

Guida ai fogli di stile (Standard CSS 2)

Demis Ballis <demis@dimi.uniud.it>

Questo documento è liberamente tratto dalla guida ai CSS di base di Cesare Lamanna, disponibile nella sua versione integrale all'URL:

<http://css.html.it/guide/leggi/2/guida-css-di-base/>

Introduzione

Dietro il semplice acronimo CSS (**Cascading Style Sheets - Fogli di stile a cascata**) si nasconde uno dei fondamentali linguaggi standard del [W3C](#). La sua storia cammina su binari paralleli rispetto a quelli di HTML, di cui vuole essere l'ideale complemento. Da sempre infatti, nelle intenzioni degli uomini del Consortium, HTML, così come la sua recente evoluzione, XHTML, dovrebbe essere visto semplicemente come un linguaggio **strutturale**, alieno da qualunque scopo attinente la **presentazione** di un documento. Per questo obiettivo, ovvero arricchire l'aspetto visuale ed estetico di una pagina, lo strumento designato sono appunto i CSS. L'ideale perseguito da anni si può sintetizzare con una nota espressione: separare il contenuto dalla presentazione.

La prima specifica ufficiale di CSS ([CSS1](#)) risale al dicembre del 1996. Nel maggio 1998 è stata la volta della seconda versione: [CSS2](#). Niente stravolgimenti, ma molte aggiunte rispetto alla prima. **CSS2** non è altro che **CSS1** più alcune nuove proprietà, valori di proprietà e definizioni per stili non canonici come quelli rivolti alla stampa o alla definizione di contenuti audio. E' attualmente allo stato di Working Draft la nuova specifica [CSS3](#).

Classificazione degli elementi

La prima lezione di questa guida potrebbe spiazzarvi. Non parleremo di CSS, ma di (X)HTML. O meglio, riprenderemo alcuni aspetti di questo linguaggio che sono propedeutici per una migliore comprensione del meccanismo di funzionamento dei CSS. Sapere bene su che **cosa si interviene con i fogli di stile** è un passo necessario, visto che le cose di cui parleremo, specie con l'avvento e l'abuso degli editor visuali, sono spesso trascurate o misconosciute dai più. Se mi passate la metafora, possiamo dire che faremo come un bravo chirurgo che prima di imparare gli strumenti deve conoscere bene il corpo umano per operare con successo e senza fare danni. Inizieremo con la classificazione degli elementi.

Elementi blocco e elementi in linea

Osservate una pagina (X)HTML tentando di non pensare al contenuto ma solo alla sua struttura. Mettete in atto, insomma, una specie di processo di astrazione. Per aiutarvi ecco un'immagine:

The image shows a visual representation of an HTML document structure. It consists of several distinct rectangular boxes stacked vertically, representing different HTML elements. At the top is a large box labeled "Titolo 1". Below it is a smaller box labeled "Primo paragrafo" containing placeholder text. This is followed by another box labeled "Titolo 2". Below that is a box labeled "Secondo paragrafo" containing placeholder text and a small image of the map of Italy. At the bottom is a table with four cells, labeled "Cella 1" through "Cella 4".

Una pagina (X)HTML, per iniziare, non è altro che un insieme di rettangoli disposti sullo schermo di un monitor. Non importa che si tratti di paragrafi, titoli, tabelle o immagini: sempre di box rettangolari si tratta.

Nell'immagine potete però osservare che non tutti i box sono uguali. Alcuni contengono altri box al loro interno. Altri sono invece contenuti all'interno dei primi e se si trovano (come si trovano) in mezzo a del testo notate che esso scorre intorno senza interrompere il suo flusso e senza andare a capo. Avete nell'immagine la rappresentazione visiva, anche se un pò semplificata, della fondamentale distinzione tra gli elementi (X)HTML, quella tra elementi **blocco** ed elementi **inline**.

Gli elementi **blocco** sono i box che possono contenere altri elementi, sia di tipo blocco che di tipo **inline**. Quando un elemento blocco è inserito nel documento viene automaticamente creata una nuova riga nel flusso del documento. Provate a inserire in una pagina (X)HTML queste due righe di codice:

```
<h1>Titolo</h1> <p>Paragrafo</p>
```

Le parole "titolo" e "paragrafo" appariranno su due righe diverse, perchè **<H1>** e **<P>** sono elementi blocco.

Gli elementi **inline** non possono contenere elementi blocco, ma solo altri elementi inline (oltre che, ovviamente, il loro stesso tipo di contenuto, essenzialmente testo). Nell'immagine sono i rettangoli con il bordo verde. Come si può notare, quando sono inseriti nel documento non danno origine ad una nuova riga.

Una terza categoria è quella degli **elementi lista**. Comprende in pratica solo l'elemento **LI (list item)**.

Elementi rimpiazzati (replaced elements)

Un'altra distinzione da ricordare è quella tra **elementi rimpiazzati** ed **elementi non rimpiazzati**. I primi sono elementi di cui uno user agent (il "motore" e la mente di un browser) conosce solo le dimensioni intrinseche. Ovvero, quelli in cui altezza e larghezza sono definite dall'elemento stesso e non da ciò che lo circonda. L'esempio più tipico di elemento rimpiazzato è IMG (un'immagine). Altri elementi rimpiazzati sono: INPUT, TEXTAREA, select e OBJECT. Tutti gli altri elementi sono in genere considerati **non rimpiazzati**.

La distinzione è importante perchè per alcune proprietà è diverso il

trattamento tra l'una e l'altra categoria, mentre per altre il supporto è solo per la prima, ma non per la seconda.

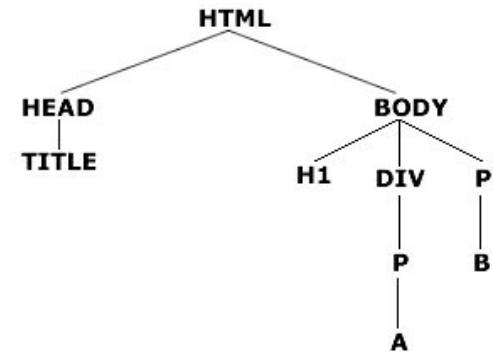
Struttura ad albero di un documento

Un altro concetto fondamentale che dovrete assimilare è quello della struttura ad albero di un documento. Il meccanismo fondamentale dei CSS è infatti **l'ereditarietà**. Esso fa sì che molte proprietà impostate per un elemento siano automaticamente ereditate dai suoi discendenti. Sapersi districare nella struttura ad albero significa padroneggiare bene questo meccanismo e sfruttare al meglio la potenza del linguaggio.

Presentiamo subito un frammento di codice HTML:

```
<html>
  <head>
    <title>Struttura del documento</title>
  </head>
  <body> <h1>Titolo</h1>
    <div> <p>Primo
      <a href="pagina.htm">paragrafo</a>
    </p>
  </div>
  <p>Secondo
    <b>paragrafo</b>
  </p>
</body>
</html>
```

Questa è la sua rappresentazione strutturale secondo il modello ad albero:



Il documento è in buona sostanza una perfetta forma di gerarchia ordinata in cui tutti gli elementi hanno tra di loro una relazione del tipo **genitore-figlio** (**parent-child** in inglese, imparate la dicitura perchè nei linguaggi come DOM o Javascript ci si riferisce agli ordini della gerarchia proprio con questi termini.). Ogni elemento è genitore e/o figlio di un altro.

Un elemento si dice **genitore** (**parent**) quando contiene altri elementi. Si dice **figlio** (**child**) quando è racchiuso in un altro elemento. In base a queste semplici indicazioni possiamo analizzare il nostro documento.

BODY, ad esempio è figlio di **HTML**, ma è anche genitore di **H1**, **DIV** e **P**. Quest'ultimo è a sua volta genitore di un elemento **B**.

Ovviamente, se osservate bene, potreste concludere che anche **BODY** è in qualche modo genitore di **B**. Non è esattamente così. Introduciamo ora un'altra distinzione, mutuata anch'essa dal linguaggio degli alberi genealogici, quella tra **antenato** (ingl: **ancestor**) e **discendente** (ingl: **descendant**). Semplice da capire. La relazione parent-child è valida solo se tra un elemento e l'altro si scende di un livello. Esattamente come in un albero familiare si indica la relazione tra padre e figlio. Pertanto possiamo dire che **HEAD** è figlio di **HTML**, che **A** è figlio di **P**, etc. tra **DIV** e **A**, invece si scende di due livelli: diciamo allora che **DIV** è un **antenato** di **A** e che questo è rispetto al primo un **discendente**.

C'è un solo elemento che racchiude tutti e non è racchiuso: **HTML**. Continuando con la metafora familiare potremmo dire che è il capostipite, ma in termini tecnici si dice che esso è **l'elemento radice** (ingl: **root**). E qui spaziamo il campo da un possibile fraintendimento:

l'elemento radice di un documento (X)HTML non è **BODY**. E che **HTML** non sia una semplice dichiarazione ma sia trattato alla stregua di qualunque altro elemento lo potete testare aprendo [questa pagina](#) con Explorer 6, Mozilla, Opera 6 o Explorer 5 Mac.

Come si vede abbiamo iniziato una guida dedicata a un linguaggio prettamente visuale tornando alle origini di HTML, ovvero al concetto di **struttura**. Non è un capriccio, è semplicemente dare il giusto inquadramento ai CSS: che sono nati per modificare lo stile di elementi strutturali e non vanno intesi come un linguaggio grafico. Per queste cose esistono Flash o Photoshop.

Inserire i fogli di stile in un documento

Iniziamo il nostro percorso dalle fondamentali nozioni di base, rimanendo ancora in parte nel territorio di (X)HTML. Se CSS è un solo linguaggio, vari sono i modi per inserire i fogli di stile CSS in un documento. Per capire il meccanismo è necessario chiarire la fondamentale distinzione tra fogli esterni e interni.

CSS esterni e interni

E' **esterno** un foglio di stile definito in un file separato dal documento. Si tratta di semplici documenti di testo editabili anche con il Blocco Note o TextEdit ai quali si assegna l'estensione **.css**.

Un foglio di stile si dice invece **interno** quando il suo codice è compreso in quello del documento. A seconda che si lavori con un CSS esterno o interno variano sintassi e modalità di inserimento. Rispetto a queste diverse modalità si parla di fogli di stile **collegati, incorporati o in linea**.

Fogli collegati

Per caricare un foglio esterno in un documento esistono due possibilità. La prima e più compatibile è quella che fa uso dell'elemento **<LINK>**. La dichiarazione va sempre collocata all'interno della sezione **<HEAD>** del documento (X)HTML:

```
<html> <head> <title>Inserire i fogli di stile in un documento</title> <link rel="stylesheet" type="text/css" href="stile.css"> </head> <body>...
```

L'elemento **<LINK>** presenta una serie di attributi di cui è importante spiegare significato e funzione:

1. rel: descrive il tipo di relazione tra il documento e il file collegato. E' **obbligatorio**. Per i CSS due sono i valori possibili: **stylesheet** e **alternate stylesheet**.

2. href: serve a definire l'URL assoluto o relativo del foglio di stile. E' **obbligatorio**.

3. type: identifica il tipo di dati da collegare. Per i CSS l'unico valore possibile è **text/css**. L'attributo è **obbligatorio**.

4. media: con questo attributo si identifica il supporto (schermo, stampa, etc) cui applicare un particolare foglio di stile. Attributo **opzionale**.

Usare @import

Un altro modo per caricare CSS esterni è usare la direttiva **@import** all'interno dell'elemento **<STYLE>**:

```
<style> @import url(stile.css); </style>
```

Questo sistema è uno dei modi più sicuri per risolvere problemi di compatibilità tra vecchi e nuovi browser. Ci torneremo quindi più avanti. Per il momento basti notare che il CSS va collegato definendo un URL assoluto o relativo da racchiudere tra parentesi tonde (ma vedremo che altri modi sono accettati) e che la dichiarazione deve chiudersi con un punto e virgola.

Fogli incorporati

I fogli incorporati sono quelli inseriti direttamente nel documento (X)HTML tramite l'elemento **<STYLE>**. Anche in questo caso la dichiarazione va posta all'interno della sezione **<HEAD>**:

```
<html> <head> <title>Inserire i fogli di stile in un documento</title> <style type="text/css"> body { background: #FFFACC; } </style> </head> <body>...
```

Come si vede il codice inizia con l'apertura del tag **<STYLE>**. Esso può avere due attributi:

1. type (obbligatorio)

2. media (opzionale)

per i quali valgono le osservazioni viste in precedenza. Seguono le regole del CSS e la chiusura di **</STYLE>**.

Fogli in linea

L'ultimo modo per formattare un elemento con un foglio di stile consiste nell'uso dell'attributo **style**. Esso fa parte della collezione di attributi (X)HTML definita **Common**: si tratta di quegli attributi applicabili a tutti gli elementi. La dichiarazione avviene a livello dei singoli tag contenuti nella pagina e per questo si parla di fogli di stile in linea. La sintassi generica è la seguente:

```
<elemento style="regole_di_stile">
```

Se, ad esempio, si vuole formattare un titolo **H1** in modo che abbia il testo di colore rosso e lo sfondo nero, scriveremo:

```
<h1 style="color: red; background: black;">...</h1>
```

Le cose da osservare nel codice sono due. Come valore di **style** si possono dichiarare più regole di stile. Esse vanno separate dal punto e virgola. I due punti si usano invece per introdurre il valore della proprietà da impostare.

Come è fatto un CSS: regole e commenti

Quanto visto finora riguarda essenzialmente il rapporto tra CSS e (X)HTML: tutti gli elementi, gli attributi e le funzionalità analizzate fanno parte della specifica del secondo linguaggio.

Con questa lezione entriamo nel vivo dell'argomento. Iniziamo con l'analisi degli elementi costitutivi di un foglio di stile. Un foglio di stile è composto da regole (ingl. **rules**) e commenti. Ecco, un foglio di stile non è altro che questo: un insieme di regole, accompagnate, volendo, da qualche nota di commento. Andiamo innanzitutto a spiegare cos'è e com'è fatta una regola.

Com'è fatta una regola



L'illustrazione mostra la tipica struttura di una regola CSS. Essa è composta da due blocchi principali:

il selettore

il blocco delle dichiarazioni

Il selettore serve a definire la parte del documento cui verrà applicata la regola. In questo caso, ad esempio, verranno formattati tutti gli elementi **<H1>**: lo sfondo sarà rosso, il colore del testo bianco. I selettori possono essere diversi e a queste varie tipologie dedicheremo una delle prossime lezioni. Per il momento sia chiara la funzione che essi svolgono.

Il blocco delle dichiarazioni è delimitato rispetto al selettore e alle altre regole da **due parentesi graffe**. Al suo interno possono trovare posto più dichiarazioni. Esse sono sempre composte da una coppia:

proprietà

valore

La proprietà definisce un aspetto dell'elemento da modificare (margini, colore di sfondo, etc) secondo il valore espresso. Proprietà e valore devono essere separati dai **due punti**. Una limitazione fondamentale da rispettare è questa: per ogni dichiarazione non è possibile indicare più di una proprietà, mentre è spesso possibile specificare più valori. Questa regola è pertanto errata:

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

Mentre questa è perfettamente valida e plausibile:

```
p {font: 12px Verdana, arial;}
```

Ancora, se in un blocco si definiscono più dichiarazioni, come nell'esempio in figura 1, esse vanno separate dal **punto e virgola**. Il linguaggio non impone che si metta il punto e virgola dopo l'ultima dichiarazione, ma alcuni browser più datati lo richiedono: aggiungetelo sempre per sicurezza e per una maggiore compatibilità.

Gli spazi bianchi lasciati all'interno di una regola non influiscono sul risultato. Il consiglio, anzi, è di lasciare sempre uno spazio tra le varie parti per una migliore leggibilità.

Commenti

Per inserire parti di commento in un CSS racchiudetelo tra questi segni:

`/*` come segno di apertura

`*/` come segno di chiusura

Proprietà singole e a sintassi abbreviata

Nelle definizioni delle regole è possibile fare uso di **proprietà singole e proprietà a sintassi abbreviata**. Con questa espressione traduciamo il termine inglese **shorthand properties** reso spesso, alla lettera, con il termine *scorciatoie*.

Le proprietà singole sono la maggior parte: impostano per un dato elemento o selettore un singolo aspetto. Con le **shorthand properties**, è possibile invece definire con una sola dichiarazione un insieme di proprietà. Chiariamo con un esempio.

Ogni elemento presenta sui suoi quattro lati un certo margine rispetto a quelli adiacenti. È possibile definire per ciascuno di essi un valore usando quattro proprietà singole separate:

margin-top

margin-right

margin-bottom

margin-left

La regola sarebbe questa:

```
div { margin-top: 10px; margin-right: 5px; margin-bottom: 10px; margin-left: 5px; }
```

Lo stesso risultato si può ottenere usando la proprietà a sintassi abbreviata **margin**:

```
div {margin: 10px 5px 10px 5px;}
```

Approfondiremo nel corso dell'analisi delle proprietà usi e costrutti sintattici di ciascuna. Per il momento ci limitiamo all'elenco:

```
background | border | border-top | border-right | border-bottom  
| border-left | cue | font | list-style | margin | outline |  
padding | pause |
```

I selettori

La parte preponderante della specifica CSS2 è dedicata all'analisi delle diverse proprietà in grado di definire l'aspetto visuale di elementi e sezioni di una pagina. Prima di tutto, però, è fondamentale capire come e a cosa queste proprietà possono essere assegnate. L'argomento sarà l'oggetto delle prossime quattro lezioni.

Fondamentalmente una regola CSS viene applicata ad un **selettore**. La parola parla da sé: si tratta di una semplice dichiarazione che serve a **selezionare** la parte o le parti di un documento soggette ad una specifica regola. Quella che segue è una lista commentata dei vari tipi di selettore. Per verificare i concetti abbiamo preparato per ciascun tipo un documento di esempio con codice e ulteriori spiegazioni.

Selettore di elementi (type selector)

È il più semplice dei selettori. È costituito da uno qualunque degli elementi di (X)HTML.

Sintassi

```
h1 {color: #000000;}  
  
p {background: white; font: 12px Verdana, arial, sans-serif;}  
  
table {width: 200px;}
```

Raggruppare

È possibile nei CSS raggruppare diversi elementi al fine di semplificare il codice. Gli elementi raggruppati vanno separati da una **virgola**.

Il raggruppamento è un'operazione molto conveniente. Pensate a questo scenario:

```
h1 {background: white;}  
h2 {background: white;}
```

```
h3 {background: white;}
```

Tutti e tre gli elementi hanno uno sfondo bianco. Invece di scrivere tre regole separate si può fare così:

```
h1, h2, h3 {background: white;}
```

Selettore universale (universal selector)

Anche nei CSS abbiamo un jolly. Il selettore universale serve a selezionare tutti gli elementi di un documento. Si esprime con il carattere * (asterisco).

Sintassi

```
* { color: black; }
```

Selettore del discendente (descendant selector)

Nella specifica CSS1 questo tipo era definito "selettore contestuale". Serve a selezionare tutti gli elementi che nella struttura ad albero di un documento siano **discendenti** di un altro elemento specificato nella regola. Ricordiamo che un elemento è discendente di un altro se è contenuto al suo interno, a qualsiasi livello.

Sintassi

```
div p {color: black;}  
p strong {color: red;}
```

Alcune considerazioni importanti di cui tenere conto. Il selettore va letto per chiarezza **da destra a sinistra**. Nel primo esempio verranno selezionati tutti i paragrafi (<p>) discendenti di elementi <div>. Nel secondo tutti gli elementi che si trovino all'interno di un paragrafo.

Fate attenzione alla struttura del documento ed evitate possibili incongruenze. Esistono regole ben precise sull'annidamento degli elementi che vanno rispettate sia in (X)HTML che nei CSS. Un paragrafo, per esempio, non può contenere un div, così come un elemento inline non può contenere elementi blocco.

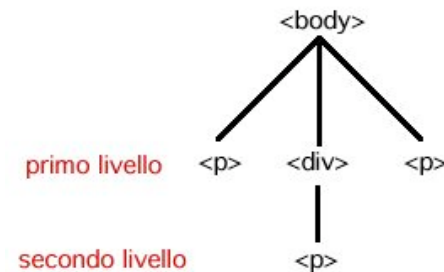
Selettore del figlio (child selector)

Seleziona un elemento che sia figlio diretto di un altro.

Il child selector è solo in apparenza simile al descendant selector. La differenza sta nella relazione di discendenza tra gli elementi, che in questo caso deve essere di primo livello. Chiariamo con un esempio utile anche per comprendere meglio il significato di descendant selector:

```
<body> <p>Primo paragrafo</p>  
      <div> <p>Secondo paragrafo</p> </div>  
      <p>Terzo paragrafo</p> </body>
```

Dei tre paragrafi solo il primo e il terzo sono figli diretti di body. Il secondo è invece figlio diretto di un elemento div. Tutti e tre, però, sono discendenti di body. L'immagine chiarisce ulteriormente il concetto. tra body e il primo e il terzo paragrafo si scende di un livello. Rispetto al secondo di due livelli:



Sintassi

```
body > p {color: black;}
```

Come si vede, un'ulteriore differenza sta nella sintassi. Per il child selector i due elementi devono essere separati dall'operatore "**maggiore di**" >. Anche in questo caso la lettura logica va fatta da destra a sinistra. Nell'esempio si selezionano tutti i paragrafi figli diretti dell'elemento body.

Questo selettore non è supportato da Explorer su Windows, cosa che ne limita notevolmente l'utilizzo.

Selettore dell'elemento adiacente (adjacent-sibling selector)

Un altro tipo di selettore introdotto con CSS2 è l'**adjacent-sibling selector**. Seleziona gli elementi che nel codice del documento siano immediatamente **vicini** (adiacenti) ad un altro.

Sintassi

Anche qui partiamo da un esempio:

```
<h1>Titolo</h1> <p>Primo paragrafo</p> <p>Secondo paragrafo</p>
```

Il primo paragrafo è adiacente al titolo h1, il secondo no e pertanto ad esso non potrà applicarsi questa regola:

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

In base a questa dichiarazione solo il primo dei due paragrafi avrà il testo rosso.

Il segno di assegnazione della relazione è per questo selettore **+**. L'adjacent-sibling selector non è supportato da Explorer Windows.

Selettore dell'attributo (attribute selector)

Seleziona gli elementi in base ai loro attributi e/o al valore di tali attributi. Il supporto di questo selettore non è diffuso. Il suo uso è però ricco di implicazioni positive consentendo una grande flessibilità nella selezione. L'utilità sarà ancora maggiore in linguaggi come XML.

Sintassi

Quattro sono i modi d'uso possibili.

Attributo semplice

```
elemento [ attributo ]
```

Con questa sintassi si selezionano tutti gli elementi che presentino nel codice un determinato attributo.

La dichiarazione:

```
a [ href ] {background: red;}
```

applicherà uno sfondo rosso a tutti gli elementi **a** per cui sia stato

impostato un attributo **href**, a prescindere dal valore ad esso assegnato.

Attributo con valore

```
elemento [ attributo = "valore" ]
```

Seleziona gli elementi che abbiano come valore dell'attributo la stringa definita nella regola. Pertanto:

```
a [ href = "http://www.uniud.it" ] { background: red; }
```

applicherà un sfondo rosso a tutti gli elementi **a** che abbiano come valore dell'attributo **href** "http://www.uniud.it". Come si vede una sintassi più restrittiva rispetto alla prima.

Attributo il cui valore contiene una stringa

```
elemento [ attributo t= "valore" ]
```

In questo caso vengono selezionati tutti gli elementi in cui il valore dell'attributo dichiarato **contenga** la stringa definita nella regola.

```
img [ alt t= "foto" ] {margin: 10px;}
```

La regola applicherà un margine di 10px a tutte le immagini in cui l'attributo **alt** contiene "foto". Quindi, saranno selezionate sia questa immagine:

```

```

sia questa:

```

```

Attributo il cui valore inizia con una stringa

```
elemento [ attributo |= "valore" ]
```

Seleziona gli elementi in cui il valore dell'attributo dichiarato **inizia** con la stringa definita nella regola. Esempio:

```
img [ alt |= "figura" ] {margin: 10px;}
```

selezionerà tutte le immagini in cui l'attributo **alt** inizia con la stringa "figura".

Id e classi: due selettori speciali

I CSS non sarebbero uno strumento così potente senza questi tipi di selettori. **Classi e ID** sono davvero una delle chiavi per sfruttare al meglio questo linguaggio.

Partiamo dall'inizio. In (X)HTML esistono due attributi fondamentali applicabili a tutti gli elementi: sono **class** e **id**. Dichiarare questi attributi a prescindere dai CSS non ha alcun senso e non modifica in alcun modo la presentazione della pagina. Ecco un esempio. Nel seguente codice (X)HTML abbiamo assegnato al paragrafo un attributo **class="testorosso"**:

```
<p class="testorosso">...</p>
```

Il valore dell'attributo **class** deve trovare una corrispondenza in un foglio di stile per avere l'effetto desiderato. Per esempio in un CSS contenente la seguente regola

```
.testorosso { font: 12px arial, Helvetica, sans-serif; color: #FF0000; }
```

Il testo del nostro paragrafo sarà ora formattato secondo i nostri desideri: testo rosso, carattere arial, dimensione di 12px.

Lo stesso meccanismo è valido per i selettori di tipo **ID**. Ma con una sola fondamentale differenza: è ad essa che dovete fare riferimento per scegliere se usare una classe o un ID. In un documento (X)HTML l'attributo **id** è usato per identificare in **modo univoco** un elemento. In pratica, se assegno ad un paragrafo l'id "testorosso", non potrò più usare questo valore nel resto della pagina. Di conseguenza, l'**ID #testorosso** dichiarato nel CSS trasformerà solo quel paragrafo specifico. Una singola classe, al contrario, può essere assegnata a più elementi, anche dello stesso tipo.

In un documento potrò avere senza problemi questa situazione:

```
<p class="testorosso">...</p>
<div class="testorosso">...</div>
<table class="testorosso">...</table>
<p class="testorosso">...</p>
```

La classe **.testorosso** presente nel CSS formatterà allo stesso modo il testo del paragrafo, del div e della tabella.

Ma non questa:

```
<p id="testorosso">...</p>
<div id="testorosso">...</div>
```

Concludendo: una classe consente di superare le limitazioni intrinseche nell'uso di un selettore di elementi. Se imposto questa regola:

```
p {color: red;}
```

tutti i paragrafi della mia pagina avranno il testo rosso. E se volessi diversificare? Avere, ad esempio, anche paragrafi con il testo nero? Sarei prigioniero della regola iniziale. Scrivo due classi, una per il rosso e una per il nero, le applico di volta in volta secondo le mie necessità e il gioco è fatto.

La strategia dovrà dunque essere questa. Se uno stile va applicato ad un solo specifico elemento usate un **ID**. Se invece prevedete di usarlo più volte ovvero su più elementi definite nel CSS una classe.

Chiariti i concetti di base, passiamo ad analizzare usi e sintassi.

Classe

Per definire una classe si usa far precedere il nome da un semplice punto:

```
.nome_della_classe
```

Questa è la sintassi di base. Un selettore classe così definito può essere applicato a tutti gli elementi di un documento (X)HTML.

Esiste un secondo tipo di sintassi:

```
<elemento>.nome_della_classe
```

Esso è più restrittivo rispetto alla sintassi generica. Se infatti definiamo questa regola:

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

lo stile verrà applicato solo ai paragrafi che presentino l'attributo **class="testorosso"**. Anche qui è importante stabilire un minimo di strategia. Il secondo tipo di sintassi va usato solo se pensate di applicare una classe ad uno specifico tipo di elemento (solo paragrafi o solo div, e così via). Se invece ritenete di doverla applicare a tipi diversi usate la sintassi generica.

Una terza possibile modalità è quella che prevede la dichiarazione di classi multiple:

```
p.testorosso.grassetto {color:red; font-weight:bold;}
```

Questa regola applicherà gli stili impostati a tutti gli elementi in cui siano presenti (in qualunque ordine) i nomi delle classi definiti nel selettore. Avranno dunque il testo rosso e in grassetto questi paragrafi:

```
<p class="grassetto testorosso maiuscolo">...</p> <p class="testorosso grassetto">...</p>
```

ma non questo, perchè solo uno dei nomi è presente come valore di **class**:

```
<p class="grassetto">...</p>
```

ID

La sintassi di un selettore **ID** è semplicissima. Basta far precedere il nome dal simbolo di cancelletto **#**:

```
#nome_id
```

Con questa regola:

```
#titolo {color: blue;}
```

assegniamo il colore blue all'elemento che presenti questa definizione:

```
<h1 id="titolo">...</h1>
```

Come per le classi è possibile usare una sintassi con elemento:

```
p#nome_id
```

In realtà questa modalità è assolutamente superflua. Se l'id è univoco non abbiamo alcun bisogno di distinguere l'elemento cui verrà applicata. Inoltre: la sintassi generica si rivela più razionale e utile. Se si dichiara un **ID** semplice è possibile assegnarlo a qualunque tipo di elemento. Posso usarlo su un paragrafo, ma se poi cambio idea posso passare tranquillamente ad un div senza dover modificare il foglio di stile. Usando la seconda sintassi, invece, sono costretto a rispettare l'elemento definito nel selettore.

Gestione del colore

Iniziamo la rassegna delle proprietà di CSS2 analizzando una di quelle che può a buon diritto dirsi fondamentale. Esamineremo prima le diverse possibilità di definire i valori dei vari colori, per poi analizzare usi e sintassi della proprietà **color**.

Definizione del colore

Parole chiave

Si tratta di 16 parole che definiscono i colori della palette VGA standard di Windows:

```
black | navy | blue | maroon | purple | green | red | teal |  
fuchsia | olive | gray | lime | aqua | silver | yellow | white
```

#RRGGBB

Le semplici 16 parole chiave non esauriscono ovviamente la gamma dei colori visualizzabili su un monitor moderno. Già in HTML era possibile impostare il colore di un elemento servendosi di **codici esadecimali**. In essi, le prime due lettere (o numeri) corrispondono ai valori per il colore rosso (**RED**), la seconda coppia fa riferimento al verde (**GREEN**), l'ultima al blue (**BLUE**).

Esempio

```
#CC0000
```

Il codice sopra rappresenta una tonalità di rosso.

#RGB

Molti dei codici esadecimali sono rappresentati da valori duplicati. E' possibile usare per essi una sintassi abbreviata in cui i valori per il rosso, il verde e il blue sono definiti solo dalla prima lettera o numero. Il colore dell'esempio precedente può essere definito anche così:

```
#C00
```

Nell'uso di questa sintassi vanno valutati i risultati con colori che non presentino coppie di valori duplicati. Il risultato può essere leggermente diverso a livello di tonalità a seconda dei casi.

rgb(rrr%, ggg%, bbb%)

Un altro modo per rappresentare i colori è quello di usare per i tre elementi base del sistema RGB una lista di valori in percentuale separati da una virgola. Per indicare il nero useremo, ad esempio:

```
rgb(0%, 0%, 0%)
```

Per il bianco:

```
rgb(100%, 100%, 100%)
```

rgb(rrr, ggg, bbb)

Ultima possibilità. Si definiscono i valori di rosso, verde e blue con tre valori compresi, rispettivamente, tra 0 e 255. Sistema ben noto a chi usa programmi di grafica. Il range 0-255 è l'equivalente decimale di quello esadecimale 00-FF visto in precedenza. Anche qui, i tre valori vanno separati da una virgola. Questo è il codice del nero:

```
rgb(0, 0, 0,)
```

La proprietà color

Visti i sistemi per rappresentare i colori, dobbiamo ora chiarire un aspetto importante. Per ogni elemento si possono definire almeno tre colori:

il colore di primo piano (foreground color)

il colore di sfondo (background color)

il colore del bordo (border color)

La proprietà **color** definisce esclusivamente:

il colore di primo piano, ovvero quello del testo.

il colore del bordo di un elemento quando non si imposti esplicitamente quest'ultimo con le proprietà **border** o **border-color**.

Per gli altri due casi esistono proprietà ad hoc che esamineremo nelle prossime lezioni. Una buona norma dei CSS vuole comunque che per un elemento di cui si sia definito il colore di primo piano si definisca anche e sempre un colore di sfondo.

Sintassi

La proprietà color si applica a tutti gli elementi ed è ereditata. Significa

che se si imposta il colore per un elemento esso sarà ereditato da tutti gli elementi discendenti per cui non si definisca esplicitamente un altro colore. La sintassi è semplice:

```
selettore { color: <valore> }
```

Valori

I valori possibili sono:

qualunque valore di un colore definito con i metodi visti sopra

la parola chiave **inherit**. Con essa si dice esplicitamente al browser di ereditare le impostazioni definite per l'elemento padre. E' stata introdotta con CSS2 e da qui in avanti sarà citata nella guida senza ulteriori spiegazioni.

Esempi

```
p {color: black; background-color: white; }
```

Gestione dello sfondo

Esamineremo in questa lezione le proprietà relative alla gestione dello sfondo. Sin dalle origini di HTML è stato possibile definire un colore o un'immagine di sfondo per le nostre pagine web. La scelta si restringe comunque a due elementi: il corpo principale della pagina (**<BODY>**) o le tabelle. Un'altra limitazione riguarda il comportamento delle immagini di sfondo: esse vengono infatti ripetute in senso orizzontale e verticale fino a riempire l'intera area della finestra, del frame o della tabella. Proprio questo comportamento ha fatto sempre propendere per la scelta di piccole textures in grado di dare la sensazione visiva dell'uniformità. Grazie ai CSS queste limitazioni vengono spazzate via e le possibilità creative, compatibilità permettendo, sono davvero infinite.

Ecco la lista delle proprietà, applicabili, ed è questa la prima grande innovazione dei CSS, a **tutti gli elementi**:

background-color
background-image

background-repeat
background-attachment
background-position

Ciascuna di essa definisce un solo, particolare aspetto relativo allo sfondo di un elemento. La proprietà **background**, invece, è una proprietà a sintassi abbreviata con cui possiamo definire sinteticamente e con una sola dichiarazione tutti i valori per lo sfondo. La analizzeremo alla fine. Prima è necessario chiarire significato e sintassi delle proprietà singole.

background-color

Definisce il colore di sfondo di un elemento. Questa proprietà non è ereditata.

Sintassi

```
selettore {background-color: <valore>;}
```

Valori

un qualunque colore

la parola chiave **transparent**

Usando **transparent** come valore un elemento avrà come colore quello dell'elemento padre.

Esempi

```
body { background-color: white; }  
p { background-color: #FFFFFF; }  
.class1 { background-color: rgb(0, 0, 0); }
```

background-image

Definisce l'URL di un'immagine da usare come sfondo di un elemento. Questa proprietà non è ereditata.

Sintassi

```
selettore { background-image: url(<valore>); }
```

Valori

un **URL** assoluto o relativo che punti ad un'immagine
la parola chiave **none**. Valore di default.

Usare **none** equivale a non impostare la proprietà: nessuna immagine verrà applicata come sfondo.

Esempi

```
body {background-image: url(sfondo.gif); }  
div {background-image:  
    url(http://www.server.it/images/sfondo.gif); }
```

Usando questa proprietà da sola si ottiene lo stesso risultato che in HTML si aveva con l'attributo **background**.

Piccolo consiglio. Una buona norma e il buon senso vogliono che quando si definisce un'immagine come sfondo si usi sempre anche un colore di sfondo e che questo colore consenta una lettura agevole del testo. Se le immagini sono disabilite, ad esempio, il browser mostrerebbe il suo colore di sfondo predefinito: se questo è bianco e il nostro testo pure sarebbe evidentemente un disastro. Attenzione, dunque!

background-repeat

Con questa proprietà iniziano le novità. Essa consente di definire la direzione in cui l'immagine di sfondo viene ripetuta. Proprietà non ereditata.

Sintassi

```
selettore {background-repeat: <valore>;}
```

Valori

repeat. L'immagine viene ripetuta in orizzontale e verticale. E' il comportamento standard.

repeat-x. L'immagine viene ripetuta solo in orizzontale.

repeat-y. L'immagine viene ripetuta solo in verticale.

no-repeat. L'immagine non viene ripetuta.

Esempi

```
body { background-image: url(sfondo.gif);
        background-repeat: repeat; }
div { background-image: url(sfondo.gif);
      background-repeat: repeat-x; }
```

background-attachment

Con questa proprietà si imposta il comportamento dell'immagine di sfondo rispetto all'elemento cui è applicata e all'intera finestra del browser. Si decide, in pratica, se essa deve scorrere insieme al contenuto o se deve invece rimanere fissa. Proprietà non ereditata.

Sintassi

```
selettore {background-attachment: <valore>;}
```

Valori

scroll. L'immagine scorre con il resto del documento quando si fa lo scrolling della pagina.

fixed. L'immagine rimane fissa mentre il documento scorre.

Questa proprietà consente risultati estetici di grande impatto ed è consigliabile, per ottenerne il meglio, usarla sia in combinazione con le altre proprietà sia con immagini grandi.

Esempi

```
body { background-image: url(back_400.gif);
        background-repeat: repeat-x;
        background-attachment: fixed; }
```

background-position

Proprietà un po' complessa. Definisce il punto in cui verrà piazzata un'immagine di sfondo non ripetuta o da dove inizierà la ripetizione di una ripetuta. Si applica solo agli elementi blocco o rimpiazzati. Attenzione alla compatibilità e alla resa, non omogenea, tra i vari browser. Proprietà non ereditata.

Sintassi

```
selettore {background-position: <valore> o <valori>;}
```

Valori

I valori possibili sono diversi. Tutti però definiscono le coordinate di un punto sull'asse verticale e su quello orizzontale. Ciò può avvenire nei seguenti modi:

con valori in **percentuale**

con valori espressi con **unità di misura**

con le parole chiave **top, left, bottom, right**

Esempio:

```
body { background-image: url(back_400.gif);
        background-repeat: no-repeat;
        background-position: 50px 50px; }
```

Significa che l'immagine apparirà a 50px dal bordo superiore e a 50px da quello sinistro della finestra. Potevo usare invece dei pixel altre unità di misura o percentuali. Come al solito, la scelta delle percentuali mi assicura una maggiore affidabilità a risoluzioni diverse. Allo stesso modo potevo utilizzare le parole chiave. Se, ad esempio, uso:

```
body { background-image: url(back_400.gif);
        background-repeat: no-repeat;
        background-position: center center; }
```

l'immagine di sfondo apparirà centrata in entrambe le direzioni.

Logica vuole, trattandosi di definire le coordinate che si impostino due valori. Definendone uno solo esso sarà usato sia per l'asse orizzontale che per quello verticale.

background

E veniamo alla proprietà **background**. Con essa, ricordiamolo, possiamo definire in un colpo solo tutti gli aspetti dello sfondo. Per essere valida, la dichiarazione non deve contenere necessariamente riferimenti a tutte le proprietà viste finora, ma **deve** contenere almeno la definizione del colore di sfondo.

Sintassi

```
selettore {background: background-color background-image
background-repeat background-attachment background-position;}
```

I valori non vanno separati da virgole. L'ordine non è importante, ma quello dato è il più logico e andrebbe rispettato: si va in ordine di

importanza.

Esempi

```
body { background: white url(sfondo.gif) repeat-x fixed; }
```

Gestione del testo: proprietà di base

Se c'è un aspetto essenziale dei CSS questo è il nuovo approccio che hanno introdotto per la gestione del testo. Prima c'era il tag ****: pagine pesanti e difficili da gestire. Oggi qualcosa che può trasformare una pagina web in un perfetto esempio di stile tipografico e che finalmente consente, almeno in parte, la potenza e la flessibilità di un word-processor.

La proprietà che definiscono il modo in cui il testo appare sullo schermo sono tante e abbiamo deciso di suddividere l'argomento in due lezioni. Iniziamo quindi dalle proprietà di base. Sono quelle che definiscono i seguenti aspetti:

il font da usare

la sua dimensione

la sua consistenza

l'interlinea tra i paragrafi

l'allineamento del testo

la sua decorazione (sottolineature e simili)

font-family

La proprietà font-family serve a impostare il tipo di carattere di una qualunque porzione di testo. Si applica a tutti gli elementi ed è ereditata.

Gli uomini del W3C hanno creato un meccanismo che consente all'autore di impostare nei CSS non un font singolo e unico, ma un elenco di caratteri alternativi. Il meccanismo funziona così:

```
p {font-family: arial, Verdana, sans-serif;}
```

Quando la pagina verrà caricata, il browser tenterà di usare il primo font della lista. Se questo non è disponibile userà il secondo. In

manca anche di questo verrà utilizzato il font principale della famiglia sans-serif presente sul sistema. La spiegazione di tutto ciò è semplice: ovviare al problema dei diversi font presenti sulle piattaforme software.

Dunque: quando si imposta la proprietà font-family si possono usare tutti i font che si vuole, ma non dimenticate mai di inserire alla fine l'indicazione di una famiglia generica. Esse sono cinque (tra parentesi riportiamo i caratteri predefiniti su ciascuna sui sistemi Windows):

serif (Times New Roman)

sans-serif (arial)

cursive (Comic Sans)

fantasy (Allegro BT)

monospace (Courier)

I nomi dei font della lista vanno separati dalla virgola. I caratteri con nomi composti da più parole vanno inseriti tra virgolette. Se usate famiglie strane e poco comuni come fantasy o cursive usate più di una famiglia generica. Questa andrebbe sempre messa alla fine, altrimenti risulta praticamente inutile la definizione di font specifici.

Sintassi

```
selettore {font-family: <font 1>, <font2>, . . . ,<famiglia generica>;}
```

Esempi

```
div {font-family: Georgia, "Times New Roman", serif;}
```

font-size

Insieme a font-family è la proprietà considerata essenziale nella definizione dell'aspetto del testo, di cui definisce le dimensioni. Applicabile a tutti gli elementi ed ereditata.

Sintassi

```
selettore { font-size: <valore>; }
```

Valori

I valori delle dimensioni del testo possono essere espressi in vari modi.

Per un approfondimento sull'uso dei diversi metodi e sulle implicazioni in termini di accessibilità e compatibilità cross-browser si consiglia la lettura dell'articolo "Stabilire le dimensioni dei font con i CSS: vantaggi, problemi e soluzioni".

I diversi sistemi si possono distinguere a seconda che definiscano le dimensioni in senso assoluto o relativo. Dimensione assoluta significa che essa non dipende da nessun altro elemento ed è quella definita dall'unità di misura usata. Dimensione relativa significa che essa viene calcolata in base alla dimensione del testo dell'elemento parente.

Sono valori assoluti:

le sette parole chiave **xx-small**, **x-small**, **small**, **medium**, **large**, **x-large**, **xx-large**

quelli espressi con le seguenti **unità di misura**: pixel (**px**), centimetri (**cm**), millimetri (**mm**), punti (**pt**), picas (**pc**), pollici (**in**), x-height(**ex**). Di tutte queste unità le uniche proponibili per il testo sono punti e pixel. Si consiglia di usare la prima solo per CSS destinati alla stampa.

Sono valori relativi:

le parole chiave **smaller** e **larger**

quelli espressi in **em** (em-height)

quelli espressi in **percentuale**

Esempi

```
p {font-size: 12px;}
div {font-size: 50%;}
#div1 {font-size: large;}
h2 {font-size: 1.2em;}
```

font-weight

Serve a definire la consistenza o "peso" visivo del testo. Si applica a tutti gli elementi ed è ereditata.

Sintassi

```
selettore {font-weight: <valore>;}
```

Valori

Il "peso" visivo di un carattere può essere espresso con una scala numerica o con parole chiave:

valori numerici 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 ordinati in senso crescente (dal leggero al pesante)

normal. Valore di default. E' l'aspetto normale del font ed equivale al valore 400

bold. Il carattere acquista l'aspetto che definiamo in genere grassetto. Equivale a 700

bolder. Misura relativa. Serve a specificare che una determinata porzione di testo dovrà apparire più pesante rispetto al testo dell'elemento parente

lighter. Misura relativa. Il testo sarà più leggero di quello dell'elemento parente

Esempi

```
p {font-weight: 900;}
div {font-weight: bold;}
```

font-style

Imposta le caratteristiche del testo in base ad uno di questi tre valori:

normal: il testo mantiene il suo aspetto normale

italic: formatta il testo in corsivo

oblique: praticamente simile a italic

La proprietà si applica a tutti gli elementi ed è ereditata.

Sintassi

```
selettore {font-style: <valore>;}
```

Esempi

```
p {font-style: italic;}
```

Line-height

Grazie a `line-height` è possibile finalmente usare nelle nostre pagine un sistema di interlinea degno di questo nome. La proprietà, in realtà, serve a definire l'altezza di una riga di testo all'interno di un elemento blocco. Ma l'effetto ottenuto è appunto quello tanto agognato da tutti i web designer: un modo per impostare uno spazio tra le righe. La proprietà si applica a tutti gli elementi ed è ereditata.

Sintassi

```
selettore {line-height: <valore>;}
```

Valori

normal. Il browser separerà le righe con uno spazio ritenuto "ragionevole". Dovrebbe corrispondere a un valore numerico compreso tra 1 e 1.2

valore numerico. Usando valori numerici tipo 1.2, 1.3, 1.5 si ottiene questo risultato: l'altezza della riga sarà uguale alla dimensione del font moltiplicata per questo valore.

un valore numerico con unità di misura. L'altezza della riga sarà uguale alla dimensione specificata.

percentuale. L'altezza della riga viene calcolata come una percentuale della dimensione del font.

Il consiglio è di non usare mai la percentuale, di valutare attentamente l'utilizzo di unità esplicite e di affidarsi senza problemi al valore numerico.

Esempi

```
p {line-height: 1.5;}  
body {line-height: 15px;}
```

font

La proprietà **font** può essere paragonata a **background**. Si tratta di una proprietà a sintassi abbreviata che serve a impostare con una sola dichiarazione tutte le principali caratteristiche del testo. Le proprietà definibili in forma abbreviata con **font** sono:

font-family
font-size

line-height
font-weight
font-style
font-variant

Sintassi

```
selettore {font: <valori>;}
```

Alcune indicazioni sull'uso. I valori relativi alle singole proprietà non vanno separati da virgole. Virgola che deve invece separare i valori definiti per la `font-family`. Anche in questo caso i nomi dei font costituiti da più parole vanno racchiusi tra virgolette. Per quanto riguarda l'ordine, la dichiarazione dovrebbe sempre finire con la coppia `font-size > font-family`.

Il valore della proprietà `line-height` si imposta invece così, facendo seguire il suo valore a quello di `font-size` e separando i due con una slash:

```
font-size/line-height
```

Esempi

```
p {font: bold 12px/1.5 Georgia, "Times New Roman", serif;}
```

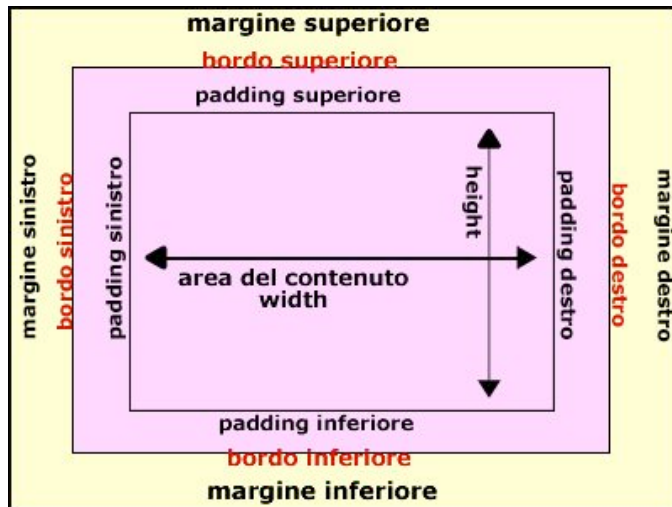
Nell'ordine abbiamo definito: `font-weight`, `dimensione/interlinea`, `font-family`.

Il box model

Se volete usare i CSS per scopi che vadano oltre la semplice gestione di sfondo e testo dovete avere ben chiaro il meccanismo che governa la presentazione dei vari elementi di una pagina. Torniamo per un attimo alla prima lezione. Abbiamo lì mostrato che una pagina (X)HTML non è altro che un insieme di box rettangolari, che si tratti di elementi **blocco** o di elementi **inline** non fa differenza. Tutto l'insieme di regole che gestisce l'aspetto visuale degli **elementi blocco** viene in genere riferito al cosiddetto **box model**.

Ogni box comprende un certo numero di componenti di base, ciascuno modificabile con proprietà dei CSS. La figura qui sotto mostra

visivamente tali componenti:



Partendo dall'interno abbiamo:

l'area del contenuto. E' la zona in cui trova spazio il contenuto vero e proprio, testo, immagini, animazioni Flash. Le dimensioni orizzontali dell'area possono essere modificate con la proprietà **width**. Quelle verticali con **height**.

il padding. E' uno spazio vuoto che può essere creato tra l'area del contenuto e il bordo dell'elemento. Come si vede dalla figura, se si imposta un **colore di sfondo** per un elemento questo si estende dall'area del contenuto alla zona di padding.

il bordo. E' una linea di dimensione, stile e colore variabile che circonda la zona del padding e l'area del contenuto.

il margine. E' uno spazio di dimensioni variabili che separa un dato elemento da quelli adiacenti.

Attenzione. Queste cose non sono state introdotte con i CSS, ma fanno parte del normale meccanismo di rendering di un documento. Quando realizziamo una pagina (X)HTML senza fogli di stile è il browser ad applicare per alcune di queste proprietà le sue impostazioni predefinite. Per esempio, introdurrà un certo margine tra un titolo e un

paragrafo o tra due paragrafi. La novità è che con i CSS possiamo controllare con precisione al pixel tutti questi aspetti.

Il **box model** è governato da una serie di regole di base concernenti la definizione di un box e il suo rapporto con gli altri elementi.

1. Larghezza del box

Bisogna distinguere tra la larghezza dell'area del contenuto e la larghezza effettiva di un box. La prima è data dal valore della proprietà **width**. La seconda è data da questa somma:

$\text{margine sinistro} + \text{bordo sinistro} + \text{padding sinistro} + \text{area del contenuto} + \text{padding destro} + \text{bordo destro} + \text{margine destro}$

Come si vede infatti nella figura margini, padding e bordi devono considerarsi a tutti gli effetti parte dell'area complessiva dell'elemento.

Codice:

```
div#1 {
background: silver;
margin: 40px;
border: 10px solid #ccc033;
padding: 10px;
width: 200px;
height: 100px;
}
```

Da qui dovrebbe emergere bene il concetto fondamentale riguardante la larghezza. La larghezza complessiva dell'elemento, ovvero l'area che occupa sullo schermo, è di 320px. Si calcola così:

$\text{margine sin.} + \text{padding sin.} + \text{bordo sin.} + \text{width} + \text{padding destro} + \text{bordo destro} + \text{margine destro}$

Dunque: $40 + 10 + 10 + 200 + 10 + 10 + 40 = 320$

Lo spazio occupato dal div va considerato, in questo caso, dal bordo della finestra fino all'estremo del margine destro.

320 pixel

Altra cosa è la proprietà **width**. Essa è di 200px. E come si vede va intesa come l'area dove trova spazio il testo dell'elemento.

2. Larghezza ed elemento contenitore

Se non si imposta alcun valore per la proprietà **width** o se il valore usato è **auto** la larghezza di un box è uguale a quella dell'area del contenuto dell'elemento contenitore. Quest'ultimo è l'elemento che racchiude il box

3. Uso del valore auto

Solo per tre proprietà è possibile impostare il valore **auto**: margini, altezza e larghezza (**width**). L'effetto è quello di lasciar calcolare al browser l'ammontare di ciascuna di esse in base alla risoluzione dello schermo e alle dimensioni della finestra.

Solo i margini possono avere valori negativi. Ciò non è consentito per

padding, bordi, altezza e larghezza.

4. Margini verticali e orizzontali tra gli elementi

Per due box adiacenti in senso verticale che abbiano impostato un margine inferiore e uno superiore la distanza non sarà data dalla somma delle due distanze. A prevalere sarà invece la distanza maggiore tra le due. E' il meccanismo del cosiddetto **margin collapsing**. Tale meccanismo non si applica ai box adiacenti in senso orizzontale.

Gestione delle dimensioni: altezza

Con le prossime cinque lezioni vedremo come sia possibile definire e controllare le diverse proprietà del [box-model](#). In questa vedremo in che modo nei CSS si impostino le dimensioni verticali di un elemento. Un primo concetto fondamentale: **in genere l'altezza di un elemento è determinata dal suo contenuto**. Più testo inserisco in box, in un paragrafo o in una cella di tabella più essi saranno estesi in senso verticale. Semplice. Le proprietà che analizzeremo servono in pratica a fornire un meccanismo in grado di controllare o superare in qualche modo questo comportamento standard.

height

Questa proprietà definisce la distanza tra il bordo superiore e quello inferiore di un elemento. Non è ereditata e si applica a tutti gli elementi tranne:

colonne di tabelle

elementi inline non rimpiazzati

Sintassi

```
selettore {height: <valore>;}
```

Valori

un valore numerico con unità di misura.

un valore in percentuale. Il valore in percentuale si riferisce sempre all'altezza del blocco contenitore, purché esso abbia un'altezza esplicitamente dichiarata. Diversamente, la percentuale viene interpretata come **auto**.

auto. L'altezza sarà quella determinata dal contenuto.

L'uso di **height** va sempre valutato con attenzione e non pensando di farne una scorciatoia per avere layout precisi al pixel. Il caso più importante da valutare è quando il contenuto dovesse superare i limiti imposti tramite la proprietà. Il comportamento dei vari browser è al riguardo notevolmente diverso. Ricordatelo: molte volte si può ottenere lo stesso risultato visivo desiderato usando proprietà come padding e margin. E considerate sempre che l'altezza è indirettamente influenzata anche dalla larghezza di un elemento.

Esempi

```
div {height: 250px;}  
p.blocco {height: 50%;}
```

min-height

Imposta un'altezza minima per un elemento. Valgono per questa proprietà le stesse osservazioni fatte per **height** relativamente al contenuto. Non è ereditata e non è supportata da Explorer Win.

Sintassi

```
selettore {min-height: <valore>;}
```

Valori

un valore numerico con unità di misura.

un valore in percentuale

Esempi

```
div {min-height: 200px;}
```

max-height

La proprietà **max-height** serve a impostare l'altezza massima di un elemento. Anche per essa valgono le osservazioni già fatte per il contenuto eccedente. Non è ereditata.

Sintassi

```
selettore {max-height: <valore>;}
```

Valori

none. Valore iniziale. L'altezza dell'elemento non è limitata.

un valore numerico con unità di misura.

un valore in percentuale

Esempi

```
div {max-height: 400px;}
```

Gestione delle dimensioni: larghezza

La definizione della larghezza e più in generale la formattazione orizzontale degli elementi sono più problematiche rispetto all'altezza e alla formattazione verticale. A questo punto è utile richiamare alla mente i concetti visti per il box model e ribadire a cosa si riferiscono le proprietà che stiamo per esaminare.

width

Le dimensioni orizzontali di un elemento sono date dalla combinazione di varie proprietà. Un errore macroscopico è ritenere che esse siano definite semplicemente dalla proprietà **width**. In pratica, dovete sempre distinguere tra la larghezza effettiva di un elemento (i pixel di spazio che occupa sulla pagina) e la larghezza dell'area del contenuto. La prima è data dalla somma di questi valori:

```
margine sinistro + bordo sinistro + padding sinistro + area del  
contenuto + padding destro + bordo destro + margine destro
```

La seconda è impostata tramite la proprietà **width** (si vedano [gli esempi](#) già visti per il box model).

E' possibile ovviamente che i due valori coincidano. Accade quando di un particolare elemento non si impostino margini, padding e bordi; o

quando il valore di tali proprietà sia uguale a 0.

E qui introduciamo una questione di rilevanza assoluta per il semplice fatto che coinvolge il browser attualmente più usato. Explorer per Windows fino alla versione 5.5 interpreta male il concetto di **width**. Nel senso che con questa proprietà intende la larghezza globale dell'elemento, compresi margini, padding e bordi. Proprio l'errore che prima abbiamo definito macroscopico! Fatta questa fondamentale premessa, possiamo analizzare nei dettagli la proprietà.

Sintassi

```
selettore {width: <valore>;}
```

Valori

auto. Valore di default. Se non si impostano margini, bordi e padding la larghezza dell'elemento sarà uguale all'area del contenuto dell'elemento contenitore.

un valore numerico con unità di misura.

un valore in percentuale. La larghezza sarà calcolata rispetto a quella dell'elemento contenitore.

La proprietà width non è ereditata.

Esempi

```
p {width: 90px;}  
div.box {width: 50%;}
```

min-width

Imposta la larghezza minima di un elemento. Si applica a tutti gli elementi tranne che a quelli in linea non rimpiazzati e agli elementi di tabelle. Proprietà non supportata da Internet Explorer e non ereditata.

Sintassi

```
selettore { min-width: <valore>; }
```

Valori

un valore numerico con unità di misura.

un valore in percentuale. La larghezza sarà come minimo quella espressa dalla percentuale riferita alla larghezza dell'elemento

contenitore.

Esempi

```
div { min-width: 400px; }
```

max-width

Imposta la larghezza massima di un elemento. Non è ereditata.

Sintassi

```
selettore { max-width: <valore>; }
```

Valori

none. Valore di default. Non c'è un limite per larghezza dell'elemento.

un valore numerico con unità di misura.

un valore in percentuale

I Margini

Una delle maggiori limitazioni di HTML è la mancanza di un meccanismo interno per spaziare gli elementi di un documento. Per ottenere questo risultato si è ricorso negli anni ad una serie di trucchetti, magari efficaci, ma fuori dalla logica originaria del linguaggio. L'unica eccezione consisteva nella possibilità di definire una distanza tra le celle di una tabella con l'attributo **cellspacing**. I CSS risolvono il problema con l'uso di una serie di proprietà in grado di definire con precisione la distanza tra gli elementi. Possiamo infatti definire il margine come la distanza tra il bordo di un elemento e gli elementi adiacenti. Per ulteriori chiarimenti e per la relazione tra i margini e gli altri elementi del box-model si veda la lezione "Il box model".

Sono cinque le proprietà con cui è possibile definire un margine. Quattro di esse sono singole e impostano la distanza per ciascun lato del box. L'ultima, **margin**, è una proprietà a sintassi abbreviata utile per definire con una sola dichiarazione tutte le impostazioni per i quattro lati.

margin-bottom

Definisce la distanza tra il lato inferiore di un elemento e gli elementi adiacenti. Si applica a tutti gli elementi e non è ereditata.

Valori

un valore numerico con unità di misura. Il valore è espresso in termini assoluti.

un valore in percentuale. Il valore è calcolato come percentuale rispetto alla larghezza (width) del blocco contenitore.

auto. E' quasi sempre corrispondente a 0.

Ricordiamo che se per un elemento si imposta un margine inferiore ed esso si trova sopra ad un altro elemento che abbia impostato un margine superiore, la distanza tra i due sarà quella maggiore e non data dalla somma delle due proprietà (meccanismo del margin collapsing).

Esempi

```
p {margin-bottom: 10px;} div.box {margin-bottom: 20%;}
```

margin-left

Definisce la distanza tra il lato sinistro di un elemento e gli elementi adiacenti. Si applica a tutti gli elementi e non è ereditata.

Valori

un valore numerico con unità di misura.

un valore in percentuale.

auto.

Esempi

```
h1 {margin-left: 10%;}  
img {margin-left: 20px;}
```

margin-top

Definisce la distanza tra il lato superiore di un elemento e gli elementi adiacenti. Si applica a tutti gli elementi e non è ereditata.

Valori

un valore numerico con unità di misura.

un valore in percentuale.

auto.

Esempi

```
p.box {margin-top: 10%;}  
img {margin-top: 20px;}
```

margin-right

Definisce la distanza tra il lato destro di un elemento e gli elementi adiacenti. Si applica a tutti gli elementi e non è ereditata.

Valori

un valore numerico con unità di misura.

un valore in percentuale.

auto.

Esempi

```
h1 {margin-right: 10%;}  
img {margin-right: 20px;}
```

margin

E' una proprietà a sintassi abbreviata. Con essa è possibile specificare i valori per tutti e quattro i lati di un elemento. Si applica a tutti gli elementi e non è ereditata.

Valori

un valore numerico con unità di misura.

un valore in percentuale.

auto. Per la proprietà margin il valore auto significa che la distanza sarà automaticamente calcolata rispetto alla larghezza dell'elemento contenitore.

Per usare al meglio questa proprietà è fondamentale conoscere le

regole che ne gestiscono il funzionamento.

In prima istanza è ovvio usare per essa quattro valori, uno per ciascun lato:

```
div {margin: 10px 15px 10px 20px;}
```

L'ordine di lettura va inteso in senso orario. Per cui: il primo valore si riferisce al lato superiore, il secondo a quello destro, il terzo al lato inferiore, il quarto a quello sinistro. In pratica usare la sintassi vista nell'esempio equivale a scrivere:

```
div { margin-top: 10px; margin-right: 15px; margin-bottom:  
10px; margin-left: 20px; }
```

Nella definizione dei valori, inoltre, è possibile mischiare percentuali con valori assoluti in unità di misura.

Un ulteriore abbreviazione della sintassi si può ottenere usando tre, due o un solo valore. Queste le regole:

- se si usano tre valori, il primo si riferisce al margine superiore, il secondo a quelli sinistro e destro, il terzo a quello inferiore;
- se si usano due valori, il primo si riferisce ai lati superiore e inferiore, il secondo al sinistro e al destro;
- se si usa un solo valore, un uguale distanza sarà ai quattro lati.

Esempi

```
p {margin: 20px 10px;}  
div {margin: 20px;}  
h1 {margin: 10px auto;}
```

Il Padding

Un altro modo per creare spazio intorno ad un elemento è quello di usare il padding. Come per i margini, HTML incorpora un meccanismo simile solo per le celle di tabella, tramite il cosiddetto **cellpadding**. Se avete presente il funzionamento di questo attributo, vi risulterà immediatamente chiara la differenza tra margini e padding. Quando si usa il padding, lo spazio di distanza viene inserito al di qua dei bordi dell'elemento e non all'esterno. La cosa risulta evidente se usate un

colore di sfondo per l'elemento diverso da quello della pagina. Nel caso del padding, lo spazio inserito avrà proprio il colore di sfondo dell'elemento, a differenza dei margini. Pertanto, valutate in base alle vostre esigenze se usare la prima o la seconda via.

Un'analogia rispetto ai margini sta nella sintassi. Anche qui quattro proprietà singole per i lati e una a sintassi abbreviata (**padding**).

padding-bottom

Imposta l'ampiezza del padding sul lato inferiore di un elemento. Si applica a tutti gli elementi e non è ereditata.

Valori

un valore numerico con unità di misura. Il valore è espresso in termini assoluti.

un valore in percentuale. Il valore è calcolato come percentuale rispetto alla larghezza (width) del blocco contenitore.

Esempi

```
p.box {padding-bottom: 10px;}
```

padding-left

Imposta l'ampiezza del padding sul lato sinistro di un elemento. Si applica a tutti gli elementi e non è ereditata.

Valori

un valore numerico con unità di misura.

un valore in percentuale.

Esempi

```
p {padding-left: 10%;}
```

padding-top

Imposta l'ampiezza del padding sul lato superiore di un elemento. Si applica a tutti gli elementi e non è ereditata.

Valori

un valore numerico con unità di misura.

un valore in percentuale.

Esempi

```
h1 {padding-top: 10px;}
```

padding-right

Imposta l'ampiezza del padding sul lato destro di un elemento. Si applica a tutti gli elementi e non è ereditata.

Valori

un valore numerico con unità di misura.

un valore in percentuale.

Esempi

```
p.box {padding-right: 10px;}
```

padding

Proprietà a sintassi abbreviata. Serve a impostare i valori del padding per tutti e quattro i lati di un elemento. Valgono per essa tutte le osservazioni e le regole sintattiche viste per **margin**.

Valori

un elenco di valori numerici con unità di misura.

un elenco di valori in percentuale.

Esempi

```
p {padding: 10px 20px;} div {padding: 10%;}
```

I Bordi

Dal punto di vista del linguaggio la definizione dei bordi può risultare un pò intricata per il numero di proprietà coinvolte, che se consentono all'autore la massima flessibilità, rendono complicata la gestione del codice.

In linea di massima possiamo suddividere le proprietà in due categorie: singole e a sintassi abbreviata. Le prime definiscono singoli aspetti di ciascuno dei quattro bordi. Le seconde consentono di riunire in una sola regola le diverse impostazioni.

Sono proprietà singole:

```
border-top-color, border-top-style, border-top-width, border-bottom-color, border-bottom-style, border-bottom-width, border-right-color, border-right-style, border-right-width, border-left-color, border-left-style, border-left-width
```

Sono proprietà a sintassi abbreviata:

```
border, border-bottom, border-top, border-right, border-left, border-color, border-style, border-width
```

Definire lo stile di un singolo bordo

Sintassi (con proprietà singole)

```
selettore { border-<lato>-color: <valore>; border-<lato>-style: <valore>; border-<lato>-width: <valore>; }
```

Sintassi (abbreviata)

```
selettore { border-<lato>: <valore width> <valore style> <valore color>; }
```

In entrambi gli esempi di sintassi sostituite a **<lato>** uno degli indicatori dei quattro lati: **top**, **right**, **bottom** o **left**.

Valori

Come si vede dall'elenco delle proprietà di ciascun lato si possono definire per il bordo tre aspetti:

il colore (color)

lo stile (style)

lo spessore (width)

I valori possibili per il colore sono:

un qualsiasi colore la parola chiave inherit

Il colore può essere espresso in uno qualunque dei modi visti nella lezione " Il box model". Se non si imposta un valore specifico, il colore sarà quello di primo piano impostato con la proprietà **color**.

Lo stile di un bordo può invece essere espresso con una delle seguenti parole chiave

none. L'elemento non presenta alcun bordo e lo spessore equivale a

hidden. Equivalente a none

dotted

dashed

solid

double.

groove

ridge

inset

outset

Infine abbiamo lo spessore. Esso può essere modificato secondo i seguenti valori:

un valore numerico con unità di misura

thin. Bordo sottile.

medium. Bordo di medio spessore.

thick. Bordo di largo spessore.

Nel caso delle parole chiave la dimensione esatta non è specificata dal linguaggio.

Esempi

```
Div { border-left-color: black;
      border-left-style: solid;
```

```
border-left-width: 1px; }
```

Molto più comodo scrivere così:

```
div {border-left: 1px solid black;}
```

Definire lo stile dei quattro bordi

Sintassi

```
selettore { border-width: <valori>;  
            border-style: <valori>;  
            border-color: <valori>; }
```

Usando questa sintassi e queste proprietà si imposta lo stile per tutti e quattro i bordi del box. I valori possibili sono quelli visti prima a proposito del colore, dello stile e dello spessore.

Per ciascuna di queste proprietà è possibile definire da uno a quattro valori, uno per lato. Se ne usiamo quattro l'ordine di lettura è questo: top, right, bottom, left. Se invece ne impostiamo uno, due o tre valgono le stesse regole viste per i margini nella lezione "Gestione delle dimensioni: larghezza"

Esempi

```
div { border-width: 1px 2px 1px 2px;  
      border-style: solid;  
      border-color: black red black red; }
```

Usare la proprietà border

L'ultima proprietà a sintassi abbreviata è **border**. Con essa possiamo definire con una sola regola le impostazioni per i quattro bordi. Il suo uso è però limitato a un solo caso, peraltro molto comune: che i quattro bordi abbiano tutti lo stesso colore, lo stesso stile e lo stesso spessore.

Sintassi

```
selettore {border: <valore spessore> <valore stile> <valore
```

```
colore>;}
```

Esempi

```
div {border: 2px solid black;}
```

Le Liste

Grazie ai CSS possiamo definire in vari modi l'aspetto delle **liste** (X)HTML. In realtà tutte le proprietà che andremo ad esaminare si riferiscono non ai canonici tag usati per inserire una lista ordinata (****) o non ordinata (****), ma ai singoli elementi che le compongono, ovvero l'elemento ****. In effetti, è solo questo che trova una rappresentazione effettiva sulla pagina, mentre **OL** e **UL** sono semplici dichiarazioni del tipo di lista usato.

Le proprietà dedicate alla formattazione delle liste sono quattro. tre definiscono singoli aspetti, l'ultima, **list-style**, è una proprietà a sintassi abbreviata.

list-style-image

Definisce l'URL di un'immagine da usare come marcatore di un list-item. Proprietà ereditata. Si applica agli elementi **LI** e a quelli per i quali si imposti la proprietà **display** sul valore **list-item**.

Sintassi

```
<selettore> {list-style-image: url(<url_immagine>);}
```

Nella definizione della sintassi per tutte queste proprietà avete a disposizione diverse strade. La prima e più semplice è quella di definire lo stile a livello degli elementi lista o list-item:

```
ul {list-style-image: url(immagine.gif);}  
ol {list-style-image: url(immagine.gif);}  
li {list-style-image: url(immagine.gif);}
```

Più correttamente e per un fatto di rigore strutturale (lo stile si applica ai list-item), è preferibile, usando **UL** o **OL**, affidarsi ad un selettore del

discendente (**descendant selector**):

```
ul li {list-style-image: url(immagine.gif);}
```

La soluzione è ottimale se prevedete di usare sempre uno stesso stile per tutte le liste. Se invece pensate di usare stili diversi, affidatevi alla potenza delle classi, che applicherete di volta in volta secondo le necessità. La sintassi consigliata è questa:

```
ul.nome_della_classe li {list-style-image: url(immagine.gif);}
```

Valori

un URL assoluto o relativo che punti ad un'immagine.

none. Non viene usata nessuna immagine.

list-style-position

Imposta la posizione del marcatore rispetto al testo del list-item. Proprietà ereditata. Si applica agli elementi **LI** e a quelli per i quali si imposti la proprietà **display** sul valore **list-item**.

Sintassi

```
<selettore> {list-style-position: <valore>;}
```

Valori

outside. Valore di default. E' il comportamento standard: il marcatore è piazzato all'esterno del testo.

inside. Il marcatore diventa parte integrante del testo e ne rappresenta in un certo senso il primo carattere. Se il testo va a capo il marcatore apparirà all'interno del box.

Esempi

```
ul li {list-style-position: inside;}  
li {list-style-position: outside;}
```

list-style-type

Definisce l'aspetto del marcatore. Proprietà ereditata. Si applica agli elementi **LI** e a quelli per i quali si imposti la proprietà **display** sul valore **list-item**.

Sintassi

```
<selettore> {list-style-type: <valore>;}
```

Valori

La scelta dei valori possibili è lunghissima. Definiamo nei dettagli solo i più importanti.

disc: un cerchietto pieno colorato. Il colore può essere modificato per tutti i tipi con la proprietà **color**.

circle: un cerchietto vuoto.

square: un quadratino

decimal: sistema di conteggio decimale 1, 2, 3,

decimal-leading-zero: sistema di conteggio decimale ma con la prima cifra preceduta da 0. 01, 02, 03,

lower-roman: cifre romane in minuscolo. i, ii, iii, iv,

upper-roman: cifre romane in maiuscolo. I, II, III, IV,

lower-alpha: lettere ASCII minuscole. a, b, c,

upper-alpha: lettere ASCII maiuscole. A, B, C,

lower-latin: simile a lower-alpha

upper-latin: simile a upper-alpha

lower-greek: lettere minuscole dell'alfabeto greco antico.

list-style

Proprietà a sintassi abbreviata con cui si definiscono tutti gli aspetti e gli stili di un list-item. Proprietà ereditata.

Sintassi

```
<selettore> {list-style: <valore tipo> <valore posizione>}
```

```
<valore immagine>;}
```

Valori

I valori possibili sono quelli visti in precedenza per le proprietà singole. A rigor di logica solo due delle tre proprietà singole dovrebbero trovare posto in questa dichiarazione abbreviata: per definire l'aspetto del marcatore, infatti, o decido di usare un'immagine o propendo per un marcatore a carattere. Se si inseriscono indicazioni per tipo e immagine, prevale l'immagine e il tipo è ignorato.

Esempi

```
ul {list-style: square inside;}  
ul.lista li {list-style: outside url(imagimgine.gif);}
```

Tre proprietà speciali: display, float clear

Le tre proprietà che esamineremo in questa lezione possono modificare radicalmente la presentazione del documento. Sono, dunque, strumenti potenti ma come quei giocattoli un pò pericolosi vanno usati con molta cautela e sapendo bene cosa si fa, tenendo anche conto di serie implicazioni a livello di rendering tra i diversi browser.

display

Torniamo per un attimo alla prima lezione. Avevamo chiarito in quella sede la fondamentale distinzione tra elementi **blocco**, **inline** e **lista** che è alla base di (X)HTML. Bene, quello che sembrava un assioma inconfutabile, può essere sconvolto con l'uso della proprietà **display**. Essa ha una sola, semplice funzione: definire il trattamento e la presentazione di un elemento. Fin quando non la si usa ognuno segue la sua natura e il suo comportamento standard, ma con **display** possiamo intervenire su di esso e in certi casi ribaltarlo. La proprietà è ereditata.

Sintassi

```
<selettore> {display: <valore>;}
```

Valori

La lista dei possibili valori è lunghissima. Solo alcuni di essi sono però di normale utilizzo e supportati:

inline. Valore di default. L'elemento assume le caratteristiche degli elementi **inline**.

block. L'elemento viene trattato come un elemento **blocco**.

list-item. L'elemento si trasforma in un elemento **lista**.

Esempi

Nel suo primo manuale sui CSS Eric Meyer si chiede: "Ma perchè esiste la proprietà display?". Domanda legittima. A che serve cambiare l'ordine delle cose? Se ho a disposizione le liste, perchè devo impostare un paragrafo o un titolo come un list-item? Vero. Ma altri casi meno fantasiosi sono da prendere in considerazione. Le immagini, per esempio, sono per loro natura elementi inline, si inseriscono nel testo ed è talvolta complicato gestirne il posizionamento. Se volessi mostrarle in una riga tutta per loro mi basterebbe impostare il display su block, così:

```
img {display: block;}
```

La vera utilità della proprietà **display**, emerge comunque in linguaggi non strutturali come XML. Uno dei modi per rendere un file XML in un browser è proprio quello di formattarlo con un CSS. Ma gli elementi XML non ci dicono nulla su presentazione e struttura, a differenza di (X)HTML! Se trovo **H1** so cosa significa e come verrà mostrato. Ma questo documento? Codice:

```
<guide> <guide_linguaggi_web> <guida>HTML</guida> <guida>CSS</guida> </guide_linguaggi_web> <guide_scripting> <guida>JavaScript</guida> <guida>VBScript</guida> </guide_scripting> </guide>
```

Nessuna informazione sulla presentazione. L'unico modo che ho per stabilire come rendere tali elementi è tramite la proprietà **display**. Ma questo è un discorso non proprio attinente a questa guida. Basta l'accenno.

float

Con questa proprietà è possibile rimuovere un elemento dal normale flusso del documento e spostarlo su uno dei lati (destra o sinistra) del suo elemento contenitore. Il contenuto che circonda l'elemento scorrerà intorno ad esso sul lato opposto rispetto a quello indicato

come valore di **float**. La proprietà non è ereditata.

Il float è un altro caso di funzionalità presenti in HTML solo per certi elementi che i CSS hanno esteso a tutti gli altri (abbiamo già incontrato il padding). Non spaventatevi della definizione di prima. Il floating è un'operazione che veniva fatta in HTML sulle immagini. Bastava usare nel tag **IMG** l'attributo **align** e impostare come valore **left, right, middle, etc.**

Sintassi

```
<selettore> {float: <valore>;}
```

Valori

left. L'elemento viene spostato sul lato sinistro del box contenitore, il contenuto scorre a destra.

right. L'elemento viene spostato sul lato destro, il contenuto scorre a sinistra.

none. Valore iniziale e di default in mancanza di una dichiarazione esplicita. L'elemento mantiene la sua posizione normale.

Una nota importantissima. Se usate il float con le immagini non avete problemi perchè esse hanno una dimensione intrinseca che il browser riconosce. Ma se lo applicate ad altri elementi **dovete** esplicitamente impostare una dimensione orizzontale con la proprietà **width**.

Esempi

```
div {width: 200px; float:right;}  
img {float: left;}
```

clear

La proprietà **clear** serve a impedire che al fianco di un elemento compaiano altri elementi con il float. Si applica solo agli elementi **blocco** e non è ereditata.

L'origine di tale proprietà è questa: visto che il float sposta un elemento dal flusso normale del documento, è possibile che esso venga a trovarsi in posizioni non desiderate, magari al fianco di altri elementi che vogliamo invece tenere separati. **clear** risolve questo problema.

Sintassi

```
<selettore> {clear: <valore>;}
```

Valori

none. Gli elementi con float possono stare a destra e sinistra dell'elemento.

left. Si impedisce il posizionamento a sinistra.

right. Si impedisce il posizionamento a destra.

both. Si impedisce il posizionamento su entrambi i lati.

Esempi

Considera il CSS costituito dalla seguente regola:

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



Titolo di esempio

Sotto invece abbiamo applicato al titolo questa classe:

```
h1.clear {  
  clear : left;  
}
```

Come conseguenza impediamo all'immagine di stare al suo fianco sinistro.



Titolo di esempio 2