

Introduction to Cascading Style Sheets

Why CSS?

CSS offers far greater control over presentation than does HTML - there is a greater range of options available, which can be tweaked with much more precision and flexibility than HTML's crude presentational attributes.

CSS uses a completely different syntax (that is, rules governing how the language can be used) from HTML. This syntax is quite simple, though the way in which the CSS rules interact with HTML can get quite complicated.

Applying CSS

CSS can be applied to a page in 3 ways:

External styles

You can use the `<link>` tag inside the `<head>` element to create a link to an external stylesheet:

```
<link rel="stylesheet" type="text/css" href="cssfolder/stylesheetname.css" />
```

The `rel` and `type` attributes should always have the values shown if the tag is linking to a stylesheet; the value of the `href` attribute is the location of the CSS file. CSS files should always be saved with a `.css` extension.

Internal styles:

An internal stylesheet is defined in the `<head>` element of a page, inside `<style>` tags:

```
<style type="text/css">
```

```
... css goes here ...
```

```
</style>
```

Inline styles:

Inline styles can be defined within individual HTML elements, using the `style` attribute:

```
<p style="color: #ff0000">paragraph text</p>
```

For a completed site, external stylesheets are undoubtedly the most useful, as they can be accessed by multiple pages; however, during production and testing it can be easier to use internal or inline styles to save having to work on two files at once.

CSS syntax

CSS works by defining how an HTML element should be displayed. Consider the following CSS - this can be inserted within `<style>` tags or saved on its own as a CSS file:

```
body {
  color: #090;
}
h1 {
  color: #ea6300;
  background: #fffaf4;
}
```

This code defines the appearance of the `<body>` and `<h1>` tags. The tag is specified without its angle brackets, and any properties given to this tag are contained within curly braces `{ }`.

You can create an **inline** style by simply inserting the properties inside the **style** attribute of the desired tag. For example:

```
<h1 style="color: #ea6300; background: #fffaf4;">Heading</h1>
```

Selectors, properties & values

In this example, `body { }` and `h1 { }` are **selectors**. A selector specifies the HTML element whose appearance is being changed.

color and **background** are **properties**. There are many properties that can be defined using CSS; in this case, **color** sets the colour of the text (and border, if any) within the element, and **background** sets the background colour of the element (it can also be used to add background images, as we shall see).

Each property is given a **value** - this follows a colon `:` immediately after the name of the property. In this example, we are using [hexadecimal colour codes](#) to define colours. Each value should be closed by a semi-colon `;` before defining the next property.

All CSS rules take the following form:

```
selector {
  property: value;
}
```

A stylesheet can contain an unlimited amount of selectors, in any order. Each selector can contain as many property-value pairs as you like; these, too, can be in any order you like. Unlike HTML, CSS contains no basic structural code; it simply lists the various selectors and their properties.

For a comprehensive reference of CSS properties, see www.w3schools.com/css/css_reference.asp or www.htmldog.com/reference/cssproperties/. While the full list might seem bewilderingly long, bear in mind that many properties are not widely supported in current browsers, and so we'll only be using a (relatively) small number of the available properties.

Grouping selectors

Suppose we wanted to change the colour of all heading tags. We could use multiple selectors - `h1 { } h2 { }...` However we can save a lot of time by **grouping** the selectors, like this:

```
h1, h2, h3, h4, h5, h6 {
    color: #ea6300;
    background: #fffaf4;
}
```

This can be used to group as many selectors as we want. Note that selectors are comma-separated.

Nesting selectors

Using selectors thus far has been rather a blunt instrument: we can change the appearance of all instances of a certain tag, but what if we only wanted to change some instances? **Nesting** is one way in which we can start to get more control over our page. Here's an example:

```
li a {
    color: #a00;
}
```

This rule changes the colour of the `<a>` tag, but it will only apply to an `<a>` which is contained in a ``. All other `<a>` tags will retain the default link colour.

You can nest to as many levels as you like. For instance,

```
li li table h3 a {
    color: #a00;
}
```

will set the colour of an `<a>` tag within an `<h3>`, which itself is inside a `<table>`, within a ``, within another ``. As you can see, nesting can get rather fiddly, so take it slowly and avoid unnecessary complexity.

Classes and ids

Classes and **ids** allow us to specify one, or a group, of HTML elements. This gives us far greater precision in changing the appearance of particular elements.

A class or id is applied to an HTML element using the **class** or **id** attribute:

```
<h1 id="mainheading">Main heading</h1>
<p class="redpara">A red paragraph</p>
```

An **id** must be **unique** - it can only be specified for a single element per page. **Classes** can be applied to as many elements as you like.

Once a class or id has been defined in the HTML, we can attach styles to them using the following CSS:

```
#mainheading {
    color: #ea6300;
}
.redpara {
    color: #a00;
}
```

Note that (in the CSS, but *not* the HTML) the id name is preceded by the # symbol, and the class name is preceded by a full stop.

CSS comments

You can add comments to CSS just as you can with HTML, to leave a note for yourself or anyone else who might need to understand your code. Unlike HTML, CSS comments begin with `/*` and end with `*/`. For example:

```
body {
/* this is a comment */
    color: #090;
}
```

Typography

You can make many changes to the type using CSS. Here are some common properties:

font-family

This defines the typeface that will be used. Be aware that this will only work if the font is installed on the site visitor's computer; unfortunately, the list of 'safe' fonts that almost everyone can be assumed to have is very small and not particularly exciting. The most commonly used are:

- Arial
- Georgia
- Times New Roman
- Trebuchet MS
- Verdana

Even though most users will have these fonts installed, it's still a good idea to specify alternative fonts to display just in case your first choice is not available. This is done by listing each font in order of priority. For example:

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

Here, Trebuchet MS is the first choice - if the user does not have this font installed, Arial will be substituted. If neither of the first two are available, Helvetica will be used, and if none of the named fonts are installed, the system default sans-serif font will be used. Note that font names of more than one word must be put in quotes.

font-size

Surprisingly enough, this specifies the size of the font. There are many units that can be used, but I would recommend using either **percentages** or **ems** (one em is equivalent to the width of the letter 'm'). These are **relative** units of measurement, that is, they are defined relative to the initial size of the font, rather than as a fixed size in pixels. This will ensure that text can be easily resized, which considerably increases the accessibility of your site. For example:

```
body {  
    font-size: 0.9em;  
}
```

font-weight

Can be used to set text as **bold** or **normal**.

```
li {  
    font-weight: bold;  
}  
  
li li {  
    font-weight: normal;  
}
```

This sets list items to be displayed in **bold**, but any list items within list items to display normally (see **Grouping and Nesting** in the previous section).

font-style

Sets text to **italic** or **normal**.

```
li {  
    font-style: italic;  
}  
  
li li {  
    font-style: normal;  
}
```

color

Sets the colour of the text. If the element has a border, the colour will be applied to that as well. There are a few named colours, but it's better to use [hexadecimal colour codes](#).

```
body {  
    color: #444;  
}
```

text-decoration

This can add or remove other decorations to the text:

- **underline** should only be applied to links, or you run the risk of confusing people
- **overline** will add a line above the text
- **line-through** adds a line through the text.
- **none** removes any default text-decoration. This can be useful when styling links, as they are underlined by default.

You can combine multiple values within this property - for example:

```
a {
    text-decoration: underline overline;
}
```

will give your links an underline and an overline.

text-align

This property is equivalent to the **align** attribute in HTML. It can take values of **left**, **center**, **right** or **justify**.

So putting it all together, we can create a set of rules like this:

```
body {
    font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
    font-size: 0.9em;
    color: #090;
}
a {
    font-weight: bold;
    color: #ea6300;
    text-decoration: none;
}
h1, h2, h3, h4, h5, h6 {
    color: #a00;
    text-align: center;
}
```

Other text properties

Once you're familiar with the properties described in this section, you may want to investigate more text-related properties, such as:

- [font-variant](#)
- [letter-spacing](#) and [word-spacing](#)
- [line-height](#)
- [text-indent](#)
- [text-transform](#)