**Guida CSS**

**I CSS e i documenti (X)HTML**

1. 2. [Classificazione di elementi e tag XHTML](http://css.html.it/guide/lezione/15/classificazione-di-elementi-e-tag-xhtml/)

Capire gli elementi di una pagina Web: elementi blocco, in linea e rimpiazzati, la struttura ad albero

1. 3. [Inserire i fogli di stile in un documento](http://css.html.it/guide/lezione/16/inserire-i-fogli-di-stile-in-un-documento/)

Come inserire i CSS in un documento: i link e l'oggetto @import

1. 4. [L'attributo media](http://css.html.it/guide/lezione/17/lattributo-media/)

Differenziare i fogli di stile in base al dispositivo che li visualizzerà

1. 5. [Fogli di stile alternativi](http://css.html.it/guide/lezione/18/fogli-di-stile-alternativi/)

Lasciate decidere all'utente quale foglio di stile utilizzare per il documento

1. 6. [Compatibilità](http://css.html.it/guide/lezione/19/compatibilita/)

Un problema sempre aperto: la compatibilità tra i CSS e i browser

**Le regole e la sintassi per scrivere i CSS**

1. 7. [Come è fatto un CSS: regole e commenti](http://css.html.it/guide/lezione/20/come-e-fatto-un-css-regole-e-commenti/)

La struttura di un foglio di stile

1. 8. [I selettori](http://css.html.it/guide/lezione/21/i-selettori/)

Le fondamenta di un foglio di stile: i selettori

1. 9. [Id e classi: due selettori speciali](http://css.html.it/guide/lezione/22/id-e-classi-due-selettori-speciali/)

Come associare i fogli di stile alle parti di un documento

1. 10. [Le pseudo-classi](http://css.html.it/guide/lezione/23/le-pseudoclassi/)

Le classi speciali: come definire lo stato di un elemento

1. 11. [Gli Pseudo-elementi](http://css.html.it/guide/lezione/24/gli-pseudoelementi/)

Creare al volo elementi fittizi non presenti nel codice del documento

1. 12. [Le @-rules](http://css.html.it/guide/lezione/25/le-rules/)

Le funzioni e la sintassi di tutte le @-rules dei CSS

**Unità di misura, ereditarietà e box model**

1. 13. [Valori e unità di misura](http://css.html.it/guide/lezione/26/valori-e-unita-di-misura/)

Tutti i valori e le unità di misura dei CSS: dalle dimensioni alle unità di tempo

1. 14. [Ereditarietà, cascade, conflitti tra stili](http://css.html.it/guide/lezione/27/ereditarieta-cascade-conflitti-tra-stili/)

Come i CSS gestiscono la priorità e la precedenza delle regole. il concetto di Cascade

1. 15. [Il box model](http://css.html.it/guide/lezione/28/il-box-model/)

Come i CSS gestiscono la visualizzazione degli elementi nella pagina

**CSS all'opera: le proprietà di base**

1. 16. [Gestione del colore](http://css.html.it/guide/lezione/29/gestione-del-colore/)

Cambiare il colore agli elementi, ai bordi, allo sfondo di un elemento

1. 17. [Gestione dello sfondo](http://css.html.it/guide/lezione/30/gestione-dello-sfondo/)

Come gestire lo sfondo di un documento con in CSS

1. 18. [Gestione del testo: proprietà di base](http://css.html.it/guide/lezione/31/gestione-del-testo-proprieta-di-base/)

Le proprietà base del testo: impostare il tipo di carattere, la grandezza, la formattazione

1. 19. [Gestione del testo: proprietà avanzate](http://css.html.it/guide/lezione/32/gestione-del-testo-proprieta-avanzate/)

Le proprietà avanzate del testo: rendere più leggibile una pagina, inserire l'ombreggiatura

**Il Box Model: come disegnare la struttura della pagina**

1. 20. [Gestione delle dimensioni: altezza](http://css.html.it/guide/lezione/33/gestione-delle-dimensioni-altezza/)

Definire l'altezza di un elemento: sintassi e valori

1. 21. [Gestione delle dimensioni: larghezza](http://css.html.it/guide/lezione/34/gestione-delle-dimensioni-larghezza/)

Definire la larghezza di un elemento: sintassi e valori

1. 22. [I Margini](http://css.html.it/guide/lezione/35/i-margini/)

Come spaziare i vari elementi tra loro: le cinque proprietà dell'elemento margin

1. 23. [Il Padding](http://css.html.it/guide/lezione/36/il-padding/)

Creare spazi attorno agli elementi senza usare i margini

1. 24. [I Bordi](http://css.html.it/guide/lezione/37/i-bordi/)

Definire lo stile, la grandezza, il colore e tutte le proprietà dei bordi di un elemento

1. 25. [Le Liste](http://css.html.it/guide/lezione/38/le-liste/)

Gestire lo stile delle liste e associare un'immagine al segnaposto

**Posizionare gli elementi nella pagina**

1. 26. [Tre proprietà speciali: display, float clear](http://css.html.it/guide/lezione/39/tre-proprieta-speciali-display-float-clear/)

Nascondere un elemento sulla pagina o posizionarlo in dipendenza di altri

1. 27. [Posizionamento degli elementi](http://css.html.it/guide/lezione/40/posizionamento-degli-elementi/)

Posizionare gli elementi in qualsiasi punto della pagina web

1. 28. [Gestione della posizione](http://css.html.it/guide/lezione/41/gestione-della-posizione/)

Come collocare con precisione gli elementi in una pagina Web

**Altre proprietà**

1. 29. [Agire sulle tabelle](http://css.html.it/guide/lezione/42/agire-sulle-tabelle/)

Modifica dello stile delle tabelle: table-layout, border-collapse, border-spacing ecc.

1. 30. [Altre proprietà](http://css.html.it/guide/lezione/43/altre-proprieta/)

Le proprietà cursor, content, direction, outline, quotes, unicode-bidi

**Scegliere i CSS in base al supporto di visualizzazione**

1. 31. [CSS per la stampa: paged media styles](http://css.html.it/guide/lezione/44/css-per-la-stampa-paged-media-styles/)

La formattazione delle pagine destinate alla stampa con i CSS

1. 32. [CSS per documenti audio: aural media styles](http://css.html.it/guide/lezione/45/css-per-documenti-audio-aural-media-styles/)

Controllare l'audio sui browser o su dispositivi vocali

**Risorse utili**

1. 33. [Tabelle di compatibilità, test, validazione](http://css.html.it/guide/lezione/46/tabelle-di-compatibilita-test-validazione/)

Risorse per testare e verificare la corretta sintassi dei CSS

1. 34. [Gli Editor CSS](http://css.html.it/guide/lezione/47/gli-editor-css/)

I principali software per scrivere i CSS correttamente e con velocità

1. 35. [Risorse on line](http://css.html.it/guide/lezione/48/risorse-on-line/)

Webografia di siti utili dedicati ai CSS

1. 36. [Bibliografia](http://css.html.it/guide/lezione/49/bibliografia/)

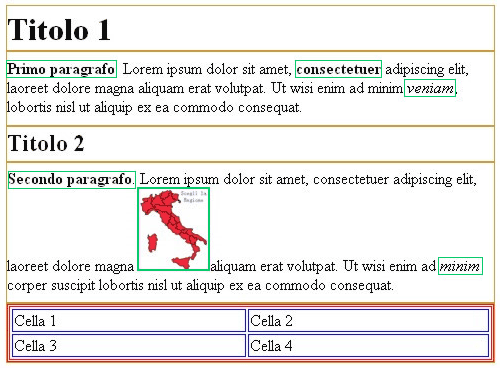
Libri in italiano e in inglese dedicati ai CSS

**Classificazione di elementi e tag XHTML**

La prima lezione di questa guida potrebbe spiazzare. 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 il classificare gli elementi.

**Elementi blocco e elementi in linea (inline-block)**

Osserviamo una pagina (X)HTML tentando di non pensare al contenuto ma solo alla sua struttura, mettendo in atto una sorta di processo di astrazione. Possiamo farci aiutare da una semplice immagine:



Possiamo iniziare pensando a una pagina (X)HTML, come a un insieme di rettangoli disposti sullo schermo. Non importa che si tratti di paragrafi, titoli, tabelle o immagini: si tratta sempre di box rettangolari.

Già dall'immagine si intuisce, però, che non tutti i box sono uguali. Alcuni contengono altri box al loro interno, altri sono invece contenuti all'interno dei primi. Alcuni, se si trovano (come accade) in mezzo al testo, lasciano che esso scorra loro intorno senza interrompere il suo flusso e senza andare a capo. Questa considerazione molto semplice già ci offre la rappresentazione della fondamentale distinzione tra elementi **blocco** ed elementi **inline**.

Gli **elementi blocco** sono 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. Possiamo verificarlo inserendo 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 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> (tag 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**

Altro concetto fondamentale ed utile da 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>

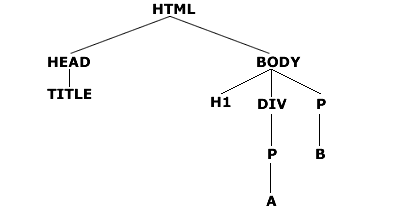
</div>

<p>Secondo<b>paragrafo</b></p>

</body>

</html>

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



Il documento è 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, 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.

Ad esempio <body> è figlio di <html>, ma è anche genitore di <h1>, <div> e <p>. Quest'ultimo è a sua volta genitore di un elemento <b>.

Si potrebbe concludere che anche <body> sia 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**).

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**). È importante spazzare il campo da un possibile fraintendimento: l'elemento radice di un documento (X)HTML non è <body>. Il fatto che <html> non sia una semplice dichiarazione ma sia trattato alla stregua di qualunque altro elemento lo si può testare aprendo [questa pagina](http://html.it/guide/esempi/css/test/html_body.html).

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

**Inserire i fogli di stile in un documento**

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

**CSS esterni e interni**

È **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:

|  |  |
| --- | --- |
| **Attributo** | **Descrizione** |
| **rel** | descrive il tipo di relazione tra il documento e il file collegato. È **obbligatorio**. Per i CSS due sono i valori possibili: **stylesheet** e **alternate stylesheet**. Approfondimenti nella lezione sui [Fogli di stile alternativi](http://css.html.it/guide/lezione/18/fogli-di-stile-alternativi/) |
| **href** | serve a definire l'URL assoluto o relativo del foglio di stile. È **obbligatorio** |
| **type** | identifica il tipo di dati da collegare. Per i CSS l'unico valore possibile è **text/css**. L'attributo è **obbligatorio** |
| **media** | con questo attributo si identifica il supporto (schermo, stampa, etc) cui applicare un particolare foglio di stile. Attributo **opzionale**. L'argomento sarà approfondito nella prossima lezione |

**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 questo stile prevarrà su quello del CSS esterno.

Se le pagine invece di una fossero 20 il discorso diventerebbe un po' 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'idea.

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**

Nella lezione precedente abbiamo accennato all'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" />

**Esempi**

Un'applicazione immediatamente fruibile, a patto di non usare browser di quarta generazione, è quella che prevede l'uso di un foglio di stile per la stampa. Le motivazioni per una simile soluzione possono essere molteplici. L'uso di caratteri o colori particolari, la presenza nella pagina di elementi che vogliamo non far comparire sulla carta, può portare alla necessità di adottare un CSS speciale. Un primo esempio lo potete valutare seguendo [questo link](http://html.it/guide/esempi/css/test/media.html). Provate a stamparlo. Il testo avrà caratteristiche diverse rispetto a quello che vedete sullo schermo: font, colore, dimensioni in punti e non in pixel. Una controprova: stampate [questo documento](http://html.it/guide/esempi/css/test/no_media.html), privo di fogli di stile speciali. Nessun cambiamento rispetto a quanto vedete sullo schermo.

Per un approfondimento trovate l'articolo Stampe perfette con i CSS nella sezione PRO di HTML.it.

Una risorsa eccellente sull'argomento: [Code Style media monitor](http://www.codestyle.org/css/media/). Articoli, linee guida, test sull'uso dei fogli di stile basati sui diversi media.

**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>**, come nell'[esempio allegato](http://html.it/guide/esempi/css/test/stili_alternativi.html). 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](http://www.webstandards.org/) 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 della 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.

Vi rimando all'articolo citato per una discussione ampia e ricca di esempi sulla questione. E vi segnalo [questa pagina](http://www.ericmeyeroncss.com/bonus/trick-hide.html) di Eric Meyer per una panoramica eccellente su altri metodi "interni" al linguaggio da affiancare al classico uso di @import.

**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.

[Suite di test di CSS2](http://www.meyerweb.com/eric/css/tests/css2/): di, manco a dirlo, Eric Meyer. È una pagina da cui potrete testare tutte le proprietà e i metodi di CSS2. L'approccio è semplice. Visitate la pagina con il browser che volete testare, scegliete la proprietà che vi interessa e verificate il supporto. Le pagine di test sono costruite in modo tale da rendere subito evidente eventuali comportamenti non-standard. L'opera monumentale è un'estenzione del lavoro fatto dal W3C e rimasto fermo a CSS1.

La seconda risorsa è una vera miniera. Si tratta dell'[archivio pubblico](http://archivist.incutio.com/css-discuss/) della migliore mailing-list sull'argomento: [css-discuss](http://two.pairlist.net/mailman/listinfo/css-discuss). Se non trovate una soluzione a un problema qui, vuol dire che non è stata ancora trovata.

**Come è fatto un CSS : regole e commenti**

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

Con questa lezione entriamo nel vivo dell'argomento. Iniziamo con l'analisi degli elementi costitutivi di un foglio di stile:

/\* Definisco le regole per il corpo del documento \*/

body {

background: White; /\* Prima dichiarazione: notate il punto e virgola finale!\*/

font-family: Verdana, Geneva, Arial, Helvetica, sans-serif; /\* Seconda dichiarazione\*/

font-size: 12px; /\* Terza dichiarazione \*/

}

/\* Titolo di primo livello \*/

h1 {

color: black;

}

/\* Paragrafo generico \*/

p {

color: black;

font: 12px/1.5 Georgia, "Times New Roman", serif;

}

#div1 {

background: Silver; /\* Sfondo colr argento \*/

width: 300px;

}

#div2 {

background: Silver;

width: 300px;

position: absolute;

top: 400px;

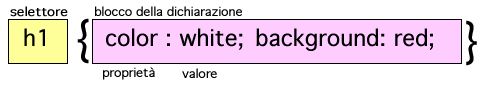
left: 400px;

}

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**

Figura 1. Struttura di 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.

**Sintassi**

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

*Esempio*

# Selettore di elementi

In questo CSS abbiamo usato solo selettori di elementi. Nel codice sono evidenziati in **grassetto**.

**Codice:**

**body** {  
background : White;  
font : 12px Verdana, Geneva, Arial, Helvetica, sans-serif;  
}  
  
**h1** {  
background: Silver;  
color: Black;   
}  
  
**p** {  
color : Red;  
}

## 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;}

Ulteriori esempi:

## Codice

**body** {  
background : White;  
font : 12px Verdana, Geneva, Arial, Helvetica, sans-serif;  
}  
  
**h1** {  
background : Silver;  
}  
  
**h1, p** {  
color : Black;  
}  
  
**p** {  
background : Red;  
}

# I titoli H1 hanno testo nero su sfondo grigio

I paragrafi hanno invece lo sfondo rosso, ma il testo è anche nero. Ecco un uso intelligente del raggruppamento.

## Selettore universale (universal selector)

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

**Sintassi**

\* { color: black; }

## Esempio

## Selettore universale

Semplice da capire. Ecco il codice.

**Codice:**

**body** {  
background : White;  
font : 12px Verdana, Geneva, Arial, Helvetica, sans-serif;  
}  
  
**\*** {  
color : Black;  
}

# Titolo H1: il testo è nero

Paragrafo: anche qui il testo è nero

## Selettore del discendente (descendant selector)

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

**Sintassi**

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

Alcune considerazioni importanti di cui tenere conto. Il selettore va letto per chiarezza **da destra a sinistra**. Nel primo esempio verranno selezionati tutti i paragrafi (<p>) discendenti di elementi <div>. Nel secondo tutti gli elementi <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. Utili chiarimenti sull'argomento nella lezione "Il corpo del documento" della Guida a XHTML.

Esempio:

# Descendant selector

Il concetto è semplice. La regola dichiarata con un descandant selector si applica solo agli elementi che sono discendenti di quello dichiarato nella regola alla sua sinistra.

##### Codice CSS:

**div p**{   
background : Silver;  
color : Black;  
}  
  
**p** {  
background : Silver;  
color : Red;  
}

Un paragrafo all'interno di un div: testo nero su sfondo grigio

Questo paragrafo non è in un div: testo rosso su sfondo grigio

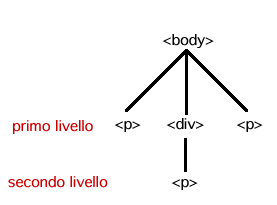
**Selettore del figlio (child selector)**

Seleziona un elemento che sia figlio diretto di un altro.

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

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

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



**Sintassi**

body > p {color: black;}

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

Questo selettore non è supportato da Explorer (almeno fino alla versione 6) su Windows, cosa che ne limita notevolmente l'utilizzo. Potete verificare il supporto negli altri browser in questo esempio:

**Selettore child**

Per capire il funzionamento di questo e di altri selettori (ma in genere per capire davvero i CSS) è necessario avere ben chiaro il concetto di struttura ad albero del documento. Concetti come ancestor, parent, descendant o child devono diventare il vostro pane quotidiano!

Questa è una parte del codice HTML di questa pagina. In grassetto abbiamo evidenziato i paragrafi che corrispondono alla definizione della regola in quanto figli diretti di un elemento **DIV**:

<table cellspacing="0" cellpadding="0" border="1">  
<tr>  
<td><p>Questo è paragrafo.......</p></td>  
</tr>  
</table>  
<div>  
**<p>Questo paragrafo è "figlio" diretto......</p>**  
**<p>Anche questo paragrafo dovrebbe essere rosso.....</p>**  
</div>

Notate le relazioni tra i vari elementi per capire l'esempio qui sotto. Questo è invece il codice del CSS:

div > p {  
color : Red;  
}  
  
p {  
color : Black;  
}

Come vedete, i paragrafi figli di un elemento div avranno il testo rosso. Tutti gli altri no.

|  |
| --- |
| Questo è paragrafo è "figlio" diretto di un elemento td: il suo testo è nero |

Questo paragrafo è "figlio" diretto dell'elemento div: il testo è rosso....ma se usate Explorer su Windows lo vedrete sempre nero. Perchè non supporta il selettore di tipo child. Provate con altri browser.

Anche questo paragrafo dovrebbe essere rosso: è sempre "figlio" di un div

## 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. Anche per questo tipo di esempio con ulteriori commenti.

## Adjacent-sibling selector

Verifichiamo il funzionamento e il supporto di questo tipo di selettore.

**Codice:**

body {  
background : White;  
font : 12px Verdana, Geneva, Arial, Helvetica, sans-serif;  
}  
  
**h1 + p** {  
color : Red;  
}  
  
p {  
color : Black;  
}

# Titolo H1

Il testo di questo paragrafo dovrebbe essere rosso (provate con Mozilla, Netscape 6/7 oppure con Opera o ancora con Explorer 5 Mac...

Il testo di questo di questo paragrafo è invece nero: non è infatti un paragrafo adiacente all'elemento h1

## Selettore dell'attributo (attribute selector)

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

**Sintassi**

Quattro sono i modi d'uso possibili.

**Attributo semplice**

elemento [ attributo ]

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

La dichiarazione:

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 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 questo esempio abbiamo assegnato al paragrafo un attributo **class="testorosso"**:

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

A questo paragrafo ho assegnato l'attributo **class** senza però definire un CSS. Risultato: il testo viene mostrato secondo le impostazione predefinite del browser.

Come vedete non succede nulla. Il problema è che il valore dell'attributo **class** deve trovare una corrispondenza in un foglio di stile. In questo secondo esempio, 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>

Ora ho definito una classe **.testorosso** in un CSS incorporato. Il risultato è evidente. L'attributo **class** ha trovato una corrispondenza in un CSS e il testo viene trasformato secondo le regole definite.

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.

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

## Classe

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

**.**nome\_della\_classe

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

Esiste un secondo tipo di sintassi:

<elemento>**.**nome\_della\_classe

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

p.testorosso {color: red;}

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

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

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

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

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

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

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

Esempi di questa e delle altre modalità in questo esempio:

# Uso delle classi

L'uso delle classi è davvero una delle basi dei CSS. Sono uno strumento estremamente potente e flessibile. Gli esempi mostrano i vari modi con cui è possibile dichiararle in un foglio di stile.

## Primo esempio

Iniziamo da un semplice paragrafo senza classe.

**Codice:**

p {  
color : Black;  
}

Questo è un paragrafo generico: il testo è nero

## Secondo esempio

Qui abbiniamo una classe ad un elemento.

**Codice:**

p.classe1 {  
color : Red;  
}

Significa che solo i pargrafi con la classe "**classe1**" saranno rossi: questo no perchè non presenta l'attributo.

Questo paragrafo ha come classe ".classe1". Il testo è rosso.

### Questo è un tag <h3> al quale abbiamo assegnato la classe "classe1". Il testo sarà sempre nero, perchè la regola fa riferimento solo ai paragrafi con attributo class="classe1".

## Terzo esempio

Dichiariamo una classe generica.

**Codice:**

.classe2 {  
color: Blue;  
}

# A questo titolo abbiamo applicato ".classe2"

### Tutti gli elementi con ".classe2" hanno il testo blue.

## Quarto esempio

Applicazione di classi multiple.

**Codice:**

p.classe3.classe4 {  
color: green;  
}

Questo paragrafo dovrebbe essere verde perchè ha come valori di class "classe3 classe4".

## 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.

Esempio anche per l'**ID**:

L'ID è uno dei più importanti selettori e anche tra i più semplici da capire. Si dichiara un id preceduto da un cancelletto nel css e si richiama nel documento HTML tramite l'attributo id. Che, ricordiamolo, dovrebbe essere univoco.

## Codice

#par1 {  
color : Red;  
}

Paragrafo 1: il testo è rosso. L'id del paragrafo è infatti "par1"

Questo paragrafo non ha nessun attributo id: il colore del testo è quello ereditato da body.

**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 (probabilmente sarà supportato anche nelle versioni di Explorer dalla 7 in poi, che dovrebbero essere più aderenti agli standard). Seleziona e formatta un elemento che si trovi ad essere il primo elemento figlio di un altro elemento.

**Sintassi**

<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><!--sarà rosso-->  
<p>blah blah blah</p>  
<table>  
  <tr>  
    <td>  
      <p>blah blah blah</p><!--sarà rosso-->  
      <p>blah blah blah</p>  
    </td>  
  </tr>  
</table>

si capisce che solo i paragrafi indicati saranno rossi, perchè sono primi figli, rispettivamente, di un elemento <div> e di un altro elemento <td>.

Per verificare il supporto provate questo esempio che contiene commenti ed esempi anche sulle altre pseudo-classi.

**Pseudo-classi**

Esempi sull'uso delle pseudo classi.

**:first-child**

La proprietà è supportata dai browser Gecko-based. In Explorer e Opera vedrete il primo paragrafo nero.

**Codice HTML:**

<div id="div1">  
<p>Primo paragrafo.....</p>  
<p>Secondo pargrafo: nero</p>  
<p>Terzo paragrafo: nero</p>  
</div>

**Codice CSS:**

#div1 {  
color: Black;  
font : 12px Verdana, Geneva, Arial, Helvetica, sans-serif;  
}  
  
#div1 p:first-child {  
color: Red;  
font-weight: bold;  
}

**Primo paragrafo: rosso e grassetto perchè primo elemento figlio di #div1.**

Secondo paragrafo: nero

Terzo paragrafo: nero

**:link**

Usare la pseudo-classe :link. Il [link](http://sitoinesistente.it) non visitato è rosso e non sottolineato.

**Codice:**

#div2 a:link {  
color: Red;  
text-decoration: none;  
}

**:visited**

Usare la pseudo classe :visited. Il [link](http://www.html.it) visitato è verde e non sottolineato.

**Codice:**

#div2 a:visited {  
color: Green;  
text-decoration: none;  
}

**:hover**

Usare la pseudoclasse :hover. Al passaggio del mouse il [link](javascript:;) diventa blue e sottolineato.

**Codice:**

#div2 a:hover {  
color: Blue;  
text-decoration: underline;  
}

**:active**

Usare la pseudoclasse :active. Mentre il tasto sinistro è premuto il [link](javascript:;) è viola e sottolineato.

**Codice:**

#div2 a:active {  
color: Purple;  
text-decoration: underline;  
}

**:focus**

Su Explorer Win non noterete nulla, ma quando il campo di testo riceve il focus, il colore di sfondo diventa giallo.

Codice:

#div3 input:focus {  
background-color: Yellow;  
}

Inizio modulo



**:lang**

Ora inseriamo del testo in inglese. Dovrebbe apparire in grassetto.

**Codice CSS:**

p:lang(en) {  
font-weight: bold;  
}

Codice HTML:

<p lang="en">This paragraph is in English!</p>

This paragraph is in English!

Fine modulo

## :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. E occhio alla ormai nota tabella di Eric Meyer.

## :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.

**Sintassi**

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.

**Sintassi**

Valgono le stesse regole generali viste per :first-letter.

**Esempi**

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...";}

:before non è supportato da Explorer per Windows (almeno fino alla versione 6).

## :after

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

**Sintassi**

h1:after {content: url(immagine.gif);}  
h1#nome\_id {content: "Testo del titolo";}

Esempi e codice anche in [questa pagina](http://html.it/guide/esempi/css/test/pseudoelementi.html).

**Pseudo-elementi**

Gli esempi mostrano l'uso degli pseudo-elementi. Testate con vari browser per verificare il supporto.

**:first-letter**

La prima lettera del paragrafo sarà di un bel rosso vivo.

**Codice:**

#par1:first-letter {  
color: Red;  
font : bold 22px Verdana, Geneva, Arial, Helvetica, sans-serif;  
}

**:first-line**

La prima linea di questo paragrafo sarà rossa e la dimensione del font di 22 pixel. Blah Blah Blah

Blah Blah Blah Blah Blah Blah Blah Blah Blah Blah Blah Blah Blah Blah BlahBlah Blah Blah BlahBlah Blah Blah Blah Blah Blah Blah Blah Blah Blah Blah Blah Blah Blah

**Codice:**

#par2:first-line {  
color: Red;  
font : bold 22px Verdana, Geneva, Arial, Helvetica, sans-serif;  
}

**:before**

Una stringa prima del paragrafo.

**Codice:**

#par3:before {  
content: "Il testo di questo paragrafo è..";  
}

**:after**

Virgolette alla fine del paragrafo.

**Codice:**

#par4:after {  
content: close-quote;  
}

**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. Esaminiamoli.

## @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**). Si veda la [pagina di esempio](http://html.it/guide/esempi/css/test/usare_@import.html):

# Usare @import

A questa pagina abbiamo collegato il foglio di stile "**stile1.css**" tramite il tag **<LINK>**:

<link href="stile1.css" rel="stylesheet" type="text/css" />

Questo il codice del CSS:

@import url(stile2.css);  
body {  
background : White;  
color : Black;  
font: 12px Verdana, Arial, sans-serif;  
}

Come vedete non comprende nessuna regola relativa all'elemento **<h1>**. Lo stile per quest'ultimo è definito all'interno del foglio "**stile2.css**" incorporato nel primo con la direttiva **@import**. Questo il codice:

h1 {  
color: green;  
}

Se la frase "Usare @import" è verde vuol dire che tutto funziona.

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;  
}  
}

I risultati li potete valutare in questo esempio:

# Titolo

La parola titolo è un elemento <h1>. Sullo schermo la vedrete rossa, sulla carta sarà nera.

Codice del CSS:

body {  
background: white;  
color: black;  
font: 12px Verdana, Arial, sans-serif;  
}  
@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 webviste 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. [L'esempio](http://html.it/guide/esempi/css/test/ereditarieta.html) è al riguardo chiarificatore. 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!

Fin qui tutto semplice. Ora scendiamo nei dettagli. L'ereditarietà non basta a spiegare le molteplici possibilità di relazione tra le regole di un CSS.

**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? Provare per credere: [esempio](http://html.it/guide/esempi/css/test/conflitti.html).

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;}

Subito la prova. Se osservate il terzo esempio di [questa pagina](http://html.it/guide/esempi/css/test/specificita.html) vedrete che il paragrafo sarà verde e non rosso.

**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 voltaprevalgono su quelli **collegati**.

Esempi e commenti in [questa pagina](http://html.it/guide/esempi/css/test/cascade.html).

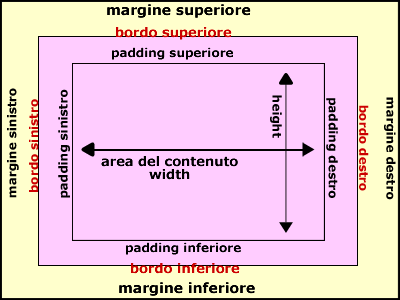
**Importanza**

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. Ancora [un esempio](http://html.it/guide/esempi/css/test/importance.html) per verificare il concetto.

**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**. È 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**. È 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**. È una linea di dimensione, stile e colore variabile che circonda la zona del padding e l'area del contenuto.
* **il margine**. È 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:

Larghezza del box

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. [L'esempio](http://html.it/guide/esempi/css/test/box_model.html#es1) chiarisce anche visivamente questo aspetto (attenzione: se visualizzate questa pagina con Explorer 5/5.5 su Windows non vedrete la cosa giusta. Questo browser interpreta male il concetto di larghezza. Per chiarimenti e soluzioni si veda la lezione dedicata alla proprietà width).

**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. [Esempio](http://html.it/guide/esempi/css/test/box_model.html#es2).

**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. È il meccansimo del cosiddetto **margin collapsing**. Tale meccanismo non si applica ai box adiacenti in senso orizzontale. [Esempio](http://html.it/guide/esempi/css/test/box_model.html#es4).

Ulteriori approfondimenti su questi argomentii li troverete nelle prossime lezioni, quando affronteremo l'analisi delle singole proprietà. Qui aggiungiamo che regole particolari su alcuni degli aspetti trattati si applicano agli elementi posizionati. Ne parleremo in una lezione specifica.

**Links**

[Il box-model secondo il W3C](http://www.w3.org/tr/REC-CSS2/box.html): complesso e un pò intricato.

[Il box-model, teoria e pratica](http://www.webreference.com/html/tutorial9/): ottimo tutorial sul sito WebReference.com. Lettura vivamente consigliata.

**Gestione del colore**

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

**Definizione del colore**

**Parole chiave**

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

Figura 1 - La palette VGA di Windows

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

L'ordine è quello dei colori visualizzati nell'immagine.

**#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** (vedi [l'esempio allegato](http://html.it/guide/esempi/css/test/color.html#p6)).

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; }

Ulteriori esempi in [questa pagina](http://html.it/guide/esempi/css/test/color.html).

**Links**

[CSS color names](http://www.w3schools.com/css/css_colornames.asp): ottima risorsa dal sito W3Schools. Tabella Websafe con valori esadecimali e nomi dei colori (ricordiamo però che nei CSS gli unici nomi accettati sono i 16 visti nella lezione). L'utilità più grande la notate se cliccate sul codice esadecimale del colore. Si aprirà una pagina in cui quel colore viene usato come sfondo e tutti gli altri vengono usati per il testo. Eccellente quando si deve valutare l'impatto visivo della coppia colore testo-sfondo.

**Gestione dello sfondo**

In questa lezione esaminiamo le proprietà relative alla gestione dello sfondo. Sin dalle origini, possiamo inserire un'immagine o un colore di sfondo direttamente nel markup HTML. 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 in HTML**: 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 sempre favorito la scelta di piccole texture in grado di dare la sensazione visiva dell'uniformità.

Grazie ai CSS questi limiti vengono spazzati via e le possibilità creative, compatibilità permettendo, sono davvero infinite.

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

* background-color
* background-image
* background-repeat
* background-attachment
* background-position

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

**Sintassi**

selettore {background-color: *valore*;}

**Valori**

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

Usando **transparent** come valore un elemento avrà come colore quello dell'elemento parente. Nell'[esempio](http://html.it/guide/esempi/css/test/back_color.html): 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.

**Sintassi**

selettore { background-image: url(*valore*); }

**Valori**

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

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

**Esempi**

body {background-image: url(sfondo.gif); }

div 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 (vedi [esempio](http://html.it/guide/esempi/css/test/back_image.html)).

Un 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.

**Sintassi**

selettore {background-repeat: *valore*;}

**Valori**

* **repeat**. L'immagine viene ripetuta in orizzontale e verticale. È 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.

[L'esempio](http://html.it/guide/esempi/css/test/back_repeat.html) riporta i quattro diversi comportamenti con un'immagine di sfondo applicata ad una tabella.

**Esempi**

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

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

**background-attachment**

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

**Sintassi**

selettore {background-attachment: *valore*;}

**Valori**

* **scroll**. L'immagine scorre con il resto del documento quando si fa lo scrolling della pagina ([esempio](http://html.it/guide/esempi/css/test/back_attachmentscroll.html))
* **fixed**. L'immagine rimane fissa mentre il documento scorre ([esempio](http://html.it/guide/esempi/css/test/back_attachmentfix.html))

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

**Esempi**

body { background-image: url(back\_400.gif);

background-repeat: repeat-x;

background-attachment: fixed; }

**background-position**

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

**Sintassi**

selettore {background-position: *valoreOriz | valoreVert*;}

**Valori**

I valori specificano le coordinate di un punto sull'asse verticale e su quello orizzontale e possono essere espressi con diverse unità di misura e modalità:

* 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. Per [questo documento](http://html.it/guide/esempi/css/test/back_position.html) di esempio abbiamo usato queste impostazioni:

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.

Invece dei pixel potevamo utilizzare altre unità di misura o le percentuali. La scelta delle percentuali ci assicura una maggiore affidabilità a risoluzioni diverse.

In alternativa avremmo potuto **utilizzare le parole chiave**, come in questo esempio:

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 ([esempio](http://html.it/guide/esempi/css/test/back_position2.html)).

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**

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

**Sintassi**

selettore { background: *background-color*

*background-image*

*background-repeat*

*background-attachment*

*background-position*;

}

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

**Esempi**

In [questa pagina](http://html.it/guide/esempi/css/test/background.html) di esempio lo sfondo della pagina è stato così settato con l'uso della proprietà **background**:

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

**Link utili**

[Usare il background](http://css.html.it/articoli/leggi/432/usare-il-background/) - Introduce ad un uso avanzato degli sfondi nel layout e all'utilizzo di effetti come gradienti o pattern.

[Creare sprite CSS](http://css.html.it/articoli/leggi/3193/creare-sprite-css/) - Sfruttare la proprietà background per creare bottoni e sfondi animati.

[Sfondi a tutta pagina con i CSS](http://css.html.it/articoli/leggi/3188/sfondi-a-tutta-pagina-con-i-css/) - Superare i limiti del classico background e creare sfondi che occupano l'intera area del viewport del browser.

[Lo sfondo con i CSS](http://xhtml.html.it/guide/lezione/1667/approfondimenti-lo-sfondo-con-i-css/) - Introduzione all'uso dei CSS nella Guida HTML.

**Gestione del testo**

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.

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

* il font da usare
* la sua dimensione
* la sua consistenza
* l'interlinea tra i paragrafi
* l'allineamento del testo
* la sua decorazione (sottolineature e simili)

La lezione è corredata di un [documento di esempio](http://html.it/guide/esempi/css/test/testo_base.html) che mostra l'uso delle proprietà esaminate e corredato di codice.

**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.

**Sintassi**

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

**Esempi**

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

**font-size**

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

**Sintassi**

selettore { font-size: <valore>; }

**Valori**

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

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

Sono valori assoluti:

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

Sono valori relativi:

* le parole chiave **smaller** e **larger**
* quelli espressi in **em** (em-height)
* quelli espressi in **percentuale**

**Esempi**

p {font-size: 12px;}  
div.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.

**Sintassi**

selettore {font-weight: <valore>;}

**Valori**

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

* **valori numerici** 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 ordinati in senso crescente (dal leggero al pesante)
* **normal**. Valore di default. E' l'aspetto normale del font ed equivale al valore 400
* **bold**. Il carattere acquista l'aspetto che definiamo in genere grassetto. Equivale a 700
* **bolder.** Misura relativa. Serve a specificare che una determinata porzione di testo dovrà apparire più pesante rispetto al testo dell'elemento parente
* **lighter**. Misura relativa. Il testo sarà più leggero di quello dell'elemento parente

**Esempi**

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

**font-style**

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

* **normal**: il testo mantiene il suo aspetto normale
* **italic**: formatta il testo in corsivo
* **oblique**: praticamente simile a italic

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

**Sintassi**

selettore {font-style: <valore>;}

**Esempi**

p {font-style: italic;}

**Line-height**

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

**Sintassi**

selettore {line-height: <valore>;}

**Valori**

* **normal.** Il browser separerà le righe con uno spazio ritenuto "ragionevole". Dovrebbe corrispondere a un valore numerico compreso tra 1 e 1.2
* **valore numerico**. Usando valori numerici tipo 1.2, 1.3, 1.5 si ottiene questo risultato: l'altezza della riga sarà uguale alla dimensione del font moltiplicata per questo valore.
* **un valore numerico con unità di misura**. L'altezza della riga sarà uguale alla dimensione specificata.
* **percentuale**. L'altezza della riga viene calcolata come una percentuale della dimensione del font.

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

**Esempi**

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

**font**

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

* font-family
* font-size
* line-height
* font-weight
* font.-style
* font-variant
* font di sistema

**Sintassi**

selettore {font: <valori>;}

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

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

font-size/line-height

**Esempi**

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

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

**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;}

**Allinerare il testo: Text-align**

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

**Sintassi**

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.

**Sintassi**

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;}

**Links**

Negli articoli che seguono troverete risposte a importanti questioni legate alla gestione del testo con i CSS e che esulano dall'impianto di una guida come questa. La lettura è pertanto fortemente consigliata:

L'arte del testo perfetto: come utilizzare i CSS per formattare il testo di una pagina web

Stabilire le dimensioni dei font in un CSS: vantaggi, problemi e soluzioni

I font e la tipografia del testo