Tecnologie Web

Guida al codice

Enrico Mensa,

Basato sulle lezioni del del prof. Gianfranco Ruffo

HTML

CSS

HTML

1) Nozioni su HTML

HTML è un linguaggio di markup. Come tutti i linguaggi di markup, quindi, il corpo è costituito da una parte che è il contenuto della pagina ed un'altra che specifica come il contenuto deve essere rappresentato.

1.1) Le DTD (Document Type Definition)

In ogni pagina dobbiamo includere una delle tre DTD a seconda dei tag che vogliamo considerare accettati.

- HTML 4.01 **Strict** DTD (include tutti gli elementi e gli attributi che non sono stati disapprovati o che non appaiono nei documenti con frame):

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

- HTML 4.01 **Transitional** DTD (include tutto ciò che fa parte della DTD rigorosa più elementi e gli attributi disapprovati):

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/loose.dtd">

- HTML 4.01 **Framset** DTD (include la DTD transitoria completa più i frame) - *SCONSIGLIATA*:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/frameset.dtd">

1.2) Schema generale dei tag

I tag iniziano sempre con il carattere **<** e sono seguiti da un nome e poi da vari attributi (eventualmente) nella forma attribute1 = "value".

Per alcuni tag si adopera anche </nome>.

Se il tag non usa </nome> è comunque suggerito l'utilizzo di <nome .... />.

HTML è case insensitive, ma per buona scrittura si suggerisce l'utilizzo di tag in minuscolo.

1.3) Gli attributi globali

Questi attributi possono essere usati in qualsiasi elemento HTML:

- style (specifica uno stile CSS per il singolo elemento)
- ID (specifica un ID unico per un certo elemento)
- class (assegna un nome di classe riportato poi nel CSS)

2) I tag

Note per la lettura di questo documento:
Con [ ] si rappresentano le scelte possibili (ove non banale indicarle) e fra () la spiegazione dell'attributo. Ogni attributo è separato da /.

<html> </html>
- *Descrizione:* delimita il documento html.

<head> </head>
- *Descrizione:* delimita le caratteristiche del documento.

<body> </body>
- *Attributi*: **bgcolor** (colore di sfondo) / **background** (nome file immagine) / **text** (colore testo) / **link** (colore link da visitare) / **vlink** (colore link visitato) / **alink** (colore link con cursore sopra).
- *Descrizione:* delimita il corpo del documento.

<title> </title>
- *Descrizione:* specifica il titolo della pagina.

<h1> </h2> ... <h6> </h6>
- *Descrizione:* rappresenta un titolo (con grandezza via via minore).

** **
- *Attributi*: **align** [center | left | right | justify] (allinea un paragrafo)
- *Descrizione:* inserisce un paragrafo (a capo automatico).

** **
- *Descrizione:* centra del testo.

</br>
- *Descrizione:* manda a capo.

** , , , <code> </code>, **
- *Descrizione:* rispettivamente il carattere è modificato in bold, italic, enfatizzato, codice, bold.

** **
- *Attributi*: **size** (dimensioni) / **color** (colore) / **face** (il font)
- *Descrizione:* rispettivamente il carattere è modificato in bold, italic, enfatizzato, codice, bold.

** , <ol> , <li> **
- *Attributi *: **type** (cambia il simbolo della lista)
- *Descrizione:* con <ul> si effettua una lista non ordinata, con <ol> una lista ordinata. Con <li> i punti della lista.

<!-- -->
- *Descrizione:* commento.

**<a> *testo del link* **

- *Attributi*: **href** [URL, #name] (crea il link ad un URL oppure ad un punto interno della pagina denominata con <a name=“”>) / **name** (definisce il nome di un link)
- *Descrizione:* crea link
- *Esempio:*

```
<a dove=”art1”></a>

<a href="#art1">Articolo1</a>
```

Clickando sul secondo link si viene portati a dove è stato scritto il primo link.

<img/>

- *Attributi*: **src** (url del file d’origine) / **align** [left | right | center] / **border** (stile del bordo) / **height** (altezza) / **width** (larghezza) / **hspace** (spazio fra l’immagine ed il testo a fianco) / **vspace** (spazio fra l’immagine ed il testo sopra e sotto) / **alt** (testo alternativo all’immagine).
- *Descrizione:* inserisce un’immagine.

<table> </table>

- *Attributi*: **width** (larghezza) / **cellspacing** (distanza fra una cella e l’altra - default 1pixel) / **cellpadding** (distanza fra lo spazio vuoto ed il dato nella cella. Esprimibile in percentuale o pixel - default 0) / **bgcolor** (colore di sfondo) / **border** (stile del bordo)
- *Descrizione:* Crea una tabella vuota.

<tr> </tr>, <td> </td>, <td> </td>

- *Attributi <td> e <th>*: **width** (larghezza) / **colspan** (occupa lo spazio di n celle orizzontalmente) / **rowspan** (occupa lo spazio di n celle verticalmente) / **bgcolor** (colore di sfondo) / **border** (stile del bordo) / **align** [left | right | center] / **valign** [top | bottom | middle] / **nowrap** (nessun contorno per quella cella)
- *Descrizione:* <tr> (table row) crea una riga, <td> (table data) crea una cella. Con <th> si crea una cella di head, cioè con il grassetto.

** **

- *Attributi* : **align** [left | right | center | justify]
- *Descrizione:* il div serve per suddividere una parte della pagina da un’altra. È l’evoluzione della table con border nullo.

<colgrup/>

- *Attributi* : **span** (numero di colonne che compongono il gruppo) / **align** [left | right | center | justify] / **width** (larghezza delle colonne componenti il gruppo)
- *Descrizione:* crea gruppi di colonne.

<frameset> </frameset>, <noframes> </noframes>

- *Attributi* : **rows** (divide la pagina orizzontalmente) / **cols** (divide la pagina verticalmente)
- *Descrizione*: sostituisce il comando <body> e fornisce la possibilità di caricare diverse pagine HTML in una sola schermata del browser. Con <noframes> si specifica cosa deve caricare il browser nel caso in cui i frame non siano supportati

<frame> </frame>

/* TODO */

<form> </form>

- *Attributi* : **action** (pagina o eseguibile che riceverà i dati del form) / **method** [get | post] (metodologia con cui comunicare con l'action, con get l'URL viene compilato in chiaro con gli attributi con il post invece viene incluso nel messaggio inviato in HTTP e non nell'URL) / **enctype** [application/x-www- form-urlencoded ! multipart/form - data] (specifica il tipo di dato che verrà inviato al server, utile per la negoziazione) / **accept-charset** (quale codifica di caratteri è accettata all'interno del form) / **accept** (specifica i tipi, elencati uno dopo l'altro e separati da virgola, che il server sarà in grado di elaborare una volta inviati i dati del form) / **name** (identificare, ma è obsoleto, meglio id. Retrocompatibilità!)
- *Descrizione:* fornisce un contenitore di moduli per form.

<input> </input>

- *Attributi* :
    - **type** [text (crea un textfield da una riga) | password (textfield ma con pallini per nascondere l'input) | checkbox | radio | submit (tasto che permette l'invio del modulo) | reset (resetta tutti i campi del form) | file (permette di caricare un file) | hidden (è un campo nascosto, utile per accogliere informazioni dal server) | image | button]
    - **name**
    - **value** (valore iniziale del controllo, facoltativo tranne per radio e checkbox)
    - **size** (larghezza del controllo, numero di caratteri per text e password)
    - **maxlength**
    - **checked** (per radio e checkbox)
    - **src** (se è image)
- *Descrizione*: crea una casella di input per interagire con l'utente.

<button> </button>

- *Attributi* : **type** [submit (tasto che permette l'invio del modulo) | reset (resetta tutti i campi del form) | button], **name**, **value** (valore iniziale del controllo)
- *Descrizione*: funziona come l'input ma fornisce più possibilità di personalizzazione.

<select> </select>, <optgroup> </optgroup>, <option> </option>

- *Attributi <select>* : **multiple** (permette selezioni multiple), **name**, **size** (quantità di elementi visibili simultaneamente nell'interfaccia)
- *Attributi <option>* : **selected** (selezionato di default), **value** (valore del controllo, se non è specificata vale il contenuto dell'elemento option), **label** (etichetta per rappresentare il value, visibile all'utente)
- *Attributi <optgroup>* : **label** (etichetta per il gruppo di opzioni)
- *Descrizione*: fornisce un menù con più scelte. Ogni option è una scelta del menù.

<textarea> </textarea>

- *Attributi* : **name**, **rows**, **cols**
- *Descrizione*: area di testo

<fieldset> </fieldset>, <legend></legend>

- *Descrizione*: crea un contorno intorno agli elementi all'interno del tag. Ottimo per IUM. Con legend si specifica quale sia il label con cui nominare il contorno.

CSS

1) Nozioni su CSS

Lo scopo di CSS è quello di separare il codice "core" dalla parte grafica del sito. Questo ne aumenta la portabilità, la facilità di sviluppo e di gestione.
CSS introduce per altro molte funzioni ed è uno strumento potente.

1.1) La struttura gerarchica

Il documento HTML è creato basandosi sul principio parent-child. Questo è da tenere a mente poiché CSS si basa molto sulla gerarchia per la definizione dei suoi selettori.

1.2) Gli statements

Il foglio di stile CSS contiene una serie di statements (regole) che a loro volta constano di:
- Selettore
- **Dichiarazione** (un insieme di coppie "proprietà = valore")

Esempio

```
body        //SELETTORE
{           //DICHIARAZIONE (con lista proprietà = valore fra le graffe)
      color: black;
      align: left;
}
```

Ciò che è espresso nella dichiarazione viene applicato alla selezione. Ogni riga della dichiarazione deve terminare con un punto e virgola, tranne l'ultima in cui può essere omesso.

1.3) Tre modi di utilizzare CSS

È possibile introdurre codice CSS in tre modi:

- **inline** mediante l'attributo style:

```
<p style="color: red; font-style: italic"> TESTO IMPORTANTE </p>
```

- **embedded** mediante il tag <style> dentro al tag <head>:

```
<head>
      <style type="text/css">
            <!-- statements di CSS -->
      </style>
</head>
```

(*si suggerisce di mantenere gli statements all'interno di commenti HTML poiché si evita l'indicizzazione dei motori*)

- **tramite file esterni** mediante il tag <link> dentro al tag <head>

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

Nella fattispecie si sta applicando il CSS scritto nel file "stile.css" che si trova nella stessa folder del file HTML.
Per link è inoltre presente l'attributo opzionale **media**, che specifica su quale dispositivo applicare lo stile CSS.
I possibili valori sono: **all** (tutti i dispositivi), **aural** (sintetizzatori vocali), **braille**, **embossed** (stampanti braille), **handheld** (palmari), **print** (materiale a pagine opache), **projection** (materiale da proiettare), **screen** (per schermi del computer), **tty** (terminali), **tv**.

È oltremodo possibile specificare il foglio esterno con la direttiva **@import**.

```
@import url(http://...);
@import "http://...";
```

ATTENZIONE! Le regole specificate in un foglio di stile annullano quelle importate, e la direttiva @import relativa al CSS deve essere la prima fra tutte.
La differenza fra @import e <link> è che nel primo caso è possibile fondere insieme regole di stile.

1.4) Regole di cascata

I fogli di stile hanno tre origini differenti:
- **Autore**: specificano nel sorgente lo stile.
- **Utente**: può aggiungere un riferimento ad un file css.
- **User Agent** (client dell'utente): utilizza l'@import.

Tutti questi fogli di stile si sovrappongono. Dunque quali regole vengono eseguite? Si segue il peso:

Autore > Utente > User Agent

Perciò le definizioni applicate all'interno del codice (attributo style) hanno priorità su quelle definite all'interno dell'head con il tag <style> e con il tag <link>, che a loro volta hanno priorità sui CSS da @import.

Fra fogli dello stesso autore, si tiene conto della **specificità** (ad esempio se abbiamo definito una regola per DIV ed una regola per un id #1234, un avrà applicata la seconda regola e non la prima.
Nel caso di pari specificità, ci si basa sull'**ordine di scrittura**: in caso di conflitto verrà eseguita solo l'ultima regola.
Quindi ad esempio:

```
<html>
<head>
      <style type="text/css"> h1 {color:green;} </style>
      <link rel="stylesheet" type="text/css" href="stile.css"> </link>
</head>

<body style="color:red">
      <h1 style="color:blue">Che colore ho?</h1>
</body>
</html>
```

Supponendo che il file "stile.css" imposti il colore degli h1 a giallo, l'h1 avrà come colore: giallo, verde, rosso o blu?
La risposta è blu, poiché h1 è l'ultimo a ridefinire il colore tramite l'attributo style (massimo peso).

2) I selettori

Il selettore universale

Descrizione: seleziona ogni singolo elemento HTML (di ogni tipo) all'interno della pagina.
Esempio:

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

Il selettore di tipo

Descrizione: seleziona il nome di un tipo di elemento di HTML. Ogni istanza di quel certo elemento verrà selezionata.
Esempio:

```
H1 { font-family: sans-serif; } // tutti gli elementi H1 avranno font sans-serif
```

Il selettore di discendenti

Descrizione: costituito da due o più selettori separati da uno spazio bianco. Dato un selettore "A B" verrà selezionato ogni B che discende da un A. Discendente significa che si trova annidato all'interno di A a qualunque livello.
Esempio:

```
body p { color: Red; }
```

Tutti i p contenuti nel body a qualunque livello avranno colore rosso

Il selettore di figli

Descrizione: costituito da due o più selettori separati da ">". Dato un selettore "A > B" verrà selezionato ogni B che è figlio diretto (al primo livello) di un A.
Esempio:

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

Tutti i p contenuti nel body al primo livello saranno colorati di rosso.

* <body> Ciao! </body> **selezionato**
* <body> Ciao! </body> **non selezionato**

Il selettore di adiacenti

Descrizione: costituito da due selettori separati da "+". Dato un selettore "E1 + E2" verrà selezionato ogni E2 che ha come padre lo stesso di E1 ed E2 precede immediatamente E1.
Esempio:

```
H1 + p { text-indent: 0; }
```

Tutti i p "fratelli" di H1 (cioè che seguono direttamente dopo un H1) non dovrebbero essere indentati.

* <h1> Testo </h1> Ciao! **selezionato**
* <h1> Testo </h1><h2> Ciao! </h2> **non selezionato (<h2> include)**
* <h1></h1><h2> Altro testo </h2> Ciao! **non selezionato (<h2> è fra <h1> e)**

Il selettore di attributi

Descrizione: a seconda della presenza dell'attributo (ed eventualmente del suo valore) in un certo elemento, quell'elemento viene selezionato.

- **E[att]**, E viene selezionato se ha specificato l'attributo att, qualunque valore esso assuma.
- **E[att = val]**, E viene selezionato se ha specificato l'attributo att con valore val.
- **E[att ~= val]**, E viene selezionato se, data la serie di parole (separate da spazio) che att assume, fra quelle ha specificato il valore val.
- **E[att |= val]**, E viene selezionato se, data la serie di parole (separate da trattino) che att assume, la prima fra quelle è il valore val.

Esempio:

```
H1[title] { color: Blue; }
```

Tutti gli h1 con title specificato (con qualunque valore) verrà selezionato.

∗ <h1 title=”titolo”> Testo </h1> **selezionato**

∗ <h1> Testo </h1> **non selezionato**

Il selettore di classe

Trattasi di un selettore di attributo (il terzo sopracitato). Se si sta specificando per l’attributo class, è possibile usare la dot notation. Pertanto **E[class ~= val]** equivale a dire **E.val**. Perciò, notazioni legittime sono:

```
.myclass { color: Red; }
h1.myclass { color: Red; }
```

Nel primo caso tutti gli elementi che specificano “myclass” fra i nomi assegnati all’attributo class verranno selezionati.

Nel secondo caso, solamente gli <h1> che specificano “myclass” fra i nomi assegnati all’attributo class veranno selezionati. Quindi gli h1 nella forma: <h1 class=”myclass”>.

Il selettore di ID

Descrizione: ogni elemento può specificare un attributo ID con valore univoco. Nel css, per selezionare quell’ID, è sufficiente inserire prima del valore di ID un hashtag “#”.

Esempio:

```
#footer { color: Red; }
```

Verrà selezionato , ad esempio.

I selettori di ID hanno precedenza rispetto a quelli di attributo.

3) Pseudo-classi e pseudo-elementi

Le **pseudo-classi** sono caratteristiche normalmente non deducibili dall'albero della pagina. Una pseudo-classe, infatti, non definisce un elemento ma bensì un particolare stato di un elemento.
Gli **pseudo-elementi** creano invece astrazioni sopra all'albero della pagina aggiungendo così funzionalità. Ad esempio l'accesso alla prima riga o alla prima lettera di un certo elemento.

Le pseudo-classi sono:

:first-child
Descrizione: seleziona un elemento che è il primo figlio di un altro elemento.
Esempio:
```
DIV > p:first-child { text-indent: 0; }
```
Verrà indentato a zero ogni P che è primo figlio di un DIV.
∗ Ciao! **selezionato**
∗ Testo </h1> Ciao! **non selezionato (non è nel)**
∗ <h1>Testo</h1> Ciao2   **non selezionato (<h1> è il primo figlio, non !)**

:link
Descrizione: si applica ai collegamenti non ancora visitati.
Esempio:
```
A:link { color: blue; }
```

:visited
Descrizione: si applica ai collegamenti già visitati.
Esempio:
```
A:visited { color: blue; }
```

:hover
Descrizione: si applica agli elementi puntati (col cursore) ma non ancora attivati.
Esempio:
```
DIV:hover { color: blue; }
```

:active
Descrizione: si applica agli elementi attivati, ovvero in quell'istante che intercorre fra il click dell'utente ed il suo rilascio.
Esempio:
```
DIV:active { color: blue; }
```

:focus
Descrizione: si applica quando l'elemento accetta eventi da tastiera o da altri input testuali.
Esempio:
```
input:focus { background-color: yellow; }
```

Gli pseudo-elementi sono:

:first-line

Descrizione: seleziona la prima linea di un paragrafo.
Esempio:

```
P:first-line { text-transform:uppercase }
```

:first-letter

Descrizione: seleziona la prima lettera di un paragrafo.
Esempio:

```
P:first-letter {
      font-size:200%;
      font-style: italic;
      font- weight:bold;
      float: left
}
```

4) CSS per la tipografia

Alcune notiazioni utili per la lettura:
1em equivale alla dimensione in punti del font. Specificare 0.6em significa impostare un carattere 0.6 volte grande quello del padre.
La misura in punti è espressa come pt, in pixel px, in em em.
I colori sono: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow.

Attributi per i font

- **font-weight**: bold, bolder, lighter, [100, ..., 900]
- **font-stretch**: normal, wider, narrower, ultra-condensed, condensed, semi-condensed, semi-expanded, expanded, ultra-expanded
- **font-size**: *misura assoluta in punti*, *misura assoluta in pixel*, xx-small, x-small, medium, large, x-large, xx-large, *misura relativa a ciò che c'è di fianco* larger, smaller, %, em.
- **font-family**: *nome di un font particolare*, serif, sans-serif, cursive, fantasy, monospace
- **font-style**: normal, italic, oblique
- **font-variant**: normal, small-caps

Attributi per il testo

- **text-indent**: *misura assoluta in punti, misura assoluta in pixel, misura assoluta in cm, valori relativi in %*
- **text-align**: center, right, left, justify
- **text-decoration**: none, underline, overline, line-through
- **text-shadow**: none, color, *length*

Attributi per lo sfondo

- **background-color**: *un colore*, trasparent
- **background-image**: url(*http://myurl*)
- **background-repeat**: repeat, repeat-x, repeat-y, no-repeat
- **background-attachment**: fixed, scroll
- **background-position**: %, top, bottom, left, right (e combinazioni dei quattro)

Attributi per le tabelle

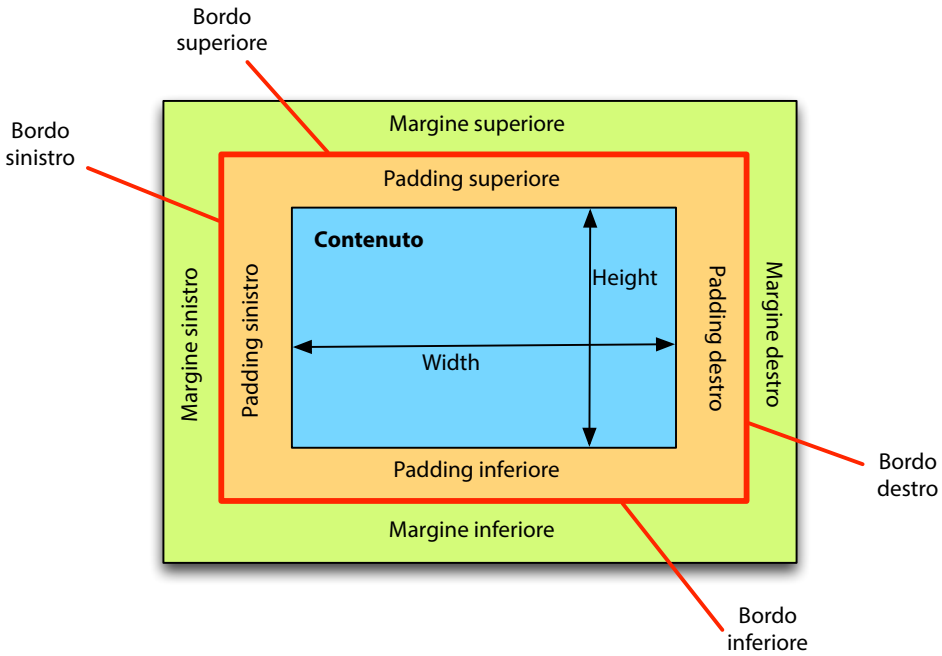
- **table-layout**: auto, fixed
- **border-spacing** (distanza fra le celle): *misura assoluta in punti, misura assoluta in pixel, misura assoluta in cm*
- **border-collapse** (gestione dei bordi fra celle di una tabella): collapse, separate
- **empty-cells** (gestione dei bordi fra celle di una tabella): show, hide

Attributi per le liste

- **list-style-image**: url(*http://myurl*), none
- **list-style-position** (il simbolo appare all'interno o all'esterno del workflow): inside, outside (default)
- **list-style-type**: disc, circle, square, decimal, decimal-leading-zero, lower-roman, upper-roman

Sintassi abbrevviata: **list-style**: <type> <position> <image>

Box model



Gestione dell'altezza (height)

- **height**: *valore numerico*, *valore in percentuale*, auto (default - la dimensione del contenuto)
- **overflow**: visibile (il contenuto esce dall'altezza) , hidden (nasconde il resto), scroll (aggiunta scrollbar), auto
- **min-height**: *valore numerico*
- **max-height**: *valore numerico*

Gestione della larghezza (width)

- **width**: *valore numerico*, *valore in percentuale*, auto (default - la dimensione del contenuto)
- **min-width**: *valore numerico*
- **max-width**: *valore numerico*

Gestione dei margini (margin)

- **margin-top**: *valore numerico*, *valore in percentuale*, auto (default - 0 solitamente)
- **margin-right**: *valore numerico*, *valore in percentuale*, auto (default - 0 solitamente)
- **margin-bottom**: *valore numerico*, *valore in percentuale*, auto (default - 0 solitamente)
- **margin-left**: *valore numerico*, *valore in percentuale*, auto (default - 0 solitamente)

Sintassi abbrevviata: **margin**: <top> <right> <bottom> <left>

Gestione del padding (padding)

- **padding-top**: *valore numerico*, *valore in percentuale*, auto (default - 0 solitamente)
- **padding-right**: *valore numerico*, *valore in percentuale*, auto (default - 0 solitamente)
- **padding-bottom**: *valore numerico*, *valore in percentuale*, auto (default - 0 solitamente)
- **padding-left**: *valore numerico*, *valore in percentuale*, auto (default - 0 solitamente)

Sintassi abbrevviata: **padding**: <top> <right> <bottom> <left>

Gestione dei bordi (border)

Vi sono diverse combinazioni di bordi. Il pattern è: border-<lato> dove lato può essere top, right, left, bottom.
Se si vuole specificare il bordo per tutto l'oggetto è sufficiente omettere il lato (quindi usare solo boder).

- **border-*lato*-color**: *colore*
- **border-*lato*-style**: none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset
- **border-*lato*-width**: *valore numerico,* thin, medium

Sintassi abbrevviata: **border-*lato***: <width> <style> <color>

Attributo display

Vi sono due diversi tipi di blocchi. I blocchi **inline** (es:) che tendono a riempire tutta la linea ed i blocchi **block** che invece occupano una sezione separata dalle altre. Per controllare questo, usiamo l'attributo display.

- **display**: inline, block, list-item (viene usato come elemento di una lista)

Attributi per il posizionamento

Vi sono tre tipologie di posizionamento: flusso normale (scatole una dopo l'altra), float (scatole fluttuanti), posizionamento esplicito (dipendente dal contenitore). Sfruttando le proprietà position e float si ottengono le combinazioni desiderate.

- **position**: static (l'oggetto è posizionato in flusso normale, dove ci si aspetta), relative (si specifica uno spostamento rispetto allo static), absolute (la posizione è specificata rispetto al contenitore), fixed (la posizione è fissa rispetto alla finestra del browser).

Dopo aver specificato l'attributo position, a meno di aver scelto static, si procede con gli attributi

- **top**: v*alore numerico*
- **right**: *valore numerico*
- **bottom**: *valore numerico*
- **left**: *valore numerico*

per esplicitare i vari spostamenti/floating.

Attributo z-index

Per definire un ordinamento fra gli oggetti nella pagina, è necessario l'utilizzo dello z-index.

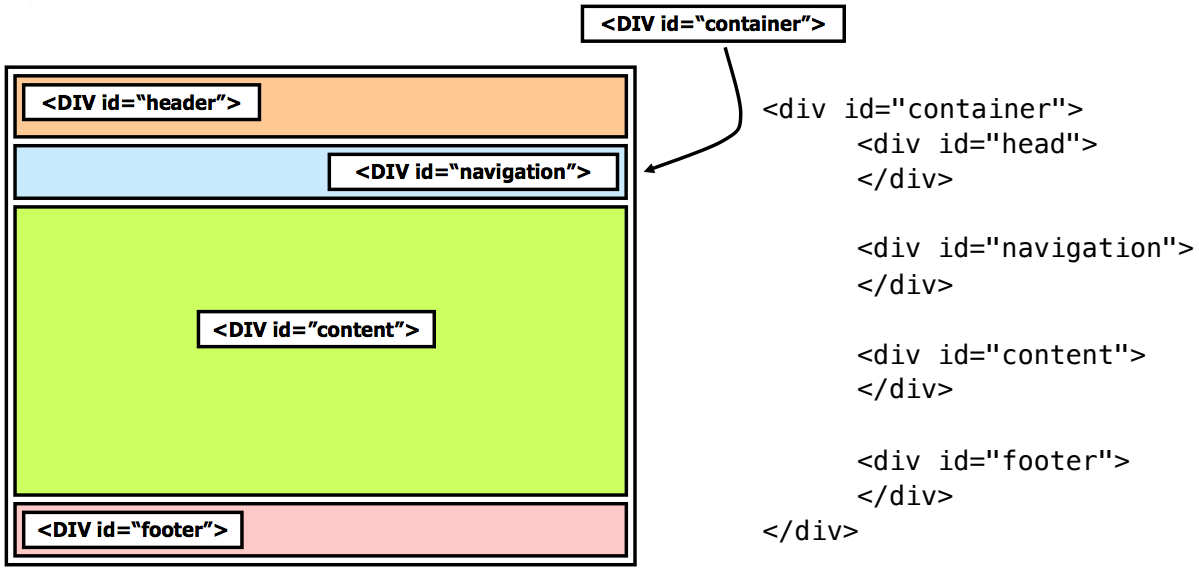
- **z-index**: *valore*, auto.

Un oggetto con z-index:10 sarà posizionato più in alto di uno con z-index:2.

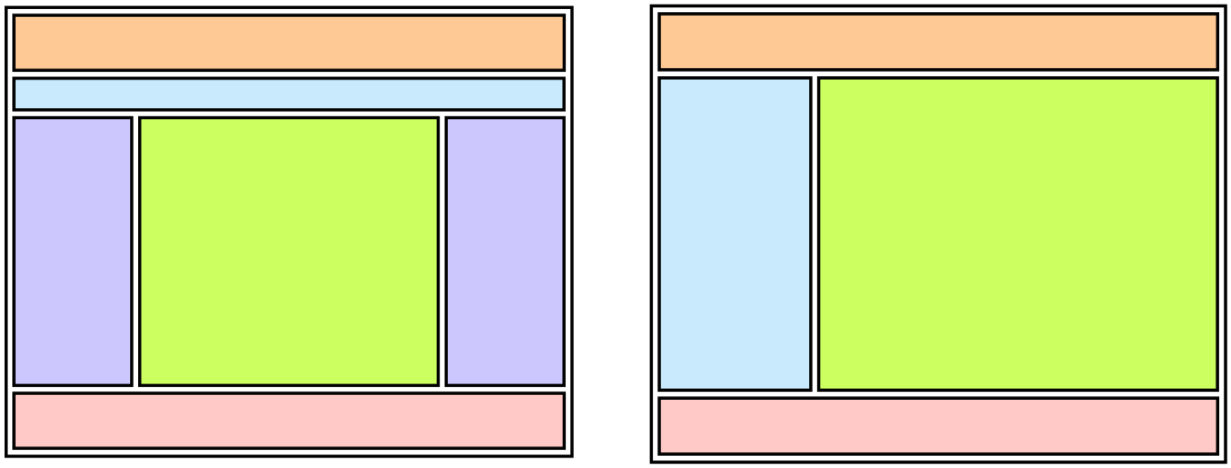
I layout tableless

Vediamo ora alcuni layout "base" adoperati nei siti. Si suggerisce l'utilizzo di un **layout fisso**, per evitare di coinvolgere più possibile gli attributi per il posizionamento, noti per la loro "difficile" implementazione.

Layout a colonna singola



Layout a più colonne



JavaScript

1) Nozioni su JavaScript

JavaScript è un linguaggio di **scripting** (quindi interpretato) **client**-**side**, **object**-**oriented**.
Il suo interprete è il browser web.

1.1) Come includerlo nel codice

È possibile introdurre codice JavaScript direttamente nel codice HTML:

```
<script language =”JavaScript” type=”text/JavaScript”>
      istruzione 1;
      ...
      istruzione n;
</script>
```

Oppure importandolo da un file esterno:

```
<script src=”myScript.js”> </script>
```

Infine è possibile scrivere codice JavaScript direttamente all'interno dei tag, associandoli ad un attributo che è nella fattispecie un **evento**. JavaScript viene quindi adoperato come gestore di eventi.

```
<a href=”#” onclick=”alert('You clicked this link!')”> Click Me </a>
```

Nella fattispecie dei link, si può anche adoperare l'attributo href direttamente (purché si specifichi che si sta adoperando JavaScript:

```
<a href=”Javascript: alert('You clicked this link!')”> Click Me </a>
```

2) Il core di JavaScript

JavaScript è **case sensitive**, i commenti sono:
- /* *testo commentato* */
- //*testo commentato*
- <!-- *testo commentato* //-->

Sono supportati: if else, for, while, do while, switch, try catch finally.

2.1) I tipi di dato

Javascript è debolmente tipato, per dichiarare una variabile è sufficiente adoperare il suffisso **var**. Prima che una variabile abbia un contenuto, essa contiene undefined. Per conoscere il tipo di una variabile è possibile adoperare il metodo **typeof**.
Le variabili possono contenere:
- Numeri
    - Infinity, -Infinity, NaN
    - Number.MAX_VALUE, Number.MIN_VALUE
- Stringhe (con relativo .length)
- Booleani (true, false)
- Valore nullo (null)
- Valore indefinito (undefined)

2.2) Input/Output con l'utente

- **alert**("Testo") --> restituisce undefined
- **prompt**("Domanda", "RispostaDefault") --> restituisce l'input se viene premuto OK, null se viene premuto annulla
- **confirm**("Domanda") --> restituisce true o false a seconda se venga premuto OK oppure annulla

2.3) I dizionari e gli array

Dizionari: associamo valori a chiavi.

```
a = {"x" : 1, "y" : 2, 5 : "ciao"};
a["x"] = a[5];
```

Array: associamo valori a chiavi.

```
a = [2, 4, "ciao"];
```

oppure:

```
a = new Array();
a[0] = 2;
a[1] = 4;
a[2] = "ciao";
```

2.4) Le funzioni: un tipo di dato

Una funzione è un oggetto istanziabile tramite un costruttore.

```
function square(x) {
      return x*x;
}
```

ma anche:

```
f = new Function("x", "return x*x;");
f = function(x) {return x*x;};
```

2.5) Un esempio su cui riflettere (passaggio di parametri)

```
<script>
   function modifyObj(obj) { obj['x'] = 123; }
   function modifyNumber(n) { n = n+1; }
   function modifyString(s) { s += " mare!" }

   obj = {"a":1,"b":2,"c":3};
   n = 33;
   s = "Ciao";

   alert(obj["x"]); alert(n); alert(s);
   modifyObj(obj);
   modifyNumber(n);
   modifyString(s);
   alert(obj["x"]); alert(n); alert(s);
</script>
```

Cosa ci aspettiamo accada?
La prima stampa darà rispettivamente: "undefined", 33, Ciao.
La seconda darà rispettivamente: 123, 33, Ciao.
Il primo valore è 123, poiché dentro alla funzione stanziamo nel dizionario un campo 'x' e gli associamo il 123 che poi andremo a stampare.

Il secondo è 33 poiché n non viene toccato.
Il terzo è Ciao per lo stesso motivo.

3) Gli oggetti

3.1) Il concetto di oggetto

Un oggetto è una collezione di "named piece of data". A seconda del loro "name" sono rintracciabili sull'oggetto, e vengono chiamati **proprieties**.
Ad esempio un oggetto image ha proprieties width e height che sono referenziabili come image.width e image.height.
Ogni oggetto ha poi metodi propri che possono essere richiamati (esempio: Math.sqrt(5);).
JavaScript fornisce una gerarchia di oggetti manipolabili che rappresentano la pagina HTML, quindi ogni oggetto della pagina è referenziabile.

3.2) Creare nuovi oggetti

Per creare un nuovo oggetto:

```
var myobj = new Object();
```

Ogni qualvolta si voglia aggiungere una proprietà ad un oggetto è sufficiente scrivere (si crea **propr1**):

```
myobj.prop1 = "text";
```

Il costruttore
Volendo è possibile creare un costruttore per creare oggetti "sempre uguali".

```
function Rectangle(w, h) {
      this.width = w;
      this.height = h;
      this.area = function() {
            return this.width * this.height;
      }
}
```

E poi semplicemente richiamarlo con l'utilizzo del costrutto **new**: `var rect1 = new Rectangle(2,4);`

3.3) Modificare gli oggetti

Un oggetto è modificabile tramite tre tipi di azioni:

- **Aggiungere** una proprietà

```
rect1.perimeter = (rect1.width + rect1.height) * 2;
```

- **Rimuovere** una proprietà

```
delete rect1.perimeter;
```

- **Iterare** sulle proprietà

```
for(i in rect) {
      alert(i);
}
```

3.4) Per una maggiore efficienza: prototipi

Nell'esempio del rettangolo di prima, ogni rettangolo possedeva una sua copia della funzione area. Ma questo è uno spreco notevole.
Per evitare questa cosa, si ricorre ai **prototipi**.

```
function Rectangle(w, h) {
      this.width = w;
      this.height = h;
}
Rectangle.prototype.area = function() {
      return this.width * this.height;
}
```

ATTENZIONE
I prototipi forniscono inoltre dinamicità. Infatti, se un prototipo viene aggiunto ad una istanza di un certo oggetto, anche tutti gli altri oggetti avranno quel prototipo (anche se il prototipo è stato aggiunto dopo la costruzione delle altre istanze!).

3.5) Ereditarietà

JavaScript supporta anche l'ereditarietà sfruttando la proprietà **prototype** (in questo modo l'oggetto subClass avrà anche la proprietà 'hello'):

```
function superClass() {
      this.hello = "Hello";
}

function subClass() {
      this.bye = "Bye";
}

subClass.prototype = new superClass;
```

3.6) Oggetti built-in

```
//Date
var oggi = new Date();
a = oggi.getDate();
b = oggi.getDay();
c = oggi.getFullYear();
d = oggi.getHours();
e = oggi.getTime();
/*restituisce i millisecondi trascorsi
dalla mezzanotte del 1/1/1970*/
```

```
//Math
a = Math.min(x,y);
b = Math.pow(x,y);
c = Math.abs(x);
d = Math.random();
e = Math.PI;
f = Math.E;
```

```
//String
var a = "Hello world";
a = a.length;
b = a.charAt(n);
c = a.indexOf(substr);
d = a.lastIndexOf(substr);
a.replace(espr1,espr2);
a.split(delimitatore);
```

La reference completa degli oggetti built-in di JavaScript è disponibile al seguente indirizzo:
http://www.w3schools.com/jsref

4) Gli eventi

JavaScript fornisce differenti, radunabili in quattro categorie:
- **Windows events**: onload, onunload
- **Form events**: onchange, onsubmit, onreset, onselect, onblur, onfocus
- **Keyboard events**: onkeyup, onkeydown, onkeypress
- **Mouse events**: onclick, ondblclick, onmousedown, onmouseup, onmousemove, onmouseover, onmouseout

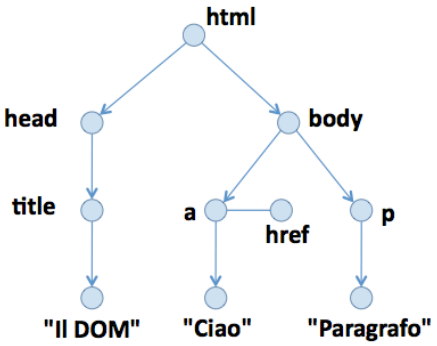
4.1) Esempio: rollover di immagini

```
<html>
   <body>
   <h3>Rollover test</h3>
   <img src="apple1.jpg" align="center"
        onmouseover="document.images[0].src='apple2.jpg'"
        onmouseout="document.images[0].src='apple1.jpg'">
   </body>
</html>
```

5) Il DOM (Document Object Model)

Il DOM è un albero rappresentante la pagina HTML, tramite il quale possiamo accedere alla pagina avendo oggetti che puntano all'elemento/tag desiderato. Ogni elemento è quindi un nodo.

```
<html>
   <head>
      <title>Il DOM</title>
   <head>
   <body>
      <a href="pagina.html">Ciao</a>
      <p>Paragrafo</p>
   </body>
</html>
```



5.1) Le tipologie di nodo

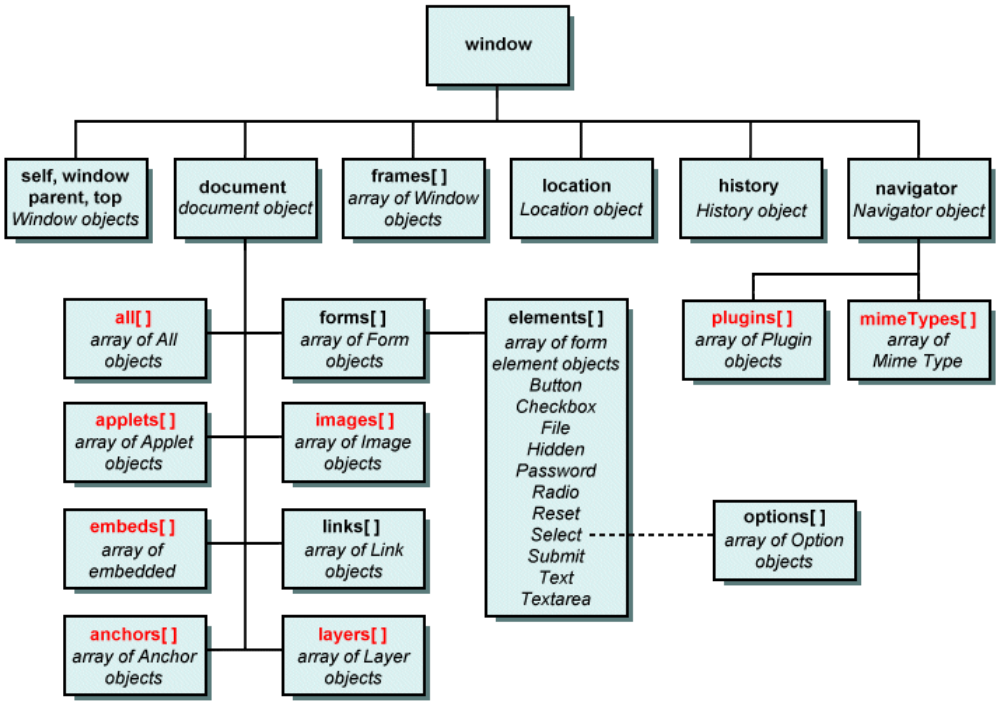
Abbiamo il **document node** che rappresenta il documento.
Ogni elemento HTML è detto **element node**, mentre ogni testo incluso all'interno dei tag è un **text node**.
Gli elementi HTML hanno poi attributi, quindi definiamo gli **attribute node** (href in <a>, ad esempio).
Infine abbiamo i **comment node** che rappresentano ovviamente i commenti.

5.2) Schema degli oggetti in JavaScript

L'albero degli oggetti viene visto da JavaScript in questo modo:
Abbiamo una radice **window** che ha come figli vari oggetti per modellare la pagina, fra cui i più importanti:
- **navigator**: appCodeName, appVersion, cookieEnabled, platform, userAgent
- **screen**: availHeight, availWidth, colorDepth, height, pixelDepth, width
- **history**: back(), forward, go(), length
- **location**: href, hostname, host, protocol, port

Il più importante è l'oggetto **document**, che è ciò che ci da accesso al contenuto vero e proprio della pagina.

Vediamo uno schema riassuntivo di tutti gli oggetti



5.3) Metodi di accesso: dotted notation

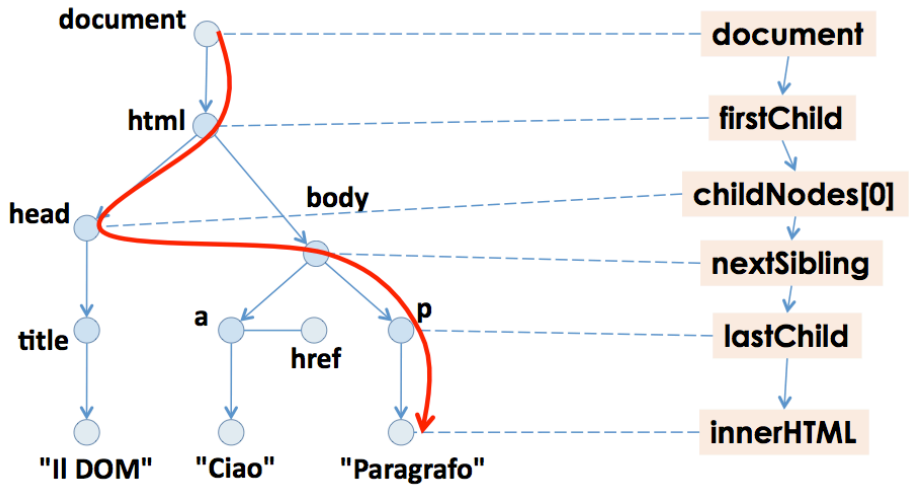
Tramite la dotted notation possiamo percorrere l'albero sopra riportato (è possibile omettere window) e raggiungere quindi qualsiasi elemento.
Ad esempio con `document.images[0]` prendiamo la prima immagine della pagina.

5.4) Metodi di accesso: navigazione albero HTML

Attributo	Descrizione
nodeName	Nome dell'elemento HTML
nodeType	Un numero che ne identifica il tipo (vedere API)
childNodes	Lista dei nodi figli (e.g. childNodes[0] è il primo figlio)
firstChild	Il primo figlio
lastChild	L'ultimo figlio
nextSibling	Il nodo successivo al nodo corrente
previousSibling	Il nodo precedente il nodo corrente
parentNode	Il nodo padre
innerHTML	Il testo contenuto nel tag

Gli attributi (non sono metodi) permettono la navigazione degli elementi.
I tipi di nodo sono:
- 1 = elemento HTML
- 2 = elemento "attributo"
- 3 = elemento "text"
- 8 = commento HTML
- 9 = documento
- 10 = type definition del documento

Esempio



5.5) Metodi di accesso: accesso diretto all'elemento

Esistono altri tre metodi per ottenere l'elemento desiderato:
- **getElementById(**"*ID*"**)**
- **getElementsByTagName(**"*tag*"**)**
- **getElementsByName(**"*name*"**)**

Si noti che gli ultimi due metodi restituiscono una lista, quindi è possibile accedere ai singoli elementi con questa sintassi:

```
getElementsByTagName("a")[0];
```

5.6) Creazione di nuovi nodi

I metodi disponibili per creare nodi sono:
- **document.createTextNode(**"*text*"**)**
- **document.createElement(**"*tagName*"**)**

Per gestire gli attributi invece:
- **setAttribute(**"*name*","*value*"**)**
- **getAttribute(**"*name*"**)**
- **removeAttribute(**"*name*"**)**

Per gestire gli attributi invece:
- **appendChild(***newChild***)**
- **removeChild(***child***)**
- **replaceChild(***newChild, oldChild***)**
- **insertBefore(***newChild, refChild***)**

5.7) Proprietà di oggetti notevoli

```
window.alert()
window.prompt()
window.confirm()
window.open()
window.close()
window.print()
window.moveBy()
window.moveTo()
window.setTimeout()
```

```
window.status // barra di stato
window.defaultStatus
window.opener /*la finestra da cui
         è stata aperta quella
         corrente*/
window.parent
window.closed
window.location // info sull'URL
window.navigator //info sul browser
window.history //back() forward() go(n)
window.document
```

```
document.bgColor
document.fgColor
document.linkColor
document.alinkColor
document.vlinkColor
document.lastModified
document.title
document.referrer
```

6) JavaScript e CSS

Attraverso JavaScript è possibile anche interagire con CSS e modificarne eventualmente le proprietà.

6.1) La proprietà style

Per ottenere un riferimento allo stile di un singolo elemento, (sfruttando il DOM per raggiungere l'elemento stesso), si adopera la proprietà style.
Dopodiché ogni proprietà di CSS è stata rimappata con lo stesso nome per l'utilizzo in JavaScript.
Quindi, volendo accedere alla proprietà color di un certo elemento, scriveremo:

```
elem.style.color = "green";
```

Si noti che per attributi con il trattino, il rimappaggio è stato effettuato usando la tecnica della **gobba di cammello** e quindi background-color è diventato backgroundColor.
Unica eccezione è **float**, che è diventato **cssFloat**.

6.2) I metodi setProperty e getPropertyValue

È possibile adoperare anche altri metodi per accedere alle proprietà:

```
elem.style.setProperty("color", "green");

actualColor = elem.style.getPropertyValue("color");
```

6.3) Accesso al foglio di stile

È oltremodo possibile accedere direttamente ai fogli di stile usando la collezione **document.styleSheet**.
Le regole di ogni foglio di stile sono poi contenute nell'array **cssRules[]**, e volendo aggiungere una regola si può usare **foglioDiStile.insertRule(***"regola", indice***)** e per rimuoverla **foglioDiStile.deleteRule(***indice***)**.
Stampa delle regole:

```
ss = document.styleSheets;

for(i=0; i<ss.length; i++) {
  for(j=0; j<ss[i].cssRules.length; j++) {

      alert( ss[i].cssRules[j].selectorText + "\n" );

  }
}
```

7) Controllo dell'input

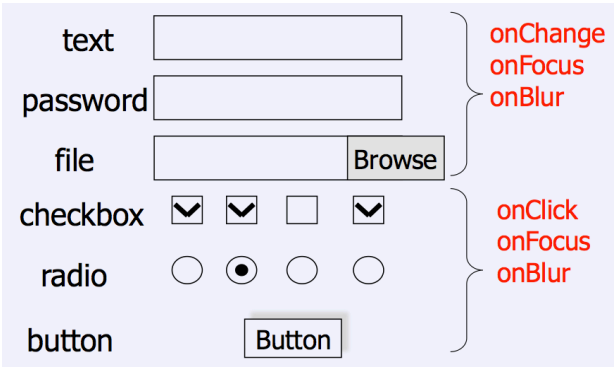
JavaScript è spesso adottato per il controllo dell'input. I dati inseriti dall'utente sono sempre da considerarsi pericolosi ed erronei, quindi richiedono controllo (sia lato client che lato server).

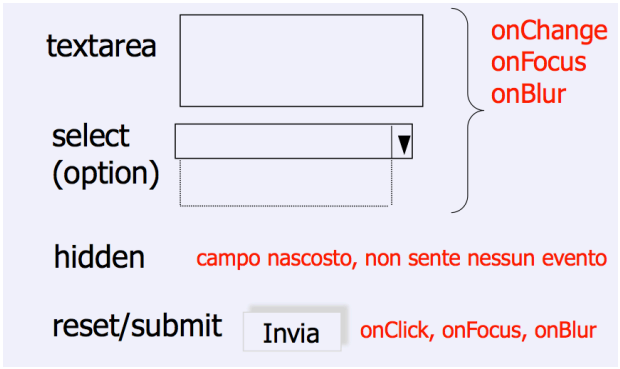
7.1) Le proprietà e gli eventi

Per fare questo si sfrutta l'oggetto form, le cui proprietà sono:

```
document.forms.length
document.forms[].action
document.forms[].method
document.forms[].encoding
document.forms[].name
document.forms[].target
document.forms[].elements[]
```

Dopodiché i vari oggetti del form reagiscono ad alcuni eventi secondo questo schema:





Inoltre, in particolare, gli oggetti **checkbox** e **radio button** hanno anche la proprietà **checked**.
Invece l'oggetto **select** possiede la proprietà **selectedIndex** e **options[]**.

7.2) onSubmit e onReset

Fondamentali sono gli attributi onSubmit e onReset dei form. Assegnandovi dei metodi che restituiscono un booleano, in caso di 'false', non verrà eseguito il submit/reset.

```
<form name="form1" method="post" action="…"
   onSubmit="return conferma_invio();"
   onReset="return conferma_canc();">
   …
</form>
```

7.3) le regExp (espressioni regolari)

Una tecnica molto sofisticata per controllare l'input è l'utilizzo delle espressioni regolari. Possiamo creare un pattern col quale poi potremo "mettere alla prova" la stringa in input, questo tramite i metodi:

- **String.search(pattern)**, restituisce la posizione del primo carattere della sottostringa che coincide col carattere oppure -1
- **String.replace(pattern, str)**, ogni occorrenza di pattern è sostituita da str
- **String.match(pattern, str)**, restituisce un array contente i risultati del confronto

Ecco una carrellata dei pattern adottabili:

/[abc]/	"a","b","c"	/[abc]{n,}/	almeno n caratteri
/[^abc]/	tutti tranne "a","b","c"	/^Java/	"JavaScript" OK (inizio riga)
/[a-z]/	solo caratteri minuscoli	/Java$/	"JavaScript" KO (fine riga)
/[A-Z]/	solo caratteri maiuscoli	/a*/	"","a","aa","aaa", ... (0,1,2,...)
/[0-9]/	solo cifre	/b+/	"b","bb","bbb", ... (1,2,3,...)
/[a-zA-Z0-9]/	stringhe alfanumeriche	/c?/	"", "c" (0,1)
/[a-z]{n}/	esattamente n caratteri	/exp.reg./i	confronto non case-sensitive
/[0-9]{n,m}/	tra n ed m cifre	/exp.reg./g	confronto globale

8) I cookies

Come sappiamo le connessioni HTTP sono stateless ma per simulare una connessione statefull si adoperano i cookies che sono piccoli file di testo adoperabili dal server per salvarsi alcune informazioni relative al client e alla connessione attuale.

I cookies hanno quattro proprietà (opzionali) oltre al **nome**: **expires** (scadenza), **path** (quali pagine hanno accesso al cookie), **domain** (specifica il dominio a cui devono essere abilitati i server per aver accesso al cookie), **secure** (il cookie è trasmissibile solamente in connessioni sicure).

L'oggetto **document** è dotato della proprietà **cookies** che null'altro è che una stringa con un elenco di coppie "attributo" = "valore" separati da ';'.

```
function getCookie(Nome)
{
    var C = document.cookie.split(";");
    for(i = 0; i < C.length; i++)
        if(C[i].indexOf(Nome+"=") != -1)
            return(unescape(
                C[i].substr(Nome.length + 2)));
    return null;
}
function setCookie(Nome, Valore)
{
    document.cookie = Nome + "=" + escape(Valore);
}
```

PHP

1) Script

Possiamo scrivere PHP direttamente nella nostra pagina html usando un nuovo tag, <?php ?>.
Grazie a questa implementazione potremo avere pagine dinamiche, con accesso a database e quindi con la possibilità di fare autenticazioni, ecc.
Il server dovrà disporre di un **modulo php** in grado di tradurre il codice contenuto nello pseudo-tag. La traduzione sostituirà completamente il codice fra i tag.
Quindi il sorgente e il file che arriva al client sono diversi.

2) Il server

Il server non si limita a distribuire risorse, ma fornisce anche supporto con ad esempio file di log e di error.
Sul server ci sono due **importanti radici**: la radice dei documenti e la radice del server.

2.1) La radice dei documenti

- /var/www/html/
- /var/www/cgi-bin/
- /var/www/icons

2.2) La radice del server

- /etc/httpd/
- /etc/httpd/conf dove troviamo **http.conf** per la configurazione
- etc/httpd/logs dove troviamo i file **access_log** e **error_log**
- /var/www/icons

Apache legge i file di configurazione e si forka con processi figli che gestiscono le singole richieste.
In httpd.conf possiamo specificare:

```
MinSpareServers 5 #numero min. processi figli
MaxSpareServers 20#numero max. processi figli
StartServers 8 #numero partenza processi figli
MaxClients 150 #richieste simultanee max
TimeOut 300 #tempo max atteso per risposta
#HTTP 1.1, uso stessa connessione per più pagine:
#richieste per figlio (poi viene terminato)
MaxRequestsPerChild 100
KeepAlive true
MaxKeepAliveRequests 100
KeepAliveTimeout 15
```

3) Core php

3.1) Variabili

Le variabili in php sono sempre precedute da $. I tipi sono i soliti, vediamo alcuni esempi:
- $a = 5;
- $t = TRUE;
- array:
    - $giorni = array("lun", "mar", "mer", "gio", "ven", "sab", "dom");
    - $giorni[0] = "lun", ...

Variabili superglobali

$GLOBALS	$_POST	$_SESSION
$_SERVER	$_FILES	$_REQUEST
$_GET	$_COOKIE	$_ENV

3.2) Assegnamenti

Esistono assegnamenti by value:
- $a = 10

oppure by reference:
- $b = &$a;

3.3) Stringhe

Esistono tre tipologie di stringhe differenti:
Single quoted **'string**'

```
echo 'Stringa';        --> Stringa
echo 'Su più           --> Su più
      righe';              righe
echo 'l/'ape';         --> L'ape
echo 'Due \n linee';   --> Due linee (no simboli di escape)
echo 'var: $a';        --> var: $a (non espande)
```

- Double quoted **"string"**

```
echo "Stringa";        --> Stringa
echo "Due \n linee";   --> Due
                           linee (sì simboli di escape)
echo "var: $a";        --> var: 10 (espande le variabili)
echo "$a"."$b";        --> 10.11 (concatena le variabili)
```

- Heredoc (*EOD; deve stare sulla prima colonna*)

```
$a = <<<EOD
Stringa su più
Righe
Yea
EOD;
```

Funzioni e comodità sulle stringhe

- **strlen($a)** *Restituisce la lunghezza di una stringa*
- **strpos($src, $str)** *Restituisce l'indice della prima lettera di str occorre src (FALSE altrimenti)*
- **stripos($src, $str)** *Come strpos ma non è case sensitive*
- **explode($src, $sep)** *Esplode in parti la stringa src basandosi su sep come stringa delimitatoria*
- **str_replace($c, $r, $s)** *Sostituisce tutte le occorrenze di $c in $s con $r*
- **nl2br($stringa)** *Sostituisce tutti gli "a capo" con
;*
- **htmlentities($stringa)** *Sostituisce tutti i caratteri non standard con l'equivalente HTML (è -> è)*
- **stripslahes($stringa)** *Rimuove tutti i caratteri backslash *

- **$s .= $a** *Equivalente a $s = $s.$a*

3.4) Array

Oltre alle classiche dichiarazioni:
- $giorni = array("lun", "mar", "mer", "gio", "ven", "sab", "dom");
- $giorni[0] = "lun";

abbiamo
- **$giorni[] = "x"** *Accoda all'array giorni la stringa x*
- **$a = array (** *Array associativo key-value (eventualmente annidabile)*

  "nome" => "Mario"

  "cognome" => "Rossi"

  );

3.5) Costrutti

Oltre ai soliti costrutti troviamo il foreach che permette di scorrere gli array associativi:

```
foreach($a as $k => $v) {
      //Ad ogni giro in $k c'è la key e in $v il value
}
```

3.6) Import

Due tipi di import:
- **include_once("nomefile");**
- **require_once("nomefile");**
- **include("nomefile");**
- **require("nomefile");**

Include e require hanno simile funzionamento, ma l'aggiunta di *_once* evita l'import ricorsivo fra i file.

3.7) Altre funzioni

- **count($array)** *Restituisce il numero di elementi in un array*
- **echo parametro** *Stampa (è un costrutto!) nel codice HTML del sorgente*
- **die($messaggio)** *Uccide l'esecuzione*
- **header($stringa)** *Emette una linea nell'header dell'HTML (usata prima di ogni altro output)*

3.8) Costanti

Una costante (si usa senza dollaro!) è definibile:
- **define(NOME_COSTANTE, valore)**

3.9) Dichiarazione di funzioni

Le funzioni si dichiarano secondo il solito pattern:

```
function nomeFunzione ($arg1, $arg2, ..., $argN) {

      return expression;
}
```

Le variabili dichiarate all'interno delle funzioni hanno scope locale tranne se precedute dal costrutto global.
- **global $var1**;

3.10) Gestione di file

- **int fopen("filename", "mode")**
  Restituisce un puntatore al file, FALSE in caso contrario.
  Mode: r, r+, w, w+, a, a+

- **int fclose(fpointer)**
  Chiude un file
- **boolean is_dir()**
- **boolean is_file()**
- **boolean is_link()**
- **boolean is_readable()**
- **boolean is_writable()**

Con fpointer aperto in lettura:
- **string fgetc (fpointer)**
- **string fread (fpointer, length)**
- **boolean feof (fpointer)**

Con fpointer aperto in scrittura:
- **int fwrite(fpointer, string [, length])**
- **boolean fflush(fpointer)**

- **resource opendir(path)** *(resource è un tipo speciale per le directory)*
- **string readdir(dir_handle)** *(legge la directory file per file, FALSE se sono finiti. dir_handle è di tipo resource)*
- **void closedir(dir_handle)**

3.11) Gestione di date

- **date(formattazione, timestamp)**
  La formattazione definisce come organizzare la data, vedi API.
- **time()**
  Restituisce l'attuale timestamp.

Per ottenere la data attuale si adopera quindi date(format, time()) o più semplicemente date(format) (caso speciale).

4) Lavorare con i database

Php offre la possibilità di comunicare con i database. Per fare questo è sufficiente passare attraverso tre semplici fasi:

- Connettersi al DBMS in ascolto.
- Selezionare con quale base di dati dialogare.
- Richiedere l'esecuzione di query sulla base di dati selezionata.

4.1) Connessione al DBMS

Viene usata la funzione **mysql_connect()** che restituisce un identificativo di connessione MySQL oppure FALSE se la connessione fallisce.
Sintassi: [*resource*] **mysql_connect(***nome_host*, *nome_utenteDB*, *password_utenteDB*, *nuova_connessione***)**.
Di default il parametro nuova_connessione è a FALSE (default), ciò significa che ogni qualvolta si richiami la funzione mysql_connect() con gli stessi parametri viene restituito lo stesso **resource** quindi non viene aperta una nuova connessione, mentre invece con nuova_connessione a TRUE ogni chiamata apre una nuova connessione.

Tutti i parametri sono opzionali, se non si specifica il *nome_host* di default è impostato localhost:3306.
Dopodiché con **mysql_close(***resource***)** si chiude la connessione corrispondente a *resource*.

4.2) Selezione della base di dati

Una volta stabilita correttamente la connessione al DBMS, si sceglie su quale database operare tramite la funzione **mysql_select_db()**.
Sintassi: [*boolean*] **mysql_select_db(***nome_database*, *resource***)**
Oltre al nome del database è eventualmente specificabile la connessione *resource* al DBMS. Se non viene specificata alcuna connessione, allora verrà selezionata l'ultima aperta.

4.3) Esecuzione delle query

Si può quindi procedere all'esecuzione della query usando la funzione **mysql_query()**.
Sintassi: [*resource*] **mysql_query(***query, resource, modo_risultato***)**
Oltre ad esprimere la query possiamo specificare opzionalmente la resource di connessione (la query verrà eseguita sul DB associato a quella resource).

mysql_query() restituisce una risorsa oppure TRUE (dipende dalla query) o in alternativa FALSE se la query è mal composta.

4.4) Gestire i risultati delle query

Abbiamo diverse funzioni per trattare i risultati (*result* è il ritorno di una mysql_query()):
- **mysql_num_rows(***result***)** restituisce il numero di righe della relazione risultato esplicitata tramite la query.
- **mysql_affected_rows(***result***)** restituisce il numero di righe della relazione coinvolte nell'operazione (usato per operazioni che modificano la tabella ad ES: DELETE, INSERTE, REPLACE, UPDATE).
- **mysql_fetch_array(***result***)** carica sotto forma di array associativo la prima riga risultato della query. Se eseguito nuovamente prende la seconda riga risultato.
- **mysql_fetch_row(***result***)** carica sotto forma di array associativo la prima colonna risultato della query. Se eseguito nuovamente prende la seconda colonna risultato.

4.5) Un esempio di connessione

```
$connessione = mysql_connect("localhost:8200", "enricomensa", "passwordbanale")
      or die ("Connessione non riuscita: " . mysql_error());

echo "Connesso con successo <br><br>";

mysql_select_db("dbenricomensa", $connessione)
      or die ("Selezione del database non riuscita");

//facciamo una query
$sql = "SELECT codice, descrizione FROM prodotti";
$result = mysql_query($sql); //risultato del db

for($i = 0; $i < mysql_num_rows($result); $i++) {
      $array = mysql_fetch_array($result);
      print "CODICE: ".$array['codice']."<br>";
      print "DESCRIZIONE: ".$array['descrizione']."<br>";
}
```

5) Cookies e sessioni

Dato che il protocollo HTTP è **stateless** sono necessarie alcune tecniche per "mantenere lo stato" e tenere traccia delle interazioni precedenti.

5.1) Le query string

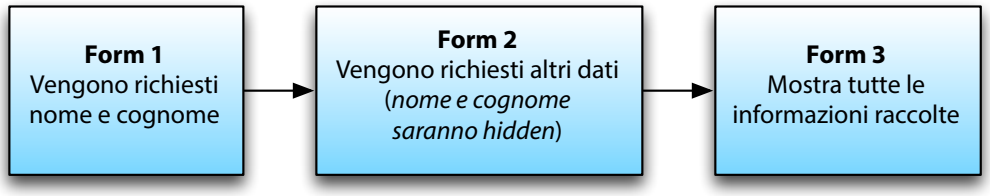
Quando chiamiamo una pagina php possiamo anche aggiungere alcuni parametri tramite i quali passare informazioni dalla pagina attuale a quella chiamata.
Ad esempio **http://script.php?var1=5&var2=ciao** passerà alla pagina script.php la variabile 'var1' con valore '5' e la variabile 'var2' con valore 'ciao'.
Prima di aggiungere/prelevare i parametri nella/dalla query string è necessario adoperare i metodi **urlencode** e **urldecode**.
Vediamo un semplice esempio:

```
<?php
# ex_state.php
$testo="Caspita, che bella giornata!";
$testo=(urlencode($testo));
echo "<a href=\"get_info.php?testo=$testo\">Vai!</A>";
?>

<?php
# get_info.php
echo "Ecco le informazioni: ".$_GET["testo"];
?>
```

5.2) Gli hidden fields

Supponiamo di avere ad esempio più form in sequenza: per poterci portare i campi da un form all'altro potremmo sfruttare dei campi hidden.



5.3) I cookies

Come già visto in JavaScript i cookies sono porzioni di informazione salvati sul client utili al server per i più svariati motivi.
I cookies su php sono molto più semplici da utilizzarsi.

- Creare un cookie: **setcookie(***name*, *value*, *expire*, *path*, *domain*, *secure*, *httponly***)**

  Path e domain sono parametri che definiscono rispettivamente il path e il dominio di validità. Con secure si indica che il cookie andrebbe trasmesso con HTTPS. Questa funzione deve essere eseguita prima di qualsiasi output.

- Accedere a un cookie: **$_COOKIE["***nome_cookie***"]**

  Per accedere a un cookie si sfrutta la variabile superglobale $_COOKIE.

- Eliminare a un cookie: **setcookie(***name*, *value*, *-expire*, *path*, *domain*, *secure*, *httponly***)**

  Assegnando a un cookie un nuovo valore di expire (negativo) praticamente lo si fa scadere prima del tempo e quindi l'effetto è quello di cancellazione.

Un esempio di utilizzo dei cookies:

```
<?php
setcookie ("test_cookie","niente di particolare",time() +43200,"/");
# cookie.php
echo "<HTML>";
echo "<BODY>";

if (isset($_COOKIE["test_cookie"])){
      echo "Ciao cookie, i tuoi contenuti sono: $_COOKIE ["test_cookie"]";
} else {
      echo "Non ho trovato alcun cookie con il nome test_cookie";
}

 echo "</BODY>"; echo "</HTML>";
?>
```

5.4) Le sessioni

Ogni sessione in PHP è identificata da un id, detto SID (Session IDentifier).
Il browser comunicherà al server il suo SID tramite query string oppure tramite cookies.
Vediamo le funzioni per gestire le sessioni:

- **session_start(***nome***)**

  Crea una nuova sessione (se è già aperta una sessione con nome *nome* allora semplicemente restituirà il SID di quella sessione già aperta).

- **session_id()**

  Restituisce il SID della sessione aperta.

- **session_name()**

  Restituisce il nome della sessione aperta.

- **$_SESSION[“***var***”]**

  Recupera la variabile *var* della sessione aperta.

- **unset($_SESSION[“***var***”])**

  Deregistra la variabile di sessione *var* della sessione aperta.

- **session_unset()** oppure **$_SESSION = array()**

  Deregistra tutte le variabili di sessione della sessione aperta.

- **session_destroy()**

  Chiude la sessione senza deregistrare le variabili della sessione stessa. Quindi facendo un session_start() si può recuperare la sessione.

- **session_get_cookie_params()**

  Restituisce un array contente i valori (i parametri) della funzione setcookie() che ha implementato la sessione (infatti la sessione si basa comunque su cookies!)