Sign in

English ▾

CSS first steps

CSS (Cascading Style Sheets) is used to style and lay out web pages — for example, to alter the font, color, size, and spacing of your content, split it into multiple columns, or add animations and other decorative features. This module provides a gentle beginning to your path towards CSS mastery with the basics of how it works, what the syntax looks like, and how you can start using it to add styling to HTML.

Prerequisites

Before starting this module, you should have:

1. Basic familiarity with using computers, and using the Web passively (i.e. looking at it, consuming the content.)
2. A basic work environment set up as detailed in Installing basic software, and an understanding of how to create and manage files, as detailed in Dealing with files.
3. Basic familiarity with HTML, as discussed in the Introduction to HTML module.

Note: If you are working on a computer/tablet/other device where you don't have the ability to create your own files, you could try out (most of) the code examples in an online coding program such as JSBin or Glitch.

Guides

This module contains the following articles, which will take you through all the basic theory of CSS, and provide opportunities for you to test out some skills.

What is CSS?

CSS (Cascading Style Sheets) allows you to create great-looking web pages, but how does it work under the hood? This article explains what CSS is, with a simple syntax example, and also covers some key terms about the language.

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.

How CSS is structured

Now that you have an idea about what CSS is and the basics of using it, it is time to look a little deeper into the structure of the language itself. We have already met many of the concepts discussed here; you can return to this one to recap if you find any later concepts confusing.

How CSS works

We have learned the basics of CSS, what it is for and how to write simple stylesheets. In this lesson we will take a look at how a browser takes CSS and HTML and turns that into a webpage.

Using your new knowledge

With the things you have learned in the last few lessons you should find that you can format simple text documents using CSS, to add your own style to them. This article gives you a chance to do that.

See also

Intermediate Web Literacy 1: Intro to CSS

An excellent Mozilla foundation course that explores and tests a lot of the skills talked about in the *Introduction to CSS* module. Learn about styling HTML elements on a webpage, CSS selectors, attributes, and values.

MDN web docs moz://a

Sign in

English ▾

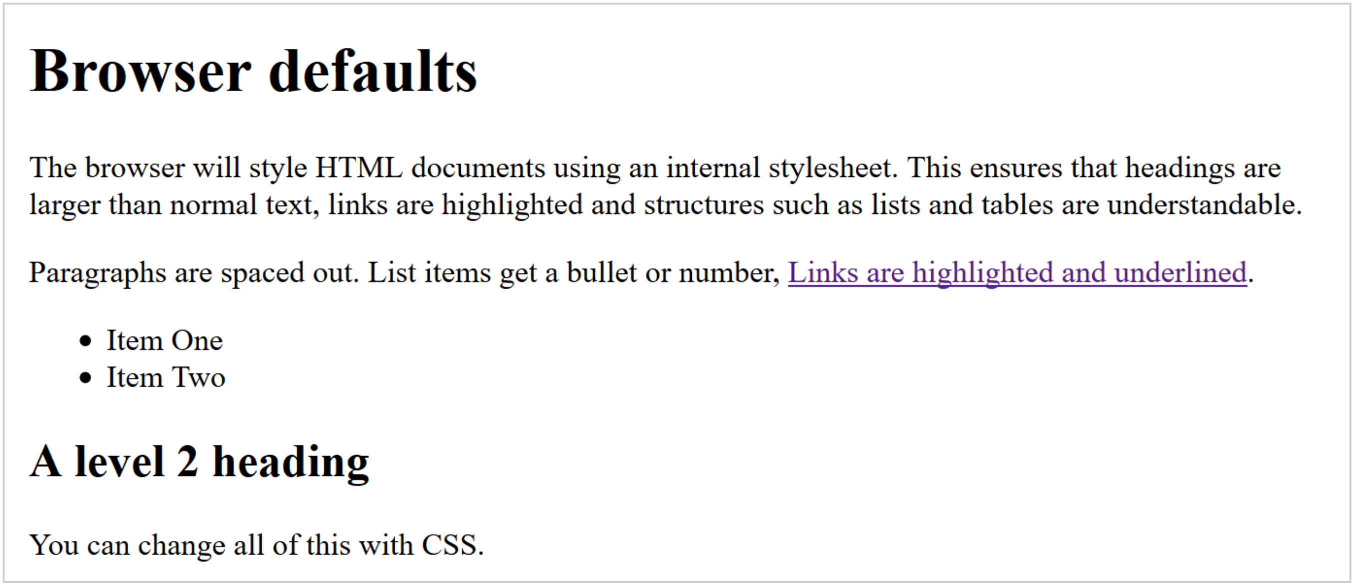
What is CSS?

Overview: First steps Next

CSS (Cascading Style Sheets) allows you to create great-looking web pages, but how does it work under the hood? This article explains what CSS is, with a simple syntax example, and also covers some key terms about the language.

Prerequisites:	Basic computer literacy, basic software installed, basic knowledge of working with files, and HTML basics (study Introduction to HTML.)
Objective:	To learn what CSS is.

In the Introduction to HTML module we covered what HTML is, and how it is used to mark up documents. These documents will be readable in a web browser. Headings will look larger than regular text, paragraphs break onto a new line and have space between them. Links are colored and underlined to distinguish them from the rest of the text. What you are seeing is the browser's default styles — very basic styles that the browser applies to HTML to make sure it will be basically readable even if no explicit styling is specified by the author of the page.



Browser defaults

The browser will style HTML documents using an internal stylesheet. This ensures that headings are larger than normal text, links are highlighted and structures such as lists and tables are understandable.

Paragraphs are spaced out. List items get a bullet or number, Links are highlighted and underlined.

- Item One
- Item Two

A level 2 heading

You can change all of this with CSS.

However, the web would be a boring place if all websites looked like that. Using CSS you can control exactly how HTML elements look in the browser, presenting your markup using whatever design you like.

What is CSS for?

As we have mentioned before, CSS is a language for specifying how documents are presented to users — how they are styled, laid out, etc.

A **document** is usually a text file structured using a markup language — HTML is the most common markup language, but you may also come across other markup languages such as SVG or XML.

Presenting a document to a user means converting it into a form usable by your audience. Browsers, like Firefox, Chrome, or Edge , are designed to present documents visually, for example, on a computer screen, projector or printer.

> **Note**: A browser is sometimes called a user agent, which basically means a computer program that represents a person inside a computer system. Browsers are the main type of user agent we think of when talking about CSS, however, it is not the only one. There are other user agents available — such as those which convert HTML and CSS documents into PDFs to be printed.

CSS can be used for very basic document text styling — for example changing the color and size of headings and links. It can be used to create layout — for example turning a single column of text into a layout with a main content area and a sidebar for related information. It can even be used for effects such as animation. Have a look at the links in this paragraph for specific examples.

CSS syntax

CSS is a rule-based language — you define rules specifying groups of styles that should be applied to particular elements or groups of elements on your web page. For example "I want the main heading on my page to be shown as large red text."

The following code shows a very simple CSS rule that would achieve the styling described above:

```css
h1 {
    color: red;
    font-size: 5em;
}
```

The rule opens with a selector . This *selects* the HTML element that we are going to style. In this case we are styling level one headings (`<h1>`).

We then have a set of curly braces `{ }`. Inside those will be one or more **declarations**, which take the form of **property** and **value** pairs. Each pair specifies a property of the element(s) we are selecting, then a value that we'd like to give the property.

Before the colon, we have the property, and after the colon, the value. CSS properties have different allowable values, depending on which property is being specified. In our example, we have the `color` property, which can take various color values. We also have the `font-size` property. This property can take various size units as a value.

A CSS stylesheet will contain many such rules, written one after the other.

```
h1 {
    color: red;
    font-size: 5em;
}

p {
    color: black;
}
```

You will find that you quickly learn some values, whereas others you will need to look up. The individual property pages on MDN give you a quick way to look up properties and their values when you forget, or want to know what else you can use as a value.

Note: You can find links to all the CSS property pages (along with other CSS features) listed on the MDN CSS reference. Alternatively, you should get used to searching for "mdn *css-feature-name*" in your favourite search engine whenever you need to find out more information about a CSS feature. For example, try searching for "mdn color" and "mdn font-size"!

CSS Modules

As there are so many things that you could style using CSS, the language is broken down into *modules*. You'll see reference to these modules as you explore MDN and many of the documentation pages are organized around a particular module. For example, you could take a look at the MDN reference to the Backgrounds and Borders module to find out what its purpose is, and what different properties and other features it contains. You will also find links to the *CSS Specification* that defines the technology (see below).

At this stage you don't need to worry too much about how CSS is structured, however it can make it easier to find information if, for example, you are aware that a certain property is likely to be found among other similar things and are therefore probably in the same specification.

For a specific example, let's go back to the Backgrounds and Borders module — you might think that it makes logical sense for the `background-color` and `border-color` properties to be defined in this module. And you'd be right.

CSS Specifications

All web standards technologies (HTML, CSS, JavaScript, etc.) are defined in giant documents called specifications (or simply "specs"), which are published by standards organizations (such as the W3C, WHATWG, ECMA, or Khronos) and define precisely how those technologies are supposed to behave.

CSS is no different — it is developed by a group within the W3C called the CSS Working Group. This group is made of representatives of browser vendors and other companies who have an interest in CSS. There are also other people, known as *invited experts*, who act as independent voices; they are not linked to a member organization.

New CSS features are developed, or specified, by the CSS Working Group. Sometimes because a particular browser is interested in having some capability, other times because web designers and developers are asking for a feature, and sometimes because the Working Group itself has identified a requirement. CSS is constantly developing, with new features coming available. However, a key thing about CSS is that everyone works very hard to never change things in a way that would break old websites. A website built in 2000, using the limited CSS available then, should still be usable in a browser today!

As a newcomer to CSS, it is likely that you will find the CSS specs overwhelming — they are intended for engineers to use to implement support for the features in user agents, not for web developers to read to understand CSS. Many experienced developers would much rather refer to MDN documentation or other tutorials. It is however worth knowing that they exist, understanding the relationship between the CSS you are using, browser support (see below), and the specs.

Browser support

Once CSS has been specified then it is only useful for us in developing web pages if one or more browsers have implemented it. This means that the code has been written to turn the instruction in our CSS file into something that can be output to the screen. We'll look at this process more in the lesson How CSS works. It is unusual for all browsers to implement a feature at the same time, and so there is usually a gap where you can use some part of CSS in some browsers and not in others. For this reason, being able to check implementation status is useful. On each property page on MDN you can see the status of the property you are interested in, so you can tell if you will be able to use it on a website.

What follows is the compat data chart for the CSS `font-family` property.

Update compatibility data on GitHub

`font-family`

Chrome	1
Edge	12
Firefox	1
IE	3
Opera	3.5
Safari	1
WebView Android	1
Chrome Android	18
Firefox Android	4
Opera Android	10.1
Safari iOS	1
Samsung Internet Android	1.0

`system-ui`

Chrome	56
Edge	79
Firefox	No
IE	No
Opera	43
Safari	9
WebView Android	56

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 computer literacy, 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.

```
<!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</
and also a <a href="http://example.com">link</a>.</p>

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

    <ul>
        <li>Item one</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

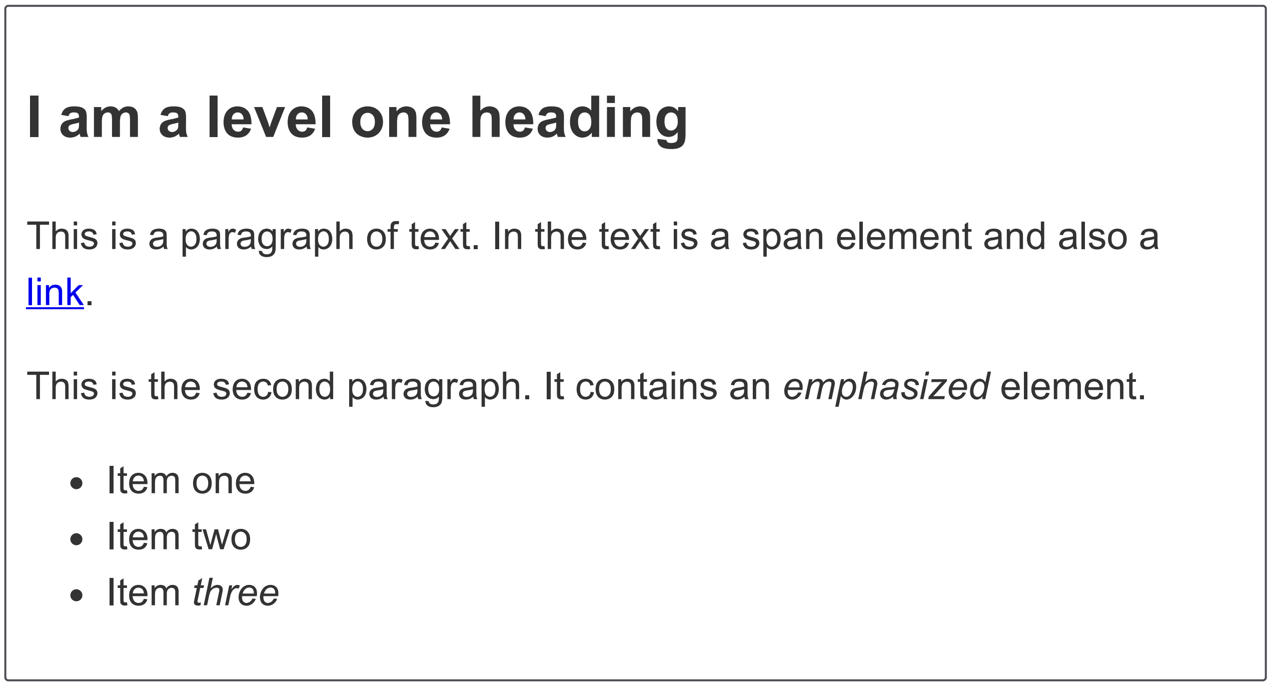
By making our heading red we have already demonstrated that we can target and style an HTML element. We do this by targeting an *element selector* — this is a selector that directly matches an HTML element name. To target all paragraphs in the document you would use the selector `p`. To turn all paragraphs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

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

You can target multiple selectors at once, by separating the selectors with a comma. If I want all paragraphs and all list items to be green my rule looks 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*

```css
h1 {

}

p {

}
```

```html
<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 simply choosing the HTML element that you want to change, and using a CSS rule to change the way it looks. A good example is our `<ul>`, an unordered list. It has list bullets, and if I decide I don't want those bullets I 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. Most of the time that isn't the case and so you will need to find a way to select a subset of the elements without changing the others. The most common way to do this is to add a class to your HTML element and target that class.

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

```
<ul>
  <li>Item one</li>

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

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

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

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:

```
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 simply 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 simply 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 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 I can use a selector called the **descendant combinator**, which simply takes the form of a space between two other selectors.

Add the following rule to your stylesheet.

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

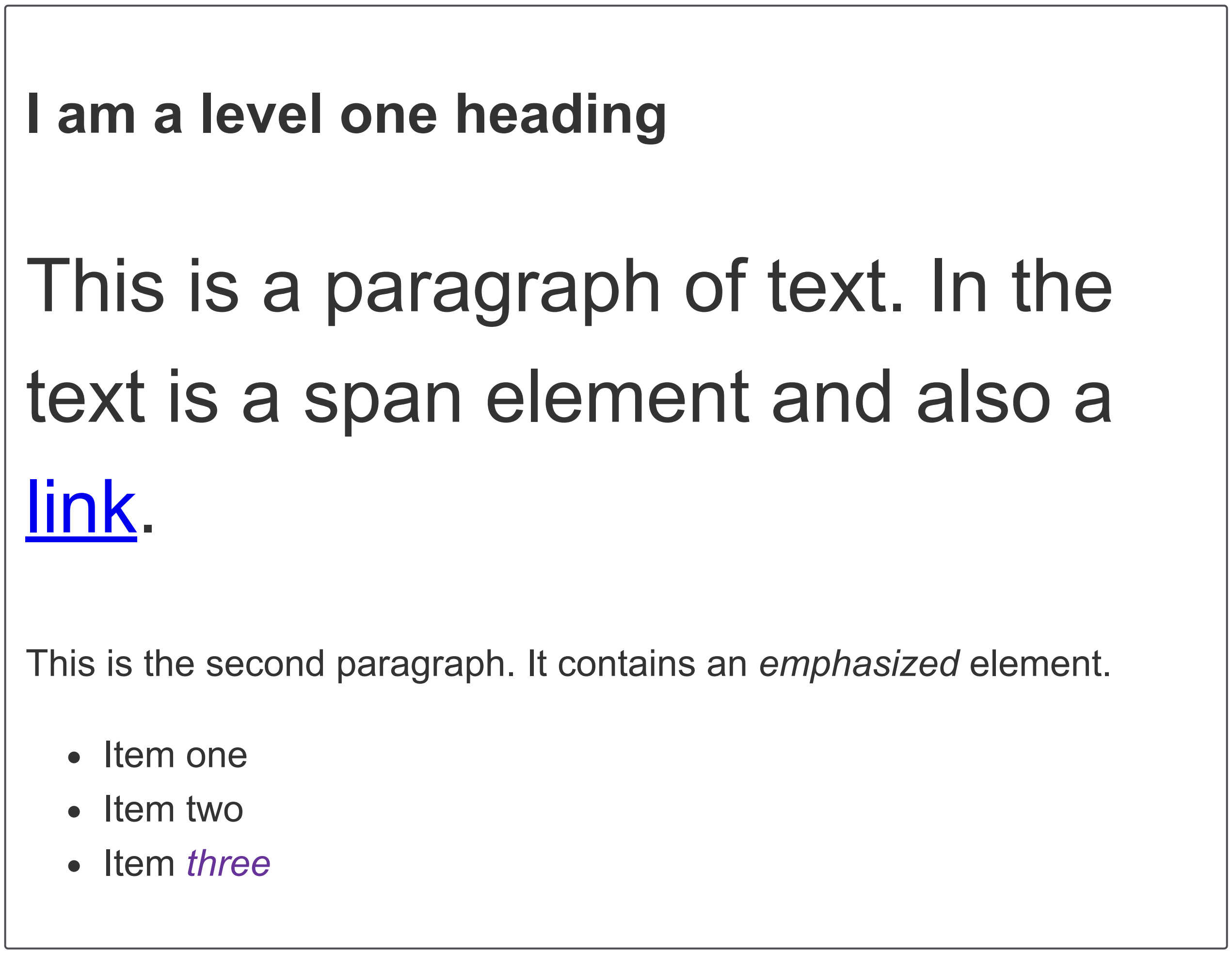
This selector will select any `<em>` element that is inside (a descendant of) an `<li>`. So in your example document, you should find that the `<em>` in the third list item is now purple, but the one inside the paragraph is unchanged.

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 **adjacent sibling combinator**) between the selectors.

Try adding this rule to your stylesheet as well:

```
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 as 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*

```
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.

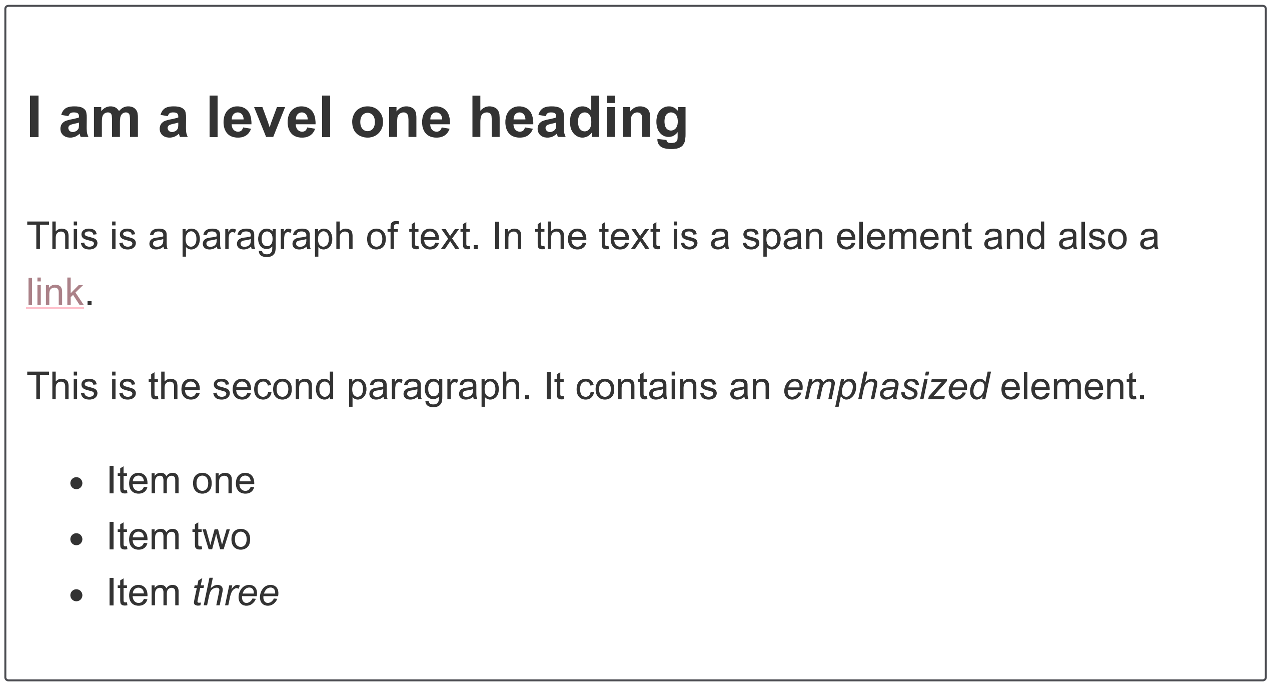
```css
a:link {
  color: pink;
}

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

You can change the way the link looks when the user hovers over it, for example removing the underline, which is achieved by in 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. I have added the rules above to it, and now realise 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*

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

a:visited {
    color: green;
}

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

```html
<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 screenreader, 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:

```
/* 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 directl
h1 + ul + p { ... }
```

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

```
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.

Wrapping up

In this tutorial, 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 will be taking a look at how CSS is structured.

Previous Overview: First steps Next

In this module

Sign in

English ▾

How CSS is structured

Previous Overview: First steps Next

Now that you are beginning to understanding the purpose and use of CSS, let's examine the structure of CSS.

Prerequisites:	Basic computer literacy, basic software installed, basic knowledge of working with files, HTML basics (study Introduction to HTML), and an idea of How CSS works.
Objective:	To learn CSS's fundamental syntax structures in detail.

Applying CSS to HTML

First, let's examine three methods of applying CSS to a document: with an external stylesheet, with an internal stylesheet, and with inline styles.

External stylesheet

An external stylesheet contains CSS in a separate file with a `.css` extension. This is the most common and useful method of bringing CSS to a document. You can link a single CSS file to multiple web pages, styling all of them with the same CSS stylesheet. In the Getting started with CSS, we linked an external stylesheet to our web page.

You reference an external CSS stylesheet from an HTML `<link>` element:

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```

The CSS stylesheet file might look like this:

```css
h1 {
  color: blue;
  background-color: yellow;
  border: 1px solid black;
}

p {
  color: red;
}
```

The `href` attribute of the `<link>` element needs to reference a file on your file system. In the example above, the CSS file is in the same folder as the HTML document, but you could place it somewhere else and adjust the path. Here are three examples:

```html
<!-- Inside a subdirectory called styles inside the current directory --:
<link rel="stylesheet" href="styles/style.css">

<!-- Inside a subdirectory called general, which is in a subdirectory cal
<link rel="stylesheet" href="styles/general/style.css">

```

```
<!-- Go up one directory level, then inside a subdirectory called styles
<link rel="stylesheet" href="../styles/style.css">
```

Internal stylesheet

An internal stylesheet resides within an HTML document. To create an internal stylesheet, you place CSS inside a `<style>` element contained inside the HTML `<head>`.

The HTML for an internal stylesheet might look like this:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
    <style>
      h1 {
        color: blue;
        background-color: yellow;
        border: 1px solid black;
      }

      p {
        color: red;
      }
    </style>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```

In some circumstances, internal stylesheets can be useful. For example, perhaps you're working with a content management system where you are blocked from modifying external CSS files.

But for sites with more than one page, an internal stylesheet becomes a less efficient way of working. To apply uniform CSS styling to multiple pages using internal stylesheets, you must have an internal stylesheet in every web page that will use the styling. The efficiency penalty carries over to site maintenance too. With CSS in internal stylesheets, there is the risk that even one simple styling change may require edits to multiple web pages.

Inline styles

Inline styles are CSS declarations that affect a single HTML element, contained within a `style` attribute. The implementation of an inline style in an HTML document might look like this:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
  </head>
  <body>
    <h1 style="color: blue;background-color: yellow;border: 1px solid bla
    <p style="color:red;">This is my first CSS example</p>
  </body>
</html>
```

Avoid using CSS in this way when possible. It is the opposite of a best practice. First, it is the least efficient implementation of CSS for maintenance. One styling change might require multiple edits within in a single web page. Second, inline CSS also mixes (CSS) presentational code with HTML and content, making everything more difficult to read and understand. Separating code and content makes maintenance easier for all who work on the the website.

There are a few circumstances where inline styles are more common. You might have to resort to using inline styles if your working environment is very restrictive. For example, perhaps your CMS only allows you to edit the HTML body. You may also see a lot of inline styles in HTML email to achieve compatibility with as many email clients as possible.

Playing with the CSS in this article

For the exercise that follows, create a folder on your computer. You can name the folder however you like. Inside the folder, copy the text below to create two files:

index.html:

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My CSS experiments</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>

    <p>Create your test HTML here</p>

  </body>
</html>
```

styles.css:

```css
/* Create your test CSS here */

p {
  color: red;
}
```

When you find CSS that you want to experiment with, replace the HTML `<body>` contents with some HTML to style, and then add your test CSS code to your CSS file.

As an altenative, you can also use the interactive editor below.

Create your test HTML here

```css
/* Create your test CSS here */

p {
  color: red;
}
```

```html
<p>Create your test HTML here</p>
```

Reset

Read on and have fun!

Selectors

A selector targets HTML to apply styles to content. We have already discovered a variety of selectors in the Getting started with CSS tutorial. If CSS is not applying to content as-expected, your selector may not match the way you think it should match.

Each CSS rule starts with a selector—or a list of selectors—in order to tell the browser which element or elements the rules should apply to. All the examples below are valid selectors, or lists of selectors.

```
h1
a:link
.manythings
#onething
*
.box p
.box p:first-child
h1, h2, .intro
```

Try creating some CSS rules that use the selectors above. Add HTML to be styled by the selectors. If any of the syntax above is not familiar, try searching MDN.

Note: You will learn more about selectors in the next module: CSS selectors.

Specificity

You may encounter scenarios where two selectors select the same HTML element. Consider the stylesheet below, with a `p` selector that sets paragraph text to blue. However, there is also a class that sets the text of selected elements to red.

```
.special {
  color: red;
}

p {
  color: blue;
}
```

Suppose that in our HTML document, we have a paragraph with a class of `special`. Both rules apply. Which selector prevails? Do you expect to see blue or red paragraph text?

```
<p class="special">What color am I?</p>
```

The CSS language has rules to control which selector is stronger in the event of a conflict. These rules are called **cascade** and **specificity**. In the code block below, we define two rules for the `p` selector, but the paragraph text will be blue. This is because the declaration that sets the paragraph text to blue appears later in the stylesheet. Later styles replace conflicting styles that appear earlier in the stylesheet. This is the **cascade** rule.

```
p {
  color: red;
}

p {
  color: blue;
}
```

However, in the case of our earlier example with the conflict between the class selector and the element selector, the class prevails, rendering red paragraph text. How can this happen even though a conflicting style appears later in the stylesheet? A class is rated as being more specific, as in having more **specificity** than the element selector, so it cancels the other conflicting style declaration.

Try this experiment for yourself! Add HTML, then add the two `p { ... }` rules to your stylesheet. Next, change the first `p` selector to `.special` to see how it changes the styling.

The rules of specificity and the cascade can seem complicated at first. These rules are easier to understand as you become more familiar with CSS. The Cascade and inheritance section in the next module explains this in detail, including how to calculate specificity.

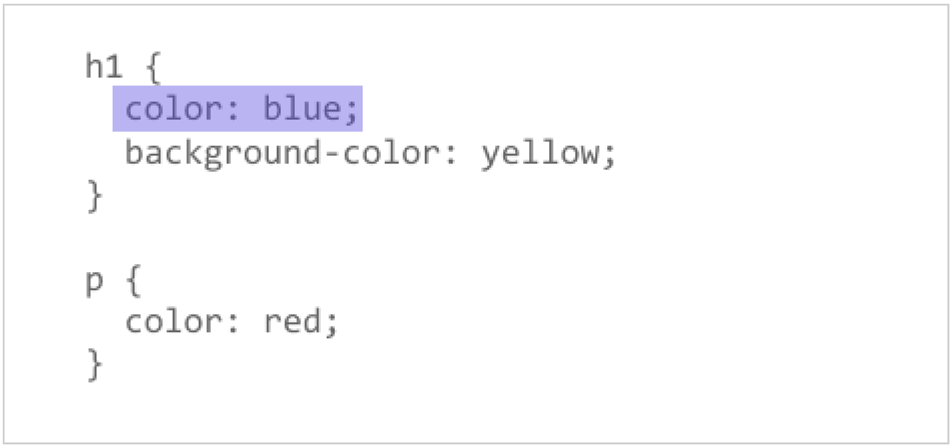
For now, remember that specificity exists. Sometimes, CSS might not apply as you expected because something else in the stylesheet has more specificity. Recognizing that more than one rule could apply to an element, is the first step in fixing these kinds of issues.

Properties and values

At its most basic level, CSS consists of two components:

- **Properties**: These are human-readable identifiers that indicate which stylistic features you want to modify. For example, `font-size`, `width`, `background-color`.
- **Values**: Each property is assigned a value. This value indicates how to style the property.

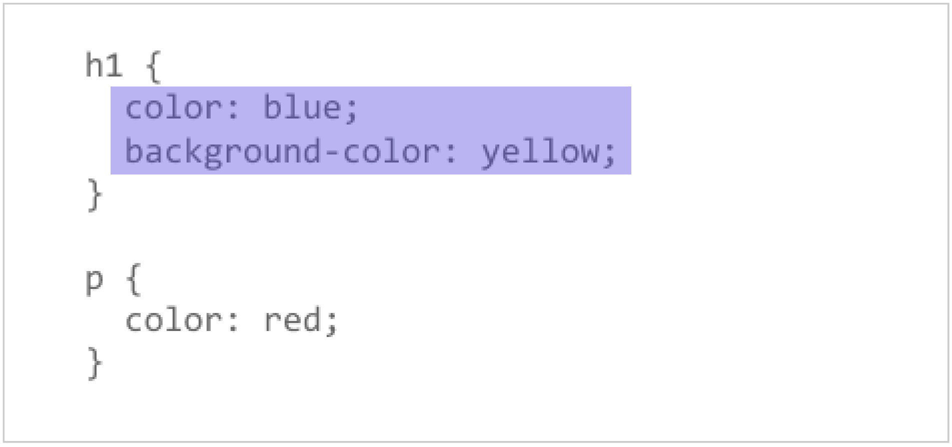
The example below highlights a single property and value. The property name is `color` and the value is `blue`.



```
h1 {
  color: blue;
  background-color: yellow;
}

p {
  color: red;
}
```

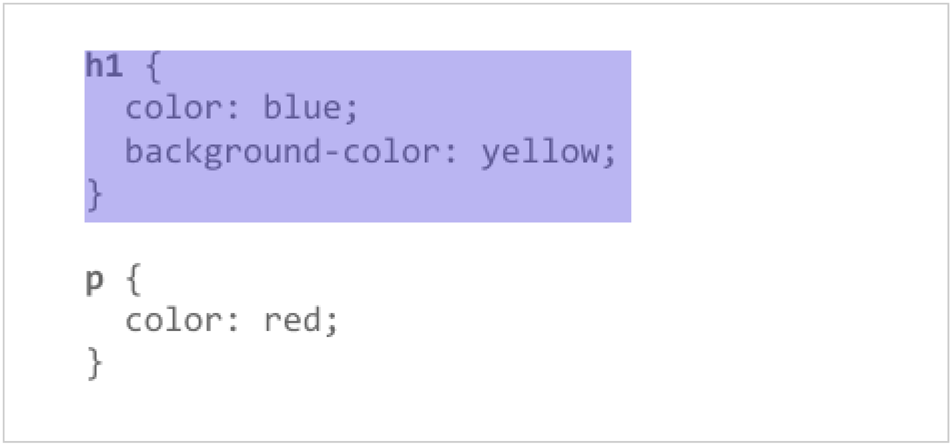
When a property is paired with a value, this pairing is called a *CSS declaration*. CSS declarations are found within *CSS Declaration Blocks*. In the example below, highlighting identifies the CSS declaration block..



```
h1 {
  color: blue;
  background-color: yellow;
}

p {
  color: red;
}
```

Finally, CSS declaration blocks are paired with *selectors* to produce *CSS rulesets* (or *CSS rules*). The example below contains two rules: one for the `h1` selector and one for the `p` selector. The colored highlighting identifies the `h1` rule.



```
h1 {
  color: blue;
  background-color: yellow;
}

p {
  color: red;
}
```

Setting CSS properties to specific values is the primary way of defining layout and styling for a document. The CSS engine calculates which declarations apply to every single element of a page.

> **Important:** CSS properties and values are case-sensitive. The property and value in each pair is separated by a colon. (`:`)

Look up different values of properties listed below. Write CSS rules that apply styling to different HTML elements:

- **`font-size`**
- **`width`**
- **`background-color`**
- **`color`**
- **`border`**

> **Important**: If a property is unknown, or if a value is not valid for a given property, the declaration is processed as *invalid*. It is completely ignored by the browser's CSS engine.

> **Important**: In CSS (and other web standards), it has been agreed that US spelling is the standard where there is language variation or uncertainty. For example, `color` should be spelled `color`, as `colour` will not work.

Functions

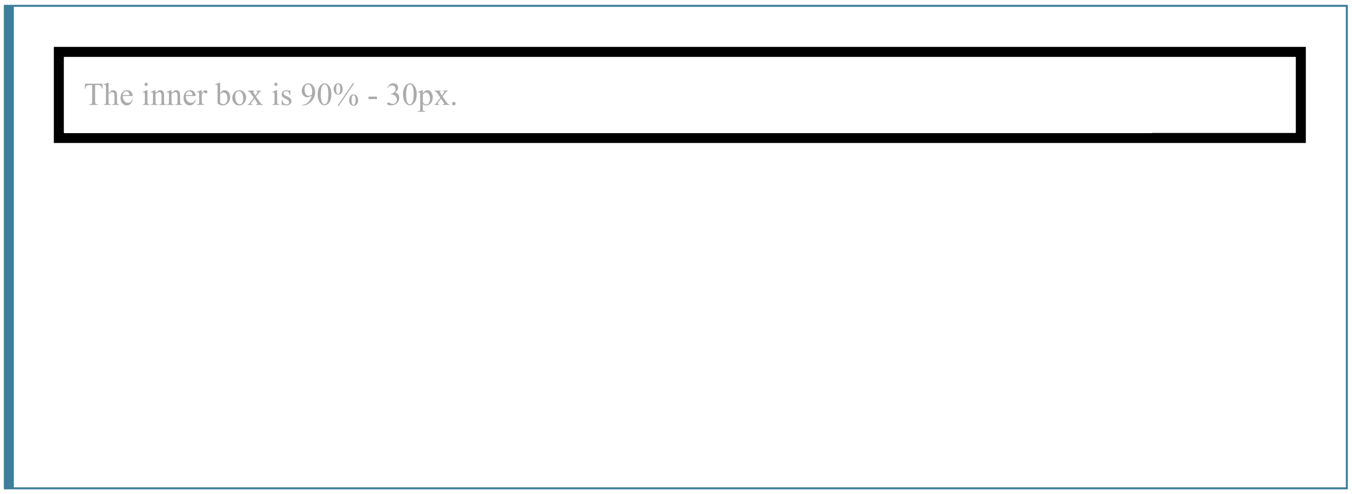
While most values are relatively simple keywords or numeric values, there are some values which take the form of a function. An example would be the `calc()` function, which can do simple math within CSS:

```html
<div class="outer"><div class="box">The inner box is 90% - 30px.</div></
```

```css
.outer {
  border: 5px solid black;
}

.box {
  padding: 10px;
  width: calc(90% - 30px);
  background-color: rebeccapurple;
  color: white;
}
```

This renders as:


The inner box is 90% - 30px.

A function consists of the function name, and brackets to enclose the values for the function. In the case of the `calc()` example above, the values define the width of this box to be 90% of the containing block width, minus 30 pixels. The result of the calculation isn't something that can be computed in advance and entered as a static value.

Another example would be the various values for `transform`, such as `rotate()`.

```html
<div class="box"></div>
```

```css
.box {
  margin: 30px;
  width: 100px;
  height: 100px;
  background-color: rebeccapurple;
  transform: rotate(0.8turn)
}
```

The output from the above code looks like this:

Look up different values of properties listed below. Write CSS rules that apply styling to different HTML elements:

- **`transform`**
- **`background-image`, in particular gradient values**
- **`color`, in particular rgb/rgba/hsl/hsla values**

@rules

CSS `@rules` (pronounced "at-rules") provide instruction for what CSS should perform or how it should behave. Some `@rules` are simple with just a keyword and a value. For example, `@import` imports a stylesheet into another CSS stylesheet:

```
@import 'styles2.css';
```

One common `@rule` that you are likely to encounter is `@media`, which is used to create media queries. Media queries use conditional logic for applying CSS styling.

In the example below, the stylesheet defines a default pink background for the `<body>` element. However, a media query follows that defines a blue background if the browser viewport is wider than 30em.

```
body {
  background-color: pink;
}

@media (min-width: 30em) {
  body {
    background-color: blue;
  }
}
```

You will encounter other `@rules` throughout these tutorials.

See if you can add a media query that changes styles based on the viewport width. Change the width of your browser window to see the result.

Shorthands

Some properties like `font`, `background`, `padding`, `border`, and `margin` are called **shorthand properties.** This is because shorthand properties set several values in a single line.

For example, this one line of code:

```
/* In 4-value shorthands like padding and margin, the values are applied
   in the order top, right, bottom, left (clockwise from the top). There
   shorthand types, for example 2-value shorthands, which set padding/mar
   for top/bottom, then left/right */
padding: 10px 15px 15px 5px;
```

is equivalent to these four lines of code:

```
padding-top: 10px;
padding-right: 15px;
padding-bottom: 15px;
padding-left: 5px;
```

This one line:

```
background: red url(bg-graphic.png) 10px 10px repeat-x fixed;
```

is equivalent to these five lines::

```
background-color: red;
background-image: url(bg-graphic.png);
background-position: 10px 10px;
background-repeat: repeat-x;
background-attachment: fixed;
```

Later in the course, you will encounter many other examples of shorthand properties. MDN's CSS reference is a good resource for more information about any shorthand property.

Try using the declarations (above) in a your own CSS exercise to become more familiar with how it works. You can also experiment with different values.

Warning: One less obvious aspect of using CSS shorthand is how omitted values reset. A value not specified in CSS shorthand reverts to its initial value. This means an omission in CSS shorthand can **override previously set values**.

Comments

As with any coding work, it is best practice to write comments along with CSS. This helps you to remember how the code works as you come back later for fixes or enhancement. It also helps others understand the code.

CSS Comments begin with `/*` and end with `*/`. In the example below, comments mark the start of distinct sections of code. This helps to navigate the codebase as it gets larger. With this kind of commenting in place, searching for comments in your code editor becomes a way to efficiently find a section of code.

```
/* Handle basic element styling */
/* ----------------------------------------------------------------------
body {
  font: 1em/150% Helvetica, Arial, sans-serif;
  padding: 1em;
  margin: 0 auto;
  max-width: 33em;
}

@media (min-width: 70em) {
  /* Let's special case the global font size. On large screen or window,
     we increase the font size for better readability */
  body {
    font-size: 130%;
  }
}

h1 {font-size: 1.5em;}

/* Handle specific elements nested in the DOM  */
/* ----------------------------------------------------------------------
div p, #id:first-line {
  background-color: red;
  border-radius: 3px;
}

div p {
  margin: 0;
  padding: 1em;
}

div p + p {
  padding-top: 0;
}
```

"Commenting out" code is also useful for temporarily disabling sections of code for testing. In the example below, the rules for `.special` are disabled by "commenting out" the code.

```
/*.special {
  color: red;
}*/

p {
  color: blue;
}
```

Add comments to your CSS.

White space

White space means actual spaces, tabs and new lines. Just as browsers ignore white space in HTML, browsers ignore white space inside CSS. The value of white space is how it can improve readability.

In the example below, each declaration (and rule start/end) has its own line. This is arguably a good way to write CSS. It makes it easier to maintain and understand CSS.

```
body {
  font: 1em/150% Helvetica, Arial, sans-serif;
  padding: 1em;
  margin: 0 auto;
  max-width: 33em;
}

@media (min-width: 70em) {
  body {
    font-size: 130%;
  }
}

h1 {
  font-size: 1.5em;
}

div p,
#id:first-line {
  background-color: red;
  border-radius: 3px;
}

div p {
  margin: 0;
  padding: 1em;
}

div p + p {
  padding-top: 0;
}
```

The next example shows the equivalent CSS in a more compressed format. Although the two examples work the same, the one below is more difficult to read.

```
body {font: 1em/150% Helvetica, Arial, sans-serif; padding: 1em; margin:
@media (min-width: 70em) { body {font-size: 130%;} }

h1 {font-size: 1.5em;}

div p, #id:first-line {background-color: red; border-radius: 3px;}
div p {margin: 0; padding: 1em;}
div p + p {padding-top: 0;}
```

For your own projects, you will format your code according to personal preference. For team projects, you may find that a team or project has its own style guide.

> **Important:** Though white space separates values in CSS declarations, **property names never have white space**.

For example, these declarations are valid CSS:

```
margin: 0 auto;
padding-left: 10px;
```

But these declarations are invalid:

```
margin: 0auto;
padding- left: 10px;
```

Do you see the spacing errors? First, `0auto` is not recognized as a valid value for the `margin` property. The entry `0auto` is meant to be two separate values: `0` and `auto`. Second, the browser does not recognize `padding-` as a valid property. The correct property name (`padding-left`) is separated by an errant space.

You should always make sure to separate distinct values from one another by at least one space. Keep property names and property values together as single unbroken strings.

To find out how spacing can break CSS, try playing with spacing inside your test CSS.

What's next?

It's useful to understand how the browser uses HTML and CSS to display a webpage. The next article, How CSS works, explains the process.

Previous Overview: First steps Next

MDN web docs moz://a

Sign in

English ▾

How CSS works

Previous Overview: First steps Next

We have learned the basics of CSS, what it is for and how to write simple stylesheets. In this lesson we will take a look at how a browser takes CSS and HTML and turns that into a webpage.

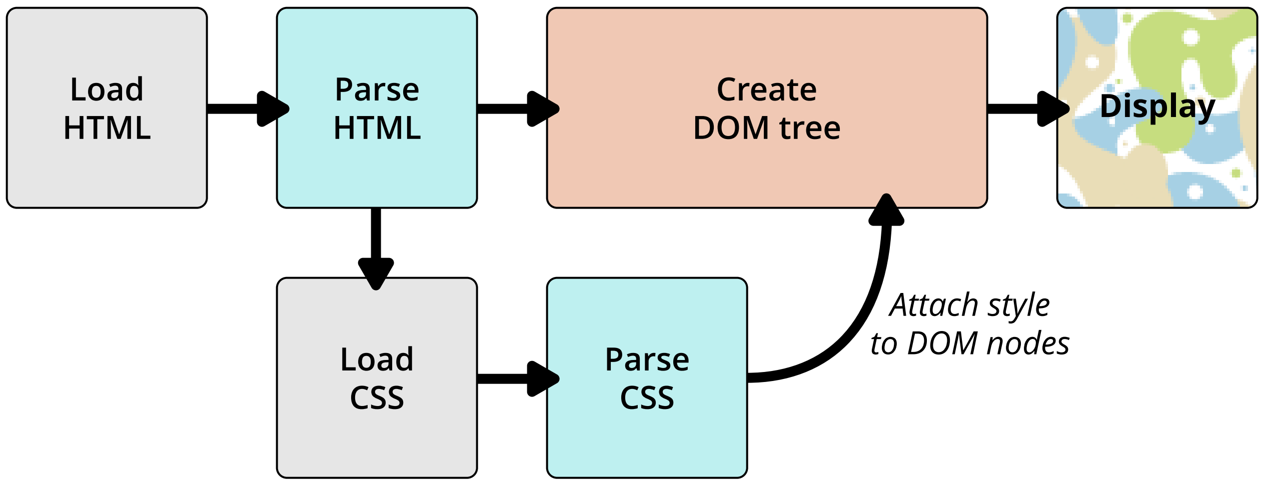
Prerequisites:	Basic computer literacy, basic software installed, basic knowledge of working with files, and HTML basics (study Introduction to HTML.)
Objective:	To understand the basics of how CSS and HTML are parsed by the browser, and what happens when a browser encounters CSS it does not understand.

How does CSS actually work?

When a browser displays a document, it must combine the document's content with its style information. It processes the document in a number of stages, which we've listed below. Bear in mind that this is a very simplified version of what happens when a browser loads a webpage, and that different browsers will handle the process in different ways. But this is roughly what happens.

1. The browser loads the HTML (e.g. receives it from the network).
2. It converts the HTML into a DOM (*Document Object Model*). The DOM represents the document in the computer's memory. The DOM is explained in a bit more detail in the next section.
3. The browser then fetches most of the resources that are linked to by the HTML document, such as embedded images and videos ... and linked CSS! JavaScript is handled a bit later on in the process, and we won't talk about it here to keep things simpler.
4. The browser parses the fetched CSS, and sorts the different rules by their selector types into different "buckets", e.g. element, class, ID, and so on. Based on the selectors it finds, it works out which rules should be applied to which nodes in the DOM, and attaches style to them as required (this intermediate step is called a render tree).
5. The render tree is laid out in the structure it should appear in after the rules have been applied to it.
6. The visual display of the page is shown on the screen (this stage is called painting).

The following diagram also offers a simple view of the process.



About the DOM

A DOM has a tree-like structure. Each element, attribute, and piece of text in the markup language becomes a DOM node in the tree structure. The nodes are defined by their relationship to other DOM nodes. Some elements are parents of child nodes, and child nodes have siblings.

Understanding the DOM helps you design, debug and maintain your CSS because the DOM is where your CSS and the document's content meet up. When you start working with browser DevTools you will be navigating the DOM as you select items in order to see which rules apply.

A real DOM representation

Rather than a long, boring explanation, let's look at an example to see how a real HTML snippet is converted into a DOM.

Take the following HTML code:

```
<p>
  Let's use:
  <span>Cascading</span>
  <span>Style</span>
  <span>Sheets</span>
</p>
```

In the DOM, the node corresponding to our `<p>` element is a parent. Its children are a text node and the three nodes corresponding to our `<span>` elements. The `SPAN` nodes are also parents, with text nodes as their children:

```
P
├─ "Let's use:"
├─ SPAN
|  └─ "Cascading"
├─ SPAN
|  └─ "Style"
└─ SPAN
   └─ "Sheets"
```

This is how a browser interprets the previous HTML snippet —it renders the above DOM tree and then outputs it in the browser like so:

Let's use: Cascading Style Sheets

Applying CSS to the DOM

Let's say we added some CSS to our document, to style it. Again, the HTML is as follows:

```
<p>
  Let's use:
  <span>Cascading</span>
  <span>Style</span>
  <span>Sheets</span>
</p>
```

Let's suppose we apply the following CSS to it:

```
span {
  border: 1px solid black;
  background-color: lime;
}
```

The browser will parse the HTML and create a DOM from it, then parse the CSS. Since the only rule available in the CSS has a `span` selector, the browser will be able to sort the CSS very quickly! It will apply that rule to each one of the three `<span>`s, then paint the final visual representation to the screen.

The updated output is as follows:

Let's use: Cascading Style Sheets

In our Debugging CSS article in the next module we will be using browser DevTools to debug CSS problems, and will learn more about how the browser interprets CSS.

What happens if a browser encounters CSS it doesn't understand?

In an earlier lesson I mentioned that browsers do not all implement new CSS at the same time. In addition, many people are not using the latest version of a browser. Given that CSS is being developed all the time, and is therefore ahead of what browsers can recognise, you might wonder what happens if a browser encounters a CSS selector or declaration it doesn't recognise.

The answer is that it does nothing, and just moves on to the next bit of CSS!

If a browser is parsing your rules, and encounters a property or value that it doesn't understand, it ignores it and moves on to the next declaration. It will do this if you have made an error and misspelled a property or value, or if the property or value is just too new and the browser doesn't yet support it.

Similarly, if a browser encounters a selector that it doesn't understand, it will just ignore the whole rule and move on to the next one.

In the example below I have used the British English spelling for color, which makes that property invalid as it is not recognised. So my paragraph has not been colored blue. All of the other CSS have been applied however; only the invalid line is ignored.

```
<p> I want this text to be large, bold and blue.</p>
```

```
p {
  font-weight: bold;
  colour: blue; /* incorrect spelling of the color property */
  font-size: 200%;
}
```

I want this text to be large, bold and blue.

This behavior is very useful. It means that you can use new CSS as an enhancement, knowing that no error will occur if it is not understood — the browser will either get the new feature or not. Coupled with the way that the cascade works, and the fact that browsers will use the last CSS they come across in a stylesheet when you have two rules with the same specificity you can also offer alternatives for browsers that don't support new CSS.

This works particularly well when you want to use a value that is quite new and not supported everywhere. For example, some older browsers do not support `calc()` as a value. I might give a fallback width for a box in pixels, then go on to give a width with a `calc()` value of `100% - 50px`. Old browsers will use the pixel version, ignoring the line about `calc()` as they don't understand it. New browsers will interpret the line using pixels, but then override it with the line using `calc()` as that line appears later in the cascade.

```
.box {
  width: 500px;
  width: calc(100% - 50px);
}
```

We will look at many more ways to support varying browsers in later lessons.

And finally

You've nearly finished this module; we only have one more thing to do. In the next article you'll use your new knowledge to restyle an example, testing out some CSS in the process.

Previous Overview: First steps Next

In this module

Last modified: Dec 4, 2019, by MDN contributors

Related Topics

Complete beginners start here!

▸ Getting started with the Web

HTML — Structuring the Web

▸ Introduction to HTML

▸ Multimedia and embedding

▸ HTML tables

MDN web docs moz://a

Sign in

English ▾

Using your new knowledge

Previous

Overview: First steps

With the things you have learned in the last few lessons you should find that you can format simple text documents using CSS, to add your own style to them. This assessment gives you a chance to do that.

Prerequisites:	Before attempting this assessment you should have worked through the rest of the CSS basics module, and also have an understanding of HTML basics (study Introduction to HTML).
Objective:	To have a play with some CSS and test your new-found knowledge.

Starting point

You can work in the live editor below, or you can download the starting point to work with in your own editor. This is a single page with the HTML, plus the starting point CSS in the head of the document. If you prefer you could move this CSS to a separate file when you create the example on your local computer. Alternatively use an online tool such as CodePen, jsFiddle, or Glitch to work on the tasks.

Note: If you get stuck, then ask us for help — see the Assessment or further help section at the bottom of this page.

Working with CSS

The following live example shows a biography, which has been styled using CSS. The CSS properties that I have used are as follows — each one links to its property page on MDN, which will give you more examples of its use.

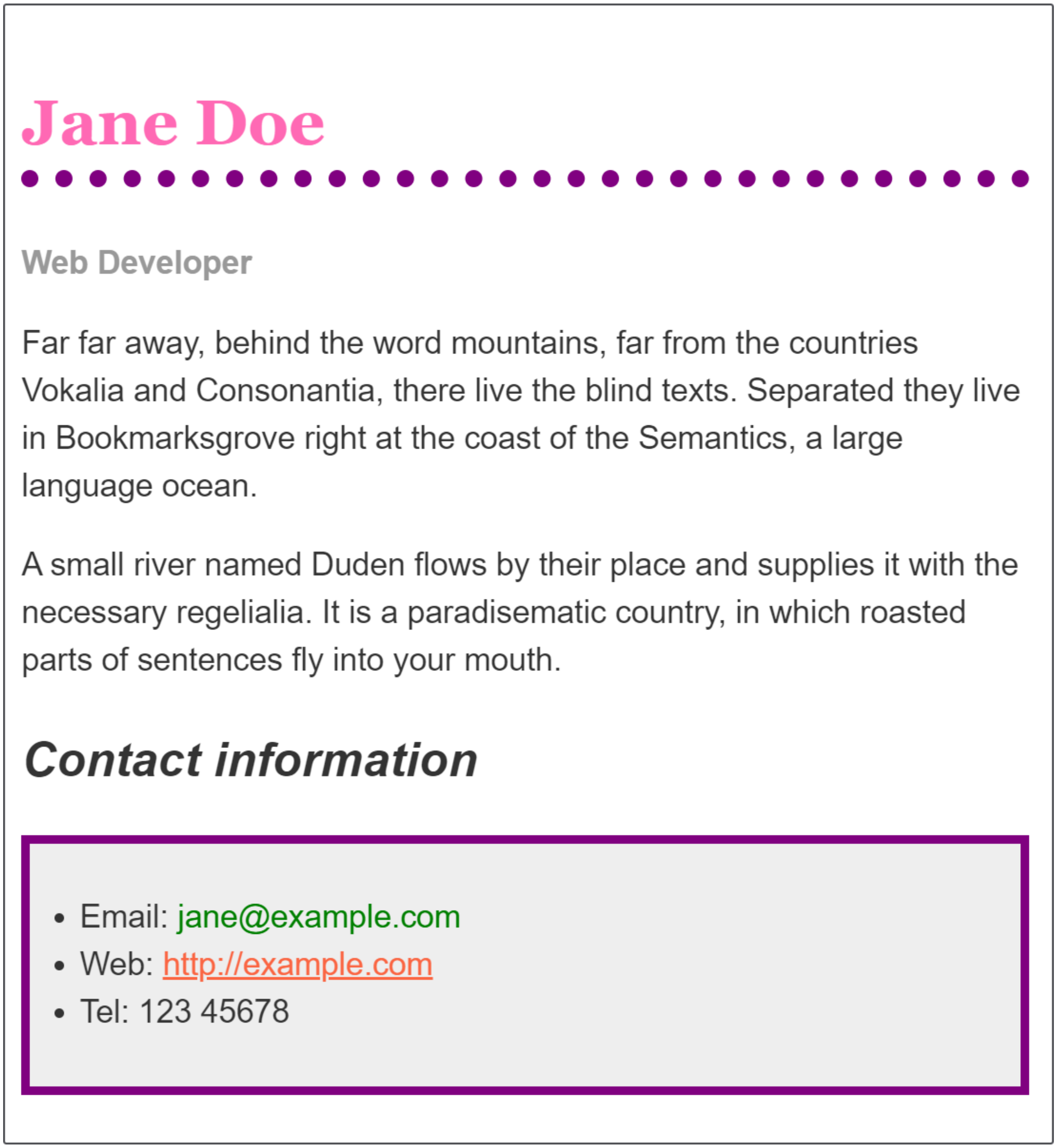
- `font-family`
- `color`
- `border-bottom`
- `font-weight`
- `font-size`
- `text-decoration`

I have used a mixture of selectors, styling elements such as h1 and h2, but also creating a class for the job title and styling that.

Use CSS to change how this biography looks by changing the values of the properties I have used.

1. Make the level one heading pink, using the CSS color keyword `hotpink`.
2. Give the heading a 10px dotted `border-bottom` which uses the CSS color keyword `purple`.
3. Make the level 2 heading italic.
4. Give the `ul` used for the contact details a `background-color` of `#eeeeee`, and a 5px solid purple `border`. Use some `padding` to push the content away from the border.
5. Make the links `green` on hover.

You should end up with something like this image.



Jane Doe

● ●

Web Developer

Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean.

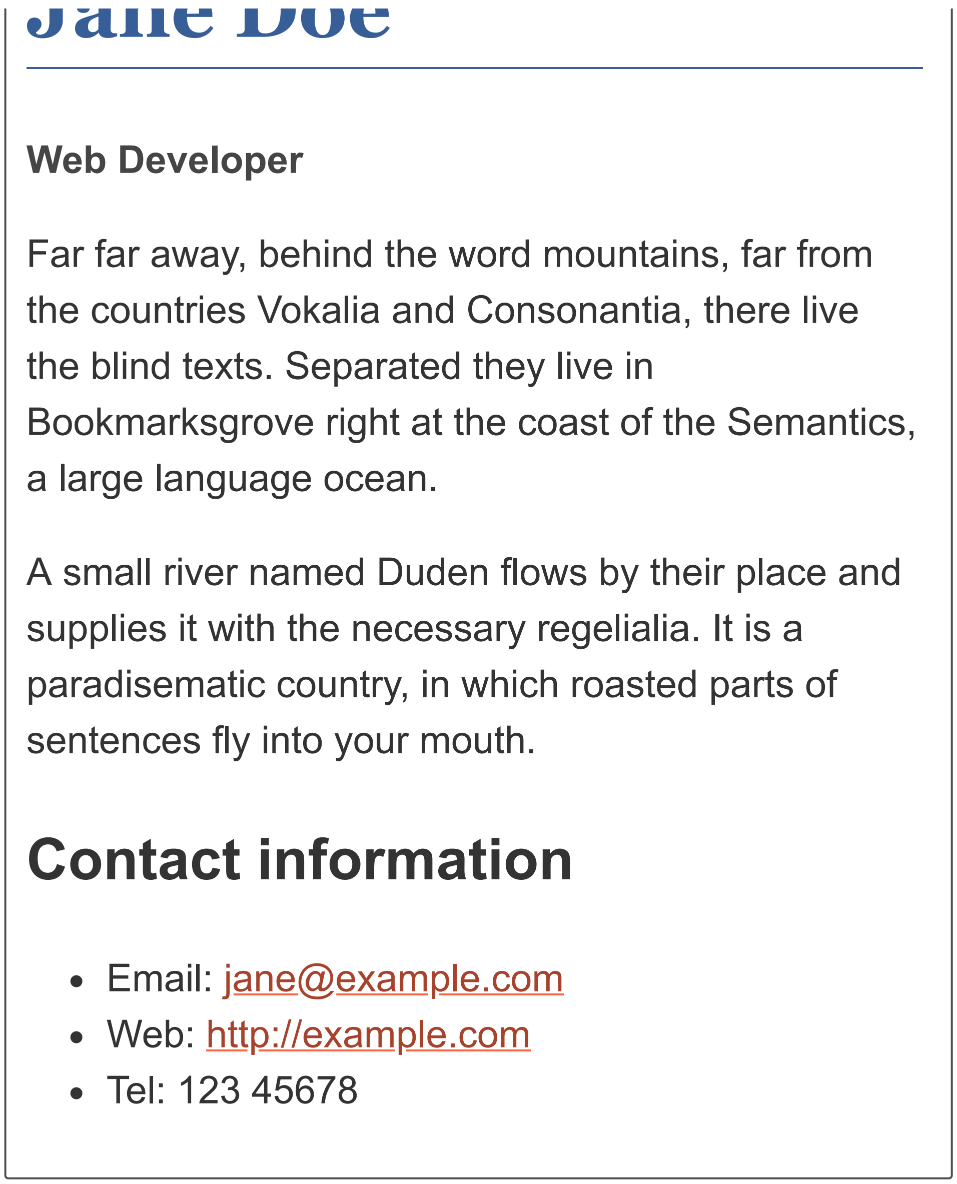
A small river named Duden flows by their place and supplies it with the necessary regelialia. It is a paradisematic country, in which roasted parts of sentences fly into your mouth.

Contact information

- Email: jane@example.com
- Web: http://example.com
- Tel: 123 45678

Afterwards try looking up some properties not mentioned on this page in the MDN CSS reference and get adventurous!

Remember that there is no wrong answer here — at this stage in your learning you can afford to have a bit of fun.



Jane Doe

Web Developer

Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean.

A small river named Duden flows by their place and supplies it with the necessary regelialia. It is a paradisematic country, in which roasted parts of sentences fly into your mouth.

Contact information

- Email: jane@example.com
- Web: http://example.com
- Tel: 123 45678

```
body {
    font-family: Arial, Helvetica, sans-serif;
}

h1 {
    color: #375e97;
    font-size: 2em;
    font-family: Georgia, 'Times New Roman', Times, serif;
    border-bottom: 1px solid #375e97;
}

h2 {
    font-size: 1.5em;
}

.job-title {
    color: #999999;
    font-weight: bold;
}

a:link, a:visited {
    color: #fb6542;
}

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

```
<h1>Jane Doe</h1>
<div class="job-title">Web Developer</div>
<p>Far far away, behind the word mountains, far from
the countries Vokalia and Consonantia, there live the
blind texts. Separated they live in Bookmarksgrove
right at the coast of the Semantics, a large language
ocean.</p>

<p>A small river named Duden flows by their place and
supplies it with the necessary regelialia. It is a
paradisematic country, in which roasted parts of
sentences fly into your mouth. </p>

<h2>Contact information</h2>
<ul>
    <li>Email: <a
href="mailto:jane@example.com">jane@example.com</a>
</li>
```

Reset

Assessment or further help

If you would like your work assessed, or are stuck and want to ask for help:

1. Put your work into an online shareable editor such as CodePen, jsFiddle, or Glitch.
2. Write a post asking for assessment and/or help at the MDN Discourse forum Learning category. Your post should include:
    - A descriptive title such as "Assessment wanted for CSS First Steps".
    - Details of what you have already tried, and what you would like us to do, e.g. if you are stuck and need help, or want an assessment.

- A link to the example you want assessed or need help with, in an online shareable editor (as mentioned in step 1 above). This is a good practice to get into — it's very hard to help someone with a coding problem if you can't see their code.
- A link to the actual task or assessment page, so we can find the question you want help with.

What's next?

Congratulations on finishing this first module. You should now have a good general understanding of CSS, and be able to understand much of what is happening in a stylesheet. In the next module, CSS building blocks, we will go on to look at a number of key areas in depth.

Previous Overview: First steps

In this module

Last modified: Feb 20, 2020, by MDN contributors

Related Topics

Complete beginners start here!