**­­­­­Basic CSS:**

1. **Change the Color of Text**   
   Now let's change the color of some of our text.  
   We can do this by changing the style of your h2 element.  
   The property that is responsible for the color of an element's text is the color style property.  
   Here's how you would set your h2 element's text color to blue:  
   <h2 style="color: blue;">CatPhotoApp</h2>  
   Note that it is a good practice to end inline style declarations with a ; .
2. **Use CSS Selectors to Style Elements**  
   With CSS, there are hundreds of CSS properties that you can use to change the way an element looks on your page.  
   When you entered <h2 style="color: red;">CatPhotoApp</h2>, you were styling that individual h2 element with inline CSS, which stands for Cascading Style Sheets.  
   That's one way to specify the style of an element, but there's a better way to apply CSS.  
   At the top of your code, create a style block like this:  
   <style>  
   </style>  
   Inside that style block, you can create a CSS selector for all h2 elements. For example, if you wanted all h2 elements to be red, you would add a style rule that looks like this:  
   <style>  
    h2 {  
    color: red;  
    }  
   </style>  
   <h2> "cat photos" </h2>  
   Note that it's important to have both opening and closing curly braces ({ and }) around each element's style rule(s). You also need to make sure that your element's style definition is between the opening and closing style tags. Finally, be sure to add a semicolon to the end of each of your element's style rules.
3. **Use a CSS Class to Style an Element**  
   Classes are reusable styles that can be added to HTML elements.  
     
   Here's an example CSS class declaration:  
   <style>  
    .blue-text {  
    color: blue;  
    }  
   </style>  
   <h2 class="blue-text">CatPhotoApp</h2>  
   You can see that we've created a CSS class called blue-text within the <style> tag. You can apply a class to an HTML element like this: <h2 class="blue-text">CatPhotoApp</h2> Note that in your CSS style element, class names start with a period. In your HTML elements' class attribute, the class name does not include the period.
4. **Style Multiple Elements with a CSS Class**  
   Classes allow you to use the same CSS styles on multiple HTML elements. You can see this by applying your red-text class to the first p element.  
   <style>  
    .red-text {  
    color: red;  
    }  
   </style>  
   <h2 class="red-text">CatPhotoApp</h2>  
   <main>  
   <p class="red-text">Click here to view more <a href="#">cat
5. **Change the Font Size of an Element**Font size is controlled by the font-size CSS property, like this:  
   h1 {  
   font-size: 30px;  
   }  
     
   <style>  
    .red-text {  
    color: red;  
    **p {  
   font-size: 16px;  
   }**  
    }  
   </style>
6. **Set the Font Family of an Element**You can set which font an element should use, by using the font-family property.  
   For example, if you wanted to set your h2 element's font to sans-serif, you would use the following CSS:  
   h2 {  
    font-family: sans-serif;  
   }  
     
   <style>  
    .red-text {  
    color: red;  
    }  
   p {  
    **font-family: monospace;** }  
   </style>
7. **Import a Google Font**In addition to specifying common fonts that are found on most operating systems, we can also specify non-standard, custom web fonts for use on our website. There are many sources for web fonts on the Internet. For this example we will focus on the Google Fonts library.  
     
   Google Fonts is a free library of web fonts that you can use in your CSS by referencing the font's URL.  
     
   So, let's go ahead and import and apply a Google font (note that if Google is blocked in your country, you will need to skip this challenge).  
     
   To import a Google Font, you can copy the font's URL from the Google Fonts library and then paste it in your HTML. For this challenge, we'll import the Lobster font. To do this, copy the following code snippet and paste it into the top of your code editor (before the opening style element):  
     
   <link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">  
     
   Now you can use the Lobster font in your CSS by using Lobster as the FAMILY\_NAME as in the following example:  
   font-family: FAMILY\_NAME, GENERIC\_NAME;.  
     
   The GENERIC\_NAME is optional, and is a fallback font in case the other specified font is not available. This is covered in the next challenge.  
   Family names are case-sensitive and need to be wrapped in quotes if there is a space in the name. For example, you need quotes to use the "Open Sans" font, but not to use the Lobster font

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">

<style>

  .red-text {

    color: red;

  }

h2 {

  font-family: Lobster;

}

  p {

    font-size: 16px;

    font-family: monospace;

  }

</style>

<h2 class="red-text">CatPhotoApp</h2>

1. Specify How Fonts Should Degrade

There are several default fonts that are available in all browsers. These generic font families include monospace, serif and sans-serif

When one font isn't available, you can tell the browser to "degrade" to another font.

For example, if you wanted an element to use the Helvetica font, but degrade to the sans-serif font when Helvetica isn't available, you will specify it as follows:

p {

font-family: Helvetica, sans-serif;

}

Generic font family names are not case-sensitive. Also, they do not need quotes because they are CSS keywords.

<!--

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">

-->

<style>

  .red-text {

    color: red;

  }

h2 {

    font-family: Lobster, monospace;

  }

p {

    font-size: 16px;

    font-family: monospace;

  }

</style>

1. **Size Your Images**

CSS has a property called width that controls an element's width. Just like with fonts, we'll use px (pixels) to specify the image's width.

For example, if we wanted to create a CSS class called larger-image that gave HTML elements a width of 500 pixels, we'd use:

<style>

.larger-image {

width: 500px;

}

</style>

Create a class called smaller-image and use it to resize the image so that it's only 100 pixels wide.

<style>

  .smaller-image {

    width: 100px;

  }

</style>

<a href="#"><img class=smaller-image src="https://bit.ly/fcc-relaxing-cat" alt="A cute orange cat lying on its back."></a>

1. **Add Borders Around Your Elements**

CSS borders have properties like style, color and width.

For example, if we wanted to create a red, 5 pixel border around an HTML element, we could use this class:

<style>

.thin-red-border {

border-color: red;

border-width: 5px;

border-style: solid;

}

</style>

Create a class called thick-green-border. This class should add a 10px, solid, green border around an HTML element. Apply the class to your cat photo.

Remember that you can apply multiple classes to an element using its class attribute, by separating each class name with a space. For example:

<img class="class1 class2">

<style>

  .smaller-image {

    width: 100px;

  }

  .thick-green-border {

    border-color: green;

    border-width: 10px;

    border-style: solid;

}

</style>

<a href="#"><img class="smaller-image thick-green-border" src="https://bit.ly/fcc-relaxing-cat" alt="A cute orange cat lying on its back."></a>

1. Add Rounded Corners with border-radius

Your cat photo currently has sharp corners. We can round out those corners with a CSS property called border-radius.

You can specify a border-radius with pixels. Give your cat photo a border-radius of 10px.

**Note:** This challenge allows for multiple possible solutions. For example, you may add border-radius to either the .thick-green-border class or the .smaller-image class.

<style>

  .thick-green-border{

    border-color: green;

    border-width: 10px;

    border-style: solid;

    border-radius: 10px;

  }

.smaller-image {

    width: 100px;

  }

</style>

<a href="#"><img class="smaller-image thick-green-border" src="https://bit.ly/fcc-relaxing-cat" alt="A cute orange cat lying on its back."></a>

1. **Make Circular Images with a border-radius**

In addition to pixels, you can also specify the border-radius using a percentage.

Give your cat photo a border-radius of 50%.

<style>

.thick-green-border {

    border-color: green;

    border-width: 10px;

    border-style: solid;

    border-radius: 50%;

  }

  .smaller-image {

    width: 100px;

  }

</style>

  <a href="#"><img class="smaller-image thick-green-border" src="https://bit.ly/fcc-relaxing-cat" alt="A cute orange cat lying on its back."></a>

1. **Give a Background Color to a div Element**

You can set an element's background color with the background-color property.

For example, if you wanted an element's background color to be green, you'd put this within your style element:

.green-background {

background-color: green;

}

Create a class called silver-background with the background-color of silver. Assign this class to your div element.

<style>

  .silver-background {

  background-color: silver;

}

</style>

  <div class="silver-background" >

. . .

  </div>

1. **Set the id of an Element**

In addition to classes, each HTML element can also have an id attribute.

There are several benefits to using id attributes: You can use an id to style a single element and later you'll learn that you can use them to select and modify specific elements with JavaScript.

id attributes should be unique. Browsers won't enforce this, but it is a widely agreed upon best practice. So please don't give more than one element the same id attribute.

Here's an example of how you give your h2 element the id of cat-photo-app:

<h2 id="cat-photo-app">

Give your form element the id cat-photo-form.

<form action="https://freecatphotoapp.com/submit-cat-photo" id="cat-photo-form">

    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>

    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>

    <label><input type="checkbox" name="personality" checked> Loving</label>

    <label><input type="checkbox" name="personality"> Lazy</label>

</form>

1. **Use an id Attribute to Style an Element**

One cool thing about id attributes is that, like classes, you can style them using CSS.

However, an id is not reusable and should only be applied to one element. An id also has a higher specificity (importance) than a class so if both are applied to the same element and have conflicting styles, the styles of the id will be applied.

Here's an example of how you can take your element with the id attribute of cat-photo-element and give it the background color of green. In your style element:

#cat-photo-element {

background-color: green;

}

Note that inside your style element, you always reference classes by putting a . in front of their names. You always reference ids by putting a # in front of their names.

Try giving your form, which now has the id attribute of cat-photo-form, a green background.

<style>

  #cat-photo-form {

  background-color: green;

}

</style>

<form action="https://freecatphotoapp.com/submit-cat-photo" id="cat-photo-form">

    <label><input type="radio" name="indoor-outdoor" checked> Indoor</label>

    <label><input type="radio" name="indoor-outdoor"> Outdoor</label><br>

</form>

1. **Adjust the Padding of an Element**

You may have already noticed this, but all HTML elements are essentially little rectangles.

Three important properties control the space that surrounds each HTML element: padding, border, and margin.

An element's padding controls the amount of space between the element's content and its border.

Here, we can see that the blue box and the red box are nested within the yellow box. Note that the red box has more padding than the blue box.

When you increase the blue box's padding, it will increase the distance (padding) between the text and the border around it.

Change the padding of your blue box to match that of your red box.

<style>

  .injected-text {

    margin-bottom: -25px;

    text-align: center;

  }

  .box {

    border-style: solid;

    border-color: black;

    border-width: 5px;

    text-align: center;

  }

  .yellow-box {

    background-color: yellow;

    padding: 10px;

  }

  .red-box {

    background-color: crimson;

    color: #fff;

    padding: 20px;

  }

  .blue-box {

    background-color: blue;

    color: #fff;

    padding: 20px;

  }

</style>

<h5 class="injected-text">margin</h5>

<div class="box yellow-box">

  <h5 class="box red-box">padding</h5>

  <h5 class="box blue-box">padding</h5>

</div>

1. **Adjust the Margin of an Element**

An element's margin controls the amount of space between an element's border and surrounding elements.

Here, we can see that the blue box and the red box are nested within the yellow box. Note that the red box has a bigger margin than the blue box, making it appear smaller.

When you increase the blue box's margin, it will increase the distance between its border and surrounding elements.

Change the margin of the blue box to match that of the red box.

<style>

  .injected-text {

    margin-bottom: -25px;

    text-align: center;

  }

  .box {

    border-style: solid;

    border-color: black;

    border-width: 5px;

    text-align: center;

  }

  .yellow-box {

    background-color: yellow;

    padding: 10px;

  }

  .red-box {

    background-color: crimson;

    color: #fff;

    padding: 20px;

    margin: 20px;

  }

  .blue-box {

    background-color: blue;

    color: #fff;

    padding: 20px;

    margin: 20px;

  }

</style>

<h5 class="injected-text">margin</h5>

<div class="box yellow-box">

  <h5 class="box red-box">padding</h5>

  <h5 class="box blue-box">padding</h5>

</div>

1. **Add a Negative Margin to an Element**

An element's margin controls the amount of space between an element's border and surrounding elements.

If you set an element's margin to a negative value, the element will grow larger.

Try to set the margin to a negative value like the one for the red box.

Change the margin of the blue box to -15px, so it fills the entire horizontal width of the yellow box around it.

<style>

  .injected-text {

    margin-bottom: -25px;

    text-align: center;

  }

  .box {

    border-style: solid;

    border-color: black;

    border-width: 5px;

    text-align: center;

  }

  .yellow-box {

    background-color: yellow;

    padding: 10px;

  }

  .red-box {

    background-color: crimson;

    color: #fff;

    padding: 20px;

    margin: -15px;

  }

  .blue-box {

    background-color: blue;

    color: #fff;

    padding: 20px;

    margin: -15px;

  }

</style>

<div class="box yellow-box">

  <h5 class="box red-box">padding</h5>

  <h5 class="box blue-box">padding</h5>

</div>

1. **Add Different Padding to Each Side of an Element**

Sometimes you will want to customize an element so that it has different amounts of padding on each of its sides.

CSS allows you to control the padding of all four individual sides of an element with the padding-top, padding-right, padding-bottom, and padding-left properties.

Give the blue box a padding of 40px on its top and left side, but only 20px on its bottom and right side.

<style>

  .injected-text {

    margin-bottom: -25px;

    text-align: center;

  }

  .box {

    border-style: solid;

    border-color: black;

    border-width: 5px;

    text-align: center;

  }

  .yellow-box {

    background-color: yellow;

    padding: 10px;

  }

  .red-box {

    background-color: crimson;

    color: #fff;

    padding-top: 40px;

    padding-right: 20px;

    padding-bottom: 20px;

    padding-left: 40px;

  }

  .blue-box {

    background-color: blue;

    color: #fff;

    padding-top: 40px;

    padding-right: 20px;

    padding-bottom: 20px;

    padding-left: 40px;

  }

</style>

<h5 class="injected-text">margin</h5>

<div class="box yellow-box">

  <h5 class="box red-box">padding</h5>

  <h5 class="box blue-box">padding</h5>

</div>

1. **Add Different Margins to Each Side of an Element**

Sometimes you will want to customize an element so that it has a different margin on each of its sides.

CSS allows you to control the margin of all four individual sides of an element with the margin-top, margin-right, margin-bottom, and margin-left properties.

Give the blue box a margin of 40px on its top and left side, but only 20px on its bottom and right side.

<style>

  .injected-text {

    margin-bottom: -25px;

    text-align: center;

  }

  .box {

    border-style: solid;

    border-color: black;

    border-width: 5px;

    text-align: center;

  }

  .yellow-box {

    background-color: yellow;

    padding: 10px;

  }

  .red-box {

    background-color: crimson;

    color: #fff;

    margin-top: 40px;

    margin-right: 20px;

    margin-bottom: 20px;

    margin-left: 40px;

  }

  .blue-box {

    background-color: blue;

    color: #fff;

    margin-top: 40px;

    margin-right: 20px;

    margin-bottom: 20px;

    margin-left: 40px;

  }

</style>

<h5 class="injected-text">margin</h5>

<div class="box yellow-box">

  <h5 class="box red-box">padding</h5>

  <h5 class="box blue-box">padding</h5>

</div>

1. Use Clockwise Notation to Specify the Padding of an Element

Instead of specifying an element's padding-top, padding-right, padding-bottom, and padding-left properties individually, you can specify them all in one line, like this:

padding: 10px 20px 10px 20px;

These four values work like a clock: top, right, bottom, left, and will produce the exact same result as using the side-specific padding instructions.

Use Clockwise Notation to give the ".blue-box" class a padding of 40px on its top and left side, but only 20px on its bottom and right side.

<style>

  .injected-text {

    margin-bottom: -25px;

    text-align: center;

  }

  .box {

    border-style: solid;

    border-color: black;

    border-width: 5px;

    text-align: center;

  }

  .yellow-box {

    background-color: yellow;

    padding: 20px 40px 20px 40px;

  }

  .red-box {

    background-color: crimson;

    color: #fff;

    padding: 20px 40px 20px 40px;

  }

  .blue-box {

    background-color: blue;

    color: #fff;

    padding: 40px 20px 20px 40px;

  }

</style>

<h5 class="injected-text">margin</h5>

<div class="box yellow-box">

  <h5 class="box red-box">padding</h5>

  <h5 class="box blue-box">padding</h5>

</div>

1. **Use Clockwise Notation to Specify the Margin of an Element**

Let's try this again, but with margin this time.

Instead of specifying an element's margin-top, margin-right, margin-bottom, and margin-left properties individually, you can specify them all in one line, like this:

margin: 10px 20px 10px 20px;

These four values work like a clock: top, right, bottom, left, and will produce the exact same result as using the side-specific margin instructions.

Use Clockwise Notation to give the element with the blue-box class a margin of 40px on its top and left side, but only 20px on its bottom and right side.

<style>

  .injected-text {

    margin-bottom: -25px;

    text-align: center;

  }

  .box {

    border-style: solid;

    border-color: black;

    border-width: 5px;

    text-align: center;

  }

  .yellow-box {

    background-color: yellow;

    padding: 20px 40px 20px 40px;

  }

  .red-box {

    background-color: crimson;

    color: #fff;

    margin: 20px 40px 20px 40px;

  }

  .blue-box {

    background-color: blue;

    color: #fff;

    margin: 40px 20px 20px 40px;

  }

</style>

<h5 class="injected-text">margin</h5>

1. **Use Attribute Selectors to Style Elements**

You have been adding id or class attributes to elements that you wish to specifically style. These are known as ID and class selectors. There are other CSS Selectors you can use to select custom groups of elements to style.

Let's bring out CatPhotoApp again to practice using CSS Selectors.

For this challenge, you will use the [attr=value] attribute selector to style the checkboxes in CatPhotoApp. This selector matches and styles elements with a specific attribute value. For example, the below code changes the margins of all elements with the attribute type and a corresponding value of radio:

[type='radio'] {

margin: 20px 0px 20px 0px;

}

Using the type attribute selector, try to give the checkboxes in CatPhotoApp a top margin of 10px and a bottom margin of 15px.

<style>

    [type='checkbox'] {

    margin: 10px 0px 15px 0px;

  }

</style>

 <form action="https://freecatphotoapp.com/submit-cat-photo" id="cat-photo-form">

    <label><input type="checkbox" name="personality" checked> Loving</label>

    <label><input type="checkbox" name="personality"> Lazy</label>

    <label><input type="checkbox" name="personality"> Energetic</label><br>

</main>

1. **Understand Absolute versus Relative Units**

The last several challenges all set an element's margin or padding with pixels (px). Pixels are a type of length unit, which is what tells the browser how to size or space an item. In addition to px, CSS has a number of different length unit options that you can use.

The two main types of length units are absolute and relative. Absolute units tie to physical units of length. For example, in and mm refer to inches and millimeters, respectively. Absolute length units approximate the actual measurement on a screen, but there are some differences depending on a screen's resolution.

Relative units, such as em or rem, are relative to another length value. For example, em is based on the size of an element's font. If you use it to set the font-size property itself, it's relative to the parent's font-size.

**Note:** There are several relative unit options that are tied to the size of the viewport. They are covered in the Responsive Web Design Principles section.

Add a padding property to the element with class red-box and set it to 1.5em.

<style>

  .injected-text {

    margin-bottom: -25px;

    text-align: center;

  }

  .box {

    border-style: solid;

    border-color: black;

    border-width: 5px;

    text-align: center;

  }

  .yellow-box {

    background-color: yellow;

    padding: 20px 40px 20px 40px;

  }

  .red-box {

    background-color: red;

    margin: 20px 40px 20px 40px;

    padding: 1.5em;

  }

  .green-box {

    background-color: green;

    margin: 20px 40px 20px 40px;

  }

</style>

<h5 class="injected-text">margin</h5>

<div class="box yellow-box">

  <h5 class="box red-box">padding</h5>

  <h5 class="box green-box">padding</h5>

</div>

1. **Style the HTML Body Element**

Now let's start fresh and talk about CSS inheritance.

Every HTML page has a body element.

We can prove that the body element exists here by giving it a background-color of black.

We can do this by adding the following to our style element:

<style>

body {

  background-color: black;

}

</style>

1. **Inherit Styles from the Body Element**

Now we've proven that every HTML page has a body element, and that its body element can also be styled with CSS.

Remember, you can style your body element just like any other HTML element, and all your other elements will inherit your body element's styles.

First, create a h1 element with the text Hello World

Then, let's give all elements on your page the color of green by adding color: green; to your body element's style declaration.

Finally, give your body element the font-family of monospace by adding font-family: monospace; to your body element's style declaration.

<h1>Hello World</h1>

<style>

  body {

    background-color: black;

    color: green;

    font-family: monospace;

      }

</style>

1. **Prioritize One Style Over Another**

Sometimes your HTML elements will receive multiple styles that conflict with one another.

For example, your h1 element can't be both green and pink at the same time.

Let's see what happens when we create a class that makes text pink, then apply it to an element. Will our class *override* the body element's color: green; CSS property?

Create a CSS class called pink-text that gives an element the color pink.

Give your h1 element the class of pink-text.

<style>

  body {

    background-color: black;

    font-family: monospace;

    color: green;

  }

  .pink-text {

    color: pink;

  }

</style>

<h1 class="pink-text">Hello World!</h1>

1. **Override Styles in Subsequent CSS**

Our "pink-text" class overrode our body element's CSS declaration!

We just proved that our classes will override the body element's CSS. So the next logical question is, what can we do to override our pink-text class?

Create an additional CSS class called blue-text that gives an element the color blue. Make sure it's below your pink-text class declaration.

Apply the blue-text class to your h1 element in addition to your pink-text class, and let's see which one wins.

Applying multiple class attributes to a HTML element is done with a space between them like this:

class="class1 class2"

**Note:** It doesn't matter which order the classes are listed in the HTML element.

However, the order of the class declarations in the <style> section is what is important. The second declaration will always take precedence over the first. Because .blue-text is declared second, it overrides the attributes of .pink-text

<style>

  body {

    background-color: black;

    font-family: monospace;

    color: green;

  }

  .pink-text {

    color: pink;

  }

  .blue-text {

    color: blue;

  }

</style>

<h1 class="blue-text pink-text">Hello World!</h1>

1. **Override Class Declarations by Styling ID Attributes**

We just proved that browsers read CSS from top to bottom in order of their declaration. That means that, in the event of a conflict, the browser will use whichever CSS declaration came last. Notice that if we even had put blue-text before pink-text in our h1 element's classes, it would still look at the declaration order and not the order of their use!

But we're not done yet. There are other ways that you can override CSS. Do you remember id attributes?

Let's override your pink-text and blue-text classes, and make your h1 element orange, by giving the h1 element an id and then styling that id.

Give your h1 element the id attribute of orange-text. Remember, id styles look like this:

<h1 id="orange-text">

Leave the blue-text and pink-text classes on your h1 element.

Create a CSS declaration for your orange-text id in your style element. Here's an example of what this looks like:

#brown-text {

color: brown;

}

**Note:** It doesn't matter whether you declare this CSS above or below pink-text class, since id attribute will always take precedence.

<style>

  body {

    background-color: black;

    font-family: monospace;

    color: green;

  }

   .pink-text {

    color: pink;

  }

   #orange-text {

    color: orange;

  }

  .blue-text {

    color: blue;

  }

 </style>

<h1 class="pink-text blue-text" id="orange-text" >Hello World!</h1>

1. **Override Class Declarations with Inline Styles**

So we've proven that id declarations override class declarations, regardless of where they are declared in your style element CSS.

There are other ways that you can override CSS. Do you remember inline styles?

Use an inline style to try to make our h1 element white. Remember, in line styles look like this:

<h1 style="color: green;">

Leave the blue-text and pink-text classes on your h1 element.

<style>

  body {

    background-color: black;

    font-family: monospace;

    color: green;

  }

  #orange-text {

    color: orange;

  }

  .pink-text {

    color: pink;

  }

  .blue-text {

    color: blue;

  }

</style>

<h1 style="color: white;" id="orange-text" class="pink-text blue-text">Hello World!</h1>

1. Override All Other Styles by using Important

Yay! We just proved that inline styles will override all the CSS declarations in your style element.

But wait. There's one last way to override CSS. This is the most powerful method of all. But before we do it, let's talk about why you would ever want to override CSS.

In many situations, you will use CSS libraries. These may accidentally override your own CSS. So when you absolutely need to be sure that an element has specific CSS, you can use !important

Let's go all the way back to our pink-text class declaration. Remember that our pink-text class was overridden by subsequent class declarations, id declarations, and inline styles.

Let's add the keyword !important to your pink-text element's color declaration to make 100% sure that your h1 element will be pink.

An example of how to do this is:

color: red !important;

<style>

  body {

    background-color: black;

    font-family: monospace;

    color: green;

  }

  #orange-text {

    color: orange;

  }

  .pink-text {

    color: pink !important;

  }

  .blue-text {

    color: blue;

  }

</style>

<h1 id="orange-text" class="pink-text blue-text" style="color: white">Hello World!</h1>

1. Use Hex Code for Specific Colors

Did you know there are other ways to represent colors in CSS? One of these ways is called hexadecimal code, or hex code for short.

We usually use *decimals*, or base 10 numbers, which use the symbols 0 to 9 for each digit. *Hexadecimals* (or *hex*) are base 16 numbers. This means it uses sixteen distinct symbols. Like decimals, the symbols 0-9 represent the values zero to nine. Then A,B,C,D,E,F represent the values ten to fifteen. Altogether, 0 to F can represent a digit in hexadecimal, giving us 16 total possible values. You can find more information about [hexadecimal numbers here](https://en.wikipedia.org/wiki/Hexadecimal).

In CSS, we can use 6 hexadecimal digits to represent colors, two each for the red (R), green (G), and blue (B) components. For example, #000000 is black and is also the lowest possible value. You can find more information about the [RGB color system here](https://en.wikipedia.org/wiki/RGB_color_model).

body {

color: #000000;

}

Replace the word black in our body element's background-color with its hex code representation, #000000.

<style>

  body {

    background-color: #000000;

  }

</style>

1. **Use Hex Code to Mix Colors**

To review, hex codes use 6 hexadecimal digits to represent colors, two each for red (R), green (G), and blue (B) components.

From these three pure colors (red, green, and blue), we can vary the amounts of each to create over 16 million other colors!

For example, orange is pure red, mixed with some green, and no blue. In hex code, this translates to being #FFA500.

The digit 0 is the lowest number in hex code, and represents a complete absence of color.

The digit F is the highest number in hex code, and represents the maximum possible brightness.

Replace the color words in our style element with their correct hex codes.

|  |  |
| --- | --- |
| **Color** | **Hex Code** |
| Dodger Blue | #1E90FF |
| Green | #00FF00 |
| Orange | #FFA500 |
| Red | #FF0000 |

<style>

  .red-text {

    color: #ff0000;

  }

  .green-text {

    color: #00FF00;

  }

  .dodger-blue-text {

    color: #1e90FF;

  }

  .orange-text {

    color: #ffa500;

  }

</style>

<h1 class="red-text">I am red!</h1>

<h1 class="green-text">I am green!</h1>

<h1 class="dodger-blue-text">I am dodger blue!</h1>

<h1 class="orange-text">I am orange!</h1>

1. **Use Abbreviated Hex Code**

Many people feel overwhelmed by the possibilities of more than 16 million colors. And it's difficult to remember hex code. Fortunately, you can shorten it.

For example, red's hex code #FF0000 can be shortened to #F00. This shortened form gives one digit for red, one digit for green, and one digit for blue.

This reduces the total number of possible colors to around 4,000. But browsers will interpret #FF0000 and #F00 as exactly the same color.

Go ahead, try using the abbreviated hex codes to color the correct elements.

|  |  |
| --- | --- |
| **Color** | **Short Hex Code** |
| Cyan | #0FF |
| Green | #0F0 |
| Red | #F00 |
| Fuchsia | #F0F |

<style>

  .red-text {

    color: #f00;

  }

  .fuchsia-text {

    color: #f0f;

  }

  .cyan-text {

    color: #0ff;

  }

  .green-text {

    color: #0f0;

  }

</style>

<h1 class="red-text">I am red!</h1>

<h1 class="fuchsia-text">I am fuchsia!</h1>

<h1 class="cyan-text">I am cyan!</h1>

<h1 class="green-text">I am green!</h1>

1. Use RGB values to Color Elements

Another way you can represent colors in CSS is by using RGB values.

The RGB value for black looks like this:

rgb(0, 0, 0)

The RGB value for white looks like this:

rgb(255, 255, 255)

Instead of using six hexadecimal digits like you do with hex code, with RGB you specify the brightness of each color with a number between 0 and 255.

If you do the math, the two digits for one color equal 16 times 16, which gives us 256 total values. So RGB, which starts counting from zero, has the exact same number of possible values as hex code.

Here's an example of how you'd change the body background to orange using its RGB code.

body {

background-color: rgb(255, 165, 0);

}

Let's replace the hex code in our body element's background color with the RGB value for black: rgb(0, 0, 0)

<style>

  body {

    background-color: rgb(0, 0, 0);

  }

</style>

1. **Use RGB to Mix Colors**

Just like with hex code, you can mix colors in RGB by using combinations of different values.

Replace the hex codes in our style element with their correct RGB values.

|  |  |
| --- | --- |
| **Color** | **RGB** |
| Blue | rgb(0, 0, 255) |
| Red | rgb(255, 0, 0) |
| Orchid | rgb(218, 112, 214) |
| Sienna | rgb(160, 82, 45) |

<style>

  .red-text {

    color: rgb(255, 0, 0);

  }

  .orchid-text {

    color: rgb(218, 112, 214);

  }

  .sienna-text {

    color: rgb(160, 82, 45);

  }

  .blue-text {

    color: rgb(0, 0, 255);

  }

</style>

<h1 class="red-text">I am red!</h1>

<h1 class="orchid-text">I am orchid!</h1>

<h1 class="sienna-text">I am sienna!</h1>

<h1 class="blue-text">I am blue!</h1>

1. **Use CSS Variables to change several elements at once**

*CSS Variables* are a powerful way to change many CSS style properties at once by changing only one value.

Follow the instructions below to see how changing just three values can change the styling of many elements.

In the penguin class, change the black value to gray, the gray value to white, and the yellow value to orange.

<style>

  .penguin {

    /\* Only change code below this line \*/

    --penguin-skin: black;

    --penguin-belly: gray;

    --penguin-beak: yellow;

    /\* Only change code above this line \*/

1. Create a custom CSS Variable

To create a CSS variable, you just need to give it a name with two hyphens in front of it and assign it a value like this:

--penguin-skin: gray;

This will create a variable named --penguin-skin and assign it the value of gray. Now you can use that variable elsewhere in your CSS to change the value of other elements to gray.

In the penguin class, create a variable name --penguin-skin and give it a value of gray.

<style>

  .penguin {

    /\* Only change code below this line \*/

--penguin-skin: gray;

    /\* Only change code above this line \*/

1. **Use a custom CSS Variable**

After you create your variable, you can assign its value to other CSS properties by referencing the name you gave it.

background: var(--penguin-skin);

This will change the background of whatever element you are targeting to gray because that is the value of the --penguin-skin variable. Note that styles will not be applied unless the variable names are an exact match.

Apply the --penguin-skin variable to the background property of the penguin-top, penguin-bottom, right-hand and left-hand classes.

<style>

  .penguin {

    --penguin-skin: gray;

    position: relative;

    margin: auto;

    display: block;

    margin-top: 5%;

    width: 300px;

    height: 300px;

  }

  .penguin-top {

    top: 10%;

    left: 25%;

    /\* Change code below this line \*/

   background: var(--penguin-skin);

    /\* Change code above this line \*/

    width: 50%;

    height: 45%;

    border-radius: 70% 70% 60% 60%;

  }

  .penguin-bottom {

    top: 40%;

    left: 23.5%;

    /\* Change code below this line \*/

   background: var(--penguin-skin);

    /\* Change code above this line \*/

1. **Attach a Fallback value to a CSS Variable**

When using your variable as a CSS property value, you can attach a fallback value that your browser will revert to if the given variable is invalid.

**Note:** This fallback is not used to increase browser compatibility, and it will not work on IE browsers. Rather, it is used so that the browser has a color to display if it cannot find your variable.

Here's how you do it:

background: var(--penguin-skin, black);

This will set background to black if your variable wasn't set. Note that this can be useful for debugging.

It looks like there is a problem with the variables supplied to the .penguin-top and .penguin-bottom classes. Rather than fix the typo, add a fallback value of black to the background property of the .penguin-top and .penguin-bottom classes.

<style>

  .penguin {

    --penguin-skin: black;

    --penguin-belly: gray;

    --penguin-beak: yellow;

    position: relative;

    margin: auto;

    display: block;

    margin-top: 5%;

    width: 300px;

    height: 300px;

  }

  .penguin-top {

    top: 10%;

    left: 25%;

    /\* Change code below this line \*/

    background: var(--pengiun-skin, black);

    /\* Change code above this line \*/

    width: 50%;

    height: 45%;

    border-radius: 70% 70% 60% 60%;

  }

  .penguin-bottom {

    top: 40%;

    left: 23.5%;

    /\* Change code below this line \*/

    background: var(--pengiun-skin, black);

    /\* Change code above this line \*/

1. **Improve Compatibility with Browser Fallbacks**

When working with CSS you will likely run into browser compatibility issues at some point. This is why it's important to provide browser fallbacks to avoid potential problems.

When your browser parses the CSS of a webpage, it ignores any properties that it doesn't recognize or support. For example, if you use a CSS variable to assign a background color on a site, Internet Explorer will ignore the background color because it does not support CSS variables. In that case, the browser will use whatever value it has for that property. If it can't find any other value set for that property, it will revert to the default value, which is typically not ideal.

This means that if you do want to provide a browser fallback, it's as easy as providing another more widely supported value immediately before your declaration. That way an older browser will have something to fall back on, while a newer browser will just interpret whatever declaration comes later in the cascade.

It looks like a variable is being used to set the background color of the .red-box class. Let's improve our browser compatibility by adding another background declaration right before the existing declaration and set its value to red.

<style>

  :root {

    --red-color: red;

  }

  .red-box {

    background: red;

    background: var(--red-color);

    height: 200px;

    width:200px;

  }

</style>

<div class="red-box"></div>

1. **Inherit CSS Variables**

When you create a variable, it is available for you to use inside the selector in which you create it. It also is available in any of that selector's descendants. This happens because CSS variables are inherited, just like ordinary properties.

To make use of inheritance, CSS variables are often defined in the *:root* element.

:root is a *pseudo-class* selector that matches the root element of the document, usually the html element. By creating your variables in :root, they will be available globally and can be accessed from any other selector in the style sheet.

Define a variable named --penguin-belly in the :root selector and give it the value of pink. You can then see that the variable is inherited and that all the child elements which use it get pink backgrounds.

<style>

  :root {

    /\* Only change code below this line \*/

    --penguin-belly: pink;

    /\* Only change code above this line \*/

  }

  body {

    background: var(--penguin-belly, #c6faf1);

  }

  .penguin {

    --penguin-skin: gray;

    --penguin-beak: orange;

    position: relative;

   }

1. **Change a variable for a specific area**

When you create your variables in :root they will set the value of that variable for the whole page.

You can then over-write these variables by setting them again within a specific element.

Change the value of --penguin-belly to white in the penguin class.

<style>

  :root {

    --penguin-skin: gray;

    --penguin-belly: pink;

    --penguin-beak: orange;

  }

  body {

    background: var(--penguin-belly, #c6faf1);

  }

  .penguin {

    /\* Only change code below this line \*/

    --penguin-belly: white;

    /\* Only change code above this line \*/

    position: relative;

    margin: auto;

    display: block;

    margin-top: 5%;

    width: 300px;

    height: 300px;

  }

1. **Use a media query to change a variable**

CSS Variables can simplify the way you use media queries.

For instance, when your screen is smaller or larger than your media query break point, you can change the value of a variable, and it will apply its style wherever it is used.

In the :root selector of the media query, change it so --penguin-size is redefined and given a value of 200px. Also, redefine --penguin-skin and give it a value of black. Then resize the preview to see this change in action.

<style>

  :root {

    --penguin-size: 300px;

    --penguin-skin: gray;

    --penguin-belly: white;

    --penguin-beak: orange;

  }

  @media (max-width: 350px) {

    :root {

      /\* Only change code below this line \*/

    --penguin-size: 200px;

    --penguin-skin: black;

      /\* Only change code above this line \*/

    }

  }