

[Back](#)[Start](#)[Next](#)

Part 5. Introduction to CSS

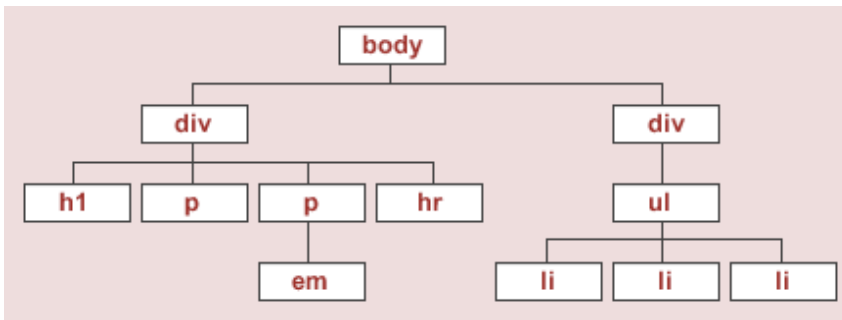
The document tree - it's a family thing

All HTML documents are trees. Each level of the tree is described in the same manner as a human family tree, with ancestors, descendants, parents, children and siblings. CSS rules are based on the document tree. If you understand the document tree concept, then CSS selectors will be easy to understand and apply.

Let's start with a sample of HTML. This sample doesn't include the head or title, as we are focussing on what is inside the body:

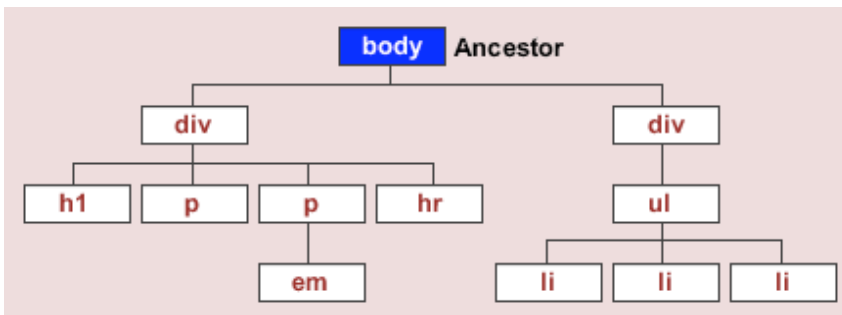
```
<body>
<div id="content">
<h1>Heading here</h1>
<p>Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor <em>sit</em> amet.</p>
<hr>
</div>
<div id="nav">
<ul>
  <li>item 1</li>
  <li>item 2</li>
  <li>item 3</li>
</ul>
</div>
</body>
```

The document tree diagram for the sample above would be:



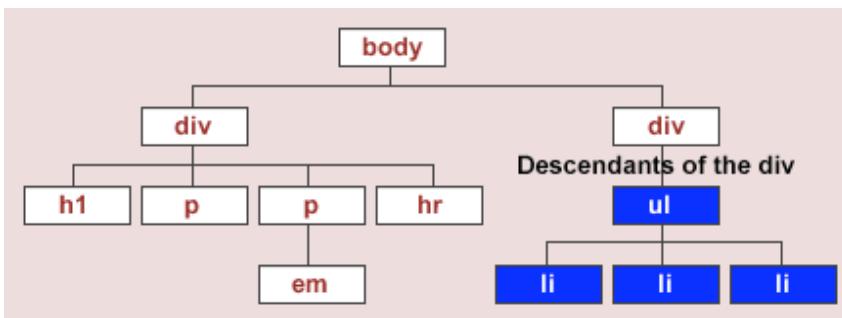
Ancestor

An ancestor refers to any element that is connected but further up the document tree - no matter how many levels higher. In the diagram below, the `<body>` element is the ancestor of all other elements on the page.



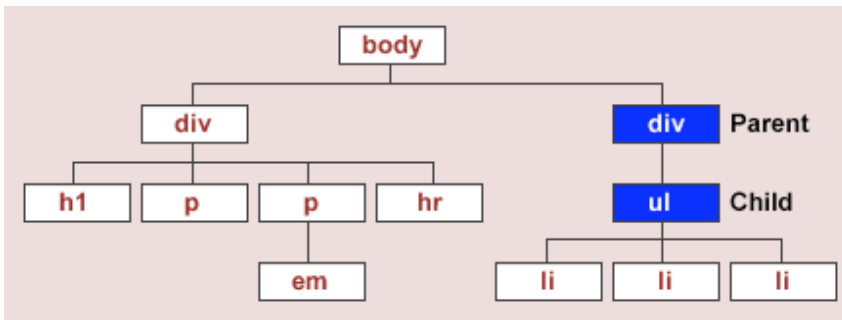
Descendant

A descendant refers to any element that is connected but lower down the document tree - no matter how many levels lower. In the diagram below, all elements that are connected below the `<div>` element are descendants of that `<div>`.



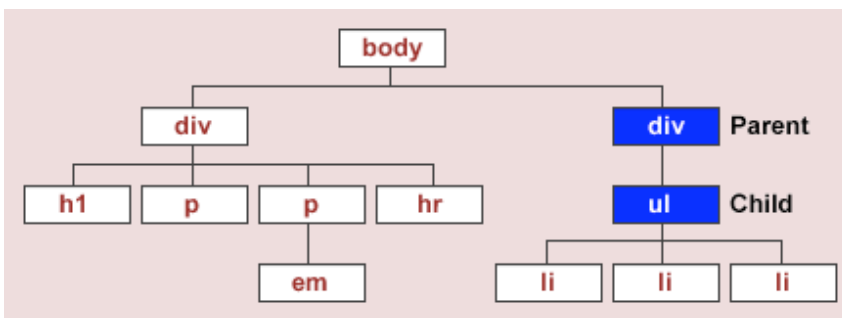
Parent

A parent is an element that is *directly* above and connected to an element in the document tree. In the diagram below, the `<div>` is a parent to the ``.



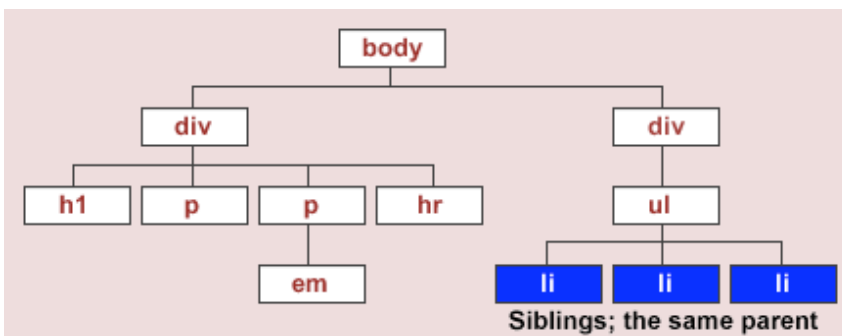
Child

A child is an element that is *directly* below and connected to an element in the document tree. In the diagram above, the `` is a child to the `<div>`.



Sibling

A sibling is an element that shares the *same parent* with another element. In the diagram below, the ``'s are siblings as they all share the same parent - the ``.



[Back](#)[Start](#)[Next](#)

Part 5. Introduction to CSS

Cascade

What does "cascading" mean?

"Cascading" means that styles can fall (or cascade) from one style sheet to another, enabling multiple style sheets to be used on one HTML document.

Even the simplest HTML document may have three or more style sheets associated with it including:

- The browser's style sheet
- The user's style sheet
- The author's style sheet

Browser style sheets

Browsers apply style sheets to all web documents. Although these style sheets vary from browser to browser, they all have common characteristics such as black text, blue links, purple visited links etc. These are referred to as a "default" browser stylesheet.

As soon as you, the **author**, apply a style sheet to a document, it will override the browser's style sheet. This is because author style sheets will always override browser style sheets.

User style sheets

A **user** is anyone who looks at your website.

Most modern browsers allow users to set their own style sheets within their browser. These style sheets will override the browsers default style sheets - **for that user only**.

It is important to know that users can set color and background color for HTML documents. If you, as an author, do not specify a color or background color, a user's style sheet could be used. More importantly, you should either specify both color and background color or neither. If you only specify a color and the user has specified the same color for their background, your entire document may be inaccessible to this user.

Author style sheets

The **author** is the person who develops the website - **you!**

As soon as you apply a basic style sheet or an inline style to a page, you have added what is referred to as an "**author style sheet**". Everything you do, from choosing fonts, colours and laying out pages in CSS is done using author style sheets.

Author style sheets can be applied inside an HTML document or by linking to an external file. You can also use multiple style sheets on an particular document to take advantage of the cascade.

Generally speaking, author styles override user styles which override browser styles.

!important rules

You can set "important" on any declaration and this will override other declarations for the same element. "important" declarations override normal declarations.

For "important" declarations, user style sheets override author style sheets which override the default style sheet. This allows users with special requirements (large fonts, color combinations, etc.) control over presentation.

Why use multiple author style sheets?

Style sheets can be used to target different user agents, so a single HTML document could have links to a number of style sheets including aural, print and screen style sheets.

For a large site, you may wish to use a range of visual style sheets for each document. This could include a general style sheet (applied across all sections of the site) for basic font information and then individual style sheets for each section of the site. Or, certain aspects of each document could be styled by individual style sheets. You could use a footer style sheet, a navigation style sheet and a range of overall content style sheets.

There is no limit to the way you break up your style sheet information. The aim is to use style sheets so that information does not have to be repeated in other style sheets. This means you can edit once and apply often.

Copyright © Max Design 2005

[Back](#) [Start](#) [Next](#)
Part 5. Introduction to CSS

Specificity

There may be times when two or more declarations are applied to the same element. It is also possible that there may be a conflict between them. When conflicts like this occur, the declaration with the most weight is used. So, how is weight determined?

When documents load in a browser, all declarations are sorted and given a weight. They are then applied to a document based on this weight. There are four steps to sorting. As the process is quite complex, we will use one HTML element (<h2>) as an example.

1. Find all declarations whose selectors match a particular element.

After looking at all style sheets (browser, user and author), all the declarations that match an <h2> element are found.

The **browser style sheet** may include the following relevant rule:

```
h2 { font-size: 1.5em; margin: .83em 0; color: black }
```

The **users style sheet** could include the following relevant rules

```
h2 { color: brown !important; }
```

The **author style sheet** could include the following relevant rules:

```
h2 { color: green; font-size: 1.2em; padding: 10px; }  
h2#main { color: red; }  
h2.navigation { color: blue;}
```

2. Sort these declarations by weight and origin

The above rules are then sorted by weight and origin. For normal declarations, the author style sheets will override both the user style sheets and the browser style sheet unless there is an

important declaration in the user's style sheet. For "important" declarations, user style sheets override author and browser style sheets.

For general users, <h2> elements that are not specifically styled with "#main" or ".navigation" will be styled with:

```
font-size: 1.2em; /* author wins over browser */
margin: .83em 0; /* specified only by browser */
padding: 10px; ; /* specified only by author */
color: green; /* author wins over browser */
```

For the user that has set an !important rule, <h2> elements that are not specifically styled with "#main" or ".navigation" will be styled with:

```
font-size: 1.2em; /* author wins over browser */
margin: .83em 0; /* specified only by browser */
padding: 10px; ; /* specified only by author */
color: brown; /* user wins due to !important */
```

3. Sort the selectors by specificity

Even though we have sorted out the general <h2> style, there still may be conflict in other areas of the document. What happens with an <h2> that has been styled with "#main"?

Every selector is given a specificity which is calculated on the entire selector. More specific selectors will override more general ones. The calculation is based on the following (simplified) rules:

- a . count the number of **id's** in the overall selector
- b . count the number of **other selectors** in the overall selector
- c . count the number of **elements** within the overall selector

These three calculations are then written out as a-b-c. Using the examples above, the following declarations can be scored as:

```
h2 { font-size: 1.5em; margin: .83em 0; color: black }
0-0-1 > specificity = 1
```

```
h2 { color: brown !important; }  
0-0-1 > specificity = 1  
  
h2 { color: green; font-size: 1.2em; padding: 10px; }  
0-0-1 > specificity = 1  
  
h2#main { color: red; }  
1-0-1 > specificity = 101  
  
h2.navigation { color: blue;}  
0-1-1 > specificity = 11
```

This means that "h2#main" has the greatest weight, followed by "h2.navigation".

For general users (who will not be affected by the users !important rule) any <h2> styled with id="main" will be:

```
color: red; /* id more specific than type selector */
```

Any <h2> styled with class="navigation" will be:

```
color: blue; /* class more specific than type selector */
```

If there was a clash between "#main" and ".navigation" the ID would win and the <h2> will be:

```
color: red; /* id more specific than class or type selector */
```

4. Sort by order specified

If two rules have the same weight, origin and specificity, the one written lower down in the style sheet wins. Rules within imported style sheets are considered to be before any rules in the style sheet itself.

For example, the style sheet could contain two identical rules in various sections of the document:

```
h2 { color: blue;}  
h2 { color: green; }
```

The second rule, that appears later in the style sheet would win.

Copyright © Max Design 2005

[Back](#)[Start](#)[Next](#)

Part 5. Introduction to CSS

Shorthand rules

Rules can be combined, allowing you specify several properties with one rule set. The biggest advantage is that your CSS code becomes shorter. For example the two rule sets below have exactly the same effect:

```
p
{
padding-top: 1em;
padding-right: 1em;
padding-bottom: 1em;
padding-left: 1em;
}

p { padding: 1em; }
```

If property values are different for all sides, shorthand can still be used. The order of each value is important. Values are applied in the following order; top, right, bottom and left - clockwise, starting at the top. For example the first rule set below can be shortened:

```
p
{
padding-top: 1em;
padding-right: 2em;
padding-bottom: 3em;
padding-left: 4em;
}

p { padding: 1em 2em 3em 4em; }
```

You can use one, two, three and four values within a shorthand declaration. For example, the rule below will apply padding to all sides of a box:

```
p { padding: 1em; }
```

The rule below will apply 1em of padding to the top and bottom, and 2em of padding to the left and right of the box.

```
p { padding: 1em 2em; }
```

The rule below will apply 1em of padding to the top, 2em of padding to the left and right, and 3em to the bottom of the box.

```
p { padding: 1em 2em 3em; }
```

The rule below will apply 1em of padding to the top, 2em of padding to the right, 3em of padding to the bottom and 4em of padding to the left of the box.

```
p { padding: 1em 2em 3em 4em; }
```

Browser support for shorthand rules varies. For detailed browser support charts, see the [Stylemaster - CSS compatibility Chart](#)

Copyright © Max Design 2005

[Back](#)[Start](#)[Next](#)

Part 5. Introduction to CSS

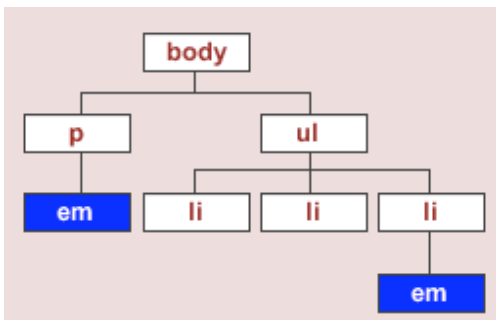
Selectors

Type selectors

The most common and easy to understand selectors are type selectors. Type selectors will select any HTML element on a page that matches the selector, regardless of their position in the document tree. For example:

```
em {color: blue; }
```

This rule will select any `` element on the page and color it blue. As you can see from the document tree diagram below, all `` elements will be colored blue, regardless of their position in the document tree.



There are a huge range of elements that you can select using type selectors, which means you can change the appearance of any or every element on your page using only type selectors.

Class selectors

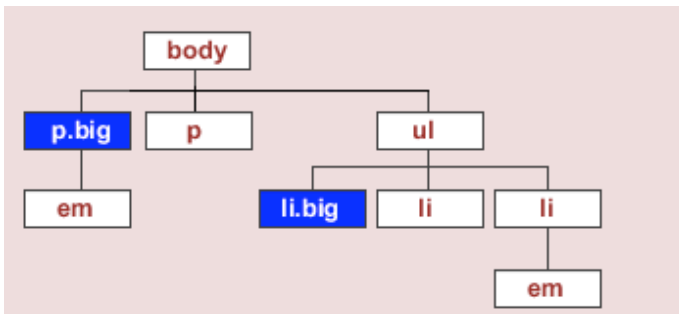
While type selectors target **every** instance of an element, class selectors can be used to select any HTML element that has a class attribute, regardless of their position in the document tree.

For example, if you want to target the first paragraph and first list items on a page to make them stand out, you could mark up the page in the following way:

```
<body>
<p class="big">This is some <em>text</em></p>
```

```
<p>This is some text</p>
<ul>
<li class="big">List item</li>
<li>List item</li>
<li>List <em>item</em></li>
</ul>
</body>
```

The tree diagram would be:



The rule used could then be:

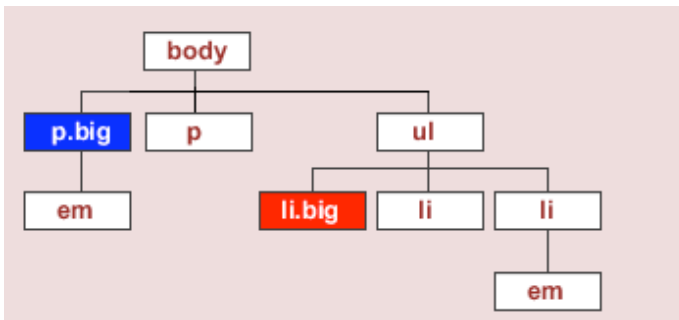
```
.big { font-size: 110%; font-weight: bold; }
```

Combining class and type selectors

If you want to be more specific, you can use class and type selectors together. Any type selectors can be used.

```
div.big { color: blue; }
td.big { color: yellow; }
label.big { color: green; }
form.big { color: red; }
```

For example, you may want to target the first paragraph and first list items on a page to give them a larger font size. You may also want only the paragraph to be bold.



To do this, you could use the following rules:

```
.big { font-size: 110%; } /* affects p and li */  
p.big { font-weight: bold; } /* affects p only */
```

ID selectors

ID selectors are similar to class selectors. They can be used to select any HTML element that has an ID attribute, regardless of their position in the document tree. Examples of ID selectors:

```
#navigation { width: 12em; color: #333; }  
div#navigation { width: 12em; color: #333; }
```

The major difference is that IDs can only be applied once per page, while classes can be used as many times on a page as needed.

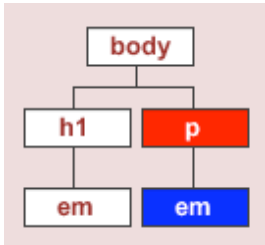
Descendant selectors

Descendant selectors are used to select elements that are descendants of another element in the document tree.

For example, you may wish to target a specific `` element on the page, but not all `` elements. A sample document could contain the following code:

```
<body>  
<h1>Heading <em>here</em> </h1>  
<p>Lorem ipsum dolor <em>sit</em> amet.</p>  
</body>
```

The document tree diagram (with the `` element to be targeted) would be:



If you use a type selector like the example below, you will select all `` elements on the page:

```
em {color: blue; }
```

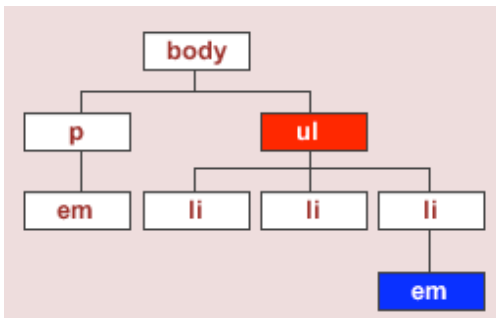
However, if you use a descendant selector, you can refine the `` elements that you select. The rule below will only select `` elements that are descendants of `<p>` elements. If this rule is applied, the `` element within the `<h1>` will not be colored blue.

```
p em {color: blue; }
```

You can also jump levels in the document tree structure to select descendants. For example, the following code:

```
<body>
<p>Lorem ipsum dolor <em>sit</em> amet.</p>
<ul>
<li>item 1</li>
<li>item 2</li>
<li><em>item 3</em></li>
</ul>
</body>
```

The document tree (with a third-level `` element highlighted) would be:



Using the following rule you can isolate any `` element inside a `` element, without having to describe the `` element. If this rule is applied, any `` element within a `` element will be colored blue. However, the `` element within the `<p>` will not be colored blue:

```
ul em {color: blue; }
```

Pseudo-classes

So far, all the selectors have been based on elements within the document tree. However, there are times when you may want to style something where there is no CSS selector available, like the state of a hyperlink (eg. active or visited).

Pseudo-classes allow you to format items that are not in the document tree. They include:

- `:first-child`
- `:link`
- `:visited`
- `:hover`
- `:active`
- `:focus`
- `:lang(n)`

Styling links

With pseudo-classes, you can style links in different ways in each of the four states.

`a:link` is the selector for normal links

`a:visited` is the selector for visited links

`a:hover` is the selector for hover state

`a:active` is the selector for active links

Due to [specificity](#), the selector that appears later in the stylesheet will be used if there is any conflict. For this reason, `link` and `link` pseudo class selectors should always be used in the

following order.

```
a {}  
a:link {}  
a:visited {}  
a:hover {}  
a:active {}
```

All of the usual properties can be used on these four states. You can also combine states if needed, as long as the order (link and visited before hover and active) is maintained:

```
a:link, a:visited { color: blue; }  
a:hover, a:active { color: red; }
```

When styling the four link states you should be aware that modifying standard hyperlink behavior (such as underlines) can be confusing for some users, who may not realise that the item is a link.

Copyright © Max Design 2005

[Back](#)[Start](#)[Next](#)

Part 5. Introduction to CSS

Grouping selectors

Selectors are used to "select" the elements on an HTML page that are affected by rules. When several selectors share the same declarations, they may be grouped together to save writing the same rule more than once. Each selector must be separated by a comma. They can also include a wide range of selectors. For example:

```
h1, h2, h3, h4 { padding: 1em; }  
.highlight p, .highlight ul { margin-left: .5em; }  
#main p, #main ul { padding-top: 1em; }
```

There are two common mistakes that people make when grouping selectors. The first is forgetting to write each selector in full. For example, if you are trying to target two elements within the same containing block, and with the same ID, chances are you will probably want them to be written in the same way. So, this is probably wrong:

```
#maincontent p, ul { border-top: 1px solid #ddd; }
```

The correct group selector would more likely be:

```
#maincontent p, #maincontent ul { padding-top: 1em; }
```

The second common mistake is to end the grouping with a comma. This can cause certain browsers to ignore the rule entirely.

```
.highlight p, .highlight ul, { margin-left: .5em; }
```

External Resources

- [W3C CSS2 Specification: Group selectors](#)
- [DevEdge CSS2 Selectors Support Chart](#)
- [WestCiv Selectors Support Chart](#)

[Back](#)[Start](#)[Next](#)

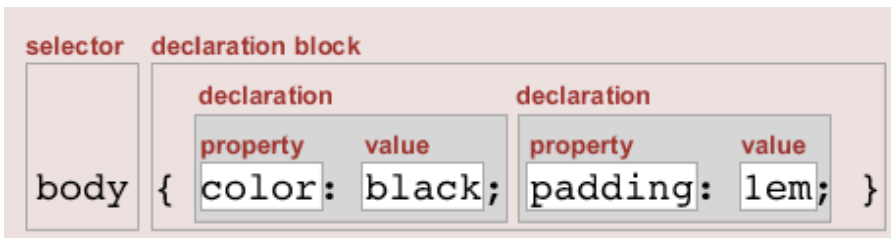
Part 5. Introduction to CSS

What is a rule or "rule set"?

Introduction

A rule or "rule set" is a statement that tells browsers how to render particular elements on an HTML page. A rule set consists of a selector followed by a declaration block.

Rule structure



Selector

The selector "selects" the elements on an HTML page that are affected by the rule set. The selector consists of everything up to (but not including) the first left curly bracket. For example:

```
h1 { color: blue; margin-top: 1em; }
p { padding: 5px; }
td { background-color: #ddd; }
```

Declaration block

The declaration block is a container that consists of anything between (and including) the curly brackets. Whitespace inside a declaration block is ignored - so it can be used to lay out rules in any way you want. For example:

```
h1 { color: blue; }
p { padding: 5px; }
td { background-color: #ddd; }
```

Or, with whitespace added to aid readability:

```
h1
{
  color: blue;
}
```

Declaration

The declaration tells a browser how to draw any element on a page that is selected. A declaration consists of a property and a value, separated by a colon ":". Although it is not essential to add a semicolon after a single declaration, it is recommended that you finish the last declaration in a block with a semicolon. For example:

```
h1 { color: blue; }
```

Property

The property is the aspect of that element that you are choosing to style. There can only be one property within each declaration. For example:

```
p { padding: 5px; }
```

Value

The value is the exact style you wish to set for the property. For example:

```
p { padding:5px; }
```

External resources

- [W3C CSS2 Specification: rule sets](#)

[Back](#)[Start](#)[Next](#)

Part 3 - Semantic markup

Heading levels - the great debate

The finer details of semantic markup come down to personal opinion. Often semantic markup details are hotly debated. Commonly debated topics include:

- Can you have multiple h1's in one document?
- Can you skip a heading level?
- Should site title or page title be considered the h1 within a document

Can you have multiple h1's in one document?

The short answer is no - you should only have one h1 per document.

Can you skip a heading level?

This will depend on the overall site in which the document sits. If you define your heading levels across a site, there may be times when a heading level is missing on a specific page, but to change it up or down a level may then take it out of context with the rest of the site.

Should the page title be considered the h1 within a document?

There are two schools of thought on this. Some people believe that the site name is the most important information and should be wrapped in an h1 element. Others believe that the page title (which should match the document title) should be wrapped in the h1, and the site title sits above this with no real semantic meaning.

Copyright © Max Design 2005