

Designing Web Applications

CSS

Francesco Marcelloni

Department of Information Engineering
University of Pisa
ITALY





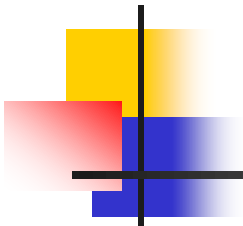
Cascading Style Sheets

- Cascading Style Sheets (CSS) is the recommended way to control the presentation layer in a web document. The main advantage of CSS over presentational HTML markup is that the styling can be kept entirely separate from the content.
- For example, it is possible to store all the presentational styles for a 10,000-page web site in a single CSS file. CSS also provides far better control over presentation than do presentational element types in HTML.



CSS Benefits

- By externalizing the presentation layer, CSS offers a number of significant benefits:
 - All styling is kept in a limited number of style sheets. Editing one style sheet is obviously more efficient than editing 10,000 HTML files!
 - The overall saving in bandwidth is measurable.
 - The style sheet is cached after the first request
 - Removing all presentational markup from your web pages reduces their size and bandwidth usage — by more than 50% in many documented cases
 - Benefits for the site owner, through lower bandwidth and storage costs,
 - Benefits for the site's visitors, for whom the web pages load faster.



CSS Benefits

- The separation of content from presentation makes it easier for site owners to reuse the content for other purposes, such as RSS feeds or text-to-speech conversion.
- Separate styling rules can be used for different output media. We no longer need to create a special version of each page for printing
 - we can simply create a single style sheet that controls how every page on the site will be printed.
- Although CSS is designed to be independent of the markup language of the documents to which it is applied, in reality, it is used mainly with HTML and XML (including XHTML).





CSS Versions

- The first CSS specification, CSS1, became a World Wide Web Consortium (W3C) recommendation in December 1996.
 - It included properties for controlling typography, such as fonts, text alignment, spacing, margins, and list formatting.
- CSS2 came out in 1998, and contained a lot of the features that designers had been longing for.
 - Boxes could be made to behave like HTML table cells, or they could be positioned in different ways;
 - more powerful selectors were available;
 - style sheets could be imported into other style sheets;
 - style rules could be specific to certain output media; and so on.
 - Vast improvements in the areas of paged media (printing), and the generation of content from the style sheet.



CSS Versions

- As it turned out, some parts of CSS2 were very difficult to implement, so the W3C decided to revise the specification and adapt it to real-world situations.
- The name of the revised version was “Cascading Style Sheets, Level 2 Revision 1”—CSS2.1 for short.
- In the last years, updates of different sections, called modules – **CSS3**



Including Style Sheets in HTML Documents

To insert CSS into an HTML document

There are three ways of inserting a style sheet into an HTML document:

- Inline style
- Internal style sheet
- External style sheet



Inline Styles

- Place the rules in inline CSS specified in a style attribute of a markup tag
- The style attribute can contain any CSS property.

`<p style="color:red; margin-left: 10%">This is a paragraph.</p>`

- **Use this method sparingly!**
 - Using style attributes **creates a strong coupling between the content and the presentation**, which is usually undesirable.
 - **Inline CSS is more limited** than internal or external style sheets. It is not possible to specify a media type, so style attributes will apply for all media types.



Internal Style Sheet

- Place the rules within a separate, internal style sheet in the head section of an HTML page, **by using the <style> tag**
- **An internal style sheet should be used when a single document has a unique style.**

```
<head>  
<style type="text/css"; media="screen">  
hr {color:red;}  
p {margin-left:20px;}  
body {background:red;}  
</style> </head>
```



Internal Style Sheet

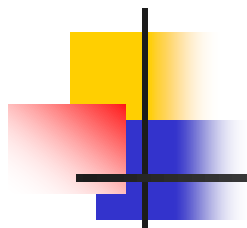
- NOTE: Do not leave spaces between the property value and the units!

"margin-left:20 px" (instead of "margin-left:20px") will work in IE, but not in Firefox or Opera.



External Style Sheet

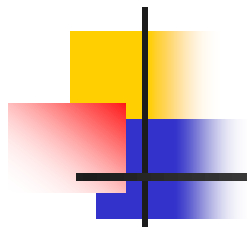
- Place the rules in a separate, external style sheet that is referenced by a link element or an @import rule in a style element
- **Ideal when the style is applied to several pages**
 - You can change the look of an entire Web site **by changing one file.**
 - An external style sheet can be written in any text editor.
 - The file should not contain any html tags.
 - Your style sheet should be saved with a **.css** extension.



External Style Sheet

```
<head>
  <title> CSS Example </title>
  <link rel="stylesheet" type="text/css" href="ex1.css">
</head>
<body>
  <h1>
    This header is 36 pt
  </h1>
  <h2>
    This header is blue
  </h2>
  <p>
    This paragraph has a left margin of 50 pixels
  </p>
</body>
```





External Style Sheet

```
/*file ex1.css*/  
body  
{ background-color:yellow;}  
h1  
{ font-size:36pt; }  
h2  
{ color:blue;}  
p  
{ margin-left:50px; }
```



Including External Style Sheet The Link Tag

First Method: the Link Tag

- The `<link>` tag is the most used method to link to style sheets
- Note: The link element must be embedded in the head section, and it can appear any number of times.
- The attribute **rel** specifies the relationship between the current document and the linked document. When used in the style sheet inclusion **rel="stylesheet"**.
- The **attribute type** specifies the Multipurpose Internet Mail Extensions (MIME) type of the linked document

Including External Style Sheet

The @import at-rule

Second Method: the directive @import

- The directive is used in the content of a style tag

```
<head>
```

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

```
  @import url(/style.css);
```

```
</style>
```

```
</head>
```




Media Type

- Through style sheets you can specify how a **document is to be presented on different media**: on the screen, on paper, with a speech synthesizer, with a braille device, etc.
- Certain **CSS properties are only designed for certain media** (e.g., the 'page-break-before' property only applies to paged media). Style sheets for different media types may share a property, but require different values for that property.

How can you specify the Media Type?

- There are currently **two ways to specify media dependencies** for style sheets:
 1. **Specify the target medium for a style sheet** with the @import at-rule:

```
@import url("fancyfonts.css") screen;
```

or with the @media at-rules.

```
@media print {  
    /* style sheet for print goes here */  
}
```



Media Type

2. Specify the target medium within the document link.

```
<head>
```

```
  <title>  Link to a target medium  </title>
```

```
  <link rel="stylesheet"
        type="text/css"
        media="print, handheld"
        href="foo.css">
```

```
</head>
```

```
<body>
```

```
  <p> The body... </p>
```

```
</body>
```



Media Type

Media Type	Description
all	Used for all media type devices
braille	Used for braille tactile feedback devices
embossed	Used for paged braille printers
handheld	Used for small or handheld devices
print	Used for printers
projection	Used for projected presentations, like slides
screen	Used for computer screens
speech	Used for speech synthesizers
tty	Used for media using a fixed-pitch character grid, like teletypes and terminals
tv	Used for television-type devices



Media Type

```
<head>
<style type="text/css" media="screen">
  h1 { background:black; color:white; text-align:center }
  h2 { color: red; font-style:italic}  h3 { display: none }
</style>
<style type="text/css" media="print">
  h1 { color: black; text-align: left } h2 { display: none }
</style>
<title>Example Media Type</title>
</head>
<body>
<h1>Example</h1>
<h2>This is not printed</h2>
<h3>This is not displayed</h3>
</body>
```





Cascading Style Sheets: Syntax



CSS style sheet

- A CSS style sheet consists of a list of **statements**.
There are two kinds of statements:
 - **at-rules**
 - **rule sets**.
- There may be white space around the statements.
- Example

```
@import "subs.css";  
@media print {  
  @import "print-main.css";  
  body { font-size: 10pt }  
}  
h1 {color: blue }
```

CSS style sheet

At-rule

■ At-rules

- At-rules **start with an at-keyword**, an '@' character followed immediately by an identifier (for example, '@import', '@page').
- An at-rule consists of everything up to and including the next semicolon (;) or the next block, whichever comes first.
- A **block** starts with a left curly brace ({) and ends with the matching right curly brace (}).

CSS style sheet

Rule sets

■ Rule sets

- A rule set (also called "rule") consists of a selector followed by a declaration block.
- A declaration block starts with a left curly brace ({) and ends with the matching right curly brace (}). In between there must be a list of zero or more semicolon-separated (;) declarations.
- The selector consists of everything up to (but not including) the first left curly brace ({). A selector always goes together with a declaration block. When a user agent cannot parse the selector (i.e., it is not valid CSS 3.0), it must ignore the selector and the following declaration block (if any) as well.

CSS style sheet

Rule set

- Rule sets (example)

h1, h2 {color: green }

h3, h4 & h5 {color: red }

h6 {color: black }

Since the "&" is not a valid token in a CSS 3.0 selector, a CSS 3.0 user agent must ignore the whole second line

CSS style sheet

Declarations

■ Declarations

- A declaration is either empty or consists of a property name, followed by a colon (:), followed by a value. Around each of these there may be white space.

```
h1 { font-weight: bold }  
h1 { font-size: 12px }  
h1 { line-height: 14px }  
h1 { font-family: Helvetica }  
h1 { font-variant: normal }  
h1 { font-style: normal }
```

=

```
h1 {  
    font-weight: bold;  
    font-size: 12px;  
    line-height: 14px;  
    font-family: Helvetica;  
    font-variant: normal;  
    font-style: normal  
}
```

CSS style sheet Declarations

■ Declarations

- A user agent must ignore a declaration with an invalid property name or an invalid value.

```
h1 { color: red; font-style: 12pt } /* Invalid value: 12pt */  
p { color: blue; font-vendor: any; /* Invalid prop.: font-vendor */  
  font-variant: small-caps }  
em { font-style: normal }
```

The CSS parser will reduce the style sheet to

```
h1 { color: red; }  
p { color: blue; font-variant: small-caps }  
em { font-style: normal }
```

CSS style sheet

Comments

■ Comments

- Begin with the characters "/*" and end with the characters "*/".
- May occur anywhere between tokens, and their contents have no influence on the rendering.
- May not be nested.

CSS style sheet

Selectors

- Pattern matching rules determine which style rules apply to elements in the document tree.
 - These patterns, called **selectors**, may range from simple element names to rich contextual patterns.
 - If all conditions in the pattern are true for a certain element, the selector matches the element.
- A **selector** is a chain of one or more simple selectors separated by combinators.
- A **simple selector** is either a type selector or universal selector followed immediately by zero or more attribute selectors, ID selectors, or **pseudo-classes**, in any order. The simple selector matches if all of its components match.
- **Combinators are: white space, ">", and "+"**. White space may appear between a combinator and the simple selectors around it.

Selectors

Type Selector

- Type selector

- A type selector matches the name of a document language element type.

```
h1 { font-family: sans-serif }
```

```
p { text-align: justify }
```

```
div { position: absolute; left: 10px; top: 50px }
```

```
table { width: 80%; height: 50% }
```

Selectors

Universal Selector

- Universal selector

- The universal selector "*" matches the name of any element type.

```
<head>
```

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

```
  body { font-size: large; }
```

```
  h1 { color: green; }
```

```
  * { color: red; }
```

```
</style>
```

```
  <title>Universal Selector</title>
```

```
</head>
```

```
<body>
```

```
  <h1> The heading is green </h1>
```

```
  <p>   This text is red   </p>
```

```
</body>
```


Selectors

Grouping

- Grouping

- When several selectors share the same declarations, they may be grouped into a comma-separated list.
- In this example, we condense three rules with identical declarations into one. Thus,

```
h1 { font-family: sans-serif }
```

```
h2 { font-family: sans-serif }
```

```
h3 { font-family: sans-serif }
```

is equivalent to:

```
h1, h2, h3 { font-family: sans-serif }
```

Selectors

Class Selectors

■ Class Selectors

- If only type selectors were available, we would have that all the element instances would have the same format.

For instance:

```
p { font-family: Arial }
```

- To have different fonts for different paragraphs, we can exploit the class attribute.
- We can assign style information to all elements with class="myclass" as follows:

```
*.myclass { color: green } /* all elements with class=myclass */  
or just
```

```
.myclass { color: green } /* all elements with class=myclass */
```

Selectors

Class Selectors

- The following example assigns different font families to different paragraphs

```
<style type="text/css">
.arial { font-family: Arial }
.serif { font-family: serif }
</style>
<title> Font </title>
</head>
<body>
<p>Paragraph</p>
<p class="arial">This paragraph is in Arial font</p>
<p class="serif">This paragraph is in Serif font</p>
</body>
```



Selectors

Multiple Classes

- Multiple classes

`p.green.bold { color: green; font-weight: bold }`

The rule matches any `p` element whose "class" attribute has been assigned a list of space-separated values that includes "green" and "bold"

`<p class="green serif bold">` (OK)

`<p class="serif bold">` (NO)

Selectors

id selector

- id selector
 - Document languages may contain attributes that are declared to be of type id.
 - Important: no two such attributes can have the same value;
 - whatever the document language, an id attribute can be used to uniquely identify its element.
 - A CSS id selector contains a "#" immediately followed by the id value, which must be an identifier

Selectors

id selector

■ id selector (Example)

```
<head>
<style type="text/css">
  .red { color: red }
  #black { color: black }
</style>
</head>
<body>
<p>...</p>
<p class="red">This paragraph is red</p>
<p class="red" id="black">This paragraph is black</p>
</body>
```

Selectors

Pseudo-Elements

- Pseudo-elements create **abstractions** about the document tree beyond those specified by the document language.
 - For instance, document languages do not offer **mechanisms** to access the first letter or first line of an element's content.
- Pseudo-elements may also **provide style sheet designers a way to assign style to content** that does not exist in the source document (e.g., the `:before` and `:after` pseudo-elements give access to generated content).

Selectors

Pseudo-Elements

■ The :first-line pseudo-elements

- applies special styles to the contents of the first formatted line of a paragraph. For instance:

```
p:first-line { text-transform: uppercase }
```

■ The :first-letter pseudo-elements

- select the first letter of the first line of a block, if it is not preceded by any other content (such as images or inline tables) on its line.
 - The :first-letter pseudo-element may be used for "initial caps" and "drop caps", which are common typographical effects.

Selectors

Pseudo-Elements

- The **:before** and **:after** pseudo-elements
 - can be used to insert generated content before or after an element's content. For instance:

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

```
p.red { color: red }
```

```
p.red:before{content: "Note that: "}
```

```
#black { color: black }
```

```
#black:before { content: "Observe that: "}
```

```
</style> <title>Color</title> </head>
```

```
<body>
```

```
<p>Normal</p>
```

```
<p class="red">This paragraph is red</p>
```

```
<p class="red" id=black>This paragraph is black</p>
```

```
</body>
```

Selectors

Pseudo-Elements

- Pseudo-Element (example)

```
<head>
```

```
<title>Drop cap initial letter</title>
```

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

```
  p          { font-size: 12pt; line-height: 1.2 }
```

```
  p:first-letter { font-size: 200%; font-style: italic;  
                  font-weight: bold; float: left }
```

```
  span       { text-transform: uppercase }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p><span>The first</span> few words of an article in The  
Economist.</p>
```

```
</body>
```

Selectors

Pseudo-Classes

- Pseudo-classes classify elements on characteristics other than their name, attributes or content; in principle characteristics that cannot be deduced from the document tree.
- Pseudo-classes may be dynamic, in the sense that an element may acquire or lose a pseudo-class while a user interacts with the document.
 - The exceptions are ':first-child', which can be deduced from the document tree, and ':lang()', which can be deduced from the document tree in some cases

Selectors

Pseudo-Classes

- The **:first-child pseudo-class** matches an element that is the first child element of some other element.

```
<style type="text/css">
  div > p:first-child { text-indent: 10pt } /*add indentation */
</style>                               /*for the first paragraph of a div */
</head>
<body>
  <div class="note">
    <p>The first P inside the note.</p>
    <p>The second P inside the note.</p>
  </div>
</body>
```



Selectors

Pseudo-Classes

- The `:link` and `:visited` pseudo-classes
 - The `:link` pseudo-class applies for links that have not yet been visited.
 - The `:visited` pseudo-class applies once the link has been visited by the user.

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

```
a:visited { color: fuchsia; text-decoration: none }
```

...

```
<a href = "pagina.html">Prova link</a>
```

Selectors

Pseudo-Classes

- The **:hover**, **:active**, and **:focus** pseudo-classes
 - The **:hover** pseudo-class applies while the user designates an element (with some pointing device), but does not activate it.
 - A visual user agent could apply this pseudo-class when the cursor (mouse pointer) hovers over a box generated by the element (must be placed after the A:link and A:visited rules).
 - The **:active** pseudo-class applies while an element is being activated by the user.
 - For example, between the times the user presses the mouse button and releases it.
 - The **:focus** pseudo-class applies while an element has the focus (accepts keyboard events or other forms of text input).

```
a:hover { text-transform: uppercase }
```

```
a:active { color: red }
```

Selectors

Pseudo-Classes

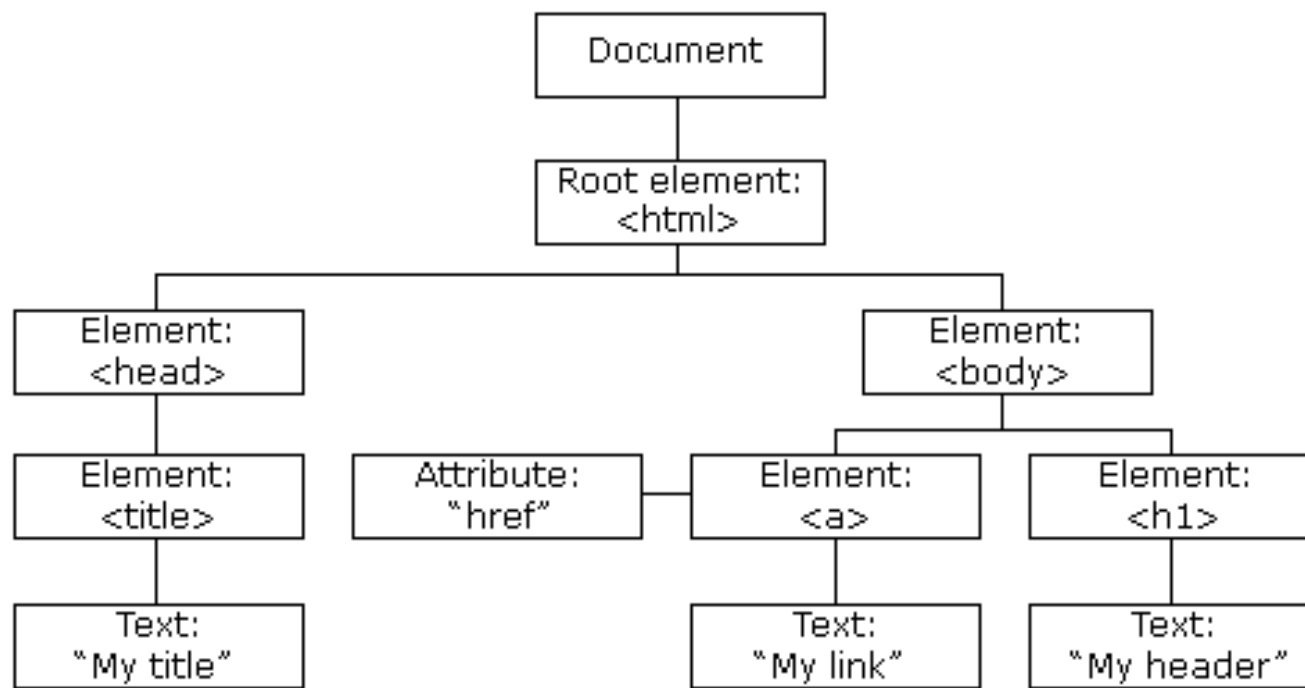
- The **:target** pseudo-class
 - URLs with an # followed by an anchor name link to a certain element within a document. The element being linked to is the target element.
 - The :target selector can be used to style the current active target element.

```
<style>
:target {
    border: 2px solid #D4D4D4;
    background-color: #e5eccc;
}
</style>
```



Inheritance

■ DOM tree (example)





Summary of the selector syntax

Pattern	Meaning	Example
E	Matches any E element (i.e., an element of type E).	h1 { font-family: sans-serif }
E F	Matches any F element that is a descendant of an E element.	h1 em { color: blue } <h1>This headline is very important</h1>
E > F	Matches any F element that is a child of an element E.	div ol>li p { color: blue }
E + F	Matches any F element immediately preceded by a sibling element E.	h1 + h2 { margin-top: -5mm }
E[attrib]	Matches any E element with the "attrib" attribute set (whatever the value).	h1[title] { color: blue; }

Summary of the selector syntax

Pattern	Meaning	Example
E[attrib="value"]	Matches any E element whose "attrib" attribute value is exactly equal to "value".	span[class="example"] { color: blue; }
E[attrib~="value"]	Matches any E element whose "attrib" attribute value is a list of space-separated values, one of which is exactly equal to "value".	a[rel~="copyright"] match the value "copyright copyleft copyeditor"
E[attrib ="value"]	Matches any E element whose "attrib" attribute has a hyphen-separated list of values beginning (from the left) with "value".	*[lang ="en"] { color : red } matches "en", "en-US", and "en-cockney"
E#myid	Matches any E element with ID equal to "myid".	h1#chapter1 { text-align: center }

Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
*	any element	Universal selector	2
E	an element of type E	Type selector	1
E[foo]	an E element with a "foo" attribute	Attribute selectors	2
E[foo="bar"]	an E element whose "foo" attribute value is exactly equal to "bar"	Attribute selectors	2
E[foo~="bar"]	an E element whose "foo" attribute value is a list of whitespace-separated values, one of which is exactly equal to "bar"	Attribute selectors	2
E[foo^="bar"]	an E element whose "foo" attribute value begins exactly with the string "bar"	Attribute selectors	3
E[foo\$="bar"]	an E element whose "foo" attribute value ends exactly with the string "bar"	Attribute selectors	3
E[foo*="bar"]	an E element whose "foo" attribute value contains the substring "bar"	Attribute selectors	3
E[foo ="en"]	an E element whose "foo" attribute has a hyphen-separated list of values beginning (from the left) with "en"	Attribute selectors	2



Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E:root	an E element, root of the document	<u>Structural pseudo-classes</u>	3
E:nth-child(n)	an E element, the n-th child of its parent	<u>Structural pseudo-classes</u>	3
E:nth-last-child(n)	an E element, the n-th child of its parent, counting from the last one	<u>Structural pseudo-classes</u>	3
E:nth-of-type(n)	an E element, the n-th sibling of its type	<u>Structural pseudo-classes</u>	3
E:nth-last-of-type(n)	an E element, the n-th sibling of its type, counting from the last one	<u>Structural pseudo-classes</u>	3
E:first-child	an E element, first child of its parent	<u>Structural pseudo-classes</u>	2



Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E:last-child	an E element, last child of its parent	<u>Structural pseudo-classes</u>	3
E:first-of-type	an E element, first sibling of its type	<u>Structural pseudo-classes</u>	3
E:last-of-type	an E element, last sibling of its type	<u>Structural pseudo-classes</u>	3
E:only-child	an E element, only child of its parent	<u>Structural pseudo-classes</u>	3
E:only-of-type	an E element, only sibling of its type	<u>Structural pseudo-classes</u>	3



Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E:empty	an E element that has no children (including text nodes)	<u>Structural pseudo-classes</u>	3
E:link	an E element being the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited)	<u>The link pseudo-classes</u>	1
E:visited			
E:active	an E element during certain user actions	<u>The user action pseudo-classes</u>	1 and 2
E:hover			
E:focus			
E:target	an E element being the target of the referring URI	<u>The target pseudo-class</u>	3
E:lang(fr)	an element of type E in language "fr" (the document language specifies how language is determined)	<u>The :lang() pseudo-class</u>	2
E:enabled	a user interface element E which is enabled or disabled	<u>The UI element states pseudo-classes</u>	3
E:disabled			



Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E:checked	a user interface element E which is checked (for instance a radio-button or checkbox)	The UI element states pseudo-classes	3
E::first-line	the first formatted line of an E element	The ::first-line pseudo-element	1
E::first-letter	the first formatted letter of an E element	The ::first-letter pseudo-element	1
E::before	generated content before an E element	The ::before pseudo-element	2
E::after	generated content after an E element	The ::after pseudo-element	2
E.warning	an E element whose class is "warning" (the document language specifies how class is determined).	Class selectors	1

Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E#myid	an E element with ID equal to "myid".	<u>ID selectors</u>	1
E:not(s)	an E element that does not match simple selector s	<u>Negation pseudo-class</u>	3
E F	an F element descendant of an E element	<u>Descendant combinator</u>	1
E > F	an F element child of an E element	<u>Child combinator</u>	2
E + F	an F element immediately preceded by an E element	<u>Adjacent sibling combinator</u>	2
E ~ F	an F element preceded by an E element	<u>General sibling combinator</u>	3

Cascade, Specificity and Inheritance



Cascade, specificity and inheritance

- The **Cascading term** indicates that style declarations cascade down to elements from many origins.
- The **cascade** computes a weight for each applicable declaration based on:
 - source order
 - importance
 - origin
 - specificity.
- This weight is used to determine exactly—and without conflict—which declaration should be applied to any given element.

The Cascade

Process of resolution

1. For a given property, the user agent finds all declarations that apply to a specific element for the target media type
 - by analysing three sources
 - the user agent
 - the author
 - the user style sheets (customized set of styles to use by default which some user agent allows a user to create).

The Cascade

Process of resolution

- Declarations are applied to the element, if
 - the associated selector **matches the element** **and**
 - the target medium **matches the media** list on **all @media rules** containing the declaration and on **all links** on the path through which the style sheet was reached.
- If there **is more than one applicable declaration** that sets a specific property, the cascade proceeds to step two.

The Cascade

Process of resolution

2. Sort the declarations according to their levels of importance, and origins.

- Declarations that are appended with the **!important** statement are called important declarations;
- Otherwise they are called normal declarations.

```
p { font-size: 1em !important; }
```

The Cascade

Process of resolution

2. Sort the declarations according to their levels of importance, and origins.

- Declarations are sorted in the following ascending order:
 1. Transition declarations
 2. Important user agent declarations
 3. Important user declarations
 4. Important override declarations
 5. Important author declarations
 6. Animation declarations
 7. Normal override declarations
 8. Normal author declarations
 9. Normal user declarations
 10. Normal user agent declarations

The Cascade

Process of resolution

3. Sort declarations with the same level of importance and origin by selector specificity.

- The specificity of a selector is represented by four comma-separated values a, b, c, d , where the values in "a" are the most important and those in "d" are least important.

The Cascade

Process of resolution

- A selector's specificity is calculated as follows:
 - $a = 1$ if the declaration is from a 'style' attribute rather than a rule with a selector (an inline style), $a = 0$ otherwise.
 - b = number of ID attributes in the selector.
 - c = number of other attributes and pseudo-classes in the selector.
 - d = number of element names and pseudo-elements in the selector.

The Cascade

Process of resolution

4. Finally, if declarations have the same level of importance, origin, and specificity, sort them by the order in which they are specified; the last declaration wins.
- Note: If the link to the external style sheet is placed after the internal style sheet in HTML <head>, the external style sheet will override the internal style sheet!

The Cascade

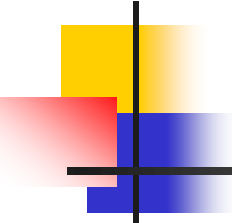
Process of resolution

Some examples

<code>* {}</code>	<code>/* a=0 b=0 c=0 d=0</code>
<code>li {}</code>	<code>/* a=0 b=0 c=0 d=1</code>
<code>li:first-line {}</code>	<code>/* a=0 b=0 c=0 d=2</code>
<code>ul li {}</code>	<code>/* a=0 b=0 c=0 d=2</code>
<code>ul ol+li {}</code>	<code>/* a=0 b=0 c=0 d=3</code>
<code>h1 + *[rel=up]{}</code>	<code>/* a=0 b=0 c=1 d=1</code>
<code>ul ol li.red {}</code>	<code>/* a=0 b=0 c=1 d=3</code>
<code>li.red.level {}</code>	<code>/* a=0 b=0 c=2 d=1</code>
<code>#x34y {}</code>	<code>/* a=0 b=1 c=0 d=0</code>
<code>style=""</code>	<code>/* a=1 b=0 c=0 d=0</code>

The Cascade

Process of resolution

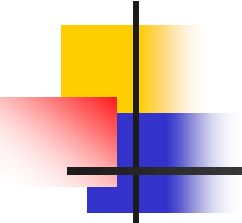


```
<style type="text/css">
  #redP { color: red }
  p.bluStyle {color:blue}
</style>
<title>Resolution Process</title>
</head>
<body>
<p id="redP" class="bluStyle" style="color:
  green"> Example 1 </p>
<p class="bluStyle"> Example 2 </p>
</body>
</html>
```



The Cascade

Process of resolution



```
<head>
  <style type="text/css">
    body {
      color: #000;
      background-color: #fff;
    }
    #wrap {
      font-size: 2em;
      color: #333;
    }
    div {
      font-size: 1em;
    }
  </style>
</head>
```

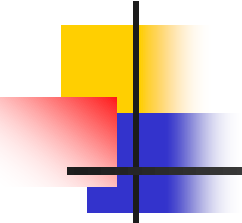
The Cascade

Process of resolution

```
em {  
  color: #666;  
}  
p.item {  
  color: #fff;  
  background-color: #ccc;  
  border-style: dashed;  
}  
p {  
  border: 1px solid black;  
  padding: 0.5em;  
}  
</style>  
</head>
```

The Cascade

Process of resolution



```
<body>  
  <div id="wrap">  
    <p>Normal Paragraph</p>  
    <p class="item">  
      This is the <em>cascade</em> in  
      <a href="#">action</a>  
    </p>  
  </div>  
</body>
```



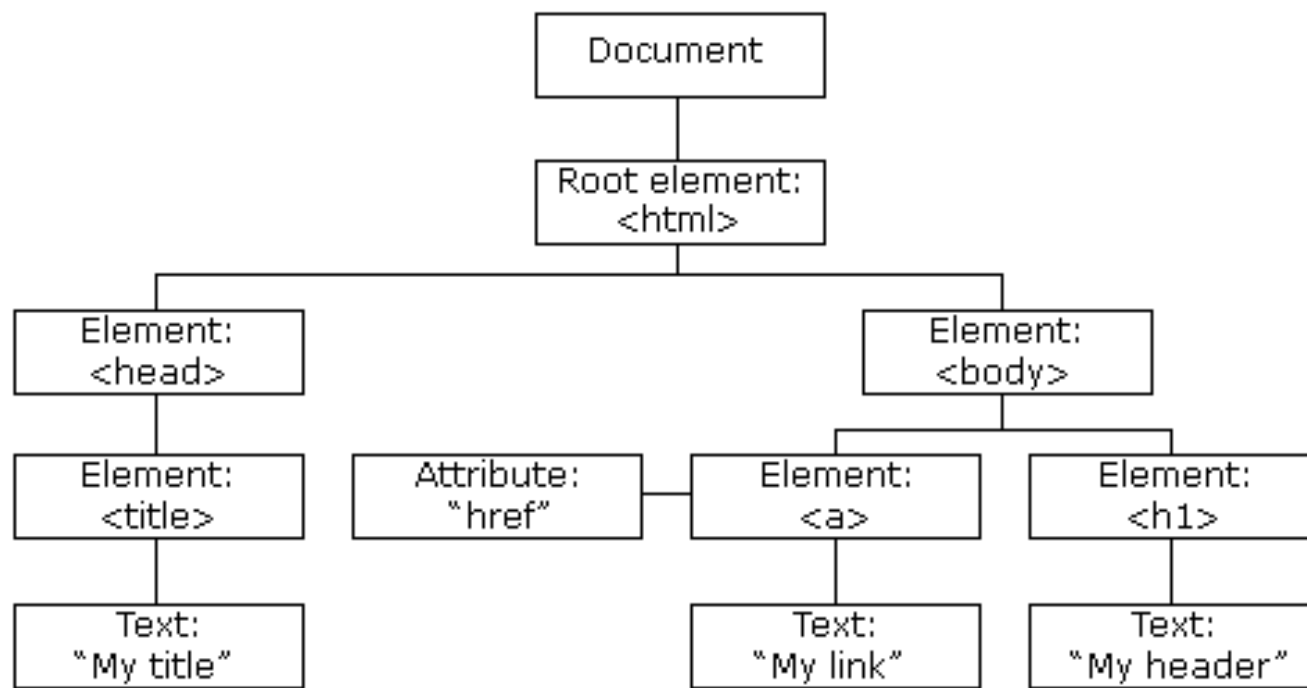


Inheritance

- Inheritance is the process by which properties are passed from parent to child elements even though those properties have not been explicitly defined by other means.
- Inheritance is a mechanism that is separate from the cascade: inheritance applies to the DOM (Document Object Model) tree, while the cascade deals with the style sheet rules.
- However, CSS properties set on an element via the cascade can be inherited by that element's child elements.

Inheritance

■ DOM tree (example)





Inheritance

```
<head>
<style type="text/css">
  body { color: red }   h2 { color: black }
</style>
  <title> Example  </title>
</head>
<body>
  <h1>Heading (red)</h1>
  <p>
    Text of the <b>paragraph</b> (the colour is inherited)
  </p>
  <h2> The style of heading 2 is redefined  </h2>
</body>
```





Cascading Style Sheets: Properties

Color

Foreground color

- Foreground color: the 'color' property

'color'

Value: <color> | inherit

Initial: depends on user agent

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

Computed value: as specified

```
em { color: red } /* predefined color name */
```

```
em { color: rgb(255,0,0) } /* RGB range 0-255 */
```

Color

Foreground color

- A `<color>` is either a keyword or a numerical RGB specification.
- The list of color keywords is: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow. These 17 colors have the following values

maroon #800000	red #ff0000	orange #ffa500	yellow #ffff00	olive #808000
purple #800080	fuchsia #ff00ff	white #ffffff	lime #00ff00	green #008000
navy #000080	blue #0000ff	aqua #00ffff	teal #008080	
black #000000	silver #c0c0c0	gray #808080		

Color

Foreground color

- **Hexadecimal notation:** a '#' immediately followed by either three or six hexadecimal characters.
 - The three-digit RGB notation (#rgb) is converted into six-digit form (#rrggbb) by replicating digits, not by adding zeros. This format allows defining 16777216 colors
- **Functional notation:** 'rgb(' followed by a comma-separated list of three numerical values (either three integer values or three percentage values) followed by ')'.
 - The integer value 255 corresponds to 100%, and to F or FF in the hexadecimal notation: `rgb(255,255,255) = rgb(100%,100%,100%) = #FFF`.
 - White space characters are allowed around the numerical values.

Color

Foreground color

■ Examples

```
body {color: black; background: white }
```

```
h1 { color: maroon }
```

```
h2 { color: olive }
```

```
em { color: #f00 }           /* #rgb */
```

```
em { color: #ff0000 }        /* #rrggbb */
```

```
em { color: rgb(255,0,0) }
```

```
em { color: rgb(100%, 0%, 0%) }
```

Color Opacity

- The **opacity property** sets the opacity level for an element.
- The opacity-level describes the transparency-level, where 1 is not transparent at all, 0.5 is 50% see-through, and 0 is completely transparent.

```
<head>
<style>
div {
  background-color: red;
  opacity: 0.2;
}
</style>
</head>
```



Color

Background color

- 'background-color'

Value: <color> | transparent | inherit

Initial: transparent

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

- This property sets the background color of an element.

```
h1 { background-color: #F00 }
```


Color

Background color

■ 'background-color' (example)

```
<head>
```

```
  <title> Background Color </title>
```

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

```
  body { background-color: #ccc }
```

```
  #father { background-color: teal }
```

```
  #son { background-color: red }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div id="father">
```

Father element

```
  <div id="son">
```

Son Element

```
</div>
```

```
</div>
```

```
</body>
```

Background Image

■ 'background-image'

Value: <uri> | none | inherit

Initial: none

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: absolute URI or none

- When setting a background image, authors should also specify a background color that will be used when the image is unavailable. When the image is available, it is rendered on top of the background color.
- By default, the image is repeated so it covers the entire element.

Background Image

- 'background-image' (example)

```
<head>
```

```
  <title> Background Image </title>
```

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

```
  body { background-color: #ccc; background-image: url("tower.jpg")}
```

```
  #father { background-color: teal; }
```

```
  #son { background-color: red; }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
  <div id="father">
```

Father element

```
    <div id="son">
```

Son Element

```
  </div>
```

```
  </div>
```

```
</body>
```



Background Repeat

■ 'background-repeat'

Value: repeat | repeat-x | repeat-y | no-repeat | space | round
| inherit

Initial: repeat

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: as specified

- If a background image is specified, this property specifies whether the image is repeated (tiled), and how.
- By default, the background-image property repeats an image both horizontally and vertically

Background Repeat

- 'background-repeat'

- repeat

- The image is repeated both horizontally and vertically.

- repeat-x

- The image is repeated horizontally only.

- repeat-y

- The image is repeated vertically only.

- no-repeat

- The image is not repeated: only one copy of the image is drawn.

Background Repeat

- 'background-repeat'

- space

The image is repeated as often as will fit within the background positioning area without being clipped and then the images are spaced out to fill the area.

- round

The image is repeated as often as will fit within the background positioning area. If it doesn't fit a whole number of times, it is rescaled so that it does.

Background Repeat

- 'background-repeat' (example)

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

```
body {
```

```
    background-color: #ccc; background-image: url("tower.jpg");
```

```
    background-repeat: space;
```

```
}
```

```
</style>
```



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

```
body {
```

```
    background-color: #ccc; background-image: url("tower.jpg");
```

```
    background-repeat: round;
```

```
}
```

```
</style>
```



Background Attachment

■ 'background-attachment'

Value: scroll | fixed | inherit

Initial: scroll

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: as specified

- If a background image is specified, this property specifies whether it is fixed with regard to the viewport ('fixed') or scrolls along with the containing block ('scroll').

Background Attachment

■ 'background-attachment' (example)

```
<head>
```

```
  <title>    Background attachment  </title>
```

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

```
  body { background-color: #ccc; background-image:
    url("tower.jpg");
```

```
    background-repeat: repeat-x;
```

```
    background-position: center;
```

```
    background-attachment: fixed;}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p>Text <br> <br><br><br><br><br> Text </p>
```

```
</body>
```

Fixed Scroll



Background Position

- 'background-position'

Value: [[<percentage> | <length> | left | center | right] [<percentage> | <length> | top | center | bottom]?] | [[left | center | right] || [top | center | bottom]] | inherit

Initial: 0% 0%

Applies to: all elements

Inherited: no

Percentages: refer to the size of the box itself

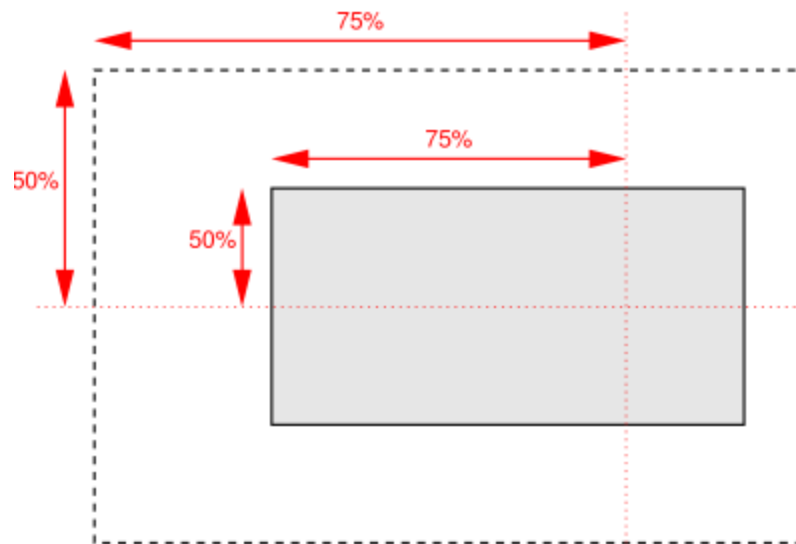
Media: visual

Computed value: for <length> the absolute value, otherwise a percentage

- If a background image has been specified, **this property specifies its initial position**. If only one value is specified, the second is assumed to be 'center'.

Background Position

- 'background-position'
 - **<percentage>** A percentage X aligns the point X% across (for horizontal) or down (for vertical) the image with the point X% across (for horizontal) or down (for vertical) the element's padding box.
 - For example, with a value pair of '75% 50%', the point 75% across and 50% down the image is to be placed at the point 75% across and 50% down the padding box.



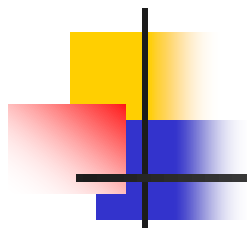
Background Position

- 'background-position'
 - **<length>** A length L aligns the top left corner of the image a distance L to the right of (for horizontal) or below (for vertical) the top left corner of the element's padding box.
 - For example, with a value pair of '2cm 1cm', the upper left corner of the image is placed 2cm to the right and 1cm below the upper left corner of the padding box.



Measurement units

- Relative units
 - **em**: the 'font-size' of the relevant font (preferred)
 - **ex**: the 'x-height' of the relevant font
- Absolute units
 - **in**: inches — 1 inch is equal to 2.54 centimeters.
 - **cm**: centimeters
 - **mm**: millimeters
 - **pt**: points — the points used by CSS 2.1 are equal to 1/72nd of an inch.
 - **pc**: picas — 1 pica is equal to 12 points.
 - **px**: pixels – 1/96 inch

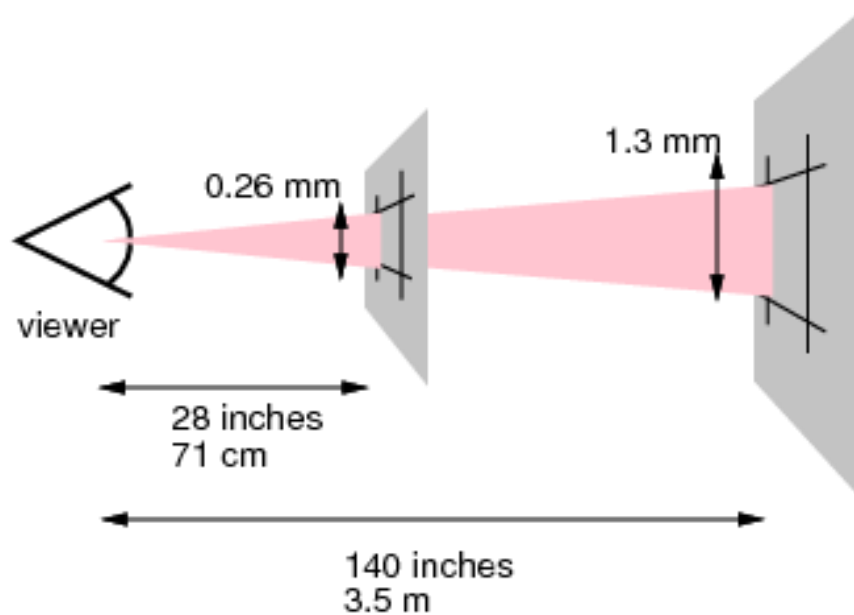


Relative Length

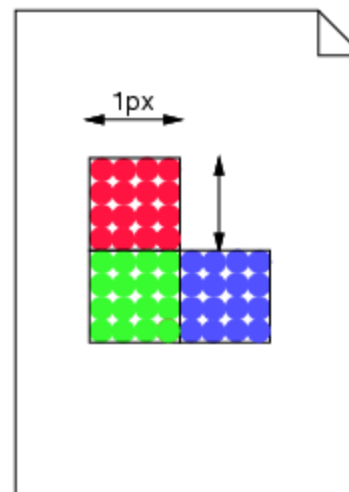
- It is recommended that the *reference pixel* be the visual angle of one pixel on a device with a pixel density of 96dpi and a distance from the reader of an arm's length.
- It is recommended that the *pixel unit refer to the whole number of device pixels that best approximates the reference pixel.*
- Relative units are:
 - For reading at arm's length, 1px thus corresponds to about 0.26 mm (1/96 inch).
 - When printed on a laser printer, meant for reading at a little less than arm's length (55 cm, 21 inches), 1px is about 0.20 mm.
 - On a 300 dots-per-inch (dpi) printer, that may be rounded up to 3 dots (0.25 mm); on a 600 dpi printer, it can be rounded to 5 dots.



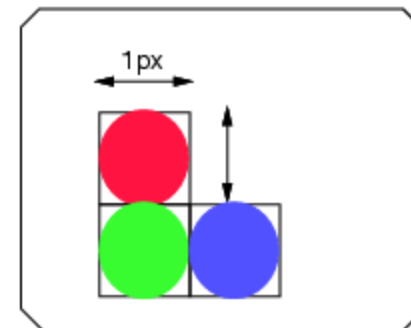
Relative Length



laserprint



monitor screen



 = 1 device pixel

Background Position

- 'background-position'
 - top
Equivalent to '0%' for the vertical position.
 - right
Equivalent to '100%' for the horizontal position.
 - bottom
Equivalent to '100%' for the vertical position.
 - left
Equivalent to '0%' for the horizontal position.
 - center
Equivalent to '50%' for the horizontal position if it is not otherwise given, or '50%' for the vertical position if it is.

Background Position

■ 'background-position' (examples)

```
body { background-image: url("tower.jpg");  
        background-repeat: no-repeat;  
        background-position: right top; /* 100% 0% */ }
```

```
body { background-image: url("tower.jpg");  
        background-repeat: no-repeat;  
        background-position: 0% 0%; /* 0% 0% */ }
```

```
body { background-image: url("tower.jpg");  
        background-repeat: no-repeat;  
        background-position: 100pt 50pt; /* 100 50 */ }
```

Background Clip

- 'background-clip'

Determines the background painting area, which determines the area within which the background is painted.

Values have the following meanings:

- 'border-box'

The background is painted within (clipped to) the border box.

- 'padding-box'

The background is painted within (clipped to) the padding box.

- 'content-box'

The background is painted within (clipped to) the content box.

Background Clip

- 'background-clip'

```
<style>
div {
  width: 300px;
  height: 300px;
  padding: 50px;
  background-color: yellow;
  background-clip: content-box;
  border: 2px solid #92b901;
}
</style>
```

content-box



padding-box



Background Origin

- 'background-origin'

The background-origin property specifies what the background-position property should be relative to.

Values have the following meanings:

'padding-box'

The position is relative to the padding box.

'border-box'

The position is relative to the border box.

'content-box'

The position is relative to the content box.



Background Size

- 'background-size'

The background-size property specifies the size of the background images. Size can be expressed in length and percentages. Further, other values are:

- 'cover'

Scale the background image to be as large as possible so that the background area is completely covered by the background image.

- 'contain'

Scale the image to the largest size such that both its width and its height can fit inside the content area





Background

■ 'background'

Value: <bg-image> || <position> [/ <bg-size>]? || <repeat-style> || <attachment> || <box> || <box>

Initial: see individual properties

Applies to: all elements

Inherited: no

Percentages: allowed on 'background-position'

Media: visual

Computed value: see individual properties

- The 'background' property **is a shorthand property** for setting the individual background properties

Text format

Alignment

■ 'text-align'

Value: left | right | center | justify | inherit

Initial: a nameless value that acts as 'left' if 'direction' is 'ltr', 'right' if 'direction' is 'rtl'

Applies to: block-level elements, table cells and inline blocks

Inherited: yes

Percentages: N/A

Media: visual

Computed value: the initial value or as specified

- This property describes **how inline content of a block is aligned.**

Text format Alignment

- 'text-align' (example)

```
<head>
```

```
  <title>    text format    </title>
```

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

```
  .centered { text-align: center }
```

```
  p { text-align: justify }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
  <h1 class="centered"> Centered text </h1>
```

```
  <p>justified text justified text justified text justified text
```

```
  justified text justified text justified text justified text </p>
```

```
</body>
```


Text format

Alignment

■ 'vertical-align'

Value: baseline | sub | super | text-top | text-bottom | middle
| top | bottom

Initial: baseline

Applies to: inline or table-cell box

Inherited: no

Percentages: line height

Media: visual

Computed value: the initial value or as specified

- This property describes **how inline content of a block is vertically aligned.**

Text format

Indentation

■ 'text-indent'

Value: <length> | <percentage> | inherit

Initial: 0

Applies to: block-level elements, table cells and inline blocks

Inherited: yes

Percentages: refer to width of containing block

Media: visual

Computed value: the percentage as specified or the absolute length

- This property specifies the indentation of the first line of text in a block.

Text format

'text-indent'

- 'text-indent' (example)

```
<head>
```

```
  <title> Text format </title>
```

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

```
  p { text-indent: 10px }
```

```
  div {text-indent: 10%; width: 200px; text-align: justify}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
  <p>Paragraph </p>
```

```
  <div>Div </div>
```

```
</body>
```

Text format

Decoration

■ 'text-decoration'

Value: none | [underline || overline || line-through || blink]
| inherit

Initial: none

Applies to: all elements

Inherited: no (see prose)

Percentages: N/A

Media: visual

Computed value: as specified

- This property describes decorations that are added to the text of an element using the element's color.

Text format Decoration

- 'text-decoration' (example)

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

```
.under { text-decoration: underline }
```

```
.over { text-decoration: overline }
```

```
.through { text-decoration: line-through }
```

```
</style>
```

```
...
```

```
<p>Normal</p>
```

```
<div class="under">Underlined
```

```
<p>underlined</p>
```

```
</div>
```

```
<p class="over">Overlined</p>
```

```
<p class="through">Line through</p>
```



Text format

Transformation

- 'text-transform'

Value: capitalize | uppercase | lowercase | none | inherit

Initial: none

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

Computed value: as specified

- This property controls capitalization effects of an element's text.

```
h1 { text-transform: uppercase }
```

Text format

Letter Spacing

- 'letter-spacing'

Value: normal | <length> | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

Computed value: 'normal' or absolute length

- This property specifies spacing behavior between text characters.

Text format

Letter Spacing

■ 'letter-spacing'

■ normal

The spacing is the **normal spacing for the current font**. This value allows the user agent to alter the space between characters in order to justify text.

■ <length>

This value indicates inter-character space in addition to the default space between characters. Values may be negative, but there may be implementation-specific limits. **User agents may not further increase or decrease the inter-character space in order to justify text.**

Text format

Letter Spacing

■ 'letter-spacing' (example)

```
<head>
```

```
<title>Text format</title>
```

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

```
    .wide { letter-spacing: 10px }
```

```
    .narrow { letter-spacing: -2px }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p>Normal</p>
```

```
<p class="wide">Text with large space</p>
```

```
<p class="narrow">Text with narrow space</p>
```

```
</body>
```

Text format

Word Spacing

■ 'word-spacing'

Value: normal | <length> | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

Computed value: for 'normal' the value '0'; otherwise the absolute length

- This property specifies spacing behavior between words.

Text format

Word Spacing

- 'word-spacing' (example)

```
<head>
```

```
<title>Text format</title>
```

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

```
    .wide { word-spacing: 10px }
```

```
    .narrow { word-spacing: -2px }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p>Normal text</p>
```

```
<p class="wide">Text with separate words</p>
```

```
<p class="narrow">Text with close words</p>
```

```
</body>
```

Text format

White Space

■ 'white-space'

Value: normal | pre | nowrap | pre-wrap | pre-line | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

Computed value: as specified

- This property declares how white space inside the element is handled.

Text format

White Space

■ 'white-space'

- **normal** - This value directs user agents to collapse sequences of white space, and break lines as necessary to fill line boxes.
- **pre** - This value prevents user agents from collapsing sequences of white space. Lines are only broken at newlines in the source.
- **nowrap** - This value collapses white space as for 'normal', but suppresses line breaks within text.
- **pre-wrap** - This value prevents user agents from collapsing sequences of white space. Lines are broken at newlines in the source as necessary to fill line boxes (not in Explorer).
- **pre-line** - This value directs user agents to collapse sequences of white space. Lines are broken at newlines in the source as necessary to fill line boxes.

Text format

White Space

■ 'white-space' (example)

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

```
  #first { white-space: pre; font-size:xx-large; }
```

```
  #second { white-space: nowrap; font-size:xx-large; }
```

```
  #third { white-space: pre-wrap; font-size:xx-large; }
```

```
  #fourth { white-space: pre-line; font-size:xx-large; }
```

```
</style>
```

```
..
```

```
<p>Here, the spaces      are collapsed</p>
```

```
<p id="first">Here, the spaces      are not collapsed</p>
```

```
<p id="second">Here, the spaces...    </p>
```

```
<p id="third">Here, the spaces  ...    </p>
```

```
<p id="fourth">Here, the spaces  ...    </p>
```

```
</body>
```

Character Style

Font Family

■ 'font-family'

Value: [[<family-name> | <generic-family>] [, <family-name> | <generic-family>]*] | inherit

Initial: depends on user agent

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

Computed value: as specified

- This property **allows choosing the family font.**

Character Style

Font Family

■ 'font-family'

- <generic family> - a group of font families with a similar look (like "Serif" or "Monospace")
- - a specific font family (like "Times New Roman" or "Arial")

Generic family	Font family	Description
Serif	Times New Roman Georgia	Serif fonts have small lines at the ends on some characters
Sans-serif	Arial Verdana	"Sans" means without - these fonts do not have the lines at the ends of characters
Monospace	Courier New Lucida Console	All monospace characters has the same width

Character Style

Font Family

■ 'font-family'

- The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font
 - Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.
- **Serif fonts** (stroke endings have ornamentations)
p {font-family:"Times New Roman", Times, serif;}
- **Sans-Serif fonts** (stroke endings are plain)
p {font-family:Arial, Helvetica, sans-serif;}
- **Monospace Fonts** (characters have the same fixed width)
p {font-family:"Courier New", Courier, monospace;}

Character Style

Font Size

■ 'font-size'

Value: <absolute-size> | <relative-size> | <length> | <percentage> | inherit

Initial: medium

Applies to: all elements

Inherited: yes

Percentages: refer to parent element's font size

Media: visual

Computed value: absolute length

- The font-size property sets the size of the text.
- If you do not specify a font size, the default size for normal text, like paragraphs, is 16px (16px=1em).

Character Style

Font Size

■ 'font-size'

- **<absolute-size>** - is an index to a table of font sizes computed and kept by the user agent. Possible values are:

[xx-small | x-small | small | medium | large | x-large | xx-large]

- **<relative-size>** - is interpreted relative to the table of font sizes and the font size of the parent element. Possible values are: [larger | smaller].

For example, if the parent element has a font size of 'medium', a value of 'larger' will make the font size of the current element be 'large'.

Character Style

Font Size

■ 'font-size' (example)

```
<head>
  <title> Text format </title>
  <style type="text/css">
h1 {font-size:300%;} /* best solution */
h2 {font-size:xx-large;}
p {font-size:2em;} /* em=16px */
em {font-size:larger}
  </style>
</head>
<body>
  <h1> This is <em>heading 1</em> </h1>
  <h2> This is heading 2 </h2>
  <p> This is a <em>paragraph</em>. </p>
</body>
</html>
```



Character Style

Font Style

■ 'font-style'

Value: normal | italic | oblique | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

Computed value: as specified

- Differences between italic and oblique are not often perceptible.

Character Style

Font Weight

■ 'font-weight'

Value: normal | bold | bolder | lighter | 100 | 200 | 300 | 400
| 500 | 600 | 700 | 800 | 900 | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

Computed value: see text

- The 'font-weight' property selects **the weight of the font**. The values '100' to '900' form an ordered sequence, where each number indicates a weight that is at least as dark as its predecessor.

Character Style

Font Weight

- **'font-weight'**
 - **'normal'** is synonymous with **'400'**
 - **'bold'** is synonymous with **'700'**
 - **'bolder'** selects the next weight that is assigned to a font that is darker than the inherited one.
 - **'lighter'** selects the next lighter keyword with a different font from the inherited one.
- There is no guarantee that there will be a darker face for each of the **'font-weight'** values; for example, some fonts may have only a normal and a bold face, while others may have eight face weights. There is no guarantee on how a UA will map font faces within a family to weight values.

Character Style

Font Weight

■ 'font-weight' (example)

```
<head>
```

```
  <title> Text format </title>
```

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

```
  div { font-weight: normal; border: thin solid red}
```

```
  div * {font-weight: bolder}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
  <p> Normal </p>
```

```
  <div> Normal
```

```
    <p> slightly bolder  </p>
```

```
  </div>
```

```
</body>
```



Character Style

Font Variant

- 'font-variant'

Value: normal | small-caps | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

Computed value: as specified

- In a small-caps font the lower case letters look similar to the uppercase ones, but in a smaller size and with slightly different proportions.

Character Style

Line Height

■ 'line-height'

Value: normal | <number> | <length> | <percentage> | inherit

Initial: normal

Applies to: all elements

Inherited: yes

Percentages: refer to the font size of the element itself

Media: visual

Computed value: for <length> and <percentage> the absolute value; otherwise as specified

- Specifies the minimal height between two text lines (in particular, between the baselines, without considering the letters such 'g' and 'q').

Character Style

Line Height

■ 'line-height'

- **<length>** The specified length is used in the calculation of the line box height. Negative values are illegal.
- **<number>** The used value of the property is this number multiplied by the element's font size. Negative values are illegal. The computed value is the same as the specified value.
- **<percentage>** The computed value of the property is this percentage multiplied by the element's computed font size. Negative values are illegal.

Character Style

Line Height

■ 'line-height' (example)

...

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

```
  div { width: 200px; border: thin solid; margin: 30px}
```

```
.par1 { font-size: 12pt; line-height: 250% }
```

```
.par2 { font-size: 12pt; line-height: 0.6 }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div>Normal flkmgnlkg sgksklgnszng lsgdknsdklgnsdgsg</div>
```

```
<div class="par1">New interline fddgdfgdhjkjhkhjk  
thgfhfghfghfghhgfhrgfhr </div>
```

```
<div class="par2"> New interline gdggfdgggdfg ddfgdgfhseuheh  
hherhhhtrhdf </div> </body>
```

Character Style

Font

■ 'font'

Value: [[<'font-style'> || <'font-variant'> || <'font-weight'>]? <'font-size'> [/ <'line-height'>]? <'font-family'>]
| caption | icon | menu | message-box | small-caption |
status-bar | inherit

Initial: see individual properties

Applies to: all elements

Inherited: yes

Percentages: see individual properties

Media: visual

Computed value: see individual properties

p { font: bold italic large Palatino, serif }

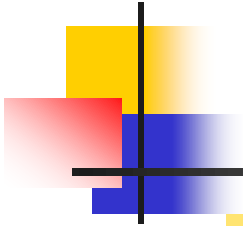
Character Style

Font

■ 'font'

- **caption** - The font used for captioned controls (e.g., buttons, drop-downs, etc.).
- **icon** - The font used to label icons.
- **menu** - The font used in menus (e.g., dropdown menus and menu lists).
- **message-box** - The font used in dialog boxes.
- **small-caption** - The font used for labeling small controls.
- **status-bar** - The font used in window status bars.

button p { font: menu }



Box Model

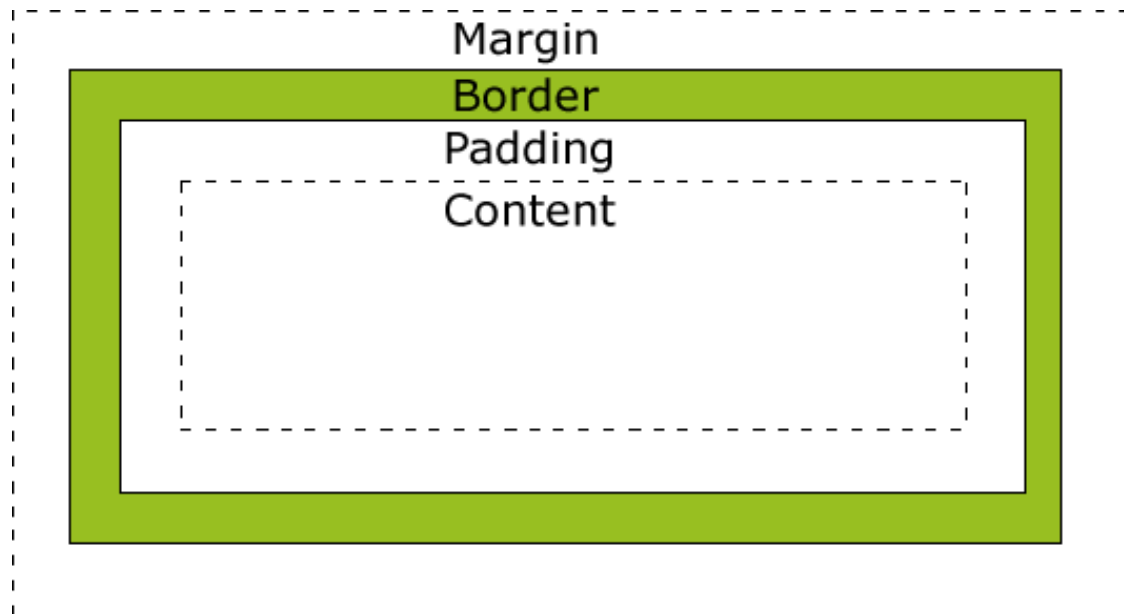


Block and Inline Elements

- A **block element** is an element that takes up the full width available, and has a line break before and after it.
- Examples of block elements:
 - <h1>
 - <p>
 - <div>
- An **inline element** only takes up as much width as necessary, and does not force line breaks.
- Examples of inline elements:
 -
 - <a>

Box Model

- All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.
- The CSS box model is essentially a box that wraps around HTML elements, and it consists of: margins, borders, padding, and the actual content.





Box Model

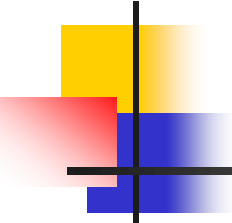
- **Margin** - does not have a background color, and it is completely transparent
- **Border** – is affected by the background color of the box
- **Padding** – is affected by the background color of the box
- **Content** - The content of the box, where text and images appear

Note: When you specify the width and height properties of an element with CSS, **you are just setting the width and height of the content area.**

Total element width = width + left padding + right padding + left border + right border + left margin + right margin

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

Box Model Example



```
<head>
  <title> Text format </title>
<style type="text/css">
  div
  { width:220px; padding:10px; border:5px solid gray; margin:30px
    10px;}
</style>
</head>
<body>
  <div>This is a box. Total width = 270px </div>
  <div>This is a box. Total width = 270px </div>
</body>
```



Box Model

Padding

- 'padding-top', 'padding-right', 'padding-bottom', 'padding-left'

Value: <padding-width> | inherit

Initial: 0

Applies to: all elements except table-row-group, table-header-group, table-footer-group, table-row, table-column-group and table-column

Inherited: no

Percentages: refer to width of containing block

Media: visual

Computed value: the percentage as specified or the absolute length

- Values for padding cannot be negative

Box Model

Padding

- 'padding-top', 'padding-right', 'padding-bottom', 'padding-left'

<padding-width>

- <length>

Specifies a fixed width.

- <percentage>

The percentage is calculated with respect to the width of the generated box's containing block, even for 'padding-top' and 'padding-bottom'.

Box Model Padding

- 'padding'

Value: <padding-width>{1,4} | inherit

Initial: see individual properties

Applies to: all elements except table-row-group, table-header-group, table-footer-group, table-row, table-column-group and table-column

Inherited: no

Percentages: refer to width of containing block

Media: visual

Computed value: see individual properties

- This property is a shorthand property for setting 'padding-top', 'padding-right', 'padding-bottom', and 'padding-left' at the same place in the style sheet.

Box Model Padding

■ 'padding'

- If there is **only one value**, it applies to all sides.
- If there are **two values**, the top and bottom paddings are set to the first value and the right and left paddings are set to the second.
- If there are **three values**, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third.
- If there are **four values**, they apply to the top, right, bottom, and left, respectively.

```
h1 {  
  background: white;  
  padding: 1em 2em;  
}
```

Box Model Padding

```
<style type="text/css">
  ul { padding: 3px 3px 3px 3px }
  li.pad30 {padding: 30px 30px 30px 30px}
          li.pad60 {padding: 60px 60px 60px 60px}
</style></head>
<body><ul>
<li>First element of list</li>
<li class="pad30">Second element with padding
  30</li>
<li class="pad60">Second element with padding
  60</li>
</ul></body>
```



Box Model

Border Width

- 'border-top-width', 'border-right-width', 'border-bottom-width', 'border-left-width'

Value: <border-width> | inherit

Initial: medium

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: absolute length; '0' if the border style is 'none' or 'hidden'

- These properties set the width of the top, right, bottom, and left border of a box.

Box Model

Border Width

- 'border-top-width', 'border-right-width', 'border-bottom-width', 'border-left-width'
 - <border-width>
 - thin - a thin border.
 - medium – a medium border.
 - thick - a thick border.
 - <length> - the border's thickness has an explicit value.
Explicit border widths cannot be negative.

The interpretation of the first three values depends on the user agent. The following relationships must hold, however:
'thin' <= 'medium' <= 'thick'.

Box Model

Border Width

- **'border-width'**

Value: <border-width>{1,4} | inherit

Initial: see individual properties

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: see individual properties

- This property is a shorthand property for setting 'border-top-width', 'border-right-width', 'border-bottom-width', and 'border-left-width' at the same place in the style sheet.

Box Model

Border Width

■ 'border-width'

- If there is only **one value**, it applies to all sides.
- If there are **two values**, the top and bottom borders are set to the first value and the right and left are set to the second.
- If there are **three values**, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third.
- If there are **four values**, they apply to the top, right, bottom, and left, respectively.

```
h1 { border-width: thin }
```

```
/* thin thin thin thin */
```

```
h1 { border-width: thin thick }
```

```
/* thin thick thin thick */
```

Box Model

Border Color

- 'border-top-color', 'border-right-color', 'border-bottom-color', 'border-left-color'

Value: <color> | transparent | inherit

Initial: the value of the 'color' property

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: when taken from the 'color' property, the computed value of 'color'; otherwise, as specified

- <color> - Specifies a color value.
- transparent - The border is transparent (though it may have width).

Box Model

Border Color

- **'border-color'**

Value: [<color> | transparent]{1,4} | inherit

Initial: see individual properties

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: see individual properties

- The 'border-color' property can have from one to four values, and the values are set on the different sides as for 'border-width'.

div {border-color: red yellow blue green}

Box Model

Border Style

- 'border-top-style', 'border-right-style', 'border-bottom-style', 'border-left-style'

Value: <border-style> | inherit

Initial: none

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: as specified

- The border style properties specify the line style of a box's border (solid, double, dashed, etc.).

Box Model

Border Style

- 'border-top-style', 'border-right-style', 'border-bottom-style', 'border-left-style'

<border-style>

- **none** - no border; the computed border width is zero.
- **hidden** - same as 'none', except in terms of border conflict resolution for table elements.
- **dotted** - the border is a series of dots.
- **dashed** - the border is a series of short line segments.
- **solid** - the border is a single line segment.
- **double** - the border is two solid lines. The sum of the two lines and the space between them equals the value of 'border-width'.
- **groove** - the border looks as though it were carved into the canvas.

Box Model

Border Style

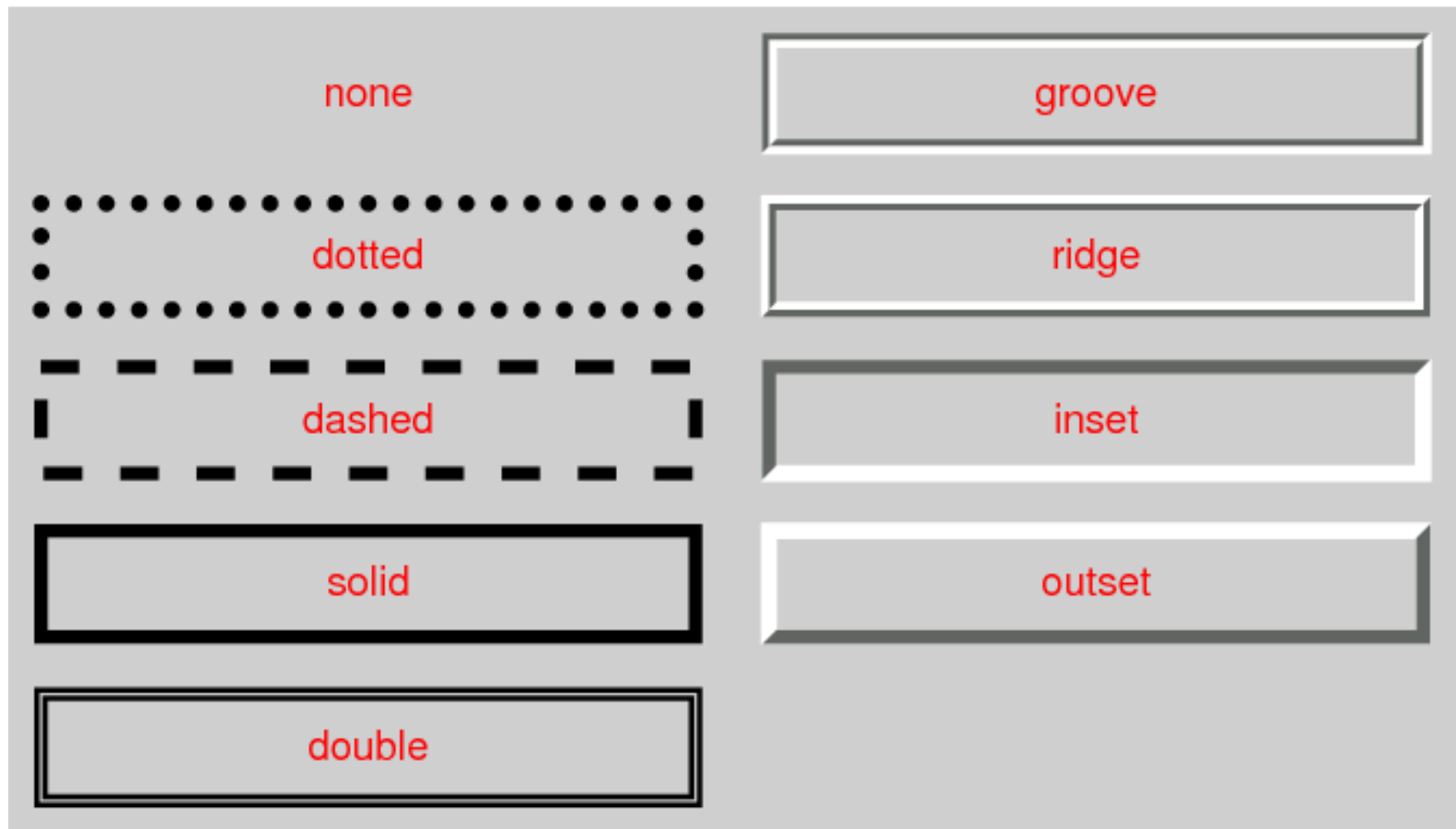
- 'border-top-style', 'border-right-style', 'border-bottom-style', 'border-left-style'

<border-style> (continued)

- **ridge** - the opposite of 'groove': the border looks as though it were coming out of the canvas.
- **inset** - the border makes the box look as though it were embedded in the canvas.
- **outset** - the opposite of 'inset': the border makes the box look as though it were coming out of the canvas.

Box Model

Border Style



Box Model

Border Style

- **'border-style'**

Value: <border-style>{1,4} | inherit

Initial: see individual properties

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: see individual properties

- The 'border-style' property **sets the style of the four borders**. It can have from one to four values, and the values are set on the different sides as for 'border-width' above.

#xy34 { border-style: solid dotted }

Box Model

Border Style

```
<style type="text/css">  
  div { border-style: inset; border-width: thick;  
        border-color: red; }  
  li {border-color: blue}  
  li.bor1 {border-style: groove; border-width:  
           medium medium medium medium}  
  li.bor2 {border-style: ridge; border-width:  
           thin thin thin thin}  
  
</style>  
</head>
```

Box Model

Border Style

<body>

<div>

Element without specific border

<li class="bor1">Element with groove border

Element without specific border

<li class="bor2">Element with ridge border

</div>

</body>

Box Model

Border

- 'border-top', 'border-right', 'border-bottom', 'border-left'

Value: [<border-width> || <border-style> || <'border-top-color'>] | inherit

Initial: see individual properties

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: see individual properties

- This is a shorthand property for setting the width, style, and color of the top, right, bottom, and left border of a box.
h1 { border-bottom: thick solid red }

Box Model

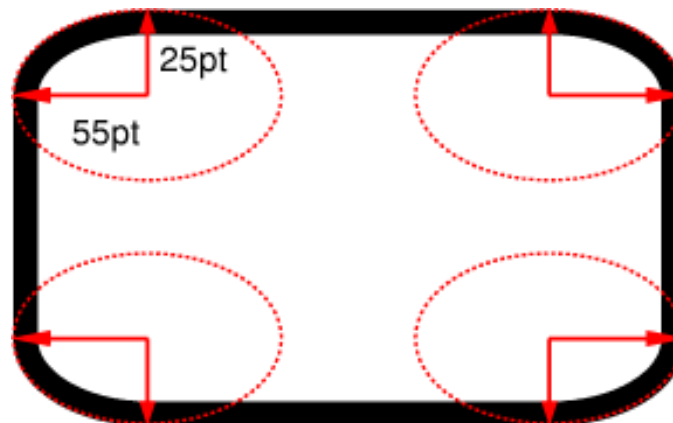
Border radius

- border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius
- **Value:** [<length> || <percentage>] | inherit
 - Initial:** 0
 - Applies to:** all elements
 - Inherited:** no
 - Percentages:** Refer to corresponding dim. of the border box
 - Media:** visual
 - Computed value:** two absolute <length> or percentages

Box Model

Border radius

- The two length or percentage values of the 'border-*-radius' properties define the radii of a quarter ellipse that defines the shape of the corner of the outer border edge (see the diagram below). The first value is the horizontal radius, the second the vertical radius.



Box Model

Border radius

Example:

```
div {  
  border: 2px solid #a1a1a1;  
  padding: 10px 40px;  
  background: #dddddd;  
  width: 300px;  
  border-radius: 25px;  
}
```



Box Model

Border image

- **border-image**
- **Value:** <'border-image-source'> || <'border-image-slice'> [/ <'border-image-width'> | / <'border-image-width'>? / <'border-image-outset'>]? || <'border-image-repeat'>
Initial: See individual properties
Applies to: See individual properties
Inherited: no
Percentages: N/A
Media: visual
Computed value: See individual properties

Allows specifying that an image can be used in place of the border styles.

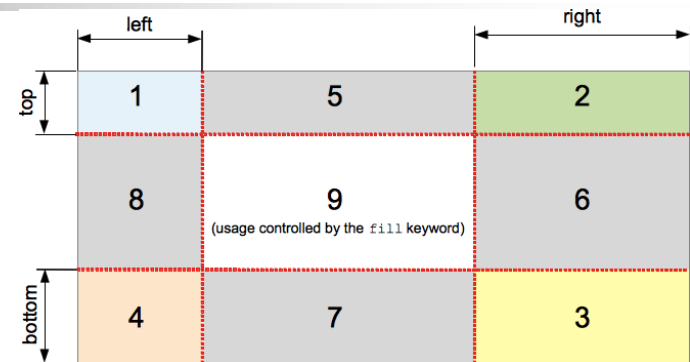
Box Model

Border image

Example:

```
<style>
div {
  background-color: lightgrey;
  border: 30px solid transparent;
  border-image: url('border.png');
  border-image-slice: 30;
  border-image-repeat: repeat;
}
</style>
```

it creates all four slices
at the same distance
from their respective
sides



- Zones 1-4 are corner regions. Each one is used a single time to form the corners of the final border image.
- Zones 5-8 are edge regions. These are repeated, scaled, or otherwise modified in the final border image to match the dimensions of the element.
- Zone 9 is the middle region. It is discarded by default, but is used like a background image if the keyword `fill` is set.



Box Model

Box shadow

- **box-shadow**

Value: none | <shadow> [, <shadow>]*

Initial: none

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: any <length> made absolute; any specified color computed; otherwise as specified

<shadow> = inset? && <length>{2,4} && <color>?

The 'box-shadow' property attaches one or more drop-shadows to the box.

Box Model

Box shadow

- 1st <length>

Specifies the horizontal offset of the shadow. A positive value draws a shadow that is offset to the right of the box, a negative length to the left.

- 2nd <length>

Specifies the vertical offset of the shadow. A positive value offsets the shadow down, a negative one up.

- 3rd <length>

Specifies the blur radius. Negative values are not allowed. If the blur value is zero, the shadow's edge is sharp. Otherwise, the larger the value, the more the shadow's edge is blurred.

- 4th <length>

Specifies the spread distance. Positive values cause the shadow to expand in all directions by the specified radius. Negative values cause the shadow to contract.

Box Model

Box shadow

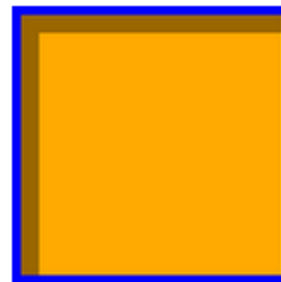
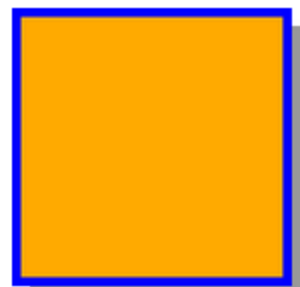
```
border:5px solid blue;  
background-color:orange;  
width: 144px;  
height: 144px;
```

```
box-shadow:  
  rgba(0,0,0,0.4)  
  10px 10px;
```

```
box-shadow:  
  rgba(0,0,0,0.4)  
  10px 10px  
  inset
```

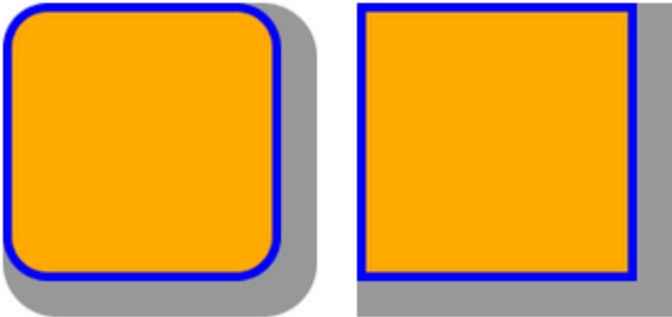
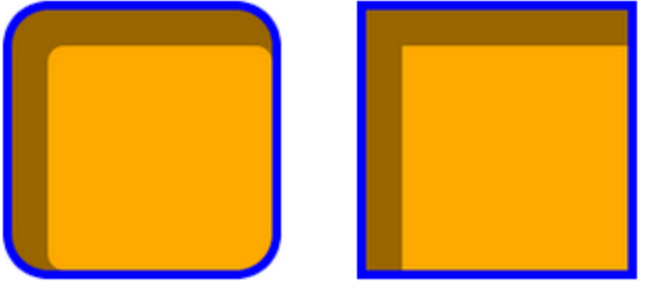
```
border-radius: 20px;
```

```
border-radius: 0;
```



Box Model

Box shadow

<pre>border:5px solid blue; background-color:orange; width: 144px; height: 144px;</pre>	<pre>border-radius: 20px; border-radius: 0;</pre>
<pre>box-shadow: rgba(0,0,0,0.4) 10px 10px 0 10px /* spread */</pre>	
<pre>box-shadow: rgba(0,0,0,0.4) 10px 10px 0 10px /* spread */ inset</pre>	

Box Model

Margin

- 'margin-top', 'margin-bottom', 'margin-right', 'margin-left'

Value: <margin-width> | inherit

Initial: 0

Applies to: all elements except elements with table display types other than table-caption, table and inline-table

Inherited: no

Percentages: refer to width of containing block

Media: visual

Computed value: the percentage as specified or the absolute length

- These properties set the top, right, bottom, and left margin of a box.

```
h1 { margin-top: 2em }
```


Box Model

Margin

- 'margin-top', 'margin-bottom', 'margin-right', 'margin-left'
<margin-width>
 - <length> - specifies a fixed width.
 - <percentage> - the percentage is calculated with respect to the width of the generated box's containing block.
 - auto – the margins are automatically computed based on the values of width, height, padding and borders in such a way that all the spaces in the box is occupied.
- Note: adjoining margins (no non-empty content, padding or border areas) of two or more boxes (which may be next to one another or nested) combine to form a single margin corresponding to the maximum value (collapsing margins).

Box Model

Margin

- 'margin'

Value: <margin-width>{1,4} | inherit

Initial: see individual properties

Applies to: all elements except elements with table display types other than table-caption, table and inline-table

Inherited: no

Percentages: refer to width of containing block

Media: visual

Computed value: see individual properties

- The 'margin' property is a shorthand property for setting 'margin-top', 'margin-right', 'margin-bottom', and 'margin-left' at the same place in the style sheet.

Box Model Margin

■ 'margin'

- If there is only **one value**, it applies to all sides.
- If there are **two values**, the top and bottom margins are set to the first value and the right and left margins are set to the second.
- If there are **three values**, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third.
- If there are **four values**, they apply to the top, right, bottom, and left, respectively.

Box Model

Margin (Example)

■ 'margin' (example)

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

```
ul {
```

```
    background: yellow;
```

```
    margin: 12px 12px 12px 12px;
```

```
    padding: 3px 3px 3px 3px;
```

```
}
```

```
li {
```

```
    color: white;           /* text color is white */
```

```
    background: blue;       /* Content, padding will be blue */
```

```
    margin: 12px 12px 12px 12px;
```

```
    padding: 12px 0px 12px 12px; /* Note 0px padding right*/
```

```
    list-style: none         /* no glyphs before a list item */
```

```
}
```

Box Model

Margin (Example)

- 'margin' (example) (continued)

```
li.withborder {  
    border-style: dashed;  
    margin: 36px 36px 36px 36px;  
    border-width: medium; /* sets border width on all sides*/  
    border-color: lime;  
}
```

```
</style>
```

```
...
```

```
<ul>
```

```
<li>First element of list</li>
```

```
<li class="withborder">Second element of list is a bit longer to  
illustrate wrapping.</li>
```

```
</ul>
```



Box Model

Width

- 'width'

Value: <length> | <percentage> | auto | inherit

Initial: auto

Applies to: all elements but non-replaced inline elements, table rows, and row groups

Inherited: no

Percentages: refer to width of containing block

Media: visual

Computed value: the percentage or 'auto' as specified or the absolute length; 'auto' if the property does not apply

- This property specifies the **content width of boxes**

Box Model Height

- 'height'

Value: <length> | <percentage> | auto | inherit

Initial: auto

Applies to: all elements but non-replaced inline elements, table columns, and column groups

Inherited: no

Percentages: refer to the height of the generated box's containing block

Media: visual

Computed value: the percentage or 'auto' or the absolute length; 'auto' if the property does not apply

- This property specifies the content height

Box Model

Width and Height (Example)

- 'width' and 'height' (example)

```
li {
```

```
...
```

```
height: 200px
```

```
}
```

```
li.withborder {
```

```
..
```

```
height: 100px
```

```
}
```


Box Model

Positioning

- A box may be laid out according to three positioning schemes:
 - **Normal flow.** Normal flow includes block formatting of block boxes, inline formatting of inline boxes, relative positioning of block or inline boxes, and positioning of run-in boxes.
 - **Floats.** In the float model, a box is first laid out according to the normal flow, then taken out of the flow and shifted to the left or right as far as possible. Content may flow along the side of a float.
 - **Absolute positioning.** In the absolute positioning model, a box is removed from the normal flow entirely (it has no impact on later siblings) and assigned a position with respect to a containing block.

Box Model Positioning

- The '**position**' and '**float**' properties determine which of the positioning algorithms is used to calculate the position of a box.

- '**position**'

Value: static | relative | absolute | fixed | inherit

Initial: static

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: as specified

Box Model Positioning

- 'position'

- static

The box is laid out according to the normal flow. The 'top', 'right', 'bottom', and 'left' properties do not apply.

```
<head>
```

```
<title>Examples of positioning </title>
```

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

```
img {position: static;}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p>Example of image positioning </p>
```

```
<p>  </p>
```

```
<p>Example of image positioning</p>
```

```
</body>
```



Box Model Positioning

- 'position'

- relative

The box's position is calculated according to the normal flow (this is called the position in normal flow). Then the box is offset relative to its normal position.

When a box B is relatively positioned, the position of the following box is calculated as though B were not offset.

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

```
img {position: relative; left: 400px; top: 200px}
```

```
</style>
```

Box Model Positioning

- 'position'

- absolute

The box's position (and possibly size) is specified with the 'top', 'right', 'bottom', and 'left' properties. These properties specify offsets with respect to the first block containing the box in which a positioning is defined. **Absolutely positioned boxes are taken out of the normal flow.** This means they have no impact on the layout of later siblings. Also, though absolutely positioned boxes have margins, they do not collapse with any other margins.

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

```
img {position: absolute; left: 400px; top: 200px}
```

```
</style>
```

Box Model Positioning

- 'position'

- fixed

The position is calculated according to the 'absolute' model, but the box is fixed with respect to some reference.

- The margins do not collapse with any other margins.
- In the case of **handheld, projection, screen, tty, and tv media types**, the box is fixed with respect to the viewport and does not move when scrolled.
- In the case of the **print media type**, the box is rendered on every page, and is fixed with respect to the page box, even if the page is seen through a viewport (in the case of a print-preview, for example).
- For other media types, the presentation is undefined.

Note: fixed does not work in Explorer

Box Model

Positioning

- 'position'

- fixed (example)

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

```
img {position:static; margin:30px}
```

```
img.fix {position: fixed; left: 400px; top: 200px}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p></p>
```

```
<p></p>
```

```
<p></p>
```

```
<p></p>
```

```
<p>Example of image positioning</p>
```

```
</body>
```



Absolute Fixed

Box Model

Positioning - offset

- An element is said to be positioned if its 'position' property has a value other than 'static'. Positioned elements generate positioned boxes, laid out according to four properties:
- 'top', 'right', 'bottom', 'left'
 - Value:** <length> | <percentage> | auto | inherit
 - Initial:** auto
 - Applies to:** positioned elements
 - Inherited:** no
 - Percentages:** refer to edges of the containing block
 - Media:** visual

Box Model

Positioning – z-index

■ 'z-index'

Value: auto | <integer> | inherit

Initial: auto

Applies to: positioned elements

Inherited: no

Percentages: N/A

Media: visual

- For a positioned box, the 'z-index' property specifies:
 - The stack level of the box in the current stacking context.
 - Whether the box establishes a local stacking context.

Box Model

Positioning – z-index

■ 'z-index'

- **<integer>** - This integer is the stack level of the generated box in the current stacking context. The box also establishes a local stacking context in which its stack level is '0'.
- **Auto** - The stack level is the same as its parent's box. The box does not establish a new local stacking context.

Each box has an integer stack level, which is its position on the z-axis relative to other boxes in the same stacking context.

- Boxes with greater stack levels are always formatted in front of boxes with lower stack levels.
- Boxes may have negative stack levels.
- Boxes with the same stack level in a stacking context are stacked back-to-front according to document tree order.

Box Model

Positioning – z-index

- 'z-index' (example)

```
<head>
```

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

```
#father1 { position: absolute; left: 10px; top: 10px; width:  
300px; height: 200px; background-color: yellow; z-index: 3;}
```

```
#father2 { position: absolute; left: 50px; top: 50px; width:  
300px; height: 200px; background-color: red; z-index: 2;}
```

```
#son1 { position: absolute; z-index: 1}
```

```
#son2 { position: absolute; z-index: 4}
```

```
</style>
```

```
</head>
```

Box Model

Positioning – z-index

- 'z-index' (example) (continued)

```
<body>
<div id="father1">  Father 1
<p id="son1"> Son 1 </p>
</div>
<div id="father2">  Father 2
<p id="son2"> Son 2 </p>
</div>
</body>
```

Box Model

Float

- 'float'

Value: left | right | none | inherit

Initial: none

Applies to: all, but positioned elements

Inherited: no

Percentages: N/A

Media: visual

- This property specifies **whether a box should float to the left, right, or not at all**. It may be set for any element, but only applies to elements that generate boxes that are not absolutely positioned.
- **Elements are floated horizontally**, this means that an element can only be floated left or right, not up or down.

Box Model

Float

- 'float'

- left

The element generates a block box that is floated to the left. Content flows on the right side of the box, starting at the top (subject to the 'clear' property).

- right

Similar to 'left', except the box is floated to the right, and content flows on the left side of the box, starting at the top.

- none

The box is not floated.

The element has to have an explicit width

Box Model

Float

■ 'float' (example)

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

```
img { float: none } /* could be also omitted*/
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p> float: none </p>
```

```
<p></p>
```

```
<p> The text follows the normal flow </p>
```

```
</body>
```

```
img { float: left }
```



```
img { float: right }
```



Box Model

Clear

■ 'clear'

Value: none | left | right | both | inherit

Initial: none

Applies to: block-level elements

Inherited: no

Percentages: N/A

Media: visual

This property indicates which **sides of an element's box(es) may not be adjacent to an earlier floating box**. The 'clear' property does not consider floats inside the element itself or in other block formatting contexts.

Box Model

Clear

■ 'clear'

- left** - place the top border edge below the bottom outer edge of any left-floating boxes that resulted from elements earlier in the source document.
- right** - place the top border edge below the bottom outer edge of any right-floating boxes that resulted from elements earlier in the source document.
- both** - place the top border edge below the bottom outer edge of any right-floating and left-floating boxes that resulted from elements earlier in the source document.
- none** - no constraint on the box's position with respect to floats.

Box Model

Clear

- 'clear' (example)

```
<head>
```

```
  <title> Examples of positioning </title>
```

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

```
  img { float: right }
```

```
  p { clear: right }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
  <p> float: none </p>
```

```
    
```

```
  <p> The text follows the normal flow </p>
```

```
</body>
```



Visual Effects

Visibility

■ 'visibility'

Value: visible | hidden | collapse | inherit

Initial: visible

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

- The 'visibility' property specifies whether the boxes generated by an element are rendered.
 - Invisible boxes still affect layout (set the 'display' property to 'none' to suppress box generation altogether).
- This property may be used in conjunction with scripts to create dynamic effects.

Visual Effects

Visibility

- 'visibility'

- **visible** - the generated box is visible.
- **hidden** - the generated box is invisible (fully transparent, nothing is drawn), but still affects layout.
 - Descendants of the element will be visible if they have 'visibility: visible'.
- **collapse** –
 - In a table, it causes the entire row or column to be removed from the display, and the space normally taken up by the row or column to be made available for other content (It does not work in IE)
 - In elements other than rows, row groups, columns, or column groups, 'collapse' has the same meaning as 'hidden'.

Visual Effects

Visibility

- 'visibility' (example)

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

```
#im1 { position: static; top: 2in; left: 2in; width: 2in;  
      visibility: hidden; }
```

```
#im2 { position: static; top: 2in; left: 2in; width: 2in; }  
</style>
```

....

```
<div id="im1">
```

```

```

```
<p>Tower 1</p>
```

```
</div>
```

```
<div id="im2">
```

```

```

```
<p>Tower 2</p>
```

```
</div>
```

Visual Effects Overflow

■ 'overflow'

Value: visible | hidden | scroll | auto | inherit

Initial: visible

Applies to: non-replaced block-level elements, table cells, and inline-block elements

Inherited: no

Percentages: N/A

Media: visual

- This property specifies **whether content of a block-level element is clipped when it overflows the element's box**. It affects the clipping of all of the element's content except any descendant elements whose containing block is the viewport or an ancestor of the element.

Visual Effects Overflow

■ 'overflow'

- **visible** - the content is not clipped, i.e., it may be rendered outside the block box.
- **hidden** - the content is clipped and that no scrolling user interface should be provided to view the content outside the clipping region.
- **scroll** – indicates that the content is clipped and that if the user agent uses a scrolling mechanism that is visible on the screen (such as a scroll bar or a panner), that mechanism should be displayed for a box whether or not any of its content is clipped.
- **auto** - the behavior of the 'auto' value is user agent-dependent, but should cause a scrolling mechanism to be provided for overflowing boxes

Visual Effects Overflow

- 'overflow' (example)

```
<head>
```

```
<title>
```

```
  Examples of visual effects
```

```
</title>
```

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

```
  div {  overflow: hidden;  width: 100px; height: 45px;}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
  <div> The text too large results to be hidden </div>
```

```
</body>
```


Visual Effects

Cursor

■ 'cursor'

Value: [[<uri> ,]* [auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize | text | wait | help | progress]] | inherit

Initial: auto

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual, interactive

- This property specifies the **type of cursor to be displayed for the pointing device** when the cursor is upon the element

Visual Effects

Cursor

■ 'cursor'

- **auto** - the user agent determines the cursor to display based on the current context.
- **crosshair** - a simple crosshair (e.g., short line segments resembling a "+" sign).
- **default** - the platform-dependent default cursor. Often rendered as an arrow.
- **pointer** - the cursor is a pointer that indicates a link.
- **move** - indicates something is to be moved.
- **e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize** - indicate that some edge is to be moved. For example, the 'se-resize' cursor is used when the movement starts from the south-east corner of the box.

Visual Effects

Cursor

- 'cursor' (continued)

- **text** - indicates text that may be selected. Often rendered as an I-beam.
- **wait** - indicates that the program is busy and the user should wait. Often rendered as a watch or hourglass.
- **progress** - a progress indicator. The program is performing some processing, but is different from 'wait' in that the user may still interact with the program. Often rendered as a spinning beach ball, or an arrow with a watch or hourglass.
- **help** - help is available for the object under the cursor. Often rendered as a question mark or a balloon.

Visual Effects

Cursor

- 'cursor' (example)

```
<head>
```

```
<title>
```

```
  Examples of visual effects
```

```
</title>
```

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

```
  div {  cursor:text;  width: 100px; height: 45px;}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
  <div> The text too large results to be hidden </div>
```

```
</body>
```

Visual Effects Display

■ 'display'

Value: inline | block | list-item | run-in | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit

Initial: inline

Applies to: all elements

Inherited: no

Percentages: N/A

Media: all

- This property is used to change the behavior of the elements: for instance, it can be used to transform a paragraph into inline, that is, does not form a new block of content.

Visual Effects Display

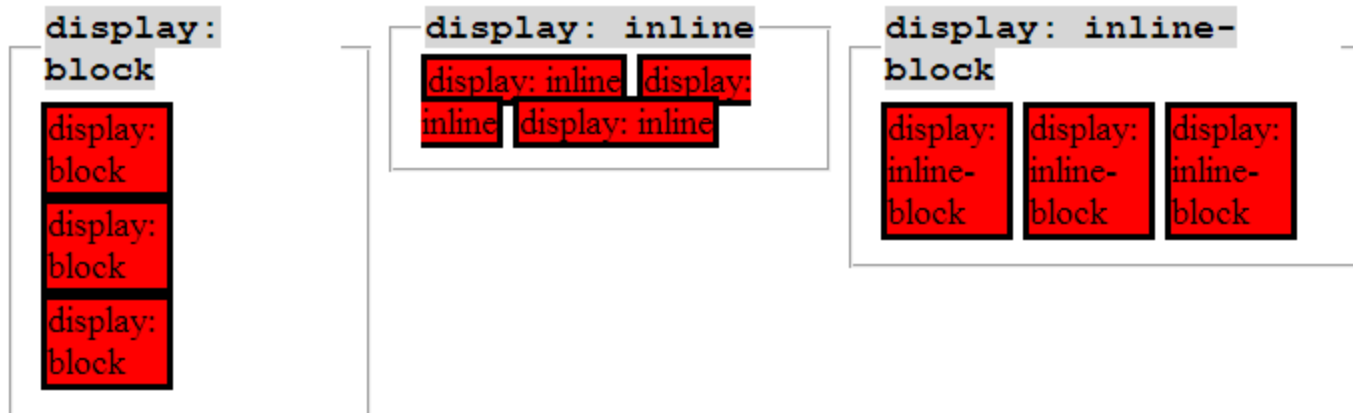
■ 'display'

- **block** - this value causes an element to generate a block box.
- **inline** - causes an element to generate one or more inline boxes.
- **inline-block** - causes an element to generate a block box, which itself is flowed as a single inline box, similar to a replaced element. The inside of an inline-block is formatted as a block box, and the element itself is formatted as an inline replaced element.
- **list-item** - causes an element (e.g., li in HTML) to generate a principal block box and a list-item inline box.

Visual Effects

Display

- `<div style="width: 50px" ...>` with different display



Visual Effects Display

- 'display' (continued)
 - **none** - causes an element to not appear in the formatting structure (i.e., in visual media the element generates no boxes and has no effect on layout).
 - Descendant elements do not generate any boxes either; the element and its content are removed from the formatting structure entirely.
 - This behavior cannot be overridden by setting the 'display' property on the descendants.
 - **run-in** - this value creates either block or inline boxes, depending on context. Properties apply to run-in boxes based on their final status (inline-level or block-level).

Visual Effects Display

■ display: run-in example

```
h2 {  
  font: normal 1.5em Georgia, serif;  
  color: #454545;  
  margin: 0 5px 5px 0;  
  display: run-in;  
  padding: 1px 1em;  
  background: #ffc;  
  border: 1px solid #999;  
  border-radius: 6px;  
}
```

<h2>...</h2>

<p>...</p>

Title

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Visual Effects Display

- 'display' (continued)
 - `table`, `inline-table`, `table-row-group`, `table-column`, `table-column-group`, `table-header-group`, `table-footer-group`, `table-row`, `table-cell`, and `table-caption` - cause an element to behave like a table element.
- Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow web standards

```
li {display:inline;}  
span {display:block;}
```



Navigation Bars

- Navigation bar from a standard HTML list

```
<head>
<style>
ul { list-style-type:none; margin:0; padding:0; }
</style>
</head>
```
- **list-style-type:none** - Removes the bullets. A navigation bar does not need list markers
- **Setting margins and padding to 0** to remove browser default settings



Navigation Bars

```
<style type="text/css">
  ul { list-style-type:none; margin:0; padding:0; }
</style>
</head>
<body>
<ul>
<li><a href="#home">Home</a></li>
<li><a href="#news">News</a></li>
<li><a href="#contact">Contact</a></li>
<li><a href="#about">About</a></li>
</ul>
</body>
</html>
```



Navigation Bars

```
<style type="text/css">
  ul { list-style-type:none; margin:0; padding:0; }
  li {display:inline}
</style>
</head>
<body>
<ul>
<li><a href="#home">Home</a></li>
<li><a href="#news">News</a></li>
<li><a href="#contact">Contact</a></li>
<li><a href="#about">About</a></li>
</ul>
</body>
</html>
```





Horizontal Navigation Bars

- Alternative solution

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

```
a {display:inline}
```

```
</style>
```

```
</head>
```

```
<body>
```

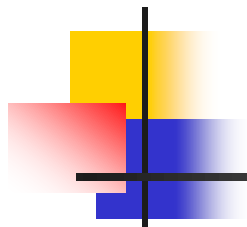
```
<p><a href="#home">Home</a>
```

```
<a href="#news">News</a>
```

```
<a href="#contact">Contact</a>
```

```
<a href="#about">About</a></p>
```

```
</body>
```



Vertical Navigation Bars

- To build a vertical navigation bar we only need to style the `<a>` elements, in addition to the code above:
`a { display:block;}`

