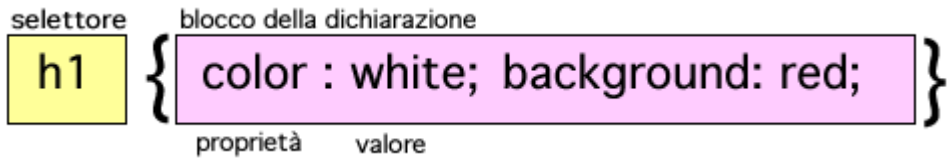
Guida ai fogli di stile CSS

Materiale tratto da: http://css.html.it/guide/leggi/2/guida-css-di-base/

Come è fatto un CSS: regole e commenti

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;}
```

Riguardo questa breve guida sull'analisi delle proprietà usi e costrutti sintattici di ciascuna, ci limitiamo all'elenco:

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

I selettori

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

Selettore di elementi (type selector)

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

Sintassi

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

Raggruppare

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

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

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

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

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

Selettore universale (universal selector)

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

Sintassi

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

Id e classi: due selettori speciali

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

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

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

Non succede nulla!! Il problema è che il valore dell'attributo **class** deve trovare una corrispondenza in un foglio di stile. Definendo un CSS incorporato creando un selettore di tipo **classe** e assegnado ad esso il nome **testorosso**:

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

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

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

In un documento potrò avere senza problemi questa situazione:

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

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

Ma non questa:

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

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

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

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

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

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:

..
...

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

...

ID

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

#nome_id

Con questa regola:

#titolo {color: blue;}

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

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

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

p#nome_id

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

Le pseudo-classi

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

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

a:link {color: blue;}

La regola vuol dire: i collegamenti ipertestuali (`<a>`) che non siano stati visitati (`:link`) avranno il colore `blue`. Da qui risulta più chiaro il concetto espresso all'inizio: la pseudo-classe `:link` definisce lo stile (colore `blue`) solo in una determinata situazione, 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.

: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.

Valori e unità di misura

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

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

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

Unità per le dimensioni

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

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

Percentuale

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

Colori

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

Stringhe

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

Valori URI

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

Unità per gli angoli

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

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

Unità di tempo

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

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

Unità di frequenza

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

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

Ereditarietà, cascade, 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.

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?

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

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

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

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

Specificità

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

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

#titolo {color: black;}

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

.classe1 {background: #C00;}

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

h1 {color: red;}

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

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

#paragrafo {color: green;}

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

div p {color: red;}

Il concetto di cascade

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

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

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

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

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

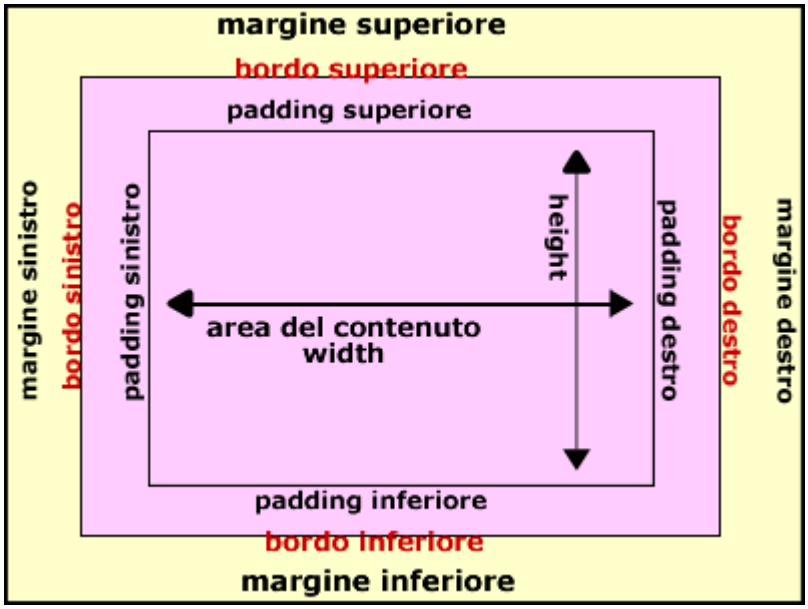
Importanza

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

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.

2. Larghezza ed elemento contenitore

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

3. Uso del valore auto

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

Solo **i margini possono avere valori negativi**. Ciò non è consentito per padding, bordi, altezza e larghezza.

4. Margini verticali e orizzontali tra gli elementi

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

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.