LEARN CSS COLORS BY BUILDING A SET OF COLORED MARKERS

Introduction

Selecting the correct colors for your webpage can greatly improve the aesthetic appeal to your readers.

In this course, you'll build a set of colored markers. You'll learn different ways to set color values and how to pair colors with each other.

Step 1:

As you've seen in the previous projects, webpages should start with a `DOCTYPE html` declaration, followed by an `html` element.

Add a `DOCTYPE html` declaration at the top of the document, and an `html` element after that. Give the `html` element a `lang` attribute with `en` as its value.

Step 2:

Nest a `head` element within the `html` element. Just after the `head` element, add a `body` element.

Step 3:

Remember that the `title` element gives search engines extra information about the page. It also displays the content of that `title` element in two more ways:

- in the title bar when the page is open

- in the browser tab for the page when you hover on it. Even if that tab is not active, once you hover on the tab, the title text is displayed.

Within the head element, nest a title element with the text Colored Markers.

Step 4:

The charset attribute specifies the character encoding used by the document. utf-8 (Unicode Transformation Format – 8-bit) is a character encoding standard used for electronic communication.

Inside the head element, nest a meta element with the attribute charset set to "utf-8".

Step 5:

You can have multiple meta elements on a web page. Each meta element adds information about the page that cannot be expressed by other HTML elements.

Add another meta element within the head. Give it a name attribute set to "viewport" and a content attribute set to "width=device-width, initial-scale=1.0" so your page looks the same on all devices.

Step 6:

Now you're ready to start adding content to the page.

Within the body, nest an h1 element with the text CSS Color Markers.

Step 7:

In this project you'll work with an external CSS file to style the page. We've already created a `styles.css` file for you. But before you can use it, you'll need to link it to the page.

Nest a `link` element within the `head` element. Give it a `rel` attribute set to `"stylesheet"` and an `href` attribute set to `"styles.css"`.

Step 8:

Now that your external CSS file is set up, you can start styling the page.

As a reminder, here's how to target a paragraph element and align it to the right:

Example Code:

```
p {

  text-align: right;

}
```

Create a new CSS rule that targets the `h1` element, and set its `text-align` property to `center`.

Step 9:

Now you'll add some elements that you'll eventually style into color markers.

First, within the `body` element, add a `div` element and set its `class` attribute to `container`. Make sure the `div` element is below the `h1` element.

Step 10:

Next, within the `div` element, add another `div` element and give it a class of `marker`.

Step 11:

It's time to add some color to the marker. Remember that one way to add color to an element is to use a color keyword like `black`, `cyan`, or `yellow`.

As a reminder, here's how to target the class `freecodecamp`:

Example Code:

```
.freecodecamp {

}
```

Create a new CSS rule that targets the class `marker`, and set its `background-color` property to `red`.

Note: You will not see any changes after adding the CSS.

Step 12:

The background color was applied, but since the marker `div` element has no content in it, it doesn't have any height by default.

In your `.marker` CSS rule, set the `height` property to `25px` and the `width` property to `200px`

Step 13:

Your marker would look better if it was centered on the page. An easy way to do that is with the `margin` shorthand property.

In the last project, you set the margin area of elements separately with properties like `margin-top` and `margin-left`. The `margin` shorthand property makes it easy to set multiple margin areas at the same time.

To center your marker on the page, set its `margin` property to `auto`. This sets `margin-top`, `margin-right`, `margin-bottom`, and `margin-left` all to `auto`.

Step 14:

Now that you've got one marker centered with color, it's time to add the other markers.

In the `container div`, add two more `div` elements and give them each a class of `marker`.

Step 15:

While you have three separate marker `div` elements, they look like one big rectangle. You should add some space between them to make it easier to see each element.

When the shorthand `margin` property has two values, it sets `margin-top` and `margin-bottom` to the first value, and `margin-left` and `margin-right` to the second value.

In your `.marker` CSS rule, set the `margin` property to `10px auto`.

Step 16:

To give the markers different colors, you will need to add a unique class to each one. Multiple classes can be added to an element by listing them in the `class` attribute and separating them with a space. For example, the following adds both the `animal` and `dog` classes to a `div` element.

Example Code:

```
<div class="animal dog">
```

If you add multiple classes to an HTML element, the styles of the first classes you list may be overridden by later classes.

To begin, add the class `one` to the first marker `div` element.

Step 17:

Next, remove the `background-color` property and its value from the `.marker` CSS rule.

Step 18:

Then, create a new CSS rule that targets the class `one` and set its `background-color` property to `red`.

Step 19:

Add the class `two` to the second marker `div`, and add the class `three` to the third marker `div`.

Step 20:

Create a CSS rule that targets the class `two` and set its `background-color` property to `green`.

Also, create a separate CSS rule that targets the class `three` and set its `background-color` to `blue`.

Step 21:

There are two main color models: **the additive RGB (red, green, blue)** model used in electronic devices, and **the subtractive CMYK (cyan, magenta, yellow, black)** model used in print.

In this project, you'll work with the RGB model. This means that colors begin as black, and change as different levels of red, green, and blue are introduced. An easy way to see this is with the CSS rgb function.

Create a new CSS rule that targets the class container and set its background-color to black with rgb(0, 0, 0).

Step 22:

A **function** is a piece of code that can take an input and perform a specific action. The CSS rgb function accepts values, or arguments, for red, green, and blue, and produces a color:

Example Code:

```
rgb(red, green, blue);
```

Each red, green, and blue value is a number from 0 to 255. 0 means that there's 0% of that color, and is black. 255 means that there's 100% of that color.

In the .one CSS rule, replace the color keyword red with the rgb function. For the rgb function, set the value for red to 255, the value for green to 0, and the value for blue to 0.

Step 23:

Notice that the background-color for your marker is still red. This is because you set the red value of the rgb function to the max of 255, or 100% red, and set both the green and blue values to 0.

Now use the rgb function to set the other colors.

In the .two CSS rule, use the rgb function to set the background-color to the max value for green, and 0 for the other values. And in the .three CSS rule, use the rgb function to set the background-color to the max value for blue, and 0 for the other values.

Step 24:

While the red and blue markers look the same, the green one is much lighter than it was before. This is because the green color keyword is actually a darker shade, and is about halfway between black and the maximum value for green.

In the .two CSS rule, set the green value in the rgb function to 127 to lower its intensity.

Step 25:

Next, you want to center the #menu horizontally. You can do this by setting its margin-left and margin-right properties to auto. Think of the margin as invisible space around an element. Using these two margin properties, center the #menu element within the body element.

Step 26:

In the additive RGB color model, **primary colors** are colors that, when combined, create pure white. But for this to happen, each color needs to be at its highest intensity.

Before you combine colors, set your green marker back to pure green. For the rgb function in the .two CSS rule, set green back to the max value of 255.

Step 27:

Now that you have the primary RGB colors, it's time to combine them.

For the rgb function in the .container rule, set the red, green, and blue values to the max of 255.

Step 28:

Secondary colors are the colors you get when you combine primary colors. You might have noticed some secondary colors in the last step as you changed the red, green, and blue values.

To create the first secondary color, yellow, update the rgb function in the .one CSS rule to combine pure red and pure green.

Step 29:

To create the next secondary color, cyan, update the rgb function in the .two CSS rule to combine pure green and pure blue.

Step 30:

To create the final secondary color, magenta, update the rgb function in the .three CSS rule to combine pure blue and pure red.

Step 31:

Now that you're familiar with secondary colors, you'll learn how to create tertiary colors. **Tertiary colors** are created by combining a primary with a nearby secondary color.

To create the tertiary color orange, update the rgb function in the .one CSS rule so that red is at the max value, and set green to 127.

Step 32:

Notice that, to create orange, you had to increase the intensity of red and decrease the intensity of the green rgb values. This is because orange is the combination of red and yellow.

To create the tertiary color spring green, combine cyan with green. Update the rgb function in the .two CSS rule so that green is at the max value, and set blue to 127.

Step 33:

And to create the tertiary color violet, combine magenta with blue. Update the rgb function in the .three CSS rule so that blue is at the max value, and set red to 127.

Step 34:

There are three more tertiary colors: chartreuse green (green + yellow), azure (blue + cyan), and rose (red + magenta).

To create chartreuse green, update the rgb function in the .one CSS rule so that red is at 127, and set green to the max value.

For azure, update the rgb function in the .two CSS rule so that green is at 127 and blue is at the max value.

And for rose, which is sometimes called bright pink, update the rgb function in the .three CSS rule so that blue is at 127 and red is at the max value.

Step 35:

Now that you've gone through all the primary, secondary, and tertiary colors on a color wheel, it'll be easier to understand other color theory concepts and how they impact design.

First, in the CSS rules .one, .two, and .three, adjust the values in the rgb function so that the background-color of each element is set to pure black. Remember that the rgb function uses the additive color model, where colors start as black and change as the values of red, green, and blue increase.

Step 36:

A **color wheel** is a circle where similar colors, or hues, are near each other, and different ones are further apart. For example, pure red is between the hues of rose and orange.

Two colors that are opposite from each other on the color wheel are called **complementary colors**. If two complementary colors are combined, they produce gray. But when they are placed side-by-side, these colors produce strong visual contrast and appear brighter.

In the rgb function for the .one CSS rule, set the red value to the max of 255 to produce pure red. In the rgb function for .two CSS rule, set the values for green and blue to the max of 255 to produce cyan.

Step 37:

Notice that the red and cyan colors are very bright right next to each other. This contrast can be distracting if it's overused on a website, and can make text hard to read if it's placed on a complementary-colored background.

It's better practice to choose one color as the **dominant color**, and use its **complementary color** as an **accent** to bring attention to certain content on the page.

First, in the h1 rule, use the rgb function to set its background-color to cyan.

Step 38:

Next, in the .one CSS rule, use the rgb function to set the background-color to black. And in the .two CSS rule, use the rgb function to set the background-color to red.

Step 39:

Notice how your eyes are naturally drawn to the red color in the center? When designing a site, you can use this effect to draw attention to important headings, buttons, or links.

There are several other important color combinations outside of complementary colors, but you'll learn those a bit later.

For now, use the rgb function in the .two CSS rule to set the background-color to black.

Step 40:

And in the h1 CSS rule, remove the background-color property and value to go back to the default white color.

Step 41:

Now it's time to add other details to the markers, starting with the first one.

In the first marker div element, change the class one to red.

Step 42:

Update the .one CSS rule to target the new red class.

Step 43:

And update the rgb function in the .red CSS rule so that the red value is at the max.

Step 44:

Next, change the class two to green in the second marker div, and the class three to blue in the third marker div.

Step 45:

Update the CSS class selector .two so it targets the new green class. And update the .three class selector so it targets the new blue class.

Step 46:

A very common way to apply color to an element with CSS is with **hexadecimal** or **hex values**. While hex values sound complicated, they're really just another form of RGB values.

Hex color values start with a # character and take six characters from 0-9 and A-F. The first pair of characters represent red, the second pair represent green, and the third pair represent blue. For example, #4B5320.

In the .green class selector, set the background-color property to a hex color code with the values 00 for red, FF for green, and 00 blue.

Step 47:

You may already be familiar with decimal, or base 10 values, which go from 0 - 9. Hexadecimal, or base 16 values, go from 0 - 9, then A - F:

Example Code:

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
```

With hex colors, `00` is 0% of that color, and `FF` is 100%. So `#00FF00` translates to 0% red, 100% green, and 0% blue, and is the same as `rgb(0, 255, 0)`.

Lower the intensity of green by setting the green value of the hex color to `7F`.

Step 48:

The **HSL color model**, or **hue, saturation**, and **lightness,** is another way to represent colors.

The CSS `hsl` function accepts 3 values: a number from 0 to 360 for hue, a percentage from 0 to 100 for saturation, and a percentage from 0 to 100 for lightness.

If you imagine a color wheel, the **hue** red is at 0 degrees, green is at 120 degrees, and blue is at 240 degrees.

Saturation is the intensity of a color from 0%, or gray, to 100% for pure color. You must add the percent sign `%` to the saturation and lightness values.

Lightness is how bright a color appears, from 0%, or complete black, to 100%, complete white, with 50% being neutral.

In the `.blue` CSS rule, use the `hsl` function to change the `background-color` property to pure blue. Set the hue to `240`, the saturation to `100%`, and the lightness to `50%`.

Step 49:

You've learned a few ways to set flat colors in CSS, but you can also use a **color transition**, or **gradient**, on an element.

A **gradient** is when one color transitions into another. The CSS `linear-gradient` function lets you control the direction of the transition along a line, and which colors are used.

One thing to remember is that the `linear-gradient` function actually creates an `image` element, and is usually paired with the `background` property which can accept an image as a value.

In the `.red` CSS rule, change the `background-color` property to `background`.

Step 50:

The `linear-gradient` function is very flexible -- here is the basic syntax you'll use in this tutorial:

Example Code:

```
linear-gradient(gradientDirection, color1, color2, ...);
```

`gradientDirection` is the direction of the line used for the transition. `color1` and `color2` are color arguments, which are the colors that will be used in the transition itself. These can be any type of color, including color keywords, hex, `rgb`, or `hsl`.

Now you'll apply a red-to-green gradient along a 90 degree line to the first marker.

First, in the `.red` CSS rule, set the `background` property to `linear-gradient()`, and pass it the value `90deg` as the `gradientDirection`.

Step 51:

You'll use the rgb function for the colors of this gradient.

In the linear-gradient function, use the rgb function to set the first color argument to pure red.

Step 52:

You won't see the gradient yet because the linear-gradient function needs at least two color arguments to work.

In the same linear-gradient function, use the rgb function to set the second color argument to pure green.

Step 53:

As you can see, the linear-gradient function produced a smooth red-green gradient. While the linear-gradient function needs a minimum of two color arguments to work, it can accept many color arguments.

Use the rgb function to add pure blue as the third color argument to the linear-gradient function.

Step 54:

Color-stops allow you to fine-tune where colors are placed along the gradient line. They are a length unit like px or percentages that follow a color in the linear-gradient function.

For example, in this red-black gradient, the transition from red to black takes place at the 90% point along the gradient line, so red takes up most of the available space:

Example Code:

```
linear-gradient(90deg, red 90%, black);
```

In the linear-gradient function, add a 75% color stop after the first red color argument. Do not add color stops to the other colors arguments.

Step 55:

Now that you know the basics of how the linear-gradient function and color-stops work, you can use them to make the markers look more realistic.

In the linear-gradient function, set gradientDirection to 180deg.

Step 56:

Next, set the color-stop for red to 0%, the color-stop for green to 50%, and the color-stop for blue to 100%.

Step 57:

Now that the color-stops are set, you'll apply different shades of red to each color argument in the linear-gradient function. The shades on the top and bottom edges of the marker will be darker, while the one in the middle will be lighter, as if there's a light above it.

For the first color argument, which is currently pure red, update the rgb function so the value for red is 122, the value for green is 74, and the value for blue is 14.

Step 58:

Now modify the second color argument in the linear-gradient function, which is currently pure green.

Update the rgb function so the value for red is 245, the value of green is 62, and the value of blue is 113.

Step 59:

Finally, modify the third color argument in the linear-gradient function, which is currently pure blue.

Update the rgb function so the value for red is 162, the value of green is 27, and the value of blue is 27.

Step 60:

The red marker looks much more realistic. Now you'll do the same for the green marker, using a combination of the linear-gradient function and hex colors.

In the .green CSS rule, change the background-color property to background.

Step 61:

For this marker, you'll use hex color codes for your gradient.

Use the linear-gradient function and set gradientDirection to 180deg. And for the first color argument, use a hex color code with the values 55 for red, 68 for green, and 0D for blue.

Step 62:

For the second color argument, use a hex color code with the values `71` for `red`, `F5` for `green`, and `3E` for `blue`.

Step 63:

That's looking better, but the bottom edge of the green marker needs to be darker to add a little more dimension.

In the same `linear-gradient` function, add a hex color code with the values `11` for `red`, `6C` for `green`, and `31` for `blue` as the third color argument.

Step 64:

Even without the color-stops, you might have noticed that the colors for the green marker transition at the same points as the red marker. The first color is at the start (0%), the second is in the middle (50%), and the last is at the end (100%) of the gradient line.

The `linear-gradient` function automatically calculates these values for you, and places colors evenly along the gradient line by default.

In the `.red` CSS rule, remove the three color stops from the `linear-gradient` function to clean up your code a bit.

Step 65:

If no `gradientDirection` argument is provided to the `linear-gradient` function, it arranges colors from top to bottom, or along a 180 degree line, by default.

Clean up your code a little more by removing the `gradientDirection` argument from both `linear-gradient` functions.

Step 66:

Now you'll apply a gradient to the blue marker, this time using the `hsl` function as color arguments.

In the `.blue` CSS rule, change the `background-color` property to `background`.

Step 67:

Use the `linear-gradient` function, and pass in the `hsl` function with the values `186` for hue, `76%` for saturation, and `16%` for lightness as the first color argument.

Step 68:

As the second color argument, pass in the `hsl` function with the values `223` for hue, `90%` for saturation, and `60%` for lightness.

Step 69:

And as the third color argument, pass in the `hsl` function with the values `240` for hue, `56%` for saturation, and `42%` for lightness.

Step 70:

Now that the markers have the correct colors, it's time to build the marker sleeves. Start with the red marker.

Inside the red marker `div` element, create a new `div` element and give it a class of `sleeve`.

Step 71:

Create a new CSS rule that targets the class `sleeve`. Set the `width` property to `110px`, and the `height` property to `25px`.

Step 72:

To make the marker look more realistic, give the sleeve a transparent white color.

First, set the sleeve element's `background-color` to `white`.

Step 73:

Opacity describes how opaque, or non-transparent, something is. For example, a solid wall is opaque, and no light can pass through. But a drinking glass is much more transparent, and you can see through the glass to the other side.

With the CSS `opacity` property, you can control how opaque or transparent an element is. With the value `0`, or 0%, the element will be completely transparent, and at `1.0`, or 100%, the element will be completely opaque like it is by default.

In the `.sleeve` CSS rule, set the `opacity` property to `0.5`.

Step 74:

Another way to set the opacity for an element is with the **alpha channel**. Similar to the `opacity` property, the **alpha channel** controls how transparent or opaque a color is.

You've already set the sleeve's opacity with a named color and the `opacity` property, but you can add an alpha channel to the other CSS color properties.

Inside the `.sleeve` rule, remove the `opacity` property and value.

Step 75:

You're already familiar with using the `rgb` function to set colors. To add an alpha channel to an `rgb` color, use the `rgba` function instead.

The `rgba` function works just like the `rgb` function, but takes one more number from `0` to `1.0` for the alpha channel:

Example Code:

```
rgba(redValue, greenValue, blueValue, alphaValue);
```

You can also use an alpha channel with `hsl` and `hex` colors. You will see how to do that soon.

In the `.sleeve` rule, use the `rgba` function to set the `background-color` property to pure white with 50% opacity.

Step 76:

Your sleeve is looking good, but it would look even better if it was positioned more toward the right side of the marker. One way to do that is to add another element before the sleeve to push it to the right.

Add a new `div` with the class `cap` before the sleeve `div` element.

Step 77:

Create a new CSS rule to target the class `cap`. In the new rule, set the `width` property to `60px`, and the `height` to `25px`.

Step 78:

It looks like your sleeve disappeared, but don't worry -- it's still there. What happened is that your new cap `div` is taking up the entire width of the marker, and is pushing the sleeve down to the next line.

This is because the default `display` property for `div` elements is `block`. So when two `block` elements are next to each other, they stack like actual blocks. For example, your marker elements are all stacked on top of each other.

To position two `div` elements on the same line, set their `display` properties to `inline-block`.

Create a new rule to target both the `cap` and `sleeve` classes, and set `display` to `inline-block`.

Step 79:

In the last project, you learned a little bit about borders and the `border-color` property.

All HTML elements have borders, though they're usually set to `none` by default. With CSS, you can control all aspects of an element's border, and set the border on all sides, or just one side at a time. For a border to be visible, you need to set its width and style.

In the `.sleeve` CSS rule, add the `border-left-width` property with the value `10px`.

Step 80:

Borders have several styles to choose from. You can make your border a solid line, but you can also use a dashed or dotted line if you prefer. Solid border lines are probably the most common.

In the `.sleeve` CSS rule, add the `border-left-style` property with the value `solid`.

Step 81:

Your border should be visible now. If no color is set, black is used by default.

But to make your code more readable, it's better to set the border color explicitly.

In the `.sleeve` CSS rule, add the `border-left-color` property with the value `black`.

Step 82:

The `border-left` shorthand property lets you set the left border's width, style, and color at the same time.

Here is the syntax:

Example Code:

```
border-left: width style color;
```

In the `.sleeve` CSS rule, replace the `border-left-width`, `border-left-style`, and `border-left-color` properties with the `border-left` shorthand property. The values for the width, style, and color of the left border should be the same.

Step 83:

Your marker is looking good. But to make it look even more realistic, you can change the border style to double solid borders.

For the `border-left` shorthand property, change the border style value from `solid` to `double`.

Step 84:

The black color of your border looks pretty harsh against the more transparent sleeve. You can use an alpha channel to lower the opacity of the black border.

For the `border-left` shorthand property, use the `rgba` function to set the color value to pure black with 75% opacity.

Step 85:

Awesome. Your red marker is looking good. Now all you need to do is add the caps and sleeves to your other markers.

Add a cap and sleeve to both the green and blue markers. You can just copy the `div` elements from the red marker and paste them into the other two markers.

Step 86:

The last thing you'll do is add a slight shadow to each marker to make them look even more realistic.

The `box-shadow` property lets you apply one or more shadows around an element. Here is basic syntax:

Example Code:

box-shadow: offsetX offsetY color;

Here's how the offsetX and offsetY values work:

- both offsetX and offsetY accept number values in px and other CSS units
- a positive offsetX value moves the shadow right and a negative value moves it left
- a positive offsetY value moves the shadow down and a negative value moves it up
- if you want a value of zero (0) for any or both offsetX and offsetY, you don't need to add a unit. Every browser understands that zero means no change.

The height and width of the shadow is determined by the height and width of the element it's applied to. You can also use an optional spreadRadius value to spread out the reach of the shadow. More on that later.

Start by adding a simple shadow to the red marker.

In the .red CSS rule, add the box-shadow property with the values 5px for offsetX, 5px for offsetY, and red for color.

Step 87:

As you can see, you added a simple red shadow around your marker that's 5 pixels to the right, and 5 pixels down.

But what if you wanted to position your shadow on the opposite side? You can do that by using negative values for offsetX and offsetY.

Update the values for the box-shadow property, and set offsetX to -5px, and offsetY to -5px.

Step 88:

Notice that the edges of the shadow are sharp. This is because there is an optional `blurRadius` value for the `box-shadow` property:

Example Code:

```
box-shadow: offsetX offsetY blurRadius color;
```

If a `blurRadius` value isn't included, it defaults to `0` and produces sharp edges. The higher the value of `blurRadius`, the greater the blurring effect is.

In the `.green` CSS rule, add the `box-shadow` property with the values `5px` for `offsetX`, `5px` for `offsetY`, `5px` for `blurRadius`, and `green` for `color`.

Step 89:

But what if you wanted to expand the shadow out further? You can do that with the optional `spreadRadius` value:

Example Code:

```
box-shadow: offsetX offsetY blurRadius spreadRadius color;
```

Like `blurRadius`, `spreadRadius` defaults to `0` if it isn't included.

Practice by adding a 5 pixel shadow directly around the blue marker.

In the `.blue` CSS rule, add the `box-shadow` property with the values `0` for `offsetX`, `0` for `offsetY`, `0` for `blurRadius`, `5px` for `spreadRadius`, and `blue` for `color`.

Step 90:

Now that you're familiar with the `box-shadow` property you can finalize the shadows, starting with the one for the red marker.

In the `.red` CSS rule, update the values for the `box-shadow` property so `offsetX` is `0`,`offsetY` is `0`, `blurRadius` is `20px`, `spreadRadius` is `0`, and `color` is `red`. Remember that you don't need to add units to a zero value.

Step 91:

Next, update the `color` value of the red marker's `box-shadow` property.

Replace the named color with the `rgba` function. Use the values `83` for red, `14` for green, `14` for blue and `0.8` for the alpha channel.

Step 92:

The shadows for your green and blue markers will have the same position, blur, and spread. The only difference will be the colors.

In the `.green` and `.blue` CSS rules, update the values for the `box-shadow` properties so `offsetX` is `0`,`offsetY` is `0`, `blurRadius` is `20px`, and `spreadRadius` is `0`. Leave the colors as `green` and `blue` for now.

Step 93:

For the green marker's `box-shadow` property, replace the named color with a hex color code. Use the values `3B` for red, `7E` for green, `20` for blue, and `CC` for the alpha channel.

Step 94:

Finally, for the blue marker's `box-shadow` property, replace the named color with the `hsla` function. Use the values `223` for hue, `59%` for saturation, `31%` for lightness, and `0.8` for the alpha channel.

And with that, your set of colored markers is complete! Well done.