Circuit Playground Bike Light

Created by Carter Nelson

https://learn.adafruit.com/circuit-playground-bike-light

Last updated on 2021-11-15 06:54:41 PM EST

Table of Contents

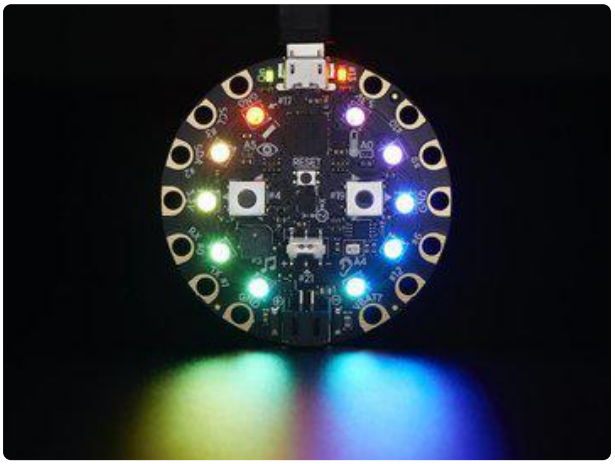
Overview

In this guide we will use our Circuit Playground to create a customizable bike light. We'll look at how we define colors in code and use that to make some fun animations with the NeoPixels. We'll provide some example animations you can customize. Or you can modify and adapt the examples to try and create your own.

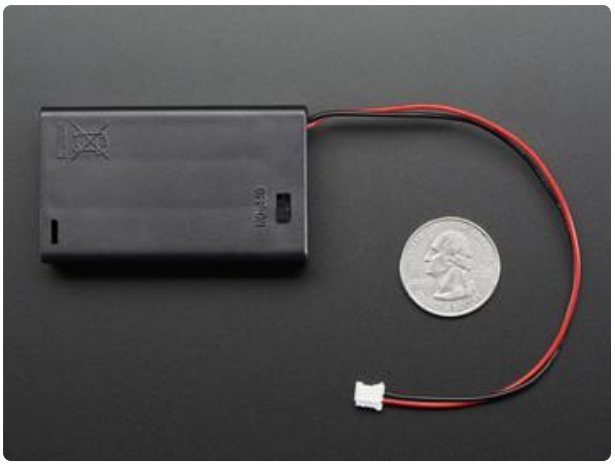
In a world of blinking red bike lights, add some color and pizazz to stand out!

Required Parts

In addition to a Circuit Playground, you will need some form of battery power so you can ride around outside with your bike light. Choose an option that works for how you plan to mount things to your bike.

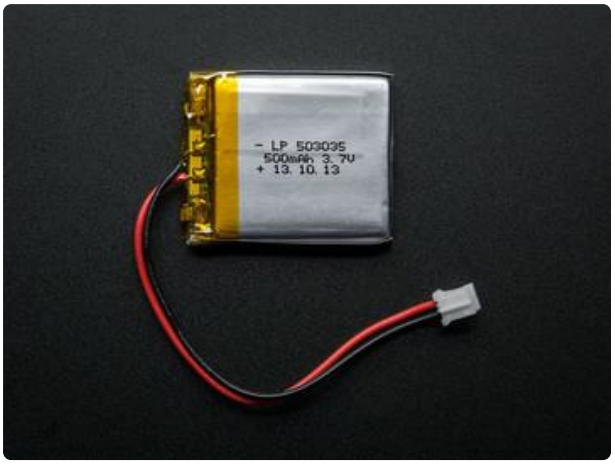


- Circuit Playground
  - Classic (http://adafru.it/3000)
  - Express (http://adafru.it/3333)



3 x AAA Battery Holder with On/Off Switch and 2-Pin JST (http://adafru.it/727)

(also need AAA batteries)



Lithium Ion Polymer Battery - 3.7v 500mAh (http://adafru.it/1578)

(or similar)

JST 2-pin Extension Cable with On/Off Switch (https://adafru.it/sPa)

(useful if using LiPo)

If you go with the LiPo battery, be sure you have a way to charge it (https://adafru.it/vof).

Before Starting

If you are new to the Circuit Playground, you may want to first read these overview guides.

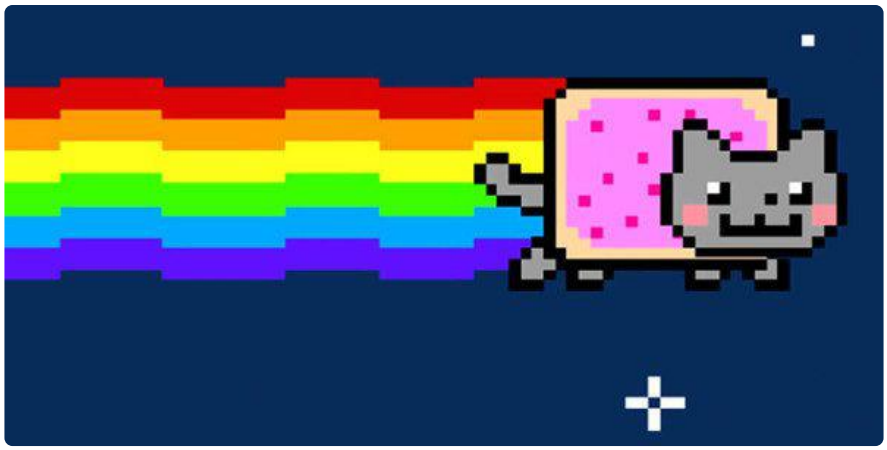
Circuit Playground Classic

- Overview (https://adafru.it/ncG)
- Lesson #0 (https://adafru.it/rb4)

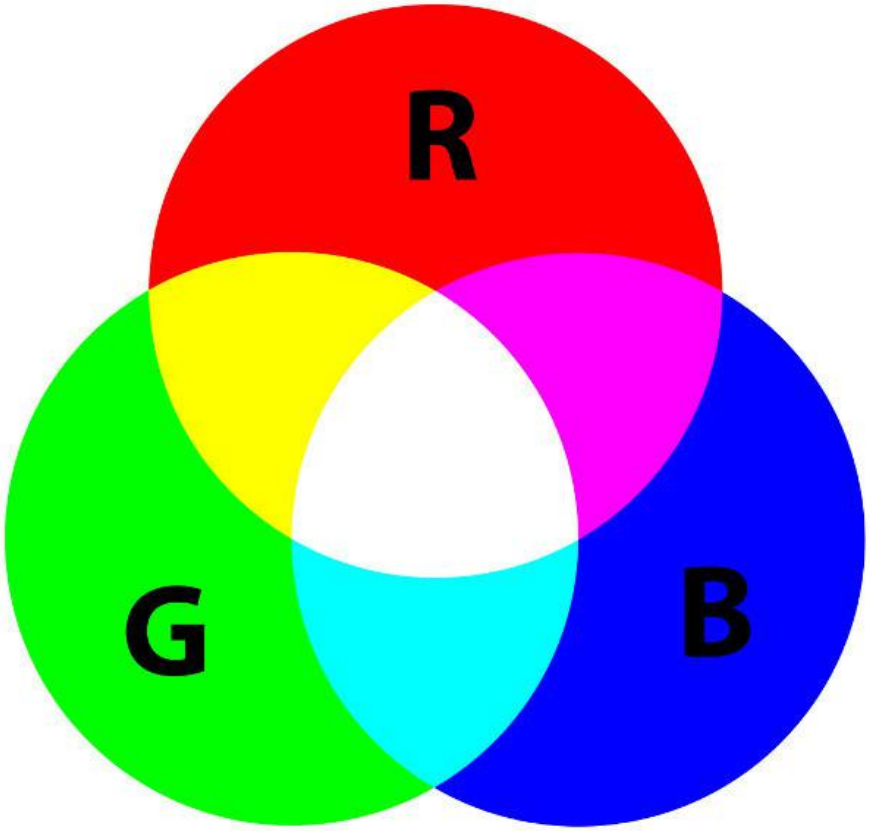
Circuit Playground Express

- Overview (https://adafru.it/AgP)

Hello Color

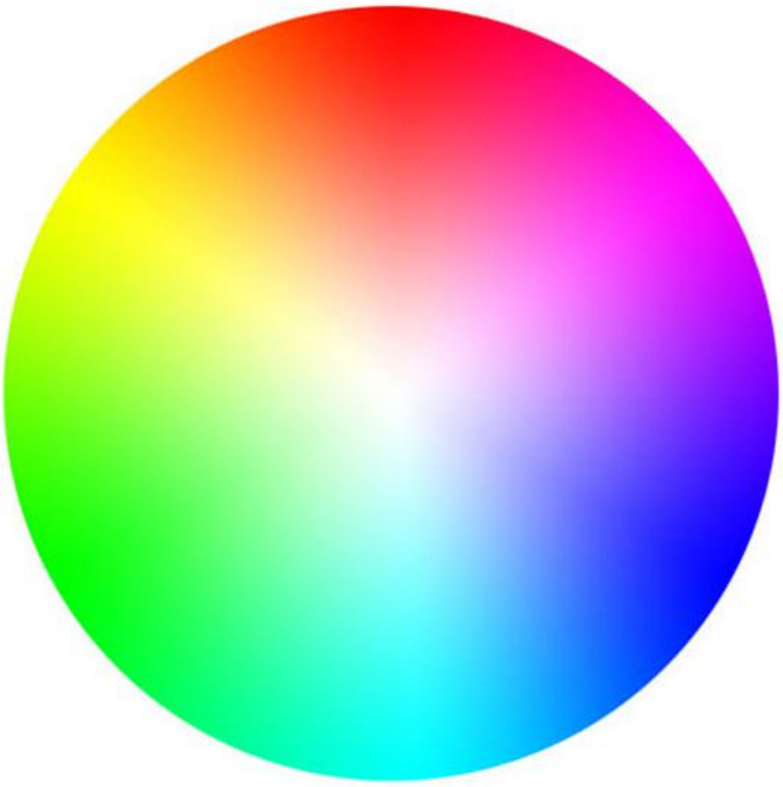


There's a rainbow of colors out there in the world. Purple. Lime green. Bright blues. Mellow yellows. etc. In order to be able to use them in our projects, we need some way to describe them. It turns out that all the colors of the rainbow can be described in terms of a combination of only three colors: Red, Green, and Blue. This is referred to as additive color (https://adafru.it/vGc).

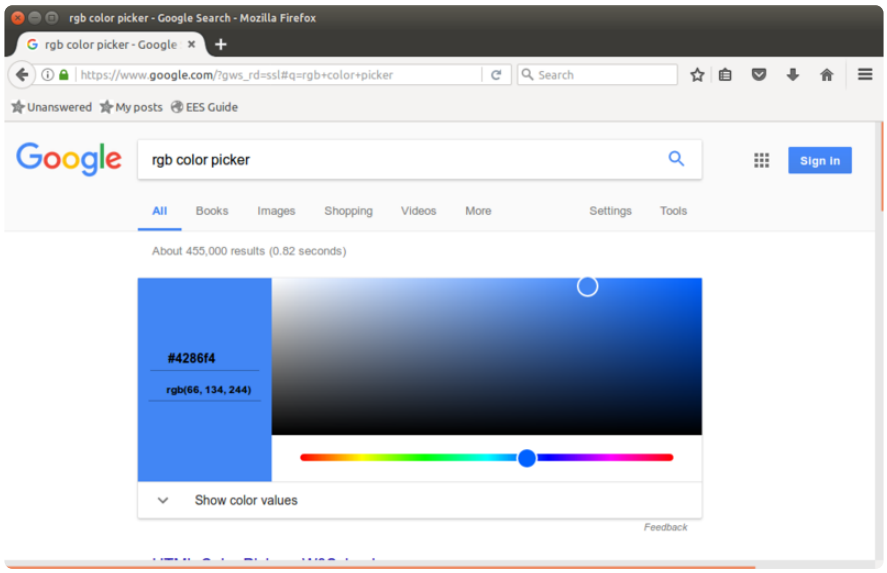


In the image above you can see the basic idea for how this works. Red is just red. Green is just green. Blue is just blue. But if you want yellow then you combine red and green. The purple color is called magenta and is a combination of red and blue. The light blue color is called cyan and is a combination of green and blue. What happens if you combine all three? Well, you get white, as shown in the middle of the diagram.

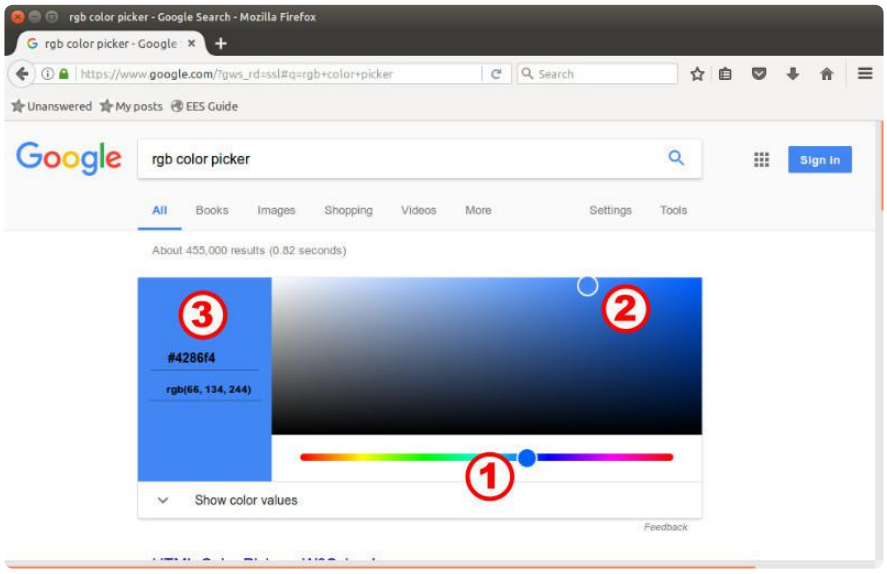
But we can also create many more colors by picking how much we mix each of the red, blue, and green together. Here is a much more colorful wheel of color.



For example, orange is mostly red with a little bit of green. Pink is mostly red with a little bit of blue. How do we come up with the right mixture of red, green, and blue for all of these pretty colors? Luckily there are lots of handy tools to help us out. In fact, you can get a very easy to use one by simply going to Google (https://adafru.it/vGd) and doing a search for 'rgb color picker'.



Of course you'll get a lot of search results as well. But this is such a commonly used tool, that Google just throws one up for you to use right away. Here's a breakdown of how to use it.



Use slider (1) to pick a color. Then move slider (2) around to change how light or dark the color is. The resulting color is shown in (3) along with some cryptic numbers.

Let's look at what those numbers mean.

Colors in Code

The color picker above provides information on the current color in two formats. The one that looks like #4286f4 is referred to as 'hex', which is short for hexadecimal. The one that looks like rgb(66, 134, 244) is referred to as 'rgb' (are-gee-bee), r for red, g for green, b for blue. They are both describing the same color but in different ways. In code, you'll typically use one form or the other, or sometimes swap between them. Basically:

- Use hex to describe the color with a single value
- Use rgb to more intuitively describe a color or for easy access to a specific color channel

RGB Color Codes

This one is pretty easy to understand. You just specify the three separate values for red, green, and blue. For example:

rgb(66, 134, 244)

is setting red to 66, green to 134, and blue to 244. The values can range from 0 to 255. What's so magic about 255? Why not 100 or 1000? It has to do with these being 8-bit values. In this coloring scheme, each color is specified by 8-bits. And the range of values that can be specified by 8-bits is 0 to 255 (2^8-1). Here is how you would specify the three primary colors:

```
CircuitPlayground.setPixelColor(0, 255,   0,   0);  // red
CircuitPlayground.setPixelColor(0,   0, 255,   0);  // green
CircuitPlayground.setPixelColor(0,   0,   0, 255);  // blue
```

But of course you can specify any random combination of values, with each ranging from 0 to 255.

```
CircuitPlayground.setPixelColor(0,  98,  12, 143);  // ??
CircuitPlayground.setPixelColor(0,   0, 251,   1);  // ??
CircuitPlayground.setPixelColor(0,  176, 42,  23);  // ??
```

And just if you're curious, there are 2^24 = 16,777,216 possible color combinations. Thats a lot of colors, yo!

HEX Color Codes Part 1

This one is a little more difficult to understand. First, you need to understand hexadecimal. We all grow up using a decimal numbering system. This means the numbering system is based on ten unique values, which (in English) we give the symbols: 0,1,2,3,4,5,6,7,8,9. With hexadecimal, the numbering system has sixteen unique values instead of ten. To represent them, the first ten are the same as before and the remaining six use the first six letters of the alphabet. So we have 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Let's count to 20 using both decimal (DEC) and hexadecimal (HEX).

DEC	HEX
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
18	12
19	13
20	14
:	:
255	FF

Note how everything is the same up to 9 (i.e. the first ten values). Once we reach 10, we've run out of decimal digits, so we need to increment. However, we haven't run out of hexadecimal digits, so we just keep counting. We run out of hexadecimal digits when we reach 16, at which point we increment like we did when we reached 10 in decimal.

One key thing to note is what the maximum 8-bit value looks like in hex. Remember this was 255 in decimal. That becomes FF in hex. To go one higher, say to 256, first we'd need an extra bit, and then the hex would become 100.

Confusing? Yeah it can be. It can take some time, but eventually you'll get the idea. There are special calculators called programmers calculators that help out with these conversions.

You can also get away with just typing the conversion in to Google. For example, try Googling the phrase "256 in hex".

Is it DEC or HEX?

We also need some way of specifying to our program if we are using decimal or hexadecimal. For example, consider the following assignment:

```
someValue = 10;
```

Is that the decimal 10 we all know and love, or is it that weird 10 in hexadecimal which is actually 16 in decimal? To sort this out, special character sequences are used in front of the number to let the program know which one it is. The default is decimal, so we just need something for hexadecimal. In Arduino programs, we prefix hexadecimal numbers with 0x. Then we can do things like this:

```
someValue1 = 10;    // this is decimal 10
someValue2 = 0x10;  // this is hexadecimal 10 (decimal 16)
```

HEX Color Codes Part 2

The idea with the HEX color code is to combine the red, green, and blue values into one value. This way we can specify a color with a single value, instead of three separate values.

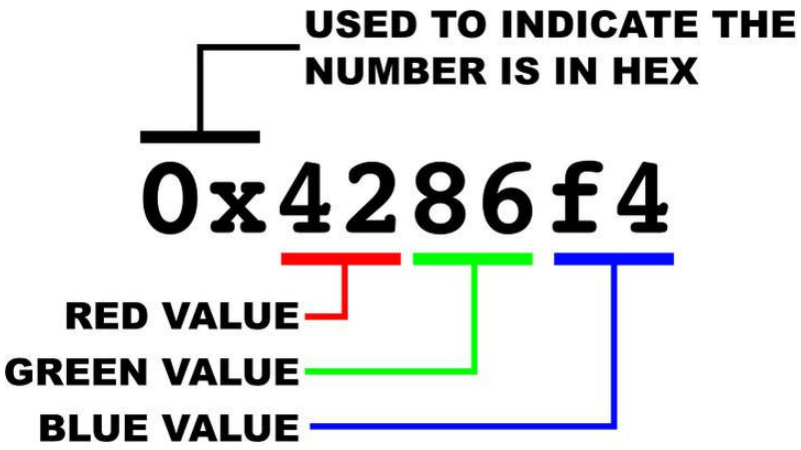
Since each color uses 8-bits, we end up needing 24-bits to store the information. You can think of it as just chaining the three separate 8-bits of color info into one 24-bit sequence. Red is 'first', then green, and finally blue. Something like this:

0xRRGGBB

Since we are using hexadecimal to specify each 8-bit value, the range for each is from 0x00 to 0xFF. For example, here is how you would specify the three primary colors:

```
CircuitPlayground.setPixelColor(0, 0xFF0000);  // red
CircuitPlayground.setPixelColor(0, 0x00FF00);  // green
CircuitPlayground.setPixelColor(0, 0x0000FF);  // blue
```

And others colors will be a mixture of values. Here's the one from the color picker:



Which would look like this in code:

```
CircuitPlayground.setPixelColor(0, 0x4286f4);
```

Note that case typically doesn't matter when specifying hex numbers. That is:

0xff = 0xFF

Picking one or the other is just a matter of style.

More Info

If you want more info on decimal, hexadecimal, and even binary, check out this guide and associated video.

Collin's Lab: Binary & Hex

https://adafru.it/wdz

Nerdy Nerd Joke

Decimal is base ten. Hexadecimal is base sixteen. But you can actually have a numbering system with any base you want. Humans tend to like base ten because we have ten fingers. Computers really like base two, otherwise known as binary, because in their world things are either on or off. And in a base two universe, you can tell jokes like this:

> There are 10 kinds of people in this world. Those that know binary and those that don't.

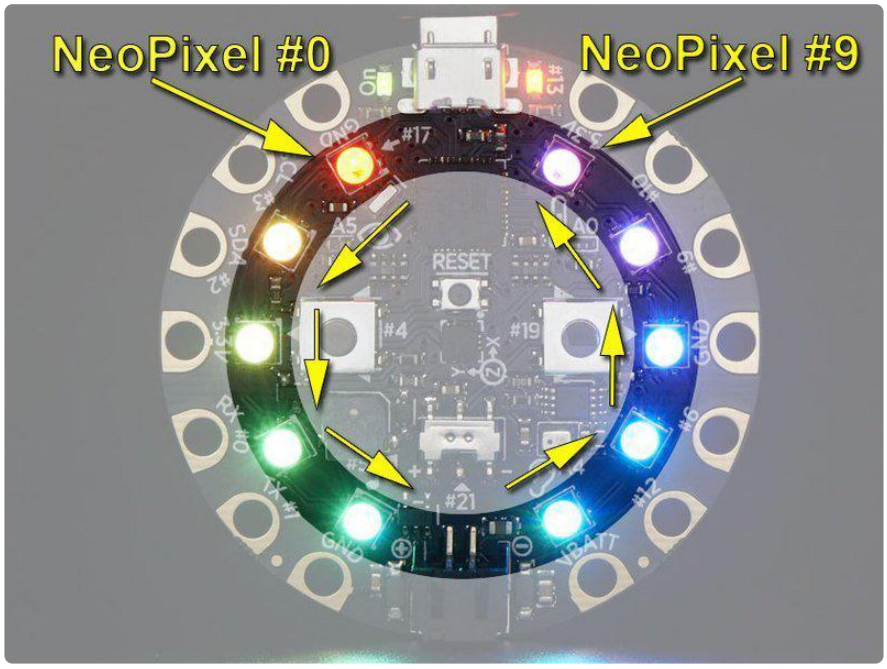
Hello NeoPixel

Be careful not to stare directly into a brightly lit NeoPixel.

Don't do this with your eyes, but let's look inside a NeoPixel to see what's going on. Here's a nice macro photo of a single NeoPixel.

There are three tiny little LEDs in there. A blue one, a red one, and a green one. Each one is controlled by the circuit next to them which acts like a brain. This is what receives the NeoPixel commands and does the actual turning on of the LEDs.

The Circuit Playground has 10 of these NeoPixels arranged in a circular pattern as shown below. All we need to do is write programs to tell them what colors we want.



So we can use the red, green, and blue LEDs in each NeoPixel to generate various colors. We just specify a color and the NeoPixel will take care of turning on the three LEDs in the right amount to generate that color.

Library Reference

There are two main functions in the Circuit Playground library for use with NeoPixels, `clearPixels()` and `setPixelColor()`. The first one simply turns off all of the NeoPixels. The second one is more fun, as it is how we turn on NeoPixels and specify a color. You can use it with either the rgb color values or the hex value to specify a color. Check out the library reference here (https://adafru.it/vGe) and here (https://adafru.it/vGf) and the example sketch that comes with the library:

File -> Examples -> Adafruit Circuit Playground -> Hello_CircuitPlayground -> Hello_NeoPixels

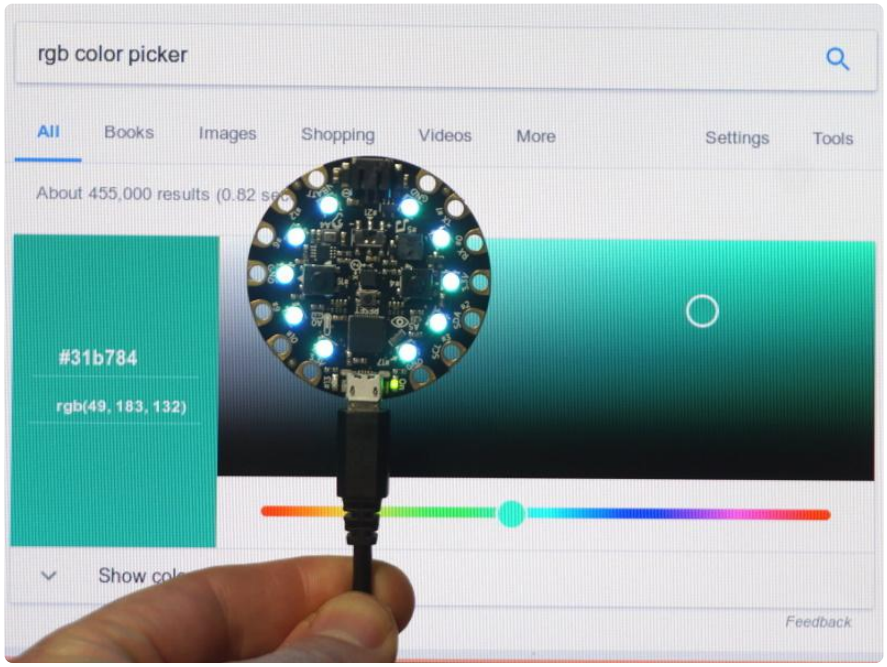
Here's a simple program that turns on all the NeoPixels to the specified `COLOR`. You can use this to play around with various color values.

```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Bike Light - Color Demo
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define COLOR  0x31b784

////////////////////////////////////////////////////////////////////////
void setup(  // Turn one on using (r,g,b) values
) {
  CircuitPlayground.begin();

  // Set all NeoPixels to COLOR
  for (int pixel=0; pixel&lt;10; pixel++) {
    CircuitPlayground.setPixelColor(pixel, COLOR);
  }
}


////////////////////////////////////////////////////////////////////////
void loop() {
  // do nothing
}
```

Use the color picker from the previous selection and find a color you like. Then take the hex value shown and modify the following line of code. Note that the Google color picker has a # in front of the value. Be sure to change that to 0x in the code as shown:

```
#define COLOR  0x31b784
```

Here's what I got when I tried out that value.



They kind of look the same. A computer monitor and a NeoPixel are pretty different, so it's not going to be an exact match. But at least it's not red or orange or something.

Turning Off a Single NeoPixel

There's only one command for turning off the NeoPixels, `clearPixels()`, and it turns them all off. But what if you wanted to turn just one NeoPixel off. The answer is pretty simple. 'Off' is just no color, or the equivalent for 'black'. So we use the `setPixelColor()` command on the pixel we want and specify the values of 0x000000 or (0, 0, 0). For example, here's how to turn off NeoPixel 6:

```
CircuitPlayground.setPixelColor(6, 0x000000);
```

Setting Brightness

The one other useful NeoPixel function is `setBrightness()`. It can take a value from 0 to 255, with higher numbers being brighter. You typically call this just once at the beginning of your code, in the `setup()` function, to set the overall brightness for all NeoPixels. It can not be used to change the brightness of a single NeoPixel.

Basic NeoPixel Animation

All NeoPixel animation comes down to simply turning on and off the pixels and/or changing their color. That's it. Everything is done by playing with three basic things:

- Location - What pixel(s) to turn on/off?
- Color - What color to use?
- Time - How long to stay on/off?

Figuring out the code for complex animations can be tricky. In that case, taking an approach as described in the NeoAnim guide can help.

NeoAnim Guide

https://adafru.it/vGA

However, for our bike light, we will keep things simple and just use a basic loop. By changing various parameters within the loop, we can create some fun animations that will make for a cool bike light.

For example, here's a basic spinner animation.

```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Bike Light - Round N Round
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define COLOR         0xFF0000    // change this to your favorite color
#define RATE          100         // lower is faster

////////////////////////////////////////////////////////////////////////
void setup(  // Turn one on using (r,g,b) values
) {
  CircuitPlayground.begin();

  // Make it bright!
  CircuitPlayground.setBrightness(255);
}


////////////////////////////////////////////////////////////////////////
void loop() {
  // The animation loop
  for (int pixel=0; pixel&lt;10; pixel++) {
    // Turn OFF all pixels
    CircuitPlayground.clearPixels();

    // Turn ON the current pixel to COLOR
    CircuitPlayground.setPixelColor(pixel, COLOR);
```

```
    // Don't spin too fast
    delay(RATE);
  }
}
```

Our location parameter of the animation is controlled by the `pixel` value in the loop.

```
  for (int pixel=0; pixel&lt;10; pixel++) {
    // animation stuff goes here
  }
```

Both the color parameter (`COLOR`) and time parameter (`RATE`) of the animation are set by global values at the top of the sketch.

```
#define COLOR         0xFF0000    // change this to your favorite color
#define RATE          100         // lower is faster
```

Play around with changing the values for `COLOR` and `RATE` and see what happens. You can use the color picker to come up with the color values.

More 'complex' animations can be created by doing things like having more than one loop, changing more than one pixel at a time, using more than one color, etc. What follows are several examples to illustrate this. After playing with them and looking through them, try and come up with your own animation.

The Flasher

This first one is your basic classic blinking flasher. Just like pretty much every bike light out there. Basically just turn all the NeoPixels on, then turn them off, then back on again, etc.

```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Bike Light - Flasher
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define COLOR         0xFF0000   // change this to your favorite color
#define FLASH_RATE    250        // lower is faster


////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();

  // Make it bright!
  CircuitPlayground.setBrightness(255);
}
```

```
////////////////////////////////////////////////////////////////////////
void loop() {
  // Turn on all the pixels to COLOR
  for (int pixel=0; pixel&lt;10; pixel++) {
    CircuitPlayground.setPixelColor(pixel, COLOR);
  }

  // Leave them on for a little bit
  delay(FLASH_RATE);

  // Turn off all the NeoPixels
  CircuitPlayground.clearPixels();

  // Leave them off for a little bit
  delay(FLASH_RATE);
}
```

Pretty simple, but it does get people's attention. Play around with these two lines of code:

```
#define COLOR         0xFF0000   // change this to your favorite color
#define FLASH_RATE    250        // lower is faster
```

Try changing the `COLOR` to 0xff07ee. What color is that? Want to make it blink faster? Try lowering `FLASH_RATE` down to say something like 100 or 50.

Congratulations. You now have a bike light you can customize to your favorite color!

But why just have a boring old blink blink blink? We can do so much more...

The Spinner

This one has kind of a cool UFO look to it. Set any two pixels (0 to 9) and they will go round and round. Change the color to whatever you want and speed it up or slow it down to your liking.

```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Bike Light - Spinner
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define COLOR         0xFF0000    // change this to your favorite color
#define SPIN_RATE     100         // lower is faster

int pixel1;
int pixel2;

////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();
```

```
  // Make it bright!
  CircuitPlayground.setBrightness(255);

  // Can be any two pixels
  pixel1 = 0;
  pixel2 = 5;
}


////////////////////////////////////////////////////////////////////////////
void loop() {
  // Turn off all the NeoPixels
  CircuitPlayground.clearPixels();

  // Turn on two pixels to COLOR
  CircuitPlayground.setPixelColor(pixel1, COLOR);
  CircuitPlayground.setPixelColor(pixel2, COLOR);

  // Increment pixels to move them around the board
  pixel1 = pixel1 + 1;
  pixel2 = pixel2 + 1;

  // Check pixel values
  if (pixel1 &gt; 9) pixel1 = 0;
  if (pixel2 &gt; 9) pixel2 = 0;

  // Wait a little bit so we don't spin too fast
  delay(SPIN_RATE);
}
```

The Cylon

Want to look like an evil robot (https://adafru.it/vGB) and/or a cool car from the 80's (https://adafru.it/vGC)? By your command. I mean, OK Michael. I mean...oh never mind. Here's the code. It scans the pixels back and forth. This one looks best with the scans moving horizontally.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Bike Light - Cylon
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define COLOR         0xFF0000    // change this to your favorite color
#define SCAN_RATE     100         // lower is faster

int pixel1;
int pixel2;

////////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();

  // Make it bright!
  CircuitPlayground.setBrightness(255);

  pixel1 = 0;
  pixel2 = 9;
}
```

```
////////////////////////////////////////////////////////////////////////
void loop() {
  // Scan in one direction
  for (int step=0; step&lt;4; step++) {
    CircuitPlayground.clearPixels();

    CircuitPlayground.setPixelColor(pixel1, COLOR);
    CircuitPlayground.setPixelColor(pixel2, COLOR);

    pixel1 = pixel1 + 1;
    pixel2 = pixel2 - 1;

    delay(SCAN_RATE);
  }

  // Scan back the other direction
  for (int step=0; step&lt;4; step++) {
    CircuitPlayground.clearPixels();

    CircuitPlayground.setPixelColor(pixel1, COLOR);
    CircuitPlayground.setPixelColor(pixel2, COLOR);

    pixel1 = pixel1 - 1;
    pixel2 = pixel2 + 1;

    delay(SCAN_RATE);
  }
}
```

The Bedazzler

The idea with this one is to just randomly flash random colors on random pixels. This is a good example of where using the (r,g,b) version of `setPixelColor()` is useful. We'll generate a random red value, a random green value, and a random blue value. Then we just call `setPixelColor(pixel, r, g, b)` for a random pixel.

This could have been done using the hex value, but coding this way is more readable and intuitive.

```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Bike Light - Bedazzler
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define BEDAZZLE_RATE   100     // lower is faster

////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();

  // Make it bright!
  CircuitPlayground.setBrightness(255);
}

////////////////////////////////////////////////////////////////////////
```

```
void loop() {
  // Turn off all the NeoPixels
  CircuitPlayground.clearPixels();

  // Turn on a random pixel to a random color
  CircuitPlayground.setPixelColor(
    random(10),     // the pixel
    random(256),    // red
    random(256),    // green
    random(256) );  // blue

  // Leave it on for a little bit
  delay(BEDAZZLE_RATE);
}
```

The Rainbow Chaser

OK, now we're getting a little more colorful. Specify 10 colors using hex values in the `colors[]` array. These colors will be displayed on the 10 NeoPixels and the pattern will be rotated round and round.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Bike Light - Rainbow Chaser
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define ROTATION_RATE   50    // lower is faster

// Define 10 colors here.
// Must be 10 entries.
// Use 0x000000 if you want a blank space.
uint32_t colors[] = {
  0xFF0000,
  0xFF5500,
  0xFFFF00,
  0x00FF00,
  0x0000FF,
  0xFF00FF,
  0x000000,
  0x000000,
  0x000000,
  0x000000
};

int colorIndex;
int startIndex;

////////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();

  // Make it bright!
  CircuitPlayground.setBrightness(255);

  // Start at the beginning
  startIndex = 0;
}
```

```
////////////////////////////////////////////////////////////////////////
void loop() {
  // Turn off all the NeoPixels
  CircuitPlayground.clearPixels();

  // Loop through and set pixels
  colorIndex = startIndex;
  for (int pixel=0; pixel&lt;10; pixel++) {
    CircuitPlayground.setPixelColor(pixel, colors[colorIndex]);
    colorIndex++;
    if (colorIndex &gt; 9) colorIndex = 0;
  }

  // Increment start index into color array
  startIndex++;

  // Check value and reset if necessary
  if (startIndex &gt; 9) startIndex = 0;

  // Wait a little bit so we don't spin too fast
  delay(ROTATION_RATE);
}
```

The All Of Them

Now what if you're out riding around and get tired of the current animation and want to change it. Ride all the way home and reload a different animation? Nah. Let's see how we can have them all loaded and then choose the one we want using the buttons on the Circuit Playground.

Doing Two (or more) Things At Once

As simple as the idea of changing animations with a button press is, it has some surprising complexities. At it's lowest level, the brain on the Circuit Playground can only do one thing at a time. So if it's busy turning NeoPixels on and off, how can it check on button presses?

There are various approaches. For our bike light, we will use the simplest of them. However, for an excellent overview of more complex approaches (including using interrupts), check out this three part series. The ideas discussed in these guides could be used on the Circuit Playground.

Multi-Tasking Arduino Part 1

https://adafru.it/vGD

Multi-Tasking Arduino Part 2

https://adafru.it/mEe

What we will do is take advantage of the fact that all of our animation loops run fairly fast. This makes it possible to simply check for any button presses at the beginning (or end) of the animation loop. Then, if a button is pressed, take some action - like move to the next animation. This isn't perfect and has some issues, as you'll see.

To do this, we first take all of the animation code from our previous sketches and move them into functions that have the form:

```
void animation1() {
  while (!buttonsPressed()) {
    // animation code goes here
  }
}
```

We'll create a function for each animation. The `while()` loop will drive the animation and will continue as long as no buttons are pressed.

Then, our main Arduino `loop()` will look something like this:

```
void loop() {
  animation1(); delay(250);
  animation2(); delay(250);
  animation3(); delay(250);
  // etc
}
```

So the first animation is launched with a call to `animation1()`. That will run until a button is pressed. When a button is pressed, the animation function exits and the next animation function `animation2()` is called. Which runs until a button is pressed, etc.

The `delay()` is needed so that you don't cycle through the animations super fast when you press a button. This is a simple form of debounce.

Here's the final code:

```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Bike Light - The All Of Them
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

// Change these to set speed (lower is faster)
#define FLASH_RATE    250
```

```
#define SPIN_RATE     100
#define CYLON_RATE    100
#define BEDAZZLE_RATE 100
#define CHASE_RATE    100

// Change these to be whatever color you want
// Use color picker to come up with hex values
#define FLASH_COLOR   0xFF0000
#define SPIN_COLOR    0xFF0000
#define CYLON_COLOR   0xFF0000

// Define 10 colors here.
// Must be 10 entries.
// Use 0x000000 if you want a blank space.
uint32_t rainbowColors[] = {
  0xFF0000,
  0xFF5500,
  0xFFFF00,
  0x00FF00,
  0x0000FF,
  0xFF00FF,
  0x000000,
  0x000000,
  0x000000,
  0x000000
};

////////////////////////////////////////////////////////////////////////////
bool buttonsPressed() {
  return CircuitPlayground.leftButton() | CircuitPlayground.rightButton();
}

////////////////////////////////////////////////////////////////////////////
void flasher() {
  while (!buttonsPressed()) {
    // Turn on all the pixels to FLASH_COLOR
    for (int pixel=0; pixel&lt;10; pixel++) {
      CircuitPlayground.setPixelColor(pixel, FLASH_COLOR);
    }

    // Leave them on for a little bit
    delay(FLASH_RATE);

    // Turn off all the NeoPixels
    CircuitPlayground.clearPixels();

    // Leave them off for a little bit
    delay(FLASH_RATE);
  }
}

////////////////////////////////////////////////////////////////////////////
void spinner() {
  // Can be any two pixels
  int pixel1 = 0;
  int pixel2 = 5;

  while (!buttonsPressed()) {
    // Turn off all the NeoPixels
    CircuitPlayground.clearPixels();

    // Turn on two pixels to SPIN_COLOR
    CircuitPlayground.setPixelColor(pixel1, SPIN_COLOR);
    CircuitPlayground.setPixelColor(pixel2, SPIN_COLOR);

    // Increment pixels to move them around the board
    pixel1 = pixel1 + 1;
    pixel2 = pixel2 + 1;
```

```
    // Check pixel values
    if (pixel1 &gt; 9) pixel1 = 0;
    if (pixel2 &gt; 9) pixel2 = 0;

    // Wait a little bit so we don't spin too fast
    delay(SPIN_RATE);
  }
}

////////////////////////////////////////////////////////////////////////////
void cylon() {
  int pixel1 = 0;
  int pixel2 = 9;

  while (!buttonsPressed()) {
    // Scan in one direction
    for (int step=0; step&lt;4; step++) {
      CircuitPlayground.clearPixels();

      CircuitPlayground.setPixelColor(pixel1, CYLON_COLOR);
      CircuitPlayground.setPixelColor(pixel2, CYLON_COLOR);

      pixel1 = pixel1 + 1;
      pixel2 = pixel2 - 1;

      delay(CYLON_RATE);
    }

    // Scan back the other direction
    for (int step=0; step&lt;4; step++) {
      CircuitPlayground.clearPixels();

      CircuitPlayground.setPixelColor(pixel1, CYLON_COLOR);
      CircuitPlayground.setPixelColor(pixel2, CYLON_COLOR);

      pixel1 = pixel1 - 1;
      pixel2 = pixel2 + 1;

      delay(CYLON_RATE);
    }
  }
}

////////////////////////////////////////////////////////////////////////////
void bedazzler() {
  while (!buttonsPressed()) {
    // Turn off all the NeoPixels
    CircuitPlayground.clearPixels();

    // Turn on a random pixel to a random color
    CircuitPlayground.setPixelColor(
      random(10),     // the pixel
      random(256),    // red
      random(256),    // green
      random(256) );  // blue

    // Leave it on for a little bit
    delay(BEDAZZLE_RATE);
  }
}

////////////////////////////////////////////////////////////////////////////
void rainbow() {
  // Start at the beginning
  int startIndex = 0;
  int colorIndex;

  while (!buttonsPressed()) {
    // Turn off all the NeoPixels
```

```
    CircuitPlayground.clearPixels();

    // Loop through and set pixels
    colorIndex = startIndex;
    for (int pixel=0; pixel&lt;10; pixel++) {
      CircuitPlayground.setPixelColor(pixel, rainbowColors[colorIndex]);
      colorIndex++;
      if (colorIndex &gt; 9) colorIndex = 0;
    }

    // Increment start index into color array
    startIndex++;

    // Check value and reset if necessary
    if (startIndex &gt; 9) startIndex = 0;

    // Wait a little bit so we don't spin too fast
    delay(CHASE_RATE);
  }
}

////////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();

  // Make it bright!
  CircuitPlayground.setBrightness(255);
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  flasher();    delay(250);
  spinner();    delay(250);
  cylon();      delay(250);
  bedazzler();  delay(250);
  rainbow();    delay(250);
  // TODO: add your animation here!
}
```

Issues

With the above code loaded and running on the Circuit Playground, try changing the animations by pressing either of the two buttons. Did it work? If not try pressing it again. Work that time?

So you might notice that the button response is a little sluggish. Sometimes it will change, sometimes it won't. This is the main issue with the simple approach we've taken for checking button presses. The responsiveness is coupled to the speed of the animation. You could make the problem even worse by slowing down an animation. Then the button is checked even less frequently and becomes even more sluggish.

Kind of annoying, isn't it? If you're motivated to try and make it better, read the guides linked above.

The All Of Them CircuitPython

Here's CircuitPython (https://adafru.it/A22) code for the "All Of Them" version. You can compare each individual animation to its equivalent in the Arduino code.

CircuitPython only works on the Circuit Playground Express.

```
# Circuit Playground Express Bike Light - The All Of Them
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
import time
import random
from adafruit_circuitplayground.express import cpx

# Change these to set speed (lower is faster)
FLASH_RATE      = 0.250
SPIN_RATE       = 0.100
CYLON_RATE      = 0.100
BEDAZZLE_RATE   = 0.100
CHASE_RATE      = 0.100

# Change these to be whatever color you want
# Use color picker to come up with hex values
FLASH_COLOR     = 0xFF0000
SPIN_COLOR      = 0xFF0000
CYLON_COLOR     = 0xFF0000

# Define 10 colors here.
# Must be 10 entries.
# Use 0x000000 if you want a blank space.
RAINBOW_COLORS = (
  0xFF0000,
  0xFF5500,
  0xFFFF00,
  0x00FF00,
  0x0000FF,
  0xFF00FF,
  0x000000,
  0x000000,
  0x000000,
  0x000000
)

def buttons_pressed():
    return cpx.button_a or cpx.button_b

def flasher():
    while not buttons_pressed():
        # Turn on all the pixels to FLASH_COLOR
        cpx.pixels.fill(FLASH_COLOR)

        # Leave them on for a little bit
        time.sleep(FLASH_RATE)

        # Turn off all the NeoPixels
        cpx.pixels.fill(0)

        # Leave them off for a little bit
        time.sleep(FLASH_RATE)

def spinner():
```

```
    # Can be any two pixels
    pixel1 = 0
    pixel2 = 5

    while not buttons_pressed():
        # Turn off all the NeoPixels
        cpx.pixels.fill(0)

        # Turn on two pixels to SPIN_COLOR
        cpx.pixels[pixel1] = SPIN_COLOR
        cpx.pixels[pixel2] = SPIN_COLOR

        # Increment pixels to move them around the board
        pixel1 = pixel1 + 1
        pixel2 = pixel2 + 1

        # Check pixel values
        pixel1 = pixel1 if pixel1 &lt; 10 else 0
        pixel2 = pixel2 if pixel2 &lt; 10 else 0

        # Wait a little bit so we don't spin too fast
        time.sleep(SPIN_RATE)

def cylon():
    pixel1 = 0
    pixel2 = 9

    while not buttons_pressed():
        # Scan in one direction
        for step in range(4):
            cpx.pixels.fill(0)

            cpx.pixels[pixel1] = CYLON_COLOR
            cpx.pixels[pixel2] = CYLON_COLOR

            pixel1 = pixel1 + 1
            pixel2 = pixel2 - 1

            time.sleep(CYLON_RATE)

        # Scan back the other direction
        for step in range(4):
            cpx.pixels.fill(0)

            cpx.pixels[pixel1] = CYLON_COLOR
            cpx.pixels[pixel2] = CYLON_COLOR

            pixel1 = pixel1 - 1
            pixel2 = pixel2 + 1

            time.sleep(CYLON_RATE)

def bedazzler():
    while not buttons_pressed():
        # Turn off all the NeoPixels
        cpx.pixels.fill(0)

        # Turn on a random pixel to a random color
        cpx.pixels[random.randrange(10)] = ( random.randrange(256),
                                             random.randrange(256),
                                             random.randrange(256) )

        # Leave it on for a little bit
        time.sleep(BEDAZZLE_RATE)

def rainbow():
    # Start at the beginning
    start_color = 0
```

```
    while not buttons_pressed():
        # Turn off all the NeoPixels
        cpx.pixels.fill(0)

        # Loop through and set pixels
        color = start_color
        for p in range(10):
            cpx.pixels[p] = RAINBOW_COLORS[color]
            color += 1
            color = color if color &lt; 10 else 0

        # Increment start index into color array
        start_color += 1

        # Check value and reset if necessary
        start_color = start_color if start_color &lt; 10 else 0

        # Wait a little bit so we don't spin too fast
        time.sleep(CHASE_RATE)

# Loop forever
while True:
    flasher()   ; time.sleep(0.25)
    spinner()   ; time.sleep(0.25)
    cylon()     ; time.sleep(0.25)
    bedazzler() ; time.sleep(0.25)
    rainbow()   ; time.sleep(0.25)
    #
    # TODO: add your animation here!
    #
```

Making the Bike Light

The Circuit Playground bike light is meant to be rear facing. It doesn't generate enough light to illuminate the path in front of you.

You can mount it any old way depending on what your bike seat looks like and what other items you might have. The seat and/or seat post are good ideas. But could be a rear fender, storage rack, pannier, etc. Just be careful not get any loose wires caught up in the tire spokes.

Here's what I came up with using one of those small little under the seat packs.

The battery can go in the seat pack.

Then zip it close and let the on/off switch hang below.

Use zip ties to attach the Circuit Playground to the seat pack.

Cut zip ties flush.

Good to go.

Questions and Code Challenges

The following are some questions related to this project along with some suggested code challenges. The idea is to provoke thought, test your understanding, and get you coding!

While the sketches provided in this guide work, there is room for improvement and additional features. Have fun playing with the provided code to see what you can do with it. Or do something new and exciting!

Questions

- Is there any green in the color defined by 0xF0000D?
- What color is defined by 0xFFBAD2?
- How do you turn on more than one NeoPixel?
- What is the hexadecimal value 0x04 in decimal?

Code Challenges

- Use the slide switch to create a way to turn off/on all animations (power saver).
- Use the light meter to automatically turn on animations when it gets dark.
- Use the accelerometer as an input to an animation. (ex: spin/blink faster with acceleration).
- Make your own animation!