suit your needs and you'd rather have the background be an image. This is easy with the PyBadger custom badge!

First, you'll need a compatible bitmap image. We've included one below. A couple more are available here (https://adafru.it/Jsd). For information on how to create your own compatible bitmaps, check out the Customization section of the Notifcation Icons page in this guide (https://adafru.it/Jfo).

Copy the desired bitmap to your **CIRCUITPY** drive.



Once the bitmap is copied to your **CIRCUITPY** drive, simply replace the `badge_background` line(s) with the following code.

```
pybadger.image_background("Blinka_CLUE.bmp")
```

You must provide the name of a compatible bitmap as a string (e.g. in quotation marks: `"Blinka.bmp"`).

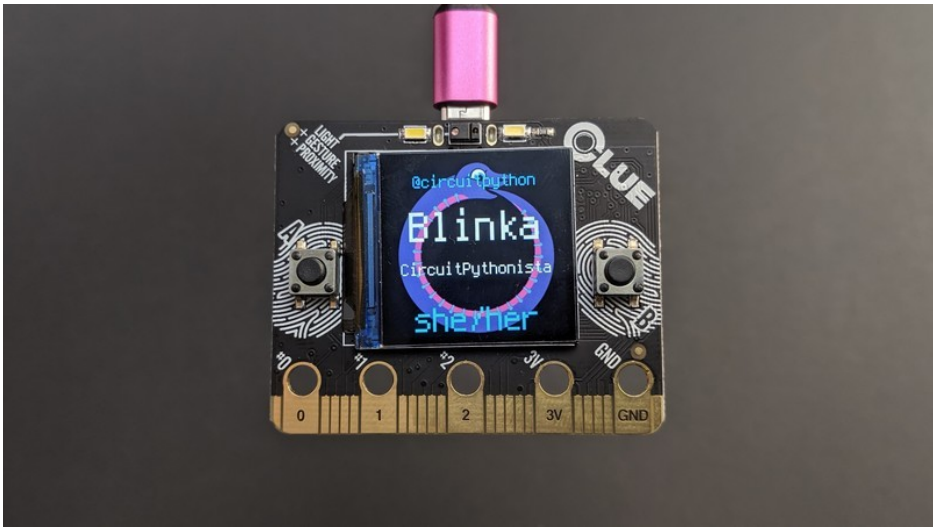
The rest of the badge set up is the same. Add your lines of text using `badge_line()` and then call `show_custom_badge()`.

```
"""Custom image badge example for Adafruit CLUE."""
from adafruit_pybadger import pybadger

pybadger.image_background("Blinka_CLUE.bmp")

pybadger.badge_line(text="@circuitpython", color=pybadger.SKY, scale=2, padding_above=2)
pybadger.badge_line(text="Blinka", color=pybadger.WHITE, scale=5, padding_above=3)
pybadger.badge_line(
    text="CircuitPythonista", color=pybadger.WHITE, scale=2, padding_above=2
)
pybadger.badge_line(text="she/her", color=pybadger.SKY, scale=4, padding_above=4)

while True:
    pybadger.show_custom_badge()
```



That's all there is to using creating a custom badge with an image background!