# Table of Contents

**Overview**  
- Materials Needed  

**Planning**  
- The Corset  
- Planning your Layout  

**Wiring Diagram**  

**Code with MakeCode**  

**On Start & Functions**  

**More Functions**  

**Mode Switching**  

**NeoPixel Strips**  

**Final Assembly**  

©Adafruit Industries
Overview

Create your own bespoke glowing NeoPixel corset and smile to yourself as all the heads in the room turn to follow you. Line your perfect silhouette with 500 programmable lights that show you off like a fairy disco queen. What could be more alluring?

This project requires careful crafting, some tricky soldering, a little sewing, and a bit of patience. It's worth it! Focused attention to the construction of your corset will lead to a masterpiece of beauty like nothing you've seen before.

---

Materials Needed

- A white or light-colored corset (see the "Planning" section for tips)

Adafruit Stuff

- Circuit Playground Express
- Soldering iron & accessories
- Around 5-6 meters of side-light NeoPixels (see "Planning" section for more info)
- Silicone stranded wire in 3 colors
- A battery & charger
- An on/off switch
- USB cable for programming

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 x Circuit Playground Express</strong></td>
<td>Circuit Playground Express Microcontroller</td>
<td><a href="https://www.adafruit.com/product/3333">https://www.adafruit.com/product/3333</a></td>
</tr>
<tr>
<td><strong>3 x Side-Light Neopixels - 120/m</strong></td>
<td>Adafruit NeoPixel LED Side Light Strip - Black 120 LED</td>
<td><a href="https://www.adafruit.com/product/3634">https://www.adafruit.com/product/3634</a></td>
</tr>
<tr>
<td><strong>3 x Side-Light Neopixels - 60/m</strong></td>
<td>Adafruit NeoPixel LED Side Light Strip - Black 90 LED</td>
<td><a href="https://www.adafruit.com/product/3635">https://www.adafruit.com/product/3635</a></td>
</tr>
<tr>
<td><strong>1 x Battery</strong></td>
<td>Lithium Ion Polymer Battery - 3.7v 2500mAh</td>
<td><a href="https://www.adafruit.com/product/328">https://www.adafruit.com/product/328</a></td>
</tr>
<tr>
<td><strong>1 x On/Off Switch</strong></td>
<td>JST Connector with On/Off Switch</td>
<td><a href="https://www.adafruit.com/product/3064">https://www.adafruit.com/product/3064</a></td>
</tr>
<tr>
<td><strong>1 x Battery Charger</strong></td>
<td>Adafruit Micro Lipo - USB LiIon/LiPoly charger - v1</td>
<td><a href="https://www.adafruit.com/product/1304">https://www.adafruit.com/product/1304</a></td>
</tr>
<tr>
<td><strong>3 x Ribbon Cable</strong></td>
<td>Silicone stranded wire - 4 wire cable, 26awg</td>
<td><a href="https://www.adafruit.com/product/3889">https://www.adafruit.com/product/3889</a></td>
</tr>
</tbody>
</table>

**Fabric Store Stuff**

- About 6 yards of 3/8" wide ribbon
- 2-3 yards of extra wide, double-fold bias tape
- A flower or other decoration that's larger than the Circuit Playground (2" in diameter)
- Scrap of fabric for a battery pocket
- Needle & thread & scissors
- Sewing machine
- Jewel or marble for the center of the flower

**Hardware Store Stuff**

- Silicone glue (I used Loctite Clear Silicone Adhesive)
- 1/2" clear heat shrink
- Hot glue gun
- Heat gun

©Adafruit Industries
We're using the Circuit Playground Express, because programming it can be as easy as downloading a file and pressing reset, or if you're a monster coder you can use Arduino or CircuitPython with the Circuit Playground's onboard sensors to create something totally unique.

Planning

The Corset

First and foremost, you'll need a corset. This method of attaching LEDs is designed to work with a white or light-colored corset, one that will reflect the lights and appear to glow on its own. A shiny satin fabric in white or silver will work best.

There are numerous online tutorials on making your own corset, or you can find inexpensive to high-end bespoke corsets online. I've tried quite a few different vendors. Here are my favorites:
Inexpensive Costume Corsets

I've found these everywhere from Halloween stores to Etsy to Amazon. The fit is never perfect, the structure is never ideal, but if you're not trying to alter your shape drastically you can get a nice looking corset for around $35.

Mid-Price Corsets

[Corset Story](https://corsetstory.com/) is my favorite vendor for pre-made corsets. Their corsets are in and around the $100 range, and the construction is solid. I used their [Classic Long White Waist Taming Corset](https://corsetstory.com/collections/corsets/products/classic-long-white-waist-taming-corset) in this tutorial. I've also ordered from them for several other costumes (including my [animatronic Medusa costume](https://corsetstory.com/collections/costumes/products/animatronic-medusa-costume)) and been very happy with the quality and fit every time.

High-End and Bespoke Corsets

My preferred vendor for high-end corsets is [Dark Garden Corsetry](https://darkgarden.corsetry.com/). They're based in San Francisco and New Orleans, and specialize in custom work. For the most comfortable fit imaginable in a corset, combined with superior construction and unbelievable shaping, dig into your savings account and get in line and these lovely ladies will create something unique for you that will last the rest of your life.
Planning your Layout

I placed one strip of pixels along each boning strut on my corset. My corset has 11 struts total (12 if you count both sides of the front closure). This worked out to just short of 5 meters of pixels total.

I could have possibly gotten away with 4 meters, except that these pixels are difficult to re-connect once they've been cut because of the tiny size of the solder pads. I recommend getting enough pixels that you have extra and don't need to "frankenstein" any of the strips together in the middle. You can always use the leftover bits for accessories - maybe a glowing pocket square or a top hat for your date.

I used black ribbon to cover the NeoPixel strips because I love the black-and-white striped look, but you could use white ribbon for a magical simplicity, or red ribbon for a Mary Poppins look.

Selecting your NeoPixels

Adafruit's Side-light NeoPixels are really great. Instead of pixels on the face of the strip, the pixels are on one side, which gives you instant indirect lighting. The strip still
flexes along its length, so these are really perfect for attaching to the outside of a costume. They'll flex along your waistline and will shine out onto the fabric instead of into the eyes of whoever's looking at you. You don't need to muck about with redirecting the lights or diffusion. They basically diffuse themselves.

You can affix the strips either face up or face down, and the visual effect is the same. The handy thing about this fact is that we can wire them tidily end-to-end and affix them to the corset in a serpentine pattern, and still control which way the lights shine by alternating which way we place the strip - face up or face down.

This is a bit hard to describe, so here's a graphic. If it still doesn't make sense, it will when you start laying out the corset.

The strips come in 3 flavors: 120 lights per meter, 90/m, or 60/m. There are advantages and disadvantages to each type.

The 120/m ones look simply stunning. The animation is smooth and the light effect is really robust and complete. They're jaw-dropping. But they do take more battery life and more processing power, since you'll use a lot more pixels at 120/m. They also cost a bit more.

The 60/m lights are kind of sparse. The battery will last twice as long, but the lights are far enough apart that I definitely notice a difference -- it looks a little bit spotty. Fine if that's what you're going for, but I found these too sparse.
The 90/m lights are a good compromise between total number of LEDs and LED spacing. They don't look all that different from the 120/m lights, at the end of the day -- the diffusion makes the difference barely noticeable. However, because of the way the strips are laid out (they're narrower with smaller soldering pads) these are devilishly tricky to solder. If you're a master solderer, that may not be a concern, but .. I've been doing this LED costume thing for a while and still had a really tough time.

Oddly enough, the 120/m ones were a snap to solder compared to the 90s (unlike most high-density strips).

A Good Compromise

I ended up using half 120/m pixels and half 90/m pixels. I put the 120s in the front of the corset, then switched to 90s for the sides and back. This kept my total number of pixels down to around 650 pixels, which is easily handled by the Circuit Playground Express. With the pixels at a decent brightness, my battery lasts over 4 hours. And I only had to solder half as many devilish 90/m strips.
The number of strips you have will be determined by how many boning pockets your corset has. Mine has 11 strips total - one for each of the 12 spines except the inside front closure spine.

You'll solder the Circuit Playground G to the NeoPixel G, VBAT to +, and A1 to DI. Be sure you're soldering to the correct end of the strips -- if you solder to D0 they won't work. Power can flow either way, but data must flow in the direction of the arrows.

We'll add the power switch and battery halfway through the series of strips, so it will end up at the back of the corset and be better able to power all the strips.

Code with MakeCode

We'll use Microsoft's drag-and-drop code editor to create the animations. MakeCode is an easy way to get up and running with the Circuit Playground Express. No prior coding knowledge is needed, and it's an easy way to experiment and learn to think like a coder.

For this project we'll be using maker.makecode.com (), which is the beta version of MakeCode. At the time of writing, it has a few more features that will give us more
control over our pixels, but it does change frequently so if something's not where you expect, look around.

Once your code is written, you'll simply plug your Circuit Playground Express into your computer via its USB port and click the "reset" button. All the lights will turn green and your Circuit Playground will appear as a drive on your computer called CPLAYBOOT. Drag your downloaded code onto this drive to program the Circuit Playground Express. Easy!

If you get a flash drive names CIRCUITPY, no worries, press the reset button twice and CPLAYBOOT should show up. It might take a time or two but it does work.

Head over to this Intro to MakeCode guide for more info on getting started with MakeCode.

If you want to skip right to the end and work backwards, here's the completed MakeCode project.

LED Corset Code

Code Design & User Interface

I love flexibility and variety, so I want to write some code that allows me to easily choose between different LED animation modes and lots of different color options. I'm a huge fan of being able to match my necklace to my hat on the fly. My corset should be just as robust.

With this in mind, here's what I want from my code:

- Matching animations for both the NeoPixel strips on the corset and the pixels on the face of the Circuit Playground, with separate brightness controls
- Around 5-6 different animation modes
- An easy way to choose any color I want
- Easy switching between modes using the Circuit Playground's onboard buttons

I decided to leave out a few other ideas, like being able to change brightness on-the-fly, or adding motion or sound reactivity, but those types of modes would be pretty easy to add in with MakeCode. Go nuts!
Here's how my finished code works. I've got six different LED modes:

1. Twinkle - white sparkles that appear and disappear randomly
2. RainbowFade - all LEDs are the same color and slowly fade through the spectrum
3. OneColor - LEDs are all one solid color
4. Gradient - Animation based on and around one color
5. RainbowSwirls - Animation based on the whole spectrum
6. Off

Each time button A is pressed, the modes cycle in order. Each time button B is pressed, the modes cycle in reverse order.

Modes 2, 3, and 4 all use the `hue` variable to select a base color. I click to RainbowFade and wait for my preferred color to appear, then hit button A once more to switch modes to OneColor. The LEDs will "freeze" on that color, giving me the ability to color the corset in any hue on the spectrum just by timing my button press. If I miss, I can click button B to go back.

One more button A press switches me to Gradient, which will start an animation using the same color I just selected, all thanks to the `hue` variable. One more press switches me to RainbowSwirls, which seamlessly adds in the rest of the colors on the spectrum. Pretty cool!

Using Functions

Six different modes? This code is gonna get complicated!

A good way to stay organized in MakeCode (and coding in general) is to use functions. A function is basically a container that holds a few lines of code, so we can give it a name and then refer it to easily later on. We'll put each of our LED modes inside its own function, so when we're ready to play each mode back, all we'll need to do is call its name.

We'll start by getting our variables and LED strips set up, then create our modes one by one and move them into their own functions. Then, we'll build a mode-switching machine and assign our buttons to switch modes. Ready to start?

Head to maker.makecode.com, choose New Project and select the Circuit Playground Express. You'll find yourself in the MakeCode Editor. From here, you can click on any of the colored tabs and drag blocks of code onto your workspace, then preview it using the emulator pictured on the left.
On Start & Functions

Click on the LOOPS tab and drag an instance of on start onto the workspace. (It may already be there) Anything inside this bracket will happen just once, when the Circuit Playground is powered up. You'll also see a FOREVER loop. Anything in this bracket will happen over and over, forever.

First we'll create our variables. Click the VARIABLES tab and add a variable called strip, one called strip2, one called hue and one called LedMode.
strip will refer to Circuit Playground's onboard NeoPixels. strip2 will refer to the side-light NeoPixel strip soldered to the Circuit Playground. hue is the variable we'll use to manipulate the color, and ledMode will hold the number of the mode we're in.

Click on the LIGHT tab and then the MORE tab that appears underneath it. Drag two instances of set strip to create WS2812 on LED with 30 pixels into your on start loop (or drag one instance and copy/paste to get two). Replace the blue bar of the first one with onboard strip. Set up the second instance to read set strip2 to create WS2812 strip on A1 with 650 pixels.

Now let's set the brightness. Drag or paste two instances of set brightness into your on start loop. Set strip's brightness to something really low - I chose 15. These pixels are shining right out from the front of the corset and we don't want to blind people. Set strip2's brightness to around 100. You can always change this later to make it brighter or dimmer.

As you've noticed, we're going to have to do a lot of copy/paste to get both the onboard pixels and the NeoPixel strip working in tandem. No worries! Just note that it's easy to miss one of these dropdowns, so if your code isn't working the way you expect later on you might have forgotten to refer to both strip and strip2.

Head to the VARIABLES tab and drag an instance of set ledMode to 0 into your on start loop. This tells the code which mode you want to start with. Later on you can change it to whichever mode number you want, and your corset will boot up in that mode.
That's it for the **on start** block. Let's make our first animation mode! We'll start with RainbowFade.

We'll build each mode inside the FOREVER loop so we can proof it on our emulator or download it to see how it looks on the NeoPixel strips. Once we're happy, we'll move the code into a **function** block so we can refer to it later.

Open the LIGHTS > MORE tab again and drag two instances of **set strip all pixels to red** into your **forever** loop. Change one to **strip2**.

Click the LIGHT tab and find **hue 255 sat 255 val 255**. Drag two instances to replace the **red** in the blocks you just made.
This little bit of awesomeness will give us lots of control over the colors. Hue refers to what we usually think of as a color: red, orange, green, etc. Red is at 0, yellow is around 50, blue 160 and it cycles back around to red at 255.

Saturation (sat) controls how intense the color is. For example, given a hue of 0 (red), saturation=0 comes out plain white, 100 is sort of pinkish, and 255 is full, bright red.

Value (val) refers to the amount of color. Think of it as intensity or brightness. We'll leave both value and saturation at 255 in this code, but you can play with these values to get pastel colors or dimmer colors if you want.

There's lots of info about HSV colors available on Wikipedia if you want to know more.

We're only concerned with the hue for this code. Remember how we made a hue variable earlier? Time to dust that puppy off and put her to use.

Click the VARIABLES tab and drag two instances of the hue variable onto your workspace, replacing the 255 next to the word "hue" in both these blocks. Then drag an instance of change hue by 1 into the forever loop.

Take a look at the emulator. It should show all pixels fading through the spectrum. Success! We made a RainbowFade animation.
If you want, you can change the speed of the animation using LOOPS > pause 100ms (change the 100 to whatever you'd like to change the speed). I didn't bother with this since I like the speed it's got.

Once you're totally happy with your animation, let's move the code into a function. Click the FUNCTIONS tab and make a function called RainbowFade.

Move all the code in your FOREVER loop into the RainbowFade function. Now we can activate it just by calling RainbowFade. We'll do that a bit later.
More Functions

Let's add more animation modes. We'll build each mode in the `forever` loop, so we can tweak the values and make it to our liking while watching the emulator, then once we're happy, move it into its own function for safe keeping. It's also helpful to download the code onto your Circuit Playground whenever you want to see how something looks, and also test with the NeoPixel strip attached. The animations look pretty different in real life -- and of course you can always come back and change things when your corset is finished.

OneColor Mode

The `OneColor` mode will be almost identical to the `RainbowFade` mode. The only difference is that we won't change the hue. Copy the two `set strip` blocks from the `RainbowFade` function and paste them into the `forever` loop. Check the emulator. All the lights are red, because we haven't defined `hue` so it's defaulting to 0 which translates to red.
This was an easy one. Make a function called `OneColor` and move all the code in the `forever` loop over.

Gradient Mode

Next we'll create the Gradient animation. We'll take the value assigned to `hue` and add a small random number to it, assign this new hue to a pixel, and then rotate the strip. Each pixel will carry its random color like an ant marching down the strip, surrounded by colors that are all slightly, randomly different.

From LIGHT > MORE drag an instance of `set strip pixel color at 0 to red`. Replace the 0 with the `hue` variable, and the `red` with the `hue / sat / val` block (from the LIGHT tab).
Notice that this is different from the blocks we used in the previous functions. Those blocks affected all the lights on the strip at once, and this block affects just one pixel at a time.

We're going to add a whole lot of randomness here. We'll change the hue, saturation, and value all by a random amount to get a really nice variety of colors and shades.

Drag out 4 instances of MATH > **pick random 0 to 10**. Place one in each of the spaces for **hue**, **sat**, and **val**.

Next, grab a **0 - 0** block from the top of the MATH tab. Replace the **10** in the **pick random 0 to 10** you assigned to the **hue** space with this block.

Then place your fourth **pick random** block into the second slot in this latest block. Change the **pick random** values to match the picture.

Add the **hue** variable as shown. Add a **strip rotate by 1** block from the MORE tab. Add a random element to the speed of the animation by choosing **LOOPS > pause 100 ms** and placing **pick random 0 to 100** (from MATH) inside it.

Check the emulator. It's a little hard to see with just red, so for testing purposes, let's assign a temporary value to our **hue** variable. Add **set hue to 50** at the beginning of your **forever** loop.
This got complicated pretty fast! It's the result of a whole lot of trial-and-error, and of playing with the different blocks in different configurations until I found a combination I love. Download the code onto your Circuit Playground to see the animation in real life, then change it all around until it's yours and you love it.

Once you're happy, copy/paste your blocks to also apply them to \texttt{strip2} (your NeoPixel strip).

Create a function and move all the code except for \texttt{set hue to 50} into the function. That block was just for testing, so delete it for now. We don't want to lock \texttt{hue} into any particular number just yet.

\textbf{RainbowSwirls Mode}

\textbf{RainbowSwirls} mode is almost exactly the same as \textbf{Gradient} mode. I want to keep the randomness (though I'll tweak the numbers a bit) and the format and speed. The main difference is that we're going to add \texttt{change hue by 1} to the end so the hue moves through all the colors in the spectrum, instead of being based on just one color.

Copy all the code from the \textbf{Gradient} function back into your FOREVER loop. Change the randomness values of the hue as shown, and add \texttt{change hue by 1} from the VARIABLES tab in at the bottom. Watch it on the emulator to see what happens.
Once you're happy with the rainbow animation, move the code into its own function called `RainbowSwirls`.

**Twinkle Mode**

I added one more mode, which gives a twinkly white glittery effect. The easy way to do this is to use one of MakeCode's preprogrammed animations. Look under LIGHT > MORE and drag an instance of strip show animation for 500 ms into your `forever` loop. Add a `pause` from the LOOPS tab to control the speed.

This is easy, and there are five other canned animations you can use as modes as well. However... I find this particular canned animation to be a bit frenetic for this
project. It also can't be interrupted for the 500 ms that it's running, which makes the button presses not work very well - you tend to get "stuck" in the canned animations. I spent a bunch of time playing around and came up with my own twinkle animation.

This code makes random pixels blink on in a variety of muted colors and whites. I achieved this by choosing random hues with low saturation (remember, saturation 0 is colorless), and then it makes other random pixels blink off. I found that for the long NeoPixel strip2, this created a satisfying amount of lights, but for the Circuit Playground's 10 lights on strip, I needed to double the "off" commands or the whole thing just sort of stayed lit up.

The one new thing here is the strip length block, which is found in the MORE tab near the bottom. This is a handy little block of code that will automatically remember how many lights you said the strip has and apply that number -- like an automatic variable.

This code picks a random light from 0 to strip length, and sets a random pale hue, saturation and value to it. The next block picks a different random light and turns the pixel off. We end up with pale pastel lights twinkling merrily all over the corset.

That's all our functions! Next we'll build our mode-switching engine and tell the buttons how to navigate through all the modes.

**Mode Switching**

We set up a variable at the beginning called ledMode. We'll use this variable to let the Circuit Playground know which mode we want, then we'll set up the two onboard buttons to cycle through the modes.

From the LOGIC tab, drag an if / then / else block into your forever loop. Drag a 0=0 comparison block in to replace the true. Then replace the first 0 with the ledMode variable.
Now head over to the ADVANCED > FUNCTIONS tab and find your `twinkle` function. Drag an instance of `call function twinkle` inside this conditional block.

Click the + button at the bottom of the conditional block to add more spaces. We have 5 modes, plus we want to add one mode where all the lights are off, for 6 modes total. Create four more spaces, copy/paste the `ledMode = 0` block into each, and fill each with a function as shown. Be sure to change the `0` to count all the different modes.
You can change the order if you'd like, but the order shown is designed to allow you to get a solid or gradient animation in any color, since it comes after the RainbowFade animation.

![Image of code diagram]

Now let's add a final mode that turns all the lights off. From the LIGHT > MORE tab drag an instance of `set strip all pixels to red` into the `else` block. (You may need to mess with the + and - buttons to get it to read `else` and not `else if`). Change the color to black. Drag a `strip stop all animations` block and a `strip clear` block in there as well, and apply them to both `strip` and `strip2`. We want to make extra-sure that all the lights go off when we're in mode 6.

![Image of code diagram]

**Buttons**

Now we'll set up the A and B buttons on the face of the Circuit Playground to change modes. We'll go forward through the modes with A, and backward through the modes with B.

There are all kinds of ways to trigger mode changes! If you'd like, you can use the onboard capacitive touch pads, or one of the other sensor triggers like tilt or shake to change modes, or even change modes if a loud sound is heard. Get creative! I like
using the onboard buttons with this design because they're easy to find with my fingers without looking, and I can change modes without thinking about it too much.

From the INPUT tab, drag an instance of **on button A click** into your workspace.

From VARIABLES, drag **change item by 1** and set it to **ledMode**.

This will make button A count up through all the modes. We need to add some code so that when it passes mode 4, it will reset to 0 so it cycles through over and over. Grab another **if/then** loop from the LOGIC tab and set it up as shown:
Since our modes are numbered 0-4, then as soon as the ledMode counts to 5 (or anything greater than 4) it will reset back to 0.

Copy/paste this entire block to set up button B. Change the values around as shown to make it count down instead of up.

That's it! We're done. Download the code to your Circuit Playground and click the buttons to make sure it works the way you expect.
NeoPixel Strips

It's time to solder the strips. This will take a bit of patience and care so give yourself some time, put on your favorite podcast, and don't rush it. You'll want to test each LED strip as you solder and be sure your solder points are rock-solid since one bad pixel or bad joint can make all the LEDs downstream simply stop working.

The 120/m strips are quite a bit easier to solder than the 90/m strips because of the way the pads are laid out. With either kind of strip, we'll use the "sacrificial pixel" method of cutting since the pads are so tiny. This means that instead of cutting through the center of the pad, and trying to solder wires to a half-pad on both ends, we'll cut one pixel out entirely leaving full, intact solder pads on each side.
Start by connecting the in end of your first strip to the Circuit Playground. Cut off the connector and the additional ground wire. Solder the red wire to VOUT, the white wire to A1 and the black wire to GND.
Decide whether your first strip will be facing up or down on the corset (determined by which way you want the lights to face). Use some hot glue to secure the wires and the top of the strip to the back of the Circuit Playground. Pin the Circuit Playground to the center front of the corset. Measure to the bottom with the LED strip and cut, leaving an entire solder pad on the end of the strip.

The easiest way to determine the correct length for each strip is to pin it carefully to the corset through the silicone sleeve. Pay attention to both the data flow direction and to which way the LEDs are facing as you lay the strips out. We'll connect them together in a serpentine layout. This means the data IN is at the top for the first strip (connected to the Circuit Playground) and at the bottom for the second strip, to keep
the wires as simple as possible. You can mount the strip either face up or face down to control which way the lights shine.

Cut each strip to the perfect length, remembering to leave a full solder pad at the bottom AND at the top of each strip. You'll sacrifice a few pixels, but your fingers will thank you later.

For now, leave the cut strips pinned to your corset. This will keep them organized and make it easy to determine the right length of wire for the connection points.

We'll solder the right side strips together, then add the on/off switch and battery connector, then solder the left side, ending at the front. This will place the battery and switch right at the back of the corset where it's out of your way.

Putting the battery in the middle of the LED strips is a great idea since we have so very many pixels. Power can travel either direction through the strips (though data only flows one way). By placing the battery in the middle of the strip, none of the pixels end up really far away from the power source. If power has to travel too far, it tends to get weaker, which means your pixels could "brown-out". Nobody likes browned-out pixels.
Prep your ribbon cable. We only need 3 wires, so look closely and find the striped wire. This will be our POWER wire. Pull on the non-striped wire on the opposite side of the cable and strip it off, leaving a 3-wire strip. Save the extra black wire for another project.

Separate the three remaining wires about 1/2", and strip a tiny bit off the end of each. On the bottom end of your first LED strip, solder the striped wire to +, the middle wire to DO and the remaining wire to G.

Cut your ribbon cable so it's the right length to hop over to the next strip on your corset. The IN end of the pixels should be at the bottom on this strip, so that means you'll only need a few inches. Cut it to length and solder to the new strip the same way: striped wire to +, middle wire to DI and remaining wire to -.

Time to test the strip! Plug a battery into your Circuit Playground and be sure both strips light up.

If everything is working, it's time to secure the end of the strips. We'll use clear heat shrink and hot glue to make the joints rock-solid, so they never wiggle apart while you're wearing the corset.
Cut a small piece of 1/2" clear heat shrink and slip it over the end of the strip. Squirt a little hot glue inside, and use a heat gun to shrink the heat shrink while the glue is still wet. This will create a solid plastic housing for your solder joints that will be virtually unbreakable.

Continue with the rest of the strips going around the right side of the corset. When you get to the back, solder on a long (at least 12" or longer for a larger corset) piece of ribbon cable to the end of the last strip. It's time to add the battery cable and on/off switch.
About halfway down this long piece of ribbon cable, separate the three wires and clip the two outer ones, leaving the middle wire intact. Strip a little off the cut ends and twist each wire back together with itself -- striped wire to striped wire, non-striped to non-striped. This bit of silliness will give us a connection point for the switch.

Plug the battery into your JST extension switch and notice which side you used. Leave this side alone! This is the male side, and it's for plugging the battery in. Unplug the battery again.

Cut off the other end of the JST extension cable (the female side). Separate and strip the wires. Solder the red wire to the striped wire connection point and the black wire to the non-striped point, remembering to slip on some heat shrink before soldering.
Plug in your battery and click the on/off switch. If all goes well, your lights and Circuit Playground will come on. Hooray! Be sure everything works right, leave another few inches of ribbon cable, and continue with soldering, testing, and sealing the rest of your NeoPixel strips.

Add a rubber band around the battery, trapping the leads inside. The connections are really delicate and will break if you don't provide some strain relief.
Final Assembly

Once all your pixels are soldered and solid, it's time for some gluing and sewing. We'll glue some ribbons to the outside of each NeoPixel strip, to disguise whether they're facing up or down. Then we'll glue the pixels to the corset, and sew some double fold quilt bias tape to the top and bottom edge of the corset to hide all the wires and solder joints.

Silicone Glue

Hardly anything sticks to silicone. You'll need specialty silicone glue to make this work. Even with silicone glue, some types of ribbon don't want to stick! I found that satin ribbon worked fine, while velvet ribbon just wants to keep separating. Test your ribbon to be sure it's compatible with the glue you're using.

My favorite silicone glue is Devcon Silicone Adhesive. For this project I used Loctite Silicone Sealant, since I couldn't find the Devcon locally and I didn't want to wait for delivery. It worked pretty well, but I still prefer the Devcon -- I think it would have worked better on the velvet ribbon.
Lay out all your strips on the table, making sure all the lights are facing in the direction you want. Tape them to the table so they don't move. Glue a ribbon onto each strip, leaving a little extra at both ends to be sure all the LEDs are covered.

Once the ribbons are dry, place the strips onto your corset with the ribbon side out. Glue them along the boning lines. It helps to pin them in place through the silicone.

To finish the edges, we'll cover them with double fold bias tape. I got the widest I could find, labeled as quilt binding. Bias tape is made specifically for finishing edges just like this (in fact, your corset probably already has some bias tape sewn along the edges). It's cut on the bias of the fabric (diagonally), meaning it will stretch and go around curves really nicely, and the double-fold means you don't have to hem and press and do a bunch of fiddly sewing work. It's all done for you. Yay, technology!
Open your double fold bias tape and place it along the bottom edge of your corset. Fold the corner in at the center front. Tuck all the wires inside so they're fully hidden by the tape. Stitch carefully between the LED strips.

When you get to the back opening, cut the tape with an extra 1/2" or so and fold it inside before sewing. The wires here will need to stick out of the tape -- it's fine to sew right over them, it won't hurt them. Be sure there's enough wire slack so that your adjusted corset will fit without strain.
Continue on the other side of the bottom, and add bias tape to both sides of the top of the corset as well.

Most corsets have a privacy flap at the back to cover the inside of the laces. This is a perfect spot to put your battery. Hand-sew a pocket from a piece of scrap fabric through just the top layer of the privacy flap, making sure your battery cable reaches.

Once the bias tape is finished, sew or glue your Circuit Playground to the center front of the corset through some of the unused pads.
To cover the Circuit Playground but still leave the buttons accessible, I used a silk carnation from the craft store. Cut off the stem and add some hot glue to the back to keep it from falling apart. To make it fancier, I colored in all the edges of the petals with a sharpie.
Give the flower a center. A clear marble or jewel works great here. I used a black leather laser-cut star left over from my LED spats project. Matchy matchy! Sew the bottom petals of the flower to the Circuit Playground, through the unused pads.

Finish by gluing or sewing the switch to the top of the back of the corset, someplace you can reach (but won’t switch the corset off accidentally when you lean back).

Accessorize with NeoPixel Spats and a Glowing Fascinator Hat, or a Color Touch Necklace. Congratulations! You’re about to outshine every Prince at the ball.