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**.

A cosa servono

Finalmente, ad esempio, potrete dare al testo delle vostre pagine un aspetto da word-processor: non solo con il **colore** o i **font** che preferite, ma con un sistema di **interlinea** pratico e funzionale, con le **decorazioni** che desiderate, riuscendo a **spaziare** lettere e parole, impostando **stili diversi** per titoli e paragrafi, sfruttando i benefici dell'**indentatura** o della **giustificazione**. Ancora, potrete finalmente distanziare gli elementi della vostra pagina in maniera semplice e intuitiva con un potente meccanismo di gestione dei **margini**. Tonnellate di gif trasparenti usate per spaziare possono essere finalmente gettate nel cestino del vostro computer. E se i margini non bastano sarete in grado ora di aggiungere bellissimi **bordi** non solo alle tabelle, ma a tutti gli elementi di una pagina. Per non parlare degli **sfondi**: li potrete applicare a quello che volete. E dimenticate quelle brutte micro-textures. Finalmente potrete sbattere su una pagina, magari ben piazzato al centro, quel vostro bellissimo disegno di 600x400 pixel senza doverlo necessariamente vedere trasformato in orribili riquadri ripetuti sulla pagina. Ora siete voi a decidere come usare un'immagine di sfondo: la potete ripetere in una sola direzione, in due o per niente! Facile e comodo. La cosa più bella è che la **gestione del sito** non sarà mai più un incubo. Se quel bellissimo disegno di prima vi stufa, non dovrete più andare a modificare una per una 300 pagine! I CSS sono separati dal documento. Aprite un foglio di stile, cambiate l'immagine e il gioco è fatto. Il risultato sono pagine più leggere e facili da modificare. Milioni di byte di banda risparmiati per la gioia degli utenti. Se poi avete a cuore l'accessibilità i CSS sono uno strumento portentoso, anche grazie al fatto di poter essere gestiti con linguaggi di scripting avanzati in grado di modificare con un solo click l'aspetto di una pagina.

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. 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

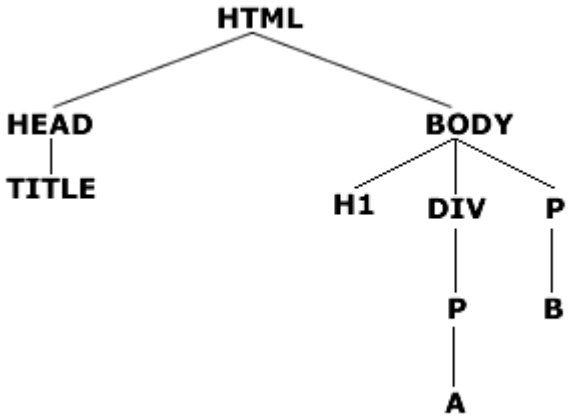
Un altro concetto fondamentale che dovreste assimilare è quello della struttura ad albero di un documento. Il meccanismo fondamentale dei CSS è infatti **l'ereditarietà**.

Esso fa sì che molte proprietà impostate per un elemento siano autamaticamente 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: **descandant**). 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 spazziamo 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.

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: #FFFFCC;
}
</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.

Consigli

A questo punto è giusto chiedersi: quando usare l'una o l'altra soluzione? Il punto di partenza nella risposta deve essere questo: i risultati nella formattazione del documento non cambiano. La giusta soluzione sarà quindi quella richiesta dalla nostra applicazione. Il consiglio sentito è semplice: **pianificate**, pensate in anticipo a quella che dovrà essere la struttura delle pagine del sito. Mettetevi davanti a un monitor o su un pezzo di carta e ragionate: qui metterò un tag <H1>, qui userò una tabella a due colonne, questo box deve avere lo sfondo rosso, etc.

A questo punto potrete costruire per prima cosa un foglio di stile generico ed esterno, da applicare a tutte le pagine del sito. Esso conterrà le regole per formattare gli elementi o le sezioni presenti in tutte queste pagine. Passate poi ad analizzare sezioni ed elementi presenti solo in certe pagine o che vogliate modificare solo in determinati casi. Supponete, ad esempio, di voler cambiare in rosso il colore di un titolo iniziale solo in una pagina delle 150 del vostro sito. Che fare? Semplice: usare uno stile incorporato solo in quella pagina:

```
<style type="text/css">
h1 {color: red; }
</style>
```

Per la legge che regola il meccanismo del cascading (vedi lezione 13 di prossima pubblicazione) questo stile prevarrà su quello del CSS esterno.

Se le pagine invece di una fossero 20 il discorso diventerebbe un pò complicato. Bisognerebbe armarsi di pazienza e modificarle una per volta. Anche qui però la soluzione è dietro l'angolo. Basta fare un nuovo CSS esterno e collegarlo al documento con @import insieme al foglio generico:

```
<link rel="stylesheet" type="text/css" href="stile.css">
<style type="text/css">
@import url(nuovo_stile.css);
</style>
```

Anche questa volta, le regole dello stile colleagto con @import vanno a sovrascrivere quelle ereditate dall'elemento **<LINK>**. Sono solo alcune delle strategie possibili e mi si perdoni la forse eccessiva semplificazione. Basta rendere l'dea.

Un'ultima notazione. L'uso estensivo di fogli in linea rischia di compromettere uno dei principali vantaggi dei CSS, ovvero avere pagine più leggere e facili da gestire. Intervenire nei meandri di una pagina per andare a modificare uno stile e ripetere l'operazione per quante sono le pagine del nostro sito può diventare davvero frustrante. Del resto, il loro uso è ultimamente considerato deprecato anche dal W3C.

L'attributo media

Grazie ad esso siamo in grado di impostare un foglio di stile per ogni supporto su cui la nostra pagina verrà distribuita. Una possibilità davvero interessante e che andrà sempre più diffondendosi con l'ampliarsi dei mezzi di diffusione delle pagine (X)HTML. Dove prima c'era unicamente un browser, domani potranno sempre più esserci palmari, cellulari e altri dispositivi. Per non parlare dei software usati da disabili come i browser vocali. Ciascuno di questi supporti presenta caratteristiche diverse in termini di memoria, ampiezza dello schermo e funzionalità. Riuscire ad adattare uno stile unico a tutti è praticamente impossibile, oltre che controproducente. La soluzione ideale sta quindi nella creazione di fogli di stile ad hoc.

Sintassi

L'attributo **media** può accompagnare sia l'elemento **<LINK>** che l'elemento **<STYLE>**. Ecco due esempi di sintassi:

```
<link rel="stylesheet" type="text/css" media="print" href="print.css" />
<style type="text/css" media="screen">...</style>
```

Per sfruttare a fondo questa opzione è fondamentale conoscere i diversi valori possibili per l'attributo:

- **all**. Il CSS si applica a tutti i dispositivi di visualizzazione.
- **screen**. Valore usato per la resa sui normali browser web.
- **print**. Il CSS viene applicato in fase di stampa del documento.
- **projection**. Usato per presentazioni e proiezioni a tutto schermo.
- **aural**. Da usare per dispositivi come browser a sintesi vocale.
- **braille**. Il CSS viene usato per supporti basati sull'uso del braille.
- **embossed**. Per stampanti braille.
- **handheld**. Palmari e simili.
- **tty**. Dispositivi a carattere fisso.
- **tv**.Web-tv.

Il valore di default è **all**, usato automaticamente in mancanza di una dichiarazione esplicita (ricordiamo infatti che **media** è un attributo opzionale). E' possibile usare più di un valore, ma i nomi della lista vanno separati da una virgola:

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

L'uso più tipico di questa funzionalità consiste nel collegare al documento vari fogli di stile adatti a ciascun supporto. Lo user agent che visualizzerà la pagina sarà in grado, se conforme agli standard, di caricare quello giusto:

```
<link rel="stylesheet" type="text/css" media="screen" href="screen.css" />
<link rel="stylesheet" type="text/css" media="print, tv, aural" href="print.css" />
```

Fogli di stile alternativi

In un singolo documento HTML è possibile dunque collegare più fogli di stile. Abbiamo visto nella precedente lezione come ciò possa diventare uno strumento molto potente grazie all'uso dell'opzione **media**. Un altro meccanismo da approfondire è quello che consente l'utilizzo di fogli di stile esterni alternativi.

Uso e sintassi

Gli autori della specifica (X)HTML hanno esplicitamente previsto questo scenario introducendo due valori possibili per l'attributo **rel** all'interno dell'elemento **<LINK>**:

- **stylesheet**
- **alternate stylesheet**

Il primo identifica il foglio di stile che servirà a formattare normalmente il documento, diciamo il CSS di default. Il secondo identifica un CSS come alternativo rispetto a quello standard. Ecco un esempio di codice:

```
<link rel="stylesheet" type="text/css" href="stile.css" />
<link rel="alternate stylesheet" type="text/css" href="stile_alternativo.css" />
```

Sta all'autore implementare un sistema che consenta all'utente di scegliere dinamicamente lo stile alternativo sostituendolo al primo. La via più semplice è di farlo con un semplice Javascript in grado di intervenire su proprietà e attributi dell'elemento **<LINK>**. Cliccando sul link viene caricato il foglio alternativo.

Consigli

Questa funzionalità può essere usata per vari scopi, tutti molto interessanti. Si potrebbero, ad esempio, impostare due stili alternativi consentendo all'utente di scegliere tra diversi layout per la pagina. O, ancora, creare meccanismi per modificare dinamicamente la dimensione dei caratteri, con ottimi vantaggi in termini di accessibilità. Per il resto affidatevi alla vostra fantasia.

Compatibilità

Questa lezione è soprattutto per chi si avvicina solo ora ai CSS. Non imparerete molto di concreto ma avrete coscienza di un problema fondamentale: quello della **compatibilità**. Giusto per capire il livello della questione, ho deciso di mettere all'inizio come una sorta di epigrafe questa frase di Todd Fahrner (la traduzione è mia):

> *È una vergogna che i CSS, nati per essere semplici e avvicinabili dai non-programmatori, si siano trasformati in una roba misteriosa come la Cabala!*

I CSS sono uno strumento meraviglioso. Se ricordate le date di definizione delle due specifiche potreste chiedervi spontaneamente: ma perchè non sono stati usati da subito? .

Il problema è che la piena coscienza da parte di produttori di browser e autori dell'importanza di adottare linguaggi standard ha fatto molta fatica ad affermarsi.

Netscape introduceva estensioni proprietarie, Explorer rispondeva. E i poveri web-designer a cercare soluzioni in grado di conciliare gli opposti. In tutto ciò ci si dimenticava di rafforzare il supporto dei linguaggi del W3C. E i CSS rimanevano una cosa di nicchia, per pochi adepti che non avevano nemmeno uno strumento per testarne seriamente le potenzialità

Una prima svolta, sul versante dei browser di massa, si ebbe con **Explorer 5**, il primo ad offrire un supporto adeguato del linguaggio. I predecessori di quarta generazione ne offrivano uno meno che accettabile. Lentamente, e anche grazie all'opera di gruppi di pressione come il Web Standards Project o di software-house come Opera, la consapevolezza è aumentata. Oggi possiamo contare su strumenti di navigazione che supportano la quasi totalità della specifica CSS2. Con un browser, però, che surclassa tutti gli altri in questo ambito: **Mozilla**. Microsoft ha continuato la sua opera di adeguamento, deludendo un pò con Explorer 6, ma sfornando un eccellente navigatore per Mac.

Nonostante tutto la via dei CSS è ancora lastricata di tanti problemi per gli sviluppatori. Due le maggiori fonti di difficoltà:

1. la compatibilità con i vecchi browser
2. le diverse modalità di rendering di certe proprietà

Sono problemi diversi, che vogliono quindi risposte diverse.

Retro-compatibilità

Per quanto riguarda la retro-compatibilità si hanno due strade. Si può scegliere che non vale più la pena sprecare tempo per Netscape 4 e colleghi. Semplice, brutale, ma pienamente accettabile. La valutazione delle statistiche sulle visite e sui software degli utenti è la chiave per prendere una decisione simile.

La seconda strada è quella di compromesso, del metodo detto **cross-browser**. Si tratta di elaborare strategie e di usare trucchetti in grado di preservare un minimo di compatibilità con il passato senza rinunciare ai vantaggi dei CSS per i browser recenti. Essenzialmente, comunque, non si potrà mai fare a meno di costruire fogli di stile distinti, adatti rispettivamente ai nuovi e ai vecchi browser. Nell'articolo "CSS cross-browser" avevo evidenziato tre strategie di base per risolvere il problema:

- l'uso di un CSS universale
- lo sniffing del browser con Javascript per servire a ciascuno il CSS adatto
- il metodo @import

Posto che un foglio di stile universale è assai riduttivo, gli altri due approcci si differenziano per l'origine della soluzione. Con lo sniffing tutto viene demandato ad una serie spesso complicata di script. Con il metodo @import si rimane nell'ambito (X)HTML/CSS.

Diversità di rendering

Il secondo campo di battaglia è ancora più minato del primo. Lì possiamo almeno decidere la fuga: Che Netscape 4 vada al suo destino. Qui i mal di testa sono assicurati. Il problema sta nel modo in cui ciascun browser, pur supportando una

proprietà, la applica. Un esempio gigantesco. Explorer 5 Windows supporta l'uso di parole chiave per definire le dimensioni dei font, ma lo fa a modo suo, interpretando il valore **medium** in maniera errata (rende il testo più grande del dovuto).

Qui la strategia non può essere univoca. Va valutata caso per caso. Spesso le proprietà coinvolte da queste differenze sono poco importanti e altrettanto spesso i problemi nascono su questioni avanzate come il posizionamento dinamico degli elementi. Fatto sta che l'unica soluzione possibile che posso darvi è: **testate** le vostre pagine. Sempre. Con il maggior numero di browser possibile. Valutate i risultati e se vi pare che le differenze siano trascurabili procedete senza problemi. Che una pagina possiate vederla allo stesso modo su tutti i browser è semplicemente un'utopia.

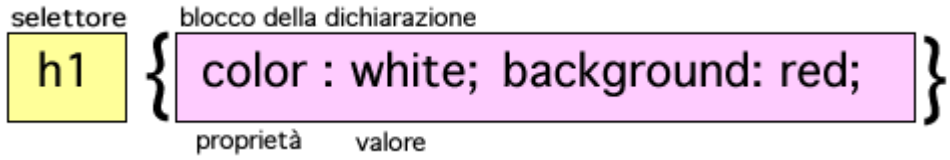
Se le differenze sono tante o tali da pregiudicare il layout della pagina affidatevi a Google o a qualche buona risorsa sui CSS per risolvere il problema. Un trucchetto prima o poi viene inventato, statene tranquilli. Due sono le risorse che vi propongo, la prima è preventiva, la seconda per i casi disperati.

Com'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. Aprite questo file, dopo averlo scaricato, con il vostro editor preferito. Tutto quello che ci trovate dentro appartiene a due tipologie di dichiarazioni: 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 definizione delle regole è posssibile 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.

```
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).

```
* { 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.

```
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 **STRONG** 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)

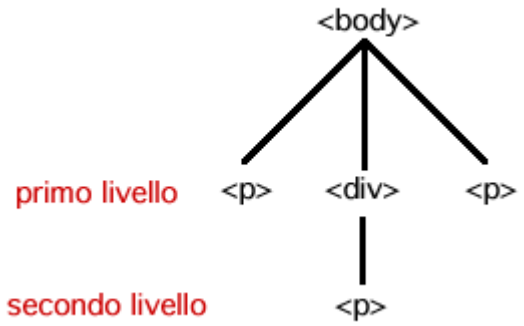
Seleziona un elemento che sia figlio diretto di un altro.

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:




```
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. 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. Dichiarazione:

```
input [ id ] {background: red;}
```

applicherà uno sfondo rosso a tutti gli elementi **input** per cui sia stato impostato un attributo **id**, 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:

```
input [ id = "text" ] { backgorund: red; }
```

applicherà un sfondo rosso a tutti gli elementi **input** che abbiano come valore dell'attributo **id** "text". 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:

```
<img src="figura1.gif" alt="Foto del Colosseo" />
```

sia questa:

```
<img src="figura2.jpg" alt="Una bella foto del Duomo di Milano" />
```

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. In questa pagina abbiamo assegnato al paragrafo un attributo **class="testorosso"**:

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

Come vedete non succede nulla. Il problema è che il valore dell'attributo **class** deve trovare una corrispondenza in un foglio di stile. In questa seconda pagina, abbiamo definito un CSS incorporato creando un selettore di tipo **classe** e assegnado ad esso il nome **testorosso**:

```
<style type="text/css">
.testorosso {
font: 12px Arial, Helvetica, sans-serif;
color: #FF0000;
}
</style>
```

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 dei 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.

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.

Le pseudo-classi

Il concetto di pseudo-classe ha qualcosa di "filosofico". Una pseudo-classe non definisce infatti un elemento ma un particolare stato di quest'ultimo. In buona sostanza imposta uno stile per un elemento al verificarsi di certe condizioni.

A livello sintattico le pseuso-classi non possono essere mai dichiarate da sole, ma per la loro stessa natura devono sempre appoggiarsi ad un selettore. Il primo costrutto che esaminiamo è quello con un elemento semplice:

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

La regola vuol dire: i collegamenti ipertestuali (**<A>**) che non siano stati visitati (**:link**) avranno il colore blue. Da qui risulta più chiaro il concetto espresso all'inizio: la pseudo-classe **:link** definisce lo stile (colore blue) solo in una determinata situzione, ovvero quando il link non è stato attivato. Appena ciò dovesse avvenire, il testo non sarà più blue, perchè la condizione originaria è venuta meno.

Torniamo alla sintassi. La pseudo-classe (tutte iniziano con i **due punti**) segue senza spazi l'elemento. Subito dopo si crea nel modo consueto il blocco delle dichiarazioni.

Una pseudo-classe può anche essere associata a selettori di tipo classe. I costrutti possibili sono due. Il primo è quello sancito nella specifica CSS1.

La pseudo-classe doveva seguire la dichiarazione della classe:

```
a.collegamento:link {color: green;}
```

Come va letta questa regola? Avranno il testo verde (green) solo i link non visitati che abbiano come attributo **class="collegamento"**. Sarà verde questo collegamento:

```
<a href="pagina.htm" class="collegamento">
```

Ma non questo:

```
<a href="pagina2.htm">
```

A partire dalla specifica CSS2 è consentita anche questa sintassi:

```
a:link.collegamento
```

in cui la classe segue la pseudo-classe. Significato e risultati sono comunque identici al primo esempio. Il primo tipo di sintassi garantisce una maggiore compatibilità con i browser più datati. Gli esempi e la sintassi presentati valgono per tutte le pseudo-classi.

:first-child

È supportata solo dai browser Gecko-based (Mozilla e Netscape 6/7) e da Explorer 5 Mac. Seleziona e formatta un elemento che si trovi ad essere il primo elemento figlio di un altro elemento.

```
<elemento>:first-child {<dichiarazioni>;}
```

Esempi

```
p:first-child {color:red;}
```

Chiariamo a partire dall'esempio il significato di questa pseudo-classe. La regola va così interpretata: assegna il colore rosso solo ai paragrafi che siano il primo elemento figlio di qualsiasi altro elemento. Esaminando questa porzione di codice:

```
<div>
<p>blah blah blah</p>
<p>blah blah blah</p>
<table>
<tr>
<td>
<p>blah blah blah</p>
<p>blah blah blah</p>
</td></tr></table>
```

si capisce che solo i paragrafi in grassetto saranno rossi, perchè sono primi figli, rispettivamente, di elemento **DIV** e di un altro elemento **TD**.

:link

Si applica solo all'elemento (X)HTML **<A>** che abbia anche l'attributo **href**. Quindi, non alle cosiddete ancore invisibili ma solo ai link ipertestuali. Definisce lo stile per questo elemento quando il collegamento punta ad un sito o ad una pagina non ancora visitati. Per sintassi ed esempi si veda sopra.

Gli pseudo-elementi

Vi sono parti in un documento (X)HTML che non sono rappresentate da nessun tag in particolare, ma che è possibile comunque modificare e formattare secondo i propri desideri con i CSS grazie ai cosiddetti pseudo-elementi. Il nome un pò criptico può essere così spiegato: nel momento in cui in un foglio di stile si costruisce una regola che ad essi fa riferimento è come se nel documento venissero inseriti nuovi elementi che hanno la caratteristica di essere appunto fittizi, presenti nel CSS ma non nel codice che definisce la struttura della pagina. Il supporto di questo tipo speciale di selettore è garantito solo dai browser più recenti, pertanto attenti all'uso.

:first-letter

Con questo selettore è possibile formattare la prima lettera di qualunque elemento contenente del testo. Le proprietà modificabili sono ovviamente tutte quelle relative al carattere e al testo, ma anche quelle legate al colore, allo sfondo, ai margini e al padding.
La sintassi di tutti gli pseudo-elementi è soggetta alla stesse regole. La più importante è che essi non possono essere dichiarati da soli, ma vanno sempre collegati ad altri selettori. Non importa di che tipo essi siano. La definizione più semplice è quella che prevede l'applicazione di uno pseudo-elemento a un selettore semplice:

```
p:first-letter {color: red; font-weight: bold;}
```

È possibile anche usare classi:

```
p.nome_classe:first-letter {color: red; font-weight: bold;}
```

O selettori di tipo ID:

```
#nome_id:first-letter {color: red; font-weight: bold;}
```

Per tutte le regole usate negli esempi il testo degli elementi presenterà la prima lettera rossa e in grassetto.

:first-line

Imposta lo stile della prima riga di un elemento contenente del testo. Valgono le stesse regole generali viste per **:first-letter**.

```
p:first-line {color: blue;}
p.nome_classe:first-letter {color: blue;}
td#nome_id:first-letter {color: blue;}
```

:before

Grazie allo pseudo-elemento **:before** è possibile inserire nel documento un contenuto non presente nello stesso. Più esattamente, con **:before** si inserisce contenuto prima dell'elemento definito nel selettore. Il contenuto da inserire si definisce tramite la proprietà **content** e può essere una semplice stringa di testo, un URL che punti a un'immagine o ad un'altro documento, un contatore numerico, semplici virgolette.

Sintassi

Facciamo un esempio per chiarire l'uso. Immaginiamo di voler inserire un'immagine particolare prima di ogni titolo **<H1>**. Useremo in questo modo lo pseudo-elemento **:before**:

```
h1:before {content: url(immagine.gif);}
```

Volendo inserire una stringa essa va racchiusa tra virgolette:

```
h1:before {content: "Il titolo è...";}
```

Non è supportato da Explorer per Windows.

:after

Pseudo-elemento complementare a **:before**. L'unica differenza è che il contenuto viene inserito dopo l'elemento specificato nel selettore.

```
h1:after {content: url(immagine.gif);}
h1#nome_id {content: "Testo del titolo";}
```

Le @-rules

Le cosiddette **@-rules** sono tipi particolari di costrutti che hanno una caratteristica comune: sono tutti introdotti dal simbolo della chiocciola. La pronuncia è identica a quella usata per gli indirizzi e-mail, ovvero **at-import**, **at-media**, etc.

Per quanto riguarda la funzione rappresentano vie alternative, ma spesso più flessibili e potenti, per realizzare cose attuabili in altri modi. Abbiamo già visto, nella lezione dedicata a come inserire i css in un documento, come il costrutto **@import** sia una valida alternativa all'elemento **<LINK>** per collegare fogli di stile esterni.
A livello sintattico le @-rules posono essere definite o nel corpo del documento, per l'esattezza all'interno dell'elemento **<STYLE>**:

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

o direttamente nel codice di un CSS esterno.

Ognuna delle diverse @-rules presenta poi scopi e criteri diversi di costruzione.

@import

@import viene usata innanzitutto per collegare un foglio di stile esterno al documento. La sintassi generica è la seguente:

```
<style type="text/css">
@import url(foglio_di_stile.css);
</style>
```

Come si vede la direttiva è accompagnata dall'indicazione **url** che precede l'indirizzo del CSS. Questo è racchiuso tra parentesi tonde. La @-rule deve presentare alla fine il punto e virgola di chiusura.

Altre possibili sintassi, accettate dai browser recenti, ma problematiche con quelli più datati:

URL con virgolette:

```
@import url("stile.css");
```

Direttiva senza l'indicazione **url**:

```
@import "stile.css";
```

L'url del foglio di stile può essere relativo, come negli esempi precedente, o assoluto, come in questo:

```
<style type="text/css">
@import url(http://www.miosito.it/foglio_di_stile.css);
</style>
```

Un principio fondamentale è che all'interno del tag **<STYLE>** @import deve essere la prima regola definita. In pratica, se si scrive così la direttiva non funzionerà e il CSS non sarà caricato in quanto @import è preceduta da una regola che applica uno stile all'elemento **<H1>**:

```
<style type="text/css">
h1 { color: black; }
@import url(foglio_di_stile.css);
</style>
```

Il modo giusto è dunque questo:

```
<style type="text/css">
@import url(foglio_di_stile.css);
h1 { color: black; }
</style>
```

È possibile invece importare all'interno di un singolo tag **<STYLE>** più fogli di stile:

```
<style type="text/css">
@import url(foglio_di_stile.css);
@import url(foglio_di_stile2.css);
</style>
```

Le regole di tutti i CSS collegati in questo modo saranno applicate al documento secondo l'ordine e i criteri stabiliti dalle norme sull'importanza e la specificità che vedremo tra poco in una prossima lezione.

Un uso interessante di @import è che può essere usata anche all'interno di un foglio di stile per incorporare un altro CSS esterno:

```
@import url(foglio.css)
h1 { color: black; }
```

Supponendo che questo sia un CSS, avremo il risultato di incorporare al suo interno il contenuto del secondo foglio di stile (**foglio.css**).

È possibile definire all'interno della direttiva @import anche il supporto cui applicare il CSS, in modo simile a quanto abbiamo visto a proposito dell'attributo **media**. Per fare ciò basta far seguire all'url del CSS l'indicazione di uno dei media previsti nella specifica:

```
<style type="text/css">
@import url(foglio_stampa.css) print;
@import url(foglio_schermo.css) screen, handheld;
</style>
```

@media

Lo stesso risultato (un CSS per ogni dispositivo) si può ottenere con la direttiva **@media**. La sintassi generica è questa:

```
@media <valore> {regole CSS}
```

@media va quindi seguito dal nome di uno dei supporti scelti come target specifico e dalle regole di stile racchiuse tra parentesi graffe. Esempio:

```
<style type="text/css">
@media screen {
h1 {color: black;}
p {color: red;}
}
@media print {
h1 {color: red;}
p {color: black;}
}
</style>
```

Anche questo costrutto va inserito, come si vede, all'interno del tag **<STYLE>**. Un uso più potente, però, è quello di inserire la direttiva nel codice di un foglio esterno.

Immaginate di avere un foglio di stile per formattare le vostre pagine e che vogliate impostare colori diversi per un elemento a seconda che si visualizzi il testo sullo schermo o su carta stampata. Invece di costruire due fogli di stile potete creare in un solo CSS esterno queste regole e diversificare l'aspetto dell'elemento:

```
@media print {
h1 {
color: black;
}
}
@media screen {
h1 {
color: red;
}
}
```

@charset

La direttiva **@charset** serve a specificare l'impostazione relativa alla codifica dei caratteri di un documento. Svolge in pratica la stessa funzione che in (X)HTML è possibile ottenere con l'uso dei meta-tag:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

La sintassi è semplicissima:

```
@charset "iso-8859-1"
```

La direttiva non può essere usata in fogli incorporati. Quando si vuole usarla in un foglio esterno essa deve essere la prima dichiarazione del CSS.

@font-face

La direttiva **@font-face** può essere usata per descrivere un font usato nel documento. È un argomento molto complesso e conoscere i dettagli della questione non aggiunge nulla che possa davvero accrescere la vostra conoscenza dei CSS. In genere potrebbe essere usata per specificare l'url di un carattere particolare da scaricare su una macchina che non lo veda presente tra i suoi font. La sintassi classica è infatti questa:

```
@font-face {
font-family: Santiago;
src: url("http://www.mioserver.it/fonts/santiago.tt");
}
```

Il supporto non è garantito e l'utilità effettiva poco più che nulla.

Valori e unità di misura

In questa lezione vedremo quali sono i tipi di valore e le unità di misura con cui è possibile impostare le tante proprietà dei CSS. Prima di tutto, però, è opportuno spiegare due fondamentali regole di base.

1. I valori di una proprietà non vanno **mai messi tra virgolette**. Uniche eccezioni i valori espressi da stringhe di testo e i nomi dei font formati da più di una parola (esempio: "Times New Roman").

2. Quando si usano valori numerici con unità di misura, non bisogna lasciare spazio tra numero e sigla dell'unità. E' corretto quindi scrivere **15px** oppure **5em**. E' invece sbagliato usare **15 px** o **5 em**. In questi casi la regola sarà ignorata o mal interpretata.

Unità per le dimensioni

Questo è la lista delle unità di misura usate per definire dimensioni, spazi o distanze. Nella pratica comune solo alcune di esse sono realmente usate. Le elenchiamo comunque tutte per completezza.

- **in (inches - pollici)**: classica misura del sistema metrico americano. Praticamente nullo il suo valore su un supporto come un browser web viste le variabili relative a risoluzione e ampiezza dei monitor.
- **cm (centimetri)**: stesso discorso visto per i pollici, la difficoltà maggiore sta nella resa su monitor, che è altra cosa rispetto alla carta stampata.
- **mm (millimetri)**: vedi centimetri.
- **pt (points - punti)**: unità di misura tipografica destinata essenzialmente a definire la dimensione dei font.
- **pc (picas)**: unità poco usata. 1 pica equivale a 12 punti.
- **em (em-height)**: unità di misura spesso usata dagli autori CSS. 1 em equivale all'altezza media di un carattere per un dato font. E' un unità di misura relativa.
- **ex (ex-height)**: poco usata. 1 ex equivale all'altezza del carattere x minuscolo del font scelto.
- **px (pixels)**: unità di misura ideale su monitor. E' quella più usata e facile da comprendere.

Percentuale

Un valore espresso in percentuale è da considerare sempre relativo rispetto ad un altro valore, in genere quello espresso per l'elemento parente. Si esprime con un valore numerico seguito (senza spazi) dal segno di percentuale: **60%** è pertanto corretto, **60 %** no.

Colori

Sui diversi modi per esprimere il valore di un colore si veda la lezione su colore e CSS.

Stringhe

Alcune proprietà dei CSS possono avere come valore una stringa di testo da inserire come contenuto aggiunto nel documento. I valori espressi da stringhe vanno sempre racchiusi tra virgolette. Le proprietà in questione sono tre: **content, quotes, text-align** (ma solo per le tabelle definite con i CSS).

Valori URI

Si tratta di URL che puntano a documenti esterni (in genere immagini, come negli sfondi). Possono essere URL assoluti o relativi. In questo caso il path fa sempre riferimento alla posizione del foglio di stile e non del documento HTML. La definizione dell'indirizzo è sempre introdotta dalla parola chiave **url** e va inserita tra parentesi tonde senza virgolette. Esempio: **url(immagini/sfondo.gif)**.

Unità per gli angoli

Due proprietà comprese nella sezione dei CSS dedicata ai dispositivi audio possono essere espresse con unità di misura relative agli angoli. Le due proprietà sono **azimuth** e **elevation**. Le unità di misura queste:

- **deg (degree - grado)**: descrive l'ampiezza di un angolo (es. 90deg).
- **grad (gradians)**: descrive l'ampiezza di un angolo su una scala 1-400 (es. 100grad = 90deg)
- **rad (radians)**: decrive l'ampiezza di un angolo su una scala 1-pi greco

Unità di tempo

Anche le unità di tempo trovano spazio solo negli stili audio. Sono usate in genere per impostare la pausa tra le parole lette da un sintetizzatore vocale. Si applicano solo a queste tre proprietà: **pause**, **pause-after**, **pause-before**. Le unità di misura sono:

- **s (secondi)**
- **ms (millisecondi)**

Unità di frequenza

Usate solo negli stili audio, definiscono la frequenza del segnale:

- **hz (Hertz)**
- **khz (Kilohertz)**

Ereditarietà, cascade e conflitti tra stili

Se c'è una lezione di questa guida che dovreste stampare e conservare in una cartellina è quella che state per leggere. Stiamo per entrare nel cuore del meccanismo di funzionamento dei CSS. Sono regole un pò complesse, ma basilari. Quindi attenzione massima. Gli esempi che accompagnano la parte teorica vi aiuteranno nel percorso di apprendimento: usateli sempre.

Ereditarietà

Il primo concetto è quello di **ereditarietà**. Secondo questo meccanismo le impostazioni stilistiche applicate ad un elemento ricadono anche sui suoi discendenti. Almeno fino a quando, per un elemento discendente, non si imposti esplicitamente un valore diverso per quella proprietà. Se impostiamo il colore rosso per il testo (**color: red;**) a livello dell'elemento **BODY** tutti gli altri elementi suoi discendenti erediteranno questa impostazione. Ma se ad un certo punto definiamo nel codice del CSS un selettore con la proprietà **color: black**; l'ereditarietà viene spezzata.

Non tutte le proprietà sono ereditate e lo renderemo esplicito nell'analisi di ciascuna di esse. In genere sono quelle attinenti la formattazione del box model: margini, bordi, padding, background le più importanti. Il motivo è semplice. Ereditare un bordo è semplicemente senza senso. Se ne imposto uno, diciamo, per un paragrafo sarebbe assurdo che un elemento **<A>** o un testo in grassetto vengano circondati dallo stesso bordo!

Peso e origine

Da qui in avanti affronteremo un'altra serie di concetti fondamentali tutti riconducibili comunque ad uno stesso ambito: i conflitti possibili tra gli stili e le regole. Tenteremo in pratica di rispondere a quesiti come questo. Se definisco queste regole in un CSS:

```
p {color: black;}
.testo {color: red;}
```

E in una pagina HTML scrivo questo codice:

```
<p class="testo">Testo del paragrafo</p>
```

Perchè il testo del paragrafo sarà rosso e non nero? Perchè il selettore classe prevale su quello semplice con elemento?

Il primo concetto da imparare è quello di **peso**. Si riferisce alla maggiore o minore importanza da assegnare a ciascuna regola. Un primo criterio di importanza è dato dall'**origine del foglio di stile.** Quando visualizziamo una pagina (X)HTML possono entrare in gioco nel modificare lo stile degli elementi tre diversi fogli di stile:

- **foglio dell'autore**
- **foglio dell'utente**
- **foglio predefinito del browser**

In ordine di importanza avremo: foglio dell'autore, foglio dell'utente, foglio predefinito del browser.

Tutti i software di navigazione di ultima generazione consentono una gestione di questo aspetto. E' possibile, ad esempio, far sì che il browser ignori i CSS definiti dall'autore delle pagine e formattare queste ultime con un CSS realizzato dall'utente. E ancora, come vedremo, è possibile modificare questa gerarchia con l'uso della parola chiave **!important**. Di base, però, l'ordine è quello definito qui sopra.

Specificità

La specificità, come il nome può suggerire, descrive il peso relativo delle varie regole all'interno di un foglio di stile. Esistono regole ben precise per calcolarla e sono quelle che applica lo user agent di un browser quando si trova davanti ad un CSS.

I fattori del calcolo sono tre e ciascuno di essi rappresenta il valore di una tripletta. Per prima cosa si conta il numero di selettori **ID** presenti nella regola. Si passa quindi a verificare la presenza di **classi** e **pseudo-classi**. Infine si conta il numero di **elementi** definiti nella regola. Mai come in questo caso urge l'esempio.

Prima regola:

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

Calcolo: un **ID**, 0 classi, 0 elementi. Tripletta dei valori: 1-0-0

```
.classe1 {background: #C00;}
```

0 ID, 1 **classe**, 0 elementi. Tripletta: 0-1-0

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

0 ID, 0 classi, un **elemento**. Tripletta: 0-0-1

Il peso specifico della prima regola è il maggiore. Quello dell'ultima il minore. In pratica: gli ID pesano più delle classi che pesano più dei singoli elementi. Non commettete l'errore di valutare il numero più grande a prescindere dalla sua posizione. Questa regola presenta la seguente specificità 1-0-0:

```
#paragrafo {color: green;}
```

ed è più importante di questa che ha i seguenti valori 0-0-2:

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

Il concetto di cascade

Ed ora la summa di tutto quello cha abbiamo detto. Il concetto e il meccanismo di **cascade** spiegato con parole semplici. E che sia un elemento chiave lo capite dal nome stesso di quello che state studiando: Cascading Style Sheets. Tenteremo di ricostruire il procedimento di un browser quando incontra un foglio di stile e lo rende sul monitor del nostro computer.

1. Per prima cosa controlla il target stabilito con l'attributo **media** o con dichiarazioni equivalenti. Scarta quindi tutti gli stili riferiti alla stampa o ad altri supporti. Allo stesso tempo scarta tutte le regole che non trovino corrispondenza negli elementi strutturali del documento.

2. Comincia ad ordinare per **peso** e **origine** secondo le regole viste sopra. C'è un CSS definito dall'autore? Userà quello. Altrimenti verificherà la presenza di un foglio di stile utente e in sua assenza applicherà le sue regole stilistiche predefinite.

3. Quindi calcola la specificità dei selettori e in caso di conflitto tra regole usa questo criterio di prevalenza.

4. Se non ci sono conflitti o se peso, origine e specificità coincidono, viene applicata la regola più vicina all'elemento nel codice del documento. L'ordine, se le dichiarazioni degli stili sono fatte nell'ordine più corretto e logico, è quindi il seguente: gli stili **in linea** prevalgono su quelli **incorporati** che a loro volta prevalgono su quelli **collegati**.

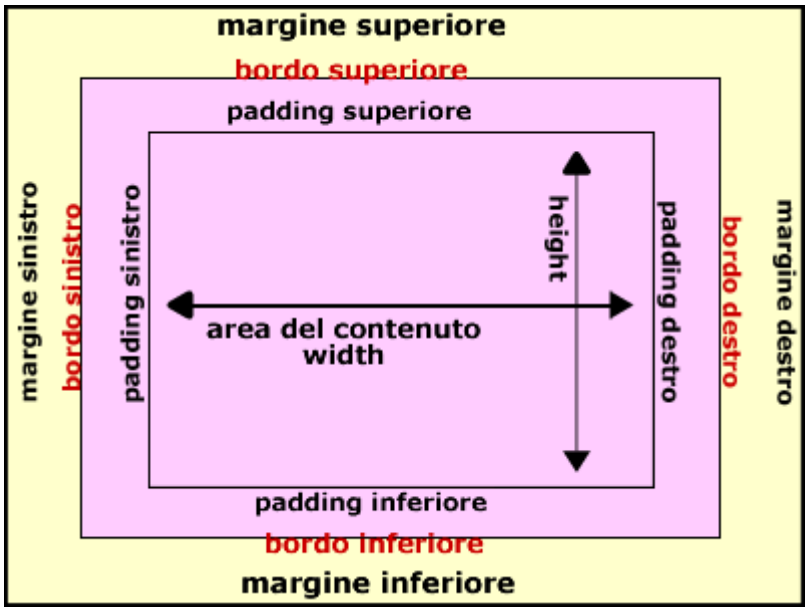
Importanza

Ed ora il concetto di importanza. Semplice e lineare la regola: se una dichiarazione viene accompagnata dalla parola chiave **!important** essa balza al primo posto nell'ordine di applicazione a prescindere da peso, origine, specificità e ordine.

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:

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

Come si vede infatti nella figura margini, padding e bordi devono considerarsi a tutti gli effetti parte dell'area complessiva 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 meccansimo del cosiddetto **margin collapsing**. Tale meccanismo non si applica ai box adiacenti in senso orizzontale.

Gestione 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 precdenza. 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:

1. **il colore di primo piano** (foreground color)
2. **il colore di sfondo** (background color)
3. **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 parente. 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

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**:

1. background-color
2. background-image
3. background-repeat
4. background-attachment
5. 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 analizeremo 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.

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

Valori

- **un qualunque colore**
- la parola chiave **transparent**

Usando **transparent** come valore un elemento avrà come colore quello dell'elemento parente. Nell'esempio: il colore di #div2 è impostato su **transparent**, per cui esso sarà, per l'appunto, trasparente, lasciando emergere il colore dell'elemento parente (#div1).

Esempi

```
body { background-color: white; }
p { background-color: #FFFFFF; }
.classe1 { 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.

```
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 body {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 disabilitate, 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.

```
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.

```
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 pò 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.

```
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**

L'esempio renderà tutto più chiaro.

```
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.

```
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 **<FONT>**: 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.

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 mancanza 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.

```
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.

```
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.titolo {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.

```
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.

```
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.

```
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
- font di sistema

```
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.

Usare i font di sistema

All'interno della proprietà font è possibile impostare, invece della normale lista di caratteri, un valore corrispondente ai cosiddetti **font di sistema**. Si tratta del tipo di carattere che usiamo sul nostro sistema operativo per definire l'aspetto di elementi dell'interfaccia come barre dei menu, icone o barra di stato. In particolare, le parole chiave relative ai font di sistema sono 6 e riguardano:

1. **caption**: carattere usato per pulsanti e menu a tendina
2. **icon**: carattere usato per la descrizione delle icone
3. **menu**: carattere usato nei menu delle varie finestre
4. **message-box**: usato per i gli alert e gli altri box con messaggi di sistema
5. **small-caption**: carattere per i controlli più piccoli
6. **status-bar**: il font usato per la barra di stato

Se si intende usare uno dei font di sistema non è necessario specificare altri valori oltre alla keyword. Il nostro testo, in pratica, erediterà le impostazioni definite nel sistema operativo relative a carattere e dimensione.

Esempi

```
p {font: menu;}
div {font: status-bar;}
```

Text-align

La proprietà serve a impostare l'allineamento del testo. E' ereditata e si applica a tutti gli elementi.

```
selettore { text-align: <valore>; }
```

Valori

- **left**. Allinea il testo a sinistra
- **right**. Allinea il testo a destra
- **center**. Centra il testo
- **justify**. Giustifica il testo

Text-decoration

Imposta particolari decorazioni e stili per il testo. Ereditabile e applicabile a tutti gli elementi.

```
p {text-decoration: <valore> o <valori>;}
```

Valori

- **none**. Il testo non avrà alcuna decorazione particolare
- **underline**. Il testo sarà sottolineato
- **overline** Il testo avrà una linea superiore
- **line-through**. Il testo sarà attraversato da una linea orizzontale al centro
- **blink**. Testo lampeggiante

Esempi

```
p {text-decoration: none;}
a {text-decoration: underline;}
```

Gestione del testo: proprietà avanzate

font-size-adjust

Questa proprietà può servire a migliorare la leggibilità di un carattere alternativo tra quelli indicati con la proprietà font-family. La proprietà è supportata solo dai browser Gecko-based (Mozilla, Netscape 6/7) ed è ereditata.

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

Valori

- **none**. Non avviene nessun aggiustamento. Valore di default.
- **valore numerico.** Indica il fattore di aggiustamento del font.

Esempi

```
p {font-size-adjust: 0.50;}
```

font-stretch

Consente di rendere il testo più o meno espanso del normale. Proprietà non supportata da nessun browser. Ereditata.

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

Valori

- **normal**. Valore di default.
- **parole chiave** che indicano diversi gradi di espansione del testo.

Nell'ordine, dal grado di maggiore espansione al minore sono:

ultra-expanded, **extra-expanded**, **expanded**, **semi-expanded**, **normal**, **semi-condensed**, **condensed**, **extra-condensed**, **ultra-condensed**

Oltre a queste abbiamo due parole chiave con funzione relativa: **wider** e **narrower**.

Esempi

```
p {font-stretch: ultra-expanded;}
```

font-variant

Consente di trasformare il testo in maiuscoletto (lettere in maiuscolo rese con dimensioni uguali ai caratteri minuscoli). Proprietà ben supportata ed ereditata.

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

Valori

- **normal**. Il testo ha il suo aspetto normale. Valore di default.
- **small-caps**. Trasforma il testo in maiuscoletto.

Esempi

```
h2 {font-variant: small-caps;}
```

text-indent

Definisce l'indentazione della prima riga in ogni elemento contenente del testo. Proprietà ben supportata ed ereditata.

```
selettore {text-indent: <valore>;}
```

Valori

- **un valore numerico con unità di misura**
- **un valore in percentuale**

Come al solito, il valore con unità di misura è assoluto, quello in percentuale è relativo. In questo caso il valore è relativo alla larghezza dell'area del contenuto. In pratica, se in un paragrafo largo 200px setto un'indentazione uguale al 10%, essa sarà uguale a 20px.

Esempi

```
p {text-indent: 10px;}
div.contenuto {text-indent: 10%;}
```

text-shadow

Crea un'ombreggiatura per il testo specificato. La proprietà non è supportata da nessun browser e non è ereditata.

```
selettore {text-shadow: <valore colore> <valore x> <valore y> <valore blur>;}
```

Esempi

```
h1 {text shadow: blue 2px 2px 3px;}
```

text-transform

La proprietà serve a cambiare gli attributi del testo relativamente a tre aspetti: maiuscolo, minuscolo, prima lettera maiuscola. Proprietà ben supportata e comodissima se avete lunghe porzioni di testo tutto in maiuscolo da trasformare (o viceversa). Ereditata.

```
selettore {text-transform: <valore>;}
```

Valori

- **none**. Valore di default. Nessuna trasformazine viene applicata.
- **capitalize**. La prima lettera di ogni parola viene trasformata in maiuscolo.
- **uppercase**. Tutto il testo è maiuscolo.
- **lowercase**. Tutto il testo è minuscolo.

Esempi

```
p {text-transform: capitalize;}
h1 {text-transform: uppercase;}
```

white-space

Questa proprietà serve a gestire il trattamento degli spazi bianchi presenti in un elemento. E' ereditata.

E' una cosa nota a chi ha un minimo di dimestichezza con il codice HTML. Se nel codice di una pagina lasciate spazi bianchi tra le parole, essi saranno automaticamente convertiti in un solo spazio quando la pagina viene visualizzata nel browser. A meno di non racchiudere il testo con tag come **<PRE>** o **<CODE>**. In questo caso spazi bianchi e fine riga saranno rispettati e manterranno l'aspetto che essi hanno nel codice. Uguale meccanismo è implementabile con i CSS con white-space.

```
selettore {white-space: <valore>;}
```

Valori

- **none**. Valore di default. Gli spazi bianchi sono ridotti a uno.
- **pre**. Gli spazi banchi e l'inizio di nuove righe sono mantenuti così come sono nel codice
- **nowrap**. Gli spazi bianchi sono ridotti a uno e l'andata a capo è disabilitata

Esempi

```
p {white-space: pre;}
div {white-space: nowrap;}
```

letter-spacing

Aumenta lo spazio tra le lettere di una parola. Proprietà ben supportata ed ereditata.

```
selettore { letter-spacing: <valore>; }
```

Valori

- **normal**. Valore di default. Le lettere mantengono il loro spazio normale.
- **un valore numerico con unità di misura**. Le lettere saranno spaziate secondo la distanza impostata.

E' possibile anche impostare valori negativi. Ciò farà sì che le lettere appaiano sempre più compresse, con il rischio ovvio dell'illegibilità.

Esempi

```
h1 { letter-spacing: 5px; }
```

word-spacing

Proprietà complementare a letter-spacing. Serve ad aumentare lo spazio tra le parole comprese in un elemento. Proprietà ben supportata ed ereditata.

```
selettore { word-spacing: <valore>; }
```

Valori

- **normal**. Valore di default. Le parole mantengono il loro spazio normale.
- **un valore numerico con unità di misura**. Le parole saranno spaziate secondo la distanza impostata.

Esempi

```
p { word-spacing: 1.2em; }
div.testo { word-spacing: 5px; }
```

Gestione delle dimensioni: altezza

Con le prossime cinque lezioni vedremo come sia possibile definire e controllare le diverse proprietà del box-model. 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

```
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. Explorer non fa altro che ignorare l'altezza impostata estendendola. Mozilla e Opera rispettano la regola, ma il contenuto eccedente va a sovrapporsi agli elementi vicini o sottostanti. 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%;}
```

overflow

Esiste un modo per gestire il contenuto che superi i limiti imposti con **height**. Usare la proprietà **overflow**. Si applica a tutti gli elementi blocco e non è ereditata.

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

Valori

- **visible**. Valore iniziale. Il contenuto eccedente rimane visibile (con i problemi visti sopra).
- **hidden**. Il contenuto eccedente non viene mostrato.
- **scroll**. Il browser crea barre di scorrimento che consentono di fruire del contenuto eccedente.
- **auto**. Il browser tratta il contenuto eccedente secondo le sue impostazioni predefinite. Di norma dovrebbe mostrare una barra di scorrimento laterale.

Esempi

```
div {overflow: auto;}
```

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.

```
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.

```
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**.

E' possibile ovviamente che i due valori coincidano. Accade quando di un particolare elemento non si impostino margini, paddding e bordi; o quando il valore di tali proprietà sia uguale a 0. I due box dovrebbero avere la stessa larghezza. Ma se usate Explorer 5 o 5.5 su Windows noterete che non è così.

```
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.

```
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.

```
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.

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

Un uso interessante del valore **auto** per i lati sinistro e destro è quello che consente di centrare in tal modo un elemento rispetto alla pagina o al box contenitore.

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%;}
```

Ricordiamo un'ultima interessante applicazione di margini e padding. Ogni browser inserisce un margine più o meno ampio tra il bordo della finestra e l'area del contenuto. In HTML era possibile eliminarlo tramite una serie di attributi (e qui Explorer e Netscape avevano sintassi diverse) dell'elemento body:

```
<body leftmargin="0" topmargin="0" marginwidth="0" marginheight="0">
```

La stessa cosa si può ottenere con i CSS:

```
body {
margin: 0px;
padding: 0px; }
```

I bordi

Riuscire a definire l'aspetto dei bordi è un'altra delle cose per cui vale la pena imparare ad usare i CSS. L'impatto visivo, se guidati dal buon gusto, è di sicuro effetto: la pagina acquista rigore e una suddivisione più logica, estetica e razionalità strutturale possono trovare un felice connubio. 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:

1. **il colore (color)**
2. **lo stile (style)**
3. **lo spessore (width)**

I valori possibili per il colore sono:

- **un qualsiasi colore**
- **la parola chiave inherit**

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 0.
- **hidden**. Equivalente a none
- **dotted**
- **dashed**
- **solid**
- **double**.
- **groove**
- **ridge**
- **inset**
- **outset**

Ciascuna definisce un particolare aspetto del bordo.

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

```
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.

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.

```
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 (**<OL>**) o non ordinata (**<UL>**), ma ai singoli elementi che le compongono, ovvero l'elemento **<LI>**. 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**.

```
<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.

Una considerazione "estetica" di cui tenere conto è la dimensione delle immagini. Devono essere sempre adeguate alla dimensione del testo.

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**.

```
<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;}
#lista 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**.

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

Valori

La scelta dei valori possibili è lunghissima. Definiamo nei dettagli solo i più importanti, limitandoci a citare quelle che fanno riferimento a sistemi di scrittura non occidentali. Tali valori sono stati inseriti per venire incontro alle esigenze di numerazione di lingue come l'ebraico o il giapponese, che usano sistemi diversi dal nostro.

- **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.

Si riferiscono a sistemi di conteggio internazionali i seguenti valori:

```
hebrew | armenian | georgian | cjk-ideographic | hiragana | katakana | hiragana-
hiroha | katakana-hiroha
```

I diversi tipi di marcatori si riferiscono ciascuno, logicamente, a liste ordinate o non ordinate. Useremmo, ad esempio, **square**, per un **UL** e **decimal** per un **OL**. In realtà, se si imposta l'aspetto del marcatore con i CSS, l'impostazione definita in (X)HTML viene ignorata. Pertanto, posso avere una lista non ordinata con list-item decimali, il consiglio è, comunque, di attenervi al buon senso e di rispettare la logica degli elementi.

list-style

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

```
<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 li {list-style: square inside;}
ul.lista1 li {list-style: outside url(imamgine.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.

```
<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**.
- **run-in**. L'elemento viene incorporato e inserito all'inizio del blocco seguente. L'effetto dovrebbe essere simile a quello visto per le liste quando si usa il valore **inside** per il marcatore. Il valore è supportato solo da Opera 5/6 e parzialmente da Explorer 5 Mac.
- **compact**. L'elemento viene piazzato al fianco di un altro. Non supportato da nessun browser.
- **marker**. Questo valore fa sì che il contenuto generato con gli pseudo-elementi **:before** e **:after** sia trattato come un marcatore di liste. Non supportato da nessun browser.
- **none**. L'elemento non viene mostrato. O meglio: è come se non fosse nemmeno presente nel documento, in quanto non genera alcun box. Diversa, come vedremo, la proprietà **visibility:hidden**, che invece si limita a nascondere l'elemento.

Una serie di valori fa riferimento alla possibilità di impostare il display degli elementi come elementi di una tabella. tali valori trovano un supporto adeguato solo in Mozilla:

```
table | inline-table | table-cell | table-row | table-row-group | table-column | table-column-group | table-header-group | table-footer-group | table-caption
```

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;}
```

Per non parlare dell'utilità del valore **none**. Volete una pagina alternativa fatta solo di testo in 10 secondi? Facile. Costruite un CSS alternativo dove imposterete questa regola:

```
img {display: none;}
```

Nella pagina principale mettete un bel link: "Versione solo testo" e ci attaccate uno javascipt per caricare il CSS alternativo. Fatto.

La vera utilità della proprietà **display**, emerge comunque in linguaggi non strutturali con 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 docuemnto e spostarlo su uno dei lati (destro o sinistro) 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**.

```
<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.foto {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

```
h1 {clear: both;}
```

Posizionamento degli elementi

position

position è la proprietà fondamentale per la gestione della posizione degli elementi, di cui determina la modalità di presentazione sulla pagina. Si applica a tutti gli elementi e non è ereditata.

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

Valori

- **static**.
- **absolute**
- **fixed**
- **relative**

static

E' il valore di default, quello **predefinito** per tutti gli elementi non posizionati secondo un altro metodo. Rappresenta la posizione normale che ciascuno di essi occupa nel flusso del documento.

absolute

L'elemento, o meglio, il box dell'elemento viene rimosso dal flusso del documento ed è posizionato in base alle coordinate fornite con le proprietà **top, left, right o bottom**. Il posizionamento avviene sempre rispetto al **box contenitore** dell'elemento. Questo è rappresentato dal primo elemento **antenato** (**ancestor**) che abbia un posizionamento diverso da **static**. Se tale elemento non esiste il posizionamento avviene in base all'elemento radice **HTML**, che in condizioni standard coincide con l'area del browser che contiene il documento e che ha inizio dall'angolo superiore sinistro di tale area. Un elemento posizionato in modo assoluto scorre insieme al resto del documento.

fixed

Usando questo valore il box dell'elemento viene, come per **absolute**, sottratto al normale flusso del documento. La differenza sta nel fatto che per **fixed** il box contenitore è **sempre** il cosiddetto *viewport*. Con questo termine si intende la finestra principale del browser, ovvero l'area del contenuto. Altra differenza fondamentale: un box posizionato con **fixed** non scorre con il resto del documento. Rimane, appunto, fisso al suo post. L'effetto è lo stesso che si può ottenere con l'uso dei frame, in cui una parte della pagina rimane fissa e il resto scorre. Solo che qui il documento è solo uno. Purtroppo, il valore non è supportato da Explorer su Windows.

relative

L'elemento viene posizionato relativamente al suo box contenitore. In questo caso il box contenitore è il posto che l'elemento avrebbe occupato nel normale flusso del documento. La posizione viene anche qui impostata con le proprietà **top, left,**

bottom, right. Ma qui esse non indicano un punto preciso, ma l'ammontare dello spostamento in senso orizzontale e verticale rispetto al box contenitore.

Gestione della posizione

Passiamo ora all'analisi delle proprietà che concretamente definiscono **dove** un elemento posizionato va a collocarsi, dal momento che che con **position** stabiliamo solo il **come**. Le proprietà **visibility**, **clip** e **z-index** influiscono invece sull'aspetto visuale dei box definendone la visibilità e la relazione con gli altri box presenti nella pagina.

top

Il significato di **top** cambia secondo la modalità di posizionamento. Per gli elementi posizionati con **absolute** o **fixed** definisce la distanza verticale rispetto al bordo superiore dell'elemento contenitore. Per gli elementi posizionati con **relative** stabilisce invece l'ammontare dello spostamento rispetto al lato superiore della posizione originaria. In questo caso, usando valori positivi il box viene spostato sotto, mentre con valori negativi lo spostamento avviene verso l'alto.

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

Valori

- **un valore numerico con unità di misura**
- **un valore in percentuale** La percentuale è relativa all'altezza dell'elemento contenitore.
- **auto**

Esempi

```
div {top: 10px;}
p {top: 10%;}
```

left

Per gli elementi con posizione assoluta o fissa definisce la distanza dal bordo sinistro del box contenitore. Per quelli con posizione relativa stabilisce lo spostamento rispetto al lato sinistro della posizione originaria. Valori positivi spostano l'elemento verso destra, valori negativi verso sinistra.

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

Valori

- **un valore numerico con unità di misura**
- **un valore in percentuale** La percentuale è relativa alla larghezza dell'elemento contenitore.
- **auto**

Esempi

```
div {left: 30px;}
```

bottom

Per i box con posizione assoluta o fissa definisce la distanza dal bordo inferiore dell'elemento contenitore. Per quelli posizionati relativamente lo spostamento rispetto al lato inferiore della posizione originaria.

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

Valori

- **un valore numerico con unità di misura**
- **un valore in percentuale**
- **auto**

Esempi

```
div {bottom: 50px;}
```

right

Per i box con posizione assoluta o fissa definisce la distanza dal bordo destro dell'elemento contenitore. Per quelli posizionati relativamente lo spostamento rispetto al lato destro della posizione originaria.

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

Valori

- **un valore numerico con unità di misura**
- **un valore in percentuale**
- **auto**

Esempi

```
div {right: 50px;}
```

Per quantpo riguarda la definizione della posizione, va detto che essa è in ogni caso definita da due coordinate. In genere si usano le proprietà **top** e **left**.

visibility

Questa proprietà determina se un elemento debab essere visibile o nascosto. Si applica a tutti gli elementi e non è ereditata.

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

Valori

- **visible**. L'elemento è visibile. Valore di default.
- **hidden**. L'elemento è nascosto, ma mantiene il suo posto nel layout dove apparirà come una zona vuota. In ciò è diverso dal valore **none** della proprietà **display**.

Esempi

```
div.box {visibility: hidden;}
p {visibility: visible;}
```

clip

Definisce la parte di un elemento posizionato che deve essere visibile. Nella definizione che ne è stata data con CSS2 non è supportata da nessun browser.

z-index

Con questa proprietà si imposta l'ordine di posizionamento dei vari elementi sulla base di una scala di livelli. E' un meccanismo simile a quello dei layer sovrapposti di Photoshop ed è utile nel contesto del posizionamento dinamico. In seguito ad un posizionamento, infatti, è possibile che un elemento si sovrapponga ad un altro rendendolo illeggibile. Impostando lo z-index è possibile modificare l'ordine di tutti gli elementi.

```
<selettore> {z-index: <valore>;}
```

Valori

- **auto**. L'ordine dei livelli è uguale per tutti gli elementi.
- **un valore numerico**. Un valore superiore indica un livello superiore .

Esempi

```
div#box1 {z-index: 34;}
```

Agire sulle tabelle

Il supporto, chiariamolo subito, non è uniforme per tutti i browser. Mozilla brilla anche qui per la sua fedeltà agli standard, mentre Explorer Windows cade su parecchi aspetti. Nell'analisi delle proprietà ci limiteremo all'enunciazione dei concetti di base, tralasciando la miriade di regole particolari e spesso complesse che ne governano il comportamento. Del resto, l'uso di queste proprietà è davvero molto ridotto, dal momento che gli stessi risultati possono essere ottenuti in diversi modi, anche con i tradizionali sistemi incorporati in (X)HTML.

table-layout

Questa proprietà imposta il metodo di layout di una tabella. Non è ereditata. Si applica solo alle tabelle.

```
<selettore> {table-layout: <valore>;}
```

Valori

- **auto**. Il layout della tabella viene definito automaticamente dal browser.
- **fixed**. Le regole di presentazione sono quelle impostate dall'autore nel CSS.

Nel caso del valore **auto** tutto è affidato al meccanismo di rendering dello user agent. Usando invece **fixed** possiamo innanzitutto definire la larghezza della tabella tramite la proprietà **width**. Volendo creare una tabella di 400px, quindi, scriveremo questa regola:

```
table {
table-layout: fixed;
width: 400px;
}
```

Capite bene che la stessa cosa si può ottenere in (X)HTML usando sempre l'attributo **width**:

```
<table width="400px">
```

Esempi

```
table.tabella1 {table-layout: fixed;}
table {table-layout: auto;}
```

border-collapse

Attraverso questa proprietà possiamo stabilire in che modo trattare i bordi e gli spazi tra le celle di una tabella. Si applica solo alle tabelle ed è ereditata.

```
<selettore> {border-collpse: <valore>;}
```

Valori

- **collapse**. Se viene impostato un bordo, le celle della tabella lo condividono.
- **separate**. Se viene impostato un bordo, ogni cella ha il suo, separato dalle altre. Lo spazio tra le celle e tra i bordi si imposta con la proprietà **border-spacing**.

Esempi

```
table {
border: 2px solid black;
border-collapse: separate;
border-spacing: 5px;
}
```

border-spacing

Imposta lo spazio tra le celle di una tabella. Va usata solo in presenza di un modello di border-collapse settato su **separate**. Proprietà ereditata e applicabile solo alle tabelle.

```
<selettore> {border-spacing: <valore>;}
```

Valori

- **un valore numerico con unità di misura**

empty-cells

Gestisce il trattamento delle celle di tabella senza contenuto. Agisce solo su quelle che non presentino al loro interno alcun tipo di markup, nemmeno il classico ** ** inserito in genere proprio per simulare la presenza di contenuto. Proprietà ereditata.

```
<selettore> {empty-cells: <valore>;}
```

Valori

- **show**. Mostra i bordi della cella.
- **hide**. I bordi non vengono mostrati e apparirà solo uno spazio vuoto.

caption-side

Le buone norme dell'accessibilità vogliono che una tabella sia sempre preceduta da una sorta di titolo/riassunto. In (X)HTML questa funzione è demandata al tag **<CAPTION>** di cui riportiamo sotto un esempio di sintassi:

```
<table>
<caption>Titolo della tabella</caption>
<tr>
<td>...
</table>
```

La proprietà **caption-side** definisce il lato su cui vogliamo far comparire tale titolo. E' ereditata. La supporta solo Mozilla.

```
<selettore> {caption-side: <valore>;}
```

Valori

- **top**. Caption sul lato superiore.
- **right**. Caption sul lato destro.
- **bottom**. Caption sul lato inferiore.
- **left**. Caption sul lato sinistro.

Esempi

```
table.tabella9 {
table-layout : fixed;
width : 300px;
background : Silver;
caption-side : bottom;
}
```

Altre proprietà

cursor

Definisce l'aspetto del cursore quando passa su un elemento. Ereditata. Si applica a tutti gli elementi.

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

Valori

I valori possibili sono molti ed è possibile praticamente usare tutti i tipi di cursore abituali dell'interfaccia utente che usate. Ecco la lista:

```
auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize
| se-resize | sw-resize | s-resize | w-resize | text | wait | help
```

E' anche possibile definire un cursore esterno, ovvero presente sul server e da scaricare. In questo caso va indicato l'URL della risorsa:

```
a {cursor: url(url_del_cursore);}
```

L'uso più tipico di questa proprietà è quello di modificare l'aspetto del cursore in corrispondenza dei link. Se volete cambiare il classico ditino con un altro cursore basta questa regola:

```
a:hover {cursor: <valore>;}
```

content

La proprietà **content** è usata per definire il contenuto da inserire nel documento con gli pseudo-elementi **:before** e **:after**.

counter increment e counter-reset

Proprietà non supportate da nessun browser. Servono a impostare lo stile e l'aspetto di un cosiddetto **contatore**, oggetti da creare con la proprietà **content** simili alle classiche liste (X)HTML.

direction

Specifica la direzione del testo da usare nella presentazione del documento. Utile se volete scrivere pagine in Ebraico.

outline

Il cosiddetto outline è un bordo che è possibile inserire attorno ad oggetti di un documento per evidenziarli. Non è supportato da nessun browser. Potenzialmente è possibile definire l'aspetto del bordo secondo le seguenti proprietà:

- **outline-color:** il colore
- **outline-style:** lo stile
- **otline-width:** lo spessore

I valori possibili sono identici a quelli visti per i bordi.

quotes

Serve a definire stile e aspetto dei segni di virgolettatura.

unicode-bidi

Influenza il layout del testo nelle pagine in cui siano presenti testi con diverse direzioni.