Learn CSS Colors by Building a Set of Colored Markers

# Step 1 – DOCTYPE html declaration

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.

<!DOCTYPE html>

<html lang="en">

</html>

# Step 2 – head and body elements

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

<head>

</head>

<body>

</body>

# Step 3 – title element

Remember that the title element gives search engines extra information about the page. It also tells browsers what text to display in the title bar when the page is open, and on the tab for the page.

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

<head>

  <title>Colored Markers</title>

</head>

# Step 4 – meta element

To tell browsers how to encode characters on your page, set the charset to utf-8. utf-8 is a universal character set that includes almost every character from all human languages.

Inside the head element, nest a meta element with the attribute charset set to utf-8. Remember that meta elements are self-closing, and do not need a closing tag.

<head>

  <meta charset="utf-8">

  <title>Colored Markers</title>

</head>

# Step 5 – name attribute

Finally, use a viewport <meta> tag to make sure your page looks the same on all devices.

Nest a self-closing 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.

<head>

  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Colored Markers</title>

</head>

# Step 6 – h1 element

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

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

<body>

  <h1>CSS Color Markers</h1>

</body>

# Step 7 – style.css

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. Give it a rel attribute set to stylesheet and an href attribute set to styles.css.

<head>

  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Colored Markers</title>

  <link rel="stylesheet" href="styles.css">

  </link>

</head>

# Step 8 – h1 element text-align property

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:

p {

text-align: right;

}

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

h1{

  text-align: center;

}

# Step 9 – div element

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

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

<body>

  <h1>CSS Color Markers</h1>

  <div class="container">

  </div>

</body>

# Step 10 – div element

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

<body>

  <h1>CSS Color Markers</h1>

  <div class="container">

    <div class="marker">

    </div>

  </div>

</body>

# Step 11 – add color to an element

It's time to add some color to the page. 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:

.freecodecamp {

}

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

.marker {

  background-color: red;

}



# Step 12 – width and height properties

Notice that your marker doesn't seem to have any color. The background color was actually applied, but since the marker div element is empty, it doesn't have any height by default.

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

.marker {

  background-color: red;

  width: 200px;

  height: 25px;

}



# Step 13 – margin property

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.

.marker {

  width: 200px;

  height: 25px;

  background-color: red;

  margin: auto;

}



# Step 14 – div elements

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.

<div class="container">

  <div class="marker">

  </div>

  <div class="marker">

  </div>

  <div class="marker">

  </div>

</div>



# Step 15 – separate markers using margin property

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.

.marker {

  width: 200px;

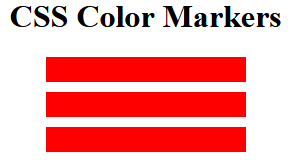
  height: 25px;

  background-color: red;

  margin: auto;

  margin: 10px auto;

}



# Step 16 – add color

In school, you might have learned that red, yellow, and blue are primary colors, and learned how to create new colors by mixing those. However, this is an outdated model.

These days, 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.

First, add the class one to the first marker div element.

<div class="exampleClass1 exampleClass2">

...content

</div>

<div class="marker one">

</div>

<div class="marker">

</div>

<div class="marker">

</div>

# Step 17 – background-color property

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

.marker {

  width: 200px;

  height: 25px;

  margin: 10px auto;

}

# Step 18 – new CSS rule for class one

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

.one{

  background-color: red;

}

A picture containing shape

Description automatically generated

# Step 19 – class two

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

<div class="marker one">

</div>

<div class="marker two">

</div>

<div class="marker three">

</div>

# Step 20 – background-color class two and three

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.

.two {

  background-color: green;

}

.three{

  background-color: blue;

}



# Step 21 – background-colour property

Earlier you learned that the RGB color model is additive. 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).

.container {

  background-color: rgb(0, 0, 0);

}



# Step 22 – rgb function

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:

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.

.one {

  background-color: rgb(255, 0, 0);

}

# Step 23 – rgb function

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.

.two {

  background-color: rgb(0, 255, 0);

}

.three {

  background-color: rgb(0, 0, 255);

}

# Step 24 – rgb function

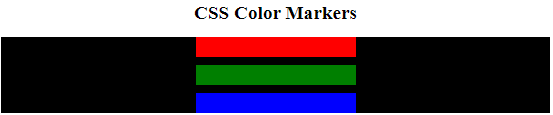
While the red and blue markers look the same, the green one is much lighter than it was before. This is because the green color keyword is actually a darker shade, and is about halfway between black and the maximum value for green.

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

.two {

  background-color: rgb(0, 127, 0);

}



# Step 25 – padding property

Now add a little more vertical space between your markers and the edge of the container element they're in.

In the .container CSS rule, use the shorthand padding property to add 10px of top and bottom padding, and set the left and right padding to 0. This works similarly to the shorthand margin property you used earlier.

.container {

  background-color: rgb(0, 0, 0);

  padding: 10px 0;

}



# Step 26 – rgb function

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.

.two {

  background-color: rgb(0, 255, 0);

}

# Step 27 – rgb function

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.

.container {

  background-color: rgb(255, 255, 255);

  padding: 10px 0;

}

Chart

Description automatically generated with medium confidence

# Step 28 – secondary colors

*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.

.one {

  background-color: rgb(255, 255, 0);

}

Chart

Description automatically generated

# Step 29 – secondary colors

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

.two {

  background-color: rgb(0, 255, 255);

}



# Step 30 – secondary colors

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

.three {

  background-color: rgb(255, 0, 255);

}

Chart, bar chart

Description automatically generated

# Step 31 – tertiary colors

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.

.one {

  background-color: rgb(255, 127, 0);

}

Chart, bar chart

Description automatically generated

# Step 32 – tertiary colors

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, and falls between the two colors on the color wheel.

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.

.two {

  background-color: rgb(0, 255, 127);

}



# Step 33 – tertiary colors

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.

.three {

  background-color: rgb(127, 0, 255);

}

Chart, bar chart

Description automatically generated

# Step 34 – tertiary colors

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 rule so that red is at 127, and set green to the max value.

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

.one {

  background-color: rgb(127, 255, 0);

}

.two {

  background-color: rgb(0, 127, 255);

}

.three {

  background-color: rgb(255, 0, 127);

}

Chart, bar chart

Description automatically generated

# Step 35 – rgb function

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 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.

.one {

  background-color: rgb(0, 0, 0);

}

.two {

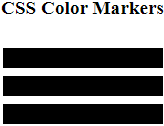
  background-color: rgb(0, 0, 0);

}

.three {

  background-color: rgb(0, 0, 0);

}



# Step 36 – rgb function

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 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.

.one {

  background-color: rgb(255, 0, 0);

}

.two {

  background-color: rgb(0, 255, 255);

}



# Step 37 – rgb function

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.

h1 {

  text-align: center;

  background-color: rgb(0, 255, 255);

}



# Step 38 – rgb function

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

.one {

  background-color: rgb(0, 0, 0);

}

.two {

  background-color: rgb(255, 0, 0);

}

A picture containing icon

Description automatically generated

# Step 39 – rgb function

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 rule to set the background-color to black.

.two {

  background-color: rgb(0, 0, 0);

}



# Step 40 – background-color property

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

h1 {

  text-align: center;

}

Background pattern

Description automatically generated

# Step 41 – div element

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.

<div class="marker red">

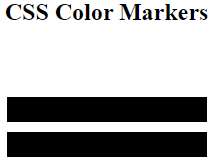
</div>

<div class="marker two">

</div>

<div class="marker three">

</div>



# Step 42 - .one class selector

Update the .one class selector to target the new red class.

.red {

  background-color: rgb(0, 0, 0);

}

.two {

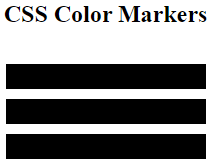
  background-color: rgb(0, 0, 0);

}

.three {

  background-color: rgb(0, 0, 0);

}



# Step 43 – update rgb function

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

.red {

  background-color: rgb(255, 0, 0);

}

.two {

  background-color: rgb(0, 0, 0);

}

.three {

  background-color: rgb(0, 0, 0);

}



# Step 44 - .two and .three class selectors

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

<div class="marker red">

</div>

<div class="marker green">

</div>

<div class="marker blue">

</div>

A picture containing shape

Description automatically generated

# Step 45 – update class selector

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

.red {

  background-color: rgb(255, 0, 0);

}

.green {

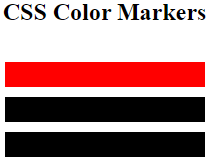
  background-color: rgb(0, 0, 0);

}

.blue {

  background-color: rgb(0, 0, 0);

}



# Step 46 – hexadecimal background-color property

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

.green {

  background-color: #00FF00;

}

A picture containing rectangle

Description automatically generated

# Step 47 – hexadecimal background-property

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:

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 green value of the hex color to 7F.

.green {

  background-color: #007F00;

}



# Step 48 – HSL color model

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.

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%.

.blue {

  background-color: hsl(240, 100%, 50%);

}

Chart, bar chart

Description automatically generated

# Step 49 – linear-gradient function

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.

.red {

  background: rgb(255, 0, 0);

}



# Step 50 – linear-gradient function

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

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.

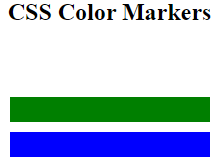
Now you'll apply a red-to-green gradient along a 90 degree line to the first marker.

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

.red {

  background: linear-gradient(90deg);

}



# Step 51 – linear-gradient function

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.

.red {

  background: linear-gradient(90deg, rgb(255, 0, 0));

}

# Step 52 – linear-gradient function

You won't see 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.

.red {

  background: linear-gradient(90deg, rgb(255, 0, 0), rgb(0, 255, 0));

}

Chart, bar chart

Description automatically generated

# Step 53 – linear-gradient function

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.

.red {

  background: linear-gradient(90deg, rgb(255, 0, 0), rgb(0, 255, 0), rgb(0, 0, 255));

}

Chart, bar chart

Description automatically generated

# Step 54 – linear-gradient function

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:

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.

.red {

  background: linear-gradient(90deg, rgb(255, 0, 0) 75%, rgb(0, 255, 0), rgb(0, 0, 255));

}

A picture containing bar chart

Description automatically generated

# Step 55 – linear-gradient function

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.

.red {

  background: linear-gradient(180deg, rgb(255, 0, 0) 75%, rgb(0, 255, 0), rgb(0, 0, 255));

}

A picture containing chart

Description automatically generated

# Step 56 – color-stop linear-gradient function

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

.red {

  background: linear-gradient(180deg, rgb(255, 0, 0) 0%, rgb(0, 255, 0) 50%, rgb(0, 0, 255) 100%);

}

Chart, bar chart

Description automatically generated

# Step 57 – linear-gradient function

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.

.red {

  background: linear-gradient(180deg, rgb(122, 74, 14) 0%, rgb(0, 255, 0) 50%, rgb(0, 0, 255) 100%);

}

A picture containing chart

Description automatically generated

# Step 58 – linear-gradient function

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.

.red {

  background: linear-gradient(180deg, rgb(122, 74, 14) 0%, rgb(245, 62, 113) 50%, rgb(0, 0, 255) 100%);

}

Chart, bar chart

Description automatically generated

# Step 59 – linear-gradient function

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.

.red {

  background: linear-gradient(180deg, rgb(122, 74, 14) 0%, rgb(245, 62, 113) 50%, rgb(162, 27, 27) 100%);

}

Chart, bar chart

Description automatically generated

# Step 60 – background property

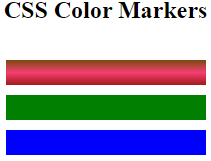
The red marker is looking much more realistic. Now you'll do the same for the green marker, using a combination of the linear-gradient function and hex colors.

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

.green {

  background: #007F00;

}



# Step 61 – linear-gradient function

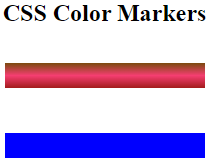
For this marker, you'll use hex color codes for your gradient.

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.

.green {

  background: linear-gradient(180deg, #55680D);

}



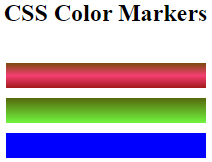
# Step 62 – linear-gradient function

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

.green {

  background: linear-gradient(180deg, #55680D, #71F53E);

}



# Step 63 – linear-gradient function

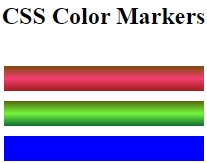
That's looking better, but the bottom edge of the green marker needs to be darker to add a little more dimension.

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.

.green {

  background: linear-gradient(180deg, #55680D, #71F53E, #116C31);

}



# Step 64 – linear-gradient function

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.

.red {

  background: linear-gradient(180deg, rgb(122, 74, 14), rgb(245, 62, 113), rgb(162, 27, 27));

}

Chart, bar chart

Description automatically generated

# Step 65 – linear-gradient function

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.

.red {

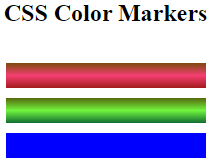
  background: linear-gradient(rgb(122, 74, 14), rgb(245, 62, 113), rgb(162, 27, 27));

}

.green {

  background: linear-gradient(#55680D, #71F53E, #116C31);

}



# Step 66 – background property blue marker

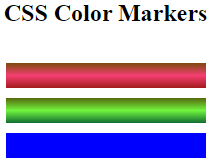
Now you'll apply a gradient to the blue marker, this time using the hsl function as color arguments.

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

.blue {

  background: hsl(240, 100%, 50%);

}



# Step 67 – linear-gradient function (hsl)

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.

.blue {

  background: linear-gradient(hsl(186, 76%, 16%));

}

A picture containing rectangle

Description automatically generated

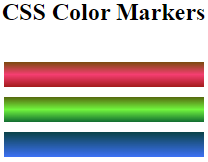
# Step 68 – linear-gradient function (hsl)

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

.blue {

  background: linear-gradient(hsl(186, 76%, 16%), hsl(223, 90%, 60%));

}



# Step 69 – light-gradient function (hsl)

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

.blue {

  background: linear-gradient(hsl(186, 76%, 16%), hsl(223, 90%, 60%), hsl(240, 56%, 42%));

}

Chart

Description automatically generated

# Step 70 – div element

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, create a new div and give it a class of sleeve.

<div class="marker red">

  <div class="sleeve">

  </div>

</div>

# Step 71 – width and height properties for class sleeve

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

.sleeve {

  width: 110px;

  height: 25px;

}

# Step 72 – sleeve element’s background-color

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

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

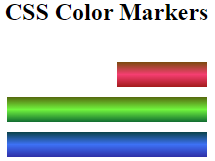
.sleeve {

  width: 110px;

  height: 25px;

  background-color: white;

}

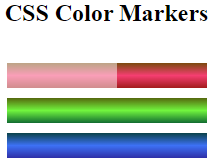


# Step 73 – opacity property

*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 – alpha channel

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 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.

.sleeve {

  width: 110px;

  height: 25px;

  background-color: white;

}

Chart, bar chart

Description automatically generated

# Step 75 – rgba function

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:

rgba(redValue, greenValue, blueValue, alphaValue);

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

.sleeve {

  width: 110px;

  height: 25px;

  background-color: rgba(255, 255, 255, 0.5);

}

Chart, bar chart

Description automatically generated