**CSS notes**

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">

Add Rounded Corners with border-radius

Your photo generally 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 example photo a border-radius of 10px.

For example:

<style> img{ Border-radius:10px;} </style>

Make Circular Images with a border-radius

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

Give your photo a border-radius of 50%. That will make the whole picture as a Circle.

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">

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.

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.

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.

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.

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.

For example:

.blue-box {

    background-color: blue;

    color: #fff;

    padding-top: 40px;

    padding-left: 40px;

    padding-right: 20px;

    padding-bottom: 20px;

  }

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.

For example:

.blue-box {

    background-color: blue;

    color: #fff;

    margin-top: 40px;

    margin-right: 20px;

    margin-bottom: 20px;

    margin-left: 40px;

  }

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.

.blue-box {

    background-color: blue;

    color: #fff;

    padding: 40px 20px 20px 40px;

  }

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.

.blue-box {

    background-color: blue;

    color: #fff;

    margin: 40px 20px 20px 40px;

  }

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;

}

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.

For example:

.red-box {

    background-color: red;

    margin: 20px 40px 20px 40px;

    padding: 1.5em;

  }

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*

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 the id attribute will always take precedence.

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, inline styles look like this:

<h1 style="color: green;">

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

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;

Hex colors

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.

body {

color: #000000;

}

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 |

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 |

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)

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.

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.

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.

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.

Example:

<style>

  :root {

    --red-color: red;

  }

  .red-box {

    background: red;

    background: var(--red-color);

    height: 200px;

    width:200px;

  }

</style>

<div class="red-box"></div>

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.

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.