mdn web docs

Getting started with CSS

In this article, we will take a simple HTML document and apply CSS to it, learning some practical things about the language along the way.

Prerequisites:	Basic software installed, basic knowledge of working with files, and HTML basics (study Introduction to HTML.)
Objective:	To understand the basics of linking a CSS document to an HTML file, and be able to do simple text formatting with CSS.

Starting with some HTML

Our starting point is an HTML document. You can copy the code from below if you want to work on your own computer. Save the code below as `index.html` in a folder on your machine.

HTML

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Getting started with CSS</title>
  </head>

  <body>
    <h1>I am a level one heading</h1>

    <p>
      This is a paragraph of text. In the text is a
      <span>span element</span> and also a
      <a href="https://example.com">link</a>.
    </p>

    <p>
```

```
      This is the second paragraph. It contains an <em>emphasized</em> element.
    </p>

    <ul>
      <li>Item <span>one</span></li>
      <li>Item two</li>
      <li>Item <em>three</em></li>
    </ul>
  </body>
</html>
```

> **Note:** If you are reading this on a device or an environment where you can't easily create files, then don't worry — live code editors are provided below to allow you to write example code right here in the page.

Adding CSS to our document

The very first thing we need to do is to tell the HTML document that we have some CSS rules we want it to use. There are three different ways to apply CSS to an HTML document that you'll commonly come across, however, for now, we will look at the most usual and useful way of doing so — linking CSS from the head of your document.

Create a file in the same folder as your HTML document and save it as `styles.css`. The `.css` extension shows that this is a CSS file.

To link `styles.css` to `index.html`, add the following line somewhere inside the `<head>` of the HTML document:

HTML

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

This `<link>` element tells the browser that we have a stylesheet, using the `rel` attribute, and the location of that stylesheet as the value of the `href` attribute. You can test that the CSS works by adding a rule to `styles.css`. Using your code editor, add the following to your CSS file:

CSS

```
h1 {
  color: red;
}
```

Save your HTML and CSS files and reload the page in a web browser. The level one heading at the top of the document should now be red. If that happens, congratulations — you have successfully applied some CSS to an HTML document. If that doesn't happen, carefully check that you've typed everything correctly.

You can continue to work in `styles.css` locally, or you can use our interactive editor below to continue with this tutorial. The interactive editor acts as if the CSS in the first panel is linked to the HTML document, just as we have with our document above.

Styling HTML elements

By making our heading red, we have already demonstrated that we can target and style an HTML element. We do this by targeting an *element selector* — this is a selector that directly matches an HTML element name. To target all paragraphs in the document, you would use the selector `p`. To turn all paragraphs green, you would use:

CSS

```
p {
  color: green;
}
```

You can target multiple selectors at the same time by separating the selectors with a comma. If you want all paragraphs and all list items to be green, your rule would look like this:

CSS

```
p,
li {
  color: green;
}
```

Try this out in the interactive editor below (edit the code boxes) or in your local CSS document.

I am a level one heading

This is a paragraph of text. In the text is a span element and also a link.

This is the second paragraph. It contains an *emphasized* element.

- Item one
- Item two
- Item *three*

Interactive editor

```
h1 {

}

p {

}
```

```
<h1>I am a level one heading</h1>

<p>This is a paragraph of text. In the text is a <span>span
element</span>
and also a <a href="http://example.com">link</a>.</p>

<p>This is the second paragraph. It contains an <em>emphasized</em>
element.</p>

<ul>
  <li>Item one</li>
  <li>Item two</li>
  <li>Item <em>three</em></li>
</ul>
```

Reset

Changing the default behavior of elements

When we look at a well-marked up HTML document, even something as simple as our example, we can see how the browser is making the HTML readable by adding some default styling. Headings are large and bold and our list has bullets. This happens because browsers have internal stylesheets containing default styles, which they apply to all pages by default; without them all of the text would run together in a clump and we would have to style everything from scratch. All modern browsers display HTML content by default in pretty much the same way.

However, you will often want something other than the choice the browser has made. This can be done by choosing the HTML element that you want to change and using a CSS rule to change the way it looks. A good example is `<ul>`, an unordered list. It has list bullets. If you don't want those bullets, you can remove them like so:

CSS

```
li {
  list-style-type: none;
}
```

Try adding this to your CSS now.

The `list-style-type` property is a good property to look at on MDN to see which values are supported. Take a look at the page for `list-style-type` and you will find an interactive example at the top of the page to try some different values in, then all allowable values are detailed further down the page.

Looking at that page you will discover that in addition to removing the list bullets, you can change them — try changing them to square bullets by using a value of `square`.

Adding a class

So far, we have styled elements based on their HTML element names. This works as long as you want all of the elements of that type in your document to look the same. To select a subset of the elements without changing the others, you can add a class to your HTML element and target that class in your CSS.

1. In your HTML document, add a class attribute to the second list item. Your list will now look like this:

HTML

```
<ul>
  <li>Item one</li>
  <li class="special">Item two</li>
  <li>Item <em>three</em></li>
</ul>
```

2. In your CSS, you can target the class of `special` by creating a selector that starts with a period. Add the following to your CSS file:

   CSS

   ```
   .special {
     color: orange;
     font-weight: bold;
   }
   ```

3. Save and refresh to see what the result is.

You can apply the class of `special` to any element on your page that you want to have the same look as this list item. For example, you might want the `<span>` in the paragraph to also be orange and bold. Try adding a `class` of `special` to it, then reload your page and see what happens.

Sometimes you will see rules with a selector that lists the HTML element selector along with the class:

CSS

```
li.special {
  color: orange;
  font-weight: bold;
}
```

This syntax means "target any `li` element that has a class of special". If you were to do this, then you would no longer be able to apply the class to a `<span>` or another element by adding the class to it; you would have to add that element to the list of selectors:

CSS

```
li.special,
span.special {
  color: orange;
```

```
  font-weight: bold;
}
```

As you can imagine, some classes might be applied to many elements and you don't want to have to keep editing your CSS every time something new needs to take on that style. Therefore, it is sometimes best to bypass the element and refer to the class, unless you know that you want to create some special rules for one element alone, and perhaps want to make sure they are not applied to other things.

Styling things based on their location in a document

There are times when you will want something to look different based on where it is in the document. There are a number of selectors that can help you here, but for now we will look at just a couple. In our document, there are two `<em>` elements — one inside a paragraph and the other inside a list item. To select only an `<em>` that is nested inside an `<li>` element, you can use a selector called the **descendant combinator**, which takes the form of a space between two other selectors.

Add the following rule to your stylesheet:

CSS

```
li em {
  color: rebeccapurple;
}
```

This selector will select any `<em>` element that is inside (a descendant of) an `<li>`. So in your example document, you should find that the `<em>` in the third list item is now purple, but the one inside the paragraph is unchanged.

Something else you might like to try is styling a paragraph when it comes directly after a heading at the same hierarchy level in the HTML. To do so, place a `+` (an **next-sibling combinator**) between the selectors.

Try adding this rule to your stylesheet as well:

CSS

```
h1 + p {
  font-size: 200%;
}
```

The live example below includes the two rules above. Try adding a rule to make a span red if it is inside a paragraph. You will know if you have it right because the span in the first paragraph will be red, but the one in the first list item will not change color.

I am a level one heading

This is a paragraph of text. In the text is a span element and also a link.

This is the second paragraph. It contains an *emphasized* element.

- Item one
- Item two
- Item *three*

Interactive editor

```
li em {
  color: rebeccapurple;
}

h1 + p {
  font-size: 200%;
}
```

```
<h1>I am a level one heading</h1>

<p>This is a paragraph of text. In the text is a <span>span
element</span>
and also a <a href="http://example.com">link</a>.</p>

<p>This is the second paragraph. It contains an <em>emphasized</em>
element.</p>

<ul>
  <li>Item <span>one</span></li>
  <li>Item two</li>
  <li>Item <em>three</em></li>
</ul>
```

Reset

> **Note:** As you can see, CSS gives us several ways to target elements, and we've only scratched the surface so far! We will be taking a proper look at all of these selectors and many more in our Selectors articles later on in the course.

Styling things based on state

The final type of styling we shall take a look at in this tutorial is the ability to style things based on their state. A straightforward example of this is when styling links. When we style a link, we need to target the `<a>` (anchor) element. This has different states depending on whether it is unvisited, visited, being hovered over, focused via the keyboard, or in the process of being clicked (activated). You can use CSS to target these different states — the CSS below styles unvisited links pink and visited links green.

CSS

```
a:link {
  color: pink;
}

a:visited {
  color: green;
}
```

You can change the way the link looks when the user hovers over it, for example by removing the underline, which is achieved by the next rule:

CSS

```
a:hover {
  text-decoration: none;
}
```

In the live example below, you can play with different values for the various states of a link. We have added the rules above to it, and now realize that the pink color is quite light and hard to read — why not change that to a better color? Can you make the links bold?

I am a level one heading

This is a paragraph of text. In the text is a span element and also a link.

This is the second paragraph. It contains an *emphasized* element.

- Item one
- Item two
- Item *three*

Interactive editor

```
a:link {
  color: pink;
}

a:visited {
  color: green;
}

a:hover {
  text-decoration: none;
}
```

```
<h1>I am a level one heading</h1>

<p>This is a paragraph of text. In the text is a <span>span
element</span>
and also a <a href="http://example.com">link</a>.</p>

<p>This is the second paragraph. It contains an <em>emphasized</em>
element.</p>

<ul>
  <li>Item one</li>
  <li>Item two</li>
  <li>Item <em>three</em></li>
</ul>
```

Reset

We have removed the underline on our link on hover. You could remove the underline from all states of a link. It is worth remembering however that in a real site, you want to ensure that visitors know that a link is a link. Leaving the underline in place can be an important clue for people to realize that some text inside a paragraph can be clicked on — this is the behavior they are used to. As with everything in CSS, there is the potential to make the document less accessible with your changes — we will aim to highlight potential pitfalls in appropriate places.

> **Note:** you will often see mention of accessibility in these lessons and across MDN. When we talk about accessibility we are referring to the requirement for our webpages to be understandable and usable by everyone.
>
> Your visitor may well be on a computer with a mouse or trackpad, or a phone with a touchscreen. Or they might be using a screen reader, which reads out the content of the document, or they may need to use much larger text, or be navigating the site using the keyboard only.
>
> A plain HTML document is generally accessible to everyone — as you start to style that document it is important that you don't make it less accessible.

Combining selectors and combinators

It is worth noting that you can combine multiple selectors and combinators together. For example:

CSS

```
/* selects any <span> that is inside a <p>, which is inside an <article>  */
article p span {
}

/* selects any <p> that comes directly after a <ul>, which comes directly after an
<h1>  */
h1 + ul + p {
}
```

You can combine multiple types together, too. Try adding the following into your code:

CSS

```
body h1 + p .special {
  color: yellow;
  background-color: black;
  padding: 5px;
}
```

This will style any element with a class of `special`, which is inside a `<p>`, which comes just after an `<h1>`, which is inside a `<body>`. Phew!

In the original HTML we provided, the only element styled is `<span class="special">`.

Don't worry if this seems complicated at the moment — you'll soon start to get the hang of it as you write more CSS.

Summary

In this article, we have taken a look at a number of ways in which you can style a document using CSS. We will be developing this knowledge as we move through the rest of the lessons. However, you now already know enough to style text, apply CSS based on different ways of targeting elements in the document, and look up properties and values in the MDN documentation.

In the next lesson, we'll be taking a look at how CSS is structured.