Programmare per il web

LATO CLIENT

- HTML4
- HTML5
- css2
- css3

Javascript -
JQuery -
Ajax -
VBscript -
Adobe Flash -
Oracle javaFX -
Microsoft Silverlight -

alessandro stella

PROGRAMMARE PER IL WEB

lato client

Alessandro Stella
(http://www.alessandrostella.it/)

Sinossi

Il libro illustra come sia possibile realizzare una completa applicazione web scrivendo codice valido su qualsiasi browser. Scrivere codice per il browser comporta la conoscenza di molteplici linguaggi, ognuno con determinate caratteristiche. Il libro fornisce una visione di insieme di tali linguaggi soffermandosi in modo dettagliato su quelli che sono assolutamente necessari: HTML, CSS e JavaScript.

La prima parte del libro è dedicata completamente alle tecnologie necessarie per produrre pagine web fruibili da chiunque e su qualunque dispositivo: HTML e CSS. Le due tecnologie vengono trattate in modo specifico sia nelle versioni attuali (HTML 4 e CSS 2) sia nelle versioni in fase di definizione (HTML 5 e CSS 3) con numerosi esempi e spiegazioni. Gli esempi sono tutti visionabili e scaricabili online.

Nella parte centrale vengono prese in considerazione tutte le tecnologie alternative o suppletive a HTML e CSS: flash, javafx e silverlight. Questa parte del libro mostra un approccio di tipo puramente teorico senza entrare nello specifico pratico delle singole tecnologie. L'obiettivo è quello di mettere il lettore a conoscenza dell'esistenza di tali tecnologie e delle motivazioni che ne hanno spinto la nascita e la diffusione.

L'ultima parte del libro si concentra sui linguaggi di scripting, con particolare attenzione riservata a javascript, alle sue tante librerie e ai suoi diversi utilizzi come jquery e ajax.

Insomma, un cammino totale nel mondo della programmazione web lato client!

Ringraziamenti

Un doveroso e sentito ringraziamento a:

- **Claudio De Sio Cesari**
  per l'impegno e il tempo che mi ha dedicato.

- **Barbara Panico**
  per l'attenta opera di lettura e correzione degli errori semantici e sintattici.

Indice generale

Introduzione

La programmazione web si basa su un tipo di **architettura** chiamata **client-server** in cui un'applicazione client richiede dati ad un'applicazione server che glieli fornisce. Programmare per il web significa quindi produrre almeno due applicazioni: una client e una server. Inoltre senza un'applicazione client che fa delle richieste, l'applicazione server sarebbe inutile perché non avrebbe nessuna richiesta da esaudire.
Questo libro ha come compito quello di insegnarci a produrre un'applicazione client. Tuttavia per scrivere un'applicazione client si può procedere in diversi modi.
Potremmo, ad esempio, scrivere codice per windows, producendo quindi un file eseguibile (.exe) che funga da client e che sia in grado di inviare le richieste e di ricevere le risposte da un'applicazione server, ma questo tipo di soluzione funzionerebbe solo su computer equipaggiato con windows perché gli eseguibili funzionano solo su PC che usano windows come sistema operativo.
Oppure potremmo seguire un'altra strada e scrivere l'applicazione client usando, ad esempio, codice java. In questo modo la stessa applicazione client potrebbe essere usata su qualunque sistema operativo (o quasi).
Ma c'è anche un altro possibile modo di procedere: **scrivere codice per il browser**.
Il browser è un'applicazione client (è infatti in grado di emettere richieste e interpretare le risposte), è installato o installabile su praticamente ogni dispositivo e funziona su ogni sistema operativo. Non lo dobbiamo scrivere noi, è stabile, è sicuro, è controllato, è aggiornato di continuo. Scrivere codice per il browser significa quindi scrivere codice che funzionerà su tutti i sistemi operativi e su tutti i dispositivi (PC, tablet, smartphone e così via). Usare il browser come applicazione client garantisce una compatibilità pressoché totale ed è quello che impareremo a fare in questo libro.

A tale scopo diventa importante conoscere un po' più da vicino il browser e, soprattutto, i linguaggi da esso gestiti. In tal senso approfondiremo i due linguaggi per eccellenza da imparare per scrivere codice per il browser: **HTML** e **CSS**. Dedicheremo un intero capitolo a fare pratica con questi due linguaggi. Inoltre, poiché le nuove versioni di HTML e CSS stanno ormai entrando sempre più prepotentemente nell'uso comune, dedicheremo molto spazio allo studio di HTML5 e CSS3 con relative applicazioni pratiche. Il tutto rimanendo sempre fedeli alle direttive del W3C, organismo che impareremo a conoscere e rispettare.
Al fine di dare la più ampia visione possibile sulla programmazione lato client per il Browser, faremo la conoscenza di altre tecnologie molto diffuse: Adobe Flash, Microsoft Silverlight e Oracle JavaFX. Tecnologie che consentono di ottenere effetti grafici spettacolari all'interno di una pagina web e che sono molto apprezzate in particolari settori di sviluppo. Non potremo entrare nei dettagli d'uso di queste tecnologie, ma è opportuno sapere cosa sono e cosa fanno.
Infine prenderemo in esame i linguaggi di scripting: VBScript e Javascript. Veri e propri linguaggi di programmazione. VBScript sarà trattato in maniera molto elementare, ma sufficiente per conoscere i meccanismi del linguaggio. Javascript invece sarà trattato ad un livello intermedio, procedendo per gradi. Non solo. Dopo aver preso confidenza con Javascript ne vedremo due fondamentali applicazioni: Ajax e JQuery.
Insomma un viaggio lungo ed entusiasmante che ci condurrà a scrivere codice per il Browser e ad avere una visione di insieme di tutto quello che è importante sapere sul fronte client!

Organizzazione dell'opera

Il libro è diviso in 13 capitoli.
Il primo capitolo è di carattere introduttivo e fornisce cognizioni di base necessarie per comprendere sia la programmazione web nel suo complesso sia la parte della programmazione web che verrà affrontata nel libro.
Il capitolo 2 introduce HTML e CSS spiegando i motivi per cui, per produrre una pagina web professionale, necessitano entrambi.
I capitoli 3 - 5 trattano HTML e CSS nelle versioni attualmente più usate, cioè HTML 4 e CSS 2. Il capitolo 5 è interamente dedicato alla pratica.
I capitoli 6 - 8 trattano HTML e CSS nelle versioni future, cioè HTML 5 e CSS 3. Anche in questo caso è previsto un capitolo intero, il capitolo 8, interamente dedicato alla pratica.
Il capitolo 9 è un rapido viaggio tra le tecnologie nate per produrre siti web graficamente accattivanti con un ridottissimo uso di HTML: flash, javaFX e silverlight.

I capitoli 10 - 13 trattano sostanzialmente di javascript, jQuery e ajax, con una piccola digressione (nel capitolo 10) su VBScript.

A chi si rivolge il libro

Il libro è scritto per tutti coloro che vogliono avvicinarsi al mondo della programmazione web lato client usando il browser come applicazione client di riferimento. E' scritto altresì per tutti coloro che vogliono avere una visione d'insieme sulla programmazione web, ossia conoscere tutte le maggiori tecnologie utilizzate nella produzione di codice per il browser.
Leggendo questo libro non diventeremo dei guru della programmazione web lato client, ma conosceremo tutte le strade da seguire per diventarlo.
Non è un libro per esperti del settore.

Prerequisiti

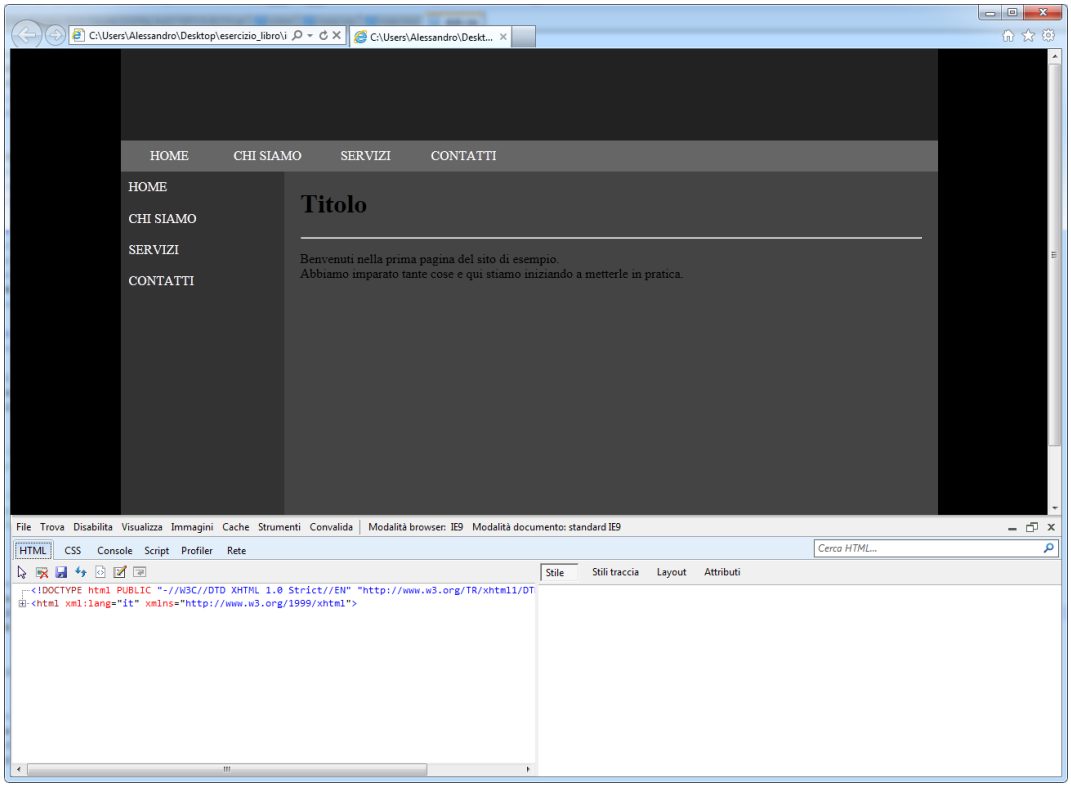
Per leggere e comprendere il libro non è necessaria alcuna conoscenza di HTML né di CSS. Tuttavia, nello studio della terza parte del libro, in cui vengono trattati i linguaggi di scripting, è necessaria una minima conoscenza dei principi della programmazione: sapere cosa è una variabile, cosa è un ciclo, cosa è una funzione sono nozioni importanti per poter studiare e usare javascript in modo proficuo.

Strumenti utilizzati

Il codice presente in questo libro è stato scritto su piattaforma windows usando un editor di testo chiamato Notepad++ (http://notepad-plus-plus.org/), rilasciato con licenza GPL.
In alternativa possiamo usare un editor online di pubblico utilizzo (http://jsfiddle.net/).
Per gli utenti di linux e mac esistono validissimi editor tra cui qui segnaliamo Komodo Edit (http://www.activestate.com/komodo-edit/downloads).
Tutte le alternative presentate ci consentono di perseguire l'obiettivo di questo libro, ossia imparare tag e codice di programmazione senza troppi aiuti da parte di sistemi automatici che ci lasciano ignoranti. Noi vogliamo sapere.
Tutto il codice del libro è disponibile online per libero download.

Quasi tutti i moderni browser integrano al loro interno risorse per gli sviluppatori, in modo che sia possibile effettuare correzioni e controlli al codice HTML e CSS direttamente dal browser. Per accedere a tali risorse basta **premere F12** in una qualsiasi finestra del browser.

Noi qui per semplicità useremo Internet Explorer 9, ma è possibile eseguire gli stessi controlli con tutti gli altri browser (Chrome, Firefox, Safari, ecc) sempre premendo F12 (nel caso di Firefox è opportuno installare Firebug).
Premendo F12 in una qualunque pagina web ci dovremmo trovare di fronte a qualcosa del genere.



Potrebbe accadere che la console di controllo che appare premendo F12 (la parte bianca in basso) venga mostrata non in basso alla pagina, ma in un nuova finestra. In tal caso ci basta premere CTRL+P per ritornare nella situazione mostrata nell'immagine.
In basso a sinistra abbiamo tutto il codice HTML. Premendo sul + possiamo infatti espandere il tag <html> e vedere i tag in esso contenuti e così via. Se selezioniamo un tag, sulla destra compariranno le proprietà CSS associate a tale tag. Non solo. Possiamo anche verificare in tempo reale eventuali modifiche apportate alle proprietà del foglio di stile.
La cosa utile è che tutte le modifiche che apportiamo tramite questa tecnica non modificano il foglio di stile. Ci basterà aggiornare la pagina (F5) per cancellare tutte le modifiche apportate.

Questo strumento è didatticamente molto utile perché mostra direttamente sul browser quali effetti avrebbero le nostre eventuali modifiche sul layout della pagina. Inoltre ci costringe a riflettere sul perché accade ciò che vediamo.

Oltre a tale strumento di controllo è importante sapere che il browser è in grado di mostrare il codice HTML della pagina visualizzata. Basta cliccare su un punto vuoto della pagina con il tasto destro del mouse e scegliere "HTML" oppure "Visualizza sorgente pagina".

Infine è importante avere un riscontro ufficiale di quello che scriviamo. Il nostro codice può essere più o meno valido secondo gli standard w3c! Per sapere se il nostro codice HTML è ben scritto, possiamo (dobbiamo) usare un validatore messo a disposizione dal w3c:
http://validator.w3.org/

Allo stesso modo è opportuno controllare il codice CSS. Anche in questo caso esiste un apposito validatore w3c:
http://jigsaw.w3.org/css-validator/

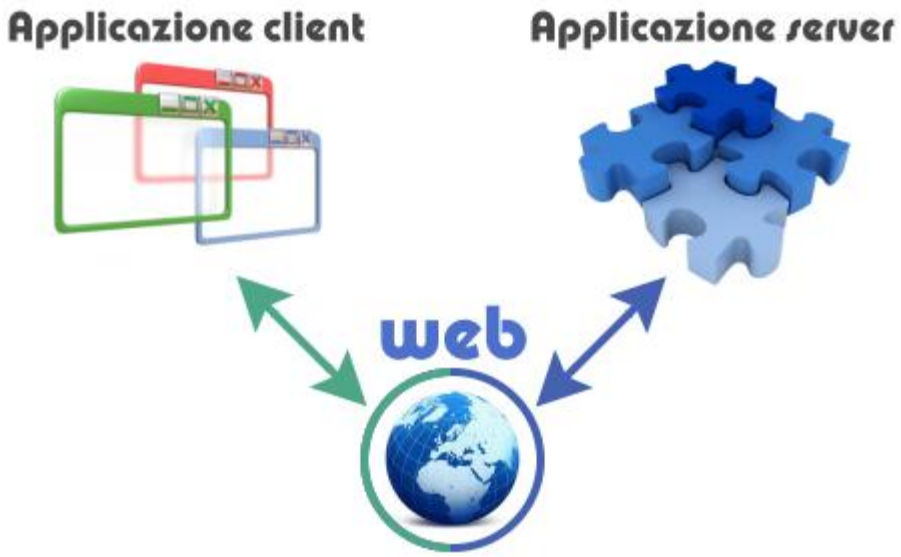
E' buona prassi controllare spesso il parere di questi validatori. Impareremo tante cose dai loro controlli.

1
Reti e programmazione web

Scrivere un'applicazione per il web ci costringe a interagire con un mondo di termini e frasi che dobbiamo padroneggiare nel più breve tempo possibile. La programmazione per il web funziona sul web. Il web è una immensa rete di computer. E' quindi necessario comprendere, anche superficialmente, come funziona una rete di calcolatori e come possano due computer, distanti migliaia di km, comunicare perfettamente e velocemente tra loro.
Questo capitolo si occuperà di fornirci queste nozioni.

Cosa significa programmare per il web?

Abbiamo già accennato che programmare per il web significa scrivere almeno due applicazioni in grado di comunicare tra loro: un'**applicazione client** che invia richieste (e interpreta le risposte) e un'**applicazione server** che riceve le richieste dell'applicazione client, prepara le risposte e le invia al client. Programmare per il web quindi implica necessariamente saper scrivere entrambi i tipi di applicazione. In questo libro imparrremo a usare il browser come applicazione client.



NOTA

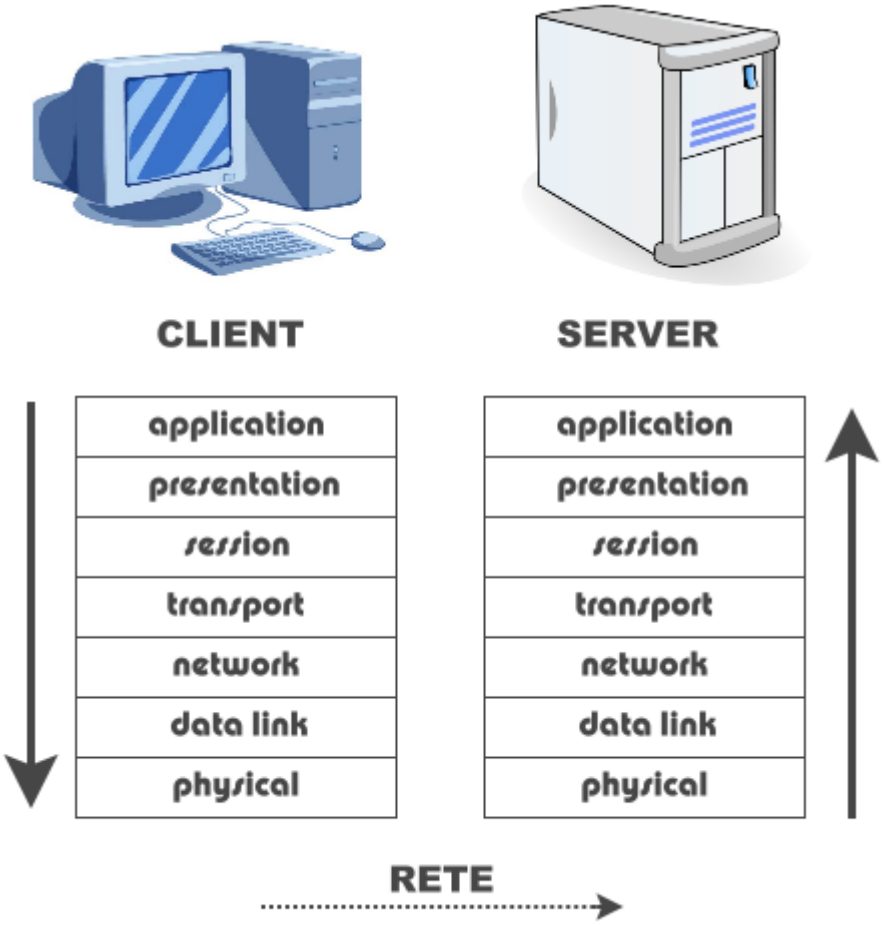
Nel gergo informatico viene spesso usato il termine server non solo per indicare l'applicazione server, ma anche per indicare il computer su cui è in esecuzione tale applicazione. Lo stesso dicasi per il termine client con il quale si tende ad indicare il computer su cui è eseguita l'applicazione client e non invece la stessa applicazione.

A questo punto si pone un problema: come possono comunicare tra loro due applicazioni eseguite su due computer diversi e magari distanti migliaia di km?
Se vogliamo realizzare un'applicazione per il web abbiamo bisogno di rispondere in qualche modo a questa domanda e per farlo è opportuna una breve ed elementare digressione sulle reti di calcolatori.
Come fanno quindi a comunicare due computer distanti migliaia di km?
Noi umani per capirci dobbiamo parlare la stessa lingua. Parlare in lingua italiana ad un cittadino britannico non porta quasi mai a grandi risultati comunicativi.
I computer non fanno eccezione. Per potersi capire hanno bisogno di "parlare la stessa lingua". La lingua che essi usano per comunicare tra loro in rete si chiama **modello ISO/OSI**.
ISO è il principale ente di standardizzazione internazionale, si occupa quindi di rendere standard una certa tecnologia. **OSI** invece è il modello vero e proprio, reso standard appunto dall'ISO. OSI sta per **Open System Interconnection**, ossia interconnessione di sistemi aperti ed è il sistema (la lingua) che i nostri computer usano per parlare tra loro.
Dobbiamo necessariamente vederne il funzionamento più da vicino.

Molti di noi sono connessi a internet tramite il cavo di rete, conosciuto anche come cavo ethernet o RJ-45. Tramite questo cavo, in qualche modo, andiamo sul web e il web arriva sul nostro computer. Ma nel cavo ci sono fili. Fili elettrici. Quindi quello che arriva e che esce dal nostro computer tramite il cavo RJ-45 (o wifi, o modem, il discorso cambia poco) in realtà è corrente elettrica! La corrente elettrica poi diventa misteriosamente una pagina web, un'email, un file o altro. Può sembrare un po' strano, ma è esattamente così e, semplificando al massimo tutto il processo, ciò che rende possibile la trasformazione della corrente elettrica in un'email (e viceversa) è proprio il modello OSI.

LIVELLO 7	APPLICATION
LIVELLO 6	PRESENTATION
LIVELLO 5	SESSION
LIVELLO 4	TRANSPORT
LIVELLO 3	NETWORK
LIVELLO 2	DATA LINK
LIVELLO 1	PHYSICAL

Il modello OSI è composto da 7 livelli, come mostrato nella tabella. Il primo livello, il livello fisico (physical), è quello che si occupa di gestire la corrente elettrica che arriva dal cavo RJ-45 (o le onde che vengono intercettata e trasformate dall'antenna wifi). Questo livello, una volta eseguite alcune operazioni sulla corrente elettrica, comunica i propri risultati al livello immediatamente superiore (data link) il quale, dopo aver eseguito le operazioni per cui è preposto, passa a sua volta il risultato al livello superiore e così via, fino all'ultimo livello che è quello gestito dall'applicazione (non a caso si chiama application). Il modello OSI ha quindi trasformato corrente elettrica in dati gestibili dalla nostra applicazione. Ognuno dei 7 livelli infatti si occupa di eseguire operazioni sugli originali segnali elettrici che arrivano dal cavo modificandoli, livello dopo livello, fino a renderli gestibili dalla nostra applicazione.
Lo stesso identico percorso, ma in senso inverso, avviene quando l'applicazione client invia una richiesta all'applicazione server. In tale situazione il modello OSI viene percorso dal livello application al livello fisico e, livello dopo livello, la richiesta dell'applicazione client viene trasformata in segnali elettrici che vengono spediti sul cavo ethernet e che arrivano poi sul server il quale ricomincia tutto da capo, dal livello fisico fino a quello application, come mostrato nella figura qui in basso.



Nella figura è mostrato il flusso dei dati quando l'applicazione client invia una richiesta all'applicazione server. Come si può osservare tutto parte dal livello application. La richiesta poi scende progressivamente di livello fino ad arrivare al livello fisico. A quel punto, tramite le rete, la richiesta del client viene inviata al server che la interpreta partendo dal livello fisico e arrivando fino al livello application.
Una volta che il server ha eseguito le operazioni richieste, risponde al client e quindi il flusso dei dati si inverte (non mostrato nella figura).

Le operazioni svolte dai singoli livelli sono molto delicate e devono quindi rispettare regole ben precise. Tali regole prendono il nome di protocolli. **Ogni livello del modello OSI usa uno o più protocolli**.
Approfondire la conoscenza dei protocolli esula dagli scopi di questo libro. Tuttavia è necessario un sommario elenco dei più comunemente usati e dei rispettivi utilizzi. Ciò è necessario perché la terminologia tecnica usata durante la programmazione web è fortemente legata a tali concetti. E' quindi doveroso capire cosa si intende quando di parla di TCP o HTTP.

HTTP(s)

L'HyperText Transfer Protocol (protocollo di trasferimento di un ipertesto) è usato come principale sistema per la trasmissione d'informazioni sul web. Le specifiche del protocollo sono gestite dal World Wide Web Consortium (W3C).

Il protocollo HTTP viene usato a **livello application** nel modello OSI e ciò ci consente di utilizzarlo direttamente dal nostro browser. Ci basta infatti osservare il prefisso http:// presente nella barra degli indirizzi del browser quando visitiamo un sito web.

La (s) sta per Secure e indica la versione sicura del protocollo HTTP.

Nella discesa dal livello application al livello fisico, questo protocollo fa uso del protocollo TCP che vedremo in seguito.

Se noi inviamo una richiesta tramite il protocollo HTTP, ci dovrà essere un server in attesa di tale richiesta. In genere i server HTTP restano in ascolto sulla porta 80 usando, come detto, il protocollo TCP. Se la richiesta viene inoltrata in HTTPs, allora la porta è la 443.

Sul concetto di "porta" ritorneremo in seguito. Per adesso pensiamo al server come fosse una casa con tante porte. Ogni applicazione client bussa su una porta diversa e il server la apre solo se chi bussa lo fa nel modo giusto.

FTP (File Transfer Protocol)

Anche questo protocollo nel modello OSI è usato a **livello application** e somiglia molto al protocollo HTTP. Viene infatti usato per gli stessi scopi e, scendendo nei livelli del modello OSI, usa anch'esso il protocollo TCP. FTP però, a differenza di HTTP, utilizza due connessioni separate per gestire comandi e dati.

Anche in questo caso, se effettuiamo una richiesta da un'applicazione client, ci dovrà essere un server in attesa della richiesta. Un server FTP rimane tipicamente in ascolto sulla porta 21 TCP a cui si connette il client.

SMTP (Simple Mail Transfer Protocol)

E' il protocollo standard per l'invio via internet di e-mail. Si trova anch'esso a **livello application** e viene utilizzato dai client di posta (outlook, thunderbird, ecc.) per inviare le email. I protocolli utilizzati per ricevere le email invece sono POP e IMAP.

POP (Post Office Protocol)

Protocollo a **livello application**. Ha il compito di consentire il download delle email da un server email. Viene utilizzato a tale scopo dai client di posta (outlook, thunderbird, ecc.). Come abbiamo visto, il protocollo per inviare la posta è invece il protocollo SMTP.

IMAP (Internet Message Access Protocol)
A volte anche chiamato Interactive Mail Access Protocol, è un protocollo di comunicazione per la ricezione di e-mail. Il significato "Interactive Mail Access Protocol" è stato valido fino alla versione 3, dalla quarta in poi è cambiato in "Internet Message Access Protocol". Il protocollo è stato creato come alternativa più moderna all'utilizzatissimo POP. Anche tale protocollo è a **livello application** ed è infatti anch'esso utilizzato dai client di posta. La porta predefinita del server IMAP sull'host è la 143. Se si utilizza una connessione sicura tramite SSL, allora la porta è la 95.

UDP (User Datagram Protocol)
E' un protocollo usato a **livello transport**. E' meno affidabile del suo "cugino" TCP, ma in compenso è più veloce perché non tiene nota dello stato della connessione. E' usato molto spesso nelle comunicazioni di video e audio in streaming, situazione in cui si può anche perdere qualche informazione durante l'invio dei dati. E' usato di solito in combinazione con il protocollo IP.

TCP/IP
Questa sigla indica due protocolli distinti, TCP e IP, usati su due livelli distinti del modello OSI, ma poiché vengono usati sempre insieme vengono anche nominati insieme.
TCP (Transmission Control Protocol)
Protocollo di rete a pacchetto di **livello transport** che si occupa di controllo di trasmissione. Su di esso si appoggiano gran parte delle applicazioni della rete internet. E' molto più usato di UDP poiché TCP prevede la correzzione degli errori, cosa non prevista invece da UDP. Per questo motivo UDP è più veloce di TCP.
IP (Internet Protocol)
Protocollo a **livello network** del modello OSI. Attualmente siamo alla versione 6, ma la più diffusa resta la versione 4 (IPv4). Il protocollo è il più usato a livello network. E' il protocollo che viene chiamato in causa ogni qualvolta ci viene chiesto: "quale è l'IP del tuo computer?". Un classico esempio di indirizzo IPv4 è 192.168.0.1.

Esistono molti altri protocolli che vengono utilizzati nel modello OSI. Quelli presentati nel precedente elenco però sono i più conosciuti e i più usati.

A margine di questa brevissima digressione sulle reti è opportuno parlare dell'**architettura software** che identifica un'**applicazione di rete** nel suo complesso. Una tale applicazione infatti può avere due tipi di architettura software:

- 2-tier (2-livelli)
- 3-tier (3-livelli)

Semplificando al massimo, la differenza tra le due architetture è riconducibile al modo in cui l'applicazione client interagisce con il database. Nell'**architettura 2-tier** l'applicazione client esegue direttamente le richieste alla fonte dati (in genere un database). Nell'**architettura 3-tier** l'applicazione client inoltra le proprie richieste ad un'applicazione server la quale, a sua volta, esegue le necessarie richieste alla fonte dati, preleva i dati, li elabora e li invia formattati nel giusto modo all'applicazione client.

Bene.
Questa breve digressione sulle reti di calcolatori era necessaria per poter intraprendere lo studio di argomenti vasti e complessi come quelli che ci accingiamo ad affrontare.

Il browser

Il browser rappresenta la porta d'accesso al web ed è un'applicazione molto complessa. Uno studio approfondito di tale software esula dai compiti di questo testo. In questa sede ci basterà sapere che il browser svolge due compiti importantissimi:

- traduce codice HTML in pagine web
- scambia dati con la rete

E' un po' come dire che legge e scrive: legge il codice che gli arriva dalla rete e scrive codice da inviare alla rete. Nell'introduzione abbiamo visto come, per espletare tali compiti, il browser utilizzi il modello ISO/OSI e in particolare alcuni protocolli tra cui HTTP.

Poiché abbiamo deciso di usare il browser per realizzare la nostra applicazione client, il nostro primo compito da programmatori per il web diventa quello di scrivere codice che il browser sia in grado di leggere.
Sappiamo che per visualizzare un sito web o una pagina web in un browser è necessario cliccare su un link o digitare l'URL (conosciuto comunemente come "indirizzo web") nella barra degli indirizzi. In questo modo il browser si collega a internet e ci mostra la pagina che abbiamo richiesto, se esiste. Tuttavia la pagina web che ci mostra il browser è stata recuperata chissà da dove, chissà da quale parte del mondo. Uno dei compiti del browser è quindi quello di inviare una richiesta del tipo «posso avere la pagina web, per favore?» ad un server web che si trova da qualche parte nella rete. Il programma server, raccoglie la richiesta dal browser web, cerca di rintracciare (o di costruire in quel momento) la pagina web richiesta dal browser e poi formula una risposta. Questa risposta varia a

seconda che il programma server sia o meno riuscito a trovare la pagina web richiesta. Se il server trova la pagina, invia la risposta in formato HTML al browser web che l'ha richiesta. Il browser a questo punto legge il codice HTML (e altre informazioni) proveniente dal server e lo traduce in una pagina web. E' questo ciò che accade ogni volta che clicchiamo su un link nel web: ogni click su un link mette in moto tutto il processo appena descritto.

Abbiamo già detto e qui ripetiamo che quella appena descritta rappresenta l'**architettura client-server** ed è alla base della programmazione per il web: **il browser (applicazione client) richiede una pagina ad un'applicazione server; il server cerca la pagina richiesta e la spedisce al client**. Nel nostro caso il server invia al client la pagina in formato HTML.

Ecco perché, dal prossimo capitolo, inizieremo il cammino che ci porterà a scrivere codice che il browser è in grado di capire.

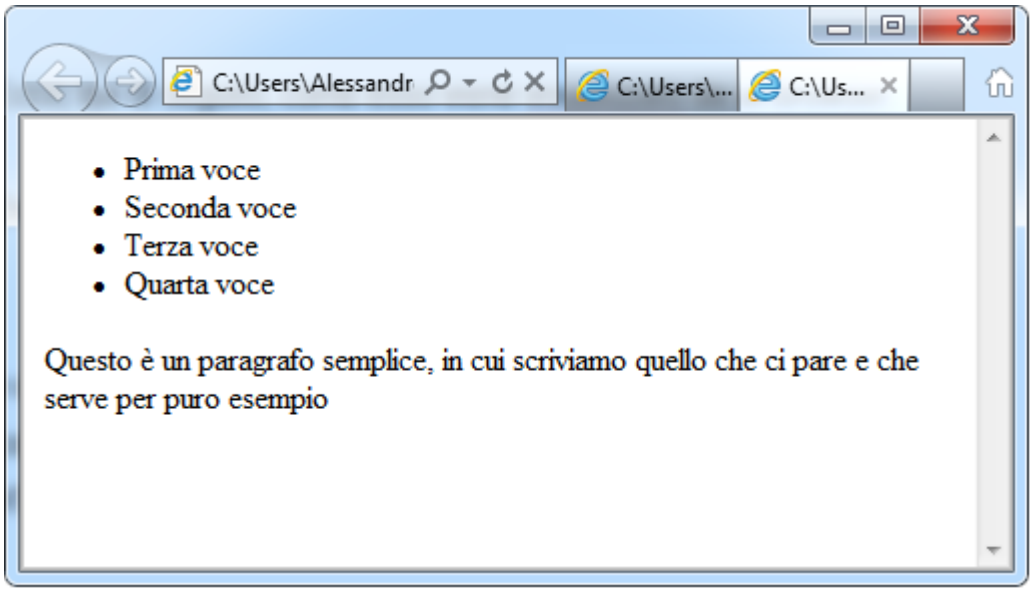
2
Il cosa e il come: HTML e CSS

A dispetto della stranezza del titolo di questo capitolo, esso rappresenta un concetto estremamente importante nella programmazione web lato client. Quando produciamo codice per il browser è assolutamente consigliato dividere il cosa visualizzare dal come visualizzarlo. **Il "cosa" viene indicato tramite codice HTML, mentre il "come" lo decidono i fogli di stile, conosciuti con il nome di CSS** (**C**ascading **S**tyle **S**heets). Questo concetto deve essere un mattone fondamentale in tutto il nostro processo formativo! Senza di esso non siamo nessuno!
Cerchiamo di comprendere meglio un tale importantissimo concetto facendo un esempio con codice vero. In questo momento non è importante comprendere il codice, c'è un libro intero da leggere per comprenderlo. Quello che ora dobbiamo osservare è il cosa e il come!
Ammettiamo che questo sia il nostro codice HTML (il cosa):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
          <meta http-equiv="content-type"
                content="text/html; charset=UTF-8">
          <title>Il cosa e il come</title>
    </head>
    <body>
          <div id="header">
                <ul>
```

```
                        <li>Prima voce</li>
                        <li>Seconda voce</li>
                        <li>Terza voce</li>
                        <li>Quarta voce</li>
                  </ul>
            </div>
            <div id="center">
                  <p>Questo è un paragrafo semplice, in
cui scriviamo quello che ci pare e che serve per puro esempio
                  </p>
            </div>
      </body>
</html>
```

Ecco, qui di seguito, come esso viene visualizzato da un browser.

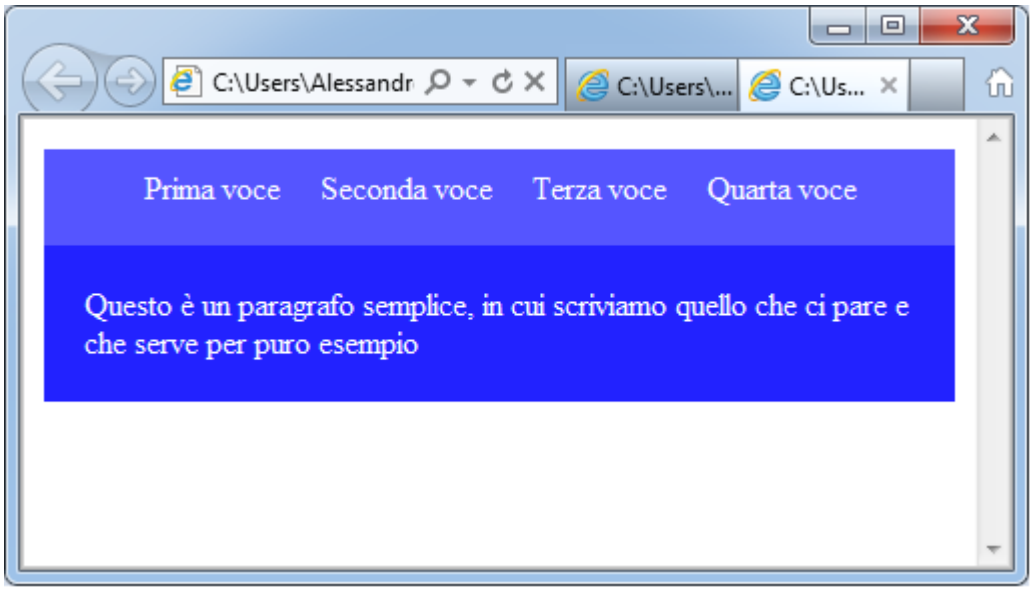


Ora ammettiamo che questo sia un foglio di stile (il come):

```
#header {
    width:100%;
    background-color:#55F;
    color:white;
}
#header ul{
    margin:0px;
```

```
    padding:0px;
    overflow:hidden;
    list-style-type:none;
}
#header ul li{
    float:left;
    padding:10px;
}
#header ul li:hover{
    background-color:#88F;
}
#center {
    width:100%;
    height:auto;
    background-color:#22F;
    color:white;
    padding-top:20px;
    padding-bottom:20px;
}
#center p{
    margin:0px;
}
```

e associamo questo foglio di stile al codice HTML di prima (vedremo in seguito come associare un foglio di stile ad un documento HTML). Fatto questo, visualizziamo nuovamente, nel nostro browser, lo stesso HTML di prima. Ecco cosa vedremo:



Stesso identico codice HTML, due risultati estetici molto diversi!
Il cosa e il come, cioè il contenuto (HTML) e l'aspetto (CSS).

Possiamo comprendere meglio il codice appena scritto, osservandolo in azione qui (proviamo a passare il mouse sopra il testo in alto):
http://www.alessandrostella.it/lato_client/cosaCome/cosaCome_00.html

La Legge: il w3c

In questo libro sarà ripetuto molte volte: quando si parla di HTML e CSS bisogna sempre fare riferimento alle raccomandazioni e agli standard scritti dal W3C, World Wide Web Consortium, il cui sito e le cui direttive devono rappresentare la nostra "legge", il faro che ci guida nella scrittura del codice! Tutto ciò che vìola le direttive del w3c andrebbe accuratamente evitato. Il consorzio è consultabile a questo indirizzo web:
http://www.w3.org
In questo libro ci atterremo scrupolosamente alle direttive del w3c a cui si farà molto spesso riferimento. I motivi per cui è opportuno agire in questo modo possono essere letti qui:
http://www.w3.org/Consortium/mission
Il w3c non lavora per nessuno in particolare. Il w3c lavora per tutti, lavora per rendere realtà concetti quali "web for all" e "web on everything". E' grazie al w3c, per esempio, che oggi possiamo scrivere lo stesso codice HTML sapendo che esso produrrà la stessa pagina web sia su Internet Explorer, sia su Firefox sia su Chrome ecc. Senza la costante

opera di standardizzazione espletata con pazienza e sapienza dal w3c, oggi tutto sarebbe molto più complesso.

Gli strumenti di lavoro

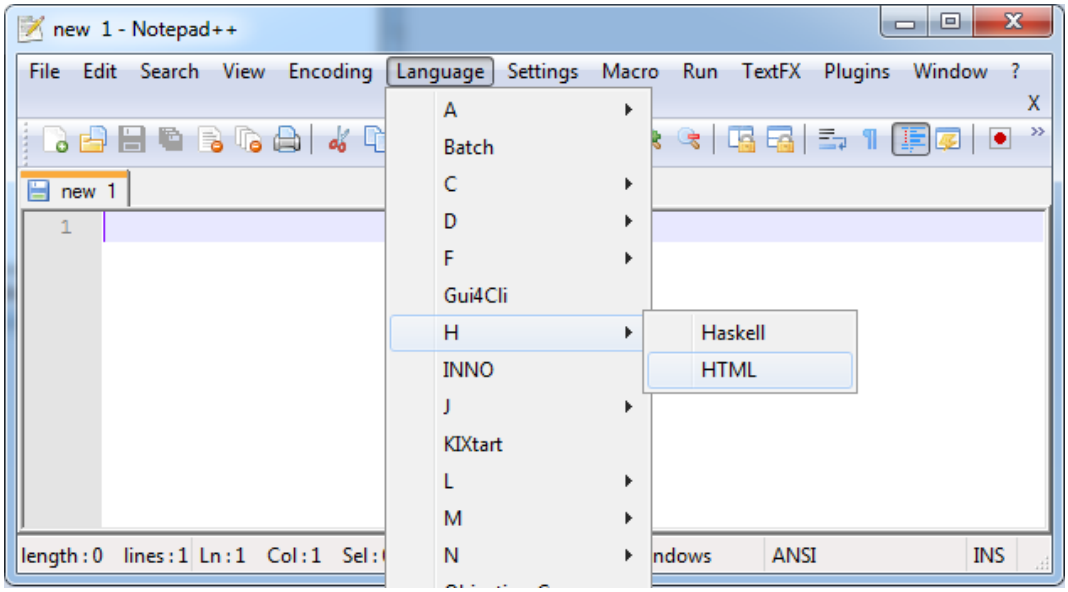
Sia il codice HTML sia quello CSS sono semplice testo, né più né meno. Quindi per scrivere codice HTML e CSS può bastare un semplice editor di testo. Tuttavia un editor di testo diventa poco produttivo nel momento in cui le pagine da scrivere diventano appena un po' complesse. Per fortuna ci sono davvero tanti programmi in grado di aiutarci nella scrittura di documenti HTML più complessi. Per iniziare in modo produttivo la scrittura di codice HTML ci possiamo affidare senz'altro a **Notepad++**. E' gratis, ne esiste una versione che non necessita di installazione e ci fornisce diversi aiuti. Possiamo scaricarlo da qui:

http://notepad-plus-plus.org/

Sulla sinistra troveremo il link "download". Una volta nella pagina di download scegliamo tra la versione installer (procedura automatica di installazione), oppure la versione zip package (si decomprime il file zip e si lancia direttamente l'eseguibile).

Notepad++ è stato tradotto in italiano, ma noi useremo la versione in inglese perché l'inglese è la lingua dell'informatica.

Una volta lanciato, andiamo in "Language/H/HTML" per indicare al programma che il file correntemente aperto sarà scritto in HTML. Oppure "Language/C/CSS" se vogliamo creare un file CSS.



Tra i tanti aiuti che Notepad++ è in grado di darci, noi ne useremo solo due:

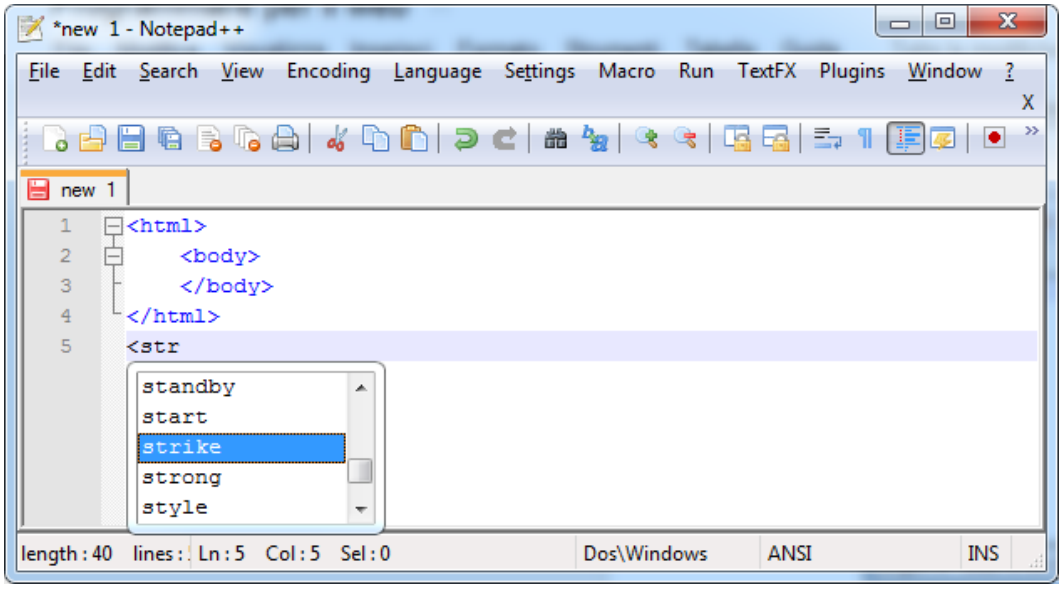
- Evidenziazzione dei tag

- Aiuto in digitazione

Per capire di cosa parliamo, ci basta scrivere poche righe di codice.
Dopo aver comunicato a Notepad++ che il file in questione è (o sarà) scritto in HTML, scriviamo il seguente codice.

```
<html>
    <body>
    </body>
</html>
```

Ora, con le frecce della tastiera, spostiamo il cursore su uno qualunque dei tag. Il sistema evidenzierà in automatico apertura e chiusura di quel tag. Questo è un aiuto molto importante che ci aiuterà a capire in modo molto rapido inizio e fine di un tag e se i nostri tag sono tutti correttamente chiusi.
L'aiuto in digitazione invece ci aiuta a scrivere correttamente i nomi dei tag. Ammettiamo di voler inserire il tag . Come mostrato nella figura qui di seguito, posizioniamoci alla fine del testo già inserito e digitiamo <str. A questo punto, tenenndo premuto il tasto CTRL della tastiera, premiamo la barra spaziatrice. Otterremo tutte le keyword che iniziano per "str". Ci basterà quindi scorrere in basso di una posizione e premere invio. Notepd++ scriverà per noi "strong". Questo ci aiuterà a non commettere errori tipici come, per esempio, scrivere <stong> invece di .
Non useremo altri aiuti.



Avremmo potuto scegliere tanti altri editor. La scelta è ricaduta su Notepad++ per un motivo ben preciso: aiuta, ma non troppo. In questa fase infatti è opportuno sforzarsi di scrivere il codice HTML senza aiuti esterni e ciò al fine di imprimere nella nostra mente i concetti.
Quando diventeremo dei professionisti del web, questo semplice programma non sarà più in grado di aiutarci. Avremo bisogno di software più complessi e utilissimi come ad esempio Adobe Dreamweaver e Microsoft Visual Studio, ma ora non è il momento giusto.

3
HTML

HTML è il cuore della programmazione web lato client. Moltissime persone ne fanno un uso quotidiano e spesso non ne sono consapevoli: leggere online le ultime notizie sportive, scrivere un messaggio su facebook, leggere la posta sul web; tutte queste attività possono essere espletate perché esiste HTML! Internet stessa si è diffusa in tutto il mondo perché esiste HTML.
E' chiaro quindi che, se vogliamo diventare programmatori per il web, è di cruciale importanza conoscere HTML. HTML è un linguaggio di pubblico dominio la cui sintassi è stabilita dal World Wide Web Consortium (W3C) ed è a questo organismo che bisogna fare sempre riferimento quando si parla di HTML (e non solo). Per questo motivo, come già accennato, tutto quello che studieremo in questo capitolo avrà come fonte il w3c.
Ad esempio, sul sito ufficiale del w3c possiamo leggere: "the World Wide Web Consortium (W3C) recommends lowercase in HTML 4 and demands lowercase tags in XHTML", ossia **è raccomandato scrivere i tag HTML in minuscolo** (anche se possiamo scriverli in maiuscolo). Tra poco capiremo cosa sono i tag! Intanto ricordiamoci di scriverli in minuscolo...
Adesso concentrazione massima. Da questo capitolo iniziamo a fare sul serio.

Cosa è?

Hyper**T**ext **M**arkup **L**anguage = **HTML**.
Prima di dare una spiegazione al geroglifico qui sopra, è opportuno ricordare che il browser web è in grado di comprendere questo linguaggio. Ossia, **se noi scriviamo codice HTML, il browser è in grado di leggerlo e "tradurlo" in una pagina web**.

Da ciò deriva chiaramente che, nella programmazione web, è fondamentale imparare a scrivere codice HTML. Infatti tutte (o quasi) le pagine che noi visitiamo sul web, tramite il nostro browser, sono scritte in HTML e tradotte dal browser in pagine web.

Html è un **markup language**, non un linguaggio di programmazione. Questo significa che in HTML non ci sono le tipiche istruzioni condizionali (if, else if, ecc.) e cicliche (for, while, ecc.) proprie di un linguaggio di programmazione. Un markup language è costituito da un insieme di **markup tags**. Essi vengono utilizzati per descrivere una pagina web; servono per comunicare al browser che in un certo punto vogliamo una casella di testo, in un altro certo punto vogliamo un'immagine e così via.

Ma cosa è markup tag in HTML?

Un **markup tag HTML** (o semplicemente tag) è costituito da una keyword (parola chiave) compresa tra due parentesi acute, come ad esempio <body>. In genere i tag HTML viaggiano in coppia, ad esempio e . Il primo tag, in una coppia di tag, viene chiamato tag di apertura, il secondo tag viene chiamato tag di chiusura.

Ora, al fine di produrre una pagina web, il browser ha bisogno di leggere codice HTML, ma abbiamo appena visto che il codice HTML altro non è che un insieme di tag. Quindi il browser prende in ingresso codice HTML e ci fa vedere la pagina web che quel codice rappresenta. Ecco pronto un esempio pratico.

Se noi creiamo un file di testo con questi caratteri

```
<html>
    <head>
```

```
        <title>Titolo della pagina</title>
    </head>
    <body>
        <h1>Il mio primo titolo h1</h1>
        <p>Il mio primo paragrafo</p>
    </body>
</html>
```

lo salviamo con nome "primo.html" e lo apriamo con un browser web, vedremo la pagina web mostrata qui di seguito.

Siamo passati da un file di testo pieno di tag, ad una pagina web in cui dei tag non c'è neanche l'ombra. Quindi il browser "legge" il codice HTML e lo trasforma (secondo specifiche regole) nella pagina web corrispondente.
Il banale esempio proposto racchiude alcune **importanti informazioni** che dobbiamo assolutamente memorizzare:

- un documento HTML deve iniziare con il tag <html>
- il tag <html> deve contenere il tag <head> e il tag <body>
- il tag <head> deve contenere il tag <title>
- i tag possono essere (e spesso sono) annidati, cioè dentro un tag possono esserci altri tag (il tag <body> è annidato nel tag <html>)
- i tag devono essere chiusi in ordine inverso a quello di apertura, cioè l'ultimo tag aperto deve essere il primo ad essere chiuso

- è uso comune (e ottima regola) indentare il codice HTML per renderlo più leggibile

Le poche informazioni contenute in questo paragrafo racchiudono tutte le regole necessarie per scrivere del buon codice HTML.
Bene, ora dovrebbe essere chiaro cosa è l'HTML e perché è di fondamentale importanza nella programmazione web.

I tag HTML

Abbiamo già accennato ai tag HTML (ricordiamo? quelli da scrivere sempre in minuscolo), ora li affronteremo.

Doctype, <html>, <head>, <body>

Scrivere un documento HTML costringe necessariamente a rispettare alcune regole. Un semplice documento HTML avrà la seguente forma:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
          <meta http-equiv="content-type"
                content="text/html; charset=UTF-8">
          <title>Titolo della pagina</title>
    </head>
    <body>
    </body>
</html>
```

Analizziamo questo codice riga per riga.

Alle righe 1 e 2 troviamo subito qualcosa di nuovo: è il doctype. L'uso di <!DOCTYPE> non è obbligatorio e può essere omesso, ma si sconsiglia di ometterlo (lo sconsiglia il w3c!). A cosa serve? Serve ad indicare al browser come interpretare il documento che sta leggendo. Non è il momento di entrare nei dettagli del codice. Quello che per il momento è importante sapere è che, quando scriviamo codice HTML4, è opportuno iniziare il file con esattamente quel codice, insomma del tipo copia/incolla da fare prima del tag <html>. Ci sono invece alcune precisazioni da fare. La prima di tutte è che il **<!DOCTYPE> non è un tag HTML!**

Sebbene troppo spesso, sia in rete sia sui libri di testo, si indica questo oggetto come un tag, esso non lo è. Sul sito del w3c infatti possiamo leggere:
"The doctype declaration is not an HTML tag; it is an instruction to the web browser about what version of the markup language the page is written in."
Si tratta quindi di una **istruzione per il browser finalizzata a indicare le regole che esso dovrà usare per leggere correttamente la pagina**.
Un'altra precisazione da fare è che la dichiarazione <!DOCTYPE> deve essere il primo elemento ad aprire il documento HTML, prima anche del tag <html>.
Alla riga 3 incontriamo il tag dei tag: <html>.
Per essere ritenuto valido, un documento HTML deve contenere almeno i tag <html>, <head>, <title>, <body> e il tag <body> non deve essere vuoto. In caso contrario il browser sarà comunque in grado di interpretare il documento, ma i validatori del w3c non lo valideranno. Per verificare questa affermazione raggiungiamo il seguente indirizzo web
http://validator.w3.org/#validate_by_input
e, nella casella di testo bianca, incolliamo il codice HTML scritto poche righe fa. Una volta incollato il codice, premiamo il bottone "Check". Risultato? Errore!
Line 10, Column 7: end tag for "BODY" which is not finished
Cosa significa? Significa che il tag <body> è vuoto, non contiene niente e ciò rende non valido il codice HTML scritto.
Se però aggiungiamo una paragrafo nel tag <body> (vedremo in seguito cosa è un paragrafo), modificando il precedente codice HTML nel seguente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
          <meta http-equiv="content-type"
                content="text/html; charset=UTF-8">
          <title>Titolo della pagina</title>
    </head>
    <body>
          <p>Questo è un paragrafo</p>
    </body>
</html>
```

e se proviamo a validare questo nuovo codice HTML, il validatore darà luce verde! Il nostro codice è ora valido:
This document was successfully checked as HTML 4.01 Strict!

I tag **<html>**, **<head>** e **<body>** sono i cardini di un documento HTML e **devono necessariamente essere presenti**. In linea generale si può dire che il tag <html> rappresenta tutto il documento, mentre <body> rappresenta la parte del documento che il browser mostrerà a video, ossia il corpo del nostro documento HTML, la pagina web visibile all'utente finale. Il tag <head> invece contiene informazioni utili al browser, ma che non vengono mostrate a video.
Alle righe 4, 5, 6, 7 e 8 troviamo proprio il tag <head>.

```
<head>
    <meta http-equiv="content-type"
          content="text/html; charset=UTF-8">
    <title>Titolo della pagina</title>
</head>
```

Il tag **<head>** è un contenitore di altri tag e **deve essere aperto subito dopo il tag <html> e chiuso prima dell'apertura del tag <body>**.
All'interno del tag <head> possono trovarsi script, istruzioni per il browser, indicazioni sui fogli di stile (vedremo in seguito cosa sono), meta informazioni, il titolo del documento e altro ancora. Per maggiore dettaglio, ecco l'elenco dei tag che possono essere inclusi nel tag <head>:
<title>, <base>, <link>, <meta>, <script> e <style>.

Il tag **<meta>** rappresenta meta informazioni. Questo tag può avere diversi attributi che ne dettagliano il tipo. In questo caso sono http-equiv e content, ma ce ne sono altri.
Il tag **<title>** rappresenta il titolo della pagina, quello che compare sulla scheda del browser.
I due tag appena trattati devono essere presenti nel tag <head>, ma come abbiamo visto non sono gli unici. Tuttavia, ai fini della validazione di un file HTML, gli altri possono anche non essere presenti.
Per informazioni sugli altri tag si rimanda più avanti nel libro quando entreremo nel vivo della scrittura del codice e incontreremo alcuni di questi tag.

<hn>

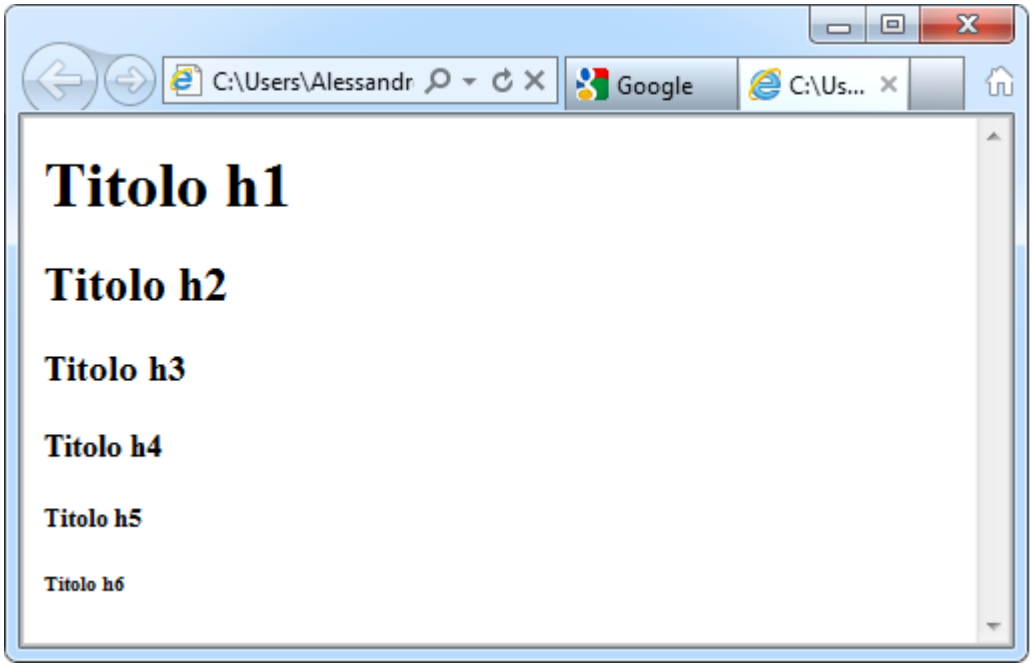
Il tag <hn> non esiste. E' solo un modo per indicare i vari tag che vuole rappresentare e cioè i tag <h1>, <h2>, <h3>, <h4>, <h5>, <h6>. Essi sono chiamati "headings" e

vengono usati per evidenziare i titoli. Per esempio il titolo "I tag HTML" potrebbe essere rappresentato così:
<h3>I tag HTML</h3>

A seconda dell'importanza del titolo si possono usare 6 tipi di Headings diversi, da **<h1>** a **<h6>**. <h1> rappresenta un titolo di primaria importanza, <h6> un titolo di ultima importanza. Nell'esempio abbiamo usato <h3>.
Facciamo attenzione al modo in cui usiamo questi tag. Non devono essere usati per mettere un testo in grassetto né, più in generale, per una questione estetica! Gli headings devono essere usati solo e soltanto per indicare effettivamente un titolo. Questo per diversi motivi, tra i quali quello che i motori di ricerca danno particolare risalto a questi tag. Un uso scorretto può quindi intaccare l'affidabilità delle ricerche sul web.
Ecco il codice che li contiene tutti (attenzione! Ricordiamo che questo codice non è valido per i validatori w3c, ma viene comunque interpretato dai browser e noi lo usiamo per semplicità!).

```
<html>
    <body>
          <h1>Titolo h1</h1>
          <h2>Titolo h2</h2>
          <h3>Titolo h3</h3>
          <h4>Titolo h4</h4>
          <h5>Titolo h5</h5>
          <h6>Titolo h6</h6>
    </body>
</html>
```

Ed ecco cosa mostra il browser.



<p>

Lo abbiamo già incontrato nell'esempio iniziale. E' il tag **<p>** e viene usato per indicare paragrafi.

Esempio

```
<p>Questo è un paragrafo</p>
<p>Questo è un altro paragrafo</p>
```

Non dimentichiamo di chiudere questo tag perché alcuni browser vanno in crisi se non lo trovano chiuso. In genere i browser "intuiscono" dove chiudere un tag non chiuso, ma è sempre opportuno chiudere tutti i tag; il tag <p> in particolare!

Il primo problema che si incontra nell'uso del tag <p> è il mandare a capo il paragrafo.

Se vogliamo andare a capo la prima cosa che viene in mente è di dare invio sulla tastiera, scrivendo un codice simile al seguente.

```
<p>Questo è un testo qualunque che finisce qui.
Dopo il punto voglio andare a capo e quindi premo invio</p>
```

Beh, non funziona.

Una volta che il browser tradurrà questo codice in una pagine web, il paragrafo verrà mostrato tutto sulla stessa riga.

E allora come facciamo a mandare a capo il testo in un paragrafo?

Se abbiamo la necessità di mandare a capo il testo, possiamo usare il tag **
** all'interno del tag <p>.
Esempio:

```
<p>Ad un certo punto devo andare a capo <br>e continuare a
scrivere su un'altra riga</p>
```

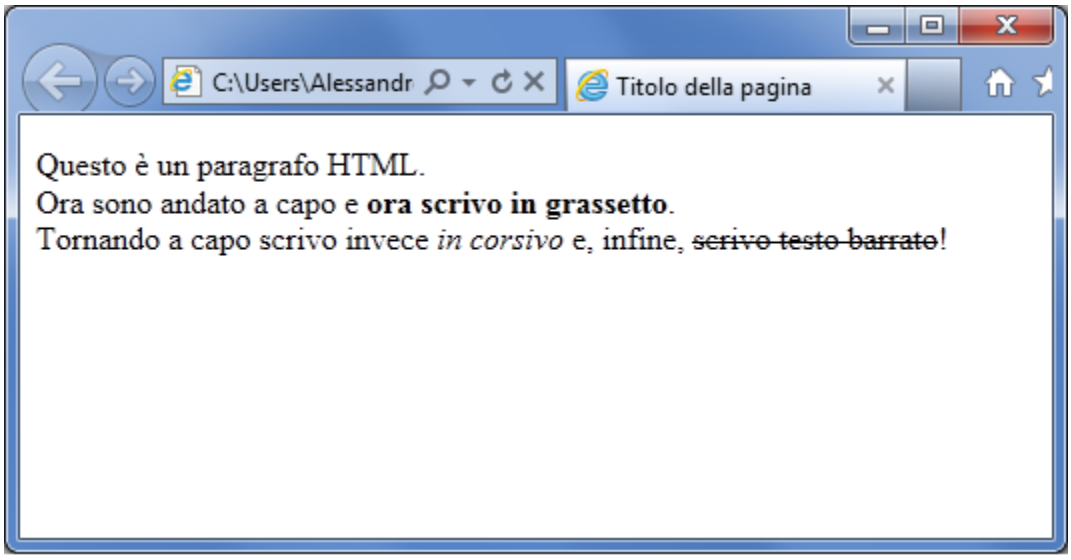
Senza il tag
 non serve dare "invio" da tastiera perché così facendo il browser non "capisce" che vogliamo mandare il testo a capo. Per comunicare al browser che, in un determinato punto, vogliamo mandare a capo il testo è necessario usare il tag
.
E' bene osservare come **in HTML4 il tag
 non preveda il tag di chiusura**!
Oltre a mandare a capo un testo potremmo volere anche formattarlo in qualche modo, per esempio in grassetto, corsivo, sottolineato, ecc. Per mettere il testo in grassetto possiamo usare il tag ****.
Esempio:

```
<p>Questo è <strong>molto importante</strong></p>
```

Possiamo inoltre usare il tag **** per il corsivo e il tag **<del>** per il testo barrato. Per un elenco completo dei tag di formattazione del testo si rimanda sul sito del w3c.
Ecco il codice per un esempio completo.

```
<html>
    <body>
          <p>Questo è un paragrafo HTML.<br>
                Ora sono andato a capo e
                <strong>ora scrivo in grassetto</strong>.<br>
                Tornando a capo scrivo invece<em>in corsivo</em>
                e, infine, <del>scrivo un testo barrato</del>!
          </p>
    </body>
</html>
```

Ed ecco cosa mostra il browser.



Possiamo osservare il risultato qui:

http://www.alessandrostella.it/lato_client/html4/html4_00.html

<a>

Un collegamento ipertestuale (o link) è una parola, o un gruppo di parole, o anche un'immagine che si può cliccare per passare a un nuovo documento o ad una nuova sezione all'interno del documento corrente.

Generalmente quando si sposta il cursore del mouse su un link in una pagina web, la freccia del mouse si trasforma in una manina.

Il tag usato per gestire un link è, ovviamente, un tag estremamente importante: è il tag **<a>**

Esempio

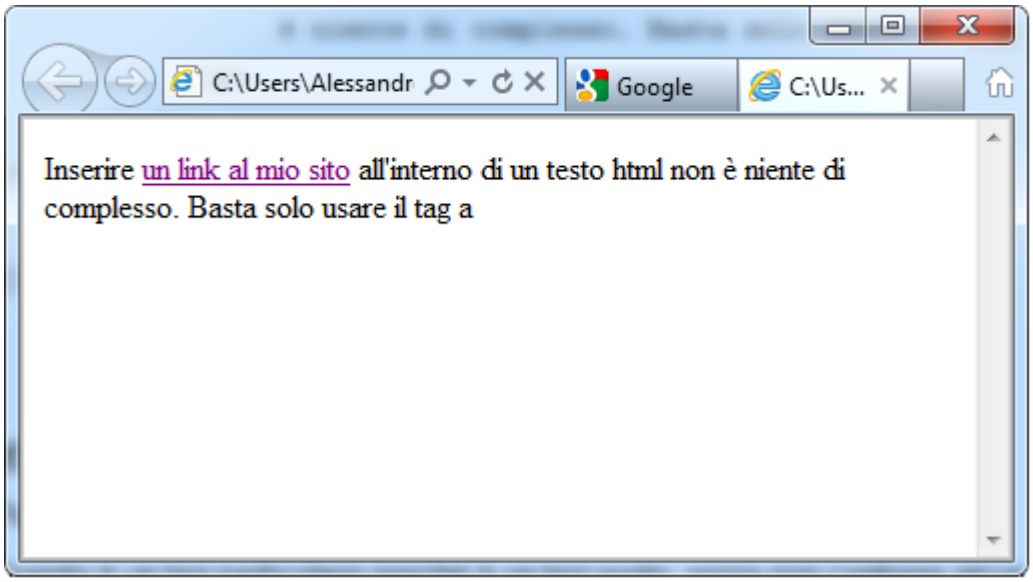
```
<a href="http://www.alessandrostella.it/">Link ad Alessandro
Stella</a>
```

Ecco il consueto codice completo.

```
<html>
    <body>
          <p>Inserire <a href="http://www.alessandrostella.it/">
                un link al mio sito</a> all'interno di un testo
                html non è niente di complesso.
                Basta solo usare il tag a.
          </p>
```

```
    </body>
</html>
```

E la rispettiva pagina web.



Possiamo osservare il risultato qui:
http://www.alessandrostella.it/lato_client/html4/html4_01.html

Alcune necessarie osservazioni.
In base a quanto visto fino ad ora ci saremmo aspettati qualcosa del genere:

```
<a>Link ad Alessandro Stella</a>
```

ma questo codice non funziona. Non funziona perché il tag <a>, per svolgere la sua funzione, necessita dell'attributo "href" (vedremo più avanti cosa è un attributo.). Se omettiamo l'attributo href il browser non sa dove punta il link e quindi lo ignora, trattandolo come fosse semplice testo.
Un'altra osservazione.
Sebbene non sia necessario, **quando si crea un link verso la home page di un sito** è opportuno terminare sempre l'indirizzo indicato dall'attributo "href" con lo slash (cioè il carattere "/"). Nell'esempio infatti è href="http://www.alessandrostella.it/". Il link finisce con "/". E' troppo complesso (e poco utile) in questo momento spiegare la motivazione di una tale necessità, resta tuttavia il concetto: **terminare sempre il link indicato dall'attributo "href" con lo slash**!

Per gestire le immagini abbiamo a disposizione il tag **<img>**

Questo è un tag particolare perché è un tag vuoto, ossia non contiene alcun elemento, ma solo attributi e non prevede quindi il tag di chiusura, cioè </img>.

Esempio

```
<img src="w3schools.jpg" width="104" height="142" alt=”testo
alternativo” />
```

Anche in questo caso, come nel caso del tag <a>, il tag necessita degli attributi per funzionare. Fortunatamente il significato di tali attributi è abbastanza intuitivo.

L'attributo "src" indica il percorso in cui si trova l'immagine.

Gli attributi "width" ed "height" indicano rispettivamente altezza e larghezza dell'immagine.

L'attributo "alt" rappresenta il testo alternativo obbligatorio da mostrare nel caso in cui non venisse caricata l'immagine o durante lo stazionamento del puntatore del mouse sull'immagine. Ricordiamoci della particolarità di questo tag: manca il tag di chiusura! Ci si aspetterebbe una chiusura del tipo </img> invece la chiusura è eseguita tramite />.

Vediamo un esempio di utilizzo.

```
<html>
    <body>
          <p>Per inserire un'immagine all'interno della nostra
                pagina web, ci basta usare il tag img</p>
          <img src="w3c.jpg" width="363" height="362"
                alt="immagine" />
    </body>
</html>
```

Ed ecco cosa risulta nel browser.

Poniamo attenzione alla scritta "immagine" che compare sul www centrale. Quel testo è esattamente quello che abbiamo inserito nell'attributo "alt" e compare se fermiamo il puntatore del mouse sull'immagine.

<table>

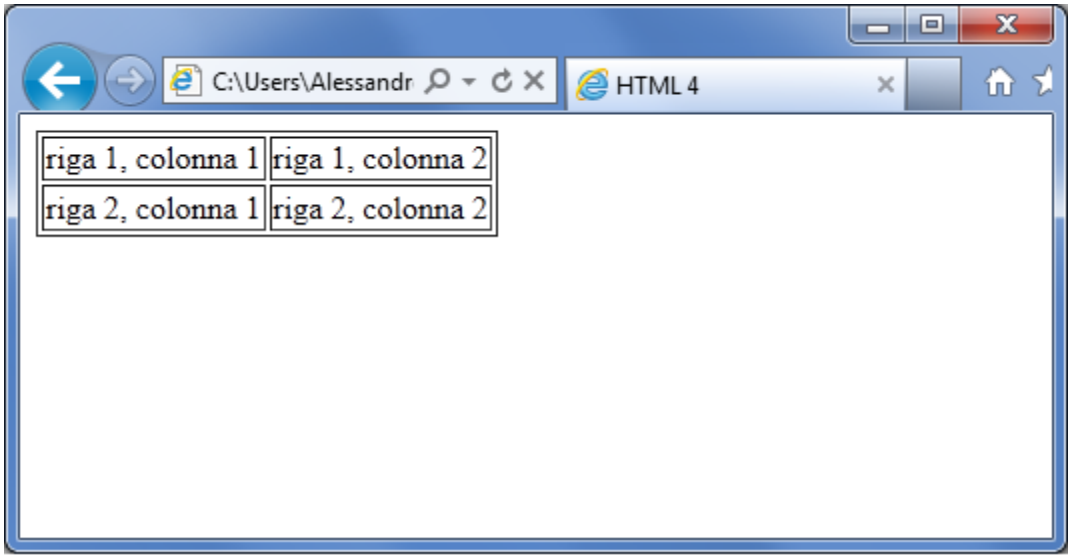
Le tabelle sono definite con il tag **<table>**.

Una tabella viene divisa in righe **<tr>** e colonne **<td>**. Vediamone subito un esempio:

```
<table border="1">
    <tr>
          <td>riga 1, colonna 1</td>
          <td>riga 1, colonna 2</td>
```

```
    </tr>
    <tr>
          <td>riga 2, colonna 1</td>
          <td>riga 2, colonna 2</td>
    </tr>
</table>
```

Ecco come viene trasformato dal browser questo codice.



Nel codice precedente, il tag <table> mostra un attributo: "border". A cosa serve? L'attributo "border" indica lo spessore del bordo esterno della tabella. Tuttavia se lo eliminassimo non vedremmo neanche i bordi interni tra righe e colonne.
Border non è certo l'unico attributo che si può usare con il tag <table>. I più utili e i più usati sono i seguenti:

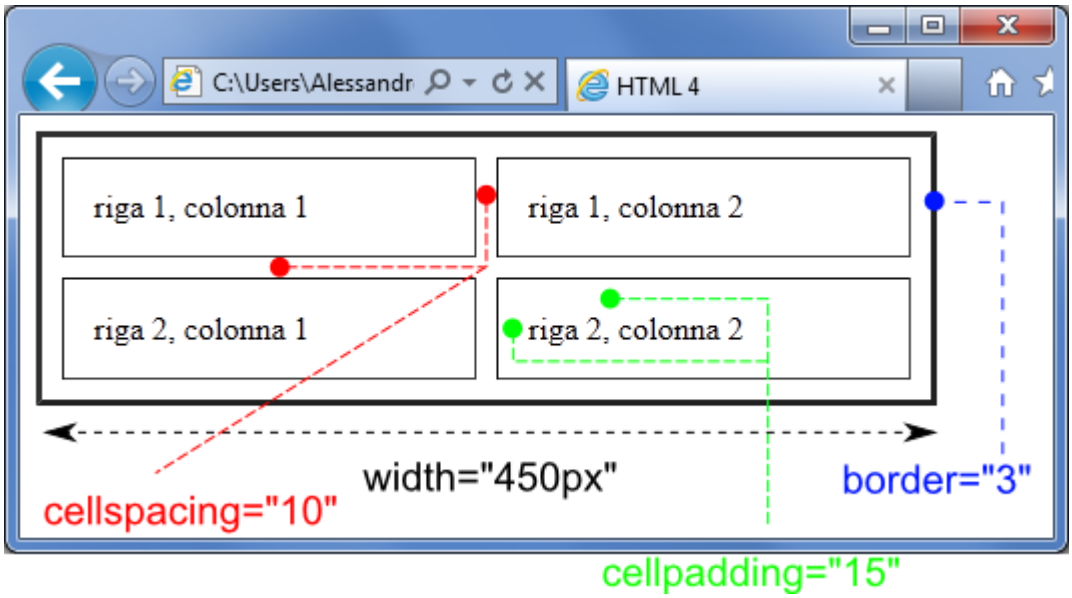
- cellpadding: specifica lo spazio che deve esserci tra il testo contenuto in una cella e i bordi della cella
- cellspacing: specifica lo spazio che deve esserci tra le celle
- width: specifica la larghezza della tabella

Facciamo subito un esempio completo in modo da comprendere meglio il significato di questi attributi.
Usando tutti gli attributi possiamo scrivere il seguente codice HTML.

```
<table border="3" cellpadding="15" cellspacing="10"
          width="450px">
```

```
    <tr>
          <td>riga 1, colonna 1</td>
          <td>riga 1, colonna 2</td>
    </tr>
    <tr>
          <td>riga 2, colonna 1</td>
          <td>riga 2, colonna 2</td>
    </tr>
</table>
```

Nella figura che segue, possiamo osservare come viene interpretato il codice dal browser.



Possiamo vederlo in azione qui:

http://www.alessandrostella.it/lato_client/html4/html4_02.html

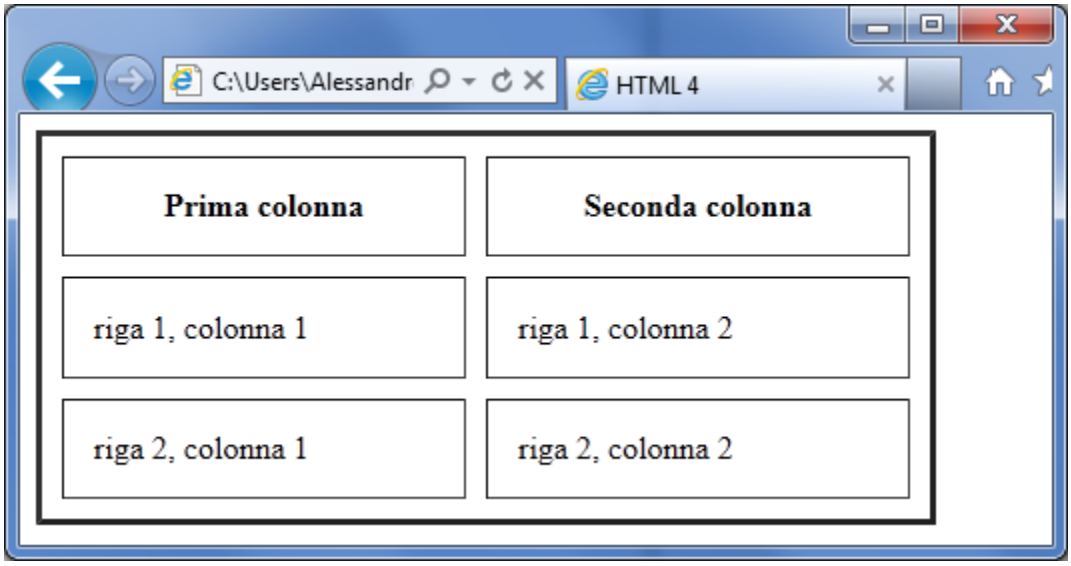
Ma non abbiamo ancora finito con il tag <table>.

Quasi sempre la prima riga di una tabella mostra un testo con la descrizione del contenuto delle relative colonne. Per identificare tale diversità esiste il tag **<th>**, da usare al posto del tag <td> per la rima riga. Tale tag si occuperà di evidenziare in grassetto il testo al suo interno, come possiamo osservare nel seguente esempio.

```
<table border="3" cellpadding="15" cellspacing="10"
          width="450px">
    <tr>
          <th>Prima colonna</th>
          <th>Seconda colonna</th>
```

```
    </tr>
    <tr>
          <td>riga 1, colonna 1</td>
          <td>riga 1, colonna 2</td>
    </tr>
    <tr>
          <td>riga 2, colonna 1</td>
          <td>riga 2, colonna 2</td>
    </tr>
</table>
```

Che conduce al risultato mostrato in figura.



Visibile in azione qui:

http://www.alessandrostella.it/lato_client/html4/html4_03.html

, <ol>,

Ovviamente in HTML non potevano mancare gli elenchi puntati e numerati. Per ottenerli HTML utilizza due tag: **<ul>** e **<ol>** che stanno rispettivamente per **u**nordered **l**inks e **o**rdered **l**inks. Il primo crea elenchi puntati, il secondo elenchi numerati. Ogni elemento dell'elenco viene indicato dal tag **<li>**

Esempio:

```
<ul>
    <li>comprare il pane</li>
    <li>comprare il latte</li>
```

```
</ul>
```

rappresenta un elenco puntato e viene disegnato dal browser così

- comprare il pane
- comprare il latte

Questo è invece un elenco numerato:

```
<ol>
    <li>comprare il pane</li>
    <li>comprare il latte</li>
</ol>
```

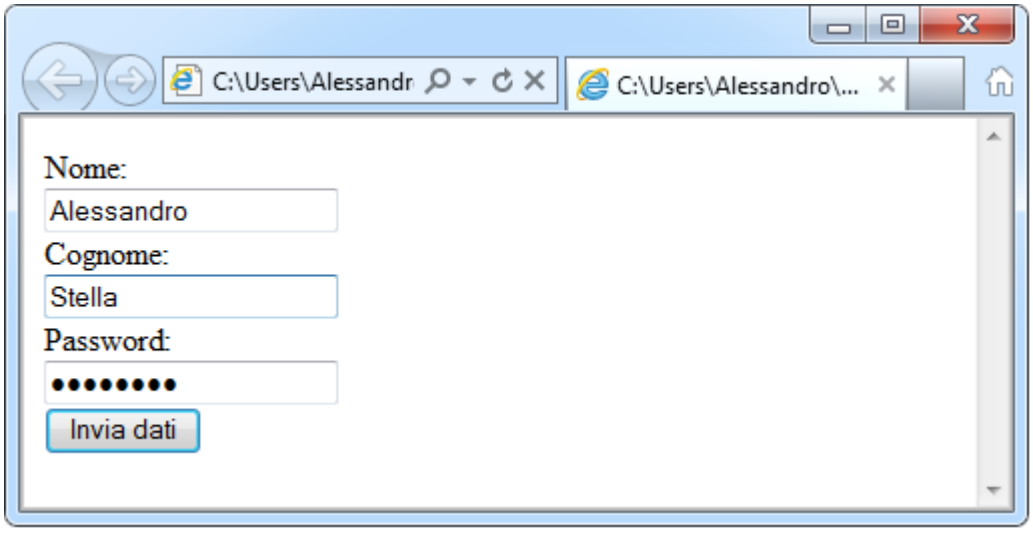
che viene quindi rappresentato dal browser così:

1. comprare il pane
2. comprare il latte

E' possibile annidare gli elenchi, cioè avere un nuovo tag <ol> (o <ul>) all'interno di un tag <li>.

<form> e i suoi figli

Un form HTML viene usato per consentire all'utente di inserire dati e passarli al server. Ecco qui di seguito un semplice form.



Questo è il codice necessario per produrlo.

```
<html>
    <body>
          <form>
                Nome:<br />
                <input type="text" name="nome" /><br />
                Cognome:<br />
                <input type="text" name="cognome" /><br />
                Password:<br />
                <input type="password" name="pwd" /><br />
                <input type="submit" value="Invia dati" />
          </form>
    </body>
</html>
```

e questo il link dove lo si può vedere in azione:
http://www.alessandrostella.it/lato_client/html4/html4_04.html

Come si può notare dal codice, il tag da usare per iniziare a definire un form è il tag **<form>**. Tipicamente poi il tag <form> contiene altri tag che rappresentano gli oggetti veri e propri in cui l'utente inserisce i dati. Eccone alcuni.

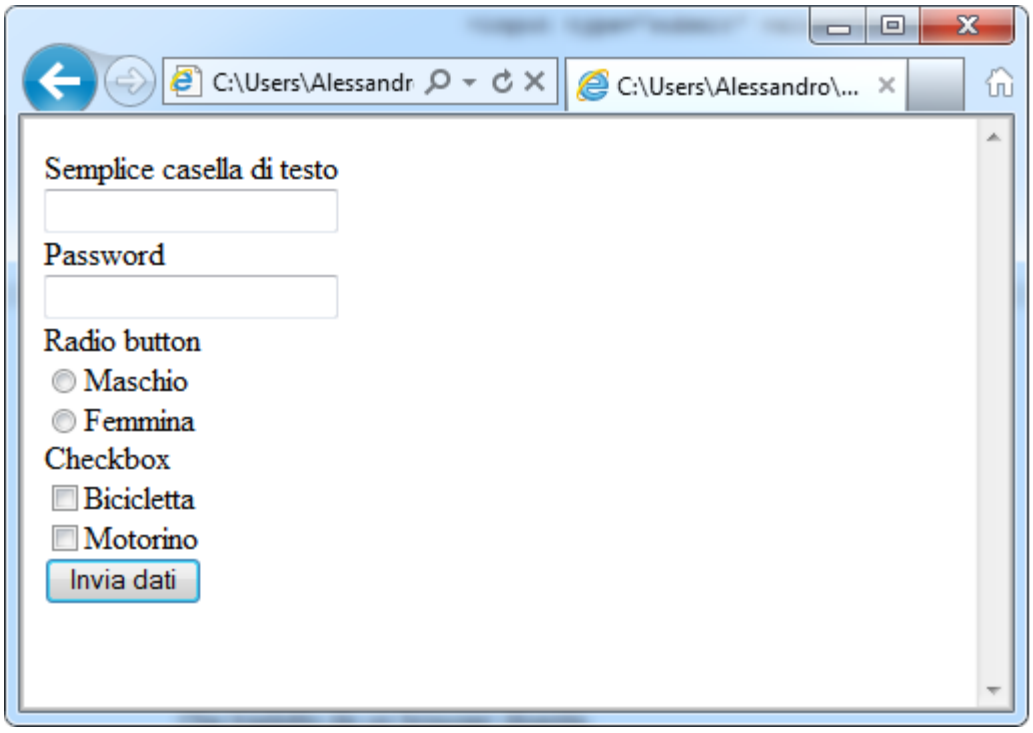
Tag	**Significato**
<input />	Rappresenta un oggetto in cui l'utente può inserire dati. Tramite l'attributo "type" può assumere diverse forme: casella di testo, bottone, checkbox, ecc. Notiamo che trattasi di elemento vuoto. Infatti termina con **/>**.
<textarea>	Rappresenta un'area di testo multiriga
<label>	Rappresenta un'etichetta
<select>	Rappresenta una lista di tipo drop-down
<optgroup>	Si usa all'interno del tag <select> e rappresenta un raggruppamento di possibili scelte
<option>	Si usa all'interno del tag <select> e rappresenta una tra le possibili scelte
<button>	Rappresenta un bottone, ma di un tipo diverso da quello mostrato nel form iniziale e con usi diversi

E' opportuno osservare che tutti i tag elencati in tabella possono essere usati anche al di fuori di un tag <form>.

Vale la pena dilungarsi sul tag <input /> perché è il più usato e il più eterogeneo. Vediamone subito tutti i suoi possibili usi.

```
<html>
    <body>
          <form>
                Semplice casella di testo<br />
                <input type="text" name="testo" /><br />
                Password<br />
                <input type="password" name="pwd" /><br />
                Radio button<br />
                <input type="radio" name="sesso"
                      value="maschio" />Maschio<br />
                <input type="radio" name="sesso"
                      value="femmina" />Femmina<br />
                Checkbox<br />
                <input type="checkbox" name="veicolo"
                      value="bicicletta" />Bicicletta<br />
                <input type="checkbox" name="veicolo"
                      value="motorino" />Motorino<br />
                <input type="submit" value="Invia dati" />
          </form>
    </body>
</html>
```

Che tradotto da un browser diventa.



Possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/html4/html4_05.html

Ci sono alcune cose da chiarire.
L'**attributo type** è colui che decide il tipo di input che ci aspettiamo dall'utente e può assumere uno dei seguenti valori:

- button
- checkbox
- file
- hidden
- image
- password
- radio
- reset
- submit
- text

Nel nostro esempio ne abbiamo visti solo alcuni. Più avanti nel testo, quando inizieremo a scrivere codice serio, avremo modo di approfondire i valori di questo fondamentale attributo.

L'**attributo name** nel caso dei radio button è molto importante. Osservando l'esempio, possiamo notare che esso coincide in entrambi i radio button, ossia in entrambi i casi il suo valore è "sesso". Questo è necessario affinché, selezionando un radio button, il sistema deselezioni automaticamente l'altro.

<script>

Gli scripts in HTML rappresentano riferimenti a codice (in genere javascript) che può essere eseguito dallo stesso browser senza nulla richiedere al server. Il tag predisposto per questo compito è il tag **<script>.** Abbiamo già incontrato questo tag quando abbiamo parlato del tag <head>. Può contenere sia codice vero e proprio sia fare riferimento, tramite l'attributo src, a codice esterno al documento HTML. In entrambi i casi viene usato l'attributo type per indicare al browser il tipo di script che troverà all'interno del tag o nel file esterno indicato dall'attributo src.
Gli usi comuni che possono essere fatti di javascript sono la manipolazione delle immagini, la validazione dei form e il cambiamento dinamico dei contenuti della pagina. Nell'esempio che segue, il tag <script> contiene codice javascript. Se eseguito, il codice javascript ha come risultato quello di scrivere "Hello World!" sul documento HTML:

```
<script type="text/javascript">
    document.write("Hello World!")
</script>
```

Ovviamente non è questo il momento in cui parlare di javascript. Lo faremo più avanti. Tuttavia ecco un esempio completo in cui viene messo in pratica quanto appena affermato.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
          <meta http-equiv="content-type"
                content="text/html; charset=UTF-8">
          <title>HTML 4</title>
          <script type="text/javascript">
                function msg() {
                      document.write("Hello World!");
                }
          </script>
    </head>
```

```
    <body>
          <a href="javascript:void(0)" onclick="msg()">Clicca
                qui per vedere il messaggio</a>
    </body>
</html>
```

E qui è possibile vederlo in azione:
http://www.alessandrostella.it/lato_client/html4/html4_06.html

Ricordiamoci del tag <script>.

<style>

I fogli di stile associati ad un documento HTML (css) possono essere integrati direttamente nel documento o richiamati da file esterni. Nel primo caso si usa il tag **<style>** come mostrato qui di seguito.

```
<head>
    <style type="text/css">
          body {background-color:yellow;}
          p {color:blue;}
    </style>
</head>
```

Nel secondo caso invece vengono richiamati usando il tag **<link>**, tag che abbiamo già incontrato quando abbiamo parlato del tag <head>. Ecco un esempio di questo secondo caso:

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

Avremo modo più avanti di approfondire cosa sono e come si usano i fogli di stile o css (**c**ascading **s**tyle **s**heets). Per adesso però memorizziamo il fatto che per usare i fogli di stile potremmo aver bisogno del tag <style>.

Gli elementi HTML

Un elemento HTML è definito da tutto ciò che si trova tra l'apertura di un tag e la rispettiva chiusura. Facciamo attenzione a non confondere il concetto di tag con quello di elemento.
Questo è un tag:

Questo invece è un elemento:
Testo contenuto nel paragrafo

Il **contenuto di un elemento** (element content) è tutto ciò che si trova tra il tag di apertura e quello di chiusura. Nel nostro esempio il contenuto è rappresentato dal testo "Testo contenuto nel paragrafo". Quindi parlare del tag significa riferirsi al solo tag, mentre parlare dell'elemento significa parlare di tutto ciò che è compreso tra e , oltre che a stesso.

Gli attributi HTML

Abbiamo già incontrato gli attributi, ma senza meglio identificarli. Ora è tempo di conoscerli un po' meglio. Lo scopo degli attributi è tanto semplice quanto fondamentale: **aggiungere dettagli agli elementi HTML**.
Esempio
Alessandro Stella

Abbiamo già visto che il tag <a> ha come compito quello di indicare un link. Ma per creare un link servono molte informazioni. Informazioni che da solo il tag <a> non è in grado di fornire. Ad esempio bisogna indicare dove punta il link, oppure se aprirlo nella stessa pagina o in un'altra. Per aggiungere tutte queste informazioni vengono usati gli attributi. Nel nostro esempio viene usato l'attributo "href" per indicare dove punta il link. Tutti gli attributi devono avere un valore. Il valore viene indicato tra doppi apici. Nell'esempio il valore è "www.alessandrostella.it/". Per assegnare tale valore all'attributo si usa il segno di uguaglianza "=". Ogni tag consente di usare alcuni attributi e questo significa che gli attributi sono tantissimi. Ecco perché per una lista completa degli attributi consentiti per ogni singolo tag, si rimanda al sito del w3c. Noi, nel prosieguo dello studio ne incontreremo e useremo tanti e di tutti spiegheremo significato e utilizzo.
Anche per gli attributi vale la raccomandazione w3c: **scriviamoli in minuscolo!**

I commenti

In HTML è anche possibile inserire dei commenti che non vengono gestiti dal browser ma che possono esserci molto utili. Per inserire un commento nel codice HTML bisogna inserire il commento tra <!-- e -->

Esempio:

<!-- Questo è un commento -->

Un commento può essere inserito in un qualunque punto del codice e non viene tradotto né visualizzato in alcun modo dal browser.

Elementi block ed elementi inline

Tutti gli elementi HTML possono essere **raggruppati in 3 grandi categorie**:

- block
- inline
- notdisplayed

Appartengono alla **categoria "block"** gli elementi che si espandono per tutta la larghezza del documento HTML e costringono ad andare a capo. Fanno parte di tale categoria gli elementi <p>, , <h1>...<h6>, <blockquote>, <ul>, <ol>, <dl>, <li>, <dt>, <dd>, <table>, <pre>, <form>. Noi non conosciamo tutti questi tag, ma non importa.

Appartengono alla **categoria "inline"** tutti gli elementi HTML che invece occupano solo lo spazio nel quale vengono utilizzati e non impongono di andare a capo. Fanno parte di tale categoria gli elementi , <a>, , , <img />,
, <input>.

Appartengono infine alla **categoria "not displayed"** tutti quegli elementi che non hanno un impatto visivo nella pagina HTML. Ad esempio i tag <head>, <meta>, <style>.

Ma cosa significa esattamente tutto ciò?

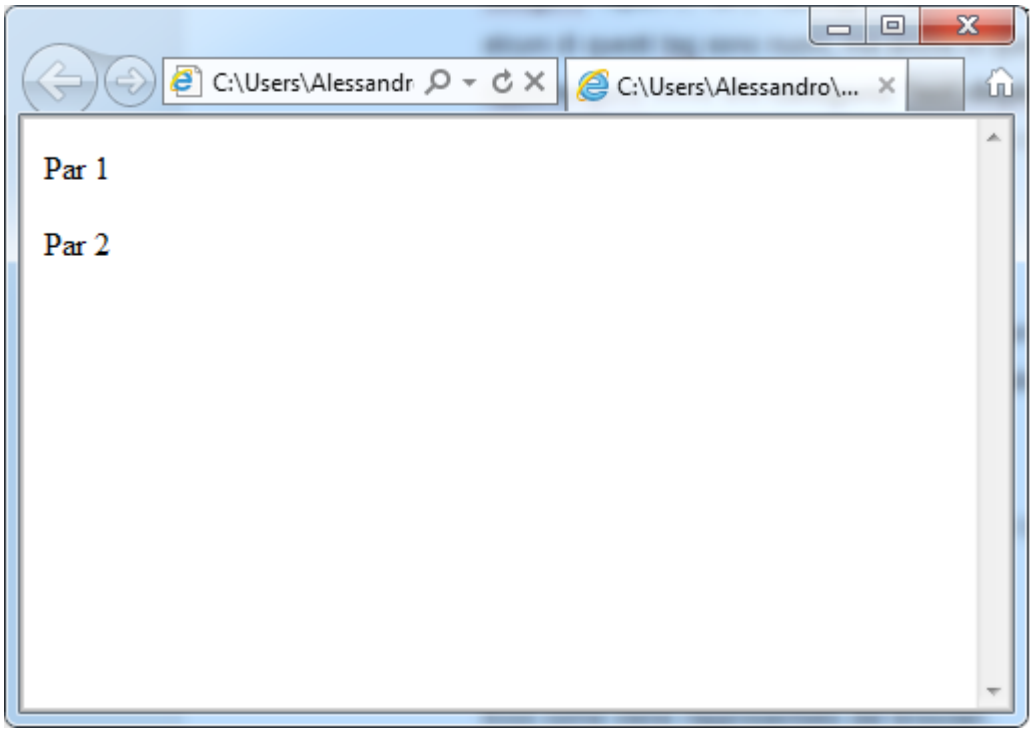
Mano al codice e tutto sarà chiaro!

Abbiamo appena detto che <p> appartiene alla categoria "block" e abbiamo detto che gli elementi appartenenti a questa categoria occupano l'intera larghezza del documento HTML e impongono un ritorno a capo. Scriviamo allora il seguente codice.

```
<html>
    <body>
        <p>Par 1</p><p>Par 2</p>
    </body>
```

</html>

e vediamo come viene rappresentato dal browser.

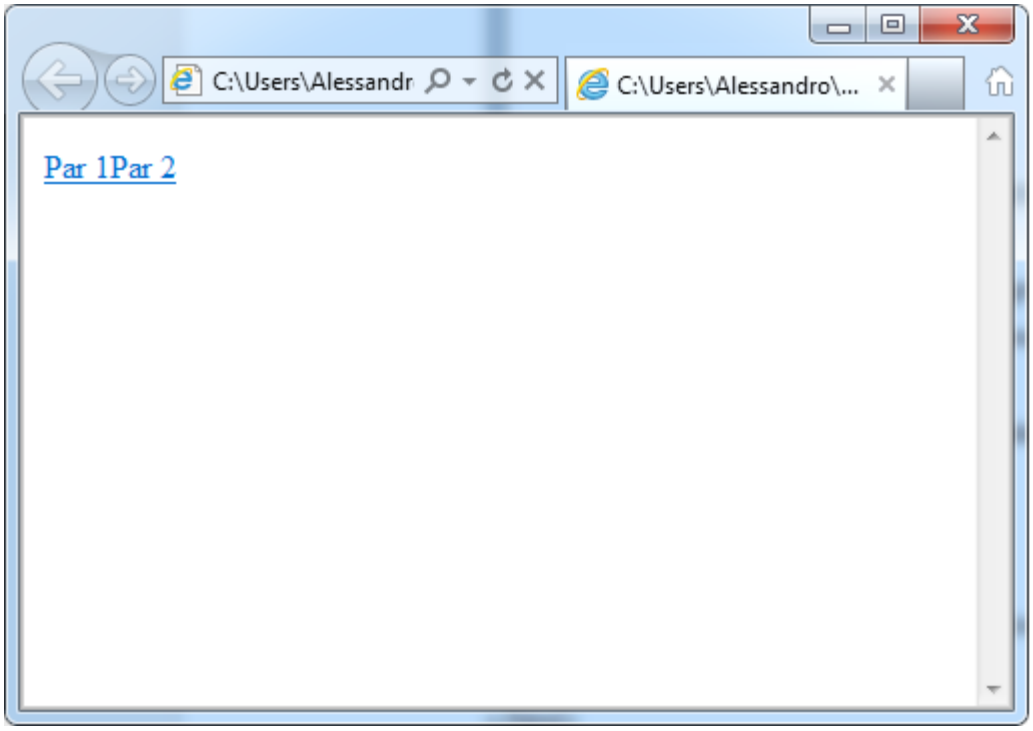


Sebbene nel codice abbiamo posizionato i 2 paragrafi uno di fianco all'altro, quando il browser disegna la corrispondente pagina web i paragrafi si dispongono uno sotto l'altro! Ecco quindi cosa significa essere di tipo "block": occupare l'intera larghezza del documento HTML e costringere un ritorno a capo. Ed ecco perché <p> appartiene alla categoria degli elementi "block": essi occupano tutta la riga e costringono ad andare a capo.

Un elemento "inline" invece non occupa tutta la riga in cui è inserito e non costringe ad andare a capo. Conosciamo il tag <a> e abbiamo appena imparato che esso appartiene alla categoria inline. Facciamo una prova con tale tag.

```
<html>
    <body>
          <a href="par1.html">Par 1</a>
          <a href="par2.html">Par 2</a>
    </body>
</html>
```

e vediamo come viene rappresentato dal browser.



Sebbene nel codice HTML abbiamo inserito i due tag <a> su 2 righe diverse, quando il browser disegna la relativa pagina essi vengono disposti uno di fianco all'altro: nessun ritorno a capo e nessuno spazio tra i due elementi. Ciò è la naturale conseguenza del fatto che entrambi gli elementi sono di tipo "inline".
Bene, ora dovrebbe essere chiaro cosa si intende per elementi "block" ed elementi "inline".

Gli elementi replaced

Un'altra distinzione da ricordare è quella tra elementi rimpiazzati (replaced) ed elementi non rimpiazzati. I primi sono elementi di cui il browser conosce solo lo spazio che occuperanno nella pagina web, ma che non contengono direttamente il contenuto.
L'esempio più tipico di elemento rimpiazzato è <img> (tag immagine). Tale elemento si presenta spesso così:

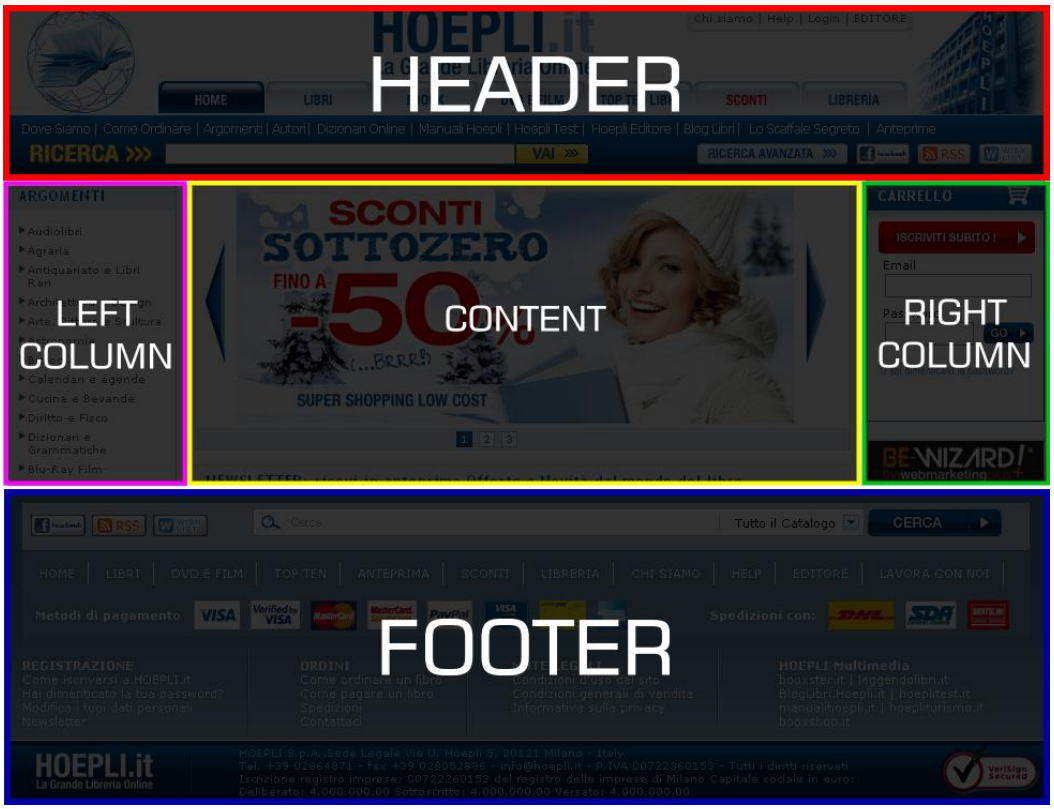
Il browser quindi è consapevole che in quel punto dovrà essere inserita un'immagine con larghezza 450px e altezza 300px, ma l'immagine non è direttamente contenuta nell'elemento. Anzi l'elemento è addirittura vuoto! Tuttavia l'attributo src indica al browser dove andare a prendere l'immagine da mettere in quel punto.

Altri elementi di tipo replaced sono: <input>, <textarea>, <select> e <object>. Tutti gli altri elementi sono in genere considerati non rimpiazzati.

La distinzione è importante perché 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.

HTML Layout

Questo è un argomento estremamente delicato.
Ecco cosa ci dice il w3c in merito: "Web page layout is very important to make your website look good. Design your webpage layout very carefully."
Come sempre il w3c non tradisce ed esprime, con poche parole, tutta l'importanza del layout in una pagina web: bisogna progettare con estrema cura il layout delle proprie pagine web.
Si, ok, ma cosa significa layout?
Con il termine layout si intende la disposizione visiva all'interno della pagina web dei vari elementi che compongono la pagina.
E' una definizione molto generica che potrebbe non essere troppo chiara. Quindi riportiamo subito nella pratica ciò di cui stiamo parlando, mostrando visivamente cosa si intende per layout di una pagina web.
Ci sono alcuni layout che sono particolarmente diffusi nel web, anzi sono così tanto diffusi che, come vedremo più avanti, HTML5 ha creato appositi tag per rappresentarli nativamente. Ma restiamo al concetto di layout. Ecco qui di seguito un classico layout definito come 3 colonne centrate. E' il sito della www.hoepli.it.



Osservando i rettangoli colorati possiamo comprendere meglio il concetto di layout. La pagina web è divisa in 5 sezioni: in rosso troviamo la sezione header, in fuxia la sezione left column, in giallo la sezione content e così via. Ecco, costruire il layout di una pagina web significa decidere quali sezioni creare, dove disporle e cosa inserisci all'interno.
I nomi usati per descrivere tali sezioni (header, footer, ecc.) sono quelli di uso comune e, chiaramente, sono in inglese. Nulla vieta di usare altri nomi. Tradotto in HTML un layout del genere potrebbe essere riprodotto come segue.

```
<html>
    <body>
          <div id="header"></div>
          <div id="left"></div>
          <div id="content"></div>
          <div id="right"></div>
          <div id="footer"></div>
    </body>
</html>
```

Nella realtà questo codice non funziona, perché non produce il layout voluto, ma rende l'idea di come si traduce un layout in codice HTML.

Bene.
Ora che sappiamo cosa si intende per layout, proviamo a comprendere come lo si realizza. Esiste un tag molto particolare, pensato appositamente per questo scopo, è il tag ** ** che indica appunto una "division" della pagina, cioè una sezione, un pezzo, un riquadro di una pagina web. Lo abbiamo visto nel codice sopra riportato in cui abbiamo usato un per ogni sezione della pagina.
E' opportuno osservare che un qualsiasi layout può essere ottenuto anche con il tag **<table>**. Possiamo infatti creare una tabella con righe e colonne tali da riprodurre qualsiasi layout; funzionerebbe tutto alla perfezione. E allora perché esiste il tag ? Sembra inutile!
Anche in questo caso è il w3c che ci risponde: "Even though it is possible to create nice layouts with HTML tables, tables were designed for presenting tabular data - NOT as a layout tool!". Ossia, sebbene sia possibile creare un buon layout usando le tabelle HTML, le tabelle sono state progettate per presentare dati - NON come strumento di layout!
Quindi è bene prendere una sana abitudine: **usare il tag <table> solo ed esclusivamente quando dobbiamo presentare dati tabellari**. Per esempio, un elenco di persone mostrate su 3 colonne (codice fiscale, nome e cognome) è un classico esempio in cui è ragionevole usare una <table>.

Codice fiscale	**Cognome**	**Nome**
AAABBB44F44F123S	Stella	Alessandro
TTTSSS23E45E333D	Cappuccio	Ciccio
SSSRRR55F55F454V	Cupeta	Uccio

Quindi memorizziamo bene questo concetto: **per costruire il layout di una pagina web dobbiamo usare il tag **.
Purtroppo (o per fortuna) il tag porta con sé una conseguenza: **non è possibile usare proficuamente il tag senza usare i fogli di stile**. Tale sua caratteristica ci spinge a fare una importante considerazione: il tag si trova **a cavallo tra il cosa e il come**. Ricordiamo il capitolo 2, vero? Abbiamo imparato che il cosa mostrare su una pagina web viene deciso dal codice HTML, mentre il come lo decidono i css. Questo tag HTML tuttavia sembra avere poco a che fare con il cosa. Sembra invece molto più legato al come. In effetti è proprio così. E' così perché è un tag

pensato esclusivamente per il layout e il layout è un concetto che è legato al come più che al cosa. Insomma il tag è una sorta di tag di confine tra il cosa e il come...

HTML DOM

DOM = **D**ocument **O**bject **M**odel.

Il DOM HTML è una cosa informe e molto antipatica, ma di fondamentale importanza! Ne avremo bisogno assoluto quando studieremo javascript e ajax.

Il DOM è un modo per rappresentare un documento HTML tramite oggetti. La rappresentazione avviene usando una struttura ad albero, composta da nodi, in cui ogni elemento HTML è un nodo e ogni nodo è un oggetto. La logica usata nella costruzione dell'albero è esattamente la stessa che usiamo per costruire un albero genealogico: padre-figlio. Abbiamo infatti detto che gli elementi possono essere annidati in altri elementi. Ad esempio:

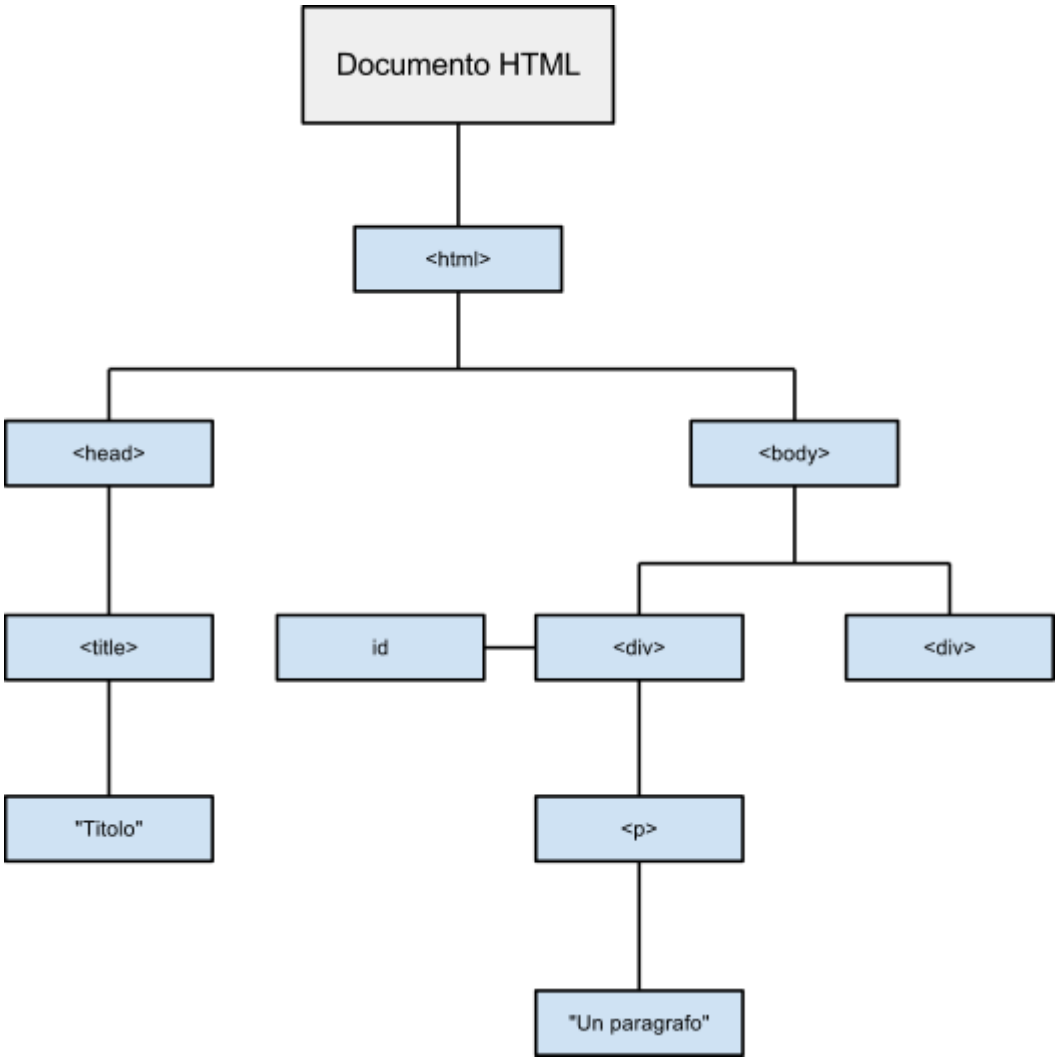
```
<body>
    <div id="primo">
          <p>Un paragrafo</p>
    </div>
</body>
```

Guardando questo codice non ci è difficile dire che l'elemento <p> è figlio dell'elemento e che l'elemento è figlio dell'elemento <body>. L'elemento <body> invece (in questo caso) non ha nessun padre: è l'origine di tutto, il primo della stirpe, quello che in gergo informatico viene chiamato root.

Detto questo non dovrebbe essere difficile comprendere il perché il seguente documento HTML

```
<html>
    <head>
          <title>Titolo</title>
    </head>
    <body>
          <div id="primo">
                <p>Un paragrafo</p>
          </div>
          <div></div>
    </body>
</html>
```

venga rappresentato dal seguente modello DOM.



Questa figura non è altro che la rappresentazione dello stesso documento HTML di prima, ma visto in ottica DOM. Il documento HTML inizia con l'elemento <html> che è il padre di tutti gli elementi del documento; infatti <html> non è contenuto in nessun altro elemento, è il primo in un qualsiasi documento HTML. L'elemento <html> ha poi due figli: <head> e <body> (che tra loro sono quindi fratelli e per questo posti sullo stesso livello nel disegno DOM). L'elemento <head> a sua volta ha un figlio, l'elemento <title>. A questo punto c'è la prima "stranezza". Infatti il contenuto dell'elemento <title> risulta essere un figlio dello stesso elemento ed è anch'esso mostrato in un rettangolo come fosse un elemento! Questo modo di rappresentare il documento HTML generalmente lascia perplessi durante il primo approccio con il DOM. In realtà vedremo più avanti che nel DOM HTML ogni rettangolo si chiama nodo e nel DOM tutto è un nodo. Lo stesso

discorso fatto con <title> vale inalterato per l'elemento <p> che risulta avere come figlio il testo in esso contenuto. Capiremo questo modo di interpretare il documento nel prossimo paragrafo.
L'elemento <html>, cioè il padre di tutti gli elementi di un documento HTML, viene chiamato **elemento root**. Tutto il documento HTML (compreso quindi l'elemento <html>) viene invece rappresentato tramite l'**oggetto document** (nella figura è il primo quadrato grigio in alto).

Perché è utile conoscere la rappresentazione DOM di un documento HTML?
Perché questo modo di vedere un documento HTML ci consente di accedere ad ogni elemento HTML, ai suoi attributi e al suo contenuto come se ogni cosa fosse un oggetto. Quando studieremo javascript useremo proprio il DOM per modificare in tempo reale i singoli oggetti (e quindi lo stesso documento HTML) senza né ricaricare la pagina né interrogare il server.

I nodi

Il concetto di nodo nel DOM è molto importante. All'interno della rappresentazione DOM, tutto è un nodo. Ossia:

- L'intero documento HTML è un nodo (l'oggetto document)
- Ogni elemento HTML è un nodo
- Il testo contenuto in un elemento HTML è un nodo
- Ogni attributo di ogni elemento è un nodo
- Anche i commenti sono nodi

Ecco dunque spiegato il motivo per cui nel paragrafo precedente l'elemento <p> risultava avere come figlio il testo "Un paragrafo". **Nel DOM tutto è un nodo**!
Questo concetto deve essere chiaro. Infatti uno degli errori più comuni durante la navigazione all'interno del DOM è quella di vedere, ad esempio, il testo "Titolo" come fosse il valore dell'elemento <title>. Non è così! "Titolo" è un nodo, non è il valore del nodo <title>. Teniamo ben chiaro questo concetto.

Come detto, la navigazione tra i nodi avviene tramite legami di parentela: padre, figlio, fratello.
Per recuperare o modificare il contenuto di un nodo possiamo usare **la proprietà innerHTML** (una delle proprietà di un nodo che vedremo nel prossimo paragrafo). Ad esempio, con riferimento al documento HTML di inizio paragrafo, il codice
var contenuto = document.getElementById("primo").innerHTML;

assegnerà alla variabile "contenuto" il valore " Un paragrafo ".
Allo stesso modo il codice
document.getElementById("primo").innerHTML="<h3>Cambio contenuto<\/h3>";

modificherebbe il contenuto del div con id="primo" con quello alla destra del simbolo di uguaglianza.
Non sarà di certo sfuggito l'apparente errore in chiusura del tag <h3>, ossia il doppio slash (<\ /h3>). Ebbene, non è un errore!
Quando si inseriscono stringhe all'interno di codice javascript che contengono tag html è opportuno inserire un backslash (\) prima di ogni slash al fine evitare che la stringa venga interpretata dal browser come elemento HTML. Questa è una chicca che conoscono in pochi perché il browser, in genere, anche in assenza del backslash non dice niente e il codice funziona perfettamente, ma... se noi proviamo a far validare dal validatore del W3C il codice senza il backslash... rosso! Niente convalida. Almeno fino a quando non inseriamo il backslash!
Avremo modo di approfondire la proprietà innerHTML quando useremo javascript. Per il momento possiamo vedere il codice precedente in azione qui:
http://www.alessandrostella.it/lato_client/html4/html4_07.html

Nell'esempio troveremo un paragrafo e due link che richiameranno due funzioni javascript. Cliccando sul primo link useremo javascript per leggere il contenuto dell'elemento #primo (che è il paragrafo). Cliccando sul secondo link invece useremo javascript per modificare il contenuto dello stesso elemento trasformandolo da paragrafo a <h3>.
Ciò che può fare javascript è molto di più, ma in questo momento non ha alcun senso andare oltre.

Proprietà e metodi

Ogni nodo nel modello DOM ha proprietà e metodi.
Una proprietà può essere vista come una caratteristica del nodo, per esempio il nome. Un metodo può essere visto come un meccanismo per modificare le proprietà del nodo stesso.
Supponendo che n sia un generico nodo nel DOM, qui di seguito elenchiamo proprietà e metodi più usati.

Proprietà.

- n.innerHTML - restituisce tutto ciò che è contenuto in n
- n.nodeName - restituisce il nome di n
- n.nodeValue - restituisce il valore id n
- n.parentNode - restituisce il nodo padre di n
- n.childNodes - restituisce l'elenco di nodi figli diretti di n
- n.attributes - restituisce l'elenco degli attributi di n

Metodi

- n.getElementById(id) - restituisce l'elemento con uno specifico id
- n.getElementsByTagName(name) - restituisce l'elenco di tutti gli elementi del tipo indicato tra parentesi e contenuti nel nodo n
- n.appendChild(node) - inserisce il figlio indicato tra partentesi come figlio di n
- n.removeChild(node) - elimina il nodo figlio di n indicato tra parentesi

Adesso ne vediamo un semplice utilizzo, usando javascript. Per comprendere questo semplice esempio non è necessario conoscere javascript.

```
<html>
    <head>
          <title>Titolo della pagina</title>
    </head>
    <body>
          <p id="testo">Un testo qualunque</p>
          <script type="text/javascript">
                txt=document.getElementById("testo").innerHTML;
                document.write("<p>Il testo contenuto nel primo
                paragrafo è: " + txt + "<\/p>");
          </script>
    </body>
</html>
```

Possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/html4/html4_08.html

Il codice HTML crea un paragrafo <p> a cui assegna id="testo". Poi inizia lo script. Sono solo 2 righe. Analizziamolo riga per riga.
txt=document.getElementById("testo").innerHTML;
Viene creata la variabile txt a cui viene assegnato il valore di

document.getElementById("testo").innerHTML;
document, come abbiamo visto, rappresenta tutto il documento DOM ed è quindi un nodo. Essendo un nodo possiamo usare su di esso il metodo
getElementById()
che richiamato su id=”testo” restituirà, se esistente, il nodo con id=”testo”. Il risultato è quindi un altro nodo. Essendo il risultato un altro nodo possiamo su di esso usare la proprietà “innerHTML” che restituisce tutto il contenuto del nodo su cui viene richiamata. Nel nostro caso tale valore è “Un testo qualunque”. Tale valore viene quindi assegnato alla variabile txt per cui alla fine della prima riga abbiamo
txt = “Un testo qualunque”;
Poi lo script prosegue:
document.write(" Il testo contenuto nel primo paragrafo è: " + txt + "<\/p>");
Questo codice scrive, nel punto in cui viene richiamato, un paragrafo il cui contenuto sarà
Il testo contenuto nel primo paragrafo è: Un testo qualunque

Per adesso non è importante andare oltre.
Tuttavia teniamo bene a mente che conoscere il DOM è necessario per usare un linguaggio di scripting (come appunto javascript) e tutte le sue derivazioni, come per esempio ajax e jquery.

4
CSS 2

I fogli di stile.
Impossibile farne a meno. Essi rappresentano la chiave di volta che trasforma uno "strimpellatore del web" in un musicista del web. Su questo argomento sono stati scritti molti libri perché è un argomento tanto importante quanto delicato. Noi non avremo lo spazio di un intero libro, ma ciò nonostante ne dedicheremo molto ai css. Inoltre online troveremo tutto quello che ci serve per completare la conoscenza degli argomenti che, per forza di cose, qui non avremo lo spazio di trattare. A proposito dello studio online, è tempo di aprire una piccola, ma importantissima parantesi: non andiamo in giro per il web a "copiare/incollare" codice scritto da altri anche se funziona. Questo modo di fare (molto diffuso!) non ci renderà mai dei professionisti, ma al massimo degli ottimi copioni! Una volta imparate le basi e la logica di funzionamento di una tecnologia, per approfondirne l'uso in modo professionale c'è un solo modo: rivolgersi a chi quella tecnologia la cura e la distribuisce!
E chi cura HTML e CSS?
Il W3C!
Esso è e deve essere la nostra guida suprema, la luce nel buio, l'unica fonte di conoscenza reale! Ecco perché, anche nello studio dei css, ci guiderà sempre esso, il sommo, la guida eterea e sicura, la Legge del web: il W3C! Tutto ciò che in questo libro non sarà scritto è scritto lì e da lì dovremo impararlo!
Lo strumento di lavoro sarà sempre il fidato Notepad++. Quando creeremo un nuovo file di tipo css ci basterà eseguire la solita procedura "Language/C/CSS".
Ora iniziamo, lo stile ci aspetta!

Cosa sono?

CSS = **C**ascading **S**tyle **S**heet = Fogli di Stile a Cascata

Per capire cosa sono i fogli di stile dobbiamo ritornare per un momento al "cosa e al come". Durante tutto lo studio che abbiamo effettuato sull'**HTML**, abbiamo sempre ricordato che esso si occupa del **cosa mostrare** a video. Studiando i **CSS** invece impareremo **come mostrare** a video i vari elementi presenti nel nostro documento HTML. Si chiamano fogli di stile proprio per questo motivo: essi rappresentano lo stile con cui mostrare ogni singolo elemento presente nella pagina web. Il termine "a cascata" invece rappresenta la possibilità, di ogni stile, di essere sovrascritto a cascata da altri fogli di stile. Lo vedremo bene nei capitoli successivi.

Come si usano?

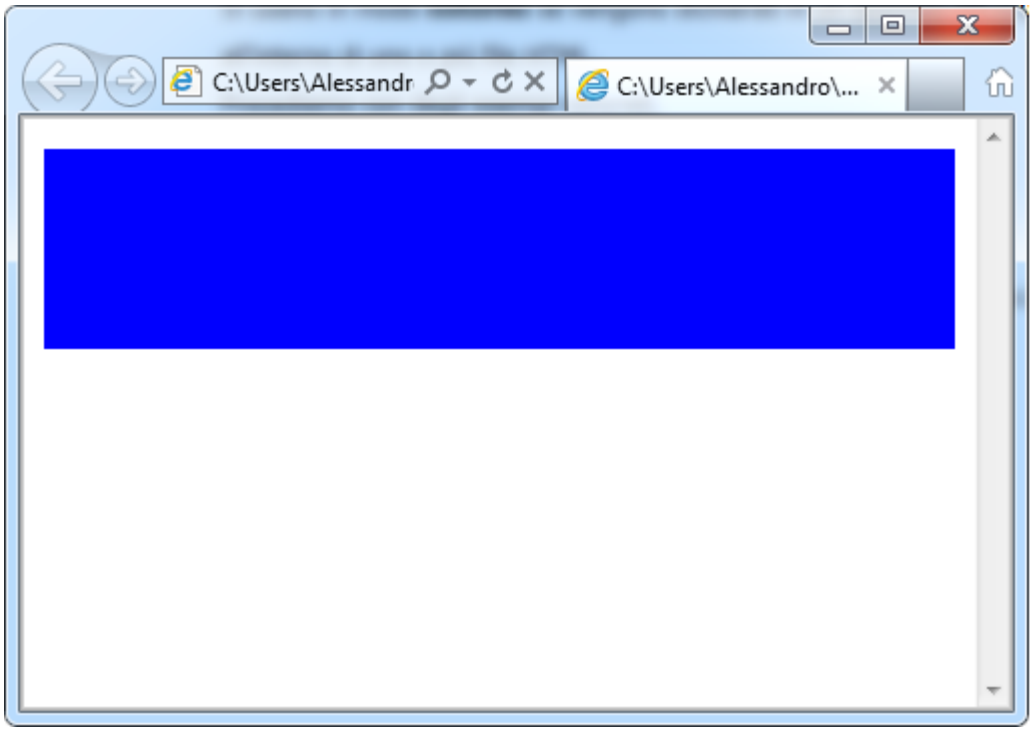
I CSS possono essere usati in 3 modi distinti:

1. **inline** (vengono dichiarati come attributo del tag HTML a cui si riferiscono)
2. **interno** (vengono dichiarati all'interno del singolo file HTML)
3. **esterno** (vengono dichiarati in un apposito file esterno al documento HTML e poi richiamati dal codice HTML)

Esempio di **codice css inline**.

```
<html>
    <body>
          <div style="height:100px;background-color:blue"></div>
    </body>
</html>
```

Che produce il seguente risultato.



Per utilizzare uno stile inline bisogna quindi usare l'attributo HTML style. Attributo che si può applicare a qualsiasi elemento HTML. Nel nostro esempio, il codice inline che colora lo sfondo è il seguente:
style="height:100px;background-color:blue"

E' dunque l'attributo "style" che si occupa di decidere lo stile del particolare elemento HTML al quale è assegnato. In questo caso assegna uno sfondo di colore blu e un'altezza di 100 pixel al tag .

Esempio di **codice css interno**.

```
<html>
    <head>
          <style type="text/css">
                div {
                      height:100px;
                      background-color:blue;
                }
          </style>
    </head>
    <body>
          <div></div>
```

```
    </body>
</html>
```

Questo codice produce esattamente lo stesso risultato del precedente (cioè il rettangolo alto 100px e con sfondo blu). Questa volta però lo stile viene dichiarato nell'elemento <head> del file HTML. Per farlo si è usato l'elemento **<style>** con l'attributo type ad indicare il tipo di stile (per noi sarà sempre "text/css").
Ovviamente la prima domanda che viene da porsi è: perché 2 modi diversi per ottenere lo stesso risultato?
La risposta è che in realtà i 2 modi non ottengono lo stesso risultato. Infatti lo stile interno che abbiamo scritto non viene applicato ad uno specifico , ma vale **per tutti gli elementi ** presenti nel documento, mentre nel caso dello stile inline il codice viene applicato solo all'elemento in cui viene scritto.

Esempio di **codice css esterno**.

```
<html>
    <head>
          <link rel="stylesheet" type="text/css"
                href="come.css" />
    </head>
    <body>
          <div></div>
    </body>
</html>
```

In questo caso non scriviamo più il codice css nel file HTML, ma lo scriviamo all'interno di un altro file (che nell'esempio abbiamo chiamato "come.css") e lo comunichiamo al browser tramite l'elemento **<link>** con attributi rel="stylesheet", type="text/css" e href che punta al file css (in questo caso "come.css").
All'interno del file "come.css" ci sarà il nostro codice css, ossia

```
div {
    height:100px;
    background-color:blue;
}
```

Il risultato sarà sempre lo stesso: un rettangolo di colore blu. Anche in questo caso il codice vale per tutti i tag presenti nel documento HTML.
Nuova ovvia domanda: a cosa serve mettere il codice CSS in un file esterno se otteniamo gli stessi risultati con il codice interno?
La risposta è nel numero e nel layout delle nostre pagine web.
In genere un sito web non è fatto da una sola pagina, giusto? E in genere buona parte di una pagina è identica a tutte le altre quindi usa lo stesso stile. Se non avessimo un foglio di stile esterno dovremmo ripetere lo stesso codice CSS all'interno di tutte le pagine. Mentre con il foglio di stile esterno basta ripetere in ogni pagina una sola istruzione tramite l'elemento <link>.

Bisogna osservare che **è possibile usare tutti e tre i sistemi di dichiarazione nello stesso momento**. Ma così facendo potrebbe capitare di scrivere stili diversi per lo stesso elemento! In tal caso cosa succede? Cioè se per il tag sul foglio di stile esterno indichiamo un colore di sfondo rosso, su quello interno indichiamo un colore verde e infine su quello inline indichiamo un colore blu, alla fine che colore avrà lo sfondo del tag ?
Sarà blu.
Se ci sono stili diversi associati allo stesso tag, la regola della cascata dice questo: **il codice css interno sovrascrive quello esterno e il codice inline li sovrascrive entrambi**.
Ora abbiamo anche capito perché si chiamano "a cascata", ma ci torneremo più avanti.

Possiamo osservare quanto appena affermato qui:
http://www.alessandrostella.it/lato_client/css2/css2_00.html

La proprietà display

Durante lo studio di HTML abbiamo visto che i suoi elementi possono essere raggruppati **in 3 grandi categorie**:

1. block
2. inline
3. not displayed

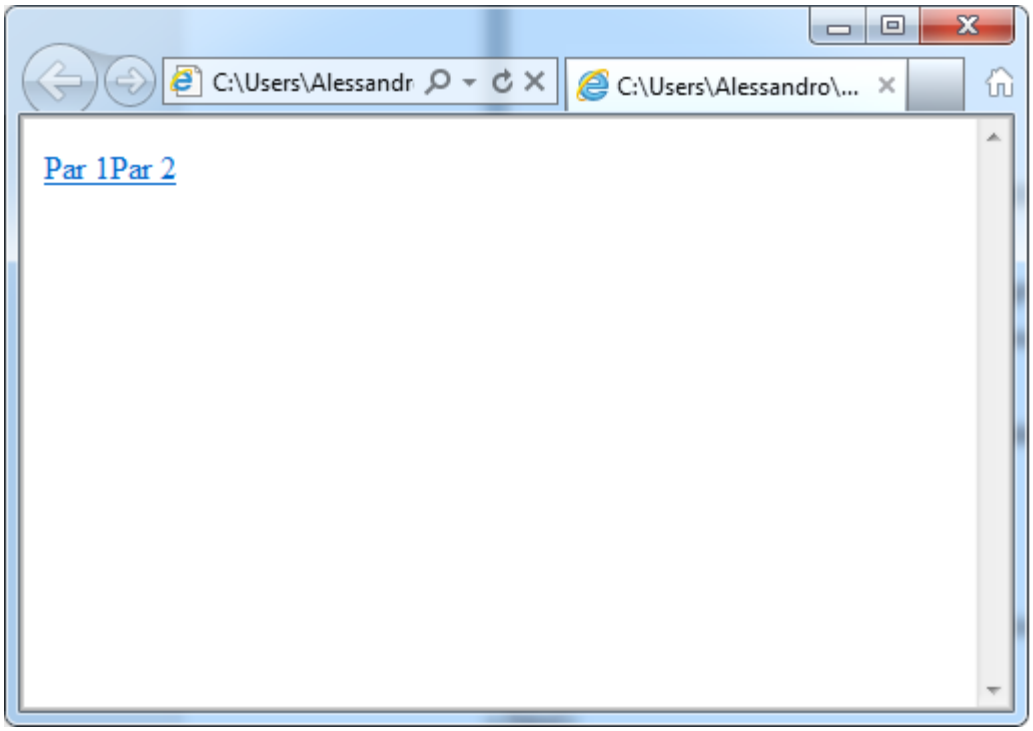
Appartengono alla **categoria "block"** gli elementi che si espandono per tutta la larghezza del documento HTML e costringono ad andare a capo.
Appartengono alla **categoria "inline"** tutti gli elementi HTML che invece occupano solo lo spazio nel quale vengono utilizzati e non impongono di andare a capo.

Appartengono infine alla **categoria "not displayed"** tutti quei tag che non hanno un impatto visivo nella pagina HTML. Ad esempio i tag <head>, <meta>, <style>.

Abbiamo quindi visto che scrivere il seguente codice:

```
<html>
    <body>
          <a href="par1.html">Par 1</a>
          <a href="par2.html">Par 2</a>
    </body>
</html>
```

significa ottenere il seguente risultato.



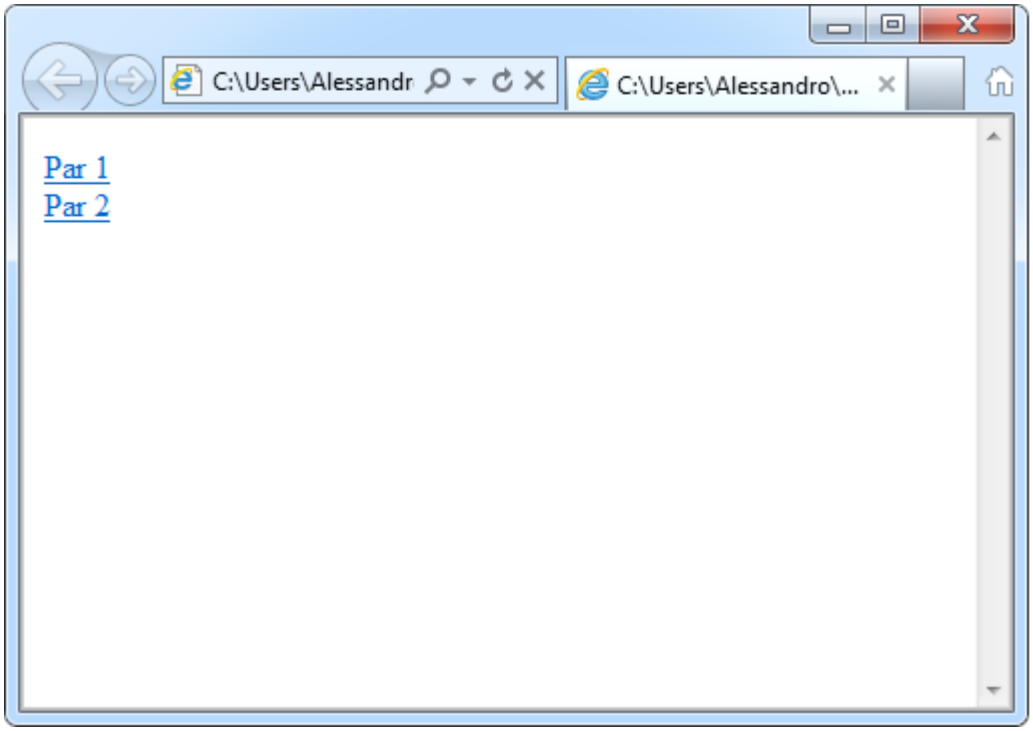
Ciò è la naturale conseguenza del fatto che entrambi gli elementi sono di tipo inline.

Tramite i fogli di stile è tuttavia possibile modificare la natura di un elemento. Esiste infatti **la proprietà "display"** i cui valori (non a caso) possono essere: block, inline, none.
Ecco qui di seguito come sia possibile usare questa proprietà per modificare la natura dell'elemento <a>.

```
<html>
```

```
    <body>
          <a href="par1.html" style=”display:block”>Par 1</a>
          <a href="par2.html">Par 2</a>
    </body>
</html>
```

Questo codice viene tradotto dal browser nella seguente pagina web.



Pur essendo <a> di tipo inline, tramite la proprietà “display” abbiamo modificato il primo elemento <a> trasformandolo in un elemento di tipo block (infatti costringe il secondo elemento ad andare a capo).

Osserviamo che avremmo ottenuto lo stesso risultato aggiungendo un tag
 dopo la chiusura del prima tag <a>.

Infine, se avessimo scritto

Par 1

avremmo fatto scomparire dalla pagina web il primo elemento <a> e il suo posto sarebbe stato occupato dal secondo elemento.

La sintassi

Dopo un'introduzione di tipo concettuale e qualche piccolo esempio pratico è tempo di entrare nel vivo dei fogli di stile studiandone la sintassi, cioè le regole che dobbiamo seguire per dichiararli.
La sintassi generale per una qualsiasi regola css è la seguente:

selettore {proprietà:valore; …; proprietà:valore;}

ad esempio
p { color:blue; margin:20px; }

oppure, equivalentemente
p {
color:blue;
margin:20px;
}

Il selettore indica l'elemento (o gli elementi) a cui sarà applicata la regola che sarà espressa tra le due parentesi graffe. In questo caso la regola sarà applicata all'elemento <p>. Tra le parentesi graffe indicheremo l'elenco delle proprietà da modificare e i rispettivi valori. In questo caso stiamo scegliendo il blu come colore del testo (color:blue;) e un margine di 20px (margin:20px;) tutto intorno all'elemento <p>. Per scrivere quindi una regola css dobbiamo conoscere i concetti di selettore, proprietà e valore.

I selettori

Come abbiamo accennato, un selettore indica al browser quale elemento HTML deve essere selezionato. A tale elemento verranno poi applicate le proprietà indicate nelle parentesi graffe. Questo significa che qualunque elemento HTML può essere un selettore.
Esempio:

```
p {
    color:blue;
    margin:20px;
}
```

indicherà al browser di colorare di blu il testo (color) di tutti i paragrafi (p) e di prevedere un margine di 20 pixel tutto intorno a tale elemento (vedremo più avanti il concetto di margine).
Esiste anche un **selettore universale**: l'asterisco (*). Le proprietà definite all'interno di tale selettore vengono applicate a tutti gli elementi del documento HTML.
Oltre al selettore universale e agli elementi HTML, ci sono altri tipi di selettori. Noi, in questa sede, prenderemo in considerazione solo i più diffusi, ossia:

- classe
- identificatore
- pseudo-classe

Il selettore di classe si basa sul concetto di classe. In questo contesto una classe rappresenta un insieme e di elementi HTML tutti accomunati dallo stesso aspetto e quindi dalle stesse regole css. Ad esempio, se avessimo quattro paragrafi, uno sotto l'altro e volessimo dare a quelli in posizione dispari un colore di sfondo diverso da quelli in posizione pari, potremmo dire che tutti i paragrafi in posizione dispari hanno caratteristiche comuni tra loro. Allo stesso modo tutti i paragrafi in posizione pari hanno caratteristiche comuni tra loro. In una tale situazione l'uso delle classi è molto utile. Vediamo.

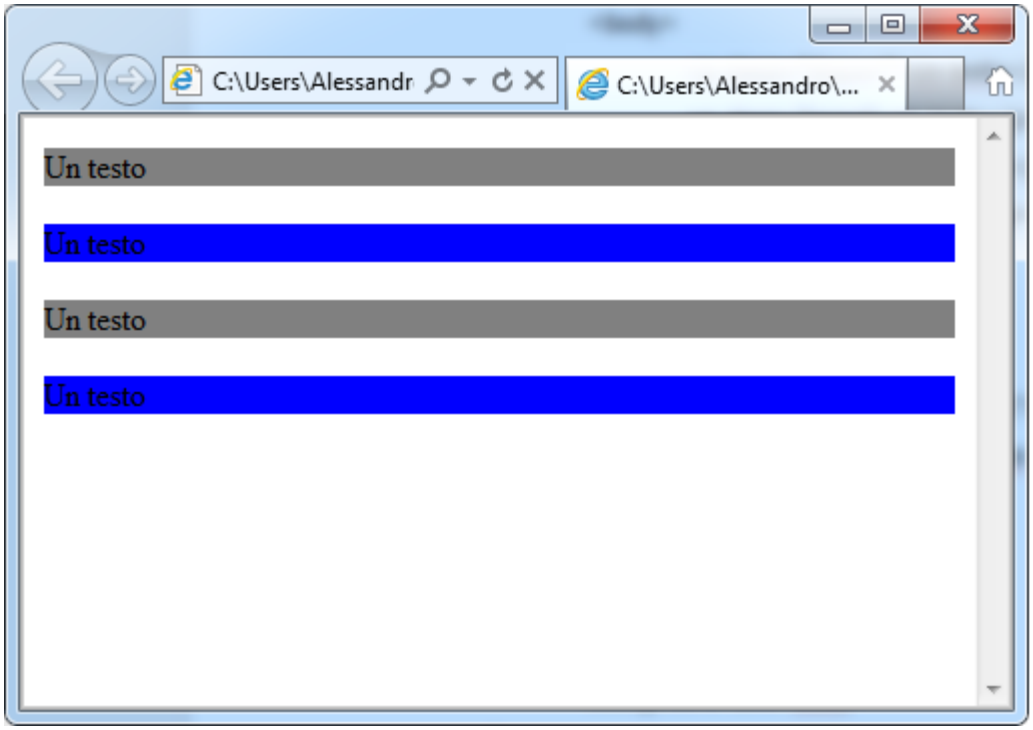
```
<html>
    <body>
          <p class="dispari">Un testo</p>
          <p class="pari">Un testo</p>
          <p class="dispari">Un testo</p>
          <p class="pari">Un testo</p>
    </body>
</html>
```

Il tag <p> è qui associato all'**attributo "class"**. E' tale attributo che associa una determinata classe all'elemento. A questo punto nel file css andremo a definire le regole per tali classi.

```
.dispari {
    background-color:grey;
}
.pari {
    background-color:blue;
```

```
}
```

Abbiamo appena imparato come si definisce il selettore di classe, ossia: nome della classe (scelto da noi) preceduto da un punto (.pari). Questo semplice codice css farà in modo che il browser andrà a cercare tutti gli elementi HTML con attributo class=”pari” e class=”dispari”. Trovati tali elementi il browser applicherà le rispettive regole trovate nel file css.
Il risultato del codice di cui sopra è il seguente.



Ok, è bruttissimo, ma per ora non dobbiamo produrre cose belle.

Il selettore tramite identificatore ha come caratteristica quella di identificare univocamente uno e un solo elemento HTML all’interno del documento. Per ottenere tale risultato si associa all’elemento HTML un identificatore univoco usando l’**attributo “id”**. Andiamo subito sul codice per comprendere meglio.

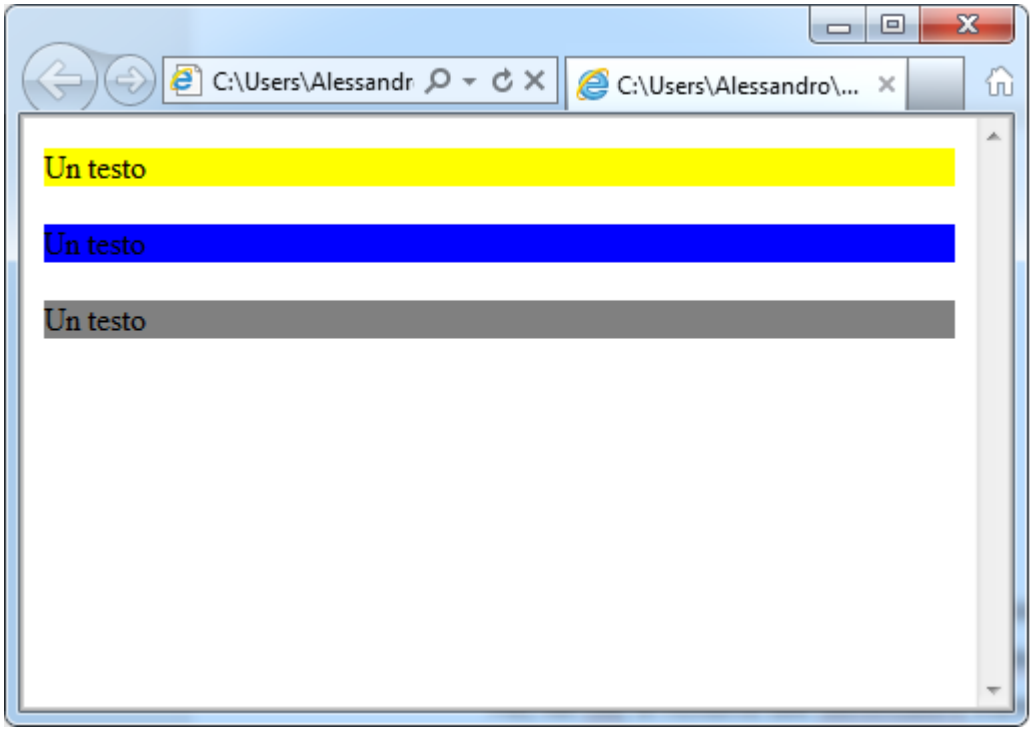
```
<html>
    <body>
          <p id=”primo”>Un testo</p>
          <p id=”secondo”>Un testo</p>
          <p id=”terzo”>Un testo</p>
    </body>
```

```
</html>
```

In questo modo possiamo identificare uno specifico paragrafo grazie al suo id.
Grazie all'assegnazione di un id ad ogni elemento <p>, possiamo associare ad ogni elemento delle caratteristiche esclusive. Esempio.

```
#primo {
    background-color:yellow;
}
#secondo {
    background-color:blue;
}
#terzo {
    background-color:grey;
}
```

Dal codice sopra riportato possiamo dedurre tutto quello che ci serve sapere. Per identificare un particolare elemento (e solo quello) in HTML gli si assegna un identificatore univoco tramite l'attributo "id". Poi, nel css, si richiama tale identificatore tramite **il cancelletto (#)** seguito dal nome dell'identificatore.
Ecco cosa otteniamo dal codice precedente



Va bene, è ancora più brutto del precedente...

Il selettore tramite pseudo-classe si basa sul concetto di pseudo-classe. Una pseudo-classe non definisce un elemento, ma un particolare stato di quest'ultimo. Ad esempio, quando noi clicchiamo su un elemento, esso si trova nello stato di "cliccato". Per dare uno stile ad un elemento cliccato, esiste una precisa pseudo-classe e altre ne esistono per altre situazioni simili. Inoltre le pseudo-classi non sono definite da noi, ma sono predefinite. Esse sono:

- :link
- :visited
- :hover
- :active
- :focus
- :first-letter
- :first-line
- :first-child
- :before
- :after
- :lang (language)

I due punti che precedono le rispettive denominazioni non sono ovviamente un errore, ma un modo per identificare la pseudo-classe (per le classi abbiamo usato il punto, qui vengono usati i due punti). Le pseudo-classi sono preassegnate a tutti gli elementi HTML, non abbiamo quindi bisogno di assegnarle manualmente tramite qualche attributo come nel caso delle classi. Bisogna semplicemente usarle nel codice css.
Si, ma cosa sono e come si usano?
L'idea è sempre la stessa: servono per identificare uno o più elementi HTML a cui applicare lo stile voluto. Ogni pseudo-classe identifica uno o più elementi HTML in base a caratteristiche predefinite. Ad esempio, scrivere p:first significa selezionare il primo paragrafo, mentre a:hover significa selezionare l'elemento <a> sul quale si passa con il mouse.
Non è molto utile esaminare l'uso di ogni singola pseudo-classe. Di seguito però vedremo alcuni tipici utilizzi tramite qualche esempio. Uno dei più importanti usi delle pseudo-classi è quello della selezione degli elementi <a> per modificarne le caratteristiche in base ai movimenti del mouse.
Ammettiamo di avere il seguente codice HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
          <meta http-equiv="content-type"
                content="text/html; charset=UTF-8">
          <title>CSS2</title>
          <link rel="stylesheet" type="text/css"
                href="style/style_01.css">
    </head>
    <body>
          <p><a href="#">Prima voce</a></p>
    </body>
</html>
```

E ammettiamo che questo sia il codice presente in style_01.css

```
a:link {              /*link mai visitato */
    color:#FF0000;
}
a:visited {           /* link già visitato */
    color:#00FF00;
}
a:hover {             /* il mouse passa sul link */
    color:#333;
}
a:active {            /* link attivo in quel momento */
    color:#0000FF;
}
```

Ciò che otterremo è visibile qui:
http://www.alessandrostella.it/lato_client/css2/css2_01.html

Come possiamo notare il link è inizialmente di colore rosso (a:link). Spostando il mouse sul link, esso si colora di grigio scuro (a:hover). Se facciamo click e teniamo premuto il bottone del mouse, possiamo osservare che il colore del link diventa blu (a:active). Infine dopo il click, il colore del link diventa verde (a:visited).
Tutto qua.

Attenzione.
Al fine di ottenere un corretto funzionamento di queste pseudo-classi l'ordine di dichiarazione deve essere esattamente quello mostrato nel codice. Cioè, bisogna dichiarare prima a:link, poi a:visited, poi a:hover e infine a:active. Inoltre non è necessario dichiararli tutti. Se, ad esempio, a noi non interessa dare un particolare stile ad a:visited, possiamo ometterlo.

Per approfondimenti nell'uso delle altre pseudo-classi si rimanda alla documentazione ufficiale w3c.

Proprietà e rispettivi valori

Dopo aver esaminato i più importanti selettori che ci consentono di selezionare uno o più elementi HTML, abbiamo bisogno di capire come sia possibile modificare le proprietà visive degli elementi selezionati dal selettore. In realtà abbiamo già incontrato diversi esempi in cui abbiamo modificato le proprietà degli elementi selezionati e abbiamo visto che il sistema è abbastanza banale. L'aspetto di ogni elemento è definito dal valore di alcune proprietà. Modificando il valore predefinito di tali proprietà possiamo alterare l'aspetto visivo dell'elemento. L'assegnazione dei nuovi valori alle proprietà deve avvenire all'interno delle parentesi graffe con una sintassi precisa:

```
proprietà:valore;
```

cioè

1. nome della proprietà che vogliamo modificare
2. due punti (:)
3. il nuovo valore da assegnare alla proprietà
4. infine un punto e virgola (;)

Seguendo questa sintassi, il browser è in grado di capire che tali proprietà vanno assegnate all'elemento (o agli elementi) selezionato dal selettore indicato subito prima delle parantesi. Ad esempio, la seguente regola:

```
p {
    background-color:black;
    padding: 10px;
}
```

usa il selettore p per selezionare tutti i paragrafi <p> presenti nel documento HTML e a tali elementi imposta a nero il colore di sfondo e distanzia di 10 pixel il testo del paragrafo dai suoi bordi.
Ora, le proprietà degli elementi HTML che possiamo modificare sono davvero tante e per comprenderne completamente il significato è necessario prima fare la conoscenza del box model. Non è pertanto molto proficuo, in questo momento, fare un mero e poco comprensibile elenco di tali proprietà, anche perché, nel prosieguo del libro, ne incontreremo e studieremo moltissime.
Tuttavia è importante avere a disposizione l'elenco completo di tutte le proprietà esistenti e, ovviamente, il w3c lo sa bene. Ecco perché ha creato per noi tale elenco:
http://www.w3.org/TR/CSS21/propidx.html

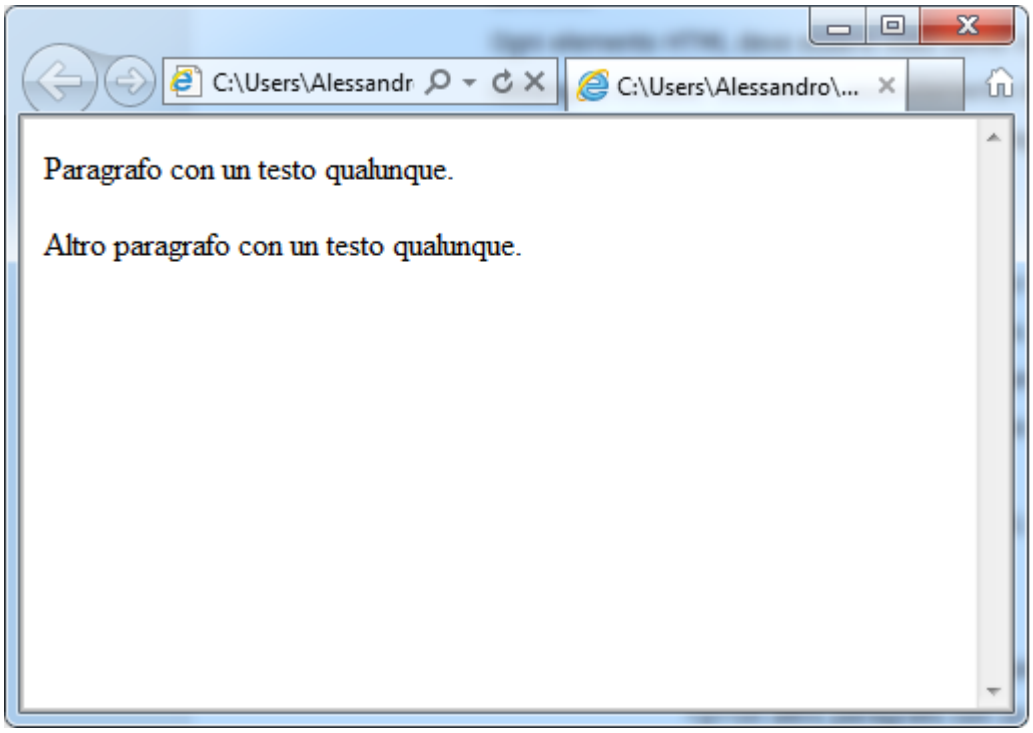
Il box model

Lo studio che stiamo per intraprendere in questo paragrafo è di fondamentale importanza sia per la versione 2 sia per la versione 3 della specifiche CSS. Dunque spegniamo il cellulare, abbandoniamo gli amici, il partner, i giochi, insomma portiamo la concentrazione ai massimi livelli perché per i prossimi minuti il box model dovrà essere il nostro unico dio.
Siamo pronti?
Ogni elemento HTML deve essere visto come fosse racchiuso dentro un box, ossia dentro un rettangolo invisibile.
E' molto importante avere chiaro cosa è il box e come poterlo gestire. Per questo motivo questo capitolo sarà particolarmente dettagliato e accurato.

Abbiamo appena affermato che ogni elemento HTML è racchiuso dentro una sorta di rettangolo e che tale rettangolo è il nostro box. Ma come possiamo fare per "vedere" questo rettangolo invisibile? Vederlo ci potrebbe aiutare a capire di cosa stiamo parlando. In realtà l'abbiamo già visto più volte quando abbiamo scelto di colorare lo sfondo di un paragrafo o di un . Ma ricominciamo.
Ammettiamo di avere il seguente codice HTML:

```
<html>
    <body>
          <p>Paragrafo con un testo qualunque.</p>
          <p>Un altro paragrafo con un testo qualunque.</p>
    </body>
</html>
```

che tradotto dal browser diventa questa pagina web.

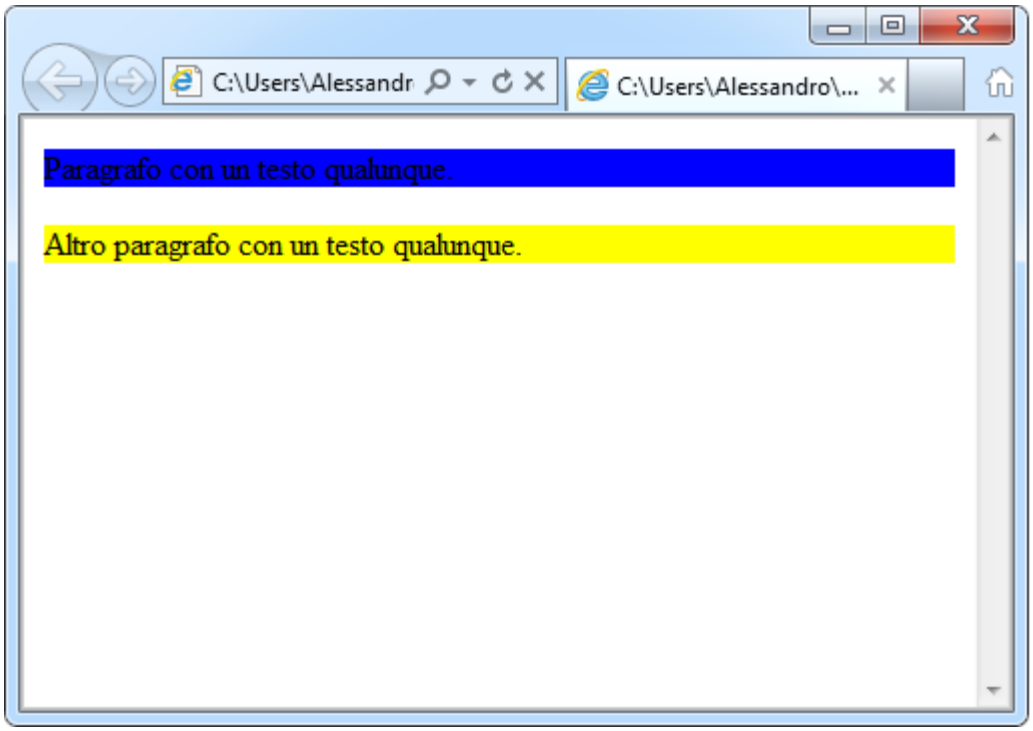


Fin qua è già tutto conosciuto.

Noi adesso sappiamo che ogni elemento HTML (quindi anche un paragrafo) è racchiuso in un rettangolo (box) invisibile. Come possiamo fare per "vederlo"? Semplice: colorando lo sfondo dei due paragrafi! E come coloriamo lo sfondo dei due paragrafi? Lo abbiamo già fatto: creiamo una regola css e usiamo la proprietà "background-color". All'opera! Useremo codice css inline (ricordiamo cosa si intende per css inline?). Il precedente HTML diventa quindi:

```
<html>
    <body>
          <p style="background-color:blue;">Paragrafo
                con un testo qualunque.</p>
          <p style="background-color:yellow;">Altro paragrafo
                con un testo qualunque.</p>
    </body>
</html>
```

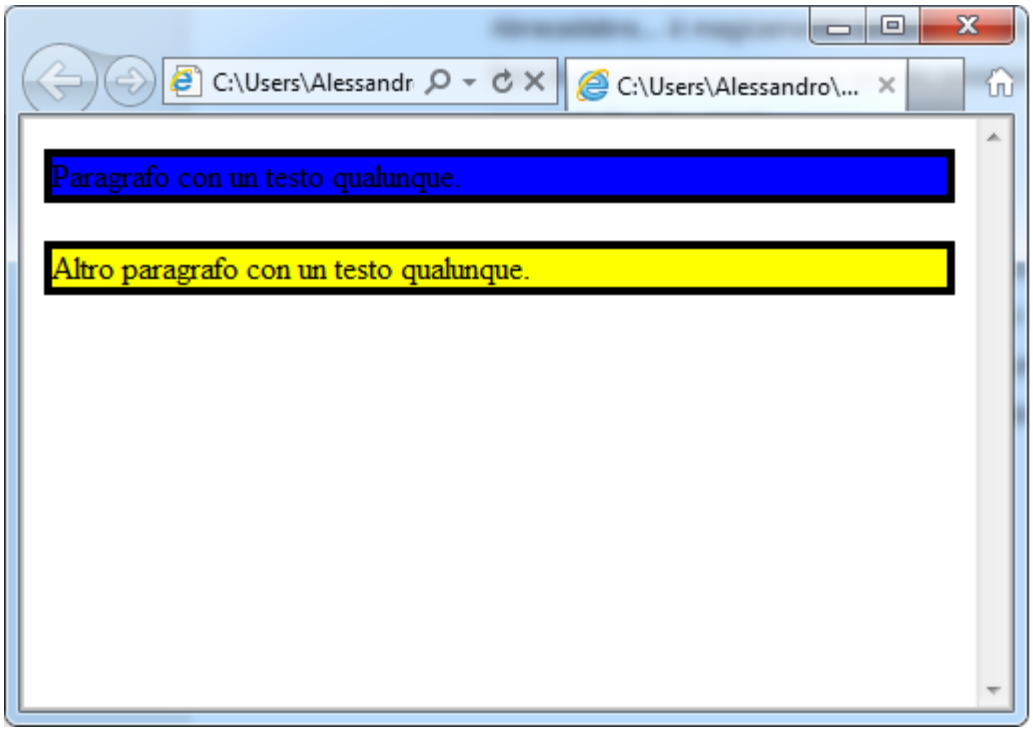
e la corrispondente pagina web diventa questa



Abracadabra... è magicamente comparso lo sfondo del misterioso e invisibile box! Possiamo fare di più! Oltre a mostrare lo sfondo del box, possiamo mostrarne anche i bordi, usando la proprietà "border-style".

```
<html>
    <body>
          <p style="background-color:blue; border-style:solid;">
                Paragrafo con un testo qualunque.</p>
          <p style="background-color:yellow;
                border-style:solid;"> Altro paragrafo con un
                testo qualunque.</p>
    </body>
</html>
```

che il browser traduce così:

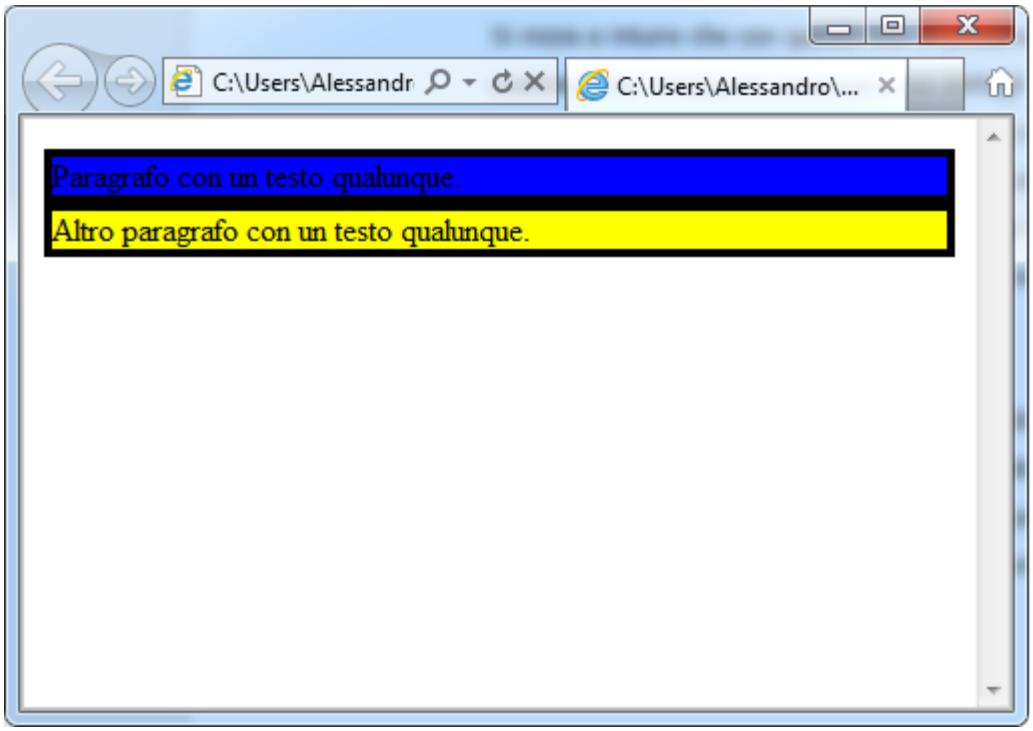


Potremmo anche cambiare il colore dei bordi e il loro spessore.
Si inizia a intuire che con questo box si possono fare tante cosette... Ma quante esattamente? Ad esempio: non ci viene la curiosità di capire cosa è quello spazio bianco tra i due paragrafi? Chi l'ha messo? Noi non abbiamo detto niente al browser! Perché allora c'è quello spazio?
La risposta è da ricercarsi nell'elemento <p>. Esso rappresenta un paragrafo e un paragrafo, per impostazione predefinita, include un margine superiore e un margine inferiore, cioè una distanza tra gli elementi che si trovano sopra e quelli che si trovano sotto. Tuttavia, usando le proprietà giuste, possiamo intervenire ed eliminarlo, modificando le impostazioni predefinite. Per fare questo ci basta usare un'altra delle sue tante proprietà, la proprietà margin.

```
<html>
    <body>
          <p style="background-color:blue;
                border-style:solid;margin:0px;">Paragrafo con
                un testo qualunque.</p>
          <p style="background-color:yellow; border-style:solid;
                margin:0px;"> Altro paragrafo con un testo
                qualunque.</p>
    </body>
```

```
</html>
```

che il browser tradurrà così



E' arrivato il momento di conoscere tutte le proprietà del box che possiamo modificare. Eccole qui elencate partendo dall'esterno del box e procedendo verso l'interno:

- **margin**
- **border**
- **padding**
- **content**

Ed eccole tradotte visivamente.

Dobbiamo comprendere tutto e bene.

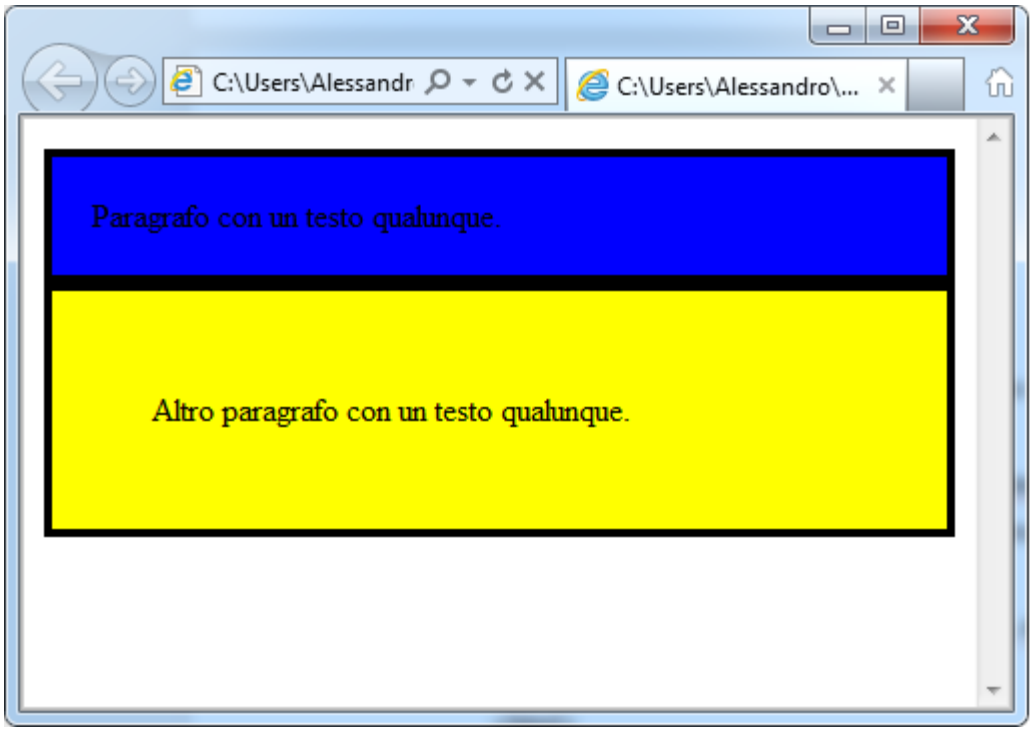
Partiamo dal content, cioè dall'interno.

Nel nostro codice di esempio, il **content** è rappresentato dal testo, cioè "Paragrafo con un testo qualunque.". Secondo la figura qui sopra tra il content e il border c'è dello spazio gestito dalla proprietà "**padding**" che non abbiamo ancora preso in considerazione. Poi ci sono i bordi e infine, oltre i bordi, c'è altro spazio gestito dalla proprietà "**margin**".

Noi abbiamo già visto tutto, tranne il padding. Aggiungiamo allora il padding al nostro codice e vediamo cosa succede nel browser.

```
<html>
    <body>
          <p style="background-color:blue;border-style:solid;
                margin:0px;padding:20px;">Paragrafo con un testo
                qualunque.</p>
          <p style="background-color:yellow; border-style:solid;
                margin:0px;padding:50px;">Altro paragrafo con un
                testo qualunque.</p>
    </body>
</html>
```

che il browser traduce così:



Abbiamo volutamente usato due padding diversi, da 20 pixel per il paragrafo blu, da 50 pixel per il paragrafo giallo. Quello che si nota dalla figura è che settando la proprietà padding, viene aggiunto dello spazio vuoto tutto intorno al content, tra il content e il bordo.

Fino ad ora abbiamo gestito le proprietà in modo omogeneo, cioè specificando lo stesso valore per tutti i lati del box, sopra, sotto a sinistra e destra. Ma quasi mai è così! Per esempio, volendo dare un margine solo a sinistra al nostro primo paragrafo, come possiamo fare?
Per margin, padding e border esistono proprietà che possono gestire singolarmente ogni lato del box. Ossia:

- margin-left
- margin-right
- margin-top
- margin-bottom
- padding-left
- padding-right
- padding-top
- padding-bottom
- border-left
- border-right

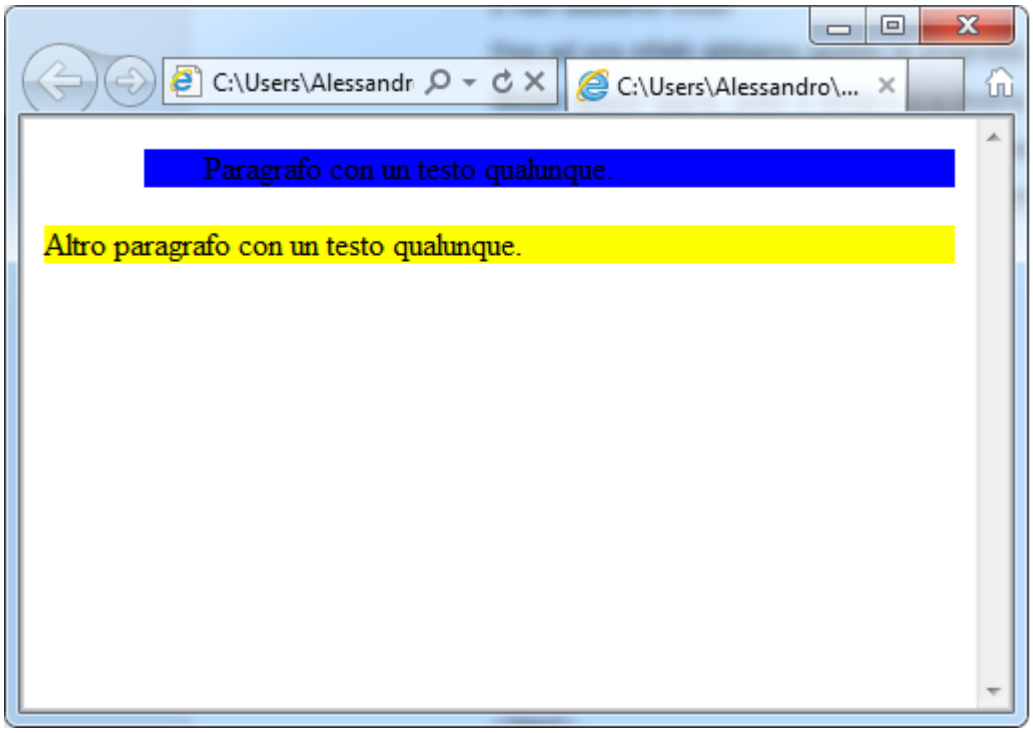
- border-top
- border-bottom

Le proprietà non sono certamente tutte qui. Sono tante e per ognuna si possono avere diversi valori. Per tutti i dettagli su tutte le proprietà disponibili si rimanda, come sempre, al sito ufficiale del w3c:
http://www.w3.org/TR/CSS2/box.html

Vediamo comunque qualche piccola applicazione.

```
<html>
    <body>
          <p style="background-color:blue;margin-left:50px;
                padding-left:30px;">Paragrafo con un testo
                qualunque.</p>
          <p style="background-color:yellow;">Altro paragrafo
                con un testo qualunque.</p>
    </body>
</html>
```

che tradotto diventa

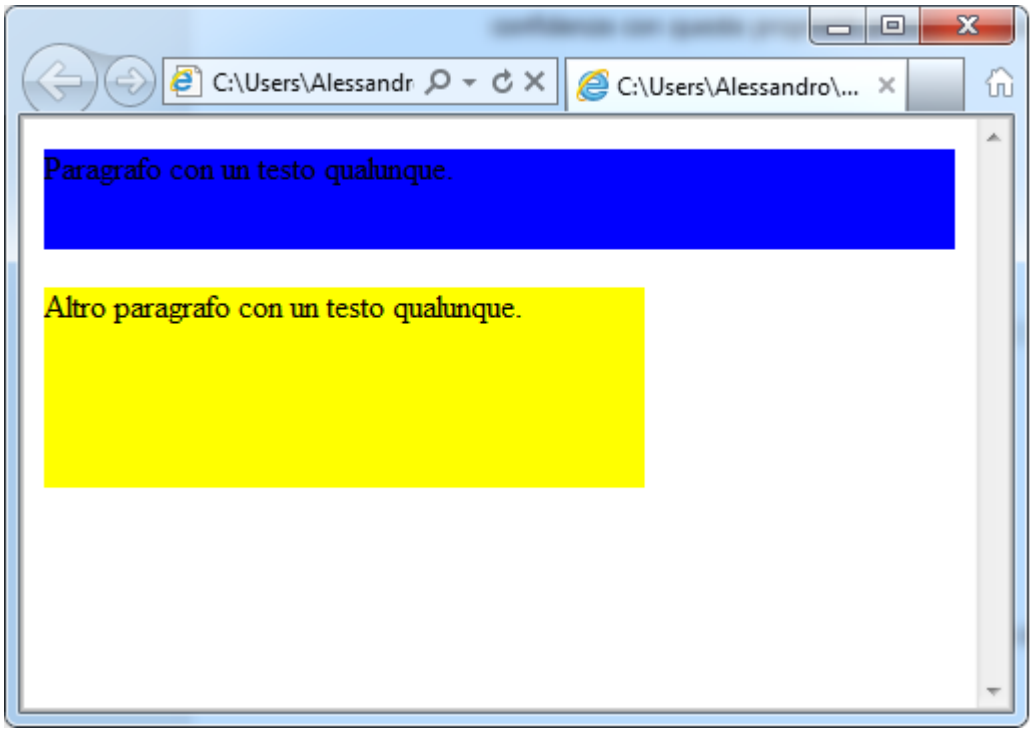


Da questa figura si può notare il margine di 50 pixel sulla sinistra del box blu (lo spazio bianco alla sinistra del box blu) e il padding di 30 pixel, sempre nel box blu, presente tra il testo e il bordo sinistro.

Oltre a queste proprietà ve ne sono altre due che è necessario prendere in considerazione: **width** e **height**. Queste due proprietà gestiscono rispettivamente larghezza e altezza del content. Poniamo attenzione a quanto abbiamo letto: **larghezza e altezza del content**!
Ecco un esempio.

```
<html>
    <body>
          <p style="background-color:blue;height:50px">Paragrafo
                con un testo qualunque.</p>
          <p style="background-color:yellow;height:100px;
                width:300px;">Altro paragrafo con un testo
                qualunque.</p>
    </body>
</html>
```

la cui traduzione del browser diventa



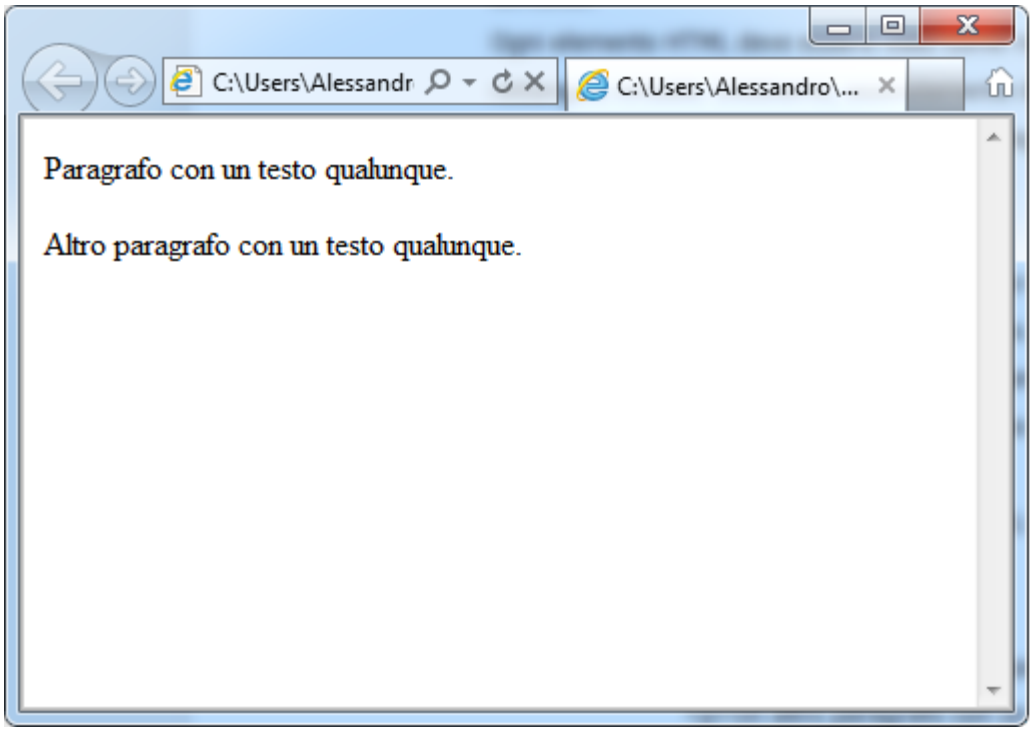
Poiché <p> è un elemento di tipo block, per impostazione predefinita il suo box ha una larghezza pari a tutta la larghezza della pagina. Intervenendo sulla proprietà width del content però noi possiamo restringere la larghezza predefinita del box, come nel caso del box giallo.
Prima di procedere ulteriormente nello studio dei css, si consiglia caldamente di prendere confidenza con queste proprietà procedendo con una buona quantità di prove. E' assolutamente indispensabile avere chiari i concetti di content, padding, border, margin, width ed height. Scriviamo dunque un po' di codice e osserviamone i risultati nel browser.

Il posizionamento degli elementi

I fogli di stile sono in grado di modificare la posizione di ogni elemento HTML. Il codice HTML viene letto dal browser da sinistra a destra e dall'alto in basso. Mano a mano che il browser incontra gli elementi, essi vengono disposti sulla pagina web.
Tramite i fogli di stile è possibile modificare questo comportamento.
Ritorniamo ai nostri due paragrafi iniziali.

```
<html>
    <body>
          <p>Paragrafo con un testo qualunque.</p>
          <p>Altro paragrafo con un testo qualunque.</p>
    </body>
</html>
```

In base a quanto appena affermato, il browser, leggendo questo codice, disegna sulla pagina web il primo paragrafo. Essendo <p> di tipo block, il browser riempie la prima riga della pagina con il primo paragrafo, va a capo e prosegue nella lettura del codice. Incontra così il secondo paragrafo ed esegue gli stessi passi fatti con il primo.
Ecco perché, alla fine, la pagina web avrà il seguente aspetto.



Tutto normale. Si tratta di due elementi di tipo block e quindi vengono disposti uno sotto l'altro. Abbiamo però già visto che, usando la **proprietà display** siamo in grado di modificare la natura di un elemento HTML, trasformando un elemento inline in uno block e viceversa.
Ammettiamo quindi che noi, per esigenze divine, abbiamo bisogno di posizionare i due paragrafi non uno sotto l'altro, ma uno di fianco all'altro. Inoltre vogliamo che uno sia incollato al bordo sinistro dell pagina web mentre l'altro sia incollato al margine destro. Come fare?

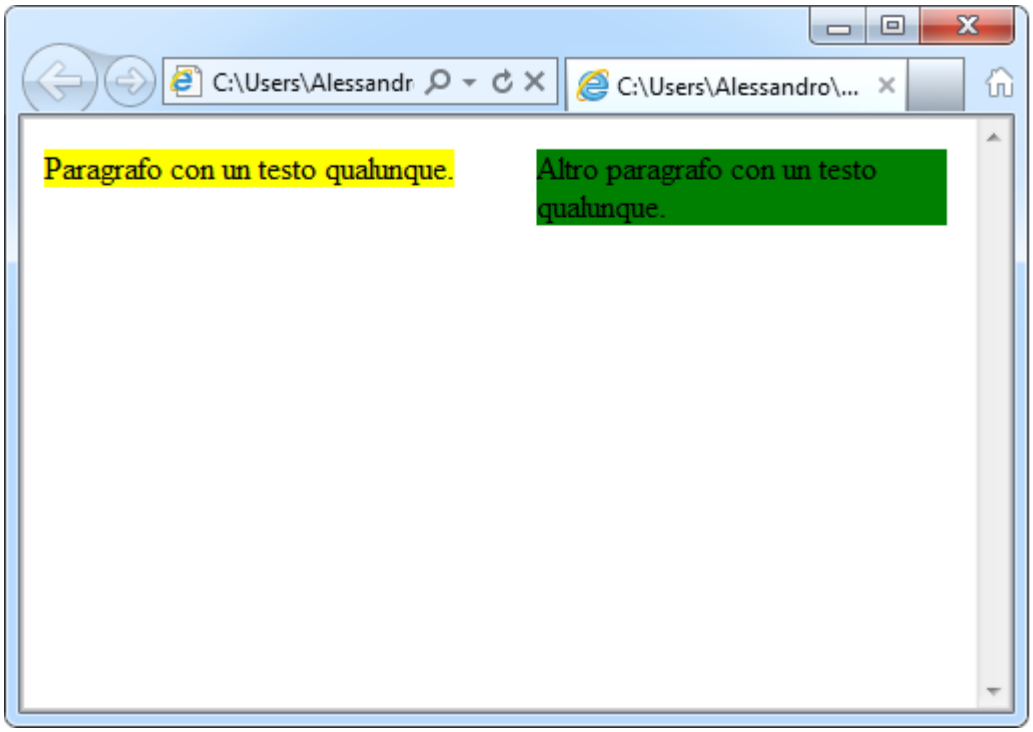
La proprietà float

Sappiamo che <p> è un elemento di tipo block, quindi occuperà tutto lo spazio a sua disposizione sulla sua riga e imporrà di andare a capo. Abbiamo anche visto che possiamo usare la proprietà display per modificare la sua natura. Ma rendendo entrambi i paragrafi di tipo inline riusciremmo solo a metterli uno di fianco all'altro. A noi non basta perché uno dei due deve incollarsi al margine destro della pagina web. E quindi?
La verità è che non abbiamo ancora incontrato la proprietà che ci risolve il problema: **la proprietà float**.
Vediamo un codice che realizza quello che ci serve:

```
<html>
    <body>
```

```
            <p style="float:left;width:45%;
                  background-color:yellow;">Paragrafo con un testo
                  qualunque.</p>
            <p style="float:right;width:45%;
                  background-color:green;"> Altro paragrafo con un
                  testo qualunque.</p>
    </body>
</html>
```

Leggendo questo codice, il browser mostra la seguente pagina web.



Si, è proprio quello che volevamo ottenere e possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/css2/css2_02.html

Ma come ci siamo riusciti?
Come accennato, abbiamo dovuto usare la proprietà float.
La proprietà float è molto importante.
Con questa proprietà è possibile **rimuovere un elemento (il suo box) dal normale flusso del documento** e spostarlo su uno dei lati (destro o sinistro) del suo elemento contenitore. Il contenuto che circonda l'elemento scorrerà intorno ad esso sul lato opposto rispetto a quello indicato come valore di float.

Nel nostro caso <body> è l'elemento contenitore di entrambi gli elementi <p>. Noi abbiamo assegnato la proprietà float in modo da allineare un elemento a sinistra e l'altro a destra. Inoltre abbiamo dato una larghezza (width) pari al 45% ad entrambi gli elementi. Ciò non è ovviamente un caso, per due motivi:

1. quando la proprietà float si applica ad elementi contenenti testo (come nel nostro caso), **è obbligatorio** usare per essi anche la proprietà width
2. volevamo dividere i due elementi da uno spazio bianco

Osserviamo che quando usiamo una percentuale per la proprietà width, tale percentuale si riferisce all'elemento contenitore. Quindi nel nostro caso la larghezza viene intesa come il 45% della larghezza di <body>.
La proprietà float può assumere i seguenti valori:

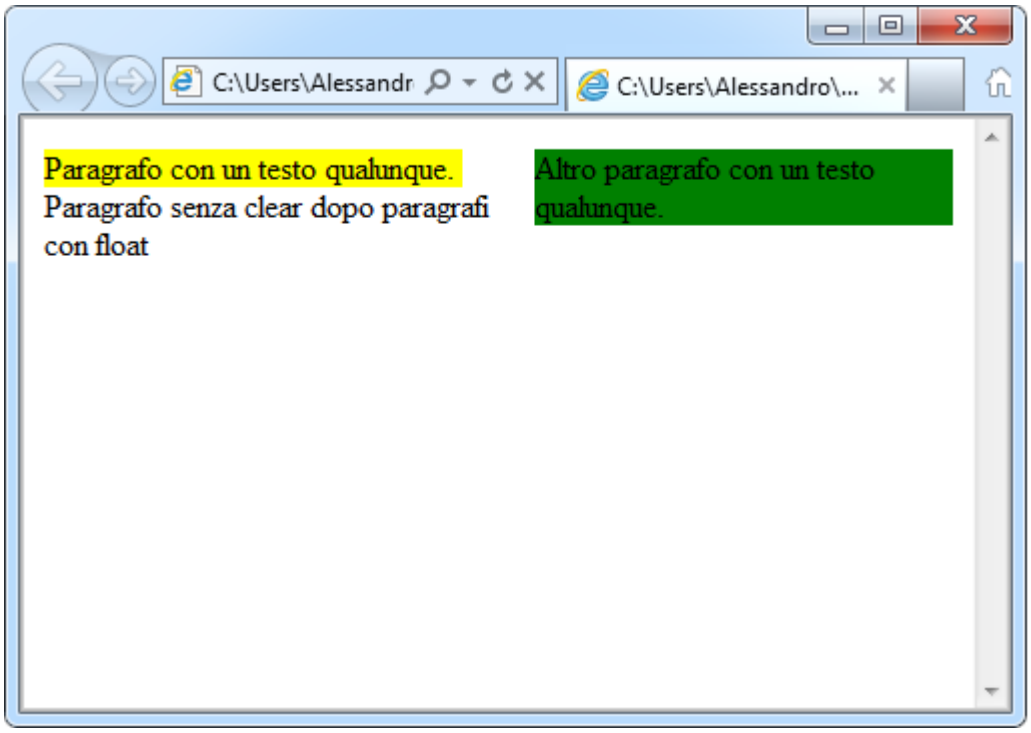
- left
- right
- none

La proprietà clear

Un'altra proprietà che interferisce nel posizionamento degli elementi è la proprietà clear.
La proprietà clear serve a impedire che al fianco di un elemento compaiano altri elementi il cui flusso sia stato alterato tramite la proprietà float. Si applica solo agli elementi blocco e non è ereditata.
L'origine di tale proprietà è questa: visto che il float sposta un elemento dal flusso normale del documento, è possibile che esso venga a trovarsi in posizioni non desiderate, magari al fianco di altri elementi che vogliamo invece tenere separati. clear risolve questo problema.
Per capire cosa significa ci basta aggiungere nel nostro documento un altro paragrafo.

```
<html>
    <body>
          <p style="float:left; width:45%;
                background-color:yellow;">Paragrafo con un testo
                qualunque.</p>
          <p style="float:right; width:45%;
                background-color:green;">Altro paragrafo con un
                testo qualunque.</p>
          <p>Paragrafo senza clear dopo paragrafi con float</p>
    </body>
</html>
```

Il browser produce questa pagine web.

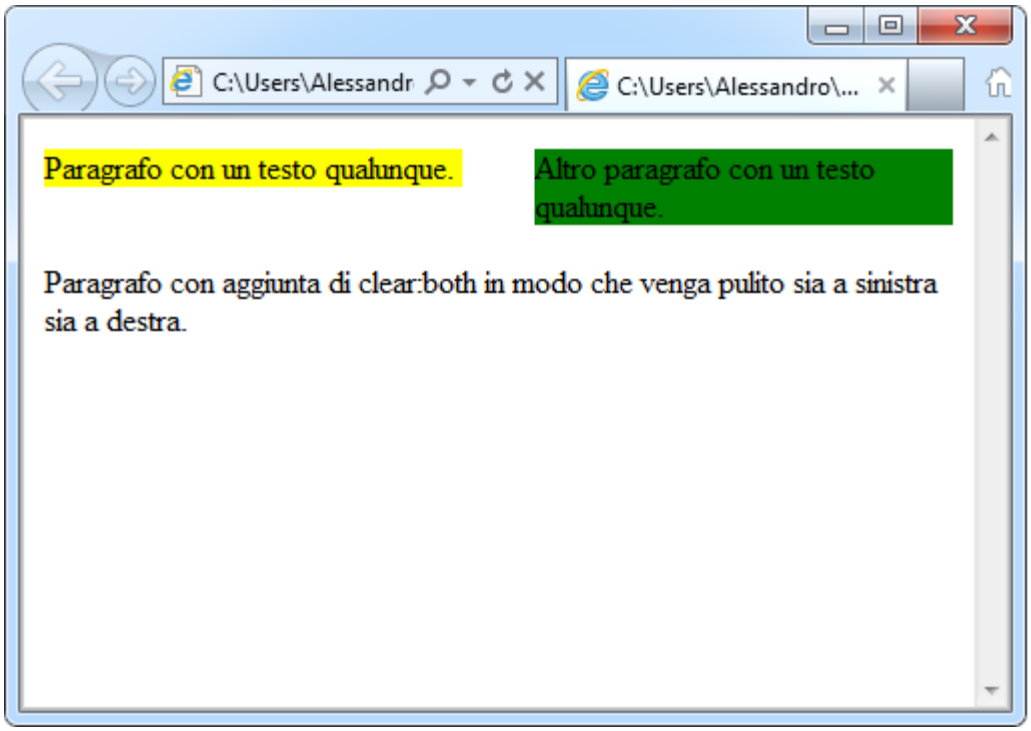


che probabilmente non è esattamente quello che vogliamo.

Per fare in modo che il terzo paragrafo inizi sotto ai due iniziali, dobbiamo usare la proprietà clear in modo che "pulisca" sia a destra sia a sinistra del terzo paragrafo.

```
<html>
    <body>
          <p style="float:left; width:46%;
                background-color:yellow;">Paragrafo con un testo
                qualunque.</p>
          <p style="float:right; width:46%;
                background-color:green;">Altro paragrafo con un
                testo qualunque.</p>
          <p style="clear:both">Paragrafo con aggiunta di
                clear:both in modo che venga pulito sia a
                sinistra sia a destra.</p>
    </body>
</html>
```

Il risultato somiglia molto di più a quanto desiderato.



Possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/css2/css2_03.html

La proprietà clear può assumere i seguenti valori:

- left
- right
- both
- none

La proprietà position

Per impostazione predefinita, tutti i box (e quindi gli elementi) HTML sono posizionati in un modo chiamato statico. Quindi tutte le operazioni che abbiamo eseguito fino a questo momento sono state eseguite su box posizionati in modo statico.
Anche in questo caso però, usando i fogli di stile, possiamo modificare le impostazioni di base. La proprietà che ci consente di fare questo è position.
La proprietà position è un'altra proprietà fondamentale per la gestione della posizione degli elementi di cui determina la modalità di presentazione sulla pagina. Si applica a tutti gli elementi e non è ereditata. Può assumere i seguenti valori:

- static
- absolute
- fixed

- relative

Vediamone significato e utilizzo.

static

E' il valore di default, quello predefinito per tutti gli elementi non posizionati secondo un altro metodo. Rappresenta la posizione normale che ciascun elemento occupa nel flusso del documento. E' molto importante ricordare che gli elementi posizionati in modo statico non sono soggetti alle proprietà left, right, top e bottom. E' quindi inutile impostare la proprietà bottom a 0 sperando che un elemento con posizione statica si sposti in basso!

absolute

L'elemento, o meglio, il box dell'elemento viene rimosso dal flusso del documento ed è posizionato in base alle coordinate fornite con le proprietà top, left, right o bottom. Il posizionamento avviene sempre rispetto al primo elemento antenato (ancestor) che abbia un posizionamento diverso da static. Se tale elemento non esiste il posizionamento avviene in base all'elemento radice HTML, che in condizioni standard coincide con l'area del browser che contiene il documento e che ha inizio dall'angolo superiore sinistro di tale area. Un elemento posizionato in modo assoluto scorre insieme al resto del documento. Possiamo osservare il comportamento del terzo paragrafo del codice precedente, qualora lo posizionassimo in modo assoluto (position:absolute) e allineato in basso (bottom:0px).
http://www.alessandrostella.it/lato_client/css2/css2_04.html

Ribadiamo quanto abbiamo detto.
Un elemento con posizione assoluta e allineamento in basso esegue l'allineamento rispetto al primo elemento che lo contiene e che non sia allineato in modo statico.
Nel caso del nostro paragrafo, quindi, essendo contenuto nell'elemento <body>, viene innanzitutto controllato il tipo di posizione di <body> che è di tipo statico. Quindi si procede verso il contenitore di <body> che è <html> il quale non ha un padre e quindi il nostro paragrafo si allinea in base ad esso.

fixed

Usando questo valore il box dell'elemento viene, come per absolute, sottratto al normale flusso del documento. La differenza sta nel fatto che per fixed il box contenitore è sempre il cosiddetto viewport. Con questo termine si intende la finestra principale del borwser, ovvero l'area del contenuto. Altra differenza fondamentale: un box posizionato con fixed non scorre con il resto del documento. Rimane, appunto, fisso al suo posto.

L'effetto è lo stesso che si può ottenere con l'uso dei frame, in cui una parte della pagina rimane fissa e il resto scorre. Solo che qui il documento è solo uno. Purtroppo, il valore non è supportato da Explorer su Windows.

relative

L'elemento viene posizionato relativamente al suo box contenitore. La posizione viene anche qui impostata con le proprietà top, left, bottom, right. Tuttavia, a differenza del posizionamento assoluto, esse non indicano un punto preciso, ma l'ammontare dello spostamento in senso orizzontale e verticale rispetto al box contenitore. In altre parole utilizzare questo tipo di posizionamento consente di usare le proprietà appena elencate senza alterare il flusso di lettura.

Ereditarietà e conflitti

Siamo arrivati di fronte all'ultimo ostacolo. Le ultime pagine di sofferenza e poi avremo completato la conoscenza dei fogli di stile. Sono regole un po' complesse, ma basilari. Quindi attenzione massima. Gli esempi che accompagnano la parte teorica ci aiuteranno nel percorso di apprendimento.

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. Ammettiamo di avere il seguente codice HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
          <title>Ereditarietà</title>
          <meta http-equiv="Content-Type" content="text/html;
                charset=utf-8">
    </head>
    <body>
          <div style="color:red">
                <p>Paragrafo con un testo qualunque.</p>
                <p>Altro paragrafo con un testo qualunque.</p>
                <p style="color:black">Ultimo paragrafo con
                      modifica del colore del testo.</p>
```

```
            </div>
        </body>
    </html>
```

L'esempio è al riguardo chiarificatore.
Se impostiamo il colore rosso per il testo (color:red;) a livello dell'elemento 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.
Possiamo vedere il codice in azione qui:
http://www.alessandrostella.it/lato_client/css2/css2_05.html

Non tutte le proprietà però sono ereditate. In genere non vengono ereditate 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, ad esempio, imposto un bordo di 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. Per esempio, se definisco queste regole in un CSS:
p {color: black;}
.testo {color: red;}

E in una pagina HTML scrivo questo codice:
Testo del paragrafo

Il testo del paragrafo viene scritto in rosso e non in nero! Perché? Cioè, perché il selettore di classe prevale su quello semplice con elemento?
Intanto possiamo vedere il codice in azione qui:
http://www.alessandrostella.it/lato_client/css2/css2_06.html

Il primo concetto da imparare è quello di peso. Si riferisce alla maggiore o minore importanza da assegnare a ciascuna regola. Se un documento HTML usa più regole e più fogli di stile, il browser deve essere in grado di decidere quale regola e quale foglio di stile applicare a ogni elemento HTML presente nel documento.
Il primo criterio utilizzato dal browser per operare tale scelta è rappresentato dall'origine della regola (o del foglio di stile). Ad ogni regola viene infatti assegnata un'importanza in base all'origine del foglio di stile al quale la regola appartiene. L'ordine di importanza, dalla maggiore alla minore, è il seguente:

1. foglio dell'autore (cioè noi che produciamo il codice)
2. foglio dell'utente
3. foglio predefinito del browser

Quindi se le impostazioni dell'utente richiedono uno sfondo bianco e un testo nero, ma le impostazioni dell'autore del documento HTML richiedono di mostrare uno sfondo nero e un testo bianco, vince la decisione dell'autore.

Tutti i software di navigazione di ultima generazione consentono di modificare la regola appena studiata. 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 specifico delle varie regole all'interno di un foglio di stile. Esistono regole ben precise per calcolarla e sono quelle che applica lo user agent (UA) di un browser quando si trova davanti ad un CSS.
I fattori usati dal browser per eseguire il calcolo sono tre:

1. selettori ID
2. classi e pseudo-classi
3. elementi

Quindi 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. Ciascun fattore viene indicato con un numero, ottenendo così una tripletta che viene usata come valutazione.
Per esempio, il seguente codice css

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

ottiene la seguente valutazione:
1 ID
0 classi e pseduo-classi
0 elementi
Tripletta di valutazione: 1-0-0

Il codice CSS:
.miaClasse {color: blue;}

ottiene la seguente valutazione:
0 ID
1 classi e pseduo-classi
0 elementi
Tripletta di valutazione: 0-1-0

Infine il codice:
h1 {color: red;}

ottiene la seguente valutazione:
0 ID
0 classi e pseduo-classi
1 elemento
Tripletta di valutazione: 0-0-1

Il peso specifico della prima regola è il maggiore (100). Quello dell'ultima il minore (001). In pratica: **gli ID pesano più delle classi che pesano più dei singoli elementi**. Non commettiamo 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

Abbiamo già visto che gli stili in linea (inline) prevalgono su quelli incorporati (interni) che a loro volta prevalgono su quelli collegati (esterni). Ora però tenteremo di ricostruire il

procedimento "a cascata" completo che esegue il browser per scegliere i fogli di stile e le singole regole al fine di trasformare un documento HTML in una pagina web.
Il browser procede in questo modo:

1. cerca di recuperare il foglio di stile corretto in base al **dispostivo collegato** in quel momento. 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. ordina le regole per **peso** e **origine** secondo le regole viste sopra. Se 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. calcola la **specificità** dei selettori e in caso di conflitto tra regole usa il criterio che abbiamo visto all'inizio di questo paragrafo: gli stili in linea (inline) prevalgono su quelli incorporati (interni) che a loro volta prevalgono su quelli collegati (esterni)

La parola chiave !important

Come sempre, le regole sono fatte per essere infrante dalle eccezioni!
La parola chiave !important è questa eccezione: se una dichiarazione viene accompagnata dalla parola chiave **!important**, tale dichiarazione balza al primo posto nell'ordine di applicazione a prescindere da peso, origine, specificità e ordine.
Il seguente codice ne mostra un esempio di utilizzo:

```
p {
        color: black !important;
}
```

Il codice impone il colore nero ai paragrafi del documento.
Possiamo osservarla in azione qui:
http://www.alessandrostella.it/lato_client/css2/css2_07.html

Bene.
Ora che abbiamo imparato l'HTML e i concetti di base dei css, possiamo finalmente passare all'azione.

5
HTML4 e CSS2 all'opera

E' arrivato il tempo di mettere in pratica tutto quello che abbiamo imparato. Questo è il capitolo della pratica, il capitolo del fare. Producendo codice reale entreremo in contatto con tanti problemi pratici che la teoria non può trattare.

In questo capitolo impareremo a gestire il layout di una pagina web e a utilizzare i contenuti teorici studiati fino a questo momento. Lo faremo partendo da zero, cioè dalla creazione di un sito sul nostro disco rigido.
Impareremo nozioni importanti come centrare un box, creare un layout a 2 colonne (centrato o fluido), creare un menù di navigazione tramite i css e tanto altro.

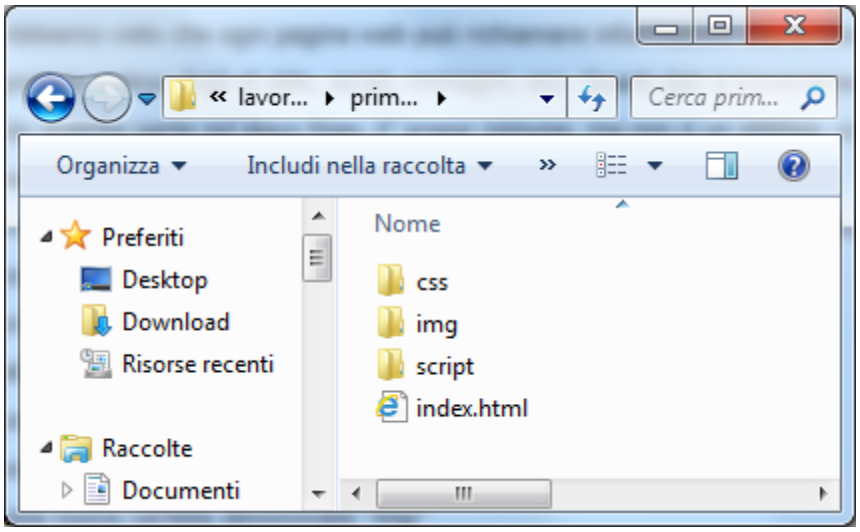
Creare un sito sul disco fisso

Abbiamo visto che ogni pagina web può richiamare informazioni che si trovano fuori dal proprio codice: fogli di stile, script, immagini, ecc. Questi dati si troveranno presumibilmente da qualche parte sul disco fisso. E' prassi comune, ma non è un obbligo, creare un sito seguendo una struttura di cartelle abbastanza diffusa e consolidata.
Ammettiamo di voler creare un sito di nome "primoProgetto". La prassi dice di procedere come segue:

1. creare sul disco una nuova cartella e chiamarla "primoProgetto"
2. all'interno di tale cartella creare:
    a. un file di testo denominato index.html (oppure index.htm)
    b. una nuova cartella denominata "css"
    c. una nuova cartella denominata "img"

d. una nuova cartella denominata "script"

In modo tale da ottenere alla fine qualcosa di simile a quanto mostrato in figura.



Dovrebbe essere abbastanza intuitivo comprendere l'uso delle cartelle così create. Nella cartella "css" vengono inseriti tutti i codici CSS utilizzati dal sito. Nella cartella "img" si inseriscono le immagini usate sia da HTML sia da CSS. Nella terza cartella vengono inseriti i vari script. I nomi delle cartelle sono a nostra discrezione. Quello che invece non è a nostra discrezione è il nome della pagina principale del sito, quella che notoriamente è chiamata home page. Essa deve necessariamente chiamarsi **index.html** oppure **index.htm**. Prima domanda: che differenza c'è tra le 2 estensioni (cioè htm e html)? La risposta è: nessuna!
Seconda domada: perché dobbiamo necessariamente creare il file index? La risposta è: perché quando noi scriviamo sul browser
http://www.alessandrostella.it/
in realtà il browser cercherà
http://www.alesandrostella.it/index.html

Se non trova il file index.html cercherà altri tipi di file in base alla tecnologia usata e ad altri parametri, ma la prima ricerca avviene per il file index.html o index.htm.

Centrare un box nella pagina

Come abbiamo visto in occasione della discussione sul layout di un sito, i layout più diffusi sono quelli a 2 o 3 colonne, centrati o fluidi. Definire un layout come centrato sta ad indicare che, quando la finestra del browser viene ridimensionata, i contenuti del sito mantengono una larghezza fissa e restano al centro della finestra. Definire un layout

fluido, invece, indica la caratteristica del sito di allargarsi e restringersi quando viene ridimensionata la finestra del browser.
Il sito di Repubblica è un tipico layout centrato.
http://www.repubblica.it/

Il sito di SalentoPaghe è un classico esempio di layout fluido.
http://www.salentopaghe.it/

Prima di arrivare a disegnare un layout complesso è però necessario imparare a gestire un singolo box. La prima cosa da imparare è **centrare un box nella pagina web**. Abbiamo appena imparato che, in base alla tipologia di layout, possiamo avere un box a larghezza fissa (layout centrato) oppure a larghezza mobile (layout fluido). Dobbiamo imparare a centrare il box in entrambi i casi. Il codice HTML sarà sempre lo stesso:

```
<body>
    <div id="centrato">BOX CENTRATO</div>
<body>
```

Creiamo una cartella nella quale produrremo il nostro codice. Diamo a questa cartella nome "mioSito". Dentro la cartella "mioSito" creiamo una cartella e diamole nome "style" (in questa cartella inseriremo il nostro codice css).
Apriamo adesso Notepad++ e creiamo un nuovo file. Poi, come visto in precedenza, scegliamo "Language/H/HTML" in modo da comunicare al programma che il file appena creato è di tipo HTML. Infine, sempre dal menù, scegliamo "Encoding" e poi "Encode in UTF-8" in modo da comunicare al programma di usare la codifica UTF che permette di rappresentare tutti i caratteri, a differenza di codifiche più vecchie.
Il nostro file appena creato avrà il seguente codice:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
          <title>Centrare un box</title>
          <meta http-equiv="Content-Type" content="text/html;
                charset=utf-8">
          <link rel="stylesheet" type="text/css"
                href="style/style_00.css">
    </head>
```

```
    <body>
          <div id="centrato">BOX CENTRATO</div>
    </body>
</html>
```

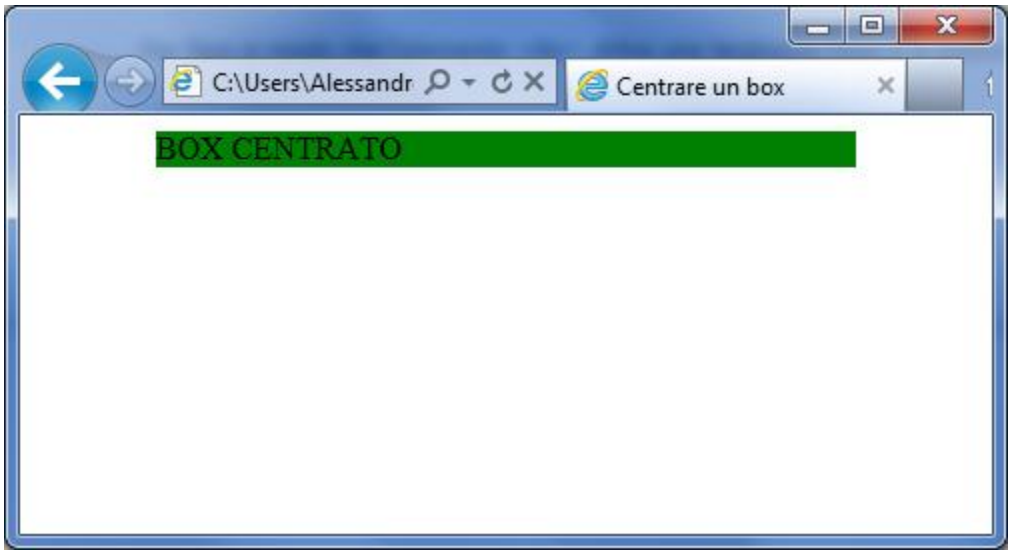
Lo salviamo nella cartella "mioSito" e gli diamo nome "html4css2_00.html".
Ora, sempre in Notepad++, creiamo un nuovo file, poi "Language/C/CSS", poi "Encoding/Encode in UTF-8", lo salviamo nella cartella "mioSito/style" e gli diamo nome "style_00.css".
Quello che scriveremo in questo file dipenderà da quanto andremo a vedere nei prossimi paragrafi.

Centrare un box a larghezza fissa

Per fare in modo che un box con id="centrato" abbia una larghezza fissa (ad esempio 350 pixel) e si posizioni sempre al centro della nostra pagina web, dobbiamo usare il seguente codice CSS:

```
#centrato {
    width:350px;
    background-color:green;
    margin-left:auto;
    margin-right:auto;
}
```

Il risultato è il seguente.



E possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_00.html

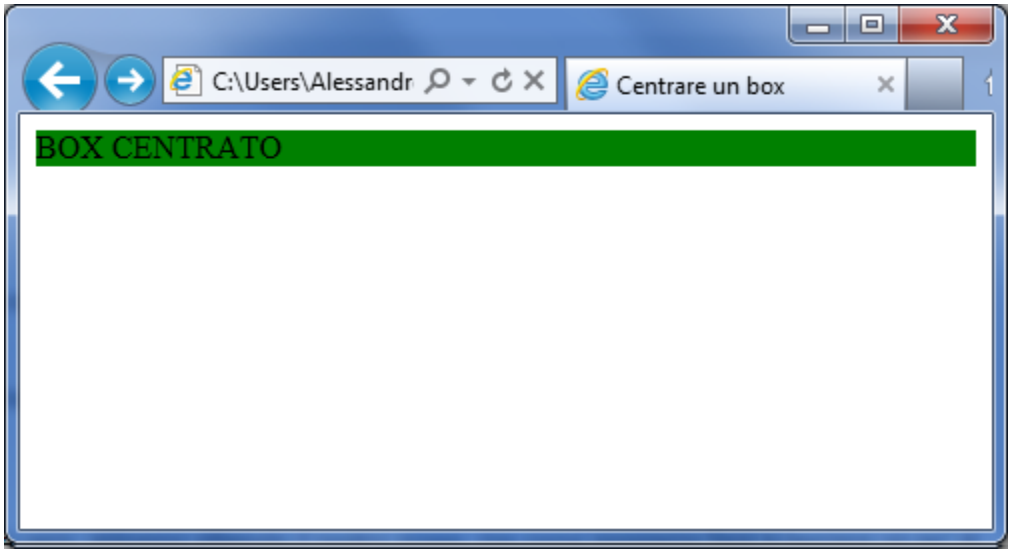
Abbiamo dunque imparato che **dando una larghezza fissa ad un box** (width:350px) **e impostandone al valore "auto" i margini destro e sinistro** (margin-left:auto; margin-right:auto;)**, il box mantiene la propria larghezza e si posiziona al centro dell'elemento che lo contiene**, nel nostro caso il box è contenuto nell'elemento <body>.

Centrare un box fluido

Ora proviamo a centrare un box fluido. Sappiamo che impostando i margini destro e sinistro ad "auto" teniamo il box al centro, quindi semplicemente eliminando la larghezza fissa, dovremmo già ottenere il nostro scopo.
Proviamo.

```
#centrato {
    background-color:green;
    margin-left:auto;
    margin-right:auto;
}
```

Il risultato ora è questo.

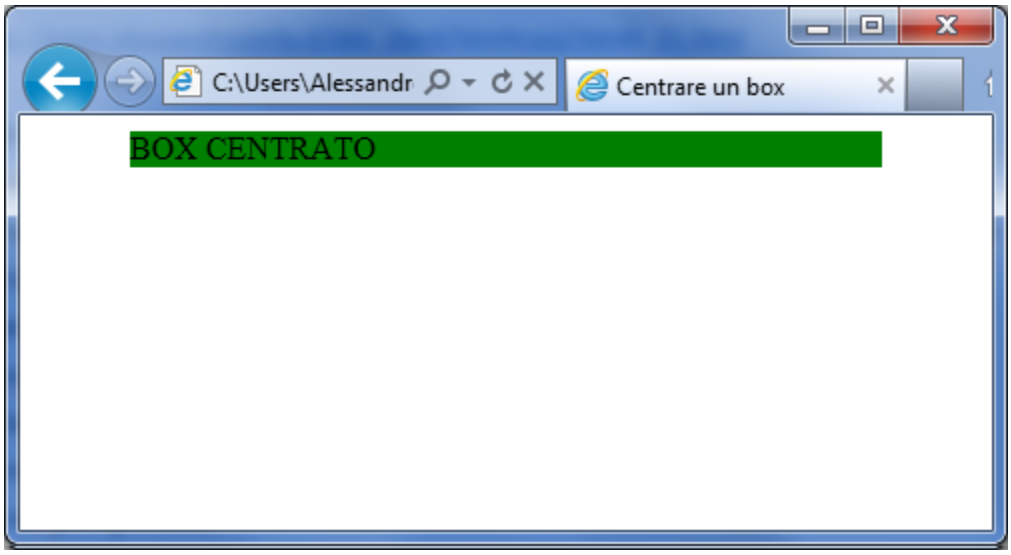


E possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_01.html

Si, è vero, abbiamo ottenuto il nostro scopo, cioè, adesso il box si allarga in base alla larghezza della finestra del browser, ma come possiamo fare a lasciare un po' di spazio tra il nostro box è i margini della finestra del browser?
L'importante è non dare al box una larghezza fissa. Quindi ci basta dargli una larghezza in percentuale. Ad esempio:

```
#centrato {
    width:80%;
    background-color:green;
    margin-left:auto;
    margin-right:auto;
}
```

Che produce infatti il seguente risultato.



Che possiamo vedere in azione qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_02.html

A differenza del box centrato a larghezza fissa, in quest'ultimo caso il box rimarrà al centro, ma non avrà una larghezza fissa, bensì essa aumenterà e diminuirà a seconda della dimensione della finestra.

Ora che siamo in grado di gestire un singolo box, possiamo pensare a gestirne due o più.

Layout a 2 colonne

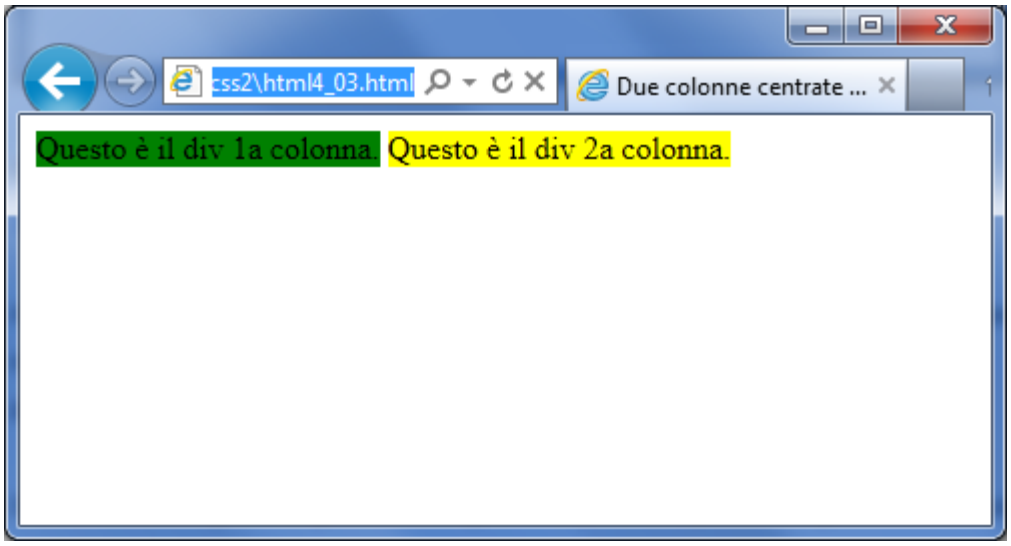
Creare un layout a 2 colonne, significa creare almeno 2 elementi .
Sappiamo però che il è un elemento di tipo "block" e sappiamo che un elemento di questo tipo non ammette altri elementi né alla sua destra né alla sua sinistra perché occupa tutto lo spazio in cui si trova e costringe ad andare a capo. Come possiamo fare dunque a posizionare due elementi sulla stessa "riga"?
La prima idea che viene in mente è quella di usare la proprietà "display" per trasformare i 2 da "block" a "inline" e quindi consentirne un posizionamento affiancato. E in effetti l'idea funziona, ma... è l'idea sbagliata. Vediamo perché.
Creiamo i nostri 2 :

```
<body>
    <div id="primaColonna">Questo è il div 1a colonna.</div>
    <div id="secondaColonna">Questo è il div 2a colonna.</div>
<body>
```

e il nostro codice CSS:

```
#primaColonna{
    display:inline;
    background-color:green;
}
#secondaColonna {
    display:inline;
    background-color:yellow;
}
```

Il risultato è il seguente.

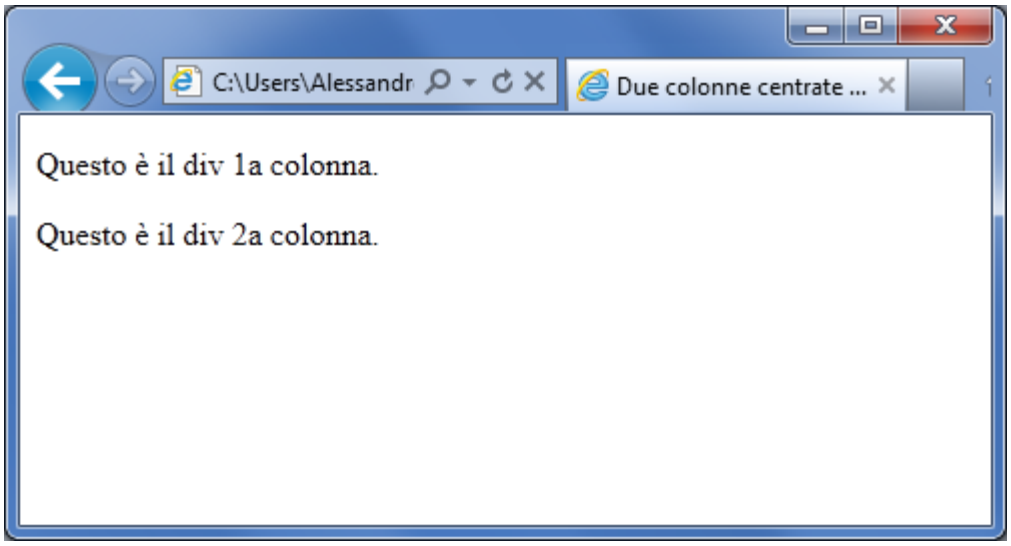


Cioè esattamente quello che volevamo. Quale è il problema?

Il problema è che quasi mai i 2 avranno all'interno del semplice testo! La realtà dice che conterranno paragrafi, immagini, altri è così via. Cosa succede se aggiungiamo elementi di tipo "block" all'interno dei nostri trasformati in tipo "inline"? Basta provare per scoprirlo! Inseriamo ad esempio i 2 testi in due paragrafi.

```
<body>
    <div id="primaColonna">
          <p>Questo è il div 1a colonna.</p>
    </div>
    <div id="secondaColonna">
          <p>Questo è il div 2a colonna.</p>
    </div>
<body>
```

Ed ecco qua cosa mostra il browser.



No, l'idea di trasformare in "inline" i 2 non può funzionare.

E allora?

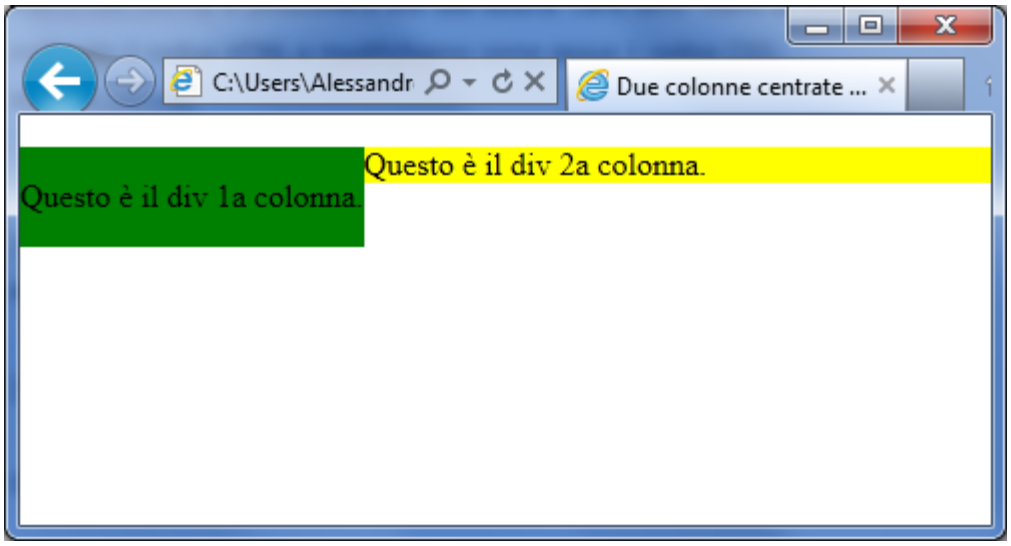
Il segreto si chiama "float".

Abbiamo detto e qui ribadiamo che con questa proprietà è possibile **rimuovere un elemento dal normale flusso del documento** e spostarlo su uno dei lati (destro o sinistro) del suo elemento contenitore. Il contenuto che circonda l'elemento scorrerà intorno ad esso sul lato opposto rispetto a quello indicato come valore di float.

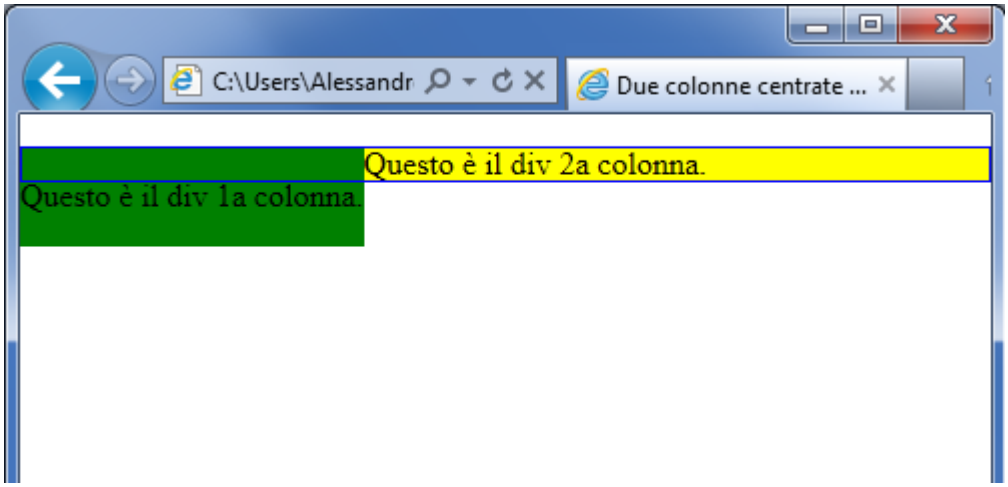
Quindi, se noi dichiariamo float:left il #primaColonna, automaticamente il <div #secondaColonna si posizionerà alla sua destra! Sarà vero? Mano al codice. Lasciamo inalterato il codice HTML e modifichiamo come segue il codice CSS:

```
#primaColonna{
    float:left;
    background-color:green;
}
#secondaColonna {
    background-color:yellow;
}
```

Ottenendo come risultato il seguente.

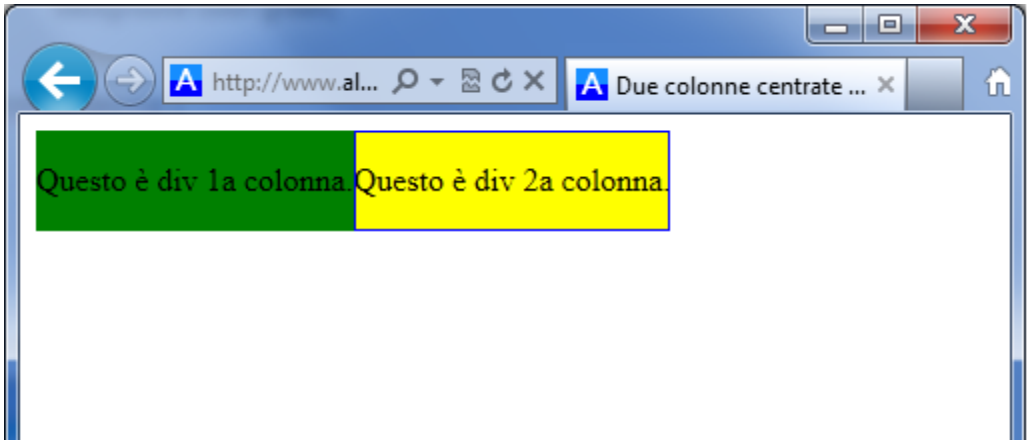


Si! E' proprio quello che volevamo! Ma... perché i testi sono sfalsati?
La risposta è sempre nella proprietà "float". Questa proprietà infatti ha posizionato il #primaColonna in alto a sinistra, costringendo **il contenuto** del secondo a disporsi alla sua destra. Il contenuto! Non il secondo . Il contenuto del secondo .
E quale è il contenuto del #secondaColonna? E' un elemento <p>. L'elemento <p> porta con se un margine superiore e inferiore ed è questo margine che crea il problema.
Osserviamo la seguente figura.



La linea blu mostra dove inizia il #secondaColonna.
Osserviamo con attenzione. Ci saremmo aspettati che il #secondaColonna iniziasse alla fine del #primaColonna. Invece no! Inizia tutto a sinistra! E non è un errore!
La proprietà "float" impostata sulla prima colonna infatti ha escluso la #primaColonna dal

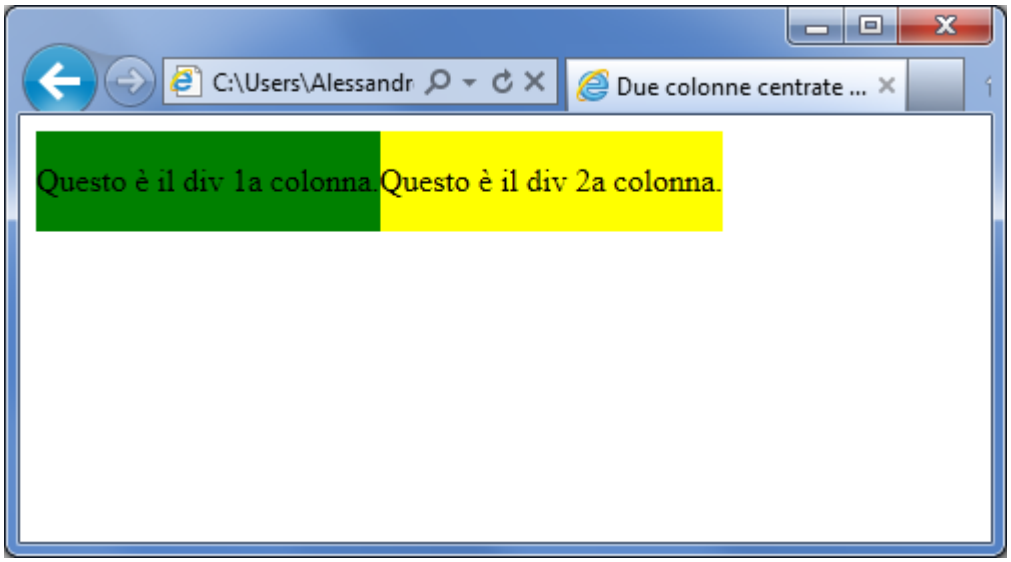
normale flusso della costruzione della pagina. Quindi il #secondaColonna la ignora e si comporta come se non esistesse. Questo non vale però per **il contenuto** della #secondaColonna il quale deve invece sottostare alle regole imposte dalla proprietà float impostata sulla #primaColonna. Tali regole impongono al contenuto della #secondaColonna di disporsi tutto intorno alla #primaColonna.
Ecco perché il codice che abbiamo scritto non funziona bene. Ma allora come facciamo ad allineare i testi?
Dobbiamo assegnare la proprietà "float:left" anche al #secondaColonna!
In questo modo la #secondaColonna sarà spostata anch'essa in alto a sinistra del suo box contenitore, ma troverà lo spazio già occupato dal #primaColonna e dovrà quindi posizionarsi alla destra di quest'ultimo.
Osserviamo adesso la seguente figura e la linea blu intorno al box giallo. Come possiamo notare essa non inizia più dove iniziava prima, ma inizia alla fine del box verde. Questa cosa fa la differenza.



Alla fine il nostro codice CSS sarà quindi:

```
#primaColonna{
    float:left;
    background-color:green;
}
#secondaColonna {
    float:left;
    background-color:yellow;
}
```

Con questo codice il risultato sarà alla fine.



E questa volta si! E' quello che vogliamo! Lo possiamo vedere in azione qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_04.html

Ma non abbiamo ancora finito! Anzi, veramente dobbiamo ancora iniziare! Il nostro obiettivo infatti è realizzare un layout centrato nella finestra del browser con 2 colonne di larghezza fissa, oppure due colonne di larghezza variabile! Fino ad ora abbiamo solo messo 2 colonne una di fianco all'altra!

Due colonne a larghezza fissa centrate

Ora l'intuito dovrebbe iniziare a venirci incontro...
Sappiamo come centrare un box e sappiamo come mettere 2 box uno di fianco all'altro. Unendo le due informazioni, dovremmo riuscire ad ottenere il nostro layout con 2 colonne a larghezza fissa centrate nella finestra del browser!
Includiamo quindi le nostre 2 colonne in un box che centreremo nella pagina del browser. Daremo a questo box una larghezza fissa e lo stesso faremo con i due box #primaColonna e #secondaColonna.

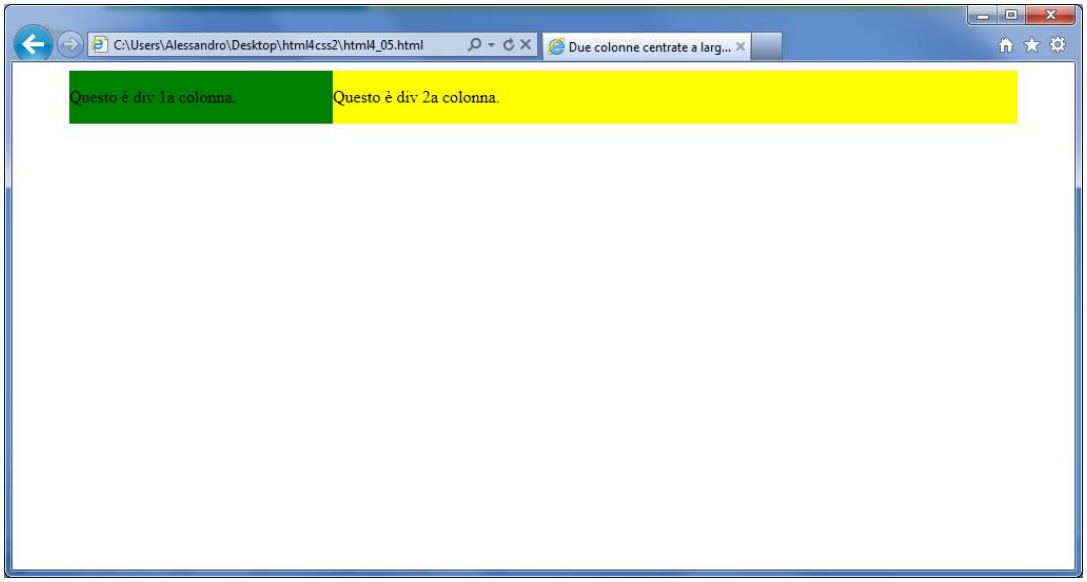
```
<body>
    <div id="centrato">
          <div id="primaColonna">
                <p>Questo è div 1a colonna.</p>
          </div>
          <div id="secondaColonna">
                <p>Questo è div 2a colonna.</p>
          </div>
```

```
    </div>
<body>
```

Il codice CSS sarà:

```
#centrato {
    width:900px;
    margin-left:auto;
    margin-right:auto;
}
#primaColonna{
    width:250px;
    float:left;
    background-color:green;
}
#secondaColonna {
    width:650px;
    float:left;
    background-color:yellow;
}
```

Il risultato è proprio lui!



Osserviamo come la larghezza del #centrato, 900px, è (deve essere) uguale alla somma delle 2 larghezze dei 2 #primaColonna (250px) e #secondaColonna (650px).
Possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_05.html

Due colonne a larghezza fluida centrate

Anche in questo caso l'intuito dovrebbe ormai indicarci la strada: **sostituire i px con le percentuali!** Quindi trasformare il nostro codice CSS, nel seguente:

```
#centrato {
    width:80%;
    margin-left:auto;
    margin-right:auto;
}
#primaColonna{
    width:25%;
    float:left;
    background-color:green;
}
#secondaColonna {
    width:75%;
    float:left;
    background-color:yellow;
}
```

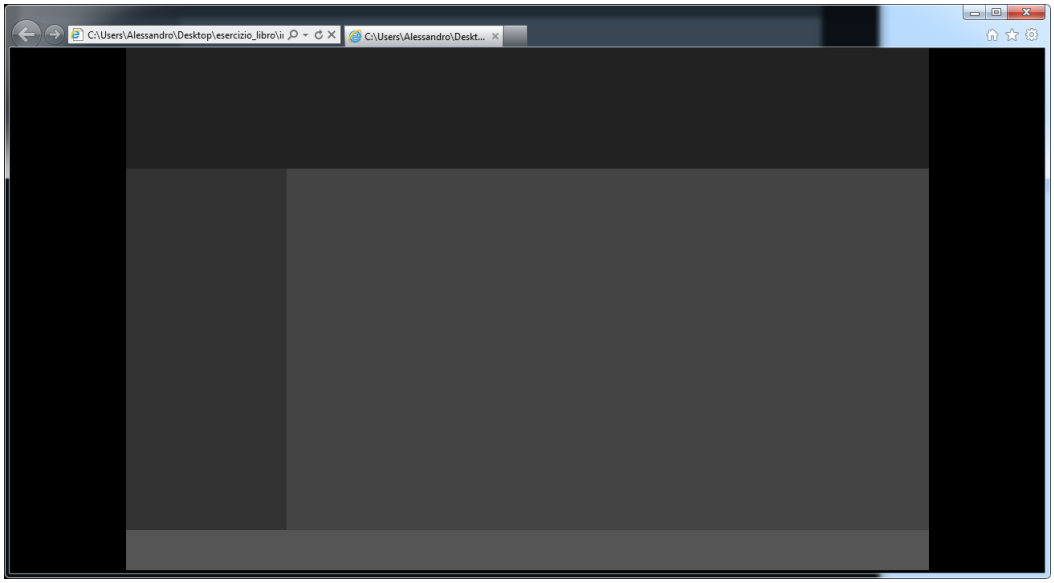
Facciamo attenzione alle percentuali!
Il box #centrato sarà l'80% del suo contenitore che è l'elemento <body>.
Le 2 colonne sono invece contenute nel box #centrato e quindi saranno rispettivamente il 25% e il 75% del box #centrato.
Possiamo vedere il codice in azione qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_06.html

Il prossimo passo sarà quello di aggiungere al nostro layout un header e un footer.

Layout completo a 2 colonne fisse centrate

Ormai ci siamo! Per avere un layout a 2 colonne completo ci manca giusto la testata (header) e il piè di pagina (footer). Qui realizzeremo solo un layout centrato a larghezza

fissa. Verrà lasciato al lettore l'implementazione del layout fluido (ossia la sostituzione dei pixel con le percentuali).
Alla fine del paragrafo saremo in grado di produrre il layout mostrato nella seguente figura.



Vediamo subito il codice HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
          <link rel="stylesheet" type="text/css"
                href="style/style_07.css" />
          <title>Layout a 2 colonne completo</title>
    </head>
    <body>
          <div id ="centrato">
                <div id="header"></div>
                <div id="primaColonna"></div>
                <div id="secondaColonna"></div>
                <div id="footer"></div>
          </div>
    </body>
</html>
```

Il codice CSS, da salvare nel file "style/style_07.css", sarà il seguente.

```
html, body {
    margin:0px;
    padding:0px;
    background-color:black;
}
#centrato {
    height:auto;
    width:1000px;
    margin-left:auto;
    margin-right:auto;
}
#header {
    background-color:#222222;
    height:150px;
    width:100%;
}
#primaColonna {
    background-color:#333333;
    float:left;
    height:450px;
    width:20%;
}
#secondaColonna {
    background-color:#444444;
    float:left;
    height:450px;
    width:80%;
}
#footer {
    background-color:#555555;
    clear:left;
    height:50px;
    width:100%;
}
```

Se adesso provassimo a visualizzare il nostro codice, tutto prenderebbe forma. Possiamo vederlo qui:

http://www.alessandrostella.it/lato_client/html4css2/html4css2_07.html

La struttura del sito appare già in modo chiaro. Se proviamo a espandere a tutto schermo il browser, il nostro sito rimane al centro della pagina. Esattamente come volevamo.
Eppure il codice scritto è davvero poco. Cerchiamo di analizzarlo.
Sul codice HTML c'è poco da dire, è abbastanza semplice. Il codice css invece va analizzato passo per passo.

```
html  body {
        margin:0px;
        padding:0px;
        background-color:black;
}
```

La prima riga

```
html  body {
```

comunica al browser gli elementi HTML a cui applicare le regole che seguono la parentesi graffa. Vediamole.

```
margin:0px;
```

Questa riga significa che i tag <html> e <body> devono annullare eventuali margini.
Ricordiamo che il margin rappresenta lo spazio tra il bordo del box e ciò che lo circonda.
Se non ce lo ricordiamo andiamo a rileggere il capitolo sul box model.

```
padding:0px;
```

Questa riga invece setta a zero la distanza tra il contenuto dei 2 tag e i bordi.

```
background-color:black;
```

Questa riga indica al browser di colorare di nero lo sfondo dei 2 tag.

```
#header {
        background-color:#222222;
        height:150px;
        width:100%;
}
```

L'unica proprietà da commentare è **background-color**.
Questa proprietà consente di assegnare un colore allo fondo del box a cui è applicata.

I colori nei fogli di stile vengono assegnati in base alla scala RGB (Red, Green, Blue), 2 cifre esadecimali per ogni colore primario. I colori primari sono il rosso, il verde e il blu. Quindi il codice #222222 significa R=22, G=22, B=22, cioè un po' di rosso, un po' di verde e un po' di blu. Il codice #000000 significa nero (R=0, G=0, B=0), mentre #FFFFFF significa bianco (R=FF, G=FF, B=FF). Tutti gli altri colori sono quindi compresi tra questi 2 valori.
Può capitare di trovare il codice colore indicato con solo 3 cifre invece di 6. Per esempio #222. Questo modo di scrivere è equivalente allo scrivere #222222 e si usa solo quando i numeri che compongono il codice del singolo colore sono uguali. Quindi #FFAABB equivale a #FAB e così via.

Proseguiamo nell'analisi del codice.

```
#centrato {
        height:auto;
        width:1000px;
        margin-left:auto;
        margin-right:auto;
}
```

Dovremmo ricordare che il cancelletto nel codice css serve per indicare l'id di un determinato elemento. In questo caso quindi il browser assegnerà le proprietà all'elemento con id="centrato".

height:auto;
E' importante sapere che un ha altezza nulla se non contiene alcun elemento HTML e resta quindi invisibile. Poiché il #container è l'unico a contenere altri , con l'impostazione height:auto; indichiamo al browser di dargli l'altezza massima necessaria a contenere l'ultimo elemento presente al suo interno.

width:1000px;
La larghezza del #centrato deve essere fissa: 1000 pixel!

margin-left:auto;
margin-right:auto;
Dovremmo ormai conoscere il significato di tali istruzioni: tenere il box al centro.

```
#footer {
```

```
        background-color:#555555;
        clear:left;
        height:50px;
        width:100%;
}
```

In questo caso la proprietà da evidenziare è la proprietà **clear**.
Questa proprietà si occupa di annullare gli effetti della proprietà "float". Al #secondaColonna è stata assegnata la proprietà "flat:left". Ciò imporrebbe al #footer di posizionarsi alla destra del #secondaColonna. Tuttavia questo non è quello che vogliamo! Imponendo "clear:left" al #footer, imponiamo a tale elemento di non avere nessun elemento alla propria sinistra.

Ora che abbiamo provato le basi, procediamo inserendo elementi tipici di un sito reale: i menù di navigazione e i contenuti.

Il menù di navigazione tramite css

Abbiamo visto come utilizzare HTML e CSS per creare un layout completo a due colonne. Abbiamo anche imparato che le sezioni di una pagina web prevedono quasi sempre una zona orizzontale in alto, che si chiama header e una, sempre orizzontale in basso, che si chiama footer.
Adesso, partendo dal layout centrato, impareremo come usare i fogli di stile per creare un menù di navigazione. Ne creeremo sia uno verticale, nella colonna di sinistra, sia uno orizzontale nell'header.

Menù css verticale

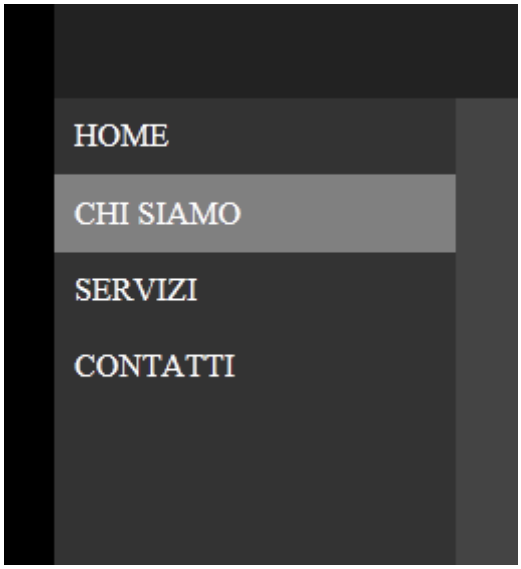
I menù di navigazione classici vengono creati usando i tag <ul> e <li> (ricordiamo che ul=unordered link, li=link). Apriamo quindi con Notepad++ il nostr file html, posizioniamoci nella #primaColonna e inseriamo la sezione che conterrà il nostro menù:

```
<div id="primaColonna">
    <div id="menuVerticale">
          <ul>
                <li><a href="#">home</a></li>
                <li><a href="#">chi siamo</a></li>
                <li><a href="#">servizi</a></li>
                <li><a href="#">contatti</a></li>
          </ul>
    </div>
```

```
</div>
```

Già adesso, salvando il file e aprendolo con il browser, possiamo notare le nostre voci di menù posizionate esattamente dove le aspettiamo. Da notare il valore assegnato all'attributo href. Il cancelletto indica al browser di non fare nulla.
In pratica la nostra prima versione di menù di navigazione verticale è già pronta e la possiamo vedere in azione qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_08.html

Eseguiamo adesso un passo successivo, un'evoluzione estetica.



Il nostro obiettivo è quello di rendere cliccabile non solo il testo del tag <a> ma tutto il box che lo contiene. Inoltre vogliamo far cambiare il colore del box quando passiamo sul link con il puntatore del mouse.
E' tutto lavoro da foglio di stile.

```
#menuVerticale ul {
    margin:0px;
    padding:0px;
    list-style-type:none;
}
#menuVerticale ul a:link, ul a:visited{
    display:block;
    padding:10px;
    text-decoration:none;
    color:white;
```

```
    text-transform:uppercase;
}
#menuVerticale ul a:hover {
    background-color:grey;
}
```

Cioè?
La prima regola (#menuVerticale ul) si applica solo all'elemento <ul> contenuto nell'elemento #menuVerticale. Quindi non a tutti gli elementi <ul> ma solo a quelli contenuti nell'elemento con id="menuVerticale". La regola elimina margini, padding e toglie i puntini laterali alle voci di menù.
La seconda regola vale per gli elementi <a> contenuti all'interno di un elemento <ul> che deve essere a sua volta contenuto in un elemento con id="menuVerticale". Viene applicata a due pseudo-classi: a:link e a:visited (da notare la virgola che le separa) .
Questa seconda regola rende <a> di tipo "block". Abbiamo infatti visto che <a> è di tipo "inline", ciò significa (come dovremmo ricordare!) che il box che contiene l'elemento <a> occupa giusto lo spazio del testo, rendendo di fatto linkabile solo il testo. Poiché noi vogliamo colorare lo sfondo di tutta la riga, dobbiamo rendere questo elemento di tipo "block" in modo che il suo box occupi tutta la riga.
La proprietà "text-decoration" si riferisce alla sottolineatura tipica del testo dei link. Impostarla su "none" significa che non vogliamo venga mostrata.
La proprietà "text-transform" impostata su "uppercase" rende il testo tutto in maiuscolo.
Infine l'ultima regola si applica sulla pseudo-classe "a:hover", ossia quando il mouse passa sopra l'elemento. In questo caso vogliamo che il colore di sfondo diventi grigio.

Possiamo ritenerci abbastanza soddisfatti!
In poche righe di codice abbiamo ottenuto un buon risultato estetico che possiamo ammirare qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_09.html

Menù css orizzontale

Aggiungiamo adesso al nostro layout a 2 colonne un menù di navigazione orizzontale, all'interno della sezione #header.
Anche in questo caso vogliamo che cambi lo sfondo del box. Vogliamo inoltre che il menù sia allineato in basso rispetto alla sezione #header.
Scriviamo quindi il nostro menù nella sezione header.

```
<div id="header">
    <div id="menuOr">
          <ul>
                <li><a href="#">home</a></li>
                <li><a href="#">chi siamo</a></li>
                <li><a href="#">servizi</a></li>
                <li><a href="#">contatti</a></li>
          </ul>
    </div>
</div>
```

Ovviamente, per poter identificare univocamente il menù orizzontale, dobbiamo assegnargli un id univoco. In questo caso abbiamo scelto di chiamare l'elemento #menuOr. Salvando il file HTML e aprendolo nel browser possiamo osservare, nella sezione header, il classico elenco puntato verticale. E' però nostra intenzione mettere le varie voci di menù una di fianco all'altra, non una sotto l'altra. Se sono posizionate una sotto l'altra è perché <li> è di tipo "block"! Dovremo quindi usare la proprietà float per fare in modo che i vari <li> si dispongano uno di fianco all'altro, ma questo la sappiamo fare. Vediamo allora il codice css completo per ottenere il nostro menù orizzontale.

```
#menuOr {
    text-align:center;
    background-color:#666;
}
#menuOr ul {
    margin:0px;
    padding:0px;
    overflow:hidden;
    list-style-type:none;
}
#menuOr ul a:link , ul a:visited{
    display:block;
    padding:10px;
    text-decoration:none;
    color:white;
    text-transform:uppercase;
}
#menuOr ul a:hover {
```

```
    background-color:grey;
}
#menuOr ul li{
    float:left;
    width:120px;
}
```

La prima regola impone al contenitore, che abbiamo indicato con #menuOr, di tenere il testo allineato al centro e di avere uno sfondo di colore #666.
La seconda regola invece si riferisce all'elemento <ul> figlio di #menuOr e conosciamo già il significato di tutte le proprietà.
Anche la terza e la quarta regola ci sono familiari.
L'ultima regola invece usa, come abbiamo detto, la proprietà float per imporre agli elementi <li> di disporsi orizzontalmente.

Il risultato è visibile qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_10.html

Come possiamo osservare abbiamo ottenuto quasi tutto, manca giusto l'allineamento in basso del nuovo menù orizzontale. La prima cosa che ci verrebbe da fare per allineare il menù in basso è quella di aggiungere al #menuOr la proprietà bottom:0px. Già, peccato però che non cambi nulla. Il resta allineato in alto! Come mai? E' una domanda tipica e dipende da quello che abbiamo detto circa **la proprietà position** degli elementi HTML. Abbiamo infatti visto che tutti gli elementi HTML vengono posizionati in modo statico salvo diversa indicazione. Poiché noi non abbiamo dato alcuna diversa indicazione, tutti gli elementi della nostra pagina sono stati disposti in modo statico e abbiamo visto che la posizione statica non risponde alle variazioni delle proprietà left, right, top e bottom. Ecco perché impostando bottom:0px il menù non subisce alcun cambiamento di posizione. Cambiamo allora la proprietà position di #menuOr. Per fare un pochino di prove, assegniamo a #menuOr la posizione assoluta (position:absolute) e assegniamo adesso bottom:0px. A questo punto il menù viene allineato in basso alla pagina e non in basso alla sezione #header. Perché?
Anche qui abbiamo la risposta in quello che abbiamo studiato:
"Quando un elemento è posizionato in modo assoluto, esso (cioè il suo box) viene rimosso dal flusso del documento e posizionato in base alle coordinate fornite con le proprietà top, left, right o bottom. **Il posizionamento avviene sempre rispetto al**

primo elemento antenato (ancestor) che abbia un posizionamento diverso da static".

L'ultima frase spiega il motivo per il quale il nostro menù viene posizionato sul fondo della pagina. Infatti poiché tutti gli elementi antenati del #menuOr sono posizionati in modo statico, il browser continua a risalire fino al padre di tutti i nodi, posizionando #menuOr sul fondo di tutta la pagina!

Per ottenere un allineamento non sul fondo della pagina, ma sul fondo della sezione #header non dobbiamo fare altro che

- assegnare a header un posizionamento diverso da static, ossia relative (position:relative)
- assegnare a #menuOr un posizionamento assoluto con bottom a zero.

Quindi il nuovo codice css per la sezione header diventa:

```
#header {
    position:relative;
    background-color:#222222;
    clear:both;
    height:150px;
    width:100%;
}
```

mentre il div del menu orizzontale diventa

```
#menuOr {
    position:absolute;
    width:100%;
    bottom:0;
    text-align:center;
    background-color:#666;
}
```

Salviamo le modifiche e riproviamo a vedere la nostra pagina nel browser. Ora si che le cose vanno proprio come volevamo che andassero! Abbiamo quindi il nostro menù orizzontale con effetto roll-over. Il tutto con il solo utilizzo di HTML e CSS.

Ricapitolando, alla fine di tutto il nostro lavoro, la pagina HTML assume questo codice:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
          <title>Menù CSS orizzontale</title>
          <meta http-equiv="Content-Type" content="text/html;
                charset=utf-8">
          <link rel="stylesheet" type="text/css"
                href="style/style_11.css">
    </head>
    <body>
          <div id ="centrato">
                <div id="header">
                      <div id="menuOr">
                            <ul>
                                  <li><a href="#">home</a></li>
                                  <li><a href="#">chi
                                        siamo</a>
                                  </li>
                                  <li><a href="#">servizi</a>
                                  </li>
```

```
                                    <li><a href="#">contatti</a>
                                    </li>
                              </ul>
                        </div>
                  </div>
                  <div id="primaColonna">
                        <div id="menuVerticale">
                              <ul>
                                    <li><a href="#">home</a></li>
                                    <li><a href="#">chi siamo</a>
                                    </li>
                                    <li><a href="#">servizi</a>
                                    </li>
                                    <li><a href="#">contatti</a>
                                    </li>
                              </ul>
                        </div>
                  </div>
                  <div id="secondaColonna"></div>
                  <div id="footer">
                  </div>
            </div>
      </body>
</html>
```

mentre il foglio di stile è diventato il seguente.

```
html, body {
    margin:0px;
    padding:0px;
    background-color:black;
}
#centrato {
    height:auto;
    width:1000px;
    margin-left:auto;
    margin-right:auto;
}
#header {
```

```
    position:relative;
    background-color:#222222;
    height:150px;
    width:100%;
}
#primaColonna {
    background-color:#333333;
    float:left;
    height:450px;
    width:20%;
}
#menuVerticale ul {
    margin:0px;
    padding:0px;
    list-style-type:none;
}
#menuVerticale ul a:link, ul a:visited{
    display:block;
    padding:10px;
    text-decoration:none;
    color:white;
    text-transform:uppercase;
}
#menuVerticale ul a:hover {
    background-color:grey;
}
#menuOr {
    position:absolute;
    bottom:0px;
    width:100%;
    text-align:center;
    background-color:#666;
}
#menuOr ul {
    margin:0px;
    padding:0px;
    overflow:hidden;
    list-style-type:none;
}
```

```
#menuOr ul a:link , ul a:visited{
    display:block;
    padding:10px;
    text-decoration:none;
    color:white;
    text-transform:uppercase;
}
#menuOr ul a:hover {
    background-color:grey;
}
#menuOr ul li{
    float:left;
    width:120px;
}
#secondaColonna {
    background-color:#444444;
    float:left;
    height:450px;
    width:80%;
}
#footer {
    background-color:#555555;
    clear:left;
    height:50px;
    width:100%;
}
```

Possiamo vedere il tutto in azione qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_11.html

Le gestione dei contenuti

Dopo aver inserito un menù di navigazione, il nostro ipotetico sito ha bisogno dei contenuti veri e propri. I contenuti vengono generalmente mostrati al centro della pagina, essendo di massima importanza. Cerchiamo quindi di inserire nel nostro sito un contenuto al quale vogliamo assegnare un titolo, un testo e una barra di separazione che divida visivamente titolo e testo.
Portiamoci nel #secondaColonna e aggiungiamo quanto segue:

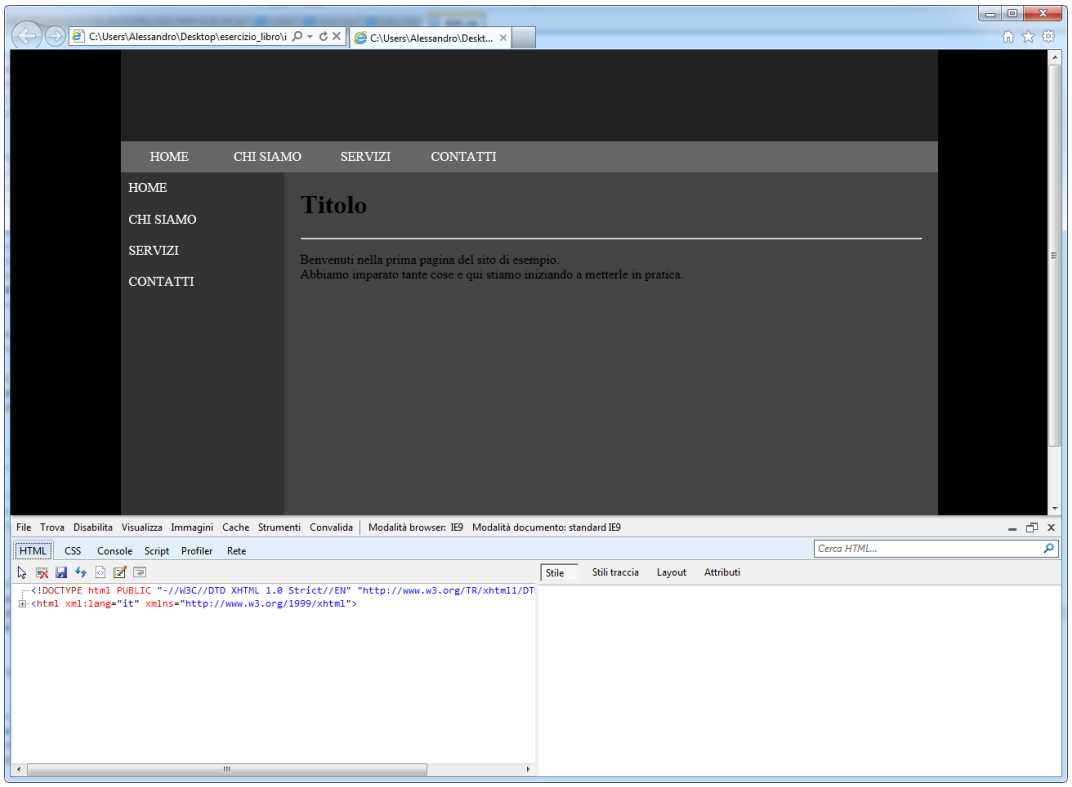
```
<div id="secondaColonna">
    <h1>Titolo</h1>
    <hr>
    <p>Paragrafo</p>
</div>
```

Da notare il tag <hr> che rappresenta una linea divisoria: non ha un tag di chiusura! In HTML4 è così.
Possiamo osservare il risultato qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_12.html

Gli strumenti di controllo

Quasi tutti i moderni browser integrano al loro interno risorse per gli sviluppatori, in modo che sia possibile effettuare correzioni e controlli al codice HTML e CSS direttamente dal browser. Per accedere a tali risorse basta **premere F12** in una qualsiasi finestra del browser.
Noi qui per semplicità useremo Internet Explorer 9, ma è possibile eseguire gli stessi controlli con tutti gli altri browser (Chrome, Firefox, Safari, ecc) sempre premendo F12.

Apriamo quindi la nostra pagina html in Internet Explorer 9 e premiamo F12. Ci dovremmo trovare di fronte a qualcosa del genere.



Potrebbe accadere che la console di controllo, che appare premendo F12, venga mostrata non in basso alla pagina, ma in un nuova finestra. In tal caso ci basta premere CTRL+P per ritornare nella situazione dell'immagine.

In basso a sinistra abbiamo tutto il codice HTML. Premendo sul + possiamo infatti espandere il tag <html> e vedere i tag in esso contenuti e così via. Se selezioniamo un tag, sulla destra compariranno le proprietà CSS associate a tale tag. Non solo. Possiamo anche verificare in tempo reale eventuali modifiche apportate alle proprietà del foglio di stile.

Per esempio, se nella parte di sinistra selezioniamo il div #centrato, sulla destra compraririanno tutte le proprietà CSS ad esso associate e potremo modificarle in tempo reale semplicemente cliccandoci sopra e inserendo i nuovi valori, oppure togliendo la spunta presente alla sinistra di ogni proprietà, "spegnendo" la proprietà.

La cosa utile è che **tutte le modifiche che apportiamo tramite questa tecnica non modificano il foglio di stile**. Ci basterà aggiornare la pagina (F5) per cancellare tutte le modifiche apportate e ritornare alla nostra pagina web originale.

Questo strumento è didatticamente molto utile perché mostra in tempo reale quali effetti avrebbero le nostre eventuali modifiche sul layout della pagina. Inoltre ci costringe a riflettere sul perché accade ciò che vediamo.

Oltre a tale strumento di controllo è importante sapere che il browser è in grado di mostrare il codice HTML della pagina visualizzata. Basta cliccare su un punto vuoto della pagina con il tasto destro del mouse e scegliere "HTML" oppure "Visualizza sorgente pagina".

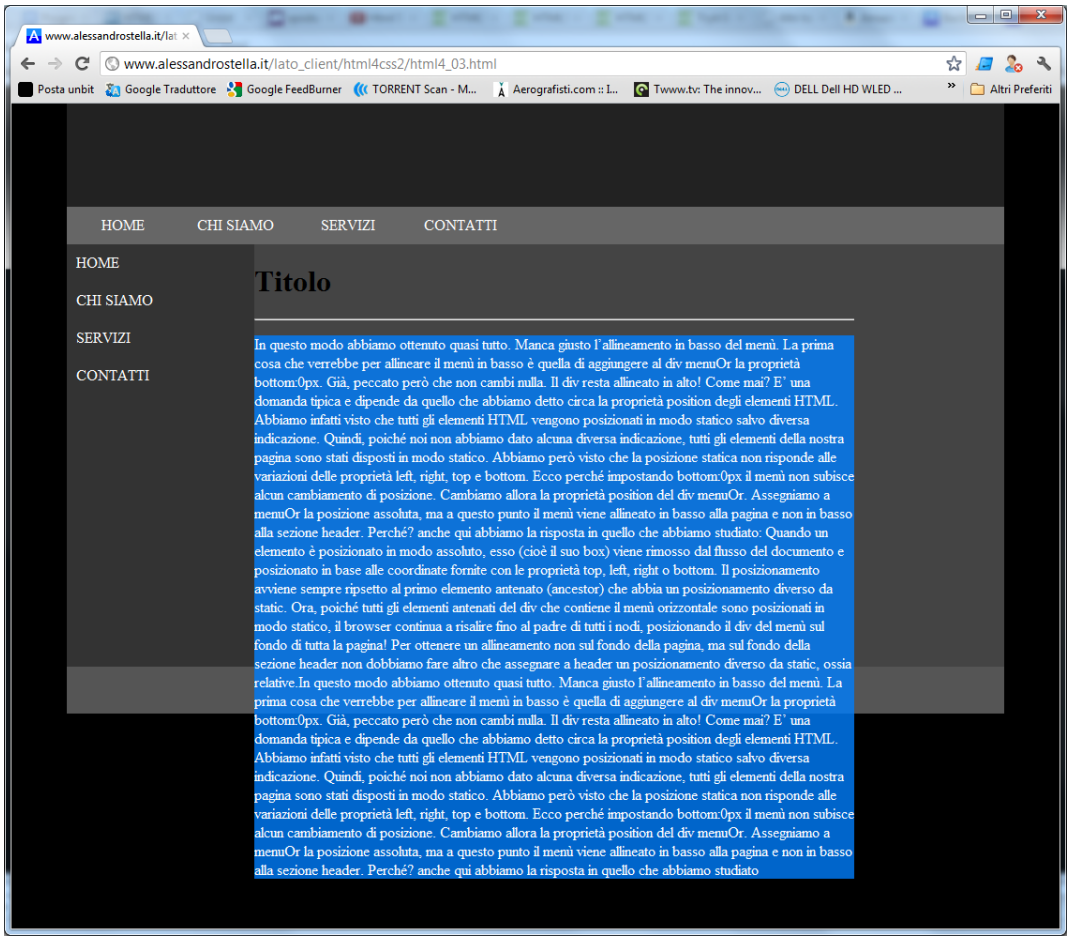
Infine è importante avere un riscontro ufficiale di quello che scriviamo. Il nostro codice può essere più o meno valido secondo gli standard w3c! Per sapere se il nostro codice HTML è ben scritto, possiamo (dobbiamo) usare un validatore messo a disposizione dal w3c:
http://validator.w3.org/

Allo stesso modo è opportuno controllare il codice CSS. Anche in questo caso esiste un apposito validatore w3c:
http://jigsaw.w3.org/css-validator/

E' buona prassi controllare spesso il parere di questi validatori. Impareremo tante cose dai loro controlli.

Gli errori commessi

Nel precedente codice abbiamo volutamente commesso alcuni errori. Errori che tipicamente si incontrano durante il primo approccio con HTML e CSS.
Ammettiamo per un momento che l'unico paragrafo presente in tutto il sito contenga un testo molto lungo, abbastanza da riempire tutto lo spazio a disposizione e andare oltre. Cosa accadrebbe?
Il risultato è mostrato nella seguente figura (nella quale è stato volutamente evidenziato il testo lungo).



E' possibile testare il risultato qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_13.html

Sicuramente non è quello che volevamo ottenere. Proprio per niente!
E adesso?
Possiamo uscirne in due modi. Uno è abbastanza banale, ma non è usato praticamente in nessun sito web: comunicare al #secondaColonna di mostrare una barra di scorrimento quando il suo contenuto eccede lo spazio a disposizione. Per fare questo ci basta usare **la proprietà overflow** impostandola su auto, trasformando quindi il codice CSS da così:

```
#secondaColonna {
    background-color:#444444;
    float:left;
    height:450px;
    width:80%;
}
```

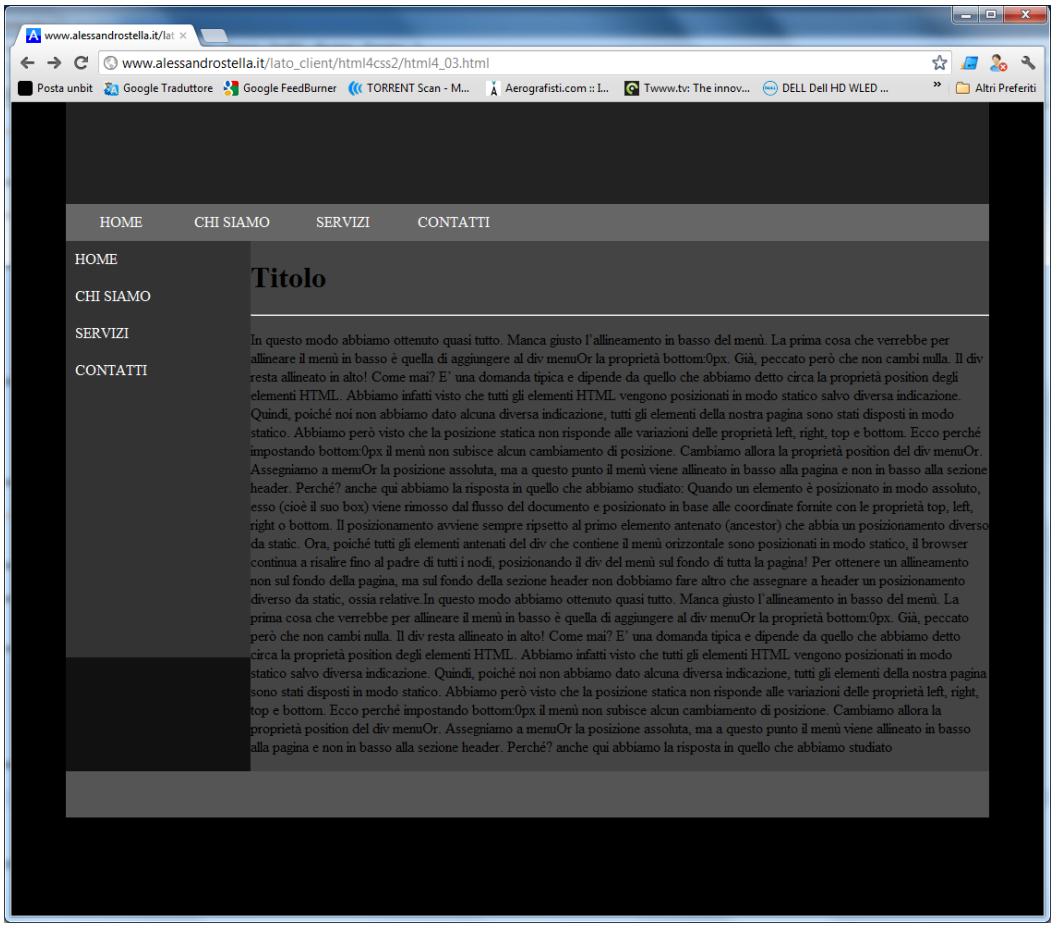
a così:

```
#secondaColonna {
    background-color:#444444;
    float:left;
    height:450px;
    width:80%;
    overflow:auto;
}
```

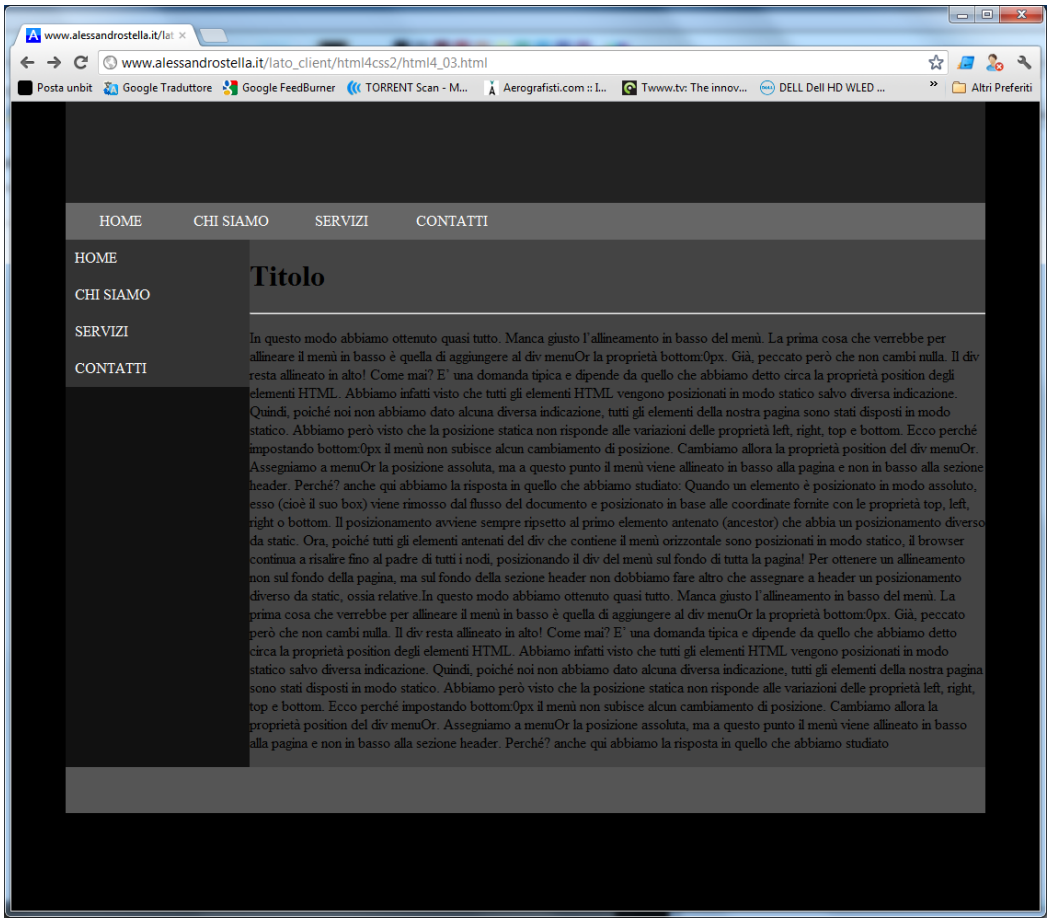
L'altro modo è più complicato, ma è quello standard: comunicare al #secondaColonna di allungarsi quando il suo contenuto eccede lo spazio dedicato. Il #secondaColonna non è stato istruito in modo tale da allungarsi in base al proprio contenuto, ma ha invece istruzioni per avere una determinata altezza, fissa, precisamente 450px. Quindi portiamoci nel codice CSS di #secondaColonna e modifichiamo l'altezza (height) del , da 450px ad "auto", ottenendo:

```
#secondaColonna {
    background-color:#444444;
    float:left;
    height:auto;
    width:80%;
}
```

Salviamo il file e ricarichiamo la pagina.



Va decisamente meglio, con il valore "auto" abbiamo detto al #secondaColonna di allungarsi in base al proprio contenuto. Essendo il testo molto lungo, il si è quindi allungato per contenerlo tutto, costringendo il "footer" a spostarsi verso il basso.
Ma... cosa è quel rettangolo nero in basso a sinistra, tra il footer e il menù laterale?
Ma certo! E' il #primaColonna! Dobbiamo fare allungare anch'esso! E quindi andiamo a modificare il codice #primaColonna assegnando "auto" alla proprietà height, ma... sorpresa! Dopo tale modifica, non otteniamo il risultato sperato!



Il rettangolo nero sulla sinistra è addirittura diventato più grande!

A rifletterci bene il browser sta facendo esattamente quello che gli abbiamo detto. Assegnando "auto" all'altezza di un infatti diciamo al browser di adeguare l'altezza di quel al contenuto in esso presente. Il #primaColonna contiene il menù di navigazione e tale menù termina proprio con il testo "CONTATTI". L'altezza del #primaColonna quindi è stata correttamente adeguata.

E allora?

Come facciamo ad allungare quel fino a farlo arrivare al footer?

Questo è uno dei più classici problemi dei novelli, ma stiamo per risolverlo!

La soluzione non c'è.

Non possiamo in alcun modo far allungare quel fino a farlo arrivare al footer.

Dobbiamo giocare d'astuzia! Invece di dare il colore #333 al #primaColonna, daremo quel colore al #centrato, lasciando senza colore di sfondo (quindi trasparente) il #primaColonna.

In questo modo nella colonna di sinistra non mostreremo il colore del #primaColonna, ma del sottostante #centrato, ottenendo l'effetto desiderato!

Quindi modifichiamo il codice del #container da così:

```
#centrato {
    background-color:#111111;
    height:auto;
    width:1000px;
    margin-left:auto;
    margin-right:auto;
}
```

a così:

```
#centrato {
    background-color:#333;
    height:auto;
    width:1000px;
    margin-left:auto;
    margin-right:auto;
}
```

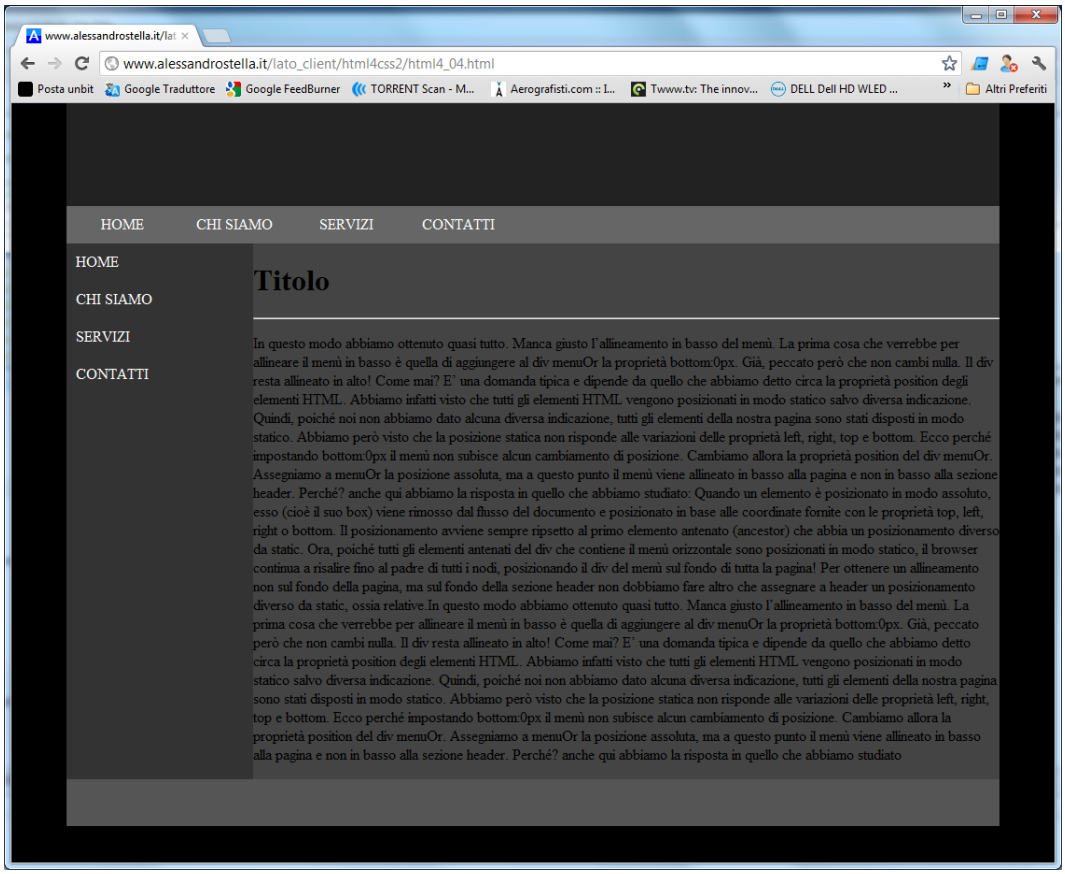
E modifichiamo il codice del #primaColonna da così:

```
#primaColonna {
    background-color:#333333;
    float:left;
    height:450px;
    width:20%;
}
```

a così:

```
#primaColonna {
    float:left;
    height:auto;
    width:20%;
}
```

Salviamo il foglio di stile, ricarichiamo la pagina e... finalmente quello che volevamo!



Possiamo testare il codice qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_14.html

Layout a 2 colonne: un caso particolare

Dovremmo essere in grado di produrre un layout liquido a due colonne (ci basta cambiare l'unità di misura della larghezza dei #centrato, #primaColonna e #secondaColonna portandola da pixel a percentuali), ma potremmo trovare difficoltà qualora volessimo avere la prima colonna a larghezza fissa, mentre la seconda a larghezza liquida.
Affronteremo questo problema in quest'ultimo paragrafo.
Tenere a larghezza fissa la prima colonna e lasciare libera la larghezza della seconda colonna, porta subito due problemi:

1. se restringiamo troppo la finestra del browser, la seconda colonna non si dispone alla destra della prima, ma scivola sotto
2. poiché la larghezza della seconda colonna è liquida, allargando la finestra del browser viene inesorabilmente prodotto uno spazio alla destra della

#secondaColonna; spazio che aumenta all'aumentare della larghezza della finestra del browser.

Possiamo osservare i problemi indicati qui:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_15.html

Come è possibile?
Il problema è causato dalla combinazione della proprietà float con la proprietà width.
La #primaColonna ha infatti il seguente codice CSS:

```
#primaColonna {
    float:left;
    width:250px;
}
```

mentre la #secondaColonna il seguente:

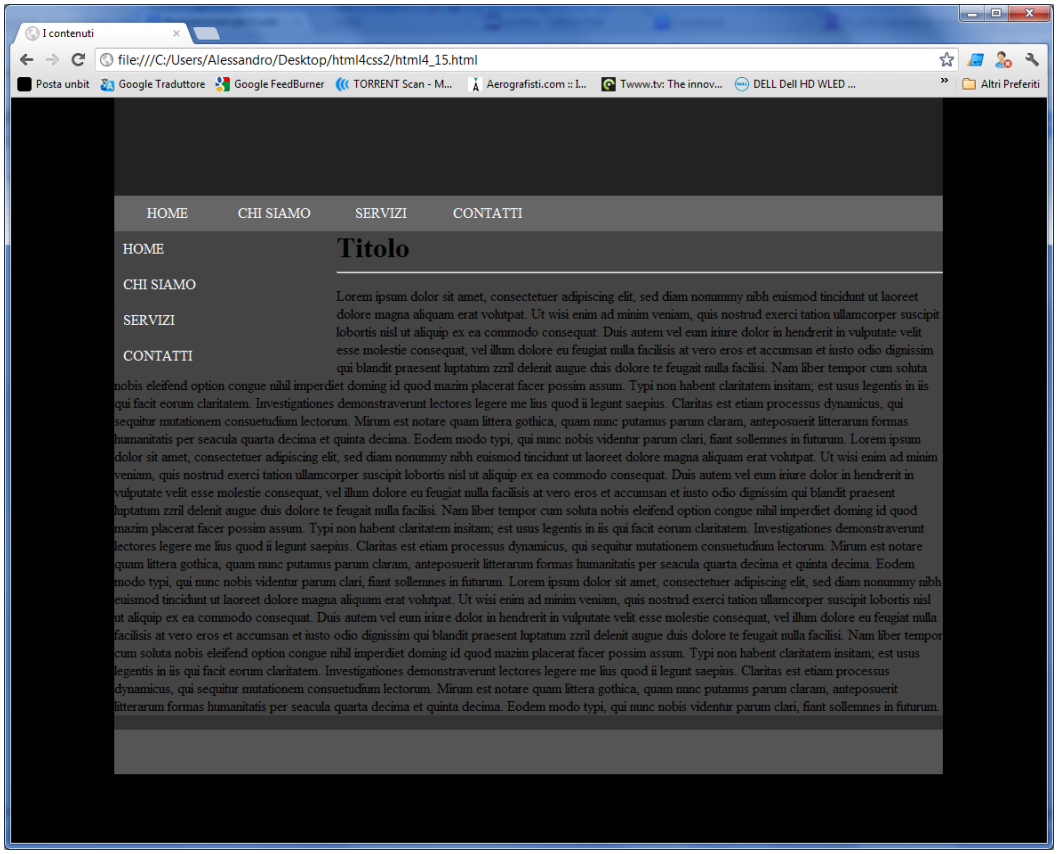
```
#secondaColonna {
    background-color:#444444;
    height:auto;
    float:left;
    width:70%
}
```

La proprietà float:left della #secondaColonna impone al #secondaColonna di posizionarsi in alto a sinistra rispetto al #centrato. Lì però trova il #primaColonna e quindi la #secondaColonna si posiziona subito alla destra della #primaColonna. Ma cosa succede se nel #centrato non c'è spazio per entrambi i #primaColonna e #secondaColonna?
Succede che la #secondaColonna viene spostata sotto alla #primaColonna.
Ecco quindi perché ad un certo punto la #secondaColonna scivola sotto la #primaColonna.
Abbiamo quindi due problemi:

1. la #secondaColonna non deve scivolare sotto la #primaColonna
2. la #secondaColonna deve allargarsi per tutto lo spazio alla destra della #primaColonna

Come risolvere?

Innanzitutto il float:left della #secondaColonna non serve più, in quanto la #primaColonna avrà una larghezza fissa e lascerà quindi ampio spazio alla sua destra. A questo punto **il contenuto** (ricordiamo?) della #secondaColonna, senza la proprietà float:left, si disporrà intorno alla #primaColonna ottenendo un risultato simile al seguente.



Questo però non è ancora quello che vogliamo.
Ora ci tocca tenere la #secondaColonna a 250px di distanza dal bordo sinistro del #centrato. Perché 250px? Perché questa è la larghezza fissa che abbiamo dato alla #primaColonna.
Per fare questo ci basta aggiungere un margine sinistro di 250px alla #secondaColonna, il cui codice CSS diventa:

```
#secondaColonna {
    background-color:#444444;
    height:auto;
    margin-left:250px;
}
```

Il risultato finale possiamo osservarlo qui:

http://www.alessandrostella.it/lato_client/html4css2/html4css2_16.html

Osservando il risultato ottenuto, possiamo notare però un ultimo problema: uno spazio finale tra la #secondaColonna e il #footer. Da dove viene fuori?
Questo problema lo abbiamo già affrontato: l'elemento <p>. Tale elemento infatti prevede un margine superiore e uno inferiore. Il margine inferiore sta creando problemi. Sarebbe quindi opportuno scrivere una regola css che comunichi al browser di eliminare il margine inferiore dell'elemento <p> figlio dell'elemento #secondaColonna... ma questa modifica viene lasciata al lettore.

Bene.
Giocando con , float, width e margin saremo in grado di produrre praticamente qualsiasi layout con HTML4, e CSS.

6
HTML 5

HTML5 è il futuro del web e quindi il web del futuro.
Un futuro non troppo lontano, visto che molti browser moderni già supportano alcune funzionalità previste da quello che sarà il nuovo standard w3c.
La precedente versione di HTML, cioè HTML 4.01 tuttora in uso in tutto il web, venne rilasciata nel 1999. In tutti questi anni il web è cambiato moltissimo. HTML 5 adegua le funzionalità di HTML a questi cambiamenti.
Attenzione però! Ad oggi, cioè Agosto 2012, HTML 5 non è ancora uno standard, si trova in fase di definizione e per molti anni ancora ci saranno vecchi browser incapaci di comprendere il codice HTML 5. L'uso di HTML 5 nei propri progetti deve quindi essere adeguatamente valutato.

HTML 5 non è un semplice aggiornamento di HTML 4. HTML 5 rappresenta una vera e propria rivoluzione in ambito web per diversi motivi, tra i quali non possiamo non citare i seguenti:

- Aggiunto l'elemento <canvas> per il disegno 2D e 3D
- Aggiunti gli elementi <video> e <audio> per la riproduzione multimediale
- Supporto per la memorizzazione locale
- Nuovi elementi per indicare particolari tipi di contenuto che assumono valore semantico, come <article>, <footer>, <header>, <nav>, <section>
- Nuovi controlli per i form, come calendar, date, time, email, url, search
- Geolocalizzazione, web storage, microdata...

Anche il doctype è cambiato. Con HTML 5 i documenti dovranno iniziare con il seguente codice (ricordiamo che è case-insensitive, cioè non fa differenza tra maiuscole e minuscole):

<!DOCTYPE HTML>

Ecco un esempio di documento conforme alle nuove specifiche HTML 5:

```
<!doctype html>
<html>
    <head>
          <meta charset="UTF-8">
          <title>Documento d'esempio</title>
    </head>
    <body>
          <p>Paragrafo d'esempio</p>
    </body>
</html>
```

Ogni informazione riportata nel presente capitolo è stata ripresa direttamente dal sito ufficiale w3c. In particolare dai seguenti articoli:
http://dev.w3.org/html5/spec/single-page.html
http://dev.w3.org/html5/html4-differences/

L'approccio a questo nuovo linguaggio sarà simile a quello utilizzato per HTML 4: teroia e pratica. Tuttavia visto l'attuale scarso supporto ad HTML 5 dei browser più diffusi, non ha senso entrare in eccessivi dettagli di tale linguaggio. Non rinunceremo però a realizzare un completo layout usando i nuovi elementi e la nuova semantica messi a disposizione da HTML 5.

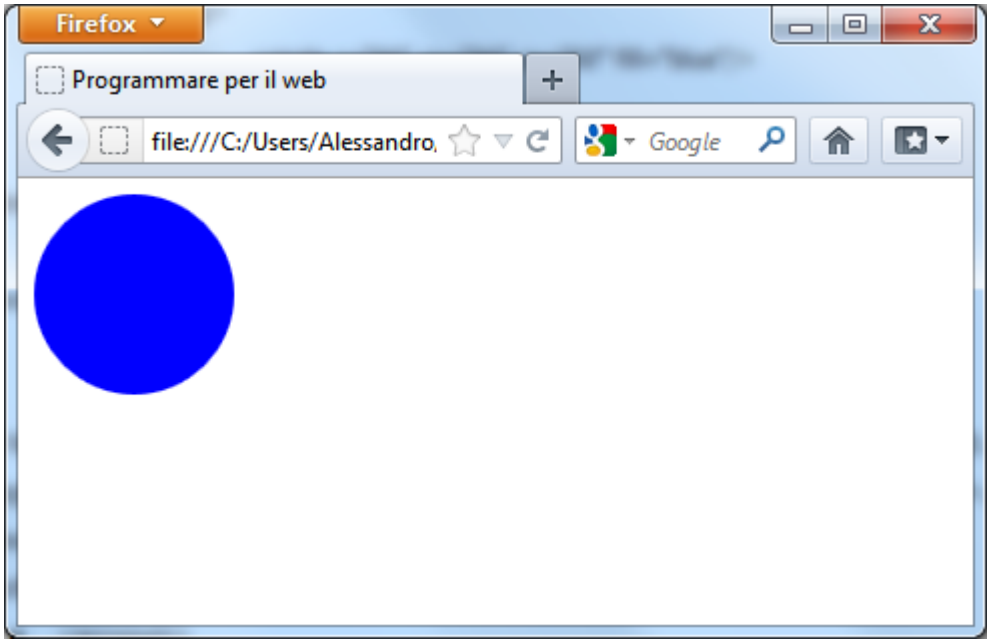
Il disegno e la matematica

La nuova sintassi di HTML 5 permette di usare **elementi MathML** e **SVG** all'interno di un documento. Ecco un esempio molto semplice che utilizza l'elemento <svg> per disegnare un cerchio blu:

```
<!doctype html>
<html>
    <head>
          <title>SVG in text/html</title>
    </head>
```

```
    <body>
        <svg>
            <circle r="50" cx="50" cy="50" fill="blue"/>
        </svg>
    </body>
</html>
```

Ed ecco il risultato.



Un quesito che prima o poi ci faremo durante l'uso di HTML5 è il seguente: **quale è la differenza tra gli elementi <canvas> e <svg>?**
Entrambi gli elementi promettono di farci disegnare sulla nostra pagina web. Ma ci sono diverse differenze tra i due, prima fra tutte quella che **<canvas> produce immagini bitmap, mentre <svg> produce immagini vettoriali**! La differenza tra i due tipi di immagine è molto importante. Semplificando il concetto possiamo dire che un'immagine bitmap tende a sgranare quando viene ingrandita, mentre un'immagine vettoriale no. Ma questa non è certamente l'unica differenza tra i due elementi.
<canvas>

1. Buon supporto tra i browser più importanti
2. La manipolazione dei pixel delle immagini più approfondito
3. Difficile gestire gli eventi
4. E' più performante (rispetto a <svg>)

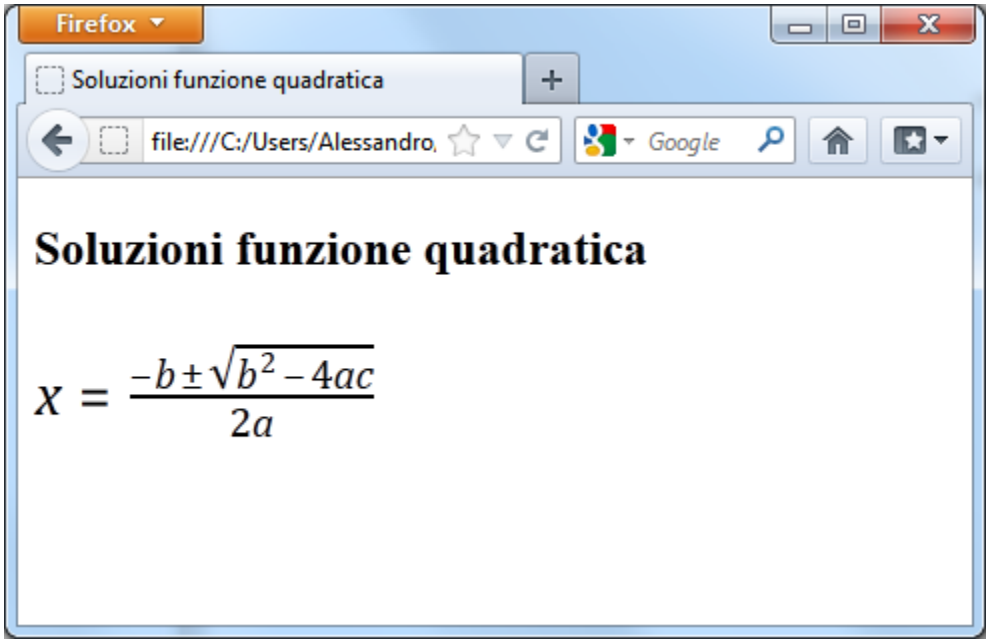
5. Il disegno avviene tramite linguaggio di scripting (tipicamente JavaScript)

<svg>

1. Non sempre supportato da tutti i browser
2. Ottimo per manipolare immagini vettoriali
3. Gestione degli eventi semplice
4. Meno efficiente del <canvas>
5. E' possibile disegnare e gestire immagini complesse grazie anche a prodotti di larga diffusione

Per quanto riguarda **MathML**, esso ha addirittura un'intera sezione del w3c dedicata: http://www.w3.org/Math/

Il suo uso è da considerasi di nicchia perché è specificatamente pensato per notazioni di tipo scientifico. Ecco un semplice esempio di cosa si può fare scrivendo solo testo (cioè codice HTML).



Ed ecco, per dovere di cronaca, il relativo codice.

```
<!DOCTYPE html>
<html>
    <head>
          <title>Soluzioni funzione quadratica</title>
    </head>
    <body>
```

```
    <h2>Soluzioni funzione quadratica</h2>
          <p style="font-size:22pt">
                <math>
                      <mi>x</mi>
                      <mo>=</mo>
                      <mfrac>
                            <mrow>
                                  <mo form="prefix">-</mo>
                                  <mi>b</mi>
                                  <mo>±</mo>
                                  <msqrt>
                                        <msup>
                                              <mi>b</mi>
                                              <mn>2</mn>
                                        </msup>
                                        <mo>-</mo>
                                        <mn>4</mn><mo>&it;</mo>
                                        <mi>a</mi><mo>&it;</mo>
                                        <mi>c</mi>
                                  </msqrt>
                            </mrow>
                            <mrow>
                                  <mn>2</mn>
                                  <mo>&it;</mo>
                                  <mi>a</mi>
                            </mrow>
                      </mfrac>
                </math>
          </p>
    </body>
</html>
```

HTML 4: elementi in pensione

Per fortuna quasi tutto quello che abbiamo imparato su HTML 4 è valido anche in HTML 5. Bisognerà tuttavia modificare alcune abitudini, in particolare nell'uso degli elementi , come vedremo in seguito. Inoltre alcuni elementi HTML 4 non sono più presenti nelle specifiche di HTML 5. In particolare sono stati eliminati:

- <acronym>

- <applet>
- <basefont>
- <big>
-
- <dir>
- <font>
- <frame>
- <frameset>
- <noframes>
- <strike>
- <tt>
- <u>

A fronte di questi elementi non più presenti, HTML 5 ne aggiunge di nuovi e molto significativi.

La nuova semantica

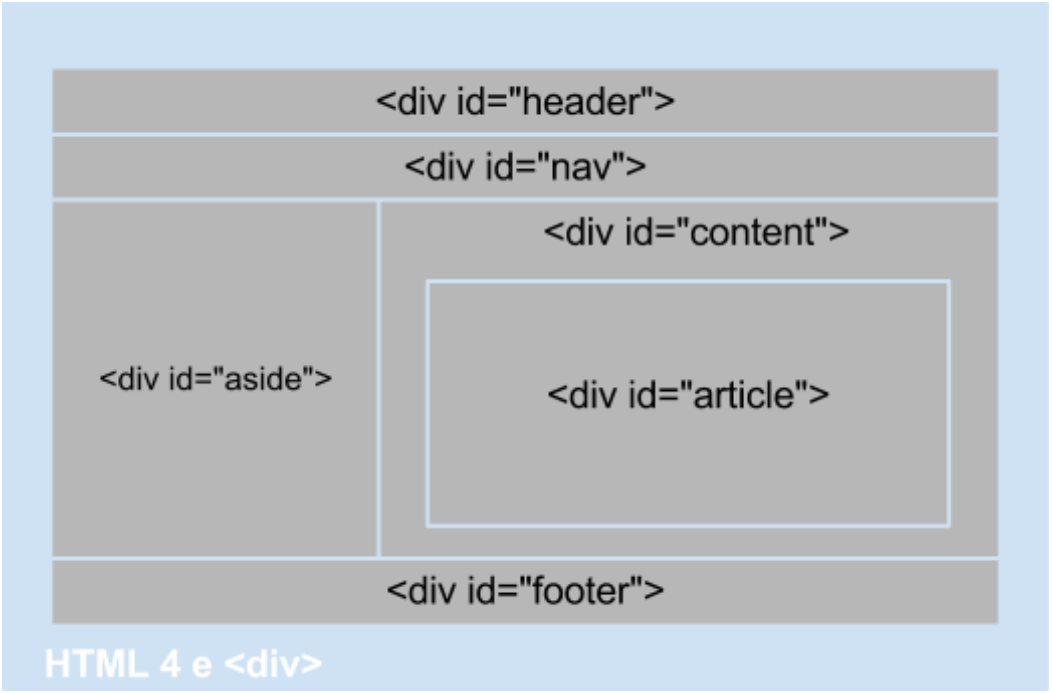
La semantica degli elementi HTML è uno dei concetti introdotti con l'arrivo di HTML 5. I nuovi elementi HTML, già inseriti nelle specifiche, daranno un significato ben preciso al proprio contenuto. Per esempio, l'elemento <article> conterrà di certo un articolo e non un menù di navigazione. Questo modo di intendere gli elementi aiuterà moltissimo i motori di ricerca nel lavoro di comprendere semanticamente il contenuto di una pagina web, oltre a modificare drasticamente l'uso dell'elemento . E' infatti diventato fin troppo evidente che in HTML 4 si è fatto e si fa un uso spasmodico dell'elemento tanto che i motori di ricerca fanno sempre più fatica a dare la giusta importanza alle sezioni in cui è divisa una pagina web. L'attributo "id" che viene spesso usato per determinare in qualche modo il contenuto del , è usato in modo non uniforme. Ad esempio per un italiano scrivere

significa che in quel è contenuto un articolo. Un programmatore inglese con lo stesso obiettivo scriverebbe

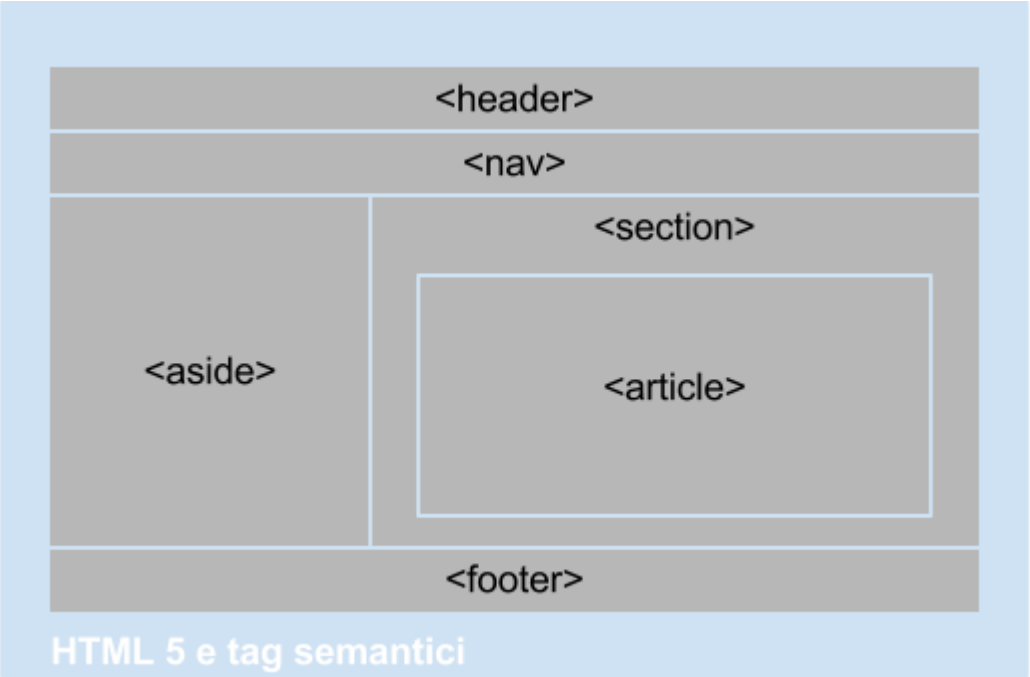
Tutte queste differenze semantiche rendono difficile per un motore di ricerca comprendere il reale contenuto di quel . Diventa quindi importante trovare un modo per uniformare il codice.
L'introduzione dei nuovi elementi con contenuto semantico va incontro a questa esigenza. Grazie al nome dell'elemento (e non più al valore del suo attributo "id") i motori di ricerca

(Google in primis) saranno in grado di capire se un elemento è da considerarsi un articolo o un footer di una pagina.
Possiamo anche vedere le differenze graficamente. Nelle due figure che seguono mostriamo lo stesso layout creato sia tramite elementi HTML 4 e sia con i nuovi elementi HTML 5.

Tipico layout HTML4



Tipico layout HTML5



Ora che abbiamo in qualche modo compreso l'importanza dei nuovi elementi introdotti da HTML5, vediamoli più da vicino.

I nuovi elementi

Per una migliore esperienza sul web, HTML5 definisce molti nuovi elementi e ne cancella alcuni che sono poco usati o usati in modo sbagliato. In questo capitolo vedremo alcuni tra i nuovi elementi definiti dal w3c. Fare un elenco completo dei nuovi elementi è inutile sia perché, ad oggi, molti di tali elementi non sono ancora supportati dai browser e quindi non si avrebbe neanche la possibilità di visualizzarne il risultato, sia perché tale elenco è in costante aggiornamento ed è visionabile pubblicamente qui:
http://dev.w3.org/html5/html4-differences/#new-elements

Nuovi elementi di struttura

TAG	**DESCRIZIONE**
<article>	Definisce un articolo (di un blog, di una testata giornalistica, ecc.)
<aside>	Rappresenta una sezione di una pagina costituita da informazioni che sono marginalmente correlate al contenuto dell'elemento padre che la contiene e che potrebbero essere considerate distinte da quest'ultimo. Un uso tipico è quello di contenitore di approfondimento in cui possiamo inserire gruppi di link, pubblicità, bookmark e così via, oppure di barra laterale
<figure>	Nell'elemento <figure> possiamo racchiudere dei contenuti,

	opzionalmente con una didascalia (<figcaption>), che rappresentano delle singole unità indipendenti rispetto al contenuto principale; ad esempio possiamo utilizzare l'elemento <figure> per annotare le illustrazioni, schemi, foto, elenchi di codice, etc... ovvero tutto ciò che fa parte del contenuto principale ma che potrebbe essere anche allontanato dallo stesso senza intaccarne il senso
<figcaption>	Elemento legato a <figure>
<footer>	Definisce la parte inferiore di una pagina web o di un articolo
<header>	Definisce l'intestazione di una pagina web o anche di un articolo
<hgroup>	Si sua quando è necessario raggruppare più tag di tipo <hx> (<h1>, <h2>, ecc)
<nav>	Rappresenta una sezione di una pagina che contiene link (collegamenti) ad altre pagine o a parti interne dello stesso documento; quindi, in breve, una sezione contenente link di navigazione
<section>	Rappresenta una sezione generica di un documento o applicazione (per esempio una sezione di un articolo). In tale contesto individua un raggruppamento tematico di contenuti e in genere contiene un titolo introduttivo

Altri elementi introdotti

TAG	Significato
<audio>	Gestisce musica o semplice audio in streaming
<bdi>	Identifica una parte di testo che potrebbe essere interpretata de destra verso sinistra (e viceversa)
<meter>	Viene utilizzato per rappresentare un range. Non è opportuno utilizzare questo elemento per rappresentare un numero unico (ad esempio quanti bambini qualcuno ha) a meno che ci sia un numero massimo noto
<progress>	Disegna una barra di avanzamento di un certa operazione (per esempio il download di un file)
<time>	Da usare per indicare una data
<track>	Contiene informazioni circa una traccia video. Si usa con l'elemento <video>
<video>	Consente di riprodurre video in streaming
<wbr>	Rappresenta la possibilità di andare a capo in caso di riduzione dello spazio a disposizione (per esempio per il ridimensionamento della finestra del browser)

Infine è opportuno porre attenzione all'**elemento <input>**, usato nei form, il quale ha subito diversi miglioramenti al fine di renderlo più vicino alle esigenze quotidiane del programmatore. Sono infatti stati aggiunti diversi valori per l'attributo "type" che, come sappiamo, identifica il tipo di input che ci si aspetta. Ecco quindi l'elenco dei nuovi valori per l'attributo type dell'elemento <input>:

- tel
- search
- url
- email
- datetime
- date
- month
- week
- time
- datetime-local
- number
- range
- color

Alcuni di questi nuovi valori meritano uno sguardo ravvicinato, anche perché già riconosciuti dai browser. Al seguente link
http://www.alessandrostella.it/lato_client/html5/html5_00.html
possiamo vedere in opera il seguente codice

```
<!DOCTYPE html>
<html>
    <head>
          <title>Programmare per il web</title>
    </head>
    <body>
          <input type="url" /><br />
          <input type="datetime" /><br />
          <input type="datetime-local" /><br />
          <input type="date" /><br />
          <input type="month" /><br />
          <input type="week" /><br />
          <input type="time" />
          <input type="number" min="0" max="100" step="5"/>
```

```
            <input type="range" min="0" max="10" step="1"/>
            <input type="color" /><br />
    </body>
</html>
```

Purtroppo al momento solo Opera dalla versione 12 in poi è in grado di interpretare correttamente e completamente il codice. Gli altri browser ne interpretano correttamente solo un parte. Probabilmente nel momento in cui verranno lette queste parole, la compatibilità dei browser sarà aumentanta.

I nuovi attributi

HTML 5 porta con sé sia nuovi attributi non presenti nella precedente versione di HTML, sia attributi già esistenti che però ora possono essere associati ad altri elementi. Per lo stesso motivo visto sopra, non ha molto senso farne un elenco esaustivo. Tale elenco è pubblicamente visibile e costantemente aggiornato qui:
http://dev.w3.org/html5/html4-differences/#new-attributes

Tuttavia nella tabella seguente ne vediamo alcuni tra i più implementati dai browser.

ATTRIBUTO	**SI APPLICA A**	**SIGNIFICATO**
media	<a>, <area>	Descrive per quale tipo di dispositivo è stato disegnato l'oggetto multimediale a cui fa riferimento l'lemento
hreflang	<a>, <area>	Se presente fornisce informazioni sul documento a cui viene fatto riferimento nell'elemento
autofocus	<input>, <textarea>, <select>, <button>	Quando viene caricata una pagina, l'elemento che ha questo attributo impostato su "autofocus" acquisisce il fuoco
placeholder	<input>, textarea>	Rappresenta un suggerimento con lo scopo di aiutare l'utente nell'immissione dei dati
form	<input>, <output>, <select>, <textarea>, <button>, <label>, <object>, <fieldset>	Consente di associare ad un form un elemento situato in un qualsiasi punto del documento HTML (quindi non più necessariamente all'interno del tag <form>)

required	<input>, <output>, <select>, <textarea>, <button>, <label>, <object>, <fieldset>	Nell'invio di un form indica che l'elemento con tale attributo deve essere compilato. Facciamo attenzione al fatto che nell'elemento <select> il primo elemento <option> deve avere l'attributo value settato con un valore vuoto.
autocomplete, min, max, multiple, pattern, step, width, height	<input>	Poniamo attenzione al fatto che gli attributi width e height funzionano solo quando type=image
dirname	<input>, <textarea>	Viene avvalorato dal sistema e comunica se bisogna leggere i dati inviati da destra a sinistra o da sinistra a destra
maxlength, wrap	<textarea>	Il primo attributo definisce la lunghezza massima dei caratteri inseribili. Il secondo consente o meno di andare a capo
novalidate	<form>	Impedisce la validazione del form garantendone l'invio in ogni caso
formaction, formenctype, formmethod, formnovalidate, formtarget	<input>, <button>	Se presenti, questi attributi sovrascrivono gli attributi action, enctype, method, novalidate e target dell'elemento form
type, label	menu	Consentono di trasformare il <menu> in un menù contestuale (ad esempio premendo il tasto destro del mouse)
async, defer	<script>	Questi 2 attributi sono di tipo booleano e indicano al browser come comportarsi con gli script. Async impone l'esecuzione asincrona dello script. Defer ne impone l'esecuzione solo dopo che la pagina è stata completamente caricata
sizes	<link>	Questo attributo viene usato insieme all'attributo "icon" e serve ad indicare la dimensione in pixel dell'icona (16x16, 32x32, ecc.)
reversed		Questo attributo indica che l'ordine in cui vengono mostrati gli elementi è inverso a quello originario
accesskey, class, dir, id, lang, style, tabindex, title	Questi sono attributi di HTML 4 che adesso, in HTML5, sono usabili su tutti	

	gli elementi	
contenteditable, contextmenu, data-*, draggable, dropzone, hidden, role and aria-*, spellcheck, translate	Questi invece sono nuovi attributi di HTML5 che possono essere applicati a qualsiasi elemento	

Il Content Model e le categorie

Il Content Model è il modello che decide se un determinato elemento può o meno essere annidato in un altro (ad esempio l'elemento <body> non può essere annidato nell'elemento <p> perché il content model non lo permette).
HTML 4 ha due grandi categorie di elementi:

- "in line" (ad esempio , <img>, <a>)
- "block-level" (ad esempio , <hr>, <table>).

oltre ad alcuni elementi che non rientrano in nessuna delle due categorie.
Questa suddivisione però causava molta confusione allorquando si prendeva confidenza con i fogli di stile in quanto, ad esempio, anche i fogli di stile prevedono la scrittura di codice che viene definita "inline".

Per questo motivo HTML5 evita i concetti di "block" e "inline" nella definizione dei propri elementi e modifica le categorie di appartenenza dei vari elementi. In HTML5 ci sono 7 categorie e gli elementi possono appartenere a una o più di esse
Le categorie sono le seguenti:

- **Metadata** content (esempio <link>, >script>)
- **Flow** content (esempio , , text). Questa categoria racchiude entrambe le vecchie categorie, "block" e inline", di HTML4
- **Sectioning** content (esempio <aside>, <section>)
- **Heading** content (esempio <h1>, <hgroup>)
- **Phrasing** content (esempio , <img>, <textarea>). Questa categoria equivale più o meno alla vecchia "inline" di HTML4. Da notare che, se un elemento appartiene a questa categoria, allora appartiene anche alla Flow content. Non è vero il viceversa!
- **Embedded** content (esmepio <img>, <iframe>, <svg>)
- **Interactive** content (esempio <a>, <button>, <label>). Non è consentito nidificare più elementi di questa categoria.

Sarebbe bene entrare in confidenza con queste nuove categorie perché nei prossimi anni diventeranno il nostro pane quotidiano. Per i dettagli sulla categorie e per averne una rappresentazione d'insieme, si rimanda al seguente link:
http://www.whatwg.org/specs/web-apps/current-work/multipage/elements.html#content-categories

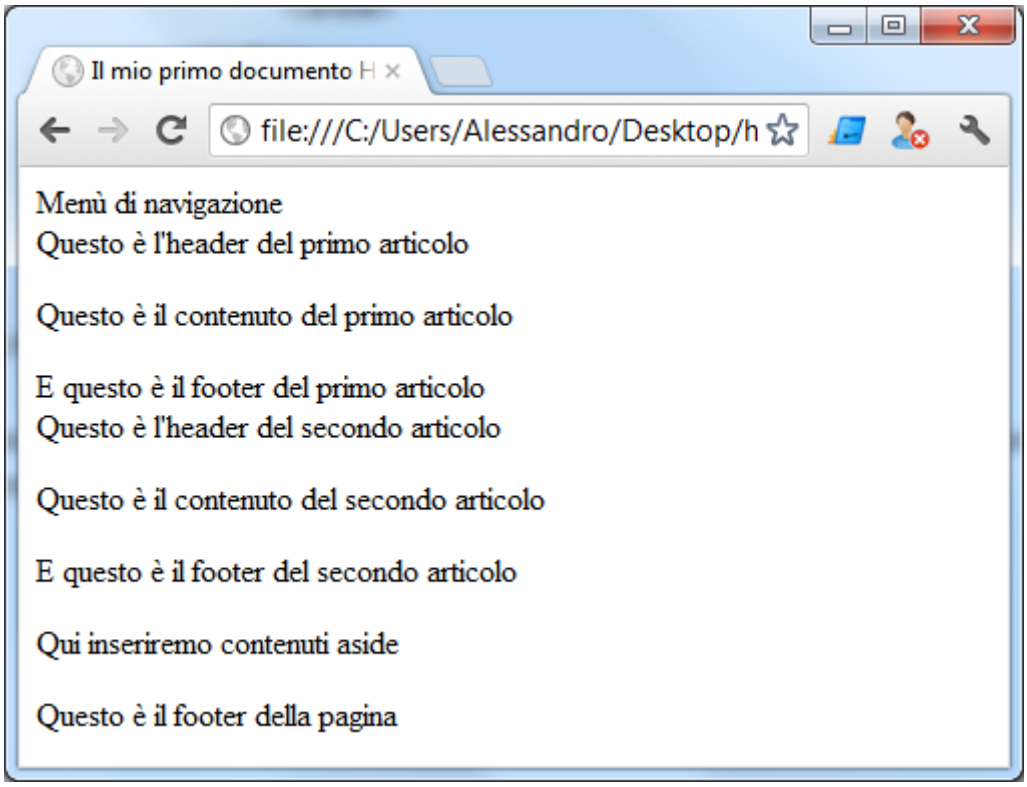
Il primo documento HTML5

Ora che abbiamo terminato il noioso elenco di novità e di differenze rispetto ad HTML4, prendiamo in esame un documento HTML5 reale, il nostro primo documento HTML5.

```
<!DOCTYPE html>
<html lang="it">
    <head>
          <meta charset="utf-8">
          <meta name="description" content="Primo document
                HTML5">
          <meta name="author" content="Alessandro Stella">
          <title>Il mio primo documento HTML 5</title>
    </head>
    <body>
          <header>
                <nav>Menù di navigazione</nav>
          </header>
          <section>
                <article>
                      <header>Questo è l'header del primo
                            articolo</header>
                      <p>Questo è il contenuto del primo
                            articolo</p>
                      <footer>E questo è il footer del primo
                            articolo</footer>
                </article>
                <article>
                      <header>Questo è l'header del second
                            articolo</header>
                      <p>Questo è il contenuto del second
                            articolo</p>
                      <footer>E questo è il footer del second
```

```
                              articolo</footer>
                  </article>
            </section>
            <aside>
                  <p>Qui inseriremo contenuti aside</p>
            </aside>
            <footer>Questo è il footer della pagina</footer>
      </body>
</html>
```

Ed ecco come verrebbe tradotto da un browser in grado di comprendere i nuovi elementi di HTML5.



Ovviamente, mancando i fogli di stile, il risultato è esteticamente poco apprezzabile.
Possiamo testare il codice qui:
http://www.alessandrostella.it/lato_client/html5/html5_01.html

Ma osserviamo il codice e facciamone un confronto con HTML4.

```
<head>
      <meta charset="utf-8">
```

```
    <meta name="description" content="Primo documento HTML5">
    <meta name="author" content="Alessandro Stella">
    <title>Il mio primo documento HTML 5</title>
</head>
```

La prima differenza la notiamo subito

<meta charset="utf-8">

Con HTML4 per indicare la stessa cosa, cioè il set di caratteri usati dalla pagina, avremmo dovuto scrivere:

<meta http-equiv=”Content-type” content=”text/html” charset="utf-8">

Una leggera semplificazione...

Le altre istruzioni restano invece invariate.

Passando all’elemento <body> ci accorgiamo che è addirittura tutto nuovo! Tutti gli elementi che abbiamo usato nel nostro primo documento HTML5 sono inesistenti in HTML4. Per produrre lo stesso tipo di struttura usando HTML4 avremmo dovuto usare molti elementi e questa è una delle modifiche più importanti che dovremo apportare alle nostre abitudini: **con HTML5 il tag subisce un drastico ridimensionamento in termini di uso**.

Ma proseguiamo nell’analisi del codice.

```
<header>
    <nav>Menù di navigazione</nav>
</header>
```

Questo codice a ben pensarci è davvero chiarissimo! Ci dice cosa inserire nella parte superiore (<header>) della pagina web. In questo caso viene inserito solo un menù di navigazione (<nav>), ma nulla vieta di usare i CSS per inserire immagini di sfondo e altri abbellimenti estetici. Anche in questo caso vale quanto detto poco fa: per produrre tale struttura con HTML4 avremmo dovuto usare almeno due elementi che invece adesso non hanno più alcun senso.

Proseguendo nel codice, dopo l’elemento <header> compare l’elemento <section>. Siamo quindi di fronte ad una sezione della pagina. In questo caso, in questa sezione, verranno inseriti gli articoli.

Ogni articolo è rappresentato da un elemento <article> (niente di più intuitivo!) e, nel nostro codice, ogni articolo è composto da un <header> (tipicamente il titolo), il corpo vero e proprio dell’articolo e un <footer> (ovvero eventuali informazioni da posizionare in

coda all'articolo). Sembra davvero tutto estremamente facile! Pensiamo per un momento a quanti sarebbero stati necessari per riprodurre una tale struttura.

Dopo questa <section>, contenente gli articoli, è stato posizionato un elemento <aside>. Come abbiamo visto il suo tipico uso è quello di "barra laterale" e quindi sarà probabilmente posizionato su un lato della pagina (usando, ad esempio, float:right) .

Infine la nostra prima pagina web in HTML5 termina con un <footer>. Anche questo sembra davvero intuitivo.

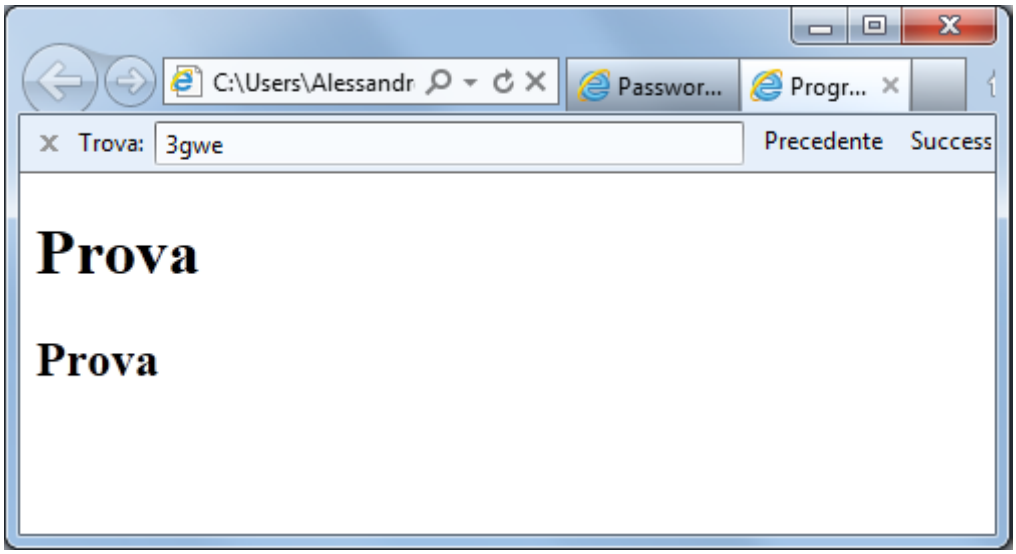
Con questo semplice documento HTML5 abbiamo potuto osservare da vicino quanto profonde siano le novità introdotte da HTML5.

Osserviamo ora quanto sia ancora poco uniforme il supporto ad HTML5 nei browser più conosciuti. Ammettiamo di avere il seguente semplice documento HTML5.

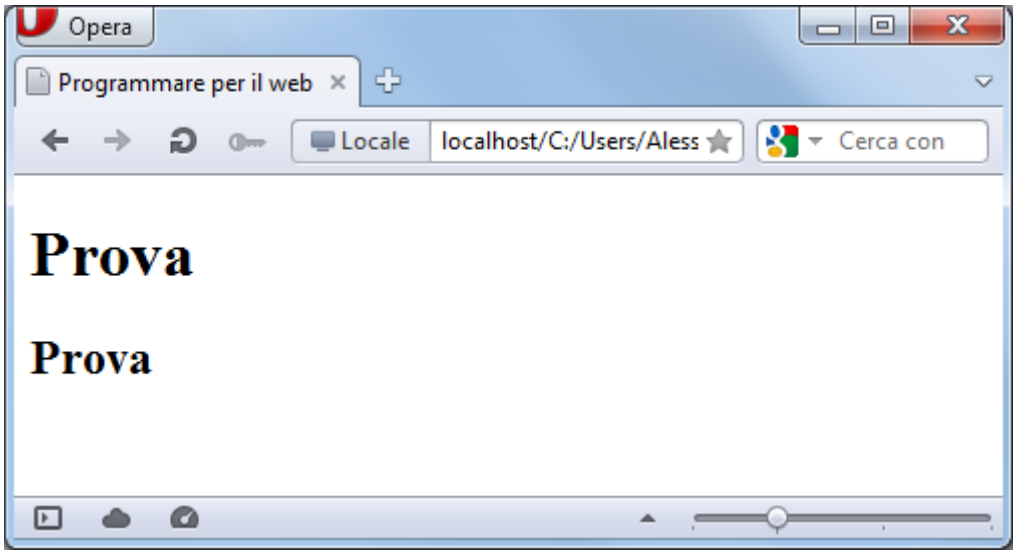
```
<!DOCTYPE html>
<html>
    <head>
          <title>Programmare per il web</title>

    </head>
    <body>
          <article>
                <section>
                      <header><h1>Prova</h1></header>
                </section>
                <section>
                      <header><h2>Prova</h2></header>
                </section>
          </article>
    </body>
</html>
```

Ecco come viene interpretato questo codice dai vari browser di ultima generazione.

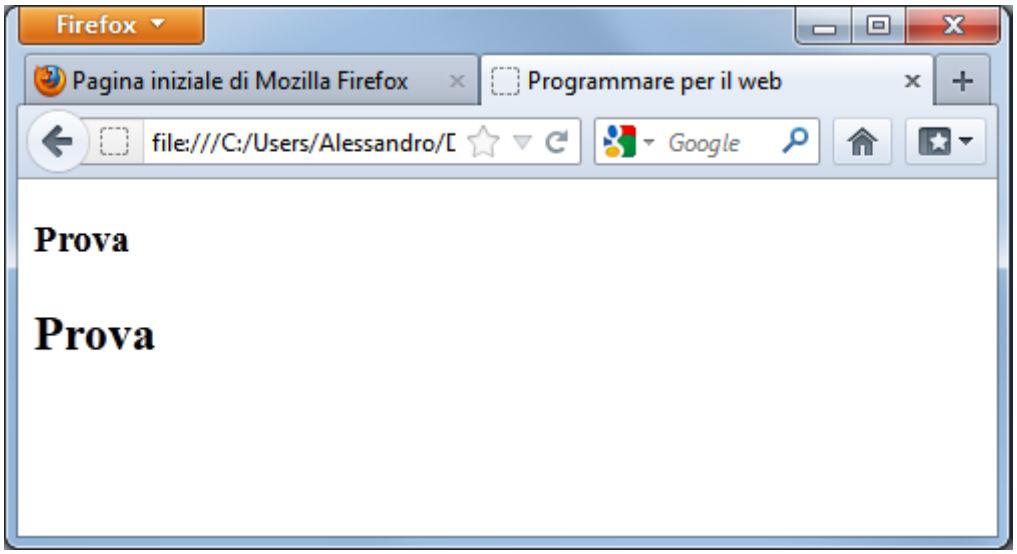
Internet Explorer 9



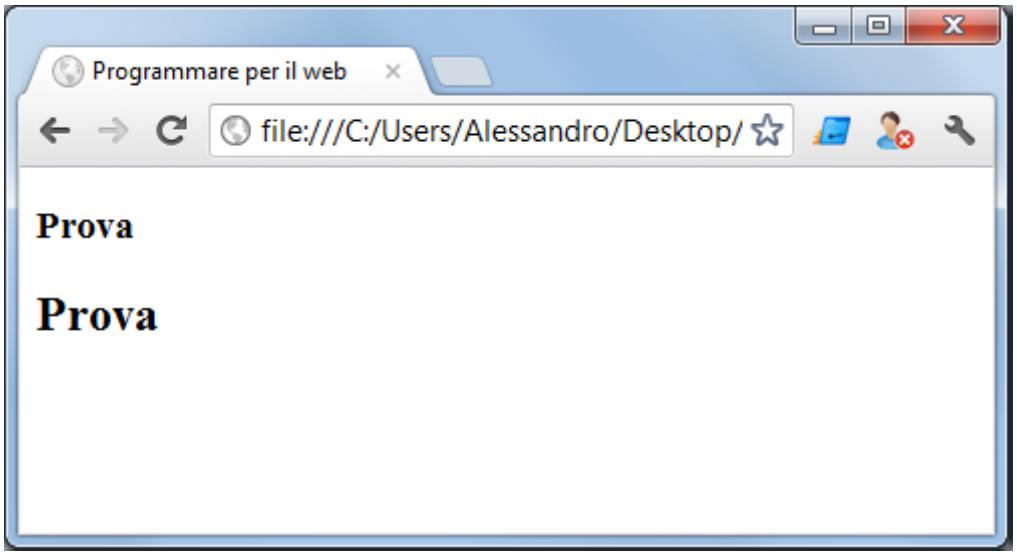
Opera 12



Firefox 13



Chromium 21



Dalle immagini possiamo osservare come, nonostante il codice sia davvero banale, in alcuni browser un elemento <h1> venga rappresentato con un carattere più piccolo

rispetto a un elemento <h2> (a causa di un elemento <article> un po' più complesso del normale). Ciò fa comprendere che c'è ancora tanto da lavorare per ottenere un supporto reale per HTML5. Tra l'altro gli errori di interpretazione vengono proprio dai due browser che ne vantano il supporto maggiore, cioè Firefox e Chrome! Tuttavia è molto probabile che in breve tempo questi errori di interpretazione vengano eliminati.

7
CSS 3

Il livello 3 dei fogli di stile porterà finalmente a compimento un cambiamento nella scrittura delle pagine web iniziato molti anni fa. Purtroppo solo nel 2008 si è ottenuta una piena compatibilità di tutti i browser con le specifiche w3c e quindi solo da pochi anni si è potuto progettare un piano di evoluzione delle specifiche degno di questo nome.
Molto di quello che sappiamo sui fogli di stile resta valido. Alcune cose cambieranno, altre saranno aggiunte ex novo. La prima modifica la troviamo subito a livello organizzativo, infatti le specifiche dei nuovi CSS sono state suddivise in **moduli**, ossia in sottoinsiemi dell'intera specifica in grado di evolvere autonomamente senza interferire con gli altri. Ciò consente ai produttori dei browser di integrare i singoli moduli di volta in volta.
Allo stato attuale tuttavia vale quanto già detto per HTML5, ossia il livello 3 delle specifiche dei fogli di stile nel complesso non è ancora uno standard. Usare i css3 necessita quindi di un'accurata valutazione. Dal sito del w3c possiamo leggere:
"CSS Level 3 builds on CSS Level 2 module by module, using the CSS2.1 specification as its core. Each module adds functionality and/or replaces part of the CSS2.1 specification. The CSS Working Group intends that the new CSS modules will not contradict the CSS2.1 specification: only that they will add functionality and refine definitions. As each module is completed, it will be plugged in to the existing system of CSS2.1 plus previously-completed modules."
In altre parole non c'è ancora una vera e propria specifica dei css3. Tutto ciò che di nuovo viene creato viene aggiunto alle già esistenti specifiche css 2.1, modulo per modulo.
Le specifiche aggiornate vengono pubblicate qui:

http://www.w3.org/Style/CSS/current-work

Sebbene non esista ancora una vera e propria specifica css3, alcuni dei moduli sono stati pubblicati come Recommendation (REC) mentre altri sono in stato Candidate Recommendation (CR), per cui molti browser ne consentono già l'utilizzo.
Per avere consapevolezza del percorso che una specifica deve fare prima di diventare una raccomandazione (REC), riportiamo qui di seguito tutte le fasi del percorso (dal meno stabile alla raccomandazione):

WD Working Draft
LC Last Call
CR Candidate Recommendation
PR Proposed Recommendation
REC Recommendation

Sul sito del w3c possiamo leggere: "W3C encourages everyday use starting from CR". Quindi i browser iniziano a integrare le specifiche quando esse si trovano ancora in stato CR. Ecco un elenco completo dei moduli w3c e del relativo stato (tra parentesi quadre) in cui si trovano al momento:

- CSS Color level 3 [REC]
- CSS Selectors level 3 [REC]
- CSS Media Queries [REC]
- CSS Backgrounds and Borders Level 3 [PR]
- CSS Image Values and Replaced Content Level 3 [CR]
- CSS Text Level 3 [WD]
- CSS Values and Units Level 3 [WD]
- CSS Fonts Level 3 [WD]
- CSS Basic User Interface Level 3 [WD]
- CSS Writing Modes Level 3 [WD]
- CSS Conditional Rules Level 3 [WD]
- CSS Lists Level 3 [WD]
- CSS Positioned Layout Level 3 [WD]
- CSS Fragmentation Level 3 [WD]

La nostra attenzione si focalizzerà sui moduli in stato REC e, in parte, sui moduli in stato PR e CR. Al momento non ha alcun senso prendere in considerazione gli altri.
Come sempre, per testare il codice proposto, possiamo usare l'editor online:
http://jsfiddle.net/

oppure usare un qualunque editor di testo (come Notepad++).

CSS Color level 3

Questo modulo è in stato REC ed è quindi auspicabile (ma non obbligatorio) che tutti i browser ne forniscano una completa implementazione. Ha come scopo quello di gestire i colori in foreground (ossia il colore del testo) degli elementi HTML e, più in generale, di specificare la sintassi da usare per assegnare un colore ad un elemento. Le specifiche del modulo sono ovviamente pubbliche e possono essere reperite al seguente indirizzo: http://www.w3.org/TR/css3-color/

Per gestire il colore del testo o dei bordi di un elemento HTML, le specifiche prevedono due proprietà:

- color
- opacity

La proprietà color

Questa proprietà definisce il colore in foreground (cioè il colore in primo piano, non quello dello sfondo) dell'elemento HTML a cui è applicata. I valori di questa proprietà possono essere espressi in diversi modi che vedremo in seguito. Ad esempio il codice

```
<html>
    <body>
        <p style="color:blue">Nome</p>
    </body>
</html>
```

produrrà come risultato un paragrafo il cui testo sarà di colore blu. Ci sono diversi metodi per dare un colore di foreground ad un elemento. Qui li vediamo tutti insieme:

```
<html>
    <body>
        <p style="color:blue">Nome</p>
        <p style="color:rgb(0,0,255)">Cognome</p>
        <p style="color:#0000FF">Indirizzo</p>
        <p style="color:hsl(240,100%,50%)">Telefono</p>
    </body>
</html>
```

In tutti e 4 i casi otterremo un testo di colore blu.
Possiamo vedere il codice in azione qui:
http://www.alessandrostella.it/lato_client/css3/css3_00.html

La funzione **rgb(0,0,255)** calcola il colore come miscela dei 3 colori di base, cioè **r**ed, **g**reen e **b**lue (rgb). Ogni colore può assumere un valore tra 0 (niente colore) e 255 (il massimo del colore). Pertanto rgb(0,0,255) significa 0 rosso, 0 verde e 255 blue, quindi blue.
Lo stesso discorso, ma in modo più compatto, vale per il valore **#0000FF**. Anche in questo caso vengono usati 3 numeri per indicare i 3 colori fondamentali (rgb). In questo caso però invece della funzione rgb() viene usato il cancelletto (#) e i valori per i i singoli colori sono espressi in esadecimale invece che in decimale, ossia i valori che possono assumere i colori partono da 00 e arrivano a FF (255 in decimale). Pertanto #0000FF significa 0 rosso, 0 verde e 255 blue.
Infine la funzione **hsl(240,100%,50%)** calcola il colore in base a 3 parametri, cioè **h**ue, **s**aturation e **l**ightness (hsl). Per comprenderne il modo di utilizzo è opportuno studiare le seguenti tabelle
http://www.w3.org/TR/css3-color/#hsl-examples

Questi sono tutti i modi che abbiamo a disposizione per decidere il colore in foreground di un elemento HTML. In realtà, vedremo in seguito che le stesse funzioni appena viste sono usate anche per assegnare un colore di sfondo, ma ovviamente la proprietà non sarà più color, ma background-color.

La proprietà opacity

Questa proprietà ha come scopo quello di rendere trasparente o opaco un elemento, indipendentemente dal colore. I valori che può assumere questa proprietà vanno da 0 a 1. Zero significa completamente trasparente, mentre 1 significa completamente opaco.
Esempio:

```
<html>
    <body>
        <p style="opacity:0.5">Nome</p>
    </body>
</html>
```

Il risultato sarà quello di ottenere il testo "Nome" opaco al 50%.

Dobbiamo porre attenzione a questa proprietà, perché essa viene ereditata da tutti i figli dell'elemento a cui è applicata. Prendiamo infatti questo codice HTML:

```
<html>
    <body>
        <div id="test">
            <p>alex</p>
        </div>
    </body>
</html>
```

e il rispettivo CSS:

```
#test {
    background:red;
    border-style:solid;
    border-color:black;
    width:250px;
    height:250px;
    opacity:0.5;
}
```

Con questo codice ci si potrebbe aspettare un quadrato con sfondo rosso e bordo nero (cioè il #test), opaco al 50%. Poi, all'interno di questo quadrato, un paragrafo con testo nero. Ma non è così. **La proprietà opacity infatti viene ereditata da tutti gli elementi figli dell'elemento a cui viene impostata**. Quindi tutti gli elementi HTML figli del #test avranno sia i colori in primo piano, sia quelli di sfondo, trasparenti al 50%. Quindi anche il testo "alex" risulterà trasparente al 50%.
Possiamo osservare quanto appena asserito qui:
http://www.alessandrostella.it/lato_client/css3/css3_01.html

Non sempre però questo è ciò che vogliamo! Per impostare trasparente al 50% solo il colore di sfondo del rettangolo, possiamo usufruire della funzione **rgba(r,g,b,a)** con cui, oltre a impostare uno specifico colore per un elemento (i primi 3 parametri), possiamo deciderne anche il livello di opacità (ultimo parametro). Basta quindi modificare il codice CSS, come segue:

```
#test {
    background:rgba(255,0,0,0.5);
```

```
    border-style:solid;
    border-color:black;
    width:250px;
    height:250px;
}
```

In questo modo otterremo il quadrato rosso opaco al 50%, ma lasceremo inalterata l'opacità sia dei bordi del quadrato sia del testo del paragrafo.
Possiamo osservare il risultato qui:
http://www.alessandrostella.it/lato_client/css3/css3_02.html

Infine una nota sulla compatibilità della proprietà opacity con i vari browser. **Questa proprietà non funziona sulle versioni di Internet Explorer precedenti alla versione 9**. Su Internet Explorer 8 per definire la trasparenza si ricorre ad un filtro proprietario di Microsoft preceduto dal prefisso -ms- (i valori vanno da 1 a 100):
-ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=50)";

Su Internet Explorer 6 e Internet Explorer 7, la sintassi è ancora diversa:
filter: alpha(opacity=50);

Quindi se vogliamo estendere la compatibilità alle versioni 6, 7 e 8 di Internet Explorer, dobbiamo usare la sintassi appena mostrata.

CSS Selectors level 3

Anche questo modulo è nello stato REC e quindi tutti i browser possono implementarne i dettami in ogni momento, anzi, in realtà questo modulo è già implememtato in modo pressoché completo da tutti i browser.
Molto di quello che diremo è già valido per i CSS2, ma l'importanza di questo modulo è tale da trattarlo in modo completo ed esteso, anche se questo significa ribadire ciò che già sappiamo.
Iniziamo con il ricordare che la sintassi di utilizzo dei fogli di stile è la seguente:

selettore {proprietà:valore; …; proprietà:valore;}

Questo modulo si occupa di definire il "selettore" e tutti i suoi possibili utilizzi, pertanto è un modulo abbastanza complesso. Se non ricordiamo bene la sintassi e gli utilizzi dei fogli di stile è opportuno rileggere il capitolo ad essi dedicato.
Un selettore CSS viene usato per selezionare un singolo elemento HTML o un insieme di elementi. A tali elementi vengono poi applicate le proprietà indicate tra parentesi graffe.
Ricordiamo che la sintassi CSS è **case-insensitive** per cui non ci sono differenze tra minuscole e maiuscole.

Per quanto noioso è opportuno dare alcune **definizioni** al fine di usare i termini nel modo corretto, così come definiti dalle specifiche. Ciò è importante perché con l'avvento di questo modulo sono state introdotte delle modifiche alla tradizionale nomenclatura (purtroppo!). Le definizioni di base (selettore, gruppo di selettori, selettore semplice, ecc.) sono state modificate. In particolare, quello che nelle specifiche CSS2 veniva denominato con il termine di "selettore semplice" è ora chiamato "sequenza di selettori semplici", mentre il termine "selettore semplice" è ora utilizzato per i componenti di questa sequenza.
Quindi investiamo qualche minuto e prendiamo confidenza con questa nuova nomenclatura.

Un selettore semplice (simple selector) è rappresentato da uno e uno solo dei seguenti tipi di selettore: selettore di tipo, selettore universale, selettore di attributo, selettore di classe, selettore di ID, pseudo-classe.
Esempi:
```
* {...}
h1 {...}
.nomeClasse {...}
```
Nel primo caso il selettore semplice è l'asterisco (chiamato selettore universale).
Nel secondo caso il selettore semplice è h1 (selettore di tipo).
Nel terzo caso il selettore semplice è il nome della classe (selettore di classe).

Una sequenza di selettori semplici (sequence of simple selectors) è un insieme di selettori semplici che NON sono separati da un combinatore (vedremo tra poco cosa si intende per combinatore). La sequenza deve iniziare sempre con un selettore di tipo o un selettore universale e non può contenere più di un selettore di tipo.
Esempio:
```
p.nomeClasse {...}
```

La sequenza di selettori semplici è data dal selettore di tipo e dal selettore di classe .nomeclasse. Questo codice css indica al browser di andare a cercare il paragrafo con attributo class=”nomeClasse”.

Un selettore (selector) è una sequenza di uno o più selettori semplici separati da combinatori (vedremo tra poco cosa è un combinatore).
Esempio:
h1 > h2 {...}

In questo caso il selettore è rappresentato da h1 > h2 ed è composto dai selettori semplici h1 e h2 combinati dal combinatore >.

Un combinatore (combinator) si occupa di combinare tra loro selettori semplici o sequenze di selettori semplici. Un combinatore è rappresentato da uno dei seguenti caratteri:

- spazio bianco
- "segno di maggiore" (U +003 E,>)
- "segno più" (U +002 B, +)
- "tilde" (U +007 E, ~).

Tra un selettore semplice e il seguente può esserci uno spazio bianco. E’ opportuno mettere in evidenza che **la virgola NON è un combinatore**!

Un gruppo di selettori (group of selectors) è una sequenza di selettori semplici divisi da una virgola.
Esempio:
h1, h2, h3 {...}
Il gruppo di selettori è la sequenza h1, h2, h3.

Queste, sommariamente, sono le definizioni che ci consentono di parlare in modo preciso dei nuovi css. Ora possiamo entrare nei dettagli.

I selettori semplici

Abbiamo visto che nelle specifiche CSS3 con il termine “selettore semplice” si intende un qualunque selettore appartenente alle seguenti categorie:

- selettore di tipo (type selector)
- selettore universale (universal selector)
- selettore di attributo (attribute selector)

- selettore di classe (class selector)
- selettore di ID (id selector)
- selettore con pseudo-classe (pseudo-classes)

Selettori di tipo

I selettori di tipo sono rappresentati dal nome di uno specifico elemento HTML. Servono a selezionare tutti gli elementi di quel tipo specifico presenti in un documento.
Esempio:

```
h1 {color: red}
p {color: green}
li {color: blue}
```

Con il selettore h1 selezioniamo tutti i titoli <h1>, con il selettore p tutti i paragrafi <p>, con il selettore li tutti gli item di lista <li> a prescindere dal fatto che la lista sia ordinata o non ordinata. A ciascun elemento assegniamo un colore diverso per il testo. Possiamo osservare il risultato qui:
http://www.alessandrostella.it/lato_client/css3/css3_03.html

Selettore universale

Il selettore universale è definito sintatticamente da un asterisco: *. La sua funzione è quella di selezionare tutti gli elementi presenti in un documento. La seguente regola assegna pertanto il colore rosso al testo di qualsiasi elemento rappresentato nell'albero del documento:

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

Selettori di attributo

I selettori di attributo permettono di selezionare gli elementi HTML in base agli attributi ad essi associati. Indicando con "att" un generico attributo e con "val" un generico valore di tale attributo, vediamo la sintassi a nostra disposizione.

[att]

Seleziona tutti gli elementi HTML che hanno l'attributo "att", qualunque sia il suo valore.
Esempio:

```
[id] {...}
```

Questo codice seleziona tutti gli elementi HTML che hanno l'attributo "id", qualunque sia il valore dell'attributo.

[att=val]

Seleziona tutti gli elementi HTML che hanno l'attributo "att" avvalorato con il valore indicato da "val".
Esempio:
[class="header"] {...}

Questo codice selezionerà tutti gli elementi HTML che hanno l'attributo "class" avvalorato con il valore "header".

[att~=val]
Seleziona tutti gli elementi HTML che hanno l'attributo "att" il cui valore contiene al suo interno, in qualunque posizione, il valore indicato da "val" con almeno uno **spazio** prima o dopo.
Ad esempio, ipotizzando un codice HTML come il seguente:
Testo qualsiasi

il seguente codice css:
[title~="colorato"] {...}

selezionerà il tag <a> perché presenta l'attributo "title" con un valore contenente il testo "colorato" con uno spazio alla sua sinistra. Possiamo osservare l'esito qui:
http://www.alessandrostella.it/lato_client/css3/css3_04.html

[att|=val]
Selettore del tutto simile al precedente, con una semplice differenza nella definizione del valore. Mentre per l'esempio precedente il valore doveva essere separato da spazi, in questo selettore il valore deve essere separato da almeno un **trattino** (-).
Ad esempio, ipotizzando un codice HTML come il seguente:
Testo colorato

il codice css
[title|="colorato"] {...}

selezionerà il tag <a> perché presenta l'attributo "title" con un valore contenente il testo "colorato" che ha un trattino alla propria sinistra.

[att^=val]

Seleziona tutti gli elementi HTML che hanno l'attributo "att" il cui valore **inizia** con il valore indicato in "val".
Rifacendoci al solito esempio, il seguente codice:
[title="Questo"] {...}

selezionerà il nostro tag <a> in quanto il valore del suo attributo "title" inizia con "Questo". La regola fa distinzione tra maiuscole e minuscole!

[att$=val]
Seleziona tutti gli elementi HTML che hanno l'attributo "att" il cui valore **termina** con il valore indicato in "val".
Quindi scrivendo
[title="colorato"] {...}

selezioneremo il nostro tag <a> in quanto il valore del suo attributo "title" termina proprio con la parola "colorato" (anche senza spazi né trattini a sinistra o a destra del testo). La regola fa distinzione tra maiuscole e minuscole!

[att*=val]
Seleziona tutti gli elementi HTML che hanno l'attributo "att" il cui valore contiene in una qualunque posizione il valore indicato in "val".
Esempio:
[title="col"] {...}

Anche in questo caso verrà selezionato il nostro tag <a> perché il valore del suo attributo "title" contiene il testo "col". La regola fa distinzione tra maiuscole e minuscole!

Selettori di classe

Come sappiamo, ad ogni elemento HTML può essere associata una classe usando l'attributo "class" e assegnando a tale attributo un valore a nostra scelta. Esempio:
<h1 class="titolo">Testo</h1>

È anche possibile assegnare ad un elemento più classi. È sufficiente definirle nel codice separandole con uno spazio:
<h1 class="titolo testata rosso">Testo</h1>

Inoltre una stessa classe può essere assegnata a più di un elemento:

<h1 class="titolo">Un certo testo</h1>
<h2 class="titolo">Un altro testo</h2>

Nei CSS, per selezionare gli elementi a cui sia stata assegnata una classe, si utilizza questa sintassi:
.nomeDellaClasse {...}

Il nome della classe viene fatto precedere da un punto (.).
Ad esempio:
.titolo {color:red}

La regola appena vista imposta il colore rosso per il testo degli elementi con classe "titolo".
È anche possibile imporre regole diverse per la stessa classe a seconda dell'elemento a cui è associata.
Ad esempio, se questo è il nostro codice HTML:

```
<h1 class="titolo">Titolo qualsiasi</h1>
<p class="titolo">Paragrafo qualsiasi</p>
```

e questo è il nostro codice css:

```
.titolo {
    color:blue;
}
h1.titolo {
    color:red;
    text-transform:uppuercase;
}
```

otterremo un testo blu per il paragrafo <p> e un testo rosso è in maiuscolo per l'head <h1>, pur avendo associato la stessa classe ad entrambi gli elementi. Possiamo osservare il risultato qui:
http://www.alessandrostella.it/lato_client/css3/css3_05.html

Avremmo potuto ottenere lo stesso risultato usando la classi multiple. Ossia, scrivendo il seguente codice HTML:

```
<h1 class="titolo maiuscolo">Titolo qualsiasi</h1>
<p class="titolo">Paragrafo qualsiasi</p>
```

e questo codice CSS:

```
.titolo {
    color:blue;
}
.maiuscolo {
    color:red;
    text-transform:uppercase
}
```

Possiamo osservare il risultato qui:
http://www.alessandrostella.it/lato_client/css3/css3_06.html

Assegnando due classi a <h1>, il suo testo ne esprime la somma, per cui il testo viene mostrato in rosso e in maiuscolo. E' molto importante osservare che se nel codice css avessimo dichiarato la classe .maiuscolo prima della classe .titolo, il colore del testo di <h1> sarebbe stato blu e non rosso! Ciò accade perché il browser legge le classi css nell'ordine in cui vengono dichiarate. Se la classe .titolo fosse l'ultima dichiarata sarebbe essa a definire il colore del testo.
Cosa accade se invece di invertire l'ordine di dichiarazione delle classi, invertiamo l'ordine di associazione nel codice HTML? Cioè, cosa succede se, lasciando inalterato il codice css, modifichiamo il codice HTML da così:
<h1 class="titolo maiuscolo">Titolo qualsiasi</h1>

a così:
<h1 class="maiuscolo titolo">Titolo qualsiasi</h1>

La risposta è: niente.
Il browser va a cercare comunque le due classi nel codice css e applica le loro regole a cascata. Poiché la classe .maiuscolo nel codice css è dichiarata dopo la classe .titolo, sarà l'ultima ad essere letta e quindi sarà comunque la classe .maiuscolo a definire il colore del testo dell'elemento <h1>.

Selettori di ID

L'attributo id è un attributo universale in HTML, ciò significa che tutti gli elementi presenti nel documento possono avere un loro id. A differenza delle classi, però, **uno specifico id può essere assegnato solo ad un elemento**. Non possiamo quindi avere in un documento HTML una situazione del genere:

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

Per selezionare l'elemento con un determinato id si usa la seguente sintassi:

```
#valoreid
```

Ad esempio, avendo li seguente codice HTML:

```
<h1 id=titolo”>Titolo</h1>
<h2 id=”sottotitolo”>Sottotitolo</h2>
```

il seguente codice css:

```
#titolo {...}
```

seleziona l'elemento <h1> in quanto il valore del suo attributo id è proprio "titolo".
È anche possibile (ma non obbligatorio) usare, prima del cancelletto, il nome dell'elemento, scrivendo ad esempio:

```
h1#titolo
```

Selettori con pseudo-classi

Dovremmo avere già chiaro il concetto di pseudo-classe, lo abbiamo imparato quando abbiamo studiato i css2: servono per selezionare un elemento in base al suo stato e non in base alla struttura del documento.
Dovremmo anche ricordare che per selezionare elementi HTML tramite una pseudo-classe bisogna usare la seguente sintassi:

```
:nomePseudoClasse()
```

Quindi usare i due punti (:) seguiti dal nome della pseudo-classe e, in alcuni casi, da un valore tra parentesi tonde.
Ricordiamo infine che le pseudo-classi sono ammesse in tutte le sequenze di selettori semplici contenute in un selettore. Inoltre sono ammesse ovunque fra sequenze di selettori semplici e anche dopo il tipo di selettore principale o selettore universale (possibilmente omesso). I nomi delle pseudo-calssi sono case-insensitive, ma è buona norma scriverli in minuscolo. Alcune pseudo-classi sono mutuamente esclusive, mentre

altre possono essere applicate simultaneamente allo stesso elemento. Ricordiamo infine che le pseudo-classi possono essere dinamiche, nel senso che un elemento HTML può acquisire o perdere una pseudo-classe mentre l'utente interagisce con la pagina web.

Ricordato tutto ciò, possiamo proseguire osservando che nelle specifiche css3, le pseudo-classi sono così suddivise:

- pseudo-classi dinamiche
- pseudo-classi degli stati degli elementi della UI (User Interface)
- pseudo-classi strutturali
- pseudo-classe :target
- pseudo-classe :lang

Vediamo nel dettaglio le prime 3.

Pseudo-classi dinamiche

Le conosciamo già. Sono le stesse dei CSS2, cioè:

- :link
- :visited
- :hover
- :active
- :focus

Esempi:

```
a:link      /* seleziona i link non visitati */
a:visited   /* seleziona i link visitati */
a:hover     /* seleziona i link sui quali passa il mouse */
a:active    /* seleziona i link attivi */
a:focus     /* seleziona i link che hanno il fuoco */
```

Possiamo osservarne un tipico utilizzo qui:
http://www.alessandrostella.it/lato_client/css3/css3_07.html

Pseudo-classi degli stati degli elementi della UI (User Interface)

Il titolo scelto dal w3c per questa sezione è davvero poco chiaro. Per UI il w3c in questo contesto intende tutti quegli elementi HTML che consentono all'utente di interagire con la pagina web. In pratica stiamo parlando dell'elemento <input> e degli elementi presenti

in un tag <form>. Quindi queste pseudo-classi hanno come scopo quello di gestire gli stati di tali campi.
Le pseudo-classi della User Interface sono:

- :enabled
- :disabled
- :checked

Non sembra particolarmente complesso comprenderne il significato.
La pseudo-classe :enabled rappresenta elementi dell'interfaccia utente che sono in uno stato attivato; tali elementi hanno un corrispondente stato disabilitato.
Al contrario, la pseudo-classe :disabled rappresenta gli elementi dell'interfaccia utente che sono in uno stato disattivato; tali elementi hanno un corrispondente stato abilitato.
Infine la pseudo-classe :checked identifica tutti gli elementi che sono in uno stato di "spuntato" o selezionato, ossia radio e checkbox.
E' opportuno osservare che in alcuni casi sia un radio sia una checkbox possono trovarsi in uno stato diverso da checked/unchecked. Ciò avviene quando nessun checkbox (o radio) è stato selezionato. Al fine di identificare questo stato è prevista, nelle specifiche future, l'introduzione di un'altra pseudo-classe, la pseudo-classe :indeterminate. Attualmente però questa pseudo-classe non esiste.
Possiamo osservarne un esempio di utilizzo qui:
http://www.alessandrostella.it/lato_client/css3/css3_08.html

Pseudo-classi strutturali

Le pseudo-classi strutturali sono uno strumento che consente di selezionare gli elementi HTML in base a informazioni sull'oggetto che non possono essere raggiunte da selettori semplici o combinatori.
Sintatticamente vengono definite come quelle previste nelle precedenti specifiche: una pseudo-classe si definisce attraverso il segno : seguito dal nome della pseudo-classe.
Prima di analizzare ogni pseudo-classe, facciamo una precisazione necessaria alla corretta comprensione dell'argomento. Dato un qualsiasi oggetto all'interno del DOM, se l'oggetto contiene degli elementi figli, l'indice dei figli, contrariamente ai linguaggi di programmazione, inizia da 1 e non da 0. Vediamo un semplicissimo esempio per chiarire bene le idee:

```
<ul>
   <li>List Item 1</li>  (Posizione 1)
   <li>List Item 2</li>  (Posizione 2)
   <li>List Item 3</li>  (Posizione 3)
```

```
    <li>List Item 4</li>  (Posizione 4)
    <li>List Item 5</li>  (Posizione 5)
</ul>
```

Fatta questa doverosa premessa, vediamo ora quali sono le pseudo-classi strutturali, facendone dapprima un elenco e analizzandole poi una per una.

- :root
- :nth-child()
- :nth-last-child()
- :last-child
- :only-child
- :nth-of-type()
- :nth-last-of-type()
- :first-of-type
- :last-of-type
- :only-of-type
- :empty

Possiamo subito osservare che alcune di esse ammettono eventuali parametri tra parentesi tonde, mentre altre non hanno questa caratteristica.

:root

La pseudo-classe :root identifica l'elemento radice della pagina. Per pagine HTML l'elemento corrispondente è proprio <html>. Pertanto scrivere

```
hml {...}
```

oppure scrivere

```
:root {....}
```

significa la stessa cosa, cioè selezionare l'elemento <html>.

:nth-child()

Sicuramente la pseudo-classe più importante e interessante introdotta dalle nuove specifiche CSS3.

Il suo scopo è di consentire la selezione di uno o più fratelli di un dato elemento in base alla posizione (nth) e alla parentela (child).

La sintassi completa è la seguente

:nth-child(an+b)
e non è così banale spiegarne il significato.
Parlando in termini matematici si può dire che:

- b rappresenta la posizione del primo elemento che si desidera selezionare
- a indica ogni quanti elementi si vuole ripetere la selezione
- n è un intero che va da zero a int(numeroElementi-1/a), se a>0; dove numeroElementi rappresenta tutti gli elementi tra loro fratelli e int() rappresenta la parte intera di ciò che c'è tra parentesi.

Comprendere tutto ciò, non è sicuramente la cosa più semplice possibile. Per questo motivo procederemo immediatamente con il classico esempio delle righe di una tabella (<table>). L'obiettivo è quello di selezionare le righe (<tr>) dispari della tabella e colorarne sfondo e testo. Creiamo quindi una tabella con 6 righe.

```
<table>
    <tr><th>Cognome</th><th>Nome</th><th>Eta</th></tr>
    <tr><td>Rossi</td><td>Mario</td><td>23</td></tr>
    <tr><td>Verdi</td><td>Giuseppe</td><td>33</td></tr>
    <tr><td>Bianchi</td><td>Bianca</td><td>24</td></tr>
    <tr><td>Gialli</td><td>Gianna</td><td>23</td></tr>
    <tr><td>Neri</td><td>Luca</td><td>43</td></tr>
</table>
```

Poi scriviamo il codice css per selezionare la righe (<tr>) dispari e cambiarne colore di testo e sfondo:

```
tr:nth-child(2n+1) {
    background:green;
    color:white;
}
```

Possiamo osservare il risultato qui:
http://www.alessandrostella.it/lato_client/css3/css3_09.html

Ora torniamo a quanto detto prima e cerchiamo di comprendere la formula (an+b).
Nel nostro esempio (an+b)=(2n+1) e quindi a=2 e b=1. Da ciò deriva che vogliamo selezionare la prima riga (b=1) e vogliamo ripetere la selezione ogni 2 righe (a=2). Per quanto riguarda "n" abbiamo detto che se a>0 (e nel nostro caso a=2), n va da zero a

int(numeroElementi-1/a). Essendo 6 le righe della tabella (cioè gli elementi <tr>) allora int(numeroElementi-1/a) = int((6-1)/2) = 2. Quindi n va da zero a 2, da cui
(2*0+1)=1
(2*1+1)=3
(2*2+1)=5
Quindi con la formula (2n+1) e 6 righe di tabella, saranno selezionate le righe (<tr>) 1, 3 e 5.
Ovviamente la formula tra parentesi può cambiare a nostro piacimento. Ad esempio il codice css:

```
tr:nth-child(2n+2) {
    background:green;
    color:white;
}
```

farebbe selezionare gli elementi
(2*0+2)=2
(2*1+2)=4
(2*2+2)=6

così come il codice

```
tr:nth-child(3n+2) {
    background:green;
    color:white;
}
```

farebbe selezionare gli elementi
(3*0+2)=2
(3*1+2)=5

Non è necessario usare la formula an+b per selezionare un elemento che si trova in una posizione nota. Ad esempio, per selezionare la prima riga (e solo la prima) basta scrivere:

```
tr:nth-child(1) {
    background:green;
    color:white;
}
```

Per agevolare le operazioni più comuni, ossia la selezione degli elementi in posizione pari e di quelli in posizione dispari, il w3c ha aggiunto 2 valori che possono essere usati al posto della complicata formula an+b. In particolare, per selezionare gli elementi in posizione dispari si può usare il seguente codice:
tr:nth-child(**odd**) {...}

mentre per selezionare gli elementi in posizione pari:
tr:nth-child(**even**) {...}

Vediamo ora alcuni casi particolari, specificando come verranno interpretate le regole dal browser.

```
/* le seguenti regole sono del tutto identiche */
tr:nth-child(2n+0)
tr:nth-child(2n)
tr:nth-child(even)

/* le seguenti regole sono del tutto identiche */
tr:nth-child(2n+1)
tr:nth-child(odd)

/* le seguenti regole sono del tutto identiche */
tr:nth-child(0n+5)
tr:nth-child(5)

/* le seguenti regole sono del tutto identiche */
tr:nth-child(1n+0)
tr:nth-child(n+0)
tr:nth-child(n)

/* le seguenti regole sono del tutto identiche */
tr:nth-child(10n-1)  /* selezionerà l'elemento 9, 19, 29, ecc */
tr:nth-child(10n+9) /* selezionerà l'elemento 9, 19, 29, ecc */
```

Soffermiamoci un attimo sull'ultimo esempio. La prima regola usa un valore negativo. La cosa importante quindi è districarsi bene tra le formule matematiche.

:nth-last-child()

Funzionamento identico alla pseudo-classe precedente, ma partendo dall'ultimo fratello. Quindi, riferendoci alla tabella e alle sue righe, dall'ultima riga. :nth-last-child(n) identifica l'elemento che è l'n-esimo figlio del suo elemento padre partendo dall'ultimo fratello.
Gli esempi fatti in precedenza sono del tutto validi anche per questa pseudo-classe.
Quindi, per non essere ripetitivi, analizzeremo solo il caso base.
Supponendo di avere la solita tabella dei precedenti esempi, un codice come il seguente:

```
tr:nth-last-child(2) {
    background-color: yellow
}
```

evidenzierà in giallo la seconda riga a partire dall'ultimo elemento fratello, ossia la penultima riga.
Possiamo vedere il codice in azione qui:
http://www.alessandrostella.it/lato_client/css3/css3_10.html

:last-child

Questa pseudo-classe seleziona l'elemento ultimo figlio del suo elemento genitore.
Notiamo la mancanza di parentesi tonde nella sintassi; ciò significa che questa pseudo-classe non accetta parametri.

Il funzionamento è molto semplice. Continuando a utilizzare la tabella precedente, la seguente regola CSS:

```
tr:last-child {
background-color: yellow
}
```

evidenzierà in giallo l'ultima riga della tabella.

:nth-of-type()

Questa pseudo-classe si occupa di selezionare gli elementi per posizione (nth) e per tipo (type). La pseudo-classe infatti identifica ogni elemento che è l'n-esimo fratello del suo tipo all'interno del DOM. Detto in parole più semplici, supponendo che un oggetto abbia diversi figli, di tipo diverso, con tale pseudo-classe si possono raggiungere solo gli elementi del tipo di cui abbiamo bisogno.

Vediamo un semplice esempio per capirne il funzionamento. Prendiamo come base il seguente codice HTML:

```
<div>
    <h3>Questo è un h3 e non sarà evidenziato</h3>
    <p>Primo paragrafo</p>
    <p>Secondo paragrafo</p>
    <h4>Questo è un h4 e non sarà evidenziato</h4>
    <p>Terzo paragrafo</p>
    <p>Quarto paragrafo</p>
    <p>Quinto paragrafo</p>
</div>
```

Sappiamo benissimo che annidati in un elemento possono esserci diversi tipi di elementi allo stesso livello di annidamento. Nel nostro caso dentro il abbiamo 5 paragrafi, un <h3> e un <h4> tutti allo stesso livello, il primo. Se ora noi volessimo raggiungere solo gli elementi di tipo <p> e solo quelli in posizione dispari, potremmo utilizzare la seguente regola:

```
div p:nth-of-type(2n-1) {
    background-color: yellow
}
```

Possiamo vedere un esempio qui:
http://www.alessandrostella.it/lato_client/css3/css3_11.html

Come nth-child(), anche questa pseudo-classe può contenere all'interno delle formule quindi il funzionamento è del tutto simile.
Qualcuno particolarmente attento potrebbe essersi accorto che, facendo riferimento alla solita tabella con 6 righe, il codice

```
tr:nth-of-type(2n+1) {
    background:green;
    color:white;
}
```

equivale a

```
tr:nth-child(2n+1) {
```

```
    background:green;
    color:white;
}
```

entrambi infatti, seppur per motivi diversi, selezioneranno le righe dispari della tabella. Il primo codice seleziona per tipo (<tr>) il secondo per parentela.
Tuttavia i figli di <table> sono tutti <tr>, quindi non ha molto senso usare questa pseudo-classe per selezionare elementi che sono tutti dello stesso tipo.

:nth-last-of-type()

Come per nth-last-child(), anche per questa pseudo-classe il funzionamento è del tutto uguale alla precedente, solo che si inizia a contare dall'ultimo elemento.
Volendo selezionare il penultimo elemento di tipo p all'interno di un div, quindi, abbiamo bisogno del seguente codice:

```
div p:nth-last-of-type(2) {
    background-color: yellow
}
```

:first-of-type

Questa pseudo-classe seleziona il primo elemento in qualità di primo figlio del suo tipo.
Concettualmente essa coincide all'uso di :nth-of-type(1).
Dato il codice utilizzato negli esempi precedenti, la seguente regola:

```
div p:first-of-type {
    background-color: yellow
}
```

selezionerà solo il primo paragrafo.

:last-of-type

Questa pseudo-classe seleziona il primo elemento in qualità di ultimo figlio del suo tipo.
Concettualmente essa coincide all'uso di :nth-last-of-type(1).
Dato il codice utilizzato negli esempi precedenti, la seguente regola:

```
div p:last-of-type {
    background-color: yellow
}
```

selezionerà solo l'ultimo paragrafo.

:only-of-type

Questa pseudo-classe identifica un elemento il cui oggetto padre non ha altri figli dello stesso tipo. La sintassi non prevede le parentesi tonde e quindi non ci sono parametri.
Quindi, dato un codice come il seguente:

```
<div>
    <p>Lorem ipsum dolor sit amet.</p>
    <ul>
          <li>List item 1</li>
          <li>List item 2</li>
    </ul>
</div>
<div>
    <p>Lorem ipsum dolor sit amet.</p>
    <p>Lorem ipsum dolor sit amet.</p>
    <ul>
          <li>List item 1</li>
          <li>List item 2</li>
    </ul>
<div>
```

e questo codice CSS:

```
div p:only-of-type {
    background: yellow
}
```

verrà selezionato solo il primo paragrafo del primo contenitore perché, nel secondo contenitore, sono presenti due paragrafi e non solo uno.
Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/css3/css3_12.html

:only-child

Questa pseudo-classe identifica un elemento il cui oggetto padre non ha altri figli. La sintassi non prevede le parentesi tonde e quindi non ci sono parametri.

Supponendo di avere un codice HTML come il seguente:

```
<p>Lorem ipsum <span>dolor</span> sit amet.</p>
<p>Lorem ipsum <span>dolor</span> sit <span>amet</span>.</p>
```

e utilizzando questa regola CSS:

```
p span:only-child {
    background-color: yellow
}
```

verrà inserito un colore giallo di sfondo solo allo span del primo paragrafo perché esso contiene un solo elemento , mentre nel secondo paragrafo ne sono presenti due.

:empty

Questa pseudo-classe identifica ogni elemento che non contiene figli. Bisogna notare che :empty include anche i nodi di testo. Quindi, un semplice paragrafo che contiene del testo al suo interno non verrà selezionato da questa pseudo-classe.
Ad esempio, data la regola CSS:

```
p:empty {
    background-color: yellow
}
```

e il seguente codice HTML:

```
<p></p>
<p><span></span></p>
<p>Lorem ipsum</p>
```

otterremo la selezione del primo paragrafo, ma non del secondo né del terzo

Altre pseudo-classi

Giunti alla fine di questo modulo prendiamo in esame alcune pseudo-classi che non appartengono a nessuna delle precedenti categorie. Esse sono:

- :target
- :lang
- :not

:target

La pseudo-classe :target ha una potenzialità davvero interessante. Ci consente di selezionare un elemento della pagina che corrisponde ad un indirizzo di riferimento. Più precisamente, data una pagina con delle àncore al proprio interno, la pseudo-classe :target ci consente di assegnare uno stile all'elemento di destinazione dell'àncora nel momento in cui è selezionato.

Vediamo un esempio per comprenderne il funzionamento partendo dal codice HTML:

```
<ul>
    <li><a href="#div1">Div1</a></li>
    <li><a href="#div2">Div2</a></li>
    <li><a href="#div3">Div3</a></li>
    <li><a href="#div4">Div4</a></li>
</ul>
<div id="div1"><p>Lorem ipsum dolor sit amet ...</p></div>
<div id="div2"><p>Sit et et scelerisque Phasellus ...</p></div>
<div id="div3"><p>Adipiscing eros quis eu fringilla..</p></div>
<div id="div4"><p>Condimentum Vivamus iaculis...</p></div>
```

Abbiamo una lista di link. Ciascuno presenta come valore di href il riferimento ad un div di destinazione identificato dal suo id. Nulla di speciale insomma, una struttura di navigazione interna alla pagina vista e stravista.

Ora definiamo questa semplicissima regola CSS:

```
div:target {border: 3px solid red}
```

In questo modo, possiamo mettere in evidenza il div selezionato, senza aver bisogno di utilizzare Javascript o altri hack particolari. Abbiamo infatti espresso con i CSS questa regola: su tutti i div che sono anche target di un àncora, quando sono attivati, impostiamo un bordo rosso spesso 3px.

:lang

Questa pseudo-classe ha come scopo quello di selezionare gli elementi HTML che presentano l'attributo lang. In particolare esegue la selezione in base al codice della lingua che gli viene passato come parametro :lang(codice).

Ipotizzando di avere il seguente codice HTML:

```
<body>
    <p lang=en>This is written in english.</p>
</body>
```

La regola CSS

```
:lang(en) {
    background:black;
}
```

setta a nero lo sfondo di <p>.

Osserviamo che sarebbe stato corretto anche il seguente codice HTML:

```
<p lang=”en”>This is written in english.</p>
```

dove il valore dell'attributo lang è riportato tra doppi apici.

Circa i codici delle lingue (en, it, fr, ecc.), il w3c invita a fare riferimento al seguente documento:

http://www.iana.org/assignments/language-subtag-registry

In tale documento, il codice della lingua è riportato nella variabile Subtag.

:not

Questa pseudo-classe identifica tutti gli elementi che non coincidono con il selettore contenuto all'interno del :not.

Cerchiamo di chiarire il concetto con qualche esempio. E partiamo da questo codice HTML:

```
<div class="nero">Lorem ipsum</div>
<div class="rosso">Lorem ipsum</div>
```

Usando questo codice CSS:

```
div:not(.nero) {color: red}
```

assegniamo un testo di colore rosso ai div che non abbiano come classe .nero.

Vediamo un'altra demo che verrà ripresa anche nella prossima sezione. Il seguente codice

```
input:not(:disabled){color: red; border: 1px solid black}
```

seleziona tutti gli elementi di tipo input che non sono disabilitati.

Gli pseudo-elementi

Gli pseudo-elementi danno la possibilità di effettuare selezioni di parti del documento HTML che diversamente non sarebbe possibile raggiungere. Ad esempio, normalmente non abbiamo meccanismi per accedere alla prima lettera o alla prima riga del contenuto di un elemento. Gli pseudo-elementi permettono agli autori di fare riferimento a queste informazioni altrimenti inaccessibili. Gli pseudo-elementi possono anche fornire agli autori un modo per fare riferimento a contenuti che non esistono nel documento di origine.
La sintassi prevede una coppia di due punti (::) seguita dal nome dello pseudo-elemento.
La **notazione dei 4 punti (::)** è stata introdotta al fine di distinguere tra pseudo-classi e pseudo-elementi. Ecco gli pseudo-elementi:
::first-line
::first-letter
::before
::after

::first-line

Questo pseudo-elemento ha come scopo quello di selezionare la prima riga di un testo.
Ad esempio, dato il codice HTML:
Questa è la prima riga del paragrafo
mentre questa è la seconda

Il codice CSS

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

cambierà in rosso il colore del testo presente sulla prima riga del paragrafo e lascerà inalterato il colore del testo sulla seconda riga.

::first-letter

Questo pseudo-elemento seleziona la prima lettera di un testo.
Esempio.
Dato il codice HTML:

```
<p>Questo è un testo qualsiasi</p>
```

Il codice CSS

```
p::first-letter {
    color:red;
    font-size:32pt;
}
```

cambierà in rosso il colore della lettera Q, la renderà molto più grande del resto del testo e lascerà inalterato il colore del resto del testo.
Possiamo osservare il tutto qui:
http://www.alessandrostella.it/lato_client/css3/css3_13.html

::before e **::after**
Questi due pseudo-elementi hanno come scopo quello di inserire contenuto prima (::before) o dopo (::after) un determinato elemento. 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.
Con ::after si fa il contrario, cioè si inserisce il contenuto dopo.
Esempio.

```
<p>Questo è sempre il solito paragrafo.</p>
```

```
p::after {
    content: " E questo testo è aggiunto dopo."
}
```

Il risultato può essere visualizzato qui:
http://www.alessandrostella.it/lato_client/css3/css3_14.html

I combinatori

Una categoria fondamentale di selettori CSS è rappresentata dai cosiddetti combinatori (detti anche selettori di relazione). Hanno la funzione di mettere in relazione elementi presenti all'interno dell'albero del documento. Sono quattro:

- Combinatore di discendenti (uno spazio bianco tra selettori semplici)
- Combinatore di figli (>)
- Combinatore di fratelli adiacenti (+)
- Combinatore generale di fratelli (~)

Combinatore di discendenti

Il combinatore di discendenti è sicuramente quello più utilizzato dei quattro. Non è presente solo nella specifica CSS3, ma anche nelle precedenti versioni ed è utilissimo per evitare l'abuso delle classi per assegnare stili agli elementi. Il combinatore seleziona un elemento che è discendente di un altro elemento. Vediamo un brevissimo esempio per capirne il funzionamento. Ammettiamo di avere il seguente codice HTML:

```
<div id="esempio">
    <span>Testo non evidenziato</span>
    <p>Un Paragrafo.</p>
    <p>Un altro paragrafo</p>
</div>
<div id="altro">
    <p>Paragrafo non interessato</p>
</div>
```

Quello che vogliamo ottenere è evidenziare in rosso il testo dei paragrafi del #esempio, ma non quelli del #altro.
Usando il selettore di discendenti sarà molto semplice.

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

Con questo codice CSS abbiamo selezionato tutti i paragrafi figli di un con id=”esempio”. Da notare lo spazio presente tra il primo selettore, div#esempio, e il selettore del paragragrafo, p. Infatti per impostare la relazione di discendenza, è sufficiente **separare l'elemento antenato dal discendente con uno spazio**.

Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/css3/css3_15.html

Combinatore di figli

Il combinatore di figli (>) consente di selezionare un elemento che è figlio diretto dell'elemento padre. Questo selettore è solo in apparenza simile al selettore di discendenti. La differenza sta nella relazione di discendenza tra gli elementi, che in questo caso deve essere di primo livello. Chiariamo con un esempio:

```
<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 . Tutti e tre, però, sono discendenti di <body>. Pertanto, usando il seguente codice css:

```
body > p {color: red}
```

solo il primo e il terzo paragrafo avranno il testo rosso.

Combinatore di fratelli adiacenti

Il combinatore di fratelli adiacenti serve a scorrere in orizzontale l'albero del DOM assegnando le regole CSS agli elementi che si trovano allo stesso livello di un altro elemento. La relazione si definisce collegando i due elementi con il segno +.
In pratica, questo tipo di selettore consente di assegnare uno stile all'elemento fratello immediatamente adiacente. Dato un codice HTML come il seguente:

```
<div>
    <h1>1. Questo è il titolo principale.</h1>
    <h2>1.1 Questo è il primo sottotitolo.</h2>
    <p>...</p>
    <h2>1.2 Questo è il secondo sottotitolo.</h2>
    <p>...</p>
</div>
```

Applicando la seguente regola

```
h1 + h2 {
    color: red;
    text-decoration: underline
}
```

verrà selezionato solo il primo <h2> dato che è immediatamente adiacente al tag <h1>. Possiamo vedere il codice in azione qui:.
http://www.alessandrostella.it/lato_client/css3/css3_16.html

Consideriamo, invece, il seguente codice HTML

```
<ul>
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>List Item 3</li>
    <li>List Item 4</li>
    <li>List Item 5</li>
</ul>
```

e la seguente regola CSS:

```
li + li {
    margin-left:10px;
    color: red
}
```

In questo caso il combinatore andrà a selezionare tutti quegli elementi che sono diretti fratelli del tag <li> e, scorrendo in orizzontale l'albero del DOM, solo il primo <li> non coinciderà con la proprietà (perché non ha nessun fratello <li> che lo precede). Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/css3/css3_17.html

Combinatore generale di fratelli

L'ultimo combinatore (~) è una generalizzazione di quello visto in precedenza. Esso assegna uno stile a tutti gli elementi che sono fratelli. Dato il codice HTML visto in precedenza

```
<div>
    <h1>1. Questo è il titolo principale.</h1>
    <h2>1.1 Questo è il primo sottotitolo.</h2>
    <p>...</p>
    <h2>1.2 Questo è il secondo sottotitolo.</h2>
    <p>...</p>
</div>
```

e applicando il codice CSS seguente

```
h1 ~ h2 {
    color:red;
    text-decoration:underline
}
```

andremo a selezionare tutti gli elementi <h2> dello stesso livello di <h1> indipendentemente dalla posizione che occupano.
Possiamo osservare il risultato qui:
http://www.alessandrostella.it/lato_client/css3/css3_18.html

Analizziamo invece il seguente caso. Dato questo codice:

```
<div>
    …
    <h2>Questo e' un sottotitolo h2</h2>
    <h3>Questo e' un sottotitolo h3</h3>
    …
    <h2>Questo e' un sottotitolo h2</h2>
    <h3>Questo e' un sottotitolo h3</h3>
    …
    <h2>Questo e' un sottotitolo h2</h2>
    <h3>Questo e' un sottotitolo h3</h3>
</div>
```

e la seguente regola CSS

```
h3 ~ h2 {color: red; text-decoration: underline}
```

notiamo che al primo <h2> presente nel codice non viene applicato alcuno stile, mentre a tutti gli altri sì. Questo succede perché la regola viene assegnata solo agli elementi che sono fratelli e successori dell'elemento <h3>.

CSS Media Queries

Dal 2 luglio 2012 questo modulo è in stato REC, quindi è una raccomandazione w3c. Inoltre questo è un modulo molto importante per le attuali tecnologie perché ci consente di disegnare le nostre pagine web in modo diverso a seconda del tipo di device che si collega al nostro sito, ossia caricare css diversi a seconda del dispositivo. In un'epoca in cui smartphone e tablet sono sempre più diffusi non è pensabile produrre un sito web ideato esclusivamente per il monitor di un PC. Per questi motivi dedicheremo tutto lo spazio necessario a comprenderne l'utilizzo.
Il concetto non è certo nuovo. Già HTML 4 e CSS2 erano in grado di gestire fogli di stile distinti per device (dispositivi) distinti. Tutto ciò che era valido resta valido.
Ricordiamo quindi che in HTML 4 si potevano richiamare più fogli di stile da applicare a diversi tipi di media. Esempio:

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

In questo modo il browser applicava il foglio di stile "style.css" ad alcuni tipi di dispositivi (screen), mentre usava il foglio di stile "stampa.css" per altri tipi (print).
C'era anche un altro modo di ottenere lo stesso risultato.
Invece di usare HTML per far caricare un file CSS diverso a seconda del dispositivo, potevamo caricare un unico foglio di stile per tutti i dispositivi. In tal caso però, all'interno dell'unico foglio di stile, dovevamo usare la direttiva "@media" per decidere quale parte del foglio di stile caricare a seconda del dispositivo. Esempio:

```
@media print {
    * { font-family: sans-serif }
}
```

Tutto ciò è ancora valido nei CSS3.

Secondo il w3c, **per definire una media query serve il tipo di media (video, stampante, proiettore, ecc.) e un insieme di condizioni che ne identificano le caratteritsiche (risoluzione, colori, ecc.)**.
In altre parole una media query viene usata dal browser al fine di capire come mostrare la pagina web e per fare questo servono due informazioni:

- il tipo di dispositivo
- le caratteristiche di dettaglio (opzionale)

Nei prossimi paragrafi vedremo come comunicare queste informazioni al browser. Ora invece occupiamoci della sintassi di una media query.

La sintassi

Abbiamo detto che possiamo definire una media query sia in HTML, sia in un foglio di stile. Ovviamente la sintassi sarà diversa.
Vediamo la **sintassi** da usare per scrivere una media query in **HTML**.

<link rel="stylesheet" media="<tipo di dispositivo> {and (<caratteristica>)... and (<caratteristica>)}" href="<css>" />

ciò che si trova tra parentesi graffe può essere omesso.
Ecco un esempio pratico:

```
<link rel="stylesheet" media="screen and (width:320px)
    and (orientation:portrait)" href="mioCss.css" />
```

in cui
<tipo di dispositivo> = screen
<caratteristica> = width:320px
<caratteristica> = orientation:portrait
<css> = mioCss.css

Questa media query quindi identifica un dispositivo con uno schermo a colori (screen) che abbia una risoluzione di 320px (width:320px) e sia in posizione verticale (orientation:portrait). Per tale tipo di dispositivo (tipicamente uno smartphone) viene caricato il CSS "mioCss.css". Ovviamente saremo costretti a caricare un figlio di stile diverso per ogni dispositivo che ci interessa identificare. Ad esempio, se volessimo

caricare un foglio di stile da usare per la stampa, oltre a quello usato per lo smartphone, dovremmo aggiungere un altro tag <link> al nostro file HTML:

```
<link rel="stylesheet" media="screen and (width:320px) and
     (orientation:portrait)" href="mioCss.css" />
<link rel="stylesheet" media="print" href="mioCssStampa.css" />
```

Abbiamo però un'alternativa.
Invece di caricare n fogli di stile nel file HTML, possiamo caricare un unico foglio di stile da HTML e delegare il riconoscimento del dispostivo al foglio di stile. Quindi in HTML inseriamo un unico tag <link> che richiama un unico foglio di stile per tutti i dispositivi, per esempio:

```
<link rel="stylesheet" media="all" href="mioCss.css" />
```

e poi nel foglio di stile "mioCss.css" scriviamo codice css diverso a seconda del dispositivo collegato. In questo caso la **sintassi** di una media query eseguita in un **CSS** diventa la seguente:

@media <tipo di dispositivo> {and (<caratteristica>)... and (<caratteristica>)}

ciò che si trova tra parentesi graffe (cioè le caratteristiche del dispositivo) può essere omesso.
Usando questa sintassi, il codice da scrivere per ottenere lo stesso risultato ottenuto in HTML con il caricamento di file css diversi, è il seguente:

```
@media screen and (width:320px) and (orientation:portrait) {
    ...
}
@media print {
    ...
}
```

Il codice che si trova dopo @media verrà eseguito solo se il dispositivo collegato alla nostra pagina web rispetta le caratteristiche indicate.
Ora che conosciamo la sintassi di una media query, entriamo nel dettaglio dei dispositivi e delle loro caratteristiche.

I tipi di dispositivi

Per formattare la nostra pagina web in un certo modo, dobbiamo innanzitutto identificare il tipo di dispositivo che la sta visualizzando. Al momento, i tipi di dispositivi identificabili tramite le media queries sono i seguenti:

all

Identifica ogni tipo di dispositivo.

braille

Dispositivo di tipo braille (il braille è un sistema di scrittura e lettura a rilievo per non vedenti ed ipovedenti messo a punto dal francese Louis Braille nella prima metà del XIX secolo).

embossed

Identifica una stampante di tipo braille.

handheld

Identifica dispositivi mobili (handheld devices) con schermo piccolo e in genere dotati di browser con limitate capacità grafiche.

print

Identifica una pagina che è formattata per la stampa.

projection

Identifica un dispositivo tipo proeittore.

screen

Identifica, molto in generale, un dispositivo equipaggiato con un monitor a colori.

speech

Identifica un sintetizzatore vocale. Poniamo attenzione al fatto che i CSS2 usano il tipo 'aural' per questo scopo.

tty

Identifica un dispositivo che utilizza il "fixed-pitch character grid", cioè carattere fisso (come i terminali).

tv

Identifca un dispositivo tipo televisore (bassa risoluzione, colori, audio disponibile).

Questi sono tutti i dispositivi che le media queries sono in grado di identificare autonomamente. Ad esempio, per identificare smartphone, tablet e PC viene usato il tipo "screen". Lavorando poi sulle caratteristiche dello "screen" saremo in grado di decidere la precisa tipologia di dispositivo. Vediamo allora quali sono le caratteristiche specifiche che abbiamo a disposizione.

Le caratteristiche dei dispositivi

Una volta identificato il tipo di dispositivo che sta visualizzando la nostra pagina, possiamo anche identificarne alcune caratteristiche più precise e agire di conseguenza. Molte delle caratteristiche qui di seguito elencate accettano i prefissi "min-" e "max-", come indicato tra parentesi quadre.

Ecco l'elenco delle caratteristiche che possiamo usare:

- [min-][max-] width
- [min-][max-] height
- [min-][max-] device-width
- [min-][max-] device-height
- orientation
- [min-][max-] aspect-ratio
- [min-][max-] device-aspect-ratio
- [min-][max-] color
- [min-][max-] color-index
- [min-][max-] monochrome
- [min-][max-] resolution
- scan
- grid

Vediamone il relativo significato e alcuni esempi di utilizzo.

Width

Questa è una caratteristica tanto importante quanto delicata. Per comprenderla fino in fondo dobbiamo fare una piccola digressione sul concetto di **viewport**. Nel nostro caso una pagina web viene mostrata sempre da un Browser. Osserviamo la seguente figura.

Possiamo facilmente notare come il contenuto della pagina web si mantenga al centro. Ora proviamo a restringere la finestra del browser. Quello che succede è che, quando la finestra diventa troppo stretta per contenere tutto il contenuto della pagina web, il browser attiva la barra di scorrimento orizzontale (visibile in basso).

Osservando questo comportamento possiamo comprendere il concetto di viewport. **Il viewport infatti è la parte del browser che si occupa di contenere la pagina web** e non sempre il viewport coincide con la larghezza della finestra del browser.

Chiarito questo concetto passiamo adesso a parlare della caratteristica width.
Width indica la larghezza dell'area di visualizzazione del documento. Su un normale browser web essa è rappresentata proprio dal viewport.
I valori di width possono essere espressi con qualunque unità di misura.

```
@media screen and (width: 400px) {
    /* regole CSS */
}
```

Come abbiamo accennato, la caratteristica width può essere preceduta dai prefissi "min-" e "max-", ottenendo così "min-width" e "max-width". È proprio così che esplica tutta la sua utilità, specie nella realizzazione di layout. La seguente espressione, per esempio, applica un certo stile quando le dimensioni sono comprese tra 400px e 1024px:

```
@media screen and (min-width: 400px) and (max-width: 1024px) {
    /* regole CSS */
}
```

Facciamo ora un esempio reale.
Vogliamo che un titolo <h1> cambi la dimensione del proprio testo a seconda della larghezza del viewport.
Possiamo osservare il codice in azione qui (allargando e restringendo la finestra del browser):
http://www.alessandrostella.it/lato_client/css3/css3_19.html

Questo è il codice HTML:

```
<body>
    <h1>Cambio dimensione a seconda del width.</h1>
</body>
```

e questo il codice CSS:

```
<style type="text/css">
    @media  screen and (max-width:1024px) {
          h1 {
                font-size:16pt;
          }
    }
    @media  screen and (max-width:700px) {
          h1 {
                font-size:14pt;
          }
```

```
    }
    @media  screen and (max-width:320px) {
          h1 {
                font-size:12pt;
          }
    }
</style>
```

Se il width del viewport è superiore a 1024px, il titolo <h1> ha il testo di dimensione standard. Se il width del viewport è tra i 700px e i 1024 px, il testo viene ridimensionato a 16pt. Tra 320px e 700px il testo ha una dimensione di 14pt. Infine sotto i 320px il testo ha una dimensione di 12pt.

Height

La caratteristica **height** indica l'altezza dell'area di visualizzazione del documento. Come nel caso di width può essere definita con i prefissi min- (min-height) e max- (max-height).

Device-width

Con **device-width** si descrive la larghezza dell'intera area di rendering di un dispositivo. Su un computer o su un dispositivo mobile si intende la larghezza dell'intero schermo del dispositivo, non semplicemente quindi dell'area di visualizzazione del documento! Pertanto ricordiamoci che **la caratteristica width si riferisce al viewport, mentre device-width si riferisce alla effettiva larghezza dello schermo del dispositivo**. Accetta i prefissi min- (min-device-width) e max- (max-device-width).
<link rel="stylesheet" media="all and (max-device-width: 480px)" href="iphone.css" type="text/css" />

Device-height

Banalmente descrive l'altezza dell'intera area di rendering di un dispositivo. Accetta i prefissi min- e max-.

Orientation

La caratteristica **orientation** indica se il dispositivo di output è in modalità landscape (la larghezza è maggiore dell'altezza) oppure portrait (l'altezza è maggiore della larghezza). Ci dice, in altre parole, se abbiamo o meno ruotato lo smartphone o il tablet. I valori che può assumere sono appunto landscape (ruotato) o portrait (non ruotato).

Di seguito possiamo osservarne l'utilizzo di questa caratteristica nel caso di un tipico smartphone con risoluzione 320x480px.

```
<link rel="stylesheet" media="screen and (max-width:320px) and
    (orientation:portrait)" href="portrait.css"
    type="text/css" />
<link rel="stylesheet" media="screen and (max-width:480px) and
    (orientation:landscape)" href="landscape.css"
    type="text/css" />
```

Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/css3/css3_20.html

Per osservare il risultato in un browser è necessario sia restringere la larghezza della finestra sia restringerne l'altezza per fare in modo di simulare rispettivamente portrait (altezza maggiore della larghezza) e landscape (larghezza maggiore dell'altezza).

Aspect-ratio

Definisce il rapporto tra la larghezza (width) e l'altezza (height) dell'area di visualizzazione di un documento. Può assumere i prefissi min- e max-.
I valori si esprimono attraverso due numeri interi separati dal simbolo /. Tipici esempi sono: 3/4, 16/9, etc.

Device-spect-ratio

Descrive il rapporto tra la larghezza (device-width) e l'altezza (device-height) dell'area di rendering di un dispositivo. Può assumere i prefissi min- e max-.

```
@media screen and (device-aspect-ratio: 16/9) {
    /* regole CSS */
}
```

Con questa media query creiamo una regola CSS che si applica solo a schermi con un rapporto di 16/9.

Color

Tecnicamente questa caratteristica indica il numero di bit per ciascuna componente colore sul dispositivo di output. Color infatti può assumere come valori dei numeri interi:
0 (zero) designa un dispositivo che non supporta il colore.

Nella pratica, comunque, salvo che non si abbiano esigenze molto specifiche, si può usare nella sua forma più semplice, andando così a indirizzare i CSS ai dispositivi che supportano il colore:

```
@media all and (color) {
    /* regole CSS */
}
```

Color-index

Descrive il numero di colori presenti nella tavolozza supportata da un certo dispositivo. Può assumere i prefissi min- e max-.

```
<link rel="stylesheet" media="all and (min-color-index: 256)"
    href="colore.css" />
```

Secondo questa media query, il foglio di stile colore.css sarà applicato su dispositivi che supportano come minimo una tavolozza di 256 colori.

Monochrome

Indica il numero di bit per pixel su un dispositivo monocromatico (a scala di grigi).

Resolution

Descrive la risoluzione (ovvero la densità di pixel) del dispositivo di output. I valori della risoluzione possono essere espressi in dpi (punti per pollice) oppure in dpcm (punti per centimetro). Può assumere i prefissi min- e max-.

```
@media print and (min-resolution: 300dpi) {
    /* regole CSS */
}
```

Scan

Si tratta di una caratteristica valida per gli schermi televisivi, quindi per il tipo di media tv. Indica il tipo di scansione, interlacciata o progressiva. I valori possono essere progressive oppure interlace.

Grid

Indica se il dispositivo è di tipo bitmap oppure 'a griglia', come certi terminali monocromatici e gli schermi per cellulari con un solo font.

CSS Backgrounds and Borders Module Lev. 3

Questo modulo si trova nello stato PR ed è pertanto pronto per essere implementato dai browser. E' un modulo molto corposo e ciò dipende dal fatto che esso riunisce in un unico modulo due precedenti direttive: CSS3 Backgrounds e CSS3 Border.
Come si intuisce dal nome, questo modulo si occupa di standardizzare la gestione dei bordi e degli sfondi; include ed estende le funzionalità di CSS di livello 2.
Le principali estensioni rispetto al livello 2 sono le seguenti:

- bordi costituiti da immagini
- box con sfondi multipli
- box con angoli arrotondati
- box con le ombre

Le novità sono quindi molteplici e ben ghiotte perché risolvono annosi problemi dei precedenti CSS. Ovviamente per comprendere fino in fondo il contenuto di questo modulo è necessario avere ben chiaro il concetto del **box model**, così come lo abbiamo visto quando abbiamo studiato i CSS2 (ricordiamo?). Ecco qui di seguito l'immagine che abbiamo usato durante lo studio del box model.

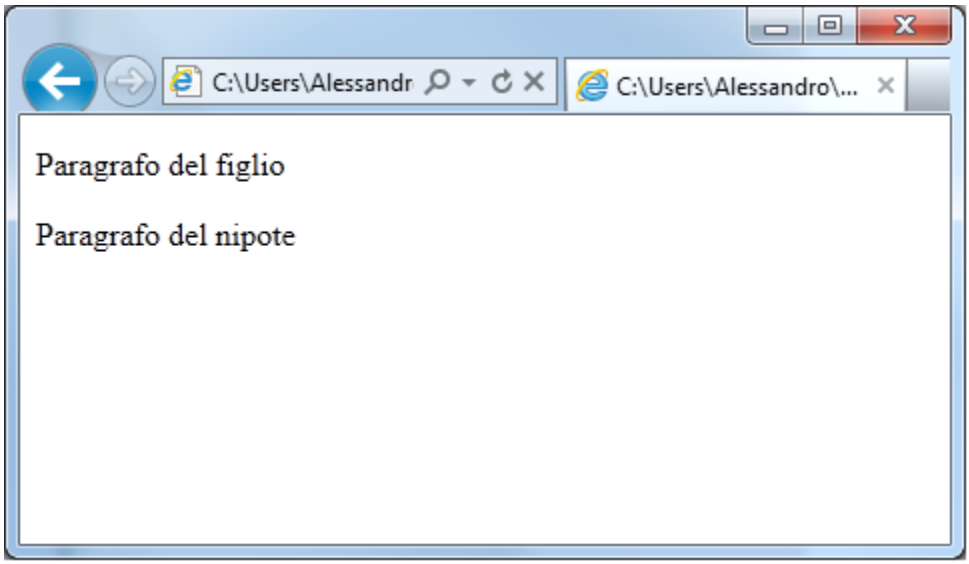
La gestione dello sfondo

Ogni box ha uno sfondo che può essere trasparente (impostazione predefinita), colorato oppure riempito con un'immagine. Se vogliamo dare un colore allo sfondo dobbiamo usare la proprietà **background-color**, mentre per gestire un'immagine come sfondo si usa la proprietà **background-image**. E' bene porre attenzione al fatto che le proprietà

dello sfondo non sono ereditate, ma lo sfondo del box genitore si vedrà attraverso lo sfondo dei box figli a causa del valore iniziale di 'transparent' per lo sfondo dei box figli.
Facciamo subito un rapido esempio su questo concetto della trasparenza.
Dato il seguente codice HTML:

```
<div id="padre">
    <div id="figlio">
        <p>Paragrafo del figlio</p>
        <div id="nipote">
            <p>Paragrafo del nipote</p>
        </div>
    </div>
</div>
```

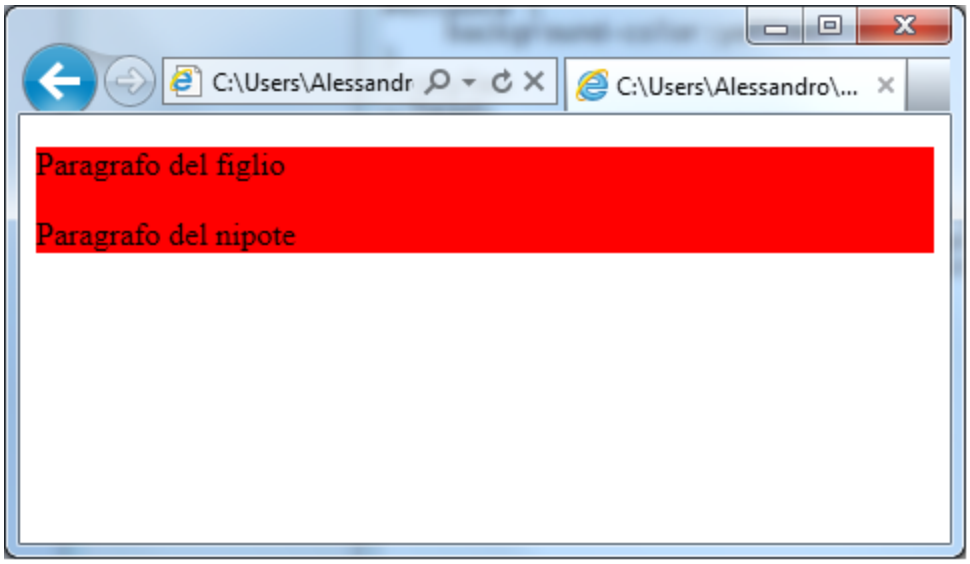
Il browser mostra il seguente risultato.



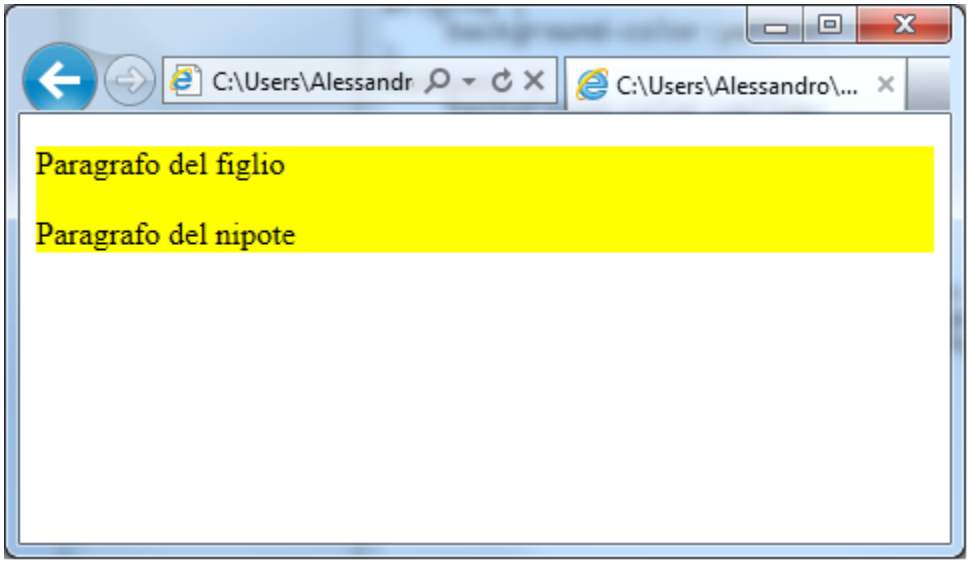
Se però coloriamo di rosso lo sfondo del #padre,

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

lasciando inalterati gli sfondi degli altri elementi, il risultato diventa il seguente.



Apparentemente lo sfondo è rosso per tutti gli elementi HTML, anche se noi lo abbiamo settato solo per il #padre. Ciò accade, come detto, perché lo sfondo degli altri elementi è settato come trasparente per default. Possiamo facilmente dimostrarlo. Ci basta infatti lasciare il #padre con sfondo rosso e contemporaneamente colorare lo sfondo degli altri . Se, ad esempio, coloriamo di giallo lo sfondo del #figlio, il risultato diventa il seguente.



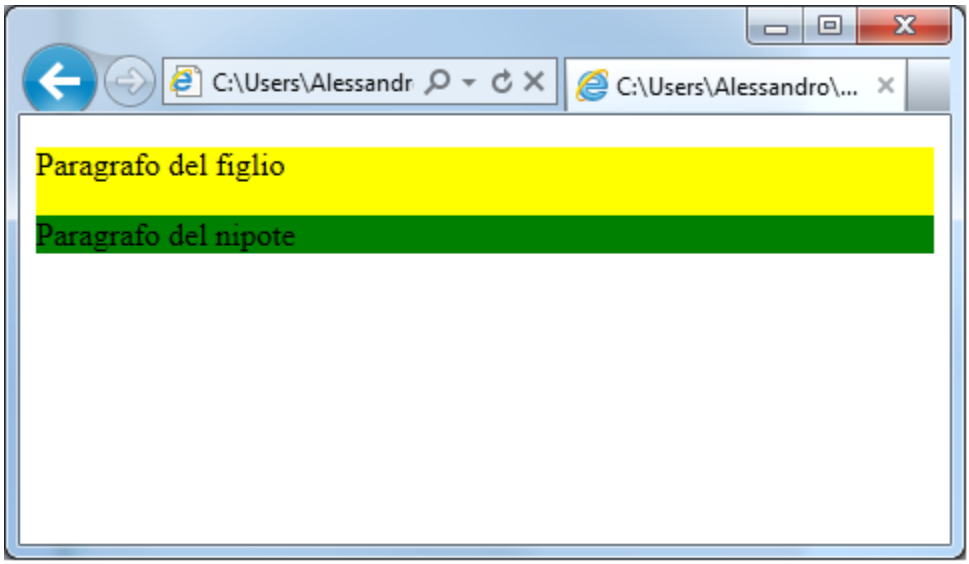
Non deve meravigliarci il fatto che ora il rosso sia completamente scomparso. Il #figlio infatti occupa esattamente lo spazio del #padre, ricoprendolo completamente e facendo quindi "sparire" il colore rosso.

Infine, se coloriamo lo sfondo del #padre in rosso, lo sfondo del #figlio in giallo e lo sfondo del #nipote in verde

```
#padre {
    background-color:red;
```

```
}
#figlio {
    background-color:yellow;
}
#nipote {
    background-color:green;
}
```

il risultato diventa il seguente.



Dovrebbe ora essere chiara la gestione della trasparenza dello sfondo tra elementi HTML. Cosa sarebbe accaduto se, invece di settare i colori di sfondo del #figlio e del #nipote, avessimo settato i colori di sfondo dei 2 paragrafi?

Qui di seguito troviamo un elenco di tutte le proprietà utilizzabili per gestire lo sfondo di un elemento HTML:

- background-color
- background-image
- background-repeat
- background-attachment
- background-position
- background-clip
- background-origin
- background-size
- background

Abbiamo già visto come usare la proprietà background-color. Ora faremo un piccolo esempio di utilizzo congiunto delle proprietà **background-image** e **background-position**.
Useremo queste proprietà e un'unica immagine per mostrare un effetto di hover su un .
Ammettiamo di avere la seguente immagine.

Si tratta di **un'unica immagine** che contiene sia il bottone normale sia il bottone da mostrare quando si passa sopra con il muose. Tramite la proprietà background-image assegneremo questa immagine allo sfondo di un . Poi, tramite la proprietà background-position decideremo quale parte dell'immagine mostrare all'utente (quella scura quando il mouse è lontano dal , quella gialla quando il mouse passa sul).
Quando l'utente passerà con il mouse sul nostro bottone non dovrà attendere che il sistema vada a cercare la relativa immagine sul server perché l'immagine è già nella cache del browser!
Questo è il codice HTML:

```
<div class="twitter">
    <a href="http://twitter.com/alexandrostella"></a>
</div>
```

e questo il rispettivo CSS:

```
a, div {
    display:block;
    width:37px;
    height:37px;
}
div.twitter{
    background-image: url(twitter-icon.png);
    background-position: top;
}
div.twitter:hover{
    background-image: url(twitter-icon.png);
    background-position: 0 -37px;
```

```
}
```

Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/css3/css3_21.html

La gestione dei bordi

Avendo ben chiaro in mente il box model, dovremmo avere altrettanto chiaro cosa intendiamo con il termine border. Per gestire i bordi di un elemento HTML si usano le seguenti proprietà:

- border-color
- border-style
- border-width
- border

La proprietà **border-color** consente di scegliere il colore del bordo. Tuttavia sappiamo che per default i bordi degli elementi HTML non sono visibili. Per prima cosa quindi è opportuno renderli visibili. Per fare questo possiamo usare la seconda proprietà, **border-style**. La sintassi è abbastanza semplice, basta assegnare alla proprietà uno o più valori tra i seguenti:
none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset
Se assegniamo un solo valore, esso varrà per tutti e 4 i lati. Se invece vogliamo differenziare lo stile dei vari bordi, basta assegnare i valori nella seguente sequenza: alto, destra, basso, sinistra. Ad esempio, il codice CSS:
border-style: solid;

assegnerà il valore "solid" a tutti e 4 i bordi.
Invece il codice CSS:
border-style: none dotted solid double;

assegnerà il valore "none" al bordo superiore, il valore "dotted" (tratteggiato) al bordo destro, il valore "solid" al bordo inferiore e infine il valore "double" (doppia linea) al bordo sinistro.
La proprietà **boder-width** gestisce lo spessore del bordo. Anche in questo caso possiamo assegnare spessori diversi ai singoli bordi nello stesso modo usato per la proprietà precedente. Abbiamo a disposizione 3 valori testuali preimpostati: thin, medium, thick. Thin significa sottile, medium significa medio, infine thick significa spesso.

Possiamo anche usare numeri seguiti dall'unità di misura. Ad esempio, il seguente codice CSS:

```
border-width: medium medium thin thin;
```

assegnerà uno spessore medio al bordo superiore e destro, uno spessore sottile al bordo inferiore e sinistro. Mentre il seguente:

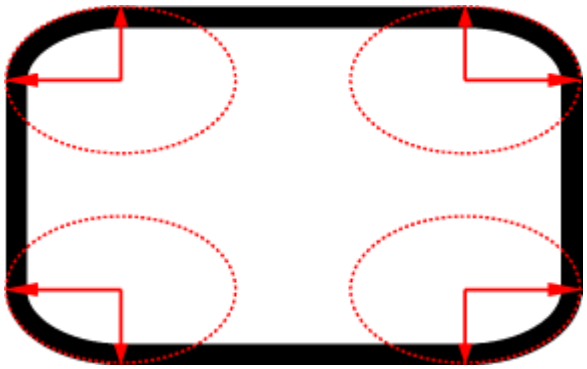
```
border-width: 1px 5px 10px 20px;
```

assegnerà, agli stessi bordi, le dimensioni indicate.
Infine la proprietà **border** è in grado di assegnare tutte le precedenti proprietà in un'unica istruzione. La sintassi è la seguente:
border: <border-width> || <border-style> || <color>
Tuttavia questa proprietà agisce su tutti e quattro i bordi contemporaneamente e non è possibile modificare questo comportamento. Ad esempio il codice CSS:
border: medium solid red;

assegnerà a tutti e quattro i bordi un colore rosso e uno spessore medio.
Possiamo osservare il funzionamento di queste proprietà qui:
http://www.alessandrostella.it/lato_client/css3/css3_22.html

La gestione degli angoli

Questa è una novità alquanto gradita! Possiamo finalmente gestire direttamente da CSS lo smussamento degli angoli del box! Niente più javascript né immagini per gli angoli!
Gli angoli del box sono ora gestiti da 4 ellissi virtuali, uno per ogni angolo, mostrati tratteggiati in rosso nella figura qui sotto. Ogni ellisse è gestito tramite 2 raggi, rappresentati dalle frecce. Se i 2 raggi sono uguali l'ellisse diventa un cerchio. La proprietà che ci consente di gestire l'arrotondamento degli angoli è la proprietà **border-radius**.



Per effettuare l'arrotondamento di un angolo è necessario agire sui due raggi (orizzontale e verticale) di ogni singolo ellisse virtuale. I raggi sono rappresentati in figura dalla 2 frecce rosse. Il sistema inizia ad assegnare i valori dall'ellisse virtuale in alto a sinistra e procede poi in senso orario. Se impostiamo un solo raggio, l'ellisse si trasforma in un cerchio. Vediamo qualche esempio.

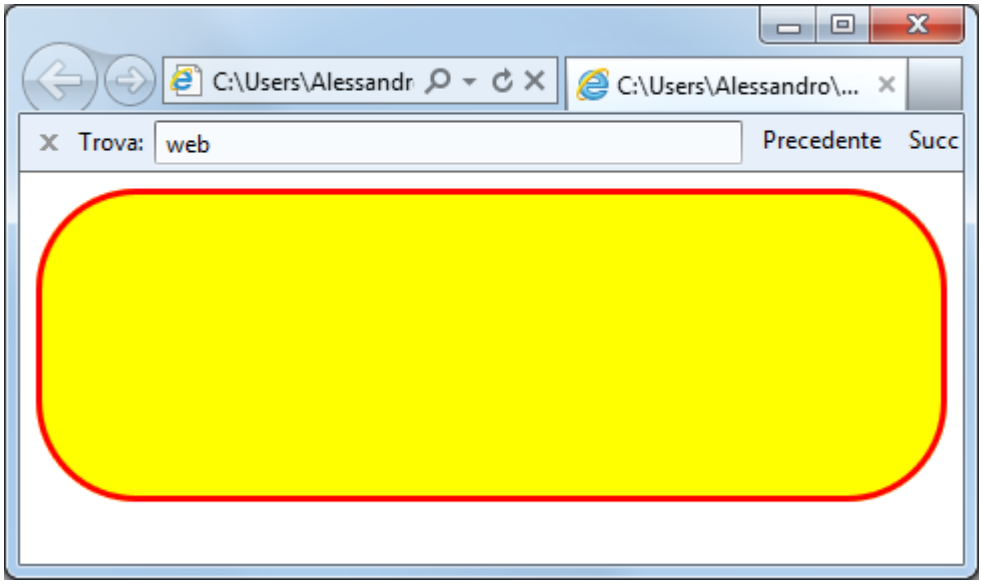
Creiamo un semplice :

```
<div id="bordi"></div>
```

e assegniamogli il nostro CSS:

```
div {
    height: 150px;
    background-color: yellow;
    border: medium solid red;
    border-radius: 50px;
}
```

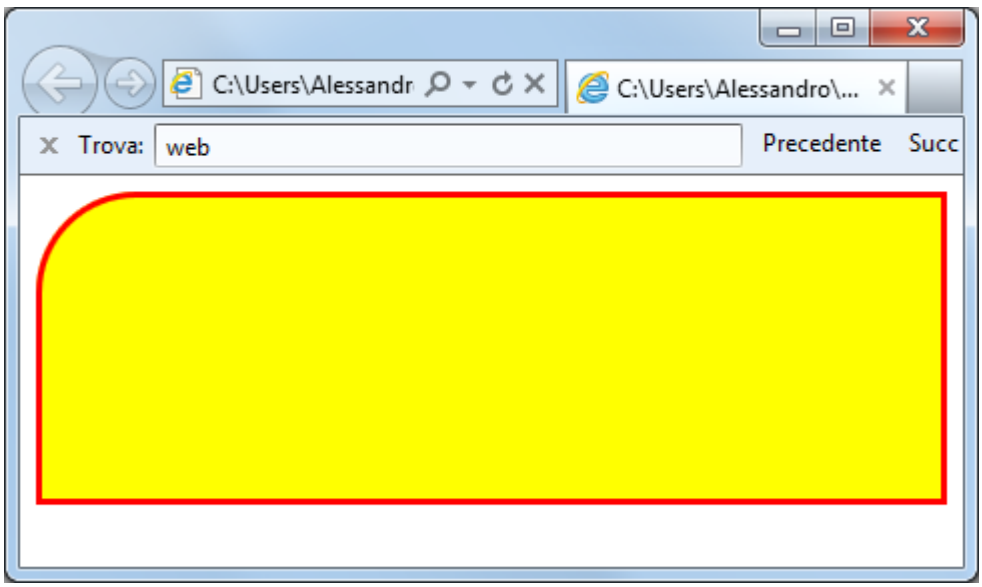
Il risultato sarà il seguente.



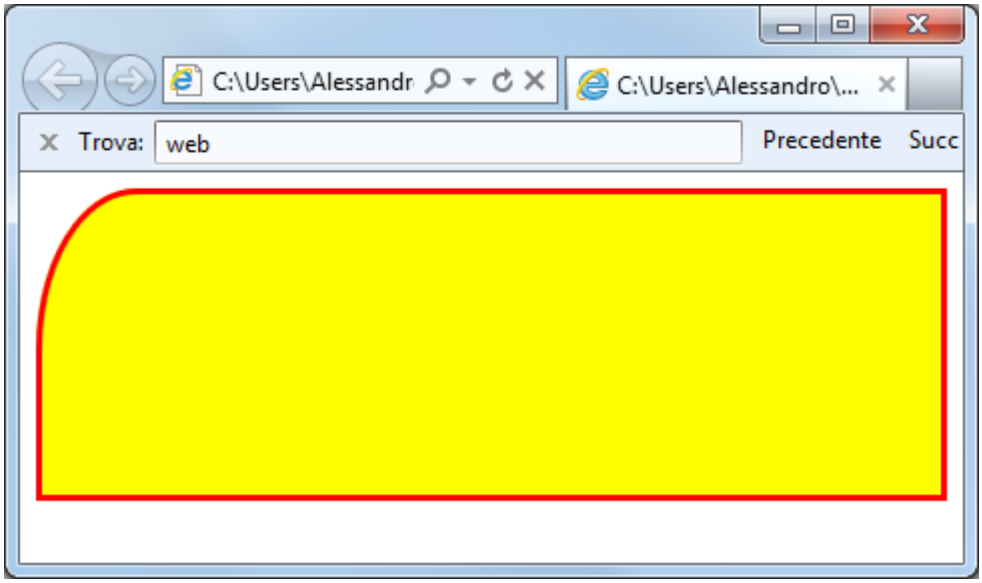
Abbiamo quindi assegnato a tutti e 4 gli ellissi virtuali lo stesso raggio orizzontale e verticale (50px) ottenendo il primo obiettivo: arrotondare tutti gli angoli.
Ora però vogliamo arrotondare solo l'angolo superiore sinistro. Come fare?

```
border-radius: 50px 0 0 0;
```

Questo codice assegna il valore di 50px al raggio orizzontale e verticale del primo ellisse (quello in alto a sinistra). Assegna invece raggio nullo a tutti gli altri. Il risultato è proprio quello che ci aspettavamo.



Ora facciamo l'ultimo passo: gestire entrambi i raggi dei 4 ellissi virtuali.
Ammettiamo di voler ottenere qualcosa di simile a quanto mostrato dalla seguente figura.



Sembra chiaro che dobbiamo intervenire sul primo ellisse virtuale (quello in alto a sinistra). Tale ellisse dovrà avere il raggio verticale più grande di quello orizzontale. Per ottenere questo risultato possiamo scrivere il seguente codice CSS:

```
border-radius: 50px 0 0 0 / 80px 0 0 0;
```

oppure

```
border-radius: 50px 0 0 0 / 80px;
```

in quanto un raggio orizzontale nullo rende inutile qualsiasi assegnazione al raggio verticale.

In questo caso, possiamo osservare il codice in azione:

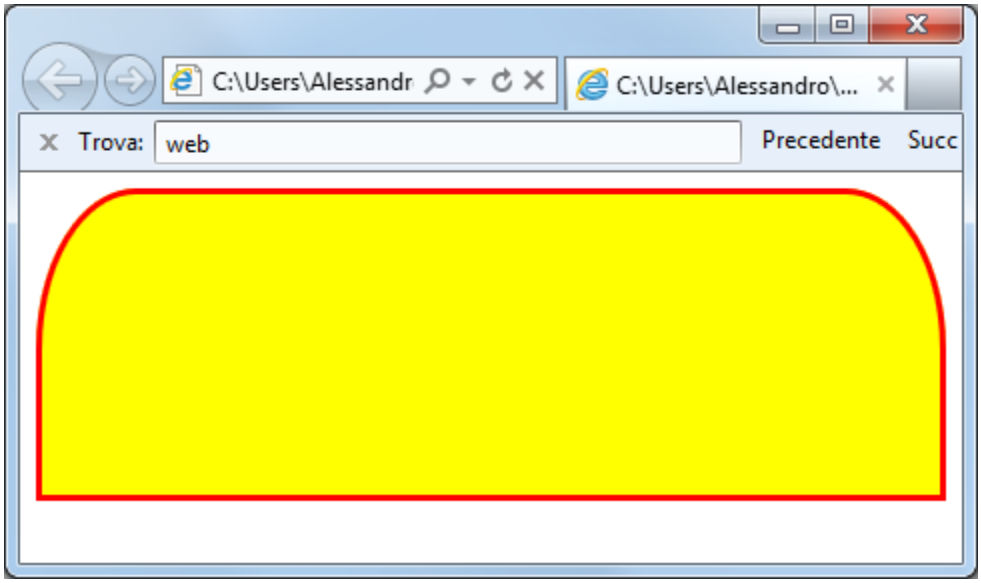
http://www.alessandrostella.it/lato_client/css3/css3_23.html

Abbiamo appena imparato che per gestire entrambi i raggi di un ellisse virtuale bisogna usare **il simbolo dello slash (/)**. Quindi, nella sintassi della proprietà border-radius, i primi 4 numeri (separati da uno spazio) indicano il raggio orizzontale dei 4 ellissi (in mancanza dello slash il sistema utilizza tali valori anche per i raggi verticali). Se però vogliamo indicare esplicitamente i valori dei raggi verticali, allora ci basta usare lo slash (/).

Possiamo ora sbizzarrirci come più ci aggrada. Ad esempio, il seguente codice:

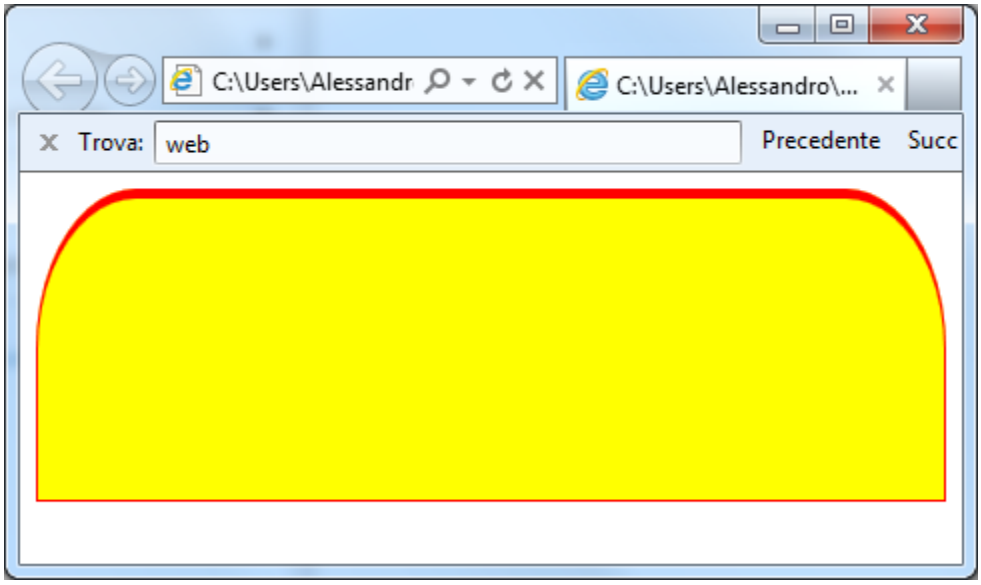
```
border-radius: 50px 50px 0 0 / 80px 80px;
```

produce il seguente risultato.



Insomma border-radius è proprio una gran bella proprietà!

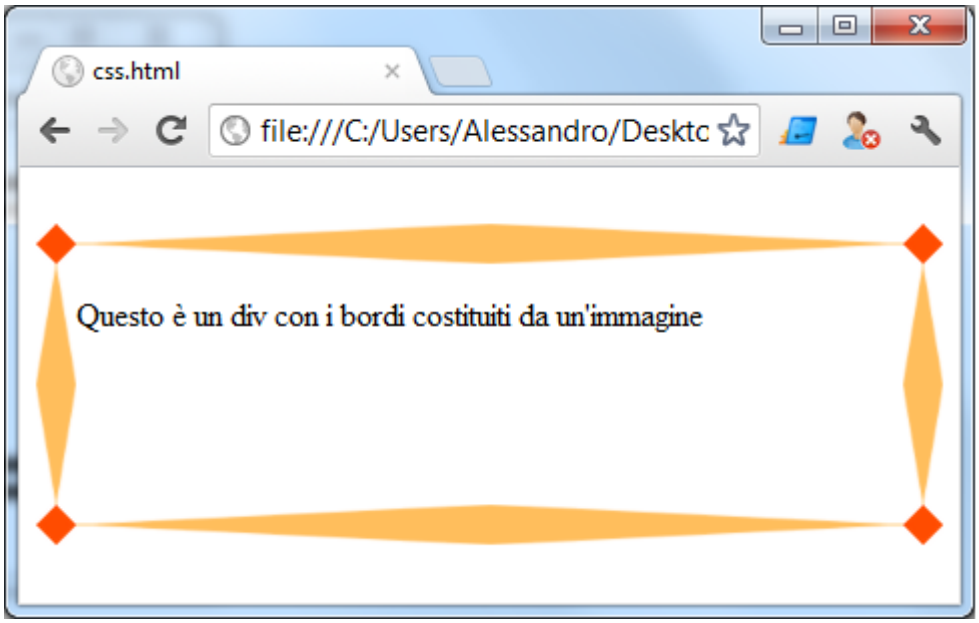
Prima di proseguire proviamo a porci una domanda: conoscendo le proprietà border-style, border-width e border-radius, quale codice CSS dovremmo scrivere per ottenere il risultato mostrato nella seguente figura?



Concludiamo questo paragrafo osservando che nell'indicare i valori da assegnare ai raggi possiamo usare diverse unità di misura: px, pt, em, %.

Usare immagini nei bordi del box

Con le proprietà viste fino a questo momento, risulterebbe impossibile ottenere un risultato come quello mostrato nella seguente figura.



Per sopperire a questa mancanza, il w3c ha introdotto la proprietà **border-image**. Attraverso border-image è possibile assegnare delle immagini ai quattro bordi di un elemento (ricordiamo che i bordi sono inizialmente nascosti. Quindi, per poter usare la proprietà border-image è necessario innanzitutto assegnare stile e dimensioni ai bordi, usando le proprietà border-style, border-width o border).
La proprietà border-image in realtà è un modo veloce per settare diverse distinte proprietà previste dal w3c per la gestione delle immagini sui bordi di un box. In particolare border-image racchiude, in ordine, le seguenti proprietà:

- border-image-source
- border-image-slice
- border-image-width
- border-image-outset
- border-image-repeat

Per poter usare la proprietà border-image è necessario comprendere come il w3c pretende che venga creata e poi usata l'immagine che abbiamo intenzione di usare come sfondo nei bordi. Osservando la figura iniziale ci verrebbe da pensare che ci servono 3 immagini:

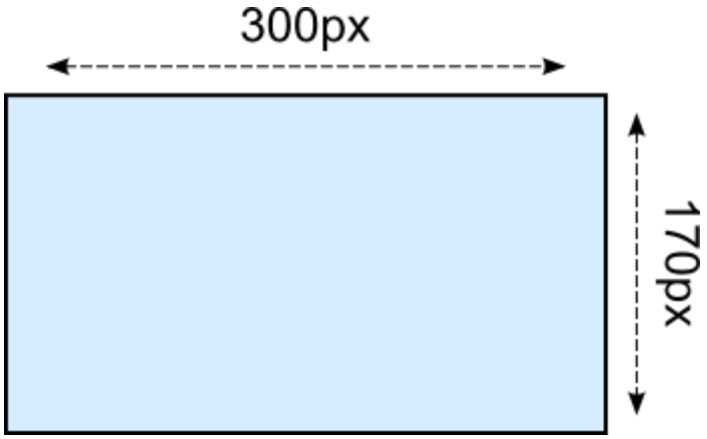
- un rombo rosso
- un rombo giallo basso e largo
- un rombo giallo alto e stretto

Ma non è così.

Il w3c ragiona a "slice". Abbiamo bisogno di **un'unica immagine** che contenga tutte le immagini che ci servono. Sarà poi il browser, opportunamente istruito dal codice CSS, ad occuparsi del ritaglio dell'immagine e di inserire i vari ritagli nei bordi e negli angoli del box.

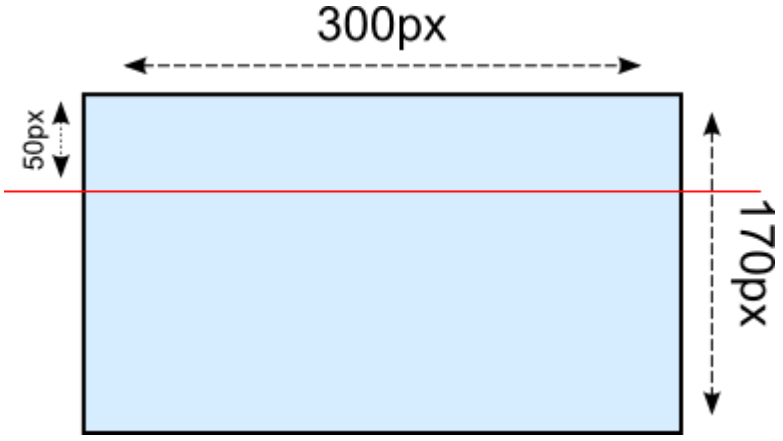
Ma come fa il browser a ritagliare l'immagine?

Ammettiamo di avere un'immagine di dimensione 300x170px, come quella qui di seguito mostrata (in questo caso l'immagine è volutamente monocolore. Una volta compreso come bisogna trattare l'immagine, procederemo con l'uso di un'immagine reale).

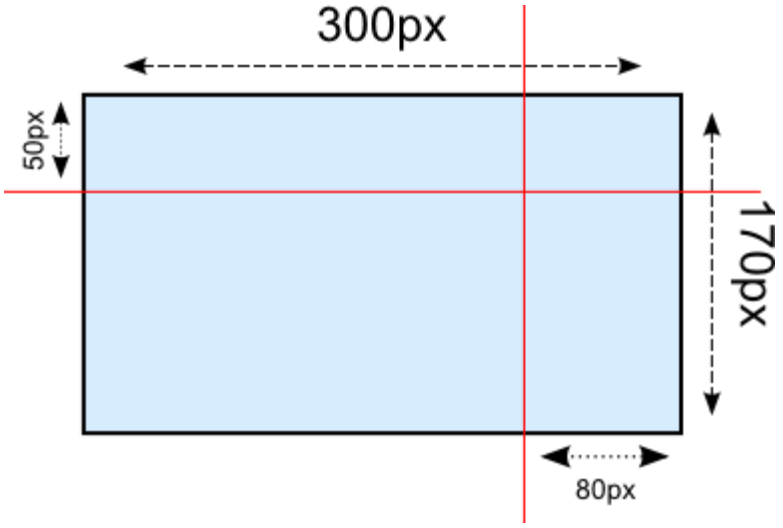


Il browser effettua il ritaglio dell'immagine tracciando 4 linee immaginarie: 2 orizzontali e 2 verticali.

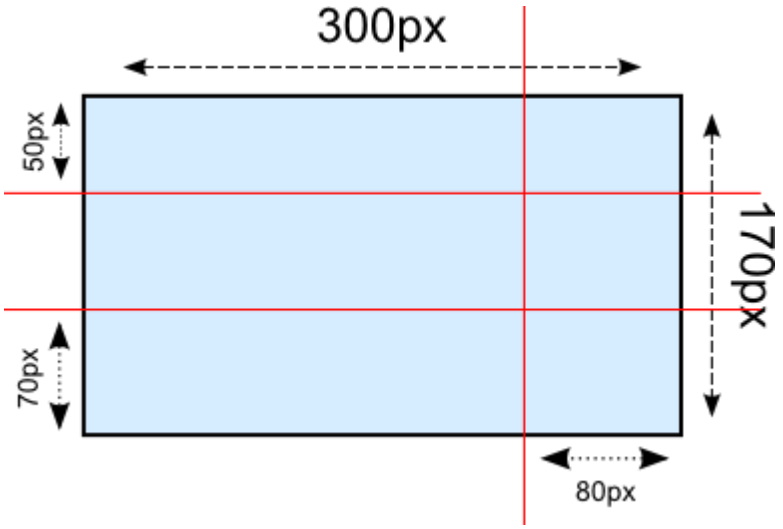
La prima linea è orizzontale e viene tracciata in base alla distanza dal margine superiore dell'immagine, ad esempio 50px (questo valore dovremo comunicarlo nel codice CSS). In questo caso il browser traccerebbe la seguente (immaginaria) linea.



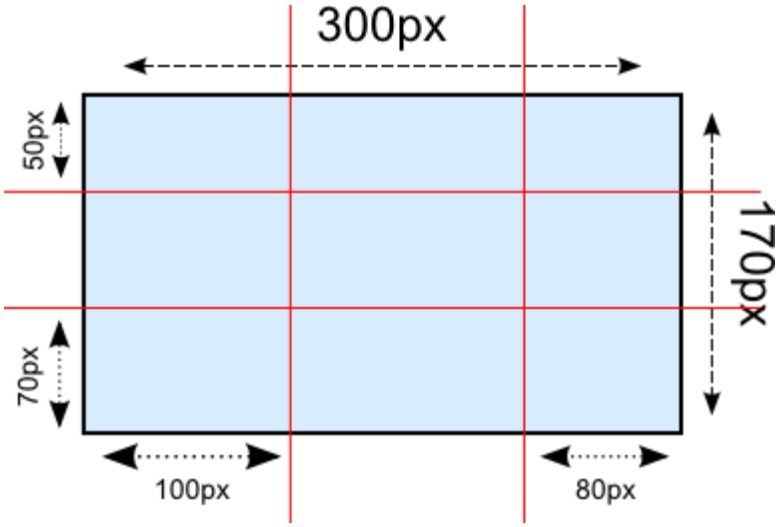
La seconda linea invece è verticale e viene tracciata in base alla distanza dal margine destro dell'immagine, ad esempio 80px (anche questo valore dovremo comunicarlo nel codice CSS).



La terza linea è orizzontale e viene tracciata in base alla distanza dal margine inferiore dell'immagine, ad esempio 70px (come sopra, dovremo indicare questo valore nel codice).



Infine **l'ultima linea, la quarta, è verticale** e viene tracciata in base alla distanza dal margine sinistro dell'immagine, ad esempio 100px (anche in questo caso dovrà essere indicato nel codice).



Così facendo **il browser è in grado di dividere la nostra immagine in 9 celle**.
Adesso dovrebbe essere intuitivo comprendere quale pezzo dell'immagine verrà usato per l'angolo superiore sinistro del nostro box, quale sul bordo superiore e così via (il giro è sempre quello: da sinistra a destra, dall'alto in basso). In particolare, la parte dell'immagine contenuta nella cella in alto a sinistra verrà usata come immagine per l'angolo superiore sinistro del box. La parte dell'immagine contenuta nella cella centrale in alto verrà usata come immagine per il bordo superiore e così via. La parte dell'immagine contenuta nella cella centrale non verrà usata.
Ora, come facciamo a comunicare tutto ciò al browser? Banalmente così:

```
border-image:url("immagine.png") 50 80 70 100 stretch stretch;
```

Il significato dei numeri 50 80 70 e 100 dovrebbe ora esserci chiaro!
Per comunicare al browser come disegnare le linee virtuali possiamo usare, oltre ai pixel, anche una percentuale. Ad esempio:

```
border-image:url("immagine.png") 30% 40% 25% 33% stretch
stretch;
```

Bene.
Ora non resta che comprendere l'ultima parte del codice: stretch stretch.
Questi due valori servono per comunicare al browser come deve trattare le immagini sui due bordi orizzontali e sui due verticali. In particolare il primo valore vale per i 2 bordi orizzontali (superiore e inferiore), mentre il secondo valore vale per quelli verticali

(sinistro e destro). Se il secondo valore viene omesso, si assume che sia uguale al primo. I valori hanno i seguenti significati:

- **stretch**: il browser deve allungare l'immagine fino a riempire tutto lo spazio.
- **repeat**: il browser deve ripetere l'immagine tutte le volte necessarie a riempire lo spazio.
- **round**: il browser deve ripetere l'immagine tutte le volte necessarie a riempire lo spazio. Se però non si riesce a riempire lo spazio ripetendo l'immagine un numero intero di volte, allora l'immagine viene riscalata in modo che lo faccia.
- **space**: il browser deve ripetere l'immagine tutte le volte necessarie a riempire lo spazio. Se però non si riesce a riempire lo spazio ripetendo l'immagine un numero intero di volte, lo spazio che resta viene distribuito tra le varie ripetizioni dell'immagine.

Bene. Sapendo tutto questo, possiamo usare la seguente immagine

per ottenere quanto visto nella figura a inizio paragrafo. Ci basta creare un e associargli il seguente codice CSS:

```
div {
    height:120px;
    border-style:solid;
    border-width:20px;
    border-image:url("http://www.w3.org/TR/css3-
background/border.png") 27 27 27 27 stretch stretch;
 }
```

Ovviamente ripetere 4 volte 27 e 2 volte stretch non è una grande idea poiché scrivere:

```
border-image:url("http://www.w3.org/TR/css3-
background/border.png") 27 27 27 27 stretch stretch;
```

oppure

```
border-image:url("http://www.w3.org/TR/css3-
background/border.png") 27 stretch;
```

è la stessa cosa.

E’ possibile vedere il codice in azione qui:
http://www.alessandrostella.it/lato_client/css3/css3_24.html

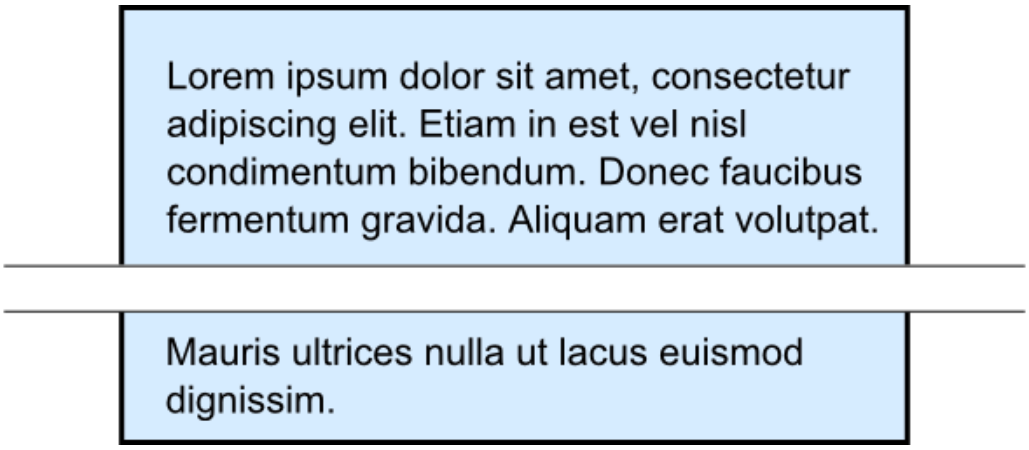
Se la proprietà non funziona è molto probabile che il browser utilizzato non ne prevede ancora il supporto.

Decorazioni e ombre

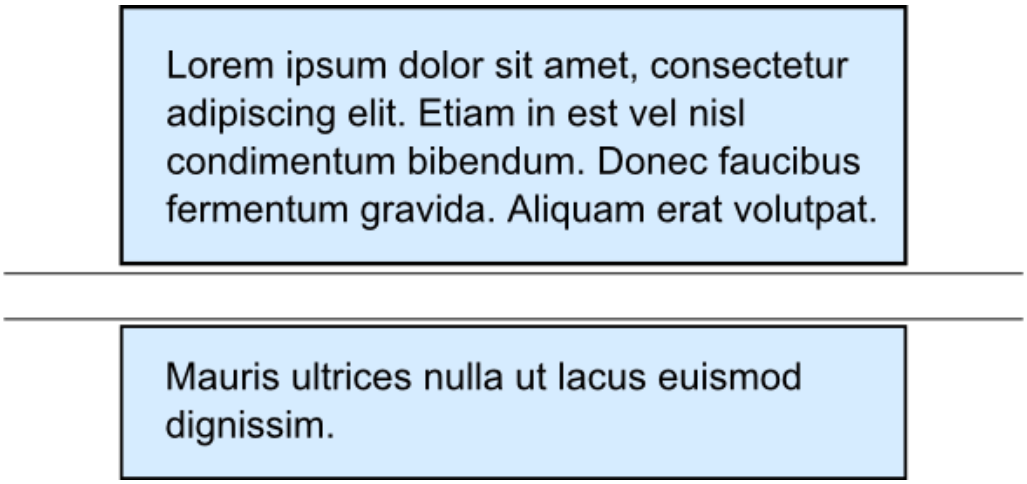
Sebbene faccia parte delle specifiche ufficiali, la proprietà **box-decoration-break** al momento non è supportata da nessun browser.
Quando un box viene diviso a causa di un'interruzione di pagina un'interruzione di colonna, o, per gli elementi inline, un'interruzione di riga, la proprietà box-decoration-break specifica se i vari pezzi in cui viene diviso il box devono essere trattati come pezzi di un unico box, oppure se ogni pezzo deve essere trattato come se fosse un altro box ma con lo stesso codice css associato. I valori che può assumere questa proprietà sono solo 2:

- slice
- clone

Nel primo caso (slice) si ottiene un risultato del genere. La parte bianca al centro della figura rappresenta un’ipotetico cambio di pagina.



Nel secondo invece (clone) si ottiene questo risultato.

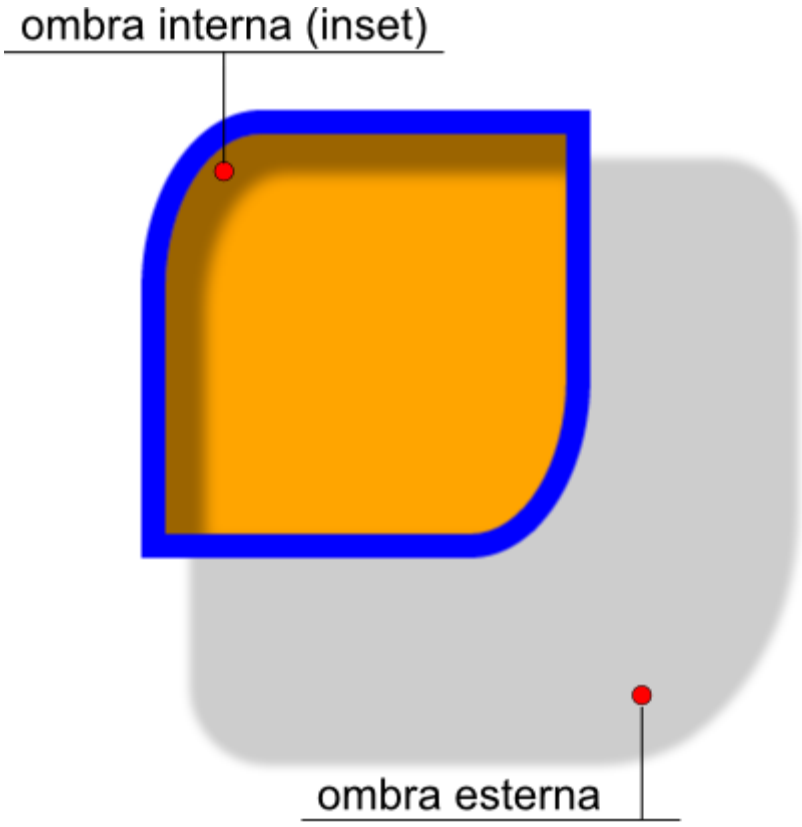


La proprietà **box-shadow** è un'altra di quelle richieste a gran voce dalla comunità degli sviluppatori. Il suo scopo è quello di gestire le ombre di un box. Ammettiamo di avere un con, associato, il seguente codice css:

```
<style type="text/css">
    div {
          border-style:solid;
          border-color:blue;
          border-width:25px;
          height:250px;
          width:250px;
          background-color:orange;
          border-radius:100px 0;
          box-shadow: 30px 30px 5px 5px rgba(0,0,0,0.2), 5px 5px
5px 5px rgba(0,0,0,0.2) inset;
    }
</style>
```

Il risultato dovrebbe essere ormai conosciuto e possiamo osservalo qui:
http://www.alessandrostella.it/lato_client/css3/css3_25.html

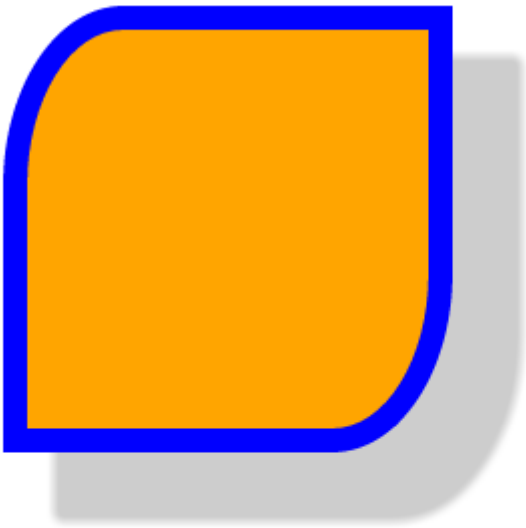
Quello che impareremo in questo paragrafo è trasformare il di prima in quello mostrato dalla seguente figura.



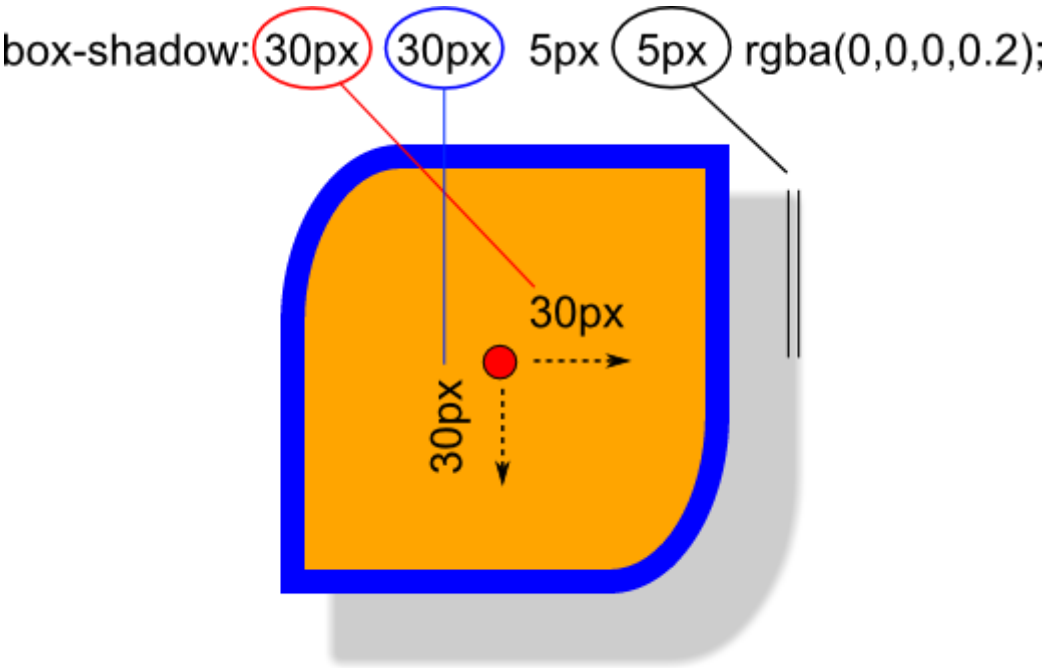
Da questa semplice figura risulta subito chiara l'enorme utilità di questa proprietà. Cerchiamo allora di comprendere come usarla, partendo dal capire come realizzare un'ombra esterna. Ad esempio, con il seguente codice:

```
box-shadow: 30px 30px 5px 5px rgba(0,0,0,0.2);
```

otterremo il seguente risultato.



Osserviamo la figura che segue per comprendere il perché di tale risultato.

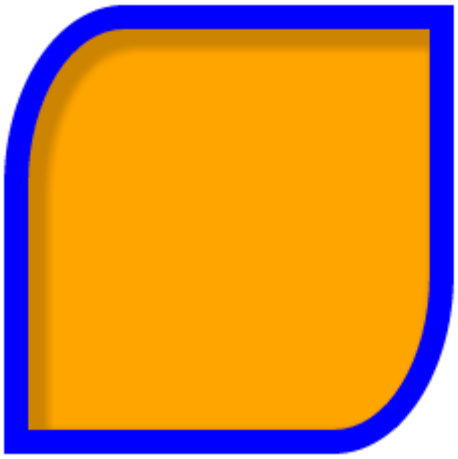


I primi due valori indicano lo spostamento dell'ombra rispetto al centro del nostro box (indicato in figura con un pallino rosso). Il primo valore indica lo spostamento orizzontale verso destra, il secondo valore indica lo spostamento verticale verso il basso. In entrambi i casi possono essere usati valori negativi, causando gli spostamenti inversi.
Il terzo valore (l'unico non cerchiato) indica il raggio di sfocatura. I valori negativi non sono ammessi. Se questo valore è zero, il bordo dell'ombra è netto, senza alcuna sfocatura. In caso contrario, maggiore è il valore, più il bordo dell'ombra sarà sfocato.

Il quarto valore consente di espandere (per valori positivi) o contrarre (per valori negativi) l'ombra originale.
Infine **la funzione rgba()** si occupa di decidere colore e trasparenza dell'ombra.

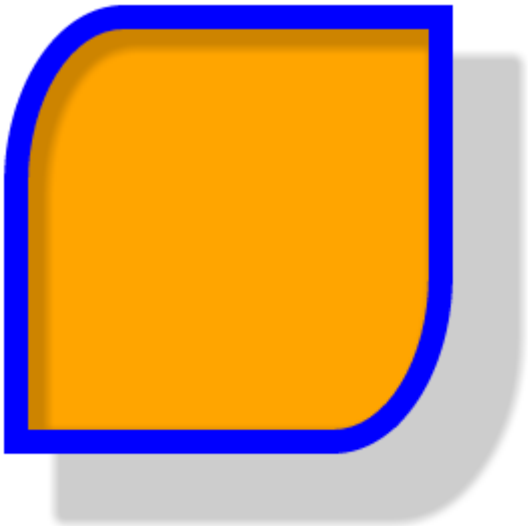
Per disegnare un'ombra interna basta aggiungere **la parola chiave inset** alla fine dell'istruzione:

```
box-shadow: 5px 5px 5px 5px rgba(0,0,0,0.2) inset;
```



Se infine volessimo **disegnare contemporaneamente entrambe le ombre**, basta dividere i due codici con una virgola:

```
box-shadow: 30px 30px 5px 5px rgba(0,0,0,0.2), 5px 5px 5px 5px
rgba(0,0,0,0.2) inset;
```



Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/css3/css3_26.html

CSS Image Values and Replaced Content

Nonostante il buono stato di avanzamento di questo modulo, i browser non hanno ancora iniziato l'opera di integrazione delle specifiche in esso contenute. Tuttavia poiché è prevedibile che ciò avvenga a breve, spenderemo un po' del nostro tempo in un veloce approfondimento.
Come dice il titolo del modulo, esso si occupa della gestione delle immagini e degli elementi di tipo replaced (ricordiamo? Gli elementi HTML di tipo replaced li abbiamo incontrati durante lo studio di HTML 4).
Le principali estensioni rispetto al livello 2 sono la generalizzazione della funzione url() nella nuova funzione image() con una serie di aggiunte, un algoritmo per consentire il dimensionamento per le immagini e altri contenuti, e diverse proprietà per controllare l'interazione degli elementi sostituiti.

La funzione image()

Questa funzione sostituirà la funzione url(). Lo scopo della funzione image() è quello di caricare un'immagine o un colore. Esempio:

```
background-image: image("wavy.svg");
```

Questo codice cercherà di caricare l'immagine "wavy.svg" come sfondo del box a cui viene applicato il codice CSS.
La stessa funzione ci consente anche di caricare un colore:

```
background-image:  image(rgba(0,0,255,0.5));
```

I motivi principali per cui il w3c sostituirà la funzione url() sono 2:

- il nome "url" non è particolarmente chiaro
- la funzione url() non gestisce un eventuale errore di caricamento dell'immagine

La funzione image() ha un nome che dice tutto da solo. Inoltre è in grado di gestire un eventuale errore in fase di caricamento dell'immagine. Esempio:

```
background-image: image("dark.png", black);
```

Se l'immagine "dark.png" non viene trovata o si verifica un errore, image() si occuperà di colorare lo sfondo di nero.

Inoltre image() può caricare una lista di immagini. Nel caso in cui la prima non venga trovata o si verifichi un errore in fase di caricamento, viene caricata la seconda e così via. Esempio:

```
background-image: image("wavy.svg", 'wavy.png' , "wavy.gif");
```

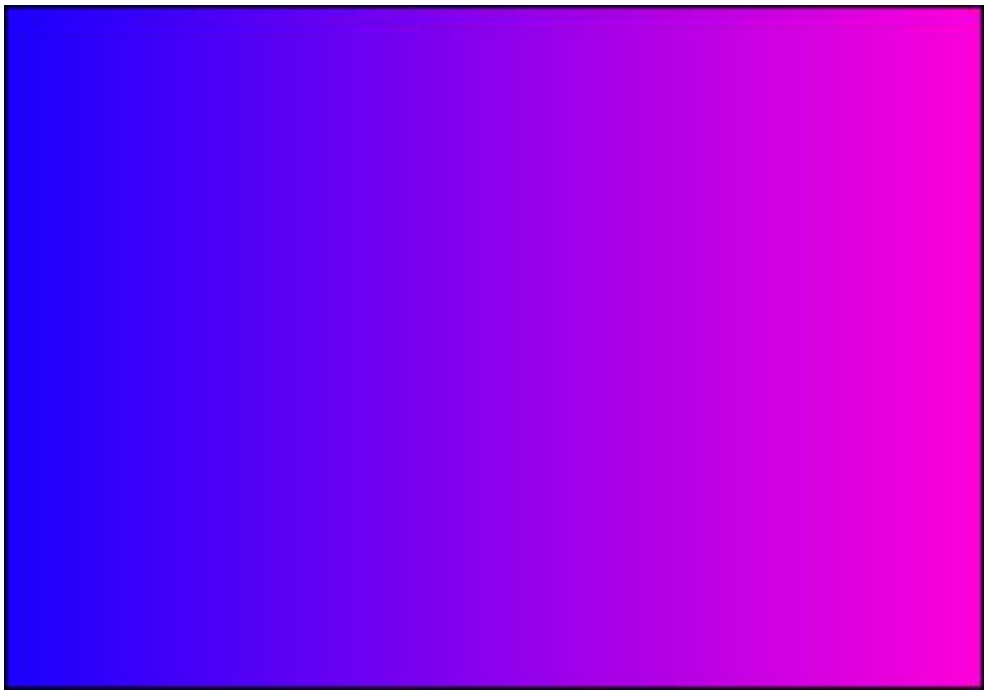
Infine, tramite la forma "xywh", la funzione image() è in grado di caricare un frammento di un'immagine:
background-image: image('sprites.png#xywh=20,0,60,20');

Con tale istruzione il browser mostrerà il pezzo dell'immagine sprites.png così individuato:

- 20px verso destra a partire dall'angolo superiore sinistro
- 0px verso il basso a partire dal bordo superiore
- una volta puntato il "taglierino" in queste coordinate taglierà un pezzo di immagine largo 60px e alto 20px

I gradienti

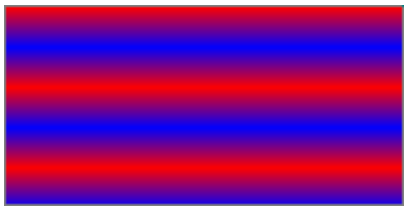
Per gradiente si intende una sfumatura da un colore ad un altro.

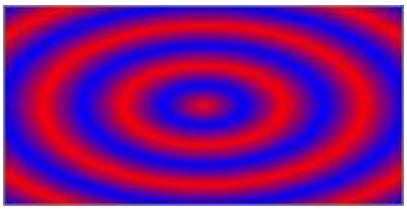


In questo caso sono state introdotte le seguenti funzioni:

- linear-gradient()
- radial-gradient()
- repeating-linear-gradient()
- repeating-radial-gradient()

Il loro utilizzo è ancora sconosciuto a tutti i browser, ma la loro funzione dovrebbe risultare chiara già dal loro nome.
La prima funzione crea un gradiente lineare (come quello mostrato nella figura iniziale). La seconda invece crea un gradiente radiale, quindi partendo dal centro dell'oggetto. Le ultime due invece consentono di ripetere più volte lo stesso gradiente.





Dimensioni e orientamento delle immagini

Ci sono altre novità che riguardano la gestione della dimensione delle immagini (e degli elementi replaced in generale) e il loro orientamento. Sarà infatti possibile gestire da CSS sia le dimensioni di un'immagine o di un video, sia il loro orientamento. Anche in questo caso i nomi scelti per le proprietà che si occuperanno di gestire queste nuove caratteristiche sono molto intuitivi:

- object-fit
- object-position
- image-resolution
- image-orientation

Purtroppo i browser ancora non gestiscono queste nuove caratteristiche.

8
HTML5 e CSS3 all'opera

In questo capitolo cercheremo di riprodurre, usando HTML5 e CSS3, il layout a 2 colonne fisse centrate, prodotto durante lo studio di HTML4 e CSS2.
Prima di scrivere il codice rivediamo quanto creato con HTML4 e CSS2:
http://www.alessandrostella.it/lato_client/html4css2/html4css2_14.html

L'osservazione del vecchio layout ci aiuta nell'identificazione delle probabili sezioni in cui dovremo dividere il nostro nuovo layout. Quello che ne scaturisce è che ci serviranno almeno le seguenti sezioni:

- un header (con menù)
- una sezione sinistra (con menù)
- una sezione centrale/destra (con i contenuti)
- un footer

Abbiamo visto che HTML5 ci fornisce molti nuovi elementi che fanno proprio al caso nostro.
Gli elementi che sembrano subito utili alla nostra causa sono <header>, <nav>, <aside>, <section>, <article>, <footer>.
Inoltre potrebbe essere utile usare un elemento per racchiudere tutta la pagina e allinearla al centro.
Una prima bozza di documento HTML5 potrebbe quindi essere la seguente.

```
<!DOCTYPE html>
<html lang="it">
    <head>
```

```
		<title>HTML 5 e CSS 3</title>
		<meta charset="utf-8">
	</head>
	<body>
		<div>
			<header>
				<nav>
					<ul>
						<li><a href="#">home</a>
						</li>
						<li><a href="#">chi siamo</a>
						</li>
						<li><a href="#">servizi</a>
						</li>
						<li><a href="#">contatti</a>
						</li>
					</ul>
				</nav>
			</header>
			<aside>
				<nav>
					<ul>
						<li><a href="#">home</a></li>
						<li><a href="#">chi siamo</a>
						</li>
						<li><a href="#">servizi</a>
						</li>
						<li><a href="#">contatti</a>
						</li>
					</ul>
				</nav>
			</aside>
			<section>
				<article>
					<header><h1>Titolo</h1><hr></header>
					<p>Lorem ipsum [...]</p>
					<footer>Questo è il footer
						dell'articolo</footer>
				</article>
```

```
                </section>
                <footer></footer>
            </div>
    </body>
</html>
```

Che, come primo risultato,produce questa pagina web.

Risultato che possiamo osservare in azione qui:
http://www.alessandrostella.it/lato_client/html5css3/html5css3_00.html

Da HTML4 a HTML5

Ovviamente senza l'utilizzo di alcun foglio di stile, il risultato è abbastanza lontano da quello che dobbiamo ottenere. Tuttavia dovremmo avere già tutte le sezioni del documento che ci servono. Facciamo un passo in avanti: iniziamo a centrare la pagina. Per fare questo assegniamo un id all'unico presente nel documento e usiamo un foglio di stile per comunicare a questo di posizionarsi al centro della pagina.
Per pura analogia con il vecchio layout chiamiamo #centrato il nostro e scriviamo il seguente codice CSS:

```
html body {
    margin:0px;
    padding:0px;
    background-color:black;
}
#centrato {
    width:1000px;
    height:auto;
    background-color:#333;
    margin-left:auto;
    margin-right:auto;
}
```

Allargando la pagina del browser oltre i 1000px possiamo effettivamente osservare come il contenuto di #centrato venga centrato nella pagina.
http://www.alessandrostella.it/lato_client/html5css3/html5css3_01.html

Ora **occupiamoci della testata della pagina e del menù orizzontale** in essa contenuto.
La prima cosa che possiamo osservare è che abbiamo 2 <header>, uno di pagina e uno di articolo. Dobbiamo quindi differenziarli. Per farlo possiamo usare il comodo attributo id. Lo stesso discorso deve essere fatto per l'elemento <nav> perché abbiamo il menù di navigazione orizzontale e quello verticale. Anche con i due elementi <nav> useremo l'attributo id per differenziarli.
Circa il menù orizzontale, dobbiamo posizionarlo in basso rispetto all'header di pagina. Memori del fatto che ogni elemento viene posizionato in modo statico, dobbiamo assegnare la posizione relativa all'header di pagina e poi assegnare posizione assoluta e bottom:0 all'elemento <nav>.
Quindi il codice HTML per l'header di pagina e il menù orizzontale diventa:

```
<header id="hdPagina">
    <nav id="menuOr">
          <ul>
                <li><a href="#">home</a></li>
                <li><a href="#">chi siamo</a></li>
                <li><a href="#">servizi</a></li>
                <li><a href="#">contatti</a></li>
          </ul>
```

```
    </nav>
</header>
```

Abbiamo usato l'attributo id per identificare univocamente sia l'header sia il menù.

In base a quanto detto prima, il codice CSS da aggiungere al precedente è il seguente:

```
#hdPagina {
    position:relative;
    background-color:#222;
    height:150px;
}
#menuOr {
    position:absolute;
    bottom:0px;
    width:100%;
    text-align:center;
    background-color:#666;
}
#menuOr ul {
    margin:0px;
    padding:0px;
    overflow:hidden;
    list-style-type:none;
}
#menuOr ul a:link , ul a:visited{
    display:block;
    padding:10px;
    text-decoration:none;
    color:white;
    text-transform:uppercase;
}
#menuOr ul a:hover {
    background-color:grey;
}
#menuOr ul li{
    float:left;
    width:120px;
}
```

Possiamo vedere il risultato qui:
http://www.alessandrostella.it/lato_client/html5css3/html5css3_02.html

Adesso **organizziamo la parte sinistra e centrale del documento**.
Questa è la parte più complicata. L'elemento <aside> rappresenta la vecchia colonna di sinistra, mentre l'elemento <section> rappresenta la vecchia seconda colonna. Questi due elementi devono posizionarsi uno alla destra dell'altro e quindi l'elemento <aside> dovrà essere dotato della proprietà float settata su left. Per identificare univocamente le due zone (sinistra e centro/destra) assegniamo id="primaColonna" alla parte di sinistra (<aside>), mentre id="secondaColonna" a quella di centro/destra (<section>).
E' doveroso un approfondimento sull'elemento <section>. In particolare è necessario definire la differenza esistente tra un elemento <section> e l'elemento .
Sull'argomento il w3c afferma: "The section element represents a generic section of a document or application. A section, in this context, is a thematic grouping of content, typically with a heading", ossia è necessario usare l'elemento <section> quando il suo contenuto è costituito da un insieme di elementi legati da un tema comune. Ad esempio, se in una certa sezione del documento verranno mostrati una sequenza di articoli, allora quella sezione è definibile con l'elemento <section>. Se invece una zona del documento deve essere centrata, allora si può pensare di racchiudere quella zona in un elemento .
Noi quindi, in previsione del fatto che nella zona centrale del nostro documento avremo una sequenza di articoli, abbiamo racchiuso questa zona in un elemento <section>.
Il nuovo codice della parte del documento a cui ci stiamo riferendo diventa quindi il seguente.

```
<aside id="primaColonna">
    <nav id="menuVerticale">
          <ul>
                <li><a href="#">home</a></li>
                <li><a href="#">chi siamo</a></li>
                <li><a href="#">servizi</a></li>
                <li><a href="#">contatti</a></li>
          </ul>
    </nav>
</aside>
<section id="secondaColonna">
    <article>
```

```
            <header id="hdArticolo"><h1>Titolo</h1><hr></header>
            <p>Lorem ipsum [...]</p>
            <footer id="ftArticolo">Questo è il footer
                  dell'articolo</footer>
    </article>
</section>
```

Da notare gli id assegnati all'header e al footer dell'articolo, al fine di differenziarli da quelli dalla pagina.

Il codice CSS associato, da aggiungere a quanto già scritto fino a questo momento, è il seguente.

```
#primaColonna {
    float:left;
    width:20%;
}
#menuVerticale ul {
    margin:0px;
    padding:0px;
    list-style-type:none;
}
#menuVerticale ul a:link, ul a:visited{
    display:block;
    padding:10px;
    text-decoration:none;
    color:white;
    text-transform:uppercase;
}
#menuVerticale ul a:hover {
    background-color:grey;
}
#secondaColonna {
    float:left;
    background-color:#444;
    width:80%;
    height:auto;
}
```

Il risultato è visibile qui:
http://www.alessandrostella.it/lato_client/html5css3/html5css3_03.html

Infine non ci resta che **gestire il footer della pagina**.
Il footer della pagina dovrà essere dotato della proprietà clear e di un id. Il codice HTML quindi diventa:

```
<footer id="ftPagina"></footer>
```

Il codice CSS da aggiungere a quanto già scritto è il seguente:

```
#ftPagina {
    clear:left;
    background-color:#666;
    height:50px;
}
```

Alla fine il codice HTML completo sarà il seguente:

```
<!DOCTYPE html>
<html lang="it">
    <head>
          <title>HTML 5 e CSS 3</title>
          <link rel="stylesheet" href="style/style_00.css">
          <meta charset="utf-8">
    </head>
    <body>
          <div id="centrato">
                <header id="hdPagina">
                      <nav id="menuOr">
                            <ul>
                                  <li><a href="#">home</a></li>
                                  <li><a href="#">chi siamo</a>
                                  </li>
                                  <li><a href="#">servizi</a>
                                  </li>
                                  <li><a href="#">contatti</a>
                                  </li>
                            </ul>
```

```
                        </nav>
                </header>
                <aside id="primaColonna">
                        <nav id="menuVerticale">
                                <ul>
                                        <li><a href="#">home</a></li>
                                        <li><a href="#">chi siamo</a>
                                        </li>
                                        <li><a href="#">servizi</a>
                                        </li>
                                        <li><a href="#">contatti</a>
                                        </li>
                                </ul>
                        </nav>
                </aside>
                <section id="secondaColonna">
                        <article>
                                <header id="hdArticolo">
                                        <h1>Titolo</h1><hr>
                                </header>
                                <p>Lorem ipsum [...]</p>
                                <footer id="ftArticolo">Questo è il
                                        footer dell'articolo</footer>
                        </article>
                </section>
                <footer id="ftPagina"></footer>
        </div>
    </body>
</html>
```

Il codice CSS completo invece sarà il seguente:

```
html body {
    margin:0px;
    padding:0px;
    background-color:black;
}
#centrato {
    width:1000px;
```

```
    height:auto;
    background-color:#333;
    margin-left:auto;
    margin-right:auto;
}
#hdPagina {
    position:relative;
    background-color:#222;
    height:150px;
}
#menuOr {
    position:absolute;
    bottom:0px;
    width:100%;
    text-align:center;
    background-color:#666;
}
#menuOr ul {
    margin:0px;
    padding:0px;
    overflow:hidden;
    list-style-type:none;
}
#menuOr ul a:link , ul a:visited{
    display:block;
    padding:10px;
    text-decoration:none;
    color:white;
    text-transform:uppercase;
}
#menuOr ul a:hover {
    background-color:grey;
}
#menuOr ul li{
    float:left;
    width:120px;
}
#primaColonna {
    float:left;
```

```
    width:20%;
}
#menuVerticale ul {
    margin:0px;
    padding:0px;
    list-style-type:none;
}
#menuVerticale ul a:link, ul a:visited{
    display:block;
    padding:10px;
    text-decoration:none;
    color:white;
    text-transform:uppercase;
}
#menuVerticale ul a:hover {
    background-color:grey;
}
#secondaColonna {
    float:left;
    background-color:#444;
    width:80%;
    height:auto;
}
#ftPagina {
    clear:left;
    background-color:#666;
    height:50px;
}
```

Il tutto può essere visionato qui:
http://www.alessandrostella.it/lato_client/html5css3/html5css3_04.html

Confrontando con il vecchio layout quanto appena ottenuto, possiamo osservare come il risultato estetico sia identico, ma il codice HTML sia molto diverso.
Pertanto, dopo questa prima esperienza nella scrittura di codice HTML5, dobbiamo mettere l'accento su 2 punti:

- la drastica riduzione nell'uso dell'elemento
- il riutilizzo quasi totale del vecchio codice css

Proviamo ora ad usare le nuove caratteristiche dei CSS3 per arrontondare il rettangolo grigio chiaro che evidenzia i nostri link quando gli passiamo sopra con il mouse.
Per ottenere il risultato sul menù orizzontale ci basta agire sul codice #menuOr aggiungendo il raggio di curvatura degli angoli.

```
#menuOr ul a:hover {
    border-radius:50px;
    background-color:grey;
}
```

Mentre per modificare il menù verticale ci basta agire nello stesso modo su #menuVerticale.

```
#menuVerticale ul a:hover {
    border-radius:50px;
    background-color:grey;
}
```

Ed ecco il risultato.
http://www.alessandrostella.it/lato_client/html5css3/html5css3_05.html

Facciamo un altro passo!
Aggiungiamo uno shadow interno al box grigio chiaro e uno shadow intorno ai testi dei link.
Per ottenere il risultato sul #menuOr, modifichiamo il codice CSS come segue:

```
#menuOr ul a:hover {
    text-shadow: #000 -1px -1px;
    box-shadow: 1px 1px 1px 1px rgba(0,0,0,0.5) inset;
    border-radius:50px;
    background-color:grey;
}
```

Con la prima istruzione creiamo un ombra di colore nero (#000) intorno al testo originale.
Poi spostiamo l'ombra in alto e a sinistra (-1px -1px) rispetto al testo originale.
Ed ecco qui il risultato.
http://www.alessandrostella.it/lato_client/html5css3/html5css3_06.html

Al momento purtroppo non tutti i browser riconoscono quest'ultimo codice, ma diventa chiaro che giocando con i nuovi CSS3 potremo ottenere grandi cose...

9
Flash, Silverlight e JavaFX

Adobe Flash, Microsoft Silverlight e Oracle JavaFX sono tre piattaforme tecnologiche che consentono di sviluppare applicazioni multimediali e interattive utilizzabili tramite un normale browser purché il browser abbia installato il relativo plug-in. Ciò significa che senza l'installazione del plug-in corretto, il browser non è in grado di visualizzare un'applicazione scritta in una qualsiasi delle piattaforme citate.

Non è ovviamente questa la sede in cui approfondire la conoscenza di queste piattaforme, tuttavia (al momento) sono parte integrante dello sviluppo web lato client e meritano pertanto un piccolo spazio in questo contesto. Probabilmente il futuro non sarà roseo per queste piattaforme poiché l'introduzione di HTML 5 renderà sempre meno necessario il loro utilizzo. Pensiamo ad esempio al fatto che Apple, nei suoi iPhone, già nel 2011 aveva escluso la possibilità di usare il plug-in di flash. C'è poco da aggiungere...

La tecnologia Flash è stata introdotta da **Macromedia** nel 1996. Negli anni seguenti si è assistito ad un crescere della popolarità di questo ambiente e, a seguito dell'acquisizione di Macromedia da parte di Adobe Systems, Macromedia Flash è diventato Adobe Flash nel dicembre 2005.
La popolarità di questo ambiente non ha però scoraggiato **Microsoft**, che nel corso del 2006 ha rilasciato la prima versione beta di Silverlight, con lo scopo di guadagnare una porzione significativa del mercato nel settore in crescita delle RIA (Rich Internet Application) e dei servizi interattivi in generale.

Con lo stesso obiettivo anche **Sun** ha lanciato la propria applicazione per realizzare applicazioni RIA: JavaFX. Nel corso del 2011 Sun è stata poi acquisita dalla Oracle.

Stiamo quindi parlando di 3 tecnologie diverse che però hanno lo stesso obiettivo e lo stesso tipo di utilizzo. Tutte vogliono produrre applicazioni di tipo RIA.
Rich Internet Application (RIA) è un termine introdotto da Macromedia, nel 2002, per definire quelle applicazioni web che forniscono un'esperienza utente simile a quella delle applicazioni desktop. La richiesta di maggior interattività e di multimedialità nel Web, ha portato alla nascita di interi framework a supporto degli sviluppatori.
Nelle RIA tutta la gestione dell'intefaccia e la presentazione dei dati viene eseguita sul client, mentre i dati risiedono sul server e vengono forniti attraverso XML o esposti tramite Web service. La presentazione di contenuti multimediali è un altro settore in cui queste applicazioni sono in grado di offrire notevoli vantaggi, fornendo una perfetta integrazione di video, immagini e suono, ma ciò, con l'avvento di HTML5, non sarà più una loro prerogativa.

Attualmente la **penetrazione nel mercato** è sicuramente a favore di Adobe Flash, ma Microsoft ha impegnato e sta impegnando molte energie nel suo Silverlight. Basti pensare che lo streaming delle rete RAI avviene usando proprio Silverlight. Al momento quindi Oracle JavaFX è la piattaforma meno usata.

Da un **punto di vista tecnico** le differenze tra queste tre tecnologie sono tante. Entrare nei dettagli non ha molto senso in questo contesto. In linea generale possiamo però dire che Flash utilizza un modello di costruzione delle animazioni frame-based, Silverlight utilizza la modalità time-based, JavaFX utilizza codice vero e proprio.

Adobe Flash

Adobe Flash è un software per uso prevalentemente grafico che consente di creare animazioni vettoriali principalmente per il web. Viene utilizzato inoltre per creare giochi o interi siti web e grazie all'evoluzione delle ultime versioni è divenuto un potente strumento per la creazione di Rich Internet Application e piattaforme di streaming audio/video.
Flash permette di creare animazioni complesse e multimediali. All'interno di esse infatti si possono inserire:

- forme vettoriali, che sono gli oggetti principali con cui Flash permette di lavorare.
- testo (sia statico sia dinamico) e caselle di input per il testo.

- immagini raster (Bitmap, GIF, Jpeg, PNG, TIFF e altri formati) sotto forma di oggetto bitmap.
- audio (MP3, WAV e altri), sia in streaming che per effetti sonori.
- video (AVI, QuickTime, MPEG, Windows Media Video, FLV).
- altre animazioni create con Flash (tramite ActionScript o interpolazioni).

Inoltre permette di creare animazioni interattive, grazie alla presenza di un linguaggio di scripting interno. Tramite questo linguaggio, denominato ActionScript e basato su ECMAScript, è possibile applicare comportamenti agli oggetti o ai fotogrammi dell'animazione. Inizialmente erano presenti solo poche azioni, ma allo stato attuale ActionScript è diventato uno strumento molto potente. Grazie ad esso si possono creare infatti menu, sistemi di navigazione, GUI, siti web completi e giochi anche complessi. Un'altra caratteristica importante è la possibilità di mantenere simboli riutilizzabili in una libreria. Esistono diverse tipologie di simboli, fra i quali semplici oggetti grafici, clip filmato (sotto-animazioni con una timeline propria e indipendente da quella principale), pulsanti, oggetti bitmap, ecc. Questa caratteristica è molto importante, in quanto permette di:

- semplificare il lavoro dello sviluppatore, che può modificare tutte le istanze presenti nell'animazione modificando il simbolo nella libreria;
- ridurre le dimensioni del file SWF prodotto;
- creare comportamenti interattivi complessi applicando del codice agli oggetti.

Ambienti e tool di sviluppo

Per sviluppare un'applicazione flash ci possiamo servire di diverse piattaforme.

Flash Builder e Flash Professional

Flash Professional (http://www.adobe.com/it/products/flash.html), attualmente alla versione CS6, è un tool per grafica e animazione e possiede strumenti di disegno integrati, come pennelli e matite virtuali. Sulla parte superiore presenta una sequenza temporale di animazione (frame per frame) che utilizza un sistema tipo film per la costruzione dell'animazione. Ad esempio, se nel primo frame posizioniamo un oggetto in alto a sinistra dello schermo e nel frame dieci lo posizioniamo nell'angolo in basso a destra, il sistema creerà automaticamente un'animazione per far spostare l'oggetto da una posizione all'altra.
Con Flash Professional è anche possibile scrivere programmi per computer, ma non è nato a questo scopo. Tuttavia è fornito di un editor di programmazione che, seppur molto

semplice, è molto migliore di Blocco note o TextEdit. Ma va bene per i programmi molto brevi.

Flash Builder (http://www.adobe.com/it/products/flash-builder.html) è uno strumento per la programmazione vero e proprio, ossia è quello che i programmatori definiscono un IDE (Integrated Development Environment) e infatti è basato su Eclipse. Eclipse è un editor molto complesso e completo usatissimo dai programmatori per scrivere codice in tanti linguaggi di programmazione: java, php, C, ecc. Quello che di base consente di fare Eclispe è scrivere codice. Per questo motivo non contempla strumenti di disegno o di animazione, ma presenta molti strumenti di debug e tante altre caratteristiche di cui necessita un programmatore.
Adobe non ha fatto altro che prendere Eclipse (che è open-source) e modificarlo aggiungendogli la possibilità di gestire Actionscript e MXML. ActionScript è un vero e proprio linguaggio di programmazione, mentre MXML è una sorta di linguaggio di scripting. Entrambi sono utili nella creazione di un file flash.
Sebbene Flash Builder non ha strumenti di disegno, esso consente di posizionare in un qualsiasi layout, tramite trascinamento, pulsanti, caselle di testo, caselle di controllo, ecc. Una volta costruita in questo modo un'interfaccia grafica è poi possibile modificare e personalizzare il codice (Actionscript e/o MXML) che il sistema scrive in automatico.

Concludendo, i file Flash possono contenere sia grafica animata sia codice di programmazione. Si può provare ad usare Flash Builder per produrre un po' di grafica animata così come si può provare ad usare Flash Professional per scrivere codice, ma in realtà il primo è ottimizzato per la codifica, mentre il secondo è ottimizzato per il disegno e l'animazione.

FlashDevelop

FlashDevelop (http://www.flashdevelop.org/) è un potente editor di codice open source che offre un eccellente supporto allo sviluppo in ActonScript e che si integra in maniera naturale con Flash.
Molto spesso la necessità è quella di cambiare ambiente di sviluppo, trovarne uno diverso dal canonico Adobe Flash Builder. Tra l'altro, la stragrande maggioranza degli sviluppatori usa attualmente sistemi operativi Microsoft (e Visual Studio come ambiente di sviluppo integrato); per questo motivo sono un po' viziati dagli ambienti di sviluppo più avanzati tipici di quel sistema operativo: in quella che sarebbe la scelta più adatta al contesto è opportuno cercare delle caratteristiche come autocompletamento del codice, Syntax Highlighting e così via.

FlashDevelop si qualifica dunque come il miglior risultato di una ricerca di questo genere. Le sue caratteristiche sono particolarmente adatte per sviluppare un prodotto Flash anche molto più complesso di una semplice animazione (per esempio, una RIA).
Questo strumento infatti permette una gestione decisamente migliore dei propri progetti e supporta svariati linguaggi (primi fra tutti ActionScript 2 e 3). Presenta, inoltre, una di quelle cosiddette "docking interface" in cui è possibile strutturare l'interfaccia a piacimento, per adattarla a quelle che sono le proprie abitudini di lavoro.
Un'altra feature interessante è l'integrazione completa con svariati tipi di compilatori: eseguire il debug e fare dei test è veramente semplice e veloce. Inoltre presenta una caratteristica molto simile all'Intellisense di Microsoft che facilita ancora di più il lavoro.
I punti a favore di FlashDevelop non finiscono qui: gli sviluppatori del progetto sono sempre attivi (una nuova release viene rilasciata approssimativamente una volta al mese, se non più spesso) e questo garantisce un IDE sicuramente al passo con i tempi.
FlashDevelop è assolutamente gratuito. Ciò non toglie che potete ricompensare gli sviluppatori per il loro lavoro: tramite la wiki si può trovare facilmente la pagina delle donazioni e dare il proprio piccolo contributo al progetto (vedendo il proprio nome tra i supporter). Con una donazione superiore ai 100$ addirittura si viene nominati "Sponsor".

Microsoft Silverlight

Microsoft Silverlight (http://www.silverlight.net/) è una tecnologia creata per gli sviluppatori web che permette di creare e pubblicare online contenuti interattivi e applicazioni di grafica 2D, audio, video, animazioni vettoriali e giochi.
Rispetto a Flash, Microsoft Silverlight crea contenuti facilmente indicizzabili dai motori di ricerca e supporta nativamente lo standard HD (video in alta definizione) e il DRM. Silverlight si integra perfettamente coi formati Windows Media ed è in grado di riprodurli senza chiamare il controllo Active X.
Scritto in XAML, Silverlight ha come idea di base quella di cancellare la diversità di interfaccia esistente tra tecnologie web e desktop creandone una vera e propria unica piattaforma. In solo 2 megabyte di dimensione, Silverlight riproduce sia su PC che su Macintosh file WMV con molteplici opzioni interattive per aumentarne l'esperienza visiva. Silverlight supporta pienamente il video ad alta definizione, quello a 720 punti e i dispositivi mobile. Infine è in grado di far girare perfettamente le nuove Rich Internet Applications cioè applicazioni che, pur arrivando dalla rete, utilizzano la potenza di calcolo del computer dell'utente per elaborare i dati.
Silverlight permette sia di definire interfacce e animazioni tramite il linguaggio XAML , sia di utilizzare la potenza del .NET Framework per la parte di programmazione con C# o

VB.NET. Questo permette la creazione di pagine Web molto accattivanti con il vantaggio di utilizzare ambienti già familiari come Visual Studio e la suite Expression.

È da precisare che il plugin non mette a disposizione tutto il CLR .NET Framework, ma un discreto sottoinsieme di essi, di fatto quando sviluppiamo applicazioni Silverlight 2 possiamo beneficiare della potenza della Base Class Library.
Il Silverlight CLR è piccolo plugin cross-platform liberamente scaricabile ed installabile in pochi secondi, può essere eseguito sia su Windows che su Mac, mentre per Linux esiste una versione open source ad-hoc che si chiama Moonlight sviluppata dal team di Mono.

Per l'utente finale installare Silverlight è un operazione estremamente semplice e veloce: il setup per il plug-in si scarica in pochi secondi (circa 4,6MB), inoltre se ci si imbatte in un contenuto Silverlight e non si possiede il plugin, appare automaticamente un link che invita a scaricarlo.

Silverlight è supportato anche dai dispositivi mobili con sistema operativo Windows Mobile 6 (o superiore) e Symbian (Serie 60). Infine Silverlight può essere utilizzato per lo sviluppo di gadget per la sidebar di Vista e di Win7.

Ambienti e tool di sviluppo

Il tool ideale per sviluppare applicazioni Silverlight è **Visual Studio** (http://www.microsoft.com/visualstudio/11/it-it/products/visualstudio) o, in alternativa, la versione Express ovvero Visual Web Developer. Una volta in possesso di Visual Studio dobbiamo installare il relativo plug-in Silverlight Tools.

Per lavorare anche con la grafica possiamo affiancare, a Visual Studio, **Expression Blend** (http://www.microsoft.com/expression/products/Blend_Overview.aspx), un editor grafico, sviluppato con WPF, che ci permette di realizzare grafica e animazioni e disegnare il layout e i controlli dell'applicazione. Inoltre utilizza i soliti file di progetto generati da Visual Studio permettendo a developers e designers di lavorare insieme sulla stessa soluzione sebbene ognuno con lo strumento più adatto al proprio mestiere.

Ultimo tool di casa Microsoft è **Deep Zoom Composer** (http://www.microsoft.com/en-us/download/details.aspx?id=24819) utile per la creazione di applicazioni Silverlight che sfruttano la tecnologia Deep Zoom per generare immagini multiscala.

Alcuni di questi strumenti possono essere installati passando per il Microsoft Web Platform Installer. Si tratta di un unico setup che ci permette di installare, in un colpo solo, la piattaforma di sviluppo Web di Microsoft. Il vantaggio è quello di avere rapidamente pronti ambienti di sviluppo e/o produzione.
La natura cross-platform di Silverlight impone comunque di poter sviluppare anche in contesti non-Microsoft. Per questo Soyatec, in collaborazione con Microsoft, ha dato vita a **Eclipse Tools for Microsoft Silverlight**, un plugin Open Source per il famoso IDE Eclipse.
Tra gli altri tool free e/o open source citiamo anche **Kaxaml**, un editor di file XAML leggero e potente, con molte caratteristiche come split view, IntelliSense, Snippets e lo Snap Shot del file in una immagine.

Oracle JavaFX

La piattaforma JavaFX è l'evoluzione della piattaforma client Java ed è stata progettata per creare e distribuire Rich Internet Application (RIA) che si comportano in modo coerente su più piattaforme. La piattaforma JavaFX fornisce un ricco set di API grafiche con accelerazione hardware e motori tali da consentire una eccellente rappresentazione grafica di applicazioni enterprise basate sui dati dei clienti. In altre parole ha lo stesso obiettivo dei suoi concorrenti.
Attualmente la piattaforma si trova nella versione 2.x. Di seguito possiamo trovarne alcune delle caratteristiche peculiari:

- **Integrazione completa con la JDK 7**. Attualmente JavaFX SDK è completamente integrato sia nella Java SE 7 Runtime Environment (JRE) sia nel Development Kit (JDK). Questa integrazione con la JDK 7 elimina la necessità di scaricare e installare JavaFX 2 SDK separatamente.
- **API Java per JavaFX** che forniscono tutte le caratteristiche familiari del linguaggio Java (generics, annotazioni e multithreading) che gli sviluppatori Java sono abituati ad usare. Le API sono comunque progettate per consentire l'uso di altri linguaggi come JRuby e Scala.
- **Un nuovo motore grafico** per gestire le moderne unità di elaborazione grafica (GPU). La base di questo nuovo motore è una pipeline hardware accelerata graficamente, chiamata Prism che è accoppiata con un toolkit a finestre nuovo, chiamato Glass. Questo motore grafico fornisce le basi per progressi attuali e futuri.
- **FXML**, un linguaggio basato su XML, utilizzato per definire l'interfaccia utente in un'applicazione JavaFX. Non è un linguaggio compilato e, quindi, non richiede di

ricompilare il codice ogni volta che si apporta una modifica al layout (ricorda molto da vicino MXML).

- **Un nuovo motore multimediale** che supporta la riproduzione di contenuti web multimediali. Si tratta di un framework basato sul framework multimediale GStreamer.
- **Molti controlli UI built-in** che includono grafici, tabelle, menu e riquadri.

Altre caratteristiche da mettere in evidenza sono:

- **Multi piattaforma**: JavaFX rende abbastanza trasparente il passaggio da un dispositivo all'altro. Con alcune accortezze (e limitazioni) si può realmente eseguire lo stesso codice JavaFX sia come applicazione desktop, sia come Applet all'interno di un browser, sia come applicazione per cellulare.
- **Trascina e installa**: una funzione innovativa, ereditata dalle ultime versioni di Java, permette di creare un Applet JavaFX da eseguire all'interno di una pagina web, che può essere letteralmente trascinata sulla scrivania di lavoro del PC; così è possibile lanciare l'applicazione web anche al di fuori del browser, direttamente dal Desktop.
- **Binding** (letteralmente "legare"): la possibilità di associare ad una variabile A (in questo caso un numero) una espressione ESPR. Scrivendo:
  var A : Number = bind ESRP
  ogni qualvolta ESPR cambia valore, anche il valore di A (ovunque sia utilizzato) varia. Questo introduce grossi problemi di prestazioni, soprattutto nel caso di abuso di binding, ma semplifica estremamente la comune interazione tra componenti grafiche. Si può facilmente associare il valore di un qualunque controllo, ad esempio una barra di scorrimento, all'attributo di un altro componente (situazione molto comune), come la dimensione di una immagine, o la velocità di un'animazione.

Ambienti e tool di sviluppo

In aggiunta al pacchetto JavaFX SDK, che include compilatore, utilità per l'esecuzione e tutta la libreria JavaFX necessaria per lo sviluppo, sono stati rilasciati alcuni strumenti che rendono lo sviluppo in JavaFX molto più agile.

NetBeans per JavaFX (http://netbeans.org/features/javafx/)
Un componente aggiuntivo per NetBeans che integra tutte le fasi di sviluppo JavaFX in un unico IDE

JavaFX Scene Builder

(http://www.oracle.com/technetwork/java/javafx/tools/index.html)

Si tratta di un tool visuale per produrre applicazioni JavaFX. Consente al programmatore di produrre interfacce grafiche senza scrivere codice. Il codice viene prodotto in automatico dal tool. Sulla barra laterale sono disposti i vari componenti dell'interfaccia grafica (caselle di testo, form, combo, ecc.); il programmatore può costruire l'interfaccia semplicemente trascinando tali componenti dalla barra laterale all'interfaccia che sta disegnando. In seguito è possibile modificare le proprietà di tali componenti. Tutto il codice è prodotto in automatico in background. Il risultato è un file FXML che potrà essere integrato nella propria applicazione JavaFX.

E(fx)clipse (http://efxclipse.org/index.html)

E(fx)clipse è un insieme di plugin per Eclipse che consentono lo sviluppo di applicazioni JavaFX. Esso prevede procedure guidate, editor specifici per CSS, XML e tutto ciò che è necessario per la produzione di applicazioni JavaFX. E' in grado di mostrare in tempo reale tutte le modifiche che si apportano all'interfaccia grafica, senza la necessità di compilare ed eseguire l'applicazione. Inoltre se non siamo soddisfatti dell'uso di XML per la progettazione dell'interfaccia utente, il sistema fornisce un piccolo JSON-Like DSL che consente di definire interfacce utente usando JSON, escludendo XML.

10
VBScript

VBScript (Visual Basic Scripting) è un sottoinsieme di Visual Basic utilizzato nelle Active Server Pages (ASP) e in Windows Script Host come linguaggio di scripting general-purpose.
VBScript è un linguaggio interpretato da uno script engine che può trovarsi sia all'interno che all'esterno di un web server. Nel primo caso, il codice VBScript, inserito all'interno del codice HTML, viene interpretato ed eseguito quando la pagina corrispondente viene richiesta. Nel secondo caso, si possono creare degli script in VBScript che possono essere eseguiti dalla shell o dal desktop.
La versione di VBScript presente all'interno di Internet Explorer offre praticamente le stesse funzionalità di JavaScript ma, data la sua incompatibilità con gli altri browser, molti programmatori preferiscono utilizzare JavaScript. Per lo stesso motivo lo spazio dedicato a questo linguaggio di scripting sarà molto limitato e non ci saranno esempi online.

Come si usa

Per usare un linguaggio di scripting in un browser è necessario comunicare al browser sia la presenza di uno script sia il linguaggio in cui lo script è scritto.
Abbiamo già visto che HTML prevede **il tag <script>** per comunicare al browser l'inizio di uno script. Usiamo quindi questo tag associato all'**attributo type** impostato sul valore "text/vbscript" per indicare al browser che il linguaggio usato sarà Visual Basic Script.

```
<html>
    <head>
```

```
        <title>Primo esempio VBS</title>
    </head>
    <body>
        <script type="text/vbscript">
            document.write "Hello world!"
        </script>
    </body>
</html>
```

In questo modo il browser scriverà, nel punto in cui è posizionato lo script, il testo "Hello world!".
Purtroppo esistono alcuni browser che non supportano il tag <script>. Tali browser invece di eseguire lo script e quindi scrivere a video "Hello world!", mostrerebbero a video tutto il testo, scrivendo document.write "Hello world!". Per evitare questo problema possiamo racchiudere in un commento il codice dello script e aggiungere **il tag <noscript>** che indica al browser cosa fare qualora non supportasse il tag <script>.

```
<script type="text/vbscript">
    <!--
    document.write "Hello, world!"
    -->
</script>
<noscript>
    Sembra che il tuo browser non supporti VBScript!
</noscript>
```

Questa tecnica vale per tutti i linguaggi di scripting.

Le variabili

Una variabile è un contenitore temporaneo per le informazioni che vogliamo memorizzare. Il valore di una variabile può variare durante l'esecuzione dello script. In VBScript tutte le variabili sono varianti; ciò significa che una variabile che in un certo momento dello script memorizza un numero intero, in un secondo momento potrebbe memorizzare una stringa o un altro qualsiasi tipo di dato.
Per definire una variabile dobbiamo darle un nome. Per dare un nome a una variabile bisogna seguire delle regole precise. In VBScript le regole da seguire nell'assegnazione di un nome ad una variabile sono le seguenti:

- devono iniziare con una lettera
- non possono contenere il punto(.)
- non possono superare i 225 caratteri

Dichiarare e assegnare un valore alle variabili

Possiamo dichiarare le variabili utilizzando Dim, Public o Private. Esempio:

```
<script type="text/vbscript">
    option explicit
    dim nomeProprio
    nomeProprio=”Alessandro”
    document.write nomeProprio
</script>
```

Nell’esempio abbiamo dichiarato una variabile e le abbiamo assegnato il nome “nomeProprio”. Poi abbiamo assegnato alla variabile “nomeProprio” il valore “Alessandro”. Infine abbiamo scritto sulla pagina web il valore della variabile.
La riga option explicit serve per impedire che passino inosservati eventuali errori di digitazione. Quando usiamo questa opzione siamo obbligati a dichiarare tutte le variabili utilizzando Dim, Public o Private.

La durata delle variabili

Quando dichiariamo una variabile con una procedura, la variabile può essere utilizzata solamente con la procedura. Quando la procedura termina, la variabile viene eliminata. Queste variabili vengono chiamate “variabili locali”. Possiamo avere delle variabili locali con lo stesso nome in diverse procedure, perché la visibilità della variabile è ristretta alla procedura in cui viene dichiarata.
Se invece dichiariamo una variabile fuori da una procedura, tutte le procedure della pagina possono utilizzarla. La durata di queste variabili inizia quando vengono dichiarate e finisce quando la pagina viene chiusa.

Gli array

Dovremmo avere già un’idea di cosa sia un’array. Tuttavia riprendiamo il concetto. Può capitare di trovarsi nella necessità di dover assegnare più di un valore ad una variabile. In questo caso possiamo creare una variabile che contenga una serie di valori. Questa variabile viene chiamata “variabile Array” o semplicemente array. Per dichiarare un array

si usano le parentesi () che seguono il nome della variabile. Nel seguente esempio, un'array contiene 3 elementi dichiarati:

```
dim nomi(2)
```

Un array inizia a contare i propri elementi dallo zero, quindi scrivere

```
dim nomi(2)
```

significa creare un aray di 3 elementi. Possiamo assegnare i valori all'array in questo modo:

```
nomi(0)="Michele"
nomi(1)="Lorenzo"
nomi(2)="Marcello"
```

Nello stesso modo, il valore può essere ripreso da uno degli elementi usando un indice in un particolare elemento dell'array. L'espressione:

```
padre=nomi(0)
```

assegnerà alla variabile "padre" il valore "Michele".
Possiamo anche usare array multidimensionali. Ad esempio una matrice. In questo caso le dimensioni multiple vengono dichiarate usando la virgola come separazione. Esempio:

```
dim matrice(4, 6)
```

dichiara un array bidimensionale chiamato "matrice" con 5 righe e 7 colonne.

Procedure e funzioni

La differenza sostanziale tra una procedura e una funzione riguarda la restituzione di un valore. Le procedure non restituiscono alcun valore; le funzioni invece devono restituire un valore. Una procedura si dichiara tramite la parola chiave **Sub**. Le funzioni tramite la parola chiave **Function**.
Vediamole nei dettagli.

Una procedura:

- è una serie di dichiarazioni, incluse tra "Sub" e "End Sub"

- può compiere azioni, ma non può restituire un valore
- può utilizzare gli argomenti che gli vengono passati da una procedura
- senza argomenti, deve includere una serie vuota di parametri ()

Esempio di procedura senza parametri.

```
Sub mysub()
    codice
End Sub
```

Esempio di procedura con parametri.

```
Sub mysub(argomento1, argomento2)
    codice
End Sub
```

Una funzione:

- è una serie di dichiarazioni, incluse tra "Function" e "End function"
- può compiere azioni e può restituire un valore
- può utilizzare gli argomenti che gli vengono passati da una procedura
- senza argomenti, deve includere una serie vuota di parentesi ()

Esempio di funzione senza parametrii.

```
Function myfunction()
    Codice
    myfunction=valore
End Function
```

Esempio di funzione con parametri.

```
Function myfunction (argomento1, argomento2)
    Codice
    myfunction=valore
End Function
```

Osserviamo come, in una funzione, prima della fine è necessario assegnare un valore myfunction=valore.

Richiamare una procedura o una funzione

Per richiamare una funzione possiamo scrivere:

```
nome=cercaNome()
```

In questo caso stiamo richiamando la funzione "cercaNome" che evidentemente restituirà un valore che sarà assegnato alla variabile "nome".
Per richiamare una procedura invece si procede nel seguente modo:

```
Call Myproc(argomento)
```

E' possibile omettere la parola chiave Call, scrivendo semplicemente:

```
Myproc argomento
```

Gestire le condizioni

In un linguaggio di programmazione ci troviamo molto spesso a dover decidere di eseguire una certa azione solo se si verificano determinate condizioni. A questo scopo esistono i controlli condizionali che hanno come scopo proprio quello di eseguire un certo codice solo in certe condizioni.
In VBScript ci sono 3 tipi di istruzioni condizionali:

- if … then … else
- if … then … elseif
- select case

La prima forma di controllo condizionale, **if ... then ... else**, la usiamo quando ci troviamo di fronte ad una scelta biunivoca, ossia o facciamo una cosa o ne facciamo un'altra. La sintassi è la seguente:

```
If (condizione) Then
    istruzioni da eseguire se "condizione" è vera
Else
    istruzioni da eseguire se "condizione" è falsa
End If
```

Nel caso in cui le condizioni da verificare sono più di 2, possiamo usare l'istruzione **if ... then ... elseif** la cui sintassi è la seguente:

```
If (condizione1) Then
    istruzioni
ElseIf (condizione2)
    istruzioni
Else
    istruzioni
End If
```

Infine l'istruzione **Select Case** consente di eseguire uno (e uno solo) dei vari gruppi di istruzioni disponibili in base al valore di un'espressione. La sua sintassi è la seguente:

```
Select Case (espressione)
    Case "valore1"
          Istruzioni
    Case "valore2"
          Istruzioni
    Case "valore3"
          Istruzioni
    Case Else
          codice_di_default
End Select
```

Se "espressione" ha valore "valore1" viene eseguito il primo Case. Se ha valore "valore2" viene eseguito il secondo Case. Se non ha nessuno dei valori indicati, viene eseguito il codice "Case Else".

I cicli

I cicli vengono utilizzati per ripetere una certa sequenza di operazioni per un numero determinato di volte o finché non si verifica una certa condizione.
In VBScript ci sono 3 diversi tipi di cicli:

- for next
- for each … next
- do … loop
- while … wend

Possiamo usare l'istruzione **For...Next** per eseguire un blocco di codice, quando sappiamo a priori quante volte dovrà essere ripetuto. La sintassi da usare è abbastanza semplice:

For contatore = valoreIniziale To valoreFinale
    codice
Next

Le istruzioni presenti nel ciclo For saranno ripetute fino a quando la variabile "contatore" non raggiungerà il valore indicato in "valoreFinale". Ad esempio:

```
For i=1 To 10
    codice
Next
```

Possiamo usare anche la parole chiave Step che ci consente di aumentare il valore del contatore di un valore arbitrario. Nell'esempio seguente, il contatore (i) è incrementato di 2 ogni volta che il ciclo riparte:

```
For i=2 To 10 Step 2
    codice
Next
```

Per decrementare il valore del contatore possiamo usare un valore negativo per Step. Nell'esempio seguente, il valore del contatore (i) viene decrementato di 2 ogni volta che il ciclo riparte:

```
For i=10 To 0 Step=-2
    codice
Next
```

I cicli **For Each...Next** ripetono un blocco di codice per ogni voce di una collection o per ogni elemento di un'array. Nel seguente esempio vengono stampati tutti i nomi presenti nell'array nomi().

```
dim nomi(2)
nomi(0)="Michele"
nomi(1)="Lorenzo"
nomi(2)="Marcello"
For Each nome in nomi
```

```
    document.write (nome)
Next
```

Il ciclo **Do ... Loop** fa eseguire il blocco di codice viene ripetuto fintanto che una condizione è vera. Vengono utilizzate le parole chiave **While** e **Until** per completare l'istruzione.
Il ciclo
Do While (condizione)
    codice
Loop

ripeterà il "codice" fino a quando la condizione tra perentesi sarà vera. Se la condizione non è mai vera il "codice" non sarà mai eseguito. Al contrario, il ciclo:
Do
    codice
Loop While (condizione)

eseguirà il "codice" almeno una volta.

Utilizzando until il codice viene eseguito fino a quando la condizione non diventa verà (se è falsa eseguo il codice, quando diventa vera esco). E' l'esatto opposto del while:
Do Until (condizione)
    codice
Loop

oppure
Do
    codice
Loop Until (condizione)

Infine il ciclo **While...Wend** è un modo analogo a do..loop per eseguire codice ciclico.
While (condizione)
    codice
Wend

11
JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web. Fu originariamente sviluppato da Brendan Eich della Netscape Communications con il nome di Mocha e successivamente di LiveScript, ma in seguito è stato rinominato JavaScript ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java di Oracle.
JavaScript è stato standardizzato per la prima volta tra il 1997 e il 1999 dalla ECMA con il nome ECMAScript. L'ultimo standard, di Marzo 2011, è ECMA-262 Edition 5.1 ed è anche uno standard ISO.
Per un programmatore che vuole produrre codice per il web conoscere javascript è un must. Non si può essere programmatori per il web senza avere una buona conoscenza di questo linguaggio. Per questo motivo lo spazio dedicato all'argomento sarà molto superiore a quello dedicato a VBScript.

Quando ci si avvicina per la prima volta a questo linguaggio sorge un dubbio mistico: quale è il legame tra java e javascript? La risposta è semplice: nessuno. Hanno in comune una parte della sintassi, ma sono due linguaggi completamente diversi che fanno cose completamente diverse.

Ecco le principali caratteristiche di javascript:

- è in grado di reagire agli eventi. Si può scrivere del codice javascript da eseguire quando una pagina web ha terminato il caricamento, oppure quando un utente fa click su un determinato elemento HTML

- è in grado di leggere e scrivere gli elementi HTML. Tramite javascript è infatti possibile modificare la struttura del documento HTML in tempo reale, senza interagire con il server
- può essere utilizzato per convalidare i dati. Quando un utente inserisce dei dati in un form HTML, possiamo usare javascript per verificare la correttezza dei dati inseriti prima che essi vengano spediti al server
- può essere utilizzato per avere informazioni sul browser del visitatore. In questo modo possiamo decidere come comportarci a seconda del browser che sta leggendo la pagina
- può essere utilizzato per creare i cookie e quindi archiviare e recuperare informazioni sul computer del visitatore

Come si usa

Abbiamo già detto che per usare un linguaggio di scripting in un browser è necessario comunicare al browser sia la presenza di uno script sia il linguaggio in cui lo script è scritto. Abbiamo visto che questa operazione viene eseguita tramite **il tag <script>** e il suo **attributo type** che, in questo caso, avrà valore text/javascript.

```
<html>
    <body>
          <h1>My First Web Page</h1>
          <script type="text/javascript">
                document.write("<p>" + Date() + "</p>");
          </script>
    </body>
</html>
```

Se invece si vuole recuperare il codice javascript da un file esterno, allora è necessario aggiungere l'**attributo src** al tag script:

```
<script type="text/javascript" src="javascript.js"></script>
```

Con questa direttiva il browser cercherà il file javascript.js nella stessa cartella in cui si trova il file html che lo richiama e renderà disponibile alla pagina HTML il codice presente nell file js. Nel primo caso il tag <script> viene in genere posizionato annidato nel tag <body>; nel secondo caso invece viene in genere posizionato nel tag <head>. Ma questa non è una regola. La regola è che non venga richiamato da javascript un

elemento HTML prima che quest'ultimo sia stato disegnato sulla pagina web. Possiamo usare un numero pressoché illimitato di tag <script> all'interno delle nostre pagine web. Proprio per questo motivo sarebbe opportuno inserirli tutti nello stesso punto, in modo da poterli trovare facilmente. L'uso comune li vuole tutti nel tag <head>.

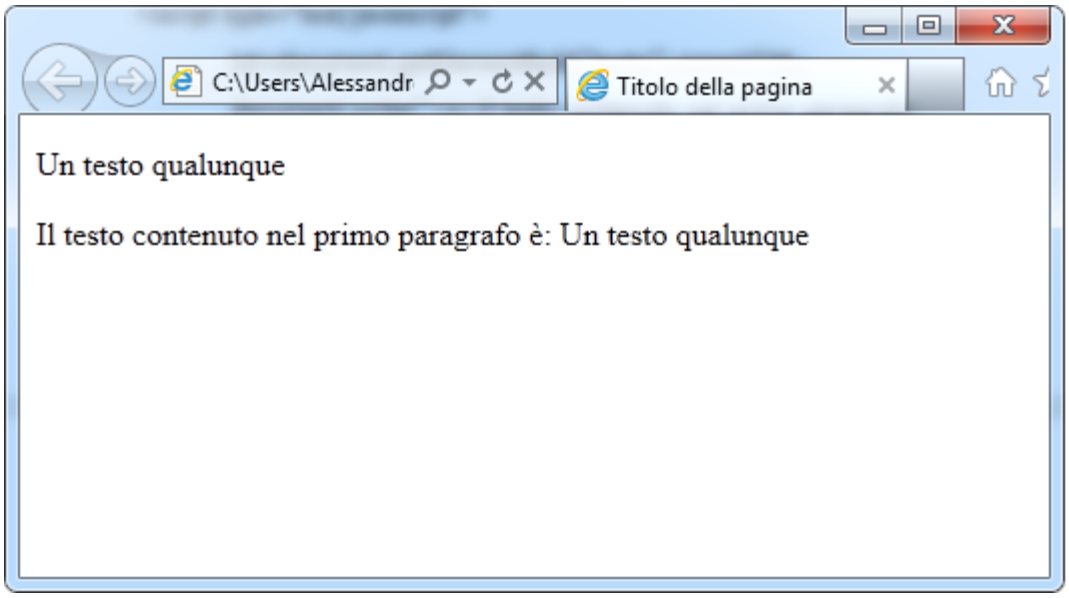
Modificare gli elementi HTML

Come abbiamo visto quando abbiamo parlato del DOM HTML, javascript è in grado di modificare, in tempo reale, gli elementi HTML e quindi la struttura stessa del documento HTML.

Riprendiamo allora l'esempio di base usato quando abbiamo parlato del DOM.

```
<html>
    <head>
          <title>Titolo della pagina</title>
    </head>
    <body>
          <p id="testo">Un testo qualunque</p>
          <script type="text/javascript">
                txt=document.getElementById("testo").innerHTML;
                document.write("<p>Il testo contenuto nel primo
                paragrafo è: " + txt + "</p>");
          </script>
    </body>
</html>
```

che produce la seguente pagina web.



Abbiamo già visto che alcuni browser non supportano il tag <script>. Tali browser invece di eseguire lo script mostrerebbero a video il codice che lo compone. Per evitare questo problema possiamo racchiudere in un commento il codice dello script e aggiungere **il tag <noscript>** che indica al browser cosa fare qualora non supportasse il tag <script>.

```
<script type="text/javascript">
    <!—
    txt=document.getElementById("testo").innerHTML;
    document.write("<p>Il testo contenuto nel primo paragrafo è:
    " + txt + "</p>");
    -->
</script>
<noscript>
    Sembra che il tuo browser non supporti JavaScript!
</noscript>
```

Il tag <noscript> deve essere un discendente (a qualsiasi livello) del tag <body>. Non può quindi essere inserito nel tag <head>.
Questa tecnica vale per tutti i linguaggi di scripting.

Il linguaggio

Javascript è un linguaggio di programmazione case-sensitive, ossia fa differenza tra minuscole e maiuscole. Questo significa che per javascript una variabile denominata var1 è diversa da una variabile denominata Var1.

E' uso comune terminare ogni comando javascript con un punto e virgola. Ciò non è obbligatorio, ma è opportuno per una maggiore leggibilità del codice.

La variabili

Una variabile, in un qualunque linguaggio di programmazione, rappresenta un generico valore che può cambiare nel tempo, da cui il termine variabile.
Per poter utilizzare una variabile all'interno di un linguaggio di programmazione bisogna innanzitutto dichiarala e assegnarle un valore.
Per dichiarare una variabile in javascript bisogna usare la parola chiave **var.** Per scegliere il nome bisogna seguire le seguenti regole:

- i nomi delle variabili sono case-sensitive (cioè, come già detto, x è diverso da X)
- il nome di una variabile deve iniziare con una lettera, il carattere $ o un underscore (trattino basso)

Infine per assegnare un valore ad una variabile dobbiamo usare il simbolo di uguaglianza. Ad esempio:

```
var nome;
var cognome;
nome="Alessandro";
cognome="Stella";
```

E' anche possibile assegnare un valore a una variabile contestualmente alla sua dichiarazione.

```
var nome="Alex";
```

In questo modo abbiamo dichiarato la variabile "nome" e le abbiamo assegnato il valore "Alex".
Un concetto molto importante quando si parla di variabili è **il concetto di scope** (in italiano visibilità). Lo scope di una variabile indica la possibilità (o meno) di richiamare una variabile in un determinato punto del programma. Una variabile infatti ha uno scope diverso a seconda del punto del codice in cui viene dichiarata. In javascript ci sono due tipi di scope: locale e globale. Una variabile ha **scope locale** quando viene dichiarata all'interno di una funzione. Dichiarandola infatti all'interno di una funzione può essere usata solo all'interno di tale funzione (all'esterno della funzione non può essere richiamata). Ha invece **scope globale** quando viene dichiarata fuori da una funzione. In

questo caso può essere richiamata da una parte qualunque del codice. Vedremo più avanti le funzioni.

Gli operatori

Per eseguire operazioni matematiche possiamo usare i seguenti **operatori aritmetici**:
+, -, *, /, %, ++, --
Supponendo che sia stata dichiarata una variabile y=10, la seguente tabella mostra esempi di utilizzo esplicativi degli operatori aritmetici.

Operatore	Esempio	Risultato	
+	x=y+5	x=15	y=10
-	x=y-5	x=5	y=10
*	x=y*5	x=50	y=10
/	x=y/5	x=2	y=10
%	x=y%3	x=1	y=10
++	x=y++	x=10	y=11
	x=++y	x=11	y=11
--	x=y--	x=10	y=9
	x=--y	x=9	y=9

L'operatore % viene chiamato modulo e restituisce la parte intera del resto della divisione tra i numeri indicati. Nel nostro caso 10/3 ha come resto intero 1.
L'operatore ++ viene chiamato incremento e, come mostrato dalla tabella, può essere usato in due modi diversi con due risultati diversi. Vediamone l'uso un po' più nel dettaglio. Dobbiamo assegnare un valore alla variabile x legato al valore della variabile y. Scrivere x=y++ significa eseguire due operazioni:

- porre x=y
- aumentare di uno il valore di y

Poiché y=10, la prima operazione rende x=10. La seconda operazione (che avviene quando la prima è già stata effettuata) aumenta di uno il valore di y, da cui y=11. Ecco perché alla fine x=10 e y=11.
Scrivere x=++y significa eseguire le precedenti due operazioni in senso inverso:

- aumentare di uno il valore di y
- porre x=y

Quindi prima diventa y=11 e poi x=y, da cui, alla fine, x=11 e y=11.
L'operatore -- viene chiamato decremento e vale il discorso fatto per l'incremento.

Per eseguire **operazioni di assegnamento**, oltre al già visto simbolo dell'uguaglianza, possono essere usati anche i seguenti operatori:
+=, -=, *=, /=, %=
Supponendo di aver dichiarato due variabili x=20 e y=10, la seguente tabella mostra degli esempi di utilizzo.

Operatore	Esempio	Equivale a	Risultato
+=	x+=y	x=x+y	x=30
-=	x-=y	x=x-y	x=10
*=	x*=y	x=x*y	x=200
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Insomma, niente di particolarmente complicato. Ci sono però alcuni casi particolari. Ammettiamo, per esempio, di trovarci nella seguente situazione.

```
var stringa="Alex";
var numero=1;
var somma=stringa+numero;
```

Quanto vale la variabile somma?
La risposta è "Alex1" perché la legge che governa il comportamento dell'operatore + è la seguente:
numero+numero=numero
numero+stringa=stringa
stringa+numero=stringa
Ossia se uno dei due addendi è una stringa, l'operatore + considera stringa anche l'altro addendo. Facciamo attenzione al fatto che un numero tra doppi apici è una stringa, non un numero. Cioè l'istruzione

```
var prova="5"+5;
```

assegnerà a prova il valore "55", mentre l'istruzione

```
var prova=5+5;
```

assegnerà a "prova" il valore numerico 10.

Facciamo un semplice esempio. Scriviamo il seguente codice javascript:

```
<script type="text/javascript">
    <!--
    var prova1="5"+5;
    var prova2=5+5;
    var prova3=5;
    var prova4=prova3+=prova2;
    document.getElementById("testo").innerHTML="prova1 (stringa
          + numero): " + prova1 + " - prova2 (numero + numero):
          " + prova2 + " - prova3 (modificata dall'ultima
          istruzione): " + prova3;
    -->
</script>
```

Possiamo osservare il risultato in azione qui:
http://www.alessandrostella.it/lato_client/js/js_01.html

I confronti

Gli operatori di confronto danno come risultato una variabile booleana, cioè vero o falso. Nella seguente tabella troviamo l'elenco degli **operatori di confronto** e alcuni esempi esplicativi. Supponiamo di aver dichiarato la variabile x=1.

Operatore	Esempio	Risultato
==	x==1; x==5	true; false
===	x===1; x==="1"	true; false
!=	x!=1; x!=5	false; true
>	x>0; x>5	true; false
<	x<0; x<5	false; true
>=	x>=0; x>=5	true; false
<=	x<=1; x<=0	true; false

Oltre agli operatori appena visti, abbiamo a disposizione anche gli **operatori logici**, che sono comunque operatori di confronto e restituiscono un valore booleano.
Supponendo di aver dichiarato due variabili x=1 e y=5.

Operatore	Esempio	Risultato
&&	(x=1 && y>5)	false
||	(x>3 || y>3)	true
!	!x<1	true

L'operatore && (and) restituisce true solo e soltanto se entrambe le condizioni poste sono vere.
L'operatore || (or) restituisce true se almeno una delle due condizioni è vera.
L'operatore ! (not) nega il risultato del confronto che lo segue (se è vero lo rende falso e viceversa).
Possiamo osservare un piccolo esempio qui:
http://www.alessandrostella.it/lato_client/js/js_02.html

Le condizioni

In javascript i controlli condizionali vengono realizzati usando i seguenti comandi:

- if
- if... else
- if... else if... else
- switch

Un controllo condizionale esegue un determinato codice al verificarsi di alcune condizioni.
Il più semplice codice condizionale possibile è il seguente.

```
if (condizione) {
    codice da eseguire se la condizione è vera
}
```

In questo caso, se è vero quanto riportato nella parentesi tonda, allora viene eseguito il codice tra le parentesi graffe.
Una piccola evoluzione consente di eseguire un codice alternativo da eseguire solo nel caso in cui la condizione nelle parentesi tonde risultasse falsa.

```
if (condizione) {
```

```
    codice da eseguire se la condizione è vera
} else {
    codice da eseguire se la condizione è falsa
}
```

Se è vero quanto riportato nella parentesi tonda, allora viene eseguito il codice immediatamente successivo, altrimenti viene eseguito il codice che segue il comando else.
Infine, qualora ci fossero molti condizioni da controllare si può usare la forma più complessa del comando if.

```
if (condizione) {
    codice da eseguire se condizione è vera
} else if (condizione2) {
    codice da eseguire se condizione2 è vera
} else if (condizione3) {
    codice da eseguire se condizione3 è vera
} else {
    codice da eseguire se TUTTE le precedenti condizioni
    sono false
}
```

In quest'ultimo caso viene verificata la prima condizione. Se è vera viene eseguito il codice nelle parentesi graffe, altrimenti si procede ad oltranza nel controllo di tutte le condizioni impostate (in questo caso 3). Se alla fine, tutte le condizioni effettuate hanno dato esito negativo, viene eseguito il codice dell'else (se presente).

Lo switch è un altro modo di intendere la sequenza di "else if" appena vista. Data una variabile n di tipo intero, un tipico switch si mostra nel seguente modo.

```
switch (n) {
    case 1:
          codice da eseguire se n=1
          break;
    case 2:
          codice da eseguire se n=2
          break;
    default:
```

```
            codice da eseguire se n non corrisponde a nessuno dei
            recedenti valori
}
```

Se n vale 1 viene eseguito il primo case e così via. Se n ha un volare diverso da tutti quelli previsti, allora si esegue il default.
Da notare l'uso obbligatorio della **parola chiave break** che serve per fermare lo switch dopo aver eseguito i comandi previsti. Se si omette la parola chiave break il codice prosegue con il case successivo.
Possiamo osservare un utilizzo dei controlli condizionali qui:
http://www.alessandrostella.it/lato_client/js/js_03.html

I cicli

Per ripetere più volte la stessa sequenza di operazioni, javascript prevede due tipi di cicli:

- for
- while

Il **ciclo for** ripete le istruzioni scritte tra parentesi graffe per il numero di volte previsto nelle parentesi tonde. Ossia:

```
var i=0;
for (i=0;i<=5;i++) {
    codice da eseguire;
}
```

Questo codice ripeterà 6 volte (da 0 a 5) il codice tra parentesi graffe.
Il **ciclo while** ripete le istruzioni scritte tra parentesi graffe fino a quando la condizione tra parentesi tonde è vera.

```
var i=0;
while (i<=5) {
    codice da eseguire;
    i++;
}
```

La differenza tra i due cicli quindi è la seguente:

- il ciclo for ripete il codice un numero specificato di volte

- il ciclo while ripete il codice fino a quando non si verifica una determinata condizione

Cerchiamo di essere più precisi con alcuni esempi.
Ammettiamo ad esempio di avere il seguente codice HTML:

```
<p id="for"></p>
```

e di voler scrivere nel paragrafo #for 5 righe indicate ognuna da un numero. Tramite un ciclo for potremmo scrivere:

```
var i=0;
var testo="";
for (i=0;i<5;i++) {
    testo=document.getElementById("for").innerHTML;
    document.getElementById("for").innerHTML=testo + "Questa è
          la riga " + i + " creata dal ciclo <i>for</i><br>";
}
```

Possiamo ottenere lo stesso risultato usando un ciclo while:

```
while (i<5) {
    testo=document.getElementById("while").innerHTML;
    document.getElementById("while").innerHTML=testo + "Questa è
          la riga " + i + " creata dal ciclo <i>while</i><br>";
    i++;
}
```

Possiamo anche osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/js/js_04.html

Le funzioni

Le funzioni sono una componente molto importante del linguaggio. Se per esempio volessimo eseguire un determinato codice al verificarsi di un determinato evento (come potrebbe essere il click del mouse su un bottone) dobbiamo usare una funzione. Vediamo allora come si dichiara e come si richiama una funzione in javascript.
In generale per dichiarare una funzione bisogna seguire la seguente sintassi.

```
function (parametro1, parametro2, …, parametroN) {
```

```
    codice della funzione;
}
```

I parametri tra parentesi tonde sono opzionali e possono quindi non essere presenti.
Vediamo un piccolo esempio.

```
<html>
    <head>
        <script type="text/javascript">
            function mostraMessaggio() {
                alert("Hai fatto click sul bottone");
            }
        </script>
    </head>
    <body>
        <button type="button"
            onclick="mostraMessaggio()">Clicca qui</button>
    </body>
</html>
```

Abbiamo dichiarato la funzione mostraMessaggio() e l'abbiamo richiamata sul click del bottone (attributo onclick del tag <button>). Ogni volta che faremo click sul bottone, comparirà una finestra di popup con scritto "Hai fatto click sul bottone".
Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/js/js_05.html

E' opportuno fare una piccola digressione sugli attributi che possiamo usare per intercettare gli eventi. Nella tabella che segue troviamo un elenco degli attributi che ci consentono di intercettare gli eventi più comuni (elenco non esaustivo). Gli eventi verrano trattati più avanti, allorquando approfondiremo il DOM HTML.

Attributo	**Uso**
onclick	Da usare per intercettare il click del **mouse**
ondblclick	Da usare per intercettare il doppio click del **mouse**
onmousedown	Da usare per intercettare la pressione di un bottone del **mouse**

onmousemove	Da usare per intercettare il movimento del del **mouse** su un certo elemento
onmouseover	Da usare per intercettare il passaggio del **mouse**
onmouseout	Da usare per intercettare l'uscita del **muose** dal box di un elemento
onkeydown	Da usare per intercettare la pressione di un tasto della **tastiera**
onkeypress	Da usare per intercettare una pressione e rilascio di un tasto della **tastiera**
onkeyup	Da usare per intercettare il rilascio di un tasto della **tastiera**
onload	Da usare per intercettare l'avvenuto caricamento della pagina web o di un <object>
onresize	Da usare per intercettare il ridimensionamento di una pagina web
onblur	Da usare in un **form** per intercettare la perdita del fuoco di un elemento del form (<input>, textarea>, ecc)
onchange	Da usare in un **form** per intercettare la modifica del testo presente in un elemento del form
onfocus	Da usare in un **form** per intercettare la presa del fuoco da parte di un elemento del form
onreset	Da usare in un **form** per intercettare la pressione del bottone di reset
onsubmit	Da usare in un **form** per intercettare la pressione del bottone di invio dati

Una funzione può anche restituire un valore dopo aver effettuato alcune operazioni. Per esempio.

```
<script type="text/javascript">
    function somma (a, b) {
          return a+b;
    }
    var sum=somma(3, 5);
</script>
```

In questo caso la variabile sum assumerà valore 8 (3+5). Notiamo **la parola chiave return** usata nella funzione per restituire il valore all'esterno.

La gestione degli errori

Scrivere codice comporta commettere errori dai più banali ai più complessi. Ogni linguaggio di programmazione si occupa di gestire gli errori e javascript non fa eccezione. A tal fine è previsto il comando **try... catch** la cui sintassi è la seguente

```
try  {
    qui un codice qualunque;
}
catch (err) {
    qui il codice da eseguire se nel blocco try si sono
          verificati errori
}
```

Esempio.

```
<html>
    <head>
          <script type="text/javascript">
                function somma (a, b) {
                      try {
                            return a+c;
                      }
                      catch (err) {
                            var errore="";
                            errore="Si è verificato il seguente
                            errore:\n";
                            errore+=err.message + "\n\n";
                            errore+="Premi OK per
                            proseguire.\n";
                            alert (errore);
                      }
                }
          </script>
    </head>
    <body>
          <button type="button"
                onclick="somma(3, 5)">Somma 3+5</button>
    </body>
</html>
```

Abbiamo volutamente commesso l'errore nello scrivere i due addendi della somma (a+c) sapendo che c non è stato definito. Quindi quando javascript cercherà di eseguire quel comando si verificherà un errore. Errore che sarà però intercettato e inviato nel blocco catch, il quale userà l'oggetto err per mostrare un messaggio all'utente in cui si comunica la descrizione dell'errore verificatosi.
Se invece di obbligare l'utente a premere su "OK" volessimo mostrare un popup con duplice scelta (OK, Annulla) ci basterebbe usare l'**oggetto confirm** invece dell'oggetto alert, sostituendo il codice alert (errore) con il seguente:

```
if (!confirm (errore)) {
    document.location.href="http://www.alessandrostella.it/";
}
```

In questo modo se l'utente non conferma (quindi non preme su OK, ma su Annulla), il sistema lo porta su un altro sito.
Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/js/js_06.html

Gli oggetti di JavaScript

Abbiamo detto che javascript è un linguaggio basato sugli oggetti. Possiamo usare sia gli oggetti propri del linguaggio, sia crearne di nostri e usarli nel codice che scriviamo.
E' molto frequente scrivere codice come quello che segue:

```
var nome = "Alex";
```

quello che molto spesso non si sa è che scrivere questo codice significa creare un oggetto di tipo Stringa. Javascript infatti "traduce" silenziosamente il precedente codice nel seguente:

```
var nome = new String("Alex");
```

In questo modo è poi possibile usare sulla variabile "nome" i metodi e le proprietà propri dell'oggetto Stringa. Per esempio potremmo scrivere
nome.length;

La proprietà length dell'oggetto Stringa restituisce il numero di caratteri che compongono l'oggetto. Dovrebbe essere noto che nella programmazione orientata agli oggetti ogni

oggetto è provvisto di proprietà e metodi. Le proprietà rappresentano caratteristiche tipiche dell'oggetto in questione; i metodi sono dei meccanismi che servono per apportare modifiche alle proprietà dell'oggetto cui appartengono.
Javascript prevede i seguenti oggetti di linguaggio:

- Number
- String
- Date
- Array
- Boolean
- Math
- RegExp

Esistono inoltre alcune proprietà e alcuni metodi che sono comuni a tutti gli oggetti del linguaggio. Li vediamo qui di seguito.

Proprietà

Proprietà	**Descrizione**
Infinity	Un valore numerico che rappresenta +/- infinito
NaN	"Not-A-Number" cioè non è un numero
undefined	Significa che l'oggetto a cui si è fatto riferimento non è definito

Metodi

Metodo	**Significato**
eval()	Valuta una stringa e la tratta come fosse uno script eseguibile
isFinite()	Determina se il numero a cui si riferisce è finito o meno
isNaN()	Determina se un numero è o meno un numero
Number()	Converte un qualunque oggetto in un numero (se possibile)
parseFloat()	Trasforma una stringa nel corrispondente numero con virgola (se possibile)
parseInt()	Trasforma una stringa nel corrispondente valore intero (se possibile)
String()	Converte un qualunque oggetto in una stringa

Teniamo sempre presente che **javascript è case-sensitive**!

Entriamo ora nel dettaglio dei singoli oggetti propri del linguaggio. Vedremo in seguito come sia possibile crearne di propri.

L'oggetto Number

Per creare un oggetto di tipo Number si usa la seguente sintassi:

```
var numero = new Number(valore);
```

oppure, equivalentemente,

```
var numero = valore;
```

Scrivendo ad esempio:

```
var numero = 33;
```

Javascript, accorgendosi che il 33 è un numero, eseguirà per noi la creazione del nuovo oggetto di tipo Number, trasformando implicitamente il precedente codice nel seguente:

```
var numero = new Number(33);
```

Un oggetto di tipo Number ha le seguenti proprietà e metodi.

Proprietà

Proprietà	**Significato**
MAX_VALUE	Restituisce il più grande numero disponibile
MIN_VALUE	Restitusice il più piccolo numero disponibile
NEGATIVE_INFINITY	Rappresenta il -infinito
POSITIVE_INFINITY	Rappresenta il +infinito

Metodi

Metodo	**Significato**
toExponential(x)	Converte un numero nella corrispondente notazione esponenziale
toFixed(x)	Formatta un numero con il numero fisso di cifre decimali indicato da x

toPrecision(x)	Formatta un numero con la precisione indicata da x
toString()	Converte il numero in stringa

L'oggetto String

Per creare un oggetto di tipo Stringa si usa la seguente sintassi:

```
var nome = new String (stringa);
```

ma è molto più in uso la forma equivalente

```
var nome = stringa;
```

Esempio:

```
var nome = "Alex";
```

Anche in questo caso, come per l'oggetto di tipo Number, si occuperà javascript di creare l'oggetto di tipo String, trasformando implicitamente il precedente codice nel seguente:

```
var nome = new String("Alex");
```

Di seguito mostreremo alcune proprietà e metodi dell'oggetto String. Vengono elencati solo proprietà e metodi più usati.

Proprietà

Proprietà	**Descrizione**
constructor	Restituisce la funzione che ha creato l'oggetto
length	Restituisce un intero con la lunghezza della stringa
proptotype	Consente di aggiungere proprietà e metodi ad un oggetto

Metodi

Metodo	**Descrizione**
charAt(index)	Restituisce il carattere che si trova nella posizione indicata dall'indice

charCodeAt(index)	Restituisce il codice Unicode del carattere che si trova nella posizione indicata dall'indice
concat(string2, string3, ..., stringX)	Unisce due o più oggetti di tipo Stringa e restituisce una copia dell'oggetto ottenuto
indexOf(searchstring, start)	Restituisce la posizione in cui si trova il primo carattere indicato dal parametro searchstring
lastIndexOf(searchstring, start)	Restituisce la posizione dell'ultimo carattere indicato dal parametro searchstring
match()	Viene utilizzato in associazione ad una regular expression per verificare il match con i criteri indicati nella regular expression
replace(regexp/substr, newstring)	Cerca una stringa all'interno della stringa e la sostituisce con quella indicata dal parametro newstring
substr(start, length)	Restituisce la stringa a partire da un certo indice (indicato da start) e lunga un certo numero di caratteri (indicato da length)
substring(from, to)	Restituisce la perta della stringa compresa tra un certo indice (from) e fino al successivo (to)
toLowerCase()	Converte la stringa in minuscolo
toUpperCase()	Converte la stringa in maiuscolo

Possiamo osservare in azioni alcuni die metodi appena visti:
http://www.alessandrostella.it/lato_client/js/js_07.html

L'oggetto Date

L'oggetto Date è usato per gestire date e orari. Per creare un oggetto di tipo Date basta usare l'apposito costruttore Date(). Per inizializzare una variabile di tipo Date abbiamo a disposizione 4 modi diversi:

```
var d = new Date();
var d = new Date(milliseconds);
var d = new Date(dateString);
var d = new Date(year, month, day, hours, minutes, seconds,
milliseconds);
```

Se viene usato il costruttore senza alcun parametro, viene restituita la data odierna.

Per confrontare due oggetti di tipo Date è sufficiente usare gli operatori <, >, ==.

Esempio:

```
var d1 = new Date(2000, 0, 1);
var d2 = new Date();
if (d1<d2) {
    alert("La prima data è precedente alla seconda");
}
```

Non può esserci sfuggito lo zero usato nel costruttore:

```
var d1 = new Date(2000, 0, 1);
```

Ebbene si, **per indicare Gennaio si usa il numero zero**.

La data viene creata secondo lo standard CEST (Central Europe Standard Time) perché siamo in Italia. Questo significa che scrivere

```
var d1 = new Date(2000, 0, 1);
```

significa assegnare a d1 il seguente valore:

Sat Jan 01 2000 00:00:00 GMT +0100

che magari non è proprio quello che noi vorremmo vedere. Magari a noi interesserebbe di più un formato del tipo 01/01/2000. A tale scopo diamo uno sguardo ai più comuni metodi che l'oggetto Date mette a disposizione.

Metodo	**Descrizione**
getMilliseconds()	Restituisce i millisecondi (0-999)
getDay()	Restituisce il giorno della settimana (0 per Domenica - 6 per Sabato)
getFullYear()	Restituisce l'anno in quattro cifre
getMonth()	Restituisce il mese (0-11)
getDate()	Restituisce il giorno del mese (1-31)
getHours()	Restituisce l'ora (0-23)
getMinutes()	Restituisce i minuti (0-59)
getSeconds()	Restituisce i secondi (0-59)
getTime()	Restituisce il numero di millisecondi trascorsi dal 1 Gennaio 1970

getTimezoneOffset()	Restituisce la differenza in minuti tra l'orario locale e quello di GMT
setMilliseconds()	Setta i millisecondi (0-999)
setFullYear()	Setta l'anno (quattro cifre)
setMonth()	Setta il mese (0-11)
setDate()	Setta il giorno del mese (1-31)
setHours()	Setta l'ora (0-23)
setMinutes()	Setta i minuti (0-59)
setSeconds()	Setta i secondi (0-59)
toDateString()	Trasforma un oggetto Date in un oggetto Stringa leggibile
toString()	Trasforma un oggetto Date in un oggetto Stringa senza modificare nulla
parse()	Trasforma una stringa in Date (se possibile) e restituisce i millisecondi dal 1 Gennaio 1970

Con i metodi appena visti possiamo provare ad ottenere una data così come la vorremmo, ossia nella forma giorno/mese/anno.

```
var d = new Date();
var giorno = d.getDate();
var mese = d.getMonth();
var anno = d.getFullYear();
var miaData=giorno+"/"+mese+"/"+anno;
```

oppure, senza impegnare spazio in 3 variabili inutili:

```
var d = new Date();
var miaData=d.getDate()+"/"+d.getMonth()+"/"+d.getFullYear();
```

Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/js/js_08.html

L'oggetto Array

Un'oggetto di tipo Array viene usato per memorizzare più valori nella stessa variabile.
Per dichiarare una variabile di tipo Array si può usare il costruttore Array():

```
var array = new Array();
var persona = new Array();
persona[0] = "Alessandro";
persona[1] = "Stella";
persona[3] = 38;
```

0	1	2
Alessandro	Stella	38

Un Array può essere immaginato come sopra rappresentato. Tramite un indice, riportato sulla prima riga, possiamo accedere ai dati riportati nella seconda riga. Gli indici iniziano da zero. Osserviamo come i primi due dati siano di tipo Stringa, mentre il terzo sia di tipo Intero. Un array può dunque contenere dati di tipo diverso. Continuando ad aggiungere elementi nell'array l'indice aumenterà.
Per accedere ai dati è sufficiente richiamare il dato contenuto in un particolare indice:

```
var nome = persona[0];
var cognome = persona[1];
var anni = persona[2];
```

E' possibile in ogni momento sovrascrivere il contenuto di un elemento dell'array semplicemente assegnando all'oggetto indicato da un determinato indice il nuovo valore. Nel precedente array questa istruzione

```
persona[0] = "Barbara";
```

modificherà il valore dell'indice zero.
Vediamo ora proprietà e metodi dell'oggetto Array usati più comunemente.

Proprietà

Proprietà	**Significato**
length	Setta o restituisce il numero di elementi che compongono l'array

Metodi

Metodo	**Significato**

concat (array2, array3, ..., arrayX)	Unisce due o più array e restituisce la copia dell'Array così ottenuto
join()	Restituisce una stringa con tutti gli oggetti contenuti nell'array, divisi da una virgola
pop()	Rimuove dall'array l'ultimo elemento inserito e lo restituisce
push()	Inserisce un nuovo elemento nell'array e restituisce l'indice in cui è stato inserito
reverse()	Inverte l'ordine degli elementi di un array
sort()	Ordina gli elementi dell'array in ordine alfabetico

L'oggetto prevede anche altri metodi.

Possiamo osservare un piccolo esempio qui:

http://www.alessandrostella.it/lato_client/js/js_09.html

L'oggetto Boolean

Un oggetto di tipo Boolean è un oggetto di tipo vero/falso.

Per dichiarare un oggetto di tipo Boolean si usa il costruttore:

```
var bool = new Boolean();
```

Se non viene passato alcun argomento al costruttore o se viene passato uno dei seguenti valori:

- 0
- -0
- null
- ""
- false
- undefined
- NaN

l'oggetto viene settato su false.

L'unico metodo usato di questo oggetto è il metodo toString() che restituisce una stringa con il valore dell'oggetto (true o false).

L'oggetto Math

L'oggetto Math viene usato esclusivamente per eseguire calcoli matematici. Math non prevede un costruttore. JavaScript prevede 8 costanti matematiche che possono essere usate per ottenere particolari valori matematici noti. Per richiamarle è sufficiente richiamare l'oggetto Math in uno dei seguenti modi (Attenzione al rispetto delle minuscole/maiuscole):

- Math.E
- Math.PI
- Math.SQRT2
- Math.SQRT1_2
- Math.LN2
- Math.LN10
- Math.LOG2E
- Math.LOG10E

Metodi

Metodo	**Significato**
Math.abs(x)	Restituisce il valore assoluto di x
Math.acos(x)	Restituisce l'arcocoseno di x, in radianti
Math.asin(x)	Restituisce l'arcoseno di x, in radianti
Math.atan(x)	Restituisce l'arcotangente di x, in radianti
Math.ceil(x)	Restituisce l'intero più vicino a x, ma maggiore del valore inserito (Math.ceil(2.1) restituisce 3)
Math.cos(x)	Restituisce il coseno di x, in radianti
Math.exp(x)	Restituisce l'esponenziale di x
Math.floor(x)	Restituisce l'intero più vicino a x, ma minore del valore inserito (Math.ceil(2.1) restituisce 2)
Math.log(x)	Restituisce il logaritmo naturale di x
Math.max(x,y,z,...,n)	Restituisce il numero con il valore più alto
Mat.min(x,y,z,...,n)	Restituisce il numero con il valore minore
Mat.random()	Restituisce un numero casuale tra 0 e 1
Math.round(x)	Restituisce un numero intero più vicino a x
Math.sin(x)	Restituisce il seno di x, in radianti
Math.sqrt(x)	Restituisce la radice quadrata di x

Math.tan(x)	Restituisce la tangente di x, in radianti

Quindi per calcolare la radice quadrata di 4 possiamo scrivere:

```
var risultato = Math.sqrt(4);
```

L'oggetto RegExp

Il nome di questo oggetto dice tutto quel che c'è da dire. Rappresenta un'espressione regolare (regular expression). Un'espressione regolare rappresenta una regola da seguire nel cercare una sequenza di caratteri in una stringa, infatti quando si cerca una particolare sequenza di caratteri in un testo è possibile utilizzare un'espressione regolare per descrivere quello che stiamo cercando.

Le espressioni regolari sono utilizzate per eseguire potenti metodi di ricerca e sostituzione all'interno di un testo e rappresentano un argomento molto complesso bisognoso di uno spazio considerevole che qui non ci possiamo permettere. A noi interessa sapere che javascript prevede dei meccanismi per implementare una regular expression.

Per dichiarare un oggetto RegExp si può procedere in due modi:

```
var regExp = new RegExp (pattern,modifiers);
```

oppure

```
var regExp = /pattern/modifiers;
```

il parametro pattern specifies the pattern of an expression

il parametro modifiers specifica se la ricerca deve essere globale, case-sensitive, ecc.

Per maggiori dettagli sull'argomento, si rimanda sul sito ufficiale

http://www.w3schools.com/jsref/jsref_obj_regexp.asp

Caratteri speciali con javascript

Può capitare che ci si trovi nella necessità di inserire un carattere speciale in una stringa.

Il tipico esempio che può essere preso in considerazione è l'inserimento dei doppi apici.

Se, ad esempio, noi scrivessimo:

```
var txt = "We are the so-called "Vikings" from the north.";
```

commetteremmo un errore. Il sistema non concede questo tipo di sintassi. Per poter inserire i doppi apici all'interno di una stringa dobbiamo procedere in questo modo:

```
var txt = "We are the so-called \"Vikings\" from the north.";
```

Si procede in questo modo perché i doppi apici sono considerati un carattere speciale e quindi vanno trattati come tali. Più in generale ecco l'elenco dei caratteri speciali in javascript e di come procedere per inserirli in un testo.

Codice	**Significato**
\'	Inserisce l'apice nel testo
\"	Inserisce i doppi apici nel testo
\\	Inserisce lo slash (\) nel testo
\n	Inserisce una nuova linea nel testo
\r	Inserisce un carattere di return carriage
\t	Inserisce un carattere di tabulazione
\b	Inserisce un carattere di backspace

JavaScript all'opera

E' arrivato il momento di mettere in pratica le nozioni imparate.
Iniziamo con un tormentone che prima o poi capita nella vita di un programmatore per il web: usare javascript per controllare i dati inseriti in un form.
Iniziamo con una cosa molto semplice, ma pur sempre interessante.
Creiamo un form html per richiedere all'utente Nome, Cognome, Indirizzo civico e cellulare. Noi verificheremo banalmente che i dati inseriti non siano nulli e procederemo alla comunicazione all'utente degli eventuali errori riscontrati. Se i dati sono inseriti in modo corretto, saluteremo l'utente con un Grazie.
Ecco tutto il codice che ci consente di fare quanto premesso.

```
<html>
    <head>
        <title>JavaScript - Controllo Form</title>
        <script type="text/javascript">
```

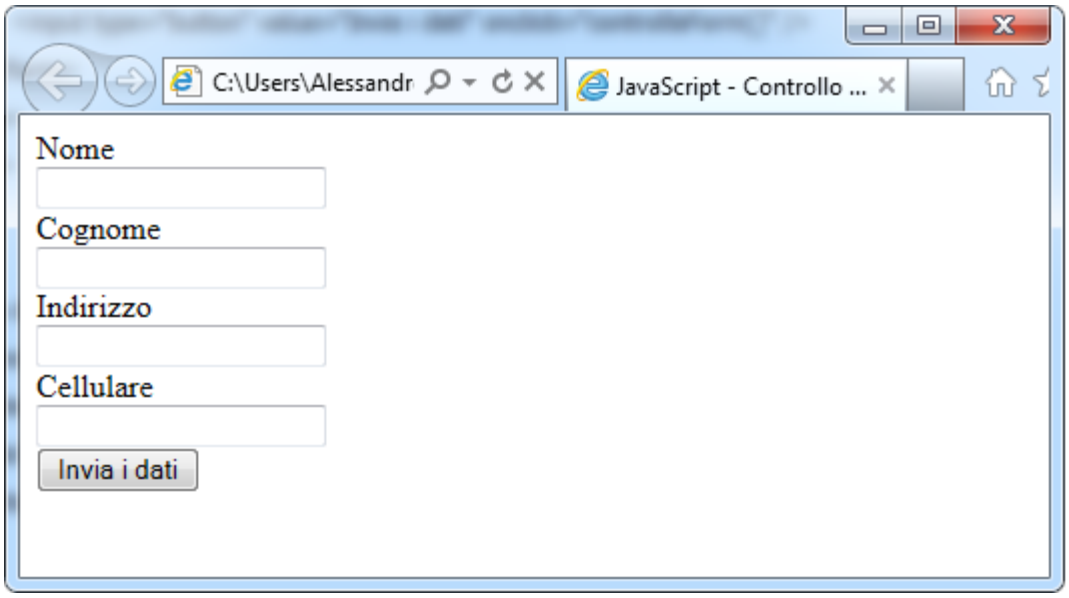
```
function controllaForm() {
    var errori="";
    document.getElementById("nome").
          style.backgroundColor="white";
    document.getElementById("cognome").
          style.backgroundColor="white";
    document.getElementById("indirizzo").
          style.backgroundColor="white";
    if (document.getElementById("nome").value==""){
          errori+="Il nome è obbligatorio.<br />";
          document.getElementById("nome").
          style.backgroundColor="#FFB3B3";
    }
    if (document.getElementById("cognome").
                value=="") {
          errori+="Il cognome è
                obbligatorio.<br />";
          document.getElementById("cognome").
                style.backgroundColor="#FFB3B3";
    }
    if (document.getElementById("indirizzo").
                value=="") {
          errori+="L'indirizzo è
                obbligatorio.<br />";
          document.getElementById("indirizzo").
                style.backgroundColor="#FFB3B3";
    }
    if (errori!="") {
          document.getElementById("risultato").
                innerHTML="Sono stati riscontrati
                i seguenti errori:<br />"+errori;
          document.getElementById("risultato").
                style.color = "red";
          document.getElementById("risultato").
                style.display = "block";
    } else {
          document.getElementById("nome").
                style.backgroundColor="white";
          document.getElementById("cognome").
```

```
                        style.backgroundColor="white";
                    document.getElementById("indirizzo").
                        style.backgroundColor="white";
                    document.getElementById("risultato").
                        style.display = "none";
                    document.getElementById("modulo").
                        innerHTML = "<h4>Grazie</h4>";
                }
            }
        </script>
    </head>
    <body>
        <p id="risultato" style="display:none"></p>
        <div id="modulo">
            <form name="input" method="post">
                <label for="nome">Nome<br /></label>
                <input id="nome" type="text" name="nome" />
                <br />
                <label for="cognome">Cognome<br /></label>
                <input id="cognome" type="text"
                    name="cognome" />
                <br />
                <label for="indirizzo">Indirizzo<br /></label>
                <input id="indirizzo" type="text"
                    name="indirizzo" />
                <br />
                <label for="cellulare">Cellulare<br /></label>
                <input id="cellulare" type="text"
                    name="cellulare" />
                <br />
                <input type="button" value="Invia i dati"
                    onclick="controllaForm()" />
            </form>
        </div>
    </body>
</html>
```

Ed ecco dove poterlo vedere in azione:

http://www.alessandrostella.it/lato_client/js/js_10.html

Cerchiamo di darne una minima interpretazione.
Il form è molto semplice ed è esteticamente abbastanza brutto. In questo contesto però non è nostro interesse produrre di meglio.
Nel codice HTML abbiamo previsto un paragrafo <p> nascosto (display="none") con id="risultato" nel quale inseriremo gli eventuali errori da mostrare all'utente. Il form è a sua volta inserito in un con id="modulo".
Prendiamo come ipotesi che i campi Nome, Cognome e Indirizzo siano obbligatori, mentre il campo Cellulare non lo sia.

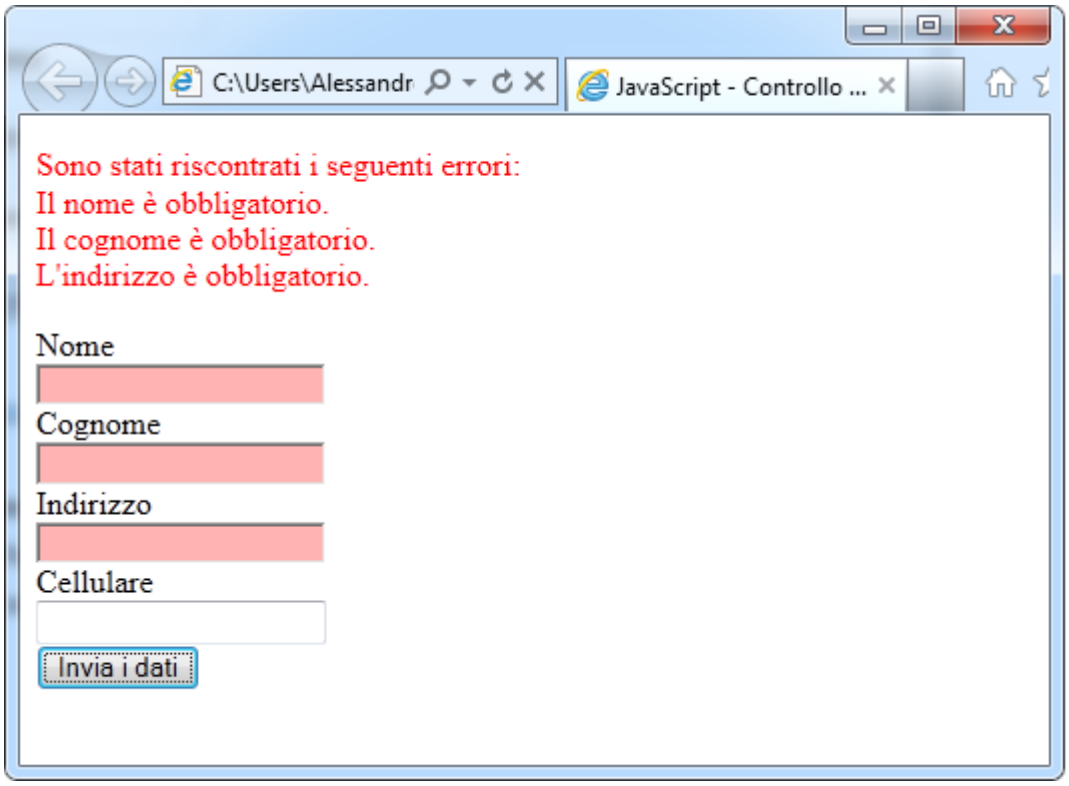


Quando l'utente preme sul bottone, il sistema lancia la funzione "controllaForm()", per fare questo abbiamo usato l'attributo onclick del bottone.
Cosa fa la funzione?
Per prima cosa crea la variabile errori e setta di colore bianco lo sfondo dei campi del modulo. Poi verifica che nei campi obbligatori sia stato inserito qualche carattere. Per ogni campo che trova vuoto aggiunge una riga alla stringa "errori" e cambia il colore di sfondo del campo. Cioè:

```
errori+="Il nome è obbligatorio.<br />";
document.getElementById("nome").style.backgroundColor="#FFB3B3";
```

Terminati i controlli sui campi obbligatori verifica che la stringa "errori" sia vuota. Se non è vuota significa che almeno uno dei campi obbligatori non è stato compilato e quindi bisogna avvisare l'utente. Per farlo esegue tre operazioni:

- scrive il contenuto della stringa "errori" nel paragrafo nascosto
- colora il testo di rosso
- mostra il paragrafo nascosto



Se invece la stringa "errori" è vuota significa che tutti i controlli sono andati a buon fine e quindi i dati sono stati inseriti correttamente. In tal caso la funzione nasconde nuovamente il paragrafo con id="risultato" e cancella il form mostrando al suo posto il testo "Grazie".

```
document.getElementById("risultato").style.display = "none";
document.getElementById("modulo").innerHTML = "<h4>Grazie</h4>";
```

Quindi, seppur semplice, questo codice per il controllo di un form presenta tante caratteristiche interessanti. Abbiamo infatti imparato a usare javascript per

- modificare il contenuto HTML di un elemento HTML
- mostrare o nascondere un elemento HTML
- cambiare lo sfondo di un elemento HTML

Le tecniche e il codice usati in questo esempio sono usabili in qualunque contesto per qualunque elemento HTML.

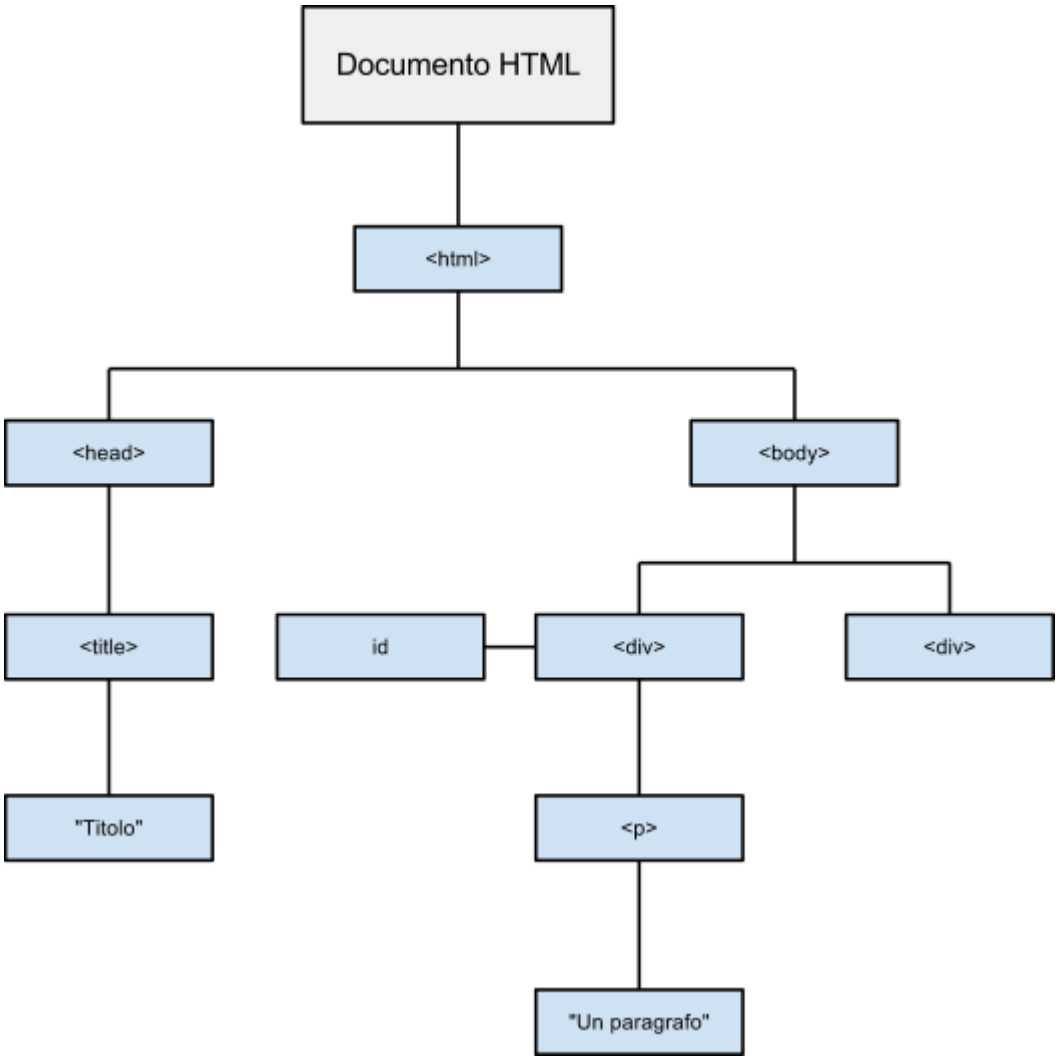
Ora però dobbiamo fare il salto di qualità e per farlo dobbiamo necessariamente affrontare il DOM. Non è infatti possibile usare proficuamente e professionalmente javascript senza avere una buona conoscenza del DOM HTML.
Ne abbiamo già parlato nel capitolo relativo a HTML, ma adesso dobbiamo entrare nei dettagli.

DOM: uno sguardo più approfondito

Riprendiamo, per semplicità, quanto già detto nel capitolo riguardante l'HTML.
DOM = **D**ocument **O**bject **M**odel.
Il DOM è un concetto generale che struttura un generale documento di markup in nodi. Poiché anche HTML è un documento di markup, esiste una versione particolare del DOM chiamato HTML DOM. Esso rappresenta tutto il documento HTML in una struttura ad albero, composta da nodi, in cui ogni elemento HTML è un nodo e ogni nodo è un oggetto. Avendo quindi a disposizione un oggetto per ogni nodo, viene anche definito un modo standard per accedere e modificare dinamicamente gli elementi di un documento HTML tramite le proprietà e i metodi dell'oggetto.
Nel DOM tutto è un nodo.
Per comprendere quest'ultima affermazione possiamo procedere con un semplice esempio. Prendiamo il seguente documento HTML:

```
<html>
    <head>
          <title>Titolo</title>
    </head>
    <body>
          <div id="primo">
                <p>Un paragrafo</p>
          </div>
          <div></div>
    </body>
</html>
```

Questo stesso documento HTML visto come DOM sarà rappresentato nel seguente modo.



In celeste sono rappresentati tutti i nodi DOM del documento HTML.
Già da una semplice osservazione della figura possiamo fare una riflessione: **non sempre un nodo DOM equivale a un elemento HTML**. Basta infatti osservare il nodo id o il nodo "Titolo" che, nella rappresentazione DOM, sono 2 nodi, ma che in HTML non sono elementi (sono rispettivamente un attributo e un testo). Quindi ogni elemento HTML è sicuramente un nodo, ma non è vero il viceversa! Ecco allora che diventa più comprensibile l'affermazione di prima: nel DOM tutto è un nodo.
La rappresentazione inizia con un rettangolo grigio. Anch'esso è un nodo, ma un nodo particolare chiamato **Document** il cui compito è quello di raccogliere le informazioni generali sul documento stesso.
Il nodo <html> (che in questo caso coincide con l'elemento <html>) essendo il padre di tutti i nodi di un documento HTML (quindi senza padre e senza fratelli), viene chiamato **root**. Tra i nodi del DOM ci si sposta tramite parentele: padre, figlio, fratello: il nodo <head> è figlio del nodo <html> ed è fratello del nodo <body> e così via...

Gli oggetti che compongono il DOM sono i seguenti:

- DOM Node
- DOM NodeList
- DOM NamedNodeMap
- DOM Document
- DOM Element
- DOM Attr

Ma a noi non interessa studiare tutto il DOM. Quello che ci serve è il sottoinsieme del DOM che fa riferimento all'HTML, cioè l'**HTML DOM**.
Pertanto gli oggetti a cui faremo riferimento saranno i seguenti:

- DOM Node
- DOM Document
- DOM Element
- DOM Event

Le specifiche attualmente in stato di raccomandazione (DOM level 3), alle quali ci rifacciamo in questa sede, si possono leggere qui:
http://www.w3.org/TR/DOM-Level-3-Core/
Le nuove specifiche in fase di definizione (DOM level 4) si possono invece leggere qui:
http://www.w3.org/TR/dom/

DOM Node

Il concetto di nodo nel DOM è molto importante. Ripetiamo ancora una volta che all'interno della rappresentazione DOM, tutto è un nodo. Ossia:

- L'intero documento HTML è un nodo (l'oggetto Document che vedremo in seguito)
- Ogni elemento HTML è un nodo (oggetto Element)
- Il testo contenuto in un elemento HTML è un nodo
- Ogni attributo è un nodo
- Anche i commenti sono nodi

Questo concetto deve essere chiaro. Infatti uno degli errori più comuni durante la navigazione all'interno del DOM è quella di vedere, ad esempio, il testo "Titolo" come se fosse il valore dell'elemento <title>. Non è così! "Titolo" è un nodo, non è il valore del nodo <title>. Teniamo ben chiaro questo concetto.

Per recuperare o modificare il contenuto di un nodo possiamo usare **la proprietà innerHTML** (una delle proprietà di un nodo che abbiamo già usato più volte e che approfondiremo a breve). Ad esempio, con riferimento al documento HTML di inizio paragrafo, il codice

var contenuto = document.getElementById("primo").innerHTML;

assegnerà alla variabile "contenuto" il valore " Un paragrafo ".
Allo stesso modo, scrivere il codice
document.getElementById("primo").innerHTML="<h3>Cambio contenuto</h3>";

significa modificare in tempo reale il contenuto del div con id="primo" con quello alla destra del simbolo di uguaglianza.
Ogni nodo nel modello DOM ha proprietà e metodi. Qui di seguito elenchiamo i più usati. come sempre per un elenco esaustivo è opportuno fare riferimento alla documentazione ufficiale.

Proprietà

Proprietà	**Significato**
innerHTML	Restituisce tutto ciò che è contenuto nel nodo
nodeName	Restituisce il nome del nodo
nodeValue	Restituisce il valore del nodo
parentNode	Restituisce il nodo padre del nodo
childNodes	Restituisce l'elenco di nodi figli diretti del nodo
attributes	Restituisce l'elenco degli attributi dl nodo

Metodi

Metodo	**Significato**
getElementById(id)	Restituisce l'elemento con l'id indicato tra parentesi
getElementsByTagName(name)	Restituisce l'elenco di tutti gli elementi del tipo indicato tra parentesi e contenuti nel nodo
appendChild(node)	Inserisce il figlio indicato tra partentesi come figlio del nodo

removeChild(node)	Elimina il nodo figlio indicato tra parentesi

DOM Document

Come abbiamo già detto, l'oggetto Document è un nodo, ma un nodo particolare. Esso infatti ha come scopo quello di contenere le informazioni generali sul documento. Vediamone proprietà e metodi più utilizzati.

Proprietà

Proprietà	**Significato**
doctype	Restituisce il doctype associato al doumento HTML
documentElement	Restituisce l'elemento <html>
inputEncoding	Restituisce l'encoding dei caratteri usata nel documento
xmlEncoding	Restituisce il valore dell'attributo
xmlStandalone	Restituisce il valore dell'attributo
xmlVersion	Restituisce il valore dell'attributo

Metodi

Metodo	**Significato**
createAttribute()	Crea un nodo di tipo Attributo
createComment()	Crea un nodo di tipo Commento
createElement()	Crea un nodo Element
createTextNode()	Crea un nodo di tipo Text
getElementById()	Restituisce un elemento identificato dall'ID indicato tra parentesi
getElementsByTagName()	Restituisce una NodeList contenente tutti i nodi con il nome indicato tra parentesi
renameNode()	Rinomina il nodo indicato

DOM Element

Questo oggetto rappresenta un elemento HTML ed è anch'esso un nodo.

Ricordiamo che un elemento HTML indica tutto ciò che si trova tra il tag di apertura e il tag di chiusura. Quindi, ad esempio, l'elemento <html> indica tutto il documento HTML. L'unica proprietà degna di nota è la proprietà tagName che restituisce il nome dell'elemento rappresentato dall'oggetto. Per i metodi invece ecco i più usati.

Metodi

Metodo	**Significato**
getAttribute()	Restituisce il valore dell'attributo indicato tra parentesi
getAttributeNode()	Restituisce il nodo che rappresneta l'attributo indicato tra parentesi
hasAttribute()	Restituisce true se l'elemento presenta l'attributo indicato tra parentesi
removeAttribute()	Rimuove l'attributo indicato tra parentesi
removeAttributeNode()	Rimuovo il nodo indicato tra parentesi
setAttribute()	Setta o sovrascrive il valore dell'attributo specificato tra parentesi
setAttributeNode()	Setta o sovrascrive il nodo specificato tra parentesi

Lo abbiamo già accennato, ma qui lo ripetiamo: facciamo attenzione a **non confondere un Node con un Element**!
Un Element si riferisce ad un elemento HTML e, come abbiamo detto a suo tempo, un elemento HTML rappresenta tutto ciò che si trova tra il tag di apertura e il tag di chiusura.
Esempio:

```
<div id="unDiv">
    <p>Un paragrafo</p>
</div>
```

Parlare dell'elemento #unDiv significa indicare tutto quanto sopra scritto.
Un oggetto Element può anche coincidere con un Node, ma un Node può anche rappresentare il testo contenuto nel tag <p>, o il semplice attributo id del .
Pertanto bisogna porre attenzione a distinguere concettualmente i due oggetti.

DOM Event

Questo oggetto è un oggetto delicato che va maneggiato con cura, ma è delicato perché è molto potente. Rappresenta un evento. Un evento è rappresentato da una qualsiasi interazione dell'utente con la pagina web. Esempi di evento sono:

- click del mouse
- pressione di un tasto sulla tastiera

Proprietà

Proprietà	**Significato**
cancelable	Restituisce true o false a seconda che l'evento sia o meno cancellabile
currentTarget	Restituisce l'elemento sul quale si è fisicamente causato l'evento
target	Restituisce l'elemento dal quale è partito l'evento
type	Restituisce il tipo di evento che è stato scatenato

Metodi

Metodo	**Significato**
initEvent ("eventType", bubblesFlag, cancelableFlag)	Inizializza un evento. Il primo parametro indica il tipo di evento (click, muosedown, keypress, ecc.); il secondo è di tipo booleano (true | false) e indica se l'evento ha come comportamento predefinito quello della propagazione; infine il terzo è di tipo booleano (true | false) e indica se l'evento è cancellabile o meno
preventDefault()	Impedisce la naturale prosecuzione dell'evento (se possibile)
stopPropagation()	Ferma la propagazione dell'evento (se possibile)

Quasi tutti gli elementi HTML hanno già di per sé associati alcuni eventi. Nel paragrafo dedicato alle funzioni di javascript abbiamo infatti incontrato una tabella contenente alcuni attributi (onclick, onkeypress, ecc.) che possiamo usare per gestire gli eventi che si verificano sugli elementi HTML.

JavaScript e il DOM

Ora che abbiamo approfondito il concetto del DOM HTML possiamo approfondire anche l'uso di javascript.

Riprendiamo quindi il form di prima e facciamo un passo in avanti. Usiamo gli eventi per impedire all'utente di inserire caratteri non congrui.

Non esistono né nomi né cognomi con meno di 2 caratteri. Inoltre in un nome e in un cognome non ci si aspetta di trovare numeri! Infine nome e cognome non possono di certo né iniziare né terminare con uno spazio. Implementiamo allora questi controlli tramite javascript.

Procederemo nei controlli nell'ordine che segue:

- niente numeri
- niente spazi
- minimo due caratteri

```
<html>
    <head>
        <title>JavaScript - Controllo Form</title>
        <script type="text/javascript">
            function controlla(ev) {
                if (ev.which==32 && ev.target.value.length==0){
                      ev.preventDefault();
                }
                if ((ev.which<65 || ev.which>90) &&
                            ev.which!=8 && ev.which!=9
                            && ev.which!=32 &&
                            ev.which!=46 &&
                            (ev.which<37 || ev.which>40) ) {
                      ev.preventDefault();
                 }
            }
        </script>
    </head>
    <body>
        <p id="risultato" style="display:none"></p>
        <div id="modulo">
            <form name="input" method="post">
                <label for="nome">Nome<br /></label>
                <input id="nome" type="text" name="nome"
                      onkeydown="controlla(event);" />
                <br />
                <label for="cognome">Cognome<br /></label>
                <input id="cognome" type="text" name="cognome"
                      onkeydown="controlla(event);/>
```

```
                <br />
                <label for="indirizzo">Indirizzo<br /></label>
                <input id="indirizzo"
                      type="text" name="indirizzo"
                      onkeydown="controlla(event);/>
                <br />
                <label for="cellulare">Cellulare<br /></label>
                <input id="cellulare" type="text"
                      name="cellulare" />
                <br />
                <input type="button" value="Invia i dati"
                      onclick="" />
            </form>
        </div>
    </body>
</html>
```

Il codice adesso è un po' più complicato, ma abbiamo tutti i concetti necessari per comprenderlo. Innanzitutto non eseguiamo più alcun controllo al click del mouse sul bottone. Abbiamo spostato tutti i controlli direttamente nel momento in cui l'utente li scrive. Per fare questo abbiamo usare l'attributo onkeydown dell'elemento <input> per lanciare la funzione controlla(event). Quindi prima di rilasciare un qualunque tasto della tastiera all'interno della casella di testo, verrà richiamata la funzione controlla(event).

```
function controlla(ev) {
    if (ev.which==32 && ev.target.value.length==0) {
          ev.preventDefault();
    }
    if ((ev.which<65 || ev.which>90) &&
                ev.which!=8 && ev.which!=9 && ev.which!=32 &&
                ev.which!=46 && (ev.which<37 || ev.which>40) ) {
          ev.preventDefault();
    }
}
```

Possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/js/js_11.html

Il **metodo ev.which** restituisce il codice del carattere che l'utente ha inserito sulla tastiera. Il codice 32 è il codice dello spazio. Quindi se l'utente inserisce uno spazio come primo carattere (ev.which==32 && ev.target.value.length==0) il sistema previene la scrittura nella casella di testo tramite il metodo preventDefault(). In altre parole l'uente non può inserire uno spazio come primo carattere.
Lo stesso tipo di atto preventivo viene applicato se l'utente preme tasti non consentiti (numeri, caratteri speciali, ecc.).
La funzione che abbiamo scritto quindi impedisce all'utente di inserire numeri e caratteri speciali nelle casella "Nome", "Cognome" e "Indirizzo". Quasi certamente le restrizioni imposte sono eccessive, soprattutto per il campo "Indirizzo", ma sono un primo passo. Possiamo intervenire sui singoli caratteri.
Non è stata invece imposta alcuna restrizione per i caratteri inseribili nel campo "Cellulare". Potrebbe essere un buon esercizio creare una funzione per il controllo del campo "Cellulare".
A questo punto ci dobbiamo porre due domande:

- dove li troviamo i codici dei caratteri?
- da dove viene fuori la proprietà which?

La prima domanda trova una duplice risposta.
La prima risposta ce la fornisce wikipedia qui:
http://it.wikipedia.org/wiki/ASCII

La seconda risposta è che possiamo rintracciare i codici dei caratteri da soli. Ad esempio inserendo un alert() all'inizio della funzione controlla(ev) che ci stampi il codice del carattere premuto dell'utente (ev.which).

Per quanto riguarda **la proprietà which**, in realtà non l'abbiamo elencata tra quelle disponibili dell'oggetto Event. Questa proprietà viene aggiunta dal browser e infatti fino alla versione 8 di IE bisognava usare un certo codice per IE e un altro codice per tutti gli altri browser. In IE 8 (e in tutte le versioni precedenti) infatti la proprietà which non esisteva. Esisteva l'equivalente proprietà keyCode dell'oggetto Window. Ecco il codice da scrivere in modo che funzioni anche per le versioni precedenti a IE 9:

```
if (window.event) {
    ev.keyCode;
} else if (e.which) {
    ev.which;
}
```

In realtà, in questo tipo di situazioni (controllo caratteri inseriti) è assolutamente consigliato (nonché professionale) utilizzare le **regular expression**. Non solo perché rendono il codice più pulito (senza andare a verificare ogni singolo carattere), ma anche perché rendono il codice molto più breve e più portabile (funziona su tutti i browser!). Purtroppo per questioni di spazio in questo testo non abbiamo modo di approfondire il concetto. Tuttavia vogliamo mostrare come cambierebbe il codice della funzione controlla(ev) utilizzando una regular expression.
La regular expression per indicare tutti i numeri è la seguente \d e questo è il codice per impedire di scrivere numeri in una casella di testo (funzionante su tutti i browser, anche i vecchi IE):

```
<script type="text/javascript">
    function controlla(ev) {
          var codice;
          var carattere;
          var regexp;
          if (window.event) {
                codice= ev.keyCode;
          } else if (e.which) {
                codice= ev.which;
          }
          carattere=String.fromCharCode(codice);
          regexp=/\d/;
          if (regexp.test(carattere)) {
                ev.preventDefault();
          }
    }
</script>
```

che possiamo osservare in azione qui:
http://www.alessandrostella.it/lato_client/js/js_12.html

Molto più pulito e professionale, no?
Pertanto l'invito è: studiamo le regular expression! Online si trovano validissimi tutorial sull'argomento.

12
JQuery, ma non solo

JQuery è una libreria JavaScript.
Lo scopo della libreria è quello di rendere l'uso di javascript il più semplice possibile. La grandissima diffusione di jQuery sottolinea come lo scopo sia stato pienamente raggiunto. Il motto di jQuery è "write less, do more", cioè scrivere di meno, fare di più. Un bel motto!
Bello e reale. Infatti per nascondere un elemento HTML con id="primo" in javascript bisogna scrivere il seguente codice:

```
document.getElementById("primo").style.display="none";
```

mentre in jQuery:

```
$("#primo").hide();
```

Dunque, per ottenere lo stesso risultato (nascondere l'elemento HTML con id="primo") siamo passati dai 53 caratteri di JavaScript ai 18 di jQuery e per di più senza perdere in leggibilità! Non c'è niente da dire: abbiamo scritto molto meno e abbiamo ottenuto lo stesso risultato! Anche scrivendo una sola riga di codice, l'utilità di jQuery appare evidente. Tuttavia ha senso usare jQuery nel caso di utilizzo intensivo di JavaScript. Se invece il codice che dobbiamo scrivere è tutto in poche funzioni, beh, allora non ha molto senso usare jQuery. Inoltre l'uso di jQuery "impigrisce" nell'uso di JavaScript, cioè

utilizzando jQuery (o altre librerie simili) si rischia di dimenticare la sintassi di JavaScript, sebbene jQuery sia scritto interamente in JavaScript.
JQuery è un valido aiuto quando il codice da scrivere è tanto e complesso ed è molto spesso richiesto dalla aziende... è quindi opportuno saperlo usare.
Per usare jQuery è necessario aggiungere la seguente riga di codice all'interno del tag <head> del documento HTML in cui vogliamo utilizzarla.

```
<script type="text/javascript" src="jquery.js"></script>
```

Ovviamente il valore dell'attributo src andrà modificato a seconda della posizione della libreria. Dal seguente link:
http://docs.jquery.com/Downloading_jQuery

possiamo sia scaricare sul nostro PC la libreria (scegliamo la versione minified) sia puntare ad una versione poggiata su un server Google (CDN Hosted jQuery).
Volendo, possiamo eseguire gli esempi direttamente online su un ottimo editor online: http://jsfiddle.net. In tal caso, nel tag <script>, puntiamo alla libreria JQuery poggiata sui server di Google:
https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.min.js
In altre parole:

```
<script type="text/javascript"
    src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jque
ry.min.js">
</script>
```

Ecco le principali caratteristiche della libreria:

- selezione e manipolazione degli elementi HTML
- selezione e manipolazione dei fogli di stile (CSS)
- gestione degli eventi HTML
- effetti ed animazioni JavaScript
- navigazione e modifica del DOM HTML
- AJAX

Per poter intraprendere con profitto lo studio di jQuery è necessario avere già le seguenti competenze:

- HTML

- CSS
- JavaScript

Nei capitoli precedenti abbiamo trattato in modo sufficiente questi argomenti.

La sintassi

La logica di funzionamento di jQuery è molto semplice ed è basata su 2 passaggi:

1. selezionare un elemento HTML
2. associargli un'azione

La sintassi di jQuery rispecchia tale semplicità:

$(selector).action()

oppure

jq(selector).action()

Tale sintassi comprende:

- un simbolo di dollaro per indicare che quanto scritto deve essere interpretato usando jQuery (tuttavia poiché anche altre librerie javascript usano il simbolo del dollaro, per evitare conflitti, jQuery consente di sostituire il simbolo del dollaro con la sequenza "jq")
- un selettore (selector) che serve per individuare uno o più elementi HTML
- un'azione (.action()) da eseguire sugli elementi selezionati al punto precedente

Esempi:

```
$("div").hide() - nasconde tutti i <div> della pagina web
$("div.primo").hide() - nasconde i <div> che hanno attributo
class="primo"
$("#test").hide() - nasconde l'elemento con id="test"
```

Prima di eseguire codice jQuery è opportuno assicurarci che la pagina web sia stata completamente caricata dal browser. Ciò è necessario perché se cercassimo di associare una qualunque azione ad un elemento non ancora caricato nella pagina otterremmo un errore. Per fare questo **è necessario inserire il codice jQuery nella funzione ready()**

```
$(document).ready()
```

Tutto il codice jQuery deve essere richiamato in tale funzione.

Esempio:

```
<html>
    <head>
          <script type="text/javascript"
                src="jquery.js">
          </script>
          <script type="text/javascript">
                $(document).ready(function(){
                      $("button").click(function(){
                            $("p").css("color","red");
                      });
                });
          </script>
    </head>
    <body><!--codice HTML--></body>
</html>
```

Questa funzione ha come scopo evidente quello di eseguire il codice jQuery solo a documento HTML completamente caricato (ready).

Il modo di ragionare sintattico di JQuery può apparire incomprensibile. Anzi, la prima volta che si ha a che fare con jQuery può capitare di pensare ad un errore di battitura: funzioni definite dentro funzioni e così via. Non c'è nessun errore. Tra le parentesi tonde della funzione ready() c'è davvero definita un'intera funzione senza nome. Non solo. Nel metodo click() ce n'è un'altra. Niente paura. Ci si abitua abbastanza velocemente.

Possiamo vedere il codice in azione qui:

http://www.alessandrostella.it/lato_client/jq/jq_01.html

Selezionare un elemento HTML

Abbiamo visto che la sintassi di jQuery è abbastanza semplice: dollaro, selettore, azione. Unica attenzione da porre è quella di inserire il codice jQuery nella funzione ready(). Lo scopo è quello di selezionare un elemento HTML e poi associargli un'azione. Per prima cosa quindi dobbiamo imparare ad effettuare la selezione di un elemento HTML.

La seguente tabella ci mostra il selettore da usare per ottenere una determinata selezione.

Selettore	Esempio	Significato
*	$("*")	Seleziona tuti gli elementi
#id	$("#primo")	Seleziona l'elemento con id="primo"
.class	$(".dispari")	Seleziona tutti gli elementi con attributo class="dispari"
element	$("div")	Seleziona tutti gli elementi
:first	$("p:first")	Seleziona solo il primo paragrafo
:last	$("p:last")	Seleziona solo l'ultimo paragrafo
:even	$("li:even")	Seleziona solo gli elementi dispari
:odd	$("li:odd")	Seleziona solo gli elementi pari
:header	$(":header")	Seleziona tutti gli elementi <hn>, cioè <h1>, <h2>, ..., <h6>
:visible	$("div:visible")	Seleziona tutti i visibili
[attribute]	$("[href]")	Seleziona tutti gli elementi che hanno un attributo href
[attribute=value]	$("[href='index.html']")	Seleziona tutti gli elementi che hanno href="index.html"
[attribute!=value]	$("[href!='index.html']")	Seleziona tutti gli elementi che hanno href diverso da "index.html"
[attribute$=value]	$("[href$='.html']")	Seleziona tutti gli elementi che hanno un attributo href che termina con i caratteri ".html"

Per gli **elementi di tipo input** valgono i seguenti selettori specifici

:input	$(":input")	Seleziona tutti gli elementi di tipo input che non sono SOLO quelli del tag <input>. A questa categoria appartiene, per esempio, anche il tag <button>
:text	$(":text")	Seleziona tutti gli elementi input con type="text"
:radio	$(":radio")	Seleziona tutti gli elementi input con type="radio"
:checkbox	$(":checkbox")	Seleziona tutti gli elementi input con type="checkbox"
:submit	$(":submit")	Seleziona tutti gli elementi input con type="submit"
:reset	$(":reset")	Seleziona tutti gli elementi input con type="reset"
:button	$(":button")	Seleziona tutti gli elementi input con type="button"

:image	$(":image")	Seleziona tutti gli elementi input con type=”image”
:enabled	$(":enabled")	Seleziona tutti gli elementi input abilitati
:disabled	$(":disabled")	Seleziona tutti gli elementi input disabilitati
:selected	$(":selected")	Seleziona tutti gli elementi input selezionati
:checked	$(":checked")	Seleziona tutti gli elementi input checked

E’ molto importante osservare che i selettori sono gerarchici. Possiamo pertanto scrivere $(“ul.elenco li.dispari”);

per selezionare gli elementi <li> con class=”dispari”, figli di elementi <ul> con class=”elenco”. E’ altrettanto importante fare un po’ di pratica con questi selettori. Facciamone un po’ insieme.

```
<html>
    <head>
    <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.
min.js">
    </script>
    <script type="text/javascript">
          $(document).ready(function(){
                $("#titolo").css("background-color","red");
          });
    </script>
    </head>
          <body>
                <div id="box" style="width:250px;">
                <div id="titolo"
                      style="width:100%;background:#ccc">
                      <p>Titolo</p>
                </div>
                <div id="testo" style="background:#aaa">
                      <p>Fai una scelta:
                            <ul id="scelta">
                                  <li><a href="#">HTML</a></li>
                                  <li><a href="#">CSS</a></li>
                                  <li><a href="#">
```

```
                                        JavaScript</a>
                                    </li>
                                </ul>
                            </p>
                        </div>
                    </div>
    </body>
</html>
```

Il codice che ci interessa è

```
<script type="text/javascript">
    $(document).ready(function(){
          $("#titolo").css("background-color","red");
    });
</script>
```

Questo codice cerca l'elemento HTML con id="titolo" e ne setta a rosso il colore di sfondo.
Possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/jq/jq_02.html

Possiamo giocare su questo codice per fare i nostri esperimenti. Ad esempio:

```
<script type="text/javascript">
    $(document).ready(function(){
          $("[href]").css("background-color","red");
    });
</script>
```

Cerca tutti gli elementi con attributo href e setta lo sfondo a rosso.
Eccolo in azione:
http://www.alessandrostella.it/lato_client/jq/jq_03.html

Ancora:

```
<script type="text/javascript">
    $(document).ready(function(){
```

```
            $("ul li:last").css("background-color","red");
    });
</script>
```

cerca l'ultimo elemento <li> di tutti gli elementi <ul> e ne setta lo sfondo a rosso. E così via. Proviamo e riproviamo... Prima di proseguire, dobbiamo entrare in confidenza con la selezione degli elementi HTML tramite jQuery.

Eseguire un'azione

Nel precedente paragrafo abbiamo imparato come usare jQuery per selezionare un elemento HTML. In questo paragrafo impareremo come eseguire un'azione su tale elemento.

Su ogni elemento possiamo eseguire diverse azioni: modificare la reazione dell'elemento agli eventi, applicare un effetto grafico, mostrare un messaggio di avviso, cambiare il colore di sfondo... ecc. Per ognuna delle azioni che possiamo eseguire, jQuery prevede i relativi metodi che possiamo dividere in 4 categorie:

- metodi per gestire gli eventi
- metodi per gestire gli effetti
- metodi per modificare il DOM
- metodi per modificare i CSS

I metodi per gestire gli eventi

Come sappiamo, per evento si intende una qualunque interazione dell'utente con la nostra pagina web, quindi click del mouse, inserimento dati, ecc. Per gestire gli eventi di un determinato elemento HTML, jQuery usa alcuni metodi di cui i più usati sono qui di seguito elencati.

Metodo	**Descrizione**
change(function)	Associa un gestore all'evento onchange dell'elemento
click(function)	Associa un gestore all'evento onclick dell'elemento
dblclick(function)	Associa un gestore all'evento ondblclick dell'elemento
focus(function)	Associa un gestore al relativo evento
focusin(function)	Associa un gestore al relativo evento
focusout(function)	Associa un gestore al relativo evento

hover(inFunction,outFunction)	Associa un gestore con una o due funzioni al relativo evento
keydown(function)	Associa un gestore al relativo evento
keypress(function)	Associa un gestore al relativo evento
keyup(function)	Associa un gestore al relativo evento
mousedown(function)	Associa un gestore al relativo evento
mouseenter(function)	Associa un gestore al relativo evento
mouseleave(function)	Associa un gestore al relativo evento
mousemove(function)	Associa un gestore al relativo evento
mouseout(function)	Associa un gestore al relativo evento
mouseover(function)	Associa un gestore al relativo evento
mouseup(function)	Associa un gestore al relativo evento
ready(function)	Associa un gestore all'evento di completato caricamento di una pagina web
resize(function)	Associa un gestore al ridimensionamento della finestra del browser in cui è contenuta la pagina web
scroll(function)	Associa un gestore al relativo evento
select(function)	Associa un gestore all'evento di selezione di un elenco
submit(function)	Associa un gestore al relativo evento

Non ha molto senso elencare tutti i metodi disponibili; quelli elencati sono i più comuni. Come si può osservare, quasi tutti i metodi si aspettano la definizione di una funzione che dovrà essere eseguita allo scatenarsi dell'evento. Esempio

```
$("button").click (function() {
    $("img").hide()
})
```

che si può scrivere in modo più compatto (ma meno comprensibile) così:

```
$("button").click(function(){$("img").hide()})
```

Con questo codice abbiamo assegnato un gestore all'evento click del mouse su tutti gli elementi di tipo <button>. Quando si fa click con il mouse su un elemento di tipo

<button> viene lanciata la funzione definita all'interno del metodo click(). In questo caso la funzione si occupa di nascondere tutti gli elementi di tipo <img> ($("img").hide()). Tutto qui. La logica è sempre la stessa. Ovviamente vale quanto abbiamo già detto: non è saggio associare una funzione ad un elemento HTML senza avere la certezza che tale elemento sia stato caricato nella pagina. Quindi, questo tipo di istruzioni, vanno inserite all'nterno della funzione $(document).ready().
Facciamo qualche esempio per prendere confidenza.
Ammettiamo di avere una casella di testo e una checkbox e ammettiamo di voler sapere cosa sta facendo l'utente su tali oggetti. Ammettiamo quindi di avere il seguente codice HTML:

```
<body>
    <input type="textarea" id="testo" /><br>
    <input type="checkbox" />
    <p>Ecco cosa stai facendo:</p>
    <h3>Nessuna azione compiuta</h3>
</body>
```

in cui nell'elemento <h3> comunicheremo all'utente cosa sta facendo con mouse e tastiera.
Per far questo dobbiamo associare alcuni eventi sia alla casella di testo, sia alla checkbox. In particolare, possiamo scrivere il seguente codice:

```
<script type="text/javascript">
    $(document).ready(function(){
          $("#testo").click(function(){
                $("h3").html("Hai dato il fuoco alla casella
                      di testo");
          });
          $("#testo").mouseover(function(){
                $("h3").html("Stai passando il mouse sulla
                      casella di testo");
          });
          $("#testo").mouseout(function(){
                $("h3").html("Il mouse ha abbandonato l'area
                      occupata dalla casella di testo");
          });
          $("#testo").keydown(function(){
                $("h3").html("Stai scrivendo o cancellando
```

```
                        qualcosa nella casella di testo");
            });
            $(":checkbox").click(function(){
                  $("h3").html("Hai cliccato su una checkbox");
            });
      });
  </script>
```

Stiamo quindi associando alla casella di testo #testo ($("#testo")) i seguenti eventi:

- click (che viene scatenato quando l'utente clicca con il mouse nella casella di testo)
- mouseover (che viene scatenato quando l'utente passa con il puntatore del mouse sulla casella di testo)
- mouseout (l'utente esce con il puntatore del mouse dalla casella di testo)
- keydown (l'utente ha premuto un qualsiasi tasto della testiera)

Stiamo inoltre associando a tutte le checkbox ($(":checkbox")) la funzione click().
Il risultato è visibile qui:
http://www.alessandrostella.it/lato_client/jq/jq_04.html

I metodi per aggiungere un effetto

JQuery consente di associare agli elementi HTML effetti grafici in modo abbastanza semplice. Anche in questo caso per ottenere l'obiettivo fa uso di alcuni metodi che si occupano di associare l'effetto voluto ad un determinato elemento.
In questo caso per indicare i metodi e il rispettivo significato non useremo la solita tabellina con le due colonne. I metodi infatti hanno un impatto puramente grafico ed è quindi più opportuno procedere per esempi in modo da vedere immediatamente il risultato del metodo.
Andiamo quindi sul nostro editor online:
http://jsfiddle.net
e proviamo i metodi seguendo gli esempi che troveremo qui di seguito.

Metodo animate()

Lo scopo del metodo è di passare da una dimensione ad un'altra in modo animato.
La sintassi completa del metodo è la seguente:
(selector).animate({styles},duration,easing,callback)

ma l'unico parametro obbligatorio è il primo, styles. Ecco un esempio di utilizzo:

```
<html>
    <head>
          <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.
min.js"></script>
          <script type="text/javascript">
                $(document).ready(function() {
                      $("#allarga").click(function(){
                            $("#box").animate({width:"350px",
                                  height:"350px"});
                      });
                      $("#stringi").click(function(){
                            $("#box").animate({width:"100px",
                                  height:"100px"});
                      });
                });
          </script>
    </head>
    <body>
          <div id="box" style="background:yellow;height:100px;
                width:100px;margin:6px;">
          </div>
          <button id="allarga">Allarga</button>
          <button id="stringi">Stringi</button>
    </body>
</html>
```

Quello che otterremo è l'espansione di un quadretto colorato in modo molto simpatico. Il metodo non funziona su tutti i selettori css, inoltre funziona solo per valori numerici, non sulle percentuali.

Possiamo osservare il codice in azione qui:

http://www.alessandrostella.it/lato_client/jq/jq_05.html

Metodi fadeIn() e fadeOut()

La sintassi completa dei 2 metodi è simile:

$(selector).fadeIn(duration,easing,callback)

$(selector).fadeOut(duration,easing,callback)

I parametri duration, easing e callback sono opzionali. Pertanto l'uso più semplice di questi metodi diventa il seguente:
$(selector).fadeIn()
$(selector).fadeOut()

Il parametro **duration** indica il tempo per il quale deve durare l'effetto.
Il parametro **easing** è stato aggiunto dalla versione 1.4.3 e serve per modificare il comportamento previsto dalle animazioni predefinite. Troppo complesso per il momento.
Il parametro **callback** indica l'eventuale funzione da eseguire al termine dell'animazione.
Questi parametri hanno lo stesso significato in tutti i metodi che vedremo.

Facciamo subito un esempio:

```
<html>
    <head>
          <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.
min.js"></script>
          <script type="text/javascript">
                $(document).ready(function(){
                      $(".fdout").click(function(){
                            $("div").fadeOut()
                      });
                      $(".fdin").click(function(){
                            $("div").fadeIn();
                      });
                });
          </script>
    </head>
    <body>
          <div style="height:150px;
                width:150px;background:black"></div>
          <button class="fdout">Fade out</button>
          <button class="fdin">Fade in</button>
    </body>
</html>
```

Questo esempio mostra/nasconde un quadretto di colore nero con un effetto di dissolvenza (fade).
Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/jq/jq_06.html

Metodi hide(), show() e toggle()

Questi 3 metodi si occupano di nascondere o mostrare un elemento HTML. Anche in questo caso le sintassi sono identiche:
$(selector).hide(duration, easing, callback)
$(selector).show(duration, easing, callback)
$(selector).toggle(duration, easing, callback)

Anche in questo caso i parametri duration, easing e callback sono opzionali. Pertanto l'uso più semplice di questi metodi diventa il seguente:
$(selector).hide()
$(selector).show()
$(selector).toggle()

Vediamo un esempio con il metodo toggle().

```
<html>
    <head>
          <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.
min.js"></script>
          <script type="text/javascript">
                $(document).ready(function(){
                      $("button").click(function(){
                            $("div").toggle();
                      });
                });
          </script>
    </head>
    <body>
          <div style="height:150px;
                width:150px;background:black"></div>
          <button>Mostra/Nascondi</button>
    </body>
```

```
</html>
```

Questo codice ci consente, con un solo bottone, di nascondere o mostrare il solito quadretto nero. Ora proviamo a fare un passo avanti: vediamo un uso un pochino più complesso del metodo toggle().

Scriviamo il seguente codice HTML:

```
<body>
    <div style="float:left; margin-right:10px;">
          <div id="dv1" style="height:150px;
                width:150px;background:black"></div>
          <button id="bt1">Mostra/Nascondi</button>
    </div>
    <div style="float:left; margin-right:10px;">
          <div id="dv2" style="height:150px;
                width:150px;background:black"></div>
          <button id="bt2">Mostra/Nascondi</button>
    </div>
    <div style="float:left; margin-right:10px;">
          <div id="dv3" style="height:150px;
                width:150px;background:black"></div>
          <button id="bt3">Mostra/Nascondi</button>
    </div>
    <div style="float:left">
          <div id="dv4" style="height:150px;
                width:150px;background:black"></div>
          <button id="bt4">Mostra/Nascondi</button>
    </div><br>
    <h3 style="display:none;clear:both;
          margin-top:200px;">Operazione terminata!</h3>
</body>
```

E associamogli il seguente codice jquery.

```
<script type="text/javascript">
    $(document).ready(function(){
          $("#bt1").click(function(){
                $("#dv1").toggle();
```

```
            });
            $("#bt2").click(function(){
                  $("#dv2").toggle("fast");
            });
            $("#bt3").click(function(){
                  $("#dv3").toggle("slow");
            });
            $("#bt4").click(function(){
                  $("#dv4").toggle(2000, function() {
                        $("h3").toggle();
                  });
            });
      });
</script>
```

Possiamo osservare il risultato qui:
http://www.alessandrostella.it/lato_client/jq/jq_07.html

Metodi slideDown(), slideUp() e slideToggle()

Ormai sia la sintassi, sia i parametri dovrebbero esserci familiari.

$(selector).slideDown(duration,easing,callback)

$(selector).slideUp(duration,easing,callback)

$(selector).slideToggle(duration,easing,callback)

Anche in questo caso i parametri sono opzionali.

Ed ecco il solito esempio da provare nel nostro editor online.

```
<html>
    <head>
          <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.
min.js"></script>
          <script type="text/javascript">
                $(document).ready(function(){
                      $("button").click(function(){
                            $("div").slideToggle();
                      });
```

```
                });
            </script>
      </head>
      <body>
            <div style="height:150px;
                  width:150px;background:black"></div>
            <button>Mostra/Nascondi</button>
      </body>
  </html>
```

I metodi non finiscono qui. Ce ne sono altri, ma quelli indicati sono i più usati. Per maggiori dettagli e approfondimenti è opportuno visitare il sito ufficiale di jQuery: http://docs.jquery.com/Main_Page

I metodi per modificare il DOM

I metodi di questa categoria ci consentono di manipolare il DOM HTML, ossia aggiungere/modificare/cancellare elementi HTML. Vediamone alcuni.

Metodo	**Significato**
after()	Inserisce semplice testo o codice HTML dopo il tag di chiusura dell'elemento selezionato
append()	Inserisce semplice testo o codice HTML dopo l'elemento selezionato, ma nello stesso elemento
attr()	Setta o restituisce un attributo
before()	Inserisce semplice testo o codice HTML prima del tag di apertura dell'elemento selezionato
html()	Setta o restituisce il contenuto dell'elemento selezionato
insertAfter()	Inserisce codice HTML o elementi dopo l'elemento selezionato
insertBefore()	Inserisce codice HTML o elementi prima dell'elemento selezionato
remove()	Rimuove gli elementi selezionati
removeAttr()	Rimuove gli attributi selezionati
removeClass()	Rimuove la classe selezionata

Ricordiamo che nella tabella non è riportato un elenco esaustivo di tutti i metodi.
Ecco un esempio di utilizzo del metodo insertBefore().

```
<html>
    <head>
          <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.
min.js"></script>
          <script type="text/javascript">
                $(document).ready(function(){
                      $("button").click(function(){
                            $("<h3>Heading aggiunto!</h3>").
                                  insertBefore("p");
                      });
                });
          </script>
    </head>
    <body>
          <p>Ad ogni click sul bottone in basso, qui sopra
                varrà aggiunto un h3</p>
          <p>Lo stesso avviene anche qui sopra</p>
          <button>Inserisci h3 subito prima di ogni
                paragrafo</button>
    </body>
</html>
```

Cliccando sul bottone aggiungiamo un <h3> subito prima di tutti i paragrafi presenti nel documento. Possiamo osservare il codice in azione qui:
http://www.alessandrostella.it/lato_client/jq/jq_08.html

I metodi per modificare i CSS

I metodi di questa categoria servono per modificare in tempo reale i fogli di stile della pagina web. Vediamo i più comuni.

Metodo	**Significato**
addClass()	Aggiunge una a più classi all'elemento
css()	Setta o restituisce le proprietà css associate all'elemento
height()	Setta o restituisce l'altezza dell'elemento
width()	Setta o restituisce la larghezza dell'elemento

Esempio di utilizzo del metodo css().

```
<html>
    <head>
          <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.
min.js"></script>
          <script type="text/javascript">
                $(document).ready(function(){
                      $("button").click(function(){
                            $("p").css("color","red");
                      });
                });
          </script>
    </head>
    <body>
          <p>Questo è un paragrafo.</p>
          <p>E questo è un fantastico paragrafo, come quello
                di sopra :-D</p>
          <button>Setta il colore per i paragrafi</button>
    </body>
</html>
```

Questo codice seleziona tutti i paragrafi della pagina e ne modifica in rosso il colore del testo. Possiamo vederlo in azione qui:
http://www.alessandrostella.it/lato_client/jq/jq_09.html

Bene.
Ora dovremmo avere le competenze minime necessarie per utilizzare jQuery. Per imparare ad usare compiutamente jQuery rivolgiamoci a chi lo cura e lo distribuisce:
http://jquery.com/

Le altre librerie JavaScript

JQuery non è ovviamente la sola libreria JavaScript e probabilmente non è nemmeno la migliore in senso assoluto. Diverse sono le alternative e i complementi. Tra le più conosciute possiamo nominare, in ordine alfabetico: Dojo, Ext, MooTools, Prototype,

script.aculo.us. Alcune fanno le stesse cose in modo diverso, altre sono specializzate in qualcosa, tutte sono cross-browser e basate su JavaScript.
Non è certamente questa la sede in cui poter approfondire lo studio di ogni singola libreria, ma è importante conoscerne l'esistenza e avere un'idea delle caratteristiche tipiche. Per questo motivo, qui di seguito, troviamo una breve introduzione ad ognuna di esse.

Dojo

Dojo è una raccolta di utility JavaScript che forniscono soluzioni per una vasta gamma di problemi comuni durante lo sviluppo di un'applicazione web. Tutte le utility sono raccolte nel Dojo Toolkit. Vediamole brevemente.
Dojo Base raccoglie le funzionalità di base che servono durante lo sviluppo di un'applicazione web come possono essere la gestione del DOM e delle chiamate AJAX, animazioni e altre funzionalità di base.
Dojo Core è un insieme più ampio di componenti costruiti su Base Dojo e comprende cose come drag and drop, l10n e componenti i18n, e archivi di dati.
Dijit è un sistema completo di widget che fornisce tutti i componenti principali dell'interfaccia utente e le utility.
DojoX è una raccolta di componenti avanzati, comprese le griglie di dati, di basso livello librerie grafiche, componenti mobili, e altri software sperimentali.
Infine, la sezione Utilities fornisce informazioni dettagliate sul sistema Dojo Build, Dojo Module Loader, e una varietà di altri Dojo relative utilities.
Sito web: http://dojotoolkit.org/

Ext

Da poco uscita la versione 4, questo framework è molto potente ed eterogeneo. Sul sito ufficiale possiamo leggere:
"Ext JS 4 è il prossimo grande progresso nel nostro framework JavaScript. Dotato di funzionalità estese, plug-free grafici, e una nuova architettura MVC è il miglior Ext JS ancora. Creare applicazioni web incredibili per ogni browser."
Ext JS 4 ha introdotto diverse novità tra cui la possibilità di implementare l'MVC. Offre una buona gamma di widget per progettare facilmente un'interfaccia grafica interattiva e tanto altro.
Sito web: http://www.sencha.com/products/extjs/

MooTools

Molto conosciuta in ambiente spiacialyst perché fortemente orientata ad oggetti, questa libreria (un vero e proprio framework) è un po' più difficile da imparare per i novizi, mentre è più semplice e davvero potente per gli esperti della programmazione ad oggetti. MooTools è framework modulare, orientato agli oggetti, basato su JavaScript progettato appositamente per uno sviluppatore JavaScript intermedio/avanzato. Consente di scrivere codice potente, flessibile e cross-browser grazie alla sua API elegante, ben documentato, e coerente.

Sito web: http://mootools.net/

Prototype

Prototype è un framework che ha come scopo principale quello di semplificare le chiamate Ajax. Prototype consente di affrontare le chiamate Ajax in un modo molto facile e divertente senza per nulla trascurare la sicurezza. Oltre a semplici richieste, questo modulo si occupa anche di gestire in modo intelligente il codice JavaScript restituito da un server e fornisce classi di supporto per il polling.

Sito web: http://prototypejs.org/

Script.aculo.us

Il nome rende chiaro l'obiettivo di questa libreria: lo spettacolo! Il suo scopo principale è infatti quello di usare JavaScript per rendere spettacolare un sito web: animazioni, sfumature, slide, drag and drop, ecc.

Sito web: http://script.aculo.us/

13
Ajax

Nel primo capitolo abbiamo visto che programmare per il web significa mettere in comunicazione un client e un server. La comunicazione in genere avviene nei seguenti termini:

1. il client (nel nostro caso il browser) richiede una pagina al server
2. il server elabora la richiesta del client e risponde al client inviandogli la pagina richiesta
3. infine il client mostra la pagina a video.

Mettiamoci in un caso reale.

Quella mostrata nella figura qui sopra è la home page di un sito.
Facciamo l'ipotesi di cliccare sul link "contatti". A questo punto il browser invierà la richiesta della pagina "contatti" al sever. Il server, ricevuta la richiesta, la elabora e spedisce la pagina al client il quale, ricevuta la nuova pagina, ricarica tutto il contenuto della pagina a video e mostra la pagina "contatti".

Facciamo una riflessione: noi abbiamo richiesto la pagina contatti, ma quando riceviamo la risposta dal server, ci viene spedita la pagina nel suo insieme: menù di navigazione, immagini che scorrono, codici javascript per gestirle e poi anche la parte che riguarda i contatti. Non sarebbe molto più rapido lasciare così come sono il menù (che tanto è sempre quello), le immagini (che tanto sono sempre quelle) e caricare solo la parte della pagina che si riferisce ai contatti?

Decisamente si! Sarebbe molto più veloce! Bene, Ajax serve esattamente a questo!

Tramite Ajax noi siamo in grado di richiedere al server solo un pezzo della pagina e, una volta giunta la risposta dal server, ricaricare nel client solo la parte della pagina richiesta. Nel nostro caso, quindi, ricaricheremo solo il riquadro "contatti". Questo velocizza di moltissimo la navigazione.

Ajax quindi non è un linguaggio di programmazione! E' piuttosto un tecnica client-server alternativa a quella classica. Ma cosa significa ajax?

AJAX = Asynchronous JavaScript and XML.

Dalla definizione si deduce che per usare ajax servono javascript e XML. In realtà si può fare a meno di XML... i modi di usare ajax sono molteplici. Ciò di cui non si può fare a meno invece è un server! **Ajax per funzionare ha bisogno di un server**! Per iniziare

a scrivere codice ajax è pertanto necessario avere un po' di dimestichezza con il lato server di un'applicazione web. Poiché non è affatto scontato che il lettore abbia una tale dimestichezza, si è pensato di produrre gli esempi online, direttamente sul sito dell'autore. Per poter provare il codice quindi è necessario avere accesso a internet.
Infine si raccomanda di inoltrarsi nello studio di Ajax solo dopo aver appreso almeno i rudimenti della programmazione lato server. Del resto, Ajax senza un server non serve a niente.
Per tutti questi motivi la trattazione che faremo di Ajax sarà assolutamente elementare, cercando di evitarne una disamina completa che comunque necessiterebbe dello studio di XML prima e di JSON poi.

Il cuore di ajax: XMLHttpRequest

Per poter usare Ajax dobbiamo usare l'oggetto XMLHttpRequest. **XMLHttpRequest è un oggetto nativo in tutti i browser di ultima generazione**. Lo scopo di tale oggetto è quello di eseguire richieste al server in modo asincrono, ossia lasciando libero l'utente di continuare a navigare mentre il sistema recupera le informazioni richieste. Per usare XMLHttpRequest dobbiamo innanzitutto dichiararlo nel nostro linguaggio di scripting, tipicamente javascript.
Ecco la sintassi per sua dichiarazione:
xmlhttp=new XMLHttpRequest();

Una cosa abbastanza banale.
Ecco invece una pagina web semplicissima che utilizza Ajax.

```
<!DOCTYPE html>
<html>
    <head>
          <script type="text/javascript">
                function loadXMLDoc() {
                      var xmlhttp;
                      xmlhttp=new XMLHttpRequest();
                      xmlhttp.onreadystatechange=function(){
                            if (xmlhttp.readyState==4 &&
                                  xmlhttp.status==200) {
                                        document.
                                        getElementById("myDiv").
                                        innerHTML=xmlhttp.
                                        responseText;
```

```
                              }
                        }
                        xmlhttp.open("GET",
                              "nuovo_testo.txt",true);
                        xmlhttp.send();
                  }
            </script>
      </head>
      <body>
            <div id="myDiv"><h2>Questo testo sarà cambiato da
                  AJAX</h2></div>
            <button type="button" onclick="loadXMLDoc()">Cambia il
                  testo</button>
      </body>
</html>
```

Sembra difficile, ma non lo è.

Possiamo osservarne il funzionamento al seguente indirizzo:

http://www.alessandrostella.it/lato_client/ajax/ajax_01.html

Sveliamone i segreti riga per riga.

```
function loadXMLDoc() {
    var xmlhttp;
    xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function() {
          if (xmlhttp.readyState==4 && xmlhttp.status==200) {
                document.
                      getElementById("myDiv").
                      innerHTML=xmlhttp.responseText;
          }
    }
    xmlhttp.open("GET","nuovo_testo.txt",true);
    xmlhttp.send();
}
```

Commentiamo il codice.

```
xmlhttp.onreadystatechange=function() {...}
```

con questa riga stiamo assegnando all'attributo "onreadystatechange" dell'oggetto XMLHttpRequest il risultato della funzione che si trova nelle parentesi graffe. A cosa serve l'attributo "onreadystatechange"? E' un gestore di eventi che viene lanciato ogni volta che c'è un cambiamento di stato. Quale stato? Lo stato di "onready" cioè "pronto"!
Vediamo cosa fa la funzione assegnata a tale attributo.

```
if (xmlhttp.readyState==4 && xmlhttp.status==200) {
    document.
          getElementById("myDiv").
          innerHTML=xmlhttp.responseText;
}
```

Non sembra poi tanto complicato.
Se l'oggetto XMLHttpRequest ha l'attributo "readyState" con valore 4 (cioè ha ricevuto una risposta del server) e se ha l'attributo "status" con valore 200 (cioè tutto ok!) allora cerca l'elemento HTML con id="myDiv" [document.getElementById("myDiv")] e scrive al suo interno quello che ha ricevuto dalla risposta del server (xmlhttp.responseText).
Quindi ogni volta che si verifica un cambiamento di stato, viene richiamata questa funzione.
Infine:

```
xmlhttp.open("GET","nuovo_testo.txt",true);
xmlhttp.send();
```

La prima riga comunica all'oggetto XMLHttpRequest di preparare una chiamata al server che abbia le seguenti caratteristiche:

- di tipo GET
- che legga il contenuto del file denominato "nuovo_testo.txt"
- che sia asincrona (true)

La seconda riga effettua realmente la richiesta al server.

Per quanto possa sembrare "semplice" questa è la base di Ajax. Non c'è nient'altro.
Facciamo un piccolo passo avanti.

Nel codice che abbiamo scritto abbiamo considerato l'ipotesi in cui il collegamento tra client e server vada a buon fine. E se invece le cose vanno male? Ecco qui di seguito una possibile soluzione:

```
if (xmlhttp.readyState==4 && xmlhttp.status==200) {
    document.getElementById("myDiv").
          innerHTML=xmlhttp.responseText;
} else if (xmlhttp.readyState==4 && xmlhttp.status!=200) {
    document.getElementById("myDiv").
          innerHTML="<p>Si &egrave; verificato un errore
          durante il collegamento con il server.</p>";
}
```

Possiamo testare questo secondo codice qui:
http://www.alessandrostella.it/lato_client/ajax/ajax_02.html

Per concludere cerchiamo di conoscere meglio questo misterioso XMLHttpRequest, elencandone metodi e attributi.

Metodi

Metodo	Descrizione
abort()	Cancella la richiesta in atto.
getAllResponseHeaders()	Restituisce sotto forma di stringa tutti gli header HTTP ricevuti dal server
getResponseHeader (nome_header)	Restituisce il valore dell'header HTTP specificato
open(metodo, URL) open(metodo, URL, async) open(metodo, URL, async, userName) open(metodo, URL, async, userName, password)	Specifica il metodo, l'URL e altri parametri opzionali per la richiesta. Il parametro metodo può assumere valore di "GET", "POST", oppure "PUT" ("GET" è utilizzato quando si richiedono dati, mentre "POST" è utilizzato per inviare dati, specialmente se la lunghezza dei dati da trasmettere è maggiore di 512 byte). Il parametro URL può essere sia relativo che assoluto. Il parametro "async" specifica se la richiesta deve essere gestita in modo asincrono oppure no – "true" significa che lo script può proseguire l'elaborazione senza aspettare la risposta dopo il metodo send(), mentre "false" significa che lo script è costretto ad aspettare una risposta dal server prima di continuare.
send(content)	Invia la richiesta

setRequestHeader(chiave, valore)	Aggiunge la coppia chiave/valore alla richiesta da inviare.

Attributi

Attributo	Descrizione
onreadystatechange	Gestore dell'evento lanciato ad ogni cambiamento di stato.
readyState	Restituisce lo stato corrente dell'istanza di XMLHttpRequest: 0 = non inizializzato, 1 = aperto, 2 = richiesta inviata, 3 = risposta in ricezione e 4 = risposta ricevuta.
responseText	Restituisce la risposta del server in formato stringa
responseXML	Restituisce la risposta del server come oggetto Document, che potrà essere esaminato secondo le specifiche DOM del W3C.
status	Restituisce il codice HTTP restituito dal server (per esempio 404 per "Not Found" e 200 per "OK").
statusText	Restituisce lo status in forma di stringa descrittiva (per esempio. "Not Found" oppure "OK").

Concludiamo questa lunga avventura studiando un sito reale:
http://www.salentopaghe.it/

Questo sito utilizza sia Ajax (tramite jQuery) sia le Media Queries (per adattarsi alle varie dimensioni della finestra del browser), ed è dunque l'ideale per vedere in azione tutto quello che abbiamo studiato. Osserviamolo con cura. Sappiamo come visualizzarne il codice sorgente (tasto destro del mouse) e siamo quindi in grado di rintracciare CSS e script utilizzati al suo interno.
Ad esempio possiamo osservare come cliccando sulla voce "MODELLI" (così come sulla voce "CONTATTI"), il sito ricarichi solo la parte centrale della pagina, usando Ajax tramite jQuery. Inoltre possiamo osservare che per farlo usa la seguente funzione:

```
function apriPagina(pagina) {
    $("#center_left").html("");
    $.ajax({
          type: "GET",
          url: "page/"+pagina+".php",
          success: function(response){
                $("#center_left").html(response);
          },
          error : function () {
```

```
                $("#center_left").css("display","none")
                alert("Si è verificato un errore durante la
comunicazione con il server.\nRiprovare più tardi.");
            }
    });
}
```

Da questa funzione (e dai link dai quali viene richiamata) possiamo dedurre che sul server esiste una cartella "page" all'interno della quale ci sono due file php denominati "modelli.php" e "contact.php" che vengono richiamati dalle rispettive voci di menù presenti sul sito.

Infine, allargando e restringendo la finestra del browser possiamo osservare dal vivo cosa sono in grado di fare le Media Queries.

Se, alla fine di questa avventura, **ritieni che valga la pena ringraziarmi** per il lavoro, l'impegno e le ricerche necessarie alla stesura di questo libro, allora puoi farlo **donandomi 1 euro**.

Si, un solo euro.

Per farlo ti basta cliccare sull'immagine qui in basso.

In ogni caso, Grazie.