Guida base

A cura di Aldo Guastafierro

Sommario

Premessa

Questa guida dedicata a HTML 5 vuole essere una guida di aiuto con basi introduttive, della versione 5 che di fatto sta rivoluzionando il web.

L'uso di HTML5, a differenza delle versioni precedenti, esige il rispetto di regole che sono decisamente più rigide di quelle usate nelle precedenti versioni, oltre alla compatibilità dei browsers moderni che lo supportino completamente.

HTML5 è uno standard ufficialmente approvato dal W3C e introduce molte nuove istruzioni che con la loro semantica rendono un documento web correttamente leggibile, e quindi perfettamente interpretabile, da altri sistemi.

A tal proposito è sempre utile fare in modo dei testare un sito HTML5
con le versioni più aggiornate dei Browsers per verificare se:

il tuo browser supporta canvas e si possono disegnare forme?

sul tuo browser si possono vedere i filmati video?

sul tuo browser si possono memorizzare dati?

dal tuo browser si può capire dove si trova?

il tuo browser supporta la navigazione off line?

HTML5

(Hyper Text Markup Language) è formato da definizioni, elementi (tags), questi impartiscono al browser (il programma usato per la navigazione) delle direttive ben precise al fine di ottenere il risultato a video, una sorta di interpretazione che varia a volte a seconda del tipo di browser adoperato. Si tratta dunque di conoscere il significato dei vari elementi e come questi possono essere combinati fra loro per avere la padronanza del linguaggio.

Per la stesura di questo metalinguaggio si fa uso di un normale editor per testi, meglio se un editor specifico per HTML.

La struttura di un documento HTML5

è composta da un contenitore <html> e due sezioni ben distinte: una parte iniziale denominata testa <head> ed una parte centrale denominata corpo <body>

Nella testa (head) vanno inseriti soltanto quei comandi che forniscono informazioni ai motori di ricerca, quali: meta comandi, titolo, ecc. oltre a tutto il codice che deve essere letto con una certa priorità: script, dichiarazioni di style, ecc. ecc.
Nel corpo (body) va inserito tutto il resto. Tutti questi elementi necessitano del relativo elemento di chiusura: </html>, </head> e </body>

Riassumendo: una struttura tipo in HTML5 potrebbe essere questa:

```
<!DOCTYPE html>
<html lang="it">
  <head>
<meta charset="xxxxx">
codice per il server: meta comandi, script, fogli di style, ecc.
  </head>
  <body>
Codice per visualizzare: testo, immagini, moduli, ecc.
  </body>
</html>
```

HTML5 introduce nuovi tag, ne abolisce altri che considera obsoleti, semplifica alcune regole ma ne complica altre perchè pretende il rispetto della semantica di quel determinato tag, in parole povere un tag creato per una determinata funzione farà soltanto quella senza alcun compromesso, gli attributi, e qualsiasi altra cosa relativa alla sezione di quel tag, vengono demandati ai fogli di style che con HTML5 non sono più opzionali ma diventano parte integrante del codice.

Vediamo subito nel listato sopra la prima differenza rispetto ad HTML e/o XHTML precedenti: la riga di prologo, il doctype, viene semplificato e ridotto drasticamente, da un ipotetico:

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

Si passa ad un semplicissimo:

```
<!DOCTYPE html>
```

Il tag html richiede l'attributo lang che forse in pochi usavano con html precedenti, serve a specificare in quale lingua è scritta la pagina web, nel nostro esempio it sta per Italiano.

Nuovi tag HTML5

HTML5 introduce nuovi tags, vediamone alcuni che servono per definire la struttura, e sono: article, aside, footer, header, hgroup, mark, nav, section, time.

Questi nuovi tags, a differenza di: html, body ed head, possono essere ripetuti più volte nella stessa pagina dal momento che non definiscono la struttura vera e propria ma servono a delimitare delle zone all'interno della pagina stessa, immaginiamo ad esempio la pagina web di un blog, per ogni nuova notizia ci sarà un header ed un footer (testa e piede) ma saranno riferiti a quella notizia e non all'intera pagina, quindi:

```
<!DOCTYPE html>
<html lang="it">
  <head>
<meta charset="xxxxx">
codice per il server: meta comandi, script, fogli di style, ecc.
  </head>
  <body>

<header> Intestazione sezione del documento </header>
<footer>Terminazione sezione del documento </footer>


<header> Intestazione sezione del documento </header>
<footer>Terminazione sezione del documento </footer>

  </body>
</html>
```

Più avanti saranno brevemente spiegati tutti quanti, per il momento è sufficiente capire il meccanismo e la struttura. I più attenti si saranno accorti di un meta tag relativo al charset, per altro molto semplificato rispetto a quello usato con html precedenti. Questo meta è importantissimo perchè definisce la codifica dei caratteri usati, per esempio: utf-8 dove non serve più inserire *http-equiv* e *content*, anche se continua ad esserci compatibilità ed html5 riconoscerebbe le forme estese dei metatags usati in versioni precedenti.

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

Questo meta comando va inserito come primo tag subito dopo head, per un discorso legato ad un aspetto tecnico di certi browser che hanno una capienza molto ristretta (512 byte) per bufferizzare le istruzioni, ne consegue che potrebbe non essere letto o non essere gestito correttamente se inserito in altro punto del listato. Il charset specifica la codifica dei caratteri permettendo così di usare caratteri speciali come le vocali accentate, simboli e lettere usate per altre lingue senza creare alterazioni.

Lo stesso paragrafo sopra senza la codifica che riconosce le vocali accentate della nostra lingua italiana:

Pi avanti saranno brevemente spiegati tutti quanti, per il momento sufficiente capire il meccanismo e la struttura. I pi attenti si saranno accorti di un meta tag relativo al charset, per altro molto semplificato rispetto a quello usato con html precedenti. Questo meta importantissimo perch definisce la codifica dei caratteri usati, per esempio: utf-8 dove non serve pi inserire *http-equiv* e *content*, anche se continua ad esserci compatibilit ed html5 riconoscerebbe le forme estese dei metatags usati in versioni precedenti.

All'interno di <head> metteremo il titolo, il richiamo ad uno script: *modernizr.js* che è la libreria che serve per testare se il browser che visita le nostre pagine è in grado di supportare certe funzioni di html5, il richiamo al foglio di style esterno: *foglio.css* ed una dichiarazione di style interna, il tutto anche per mostrare come sono stati semplificati i tags per richiamare queste parti di codice rispetto ad html precedenti.

```
<!DOCTYPE html>
<html lang="it">
  <head>
<meta charset="utf-8">
<title>La mia prima pagina HTML5</title>
<script src="modernizr.js"></script>
<link rel="stylesheet" href="foglio.css">
<style>
header, footer, article {display: block;}
</style>
  </head>
  <body>

<header><h1>Intestazione sezione del documento</h1></header>
<article><p>Testo dell'articolo</p></article>
<footer>Terminazione sezione del documento </footer>


<header><h1>Intestazione sezione del documento</h1></header>
<article><p>Testo dell'articolo</p></article>
<footer>Terminazione sezione del documento </footer>

  </body>
</html>
```

```
/* Foglio di style esterno */

p {font-family: Verdana, sans-serif;}
h1 {font-size: 12px; font-weight: bold;}
```

Piccola nota: chi avesse fatto uso di xhtml dove alcuni tags, come quelli dei meta o del link rel, venivano chiusi sulla stessa riga dalla barra finale: /> in HTML5 restano validi ma non più obbligatori, in pratica è possibile farne uso o meno senza che vengano considerati errore, che ci siano o no va bene ugualmente, stessa cosa dicasi per
 o

Nuovi Elementi Semantici della Struttura HTML5

Elementi o Tags: <section> <nav> <article> <aside> <hgroup> <header> <footer>

Prima di passare in rassegna alcuni nuovi elementi semantici di HTML5 ci sofermiamo su questo termine "**semantico**" che a prima vista sembra non sortire alcun effetto pratico, ed infatti con o senza questi nuovi elementi l'aspetto estetico delle nostre pagine web non è cambiato molto, in alcuni casi direi proprio per niente. Ma allora a che cosa serve rispettare la semantica di un elemento e perchè a questa viene data tanta importanza?

Chi fa pagine web sa che fino ad oggi soltanto gli elementi <h1> ... <h6> usati per i titoli, creavano strutture a blocchi a livello di DOM, dando alla pagina web una struttura visibile anche da sistemi diversi dai normali browser, come ad esempio quelli usati dai non vedenti. L'introduzione dei CSS ha permesso di rivoluzionare questa struttura potendo assegnare ai vari elementi attributi tali da renderli a loro volta blocchi o comunque elementi diversi da quello per i quali erano stati pensati, e così ognuno ha impaginato dati come meglio credeva, l'aspetto estetico era più che gradevole e nessuno si è mai preoccupato di andare oltre. HTML5 invece si preoccupa proprio di questo aspetto, creare nuovi elementi per definire blocchi che un browser è in grado di interpretare come tali a prescindere dagli attributi impostati nel suo foglio di style.

<section>...</section>

L'elemento section demarca un insieme di contenuti fra di loro correlati. Rappresenta una sezione generica del documento.
Viene dunque usato per i capitoli, le varie sezioni di una tesi, la stessa home page potrebbe essere divisa in sezioni: introduzione, notizie, contatti.

```
<article>
<hgroup>
<h1>Web-Link</h1>
<h2>tutto per fare web</h2>
</hgroup>
<p>Guide utili per fare pagine web.</p>
<section>

<h1>Guida HTML</h1>
<p>La guida di riferimento per il linguaggio html</p>

</section>
<section>

<h1>Guida CSS</h1>
<p>Una valida integrazione ad html.</p>
```

```
</section>


</article>
```

<nav>…</nav>

L'elemento <nav> demarca una serie di link utili per la navigazione.

Viene dunque usato per raccogliere blocchi di links come i menù o qualsiasi altro collegamento utile alla navigazione, non necessariamente tutti i links devono essere contenuti in questo tag.

```
<nav>

<ul>
<li><a href="intro.html">Introduzione guida html5</a></li>
<li><a href="tags.html">I nuovi elementi di html5</a></li>
<li><a href="obsoleti.html">Gli elementi deprecati in html5</a></li>
</ul>

</nav>
```

<article>…</article>

L'elemento <article> demarca un articolo, un'informazione, un contenuto che sia indipendente dal resto del documento.
Viene dunque usato per racchiudere tutte quelle informazioni che possono essere indipendenti dal resto del documento e come tali essere riprese e riutilizzate per conto proprio.

```
<article>

<header>
<h1>Guida HTML5</h1>
<p>Di Aldo Guastafierro – linguaggi_web.it</p>
</header>
<aside>
<p>visita il mio sito per maggiori informazioni</p>
</aside>

</article>
<article>

<header>
<h1>Guida CSS</h1>
<p>Di Aldo Guastafierro</p>
</header>
```

```
<aside>
</aside>

</article>
```

<aside>...</aside>

L'elemento <aside> demarca un contenuto di poca importanza rispetto al documento, se ne potrebbe fare a meno senza incidere col resto del documento stesso.
Viene dunque usato per note o commenti non importanti, per sezioni laterali, per annunci pubblicitari e qualsiasi altro contenuto che si ritiene slegato dal contenuto principale della pagina stessa.

```
<aside>

<p>visita il sito di Aldo Guastafierro per maggiori informazioni</p>

</aside>
```

<hgroup>...</hgroup>

L'elemento hgroup funge da contenitore per due o più elementi di intestazione correlati fra loro.

Viene dunque usato per raggruppare più elementi di intestazione quando ci sono diversi livelli di titoli dal primo al sesto livello: <h1>...<h6> come sottotitoli, ecc. assegnando così una sola intestazione di titolo più importante.

```
<hgroup>

<h1>Guida HTML5</h1>
<h2>Di Aldo Guastafierro </h2>

</hgroup>
```

<header>...</header>

L'elemento header demarca l'intestazione di una sezione.

Viene dunque usato per elementi introduttivi o di supporto al documento, può essere ripetuto dal momento che in un documento possono esserci più sezioni e di conseguenza più intestazioni, basti pensare ad un blog, per esempio, dove ogni notizia ha la sua intestazione.

```
<header>

<h1>Guida HTML5</h1>
<p>Di Aldo Guastafierro</p>
```

```
</header>
```

<footer>...</footer>

L'elemento footer demarca il pie di pagina o di sezione.

Viene dunque usato per parti di codice che di solito si inseriscono a fine pagina ma anche a fine sezione nel caso in cui una pagina avesse più di una sola sezione. Per esempio le note sull'autore, i links di riferimento, ecc.

```
<footer>
<p>© Aldo Guastafierro</p>
</footer>
```

I nuovi Tags introdotti da HTML5

Elementi o Tags: <time> <mark> <figure> <figcaption> <ruby> <rt> <rp> <progress> <meter> <wbr>

Nuovi elementi

HTML5 ha introdotto nuovi elementi, alcuni già visti nel capitolo precedente (elementi semantici). Qui ne vediamo altri che non riguardano la struttura ma che servono a migliorare e potenziare i contenuti.

<time>...</time>

L'elemento time demarca la data e l'ora su un orologio con formato 24 ore. Viene dunque usato per esprimere data ed ora, se non è inserito all'interno di un elemento <article> farà riferimento alla data e all'ora dell'intero documento. Ha attributi opzionali come datetime che riporta data ed ora nel formato ISO. Usando questo elemento sarà possibile passare il dato ad un sistema automatico e renderlo visibile per l'utente.

```
<time datetime="2013-01-13"> Domenica 13 Gennaio 2013 </time>
```

La data viene espressa inserendo prima l'anno (2013), poi il mese (01), ed infine il giorno (13) separati dal trattino - (meno).

Volendo aggiungere anche l'ora si dovrà inserire a fine data una T seguita da ore, minuti e fuso orario separati dai : (due punti).

```
<time datetime="2013-01-13T15:07+01:00"> alle ore 15:07 </time>
```

Questo il risultato:

Domenica 13 Gennaio ore 15:07

<mark>...</mark>

L'elemento mark demarca una parte di testo del documento.
Viene dunque usato per evidenziare parti di testo che per qualche motivo necessitano di essere poste in evidenza, alcuni browser mostrano questo testo in grassetto, altri lo evidenziano pennellandolo di giallo.

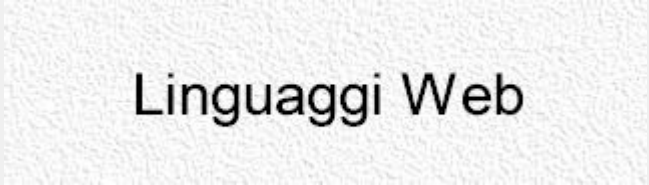
```
<p>Guida HTML5 di <mark> Aldo Guastafierro </mark>linguaggi web</p>
```

Questo il risultato:

<figure>…</figure>

L'elemento figure demarca un insieme di contenuti autonomi quali: immagini grafiche, fotografie, listati, diagrammi ecc. che possono essere anche spostati su altre pagine o parti diverse della stessa pagina perchè considerati di poca importanza e di conseguenza non alterano il senso del testo che li citava.

```
<figure>
<img src="linguaggi_web.png" alt="Forum di web-link">
</figure>
```



Anche parti di codice che non incidono col contenuto della pagina

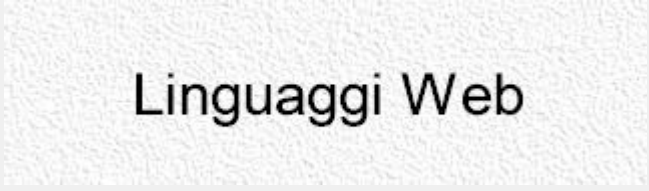
```
<figure>
Questa pagina è esclusiva proprietà di © Aldo Guastafierro
</figure>
```

Questa pagina è esclusiva proprietà di © Aldo Guastafierro

<figcaption>...</figcaption>

L'elemento **figcaption** serve per inserire una didascalia, un titolo, una descrizione e deve essere necessariamente inserito all'interno dell'elemento figure visto sopra dal momento che questa didascalia riguarda solo ed esclusivamente quanto contenuto in <figure>.

```
<figure>
<img src="linguagi_.png" alt="linguaggi web">
<figcaption> Logo del sito linguaggi web <figcaption>
</figure>
```



Logo del sito linguaggi web

Più immagini ed una sola didascalia

```
<figure>
<img src="foto1.png" alt="Foto1"><img src="foto2.png" alt="Foto2"><img
src="foto3.png" alt="Foto3">
<figcaption>Una sola descrizione per le tre immagini sopra<figcaption>
</figure>
```

Una sola descrizione per le tre immagini sopra.

<ruby>…</ruby> <rt>…</rt> <rp>…</rp>

L'elemento **<ruby>** in giapponese - furigana e romaji, insieme a **<rt> e <rp>**servono per il supporto alle lingue orientali, servono per indicare la pronuncia corretta, li cito solo per informazione ma ritengo in questo contesto del tutto inutile approfondirli ulteriormente. <rt> acronimo di Ruby Text è utilizzabile all'interno di ruby per contrassegnare gli ideogrammi ruby. <rp> acronimo di Ruby Parenthesis è utilizzabile all'interno di ruby per circoscrivere le annotazioni ideogrammi ruby.

```
<ruby>
?????<rp>(</rp><rt>?????</rt><rp>)</rp>?????
</ruby>
```

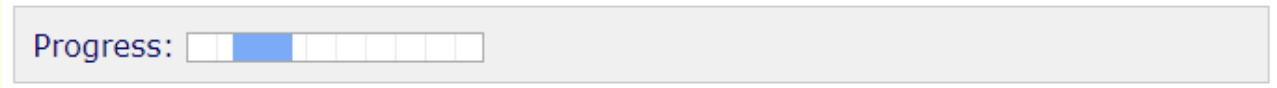
Al posto dei punti interrogativi ????? ci saranno gli ideogrammi della lingua utilizzata.

<progress>...</progress>

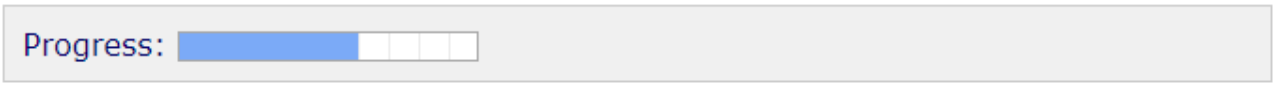
L'elemento **<progress>** indica il progresso per il completamento di una attività, per esempio il caricamento di un file. Potrebbe però non essere conosciuto il suo valore per via di diversi fattori, per esempio essere in attesa di collegarsi al server dove si trova il file.

Ci sono due attributi che determinano il completamento dell'attività: value specifica la quantità di attività che è stata completata, max specifica quanto viene richiesto in totale. Il valore scritto all'interno del tag serve per i browser che non supportano progress.

```
<section>
Progress: <progress max="100">0%</progress>
</section>
```




```
<section>
Progress: <progress value="60" max="100">60%</progress>
</section>
```



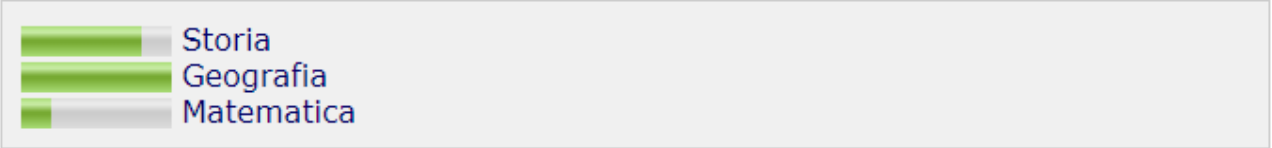
E' possibile lasciare a javascript il compito di calcolare la progressione con questo script che è l'esempio ufficiale del W3C

```
Progress: <progress id="p" max=100> <span>0</span>%</progress>
<script>
var progressBar = document.getElementById('p');
function updateProgress(newValue) {
progressBar.value = newValue;
progressBar.getElementsByTagName('span')[0].textContent = newValue;}
</script>
```

<meter>...</meter>

L'elemento <meter> indica il valore di un indicatore quando i suoi valori di min.e max. sono conosciuti, non va confuso con progress visto sopra e di conseguenza non accetta valori in percentuale. Ci sono tre attributi per determinare la scala ed il suo valore: value, min , e max. Il valore scritto all'interno del tag serve per i browser che non supportano meter.
Per fare un esempio, si vogliono rappresentare graficamente 3 votazioni da 1 a 10

```
<section>
<meter min="0" max="10" value="8"> 8 di 10 </meter> Storia
<meter min="0" max="10" value="10"> 10 di 10 </meter> Geografia
<meter min="0" max="10" value="2"> 2 di 10 </meter> Matematica
</section>
```



<wbr>

L'elemento **<wbr>** (Word Break) inserito in una parola lunga indica il punto in cui è possibile ritornare a capo nel caso in cui la visualizzazione di quel testo fosse all'interno di una finestra del browser più ristretta rispetto a quella da noi prevista. Se infatti si prova a stringere o allargare la finestra del browser si noterà come i contenuti si ridispongano ogni volta per riempire tutto lo spazio, ci sono parole o frasi che vengono suddivise per il ritorno a capo in modo del tutto casuale, con <wbr> è invece possibile indicare al browser il punto in cui spezzare per tornare a capo.

```
<p> Nel caso di un forzato ritorno a capo, la parola http://<wbr>Web-Link.it potrebbe essere spezzata, seguendo le direttive impartite con <wbr></p>
```

Nel caso di un forzato ritorno a capo, la parola http://Web-Link.it può essere spezzata, seguendo le direttive impartite con <wbr>

Quindi: Nel caso di un ritorno
a capo, la parola http://
Web-Link.it può essere spezzata

Nuovi elementi di HTML5 per i Forms

Nuovi Elementi Form: <datalist> <keygen> <output>

HTML5 ha introdotto nuovi elementi anche per i moduli o forms, a dire la verità di alcuni se ne sentiva davvero il bisogno visto che per ottenere quello che adesso è possibile fare specificando un semplice attributo prima lo si otteneva con diverse righe di codice javascript. Vorrei ricordare che in questa guida vengono mostrati soltanto quelli che sono una novità per i forms di HTML5, dando per scontato che voi conosciate già anche gli altri di HTML precedenti, se così non fosse, su web-link ci sono guide per tutti i livelli di conoscenza, principianti compresi, grazie alle quali sarà possibile comprendere al meglio lo scopo ed il significato di questi nuovi elementi introdotti da HTML5.

Di seguito la descrizione con esempi dei nuovi elementi usabili nei forms. Nelle attuali versioni di alcuni browser non tutti sono riconosciuti.

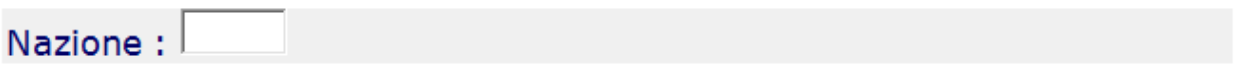
<datalist>...</datalist>

L'elemento **<datalist>** propone una lista di dati che è possibile selezionare con un click per inserirli in un campo input, potrebbe essere considerato anche un attributo oltre che ad un elemento vero e proprio dal momento che funziona solo se vi è un input a cui fare riferimento.

Nell'esempio sono stati inseriti alcuni nomi di nazioni, provate ad iniziare a digitare il nome della vostra, se è presente nella lista (datalist) questa vi viene proposta permettendovi di selezionarla, con un doppio click si mostrano tutte le voci, non viene obbligata la scelta, è infatti possibile inserire anche un nome non presente nella datalist.

```
<input list="nazione">
<datalist id="nazione">
<option value="Italia">
<option value="Francia">
<option value="Inghilterra">
<option value="Germania">
<option value="Spagna">
<option value="America">
</datalist>
```

Questo il risultato:



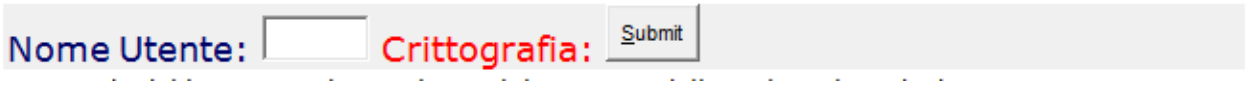
Provate ad inserire una lettera, poi due, e così via, noterete come le proposte della lista cambiano fino a lasciare soltanto quella che ha i requisiti giusti.

<keygen>

L'elemento **<keygen>** permette un modo sicuro per autenticare un utente. Questo elemento genera due chiavi crittografate nel momento dell'invio dati, una pubblica ed una privata. La chiave privata viene memorizzata nel dispositivo client mentre la chiave pubblica viene invece inviata al server e può essere poi riutilizzata in futuro per autenticare l'utente in possesso dell'altra chiave privata.

```
Nome Utente: <input type="text">
Crittografia: <keygen>
```

Questo il risultato:



A seconda del browser utilizzato è possibile operare delle scelte sul grado di sicurezza.

<output>...<output>

L'elemento **output** esegue dei calcoli e ne mostra il risultato.

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">
A<input type="number" name="a" value="50">+
B<input type="number" name="b" value="50">=
<output name="x" for="a b"></output>
</form>
```

Questo il risultato:

A + B =

questo sotto per i browser che accettano range:

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">0
<input type="range" name="a" value="50">100 +
<input type="number" name="b" value="50">=
<output name="x" for="a b"></output>
</form>
```

0 100 + =

Provate questi esempi di output con il browser Opera o Safari, e resterete piacevolmente sorpresi.

Input Type di HTML5 per i Forms

Nuovi tipi di input (input type):

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

Nuovi Type per Input.

HTML5 ha introdotto nuovi type di input per i moduli o forms, a dire la verità di alcuni se ne sentiva davvero il bisogno, peccato che facendo delle verifiche non tutti i browser li supportano. Provate ad immaginare qual'è il browser che più di tutti gli altri non vuol saperne di queste novità? Proprio lui l'ormai famigerato IE, e poco importa se è arrivato alla versione 10. Vorrei ricordare che in questa guida vengono mostrati soltanto i type che sono una novità per gli input di HTML5, dando per scontato che voi conosciate già anche gli altri di HTML precedenti, se così non fosse, su web-link ci sono guide per tutti i livelli di conoscenza, principianti compresi, grazie alle quali sarà possibile comprendere al meglio lo scopo ed il significato di questi nuovi elementi introdotti da HTML5.

Di seguito la descrizione con esempi dei nuovi type usabili con input nei forms che per default è text, libero testo. Nelle attuali versioni di alcuni browser non tutti sono riconosciuti.

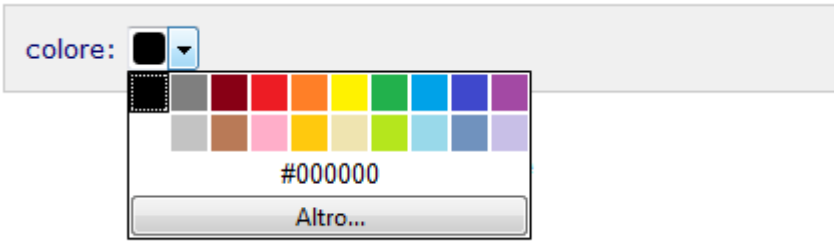
color

Il type color permette di selezionare un colore da una tavolozza di colori predefiniti o direttamente dal suo codice in #esadecimale.

```
<input type="color">
```

Questo il risultato:

colore:



color è supportato dai browser FireFox Opera e Chrome.

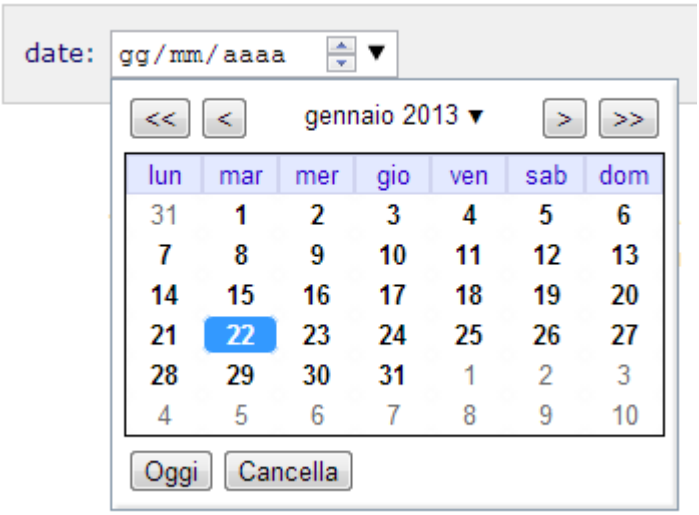
date

Il type **date** permette di selezionare la data direttamente da un calendario grafico generato dal browser.

```
<input type="date">
```

Questo il risultato:

date:



date è supportato dai browser Opera, Chrome e Safari, anche se in modi leggermente differenti, per esempio Opera, a differenza di Chrome, una volta selezionata la data col click, questa viene introdotta nel box nel formato americano: anno, mese e giorno.

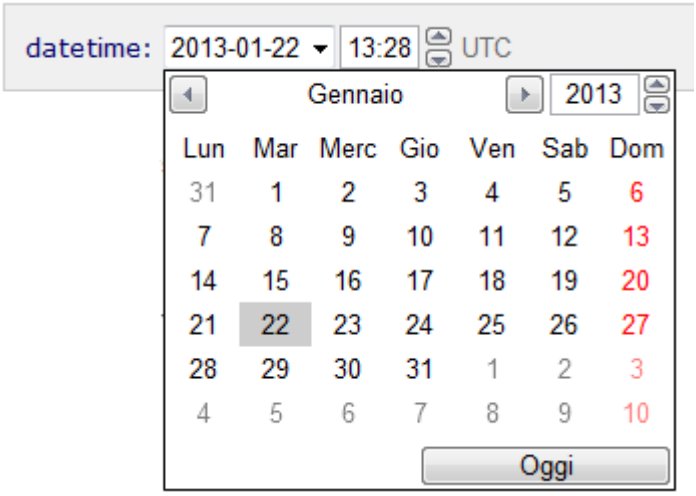
datetime

Il type **datetime** permette di selezionare la data e l'ora con l'aggiunta dell'informazione del fuso orario.

```
<input type="datetime">
```

Questo il risultato:

datetime:



datetime è supportato dai browser Opera e Safari, anche se in modi leggermente differenti, per esempio Safari scrive all'interno del box i riferimenti del fuso orario.

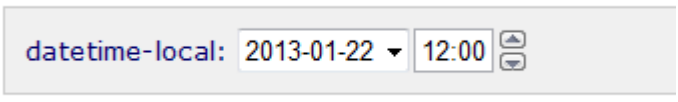
datetime-local

Il type **datetime-local** permette di selezionare la data e l'ora locale, cioè senza informazioni sul fuso orario presente in datetime.

```
<input type="datetime-local">
```

Questo il risultato:

datetime-local:



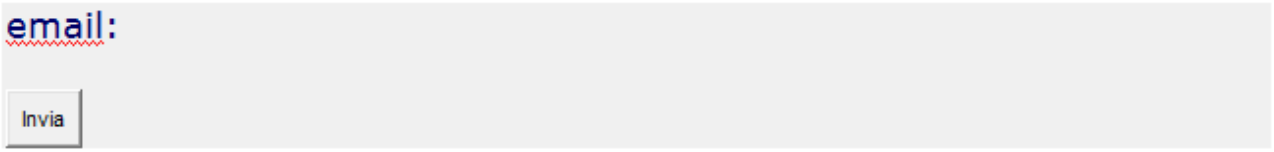
datetime-local è supportato dai browser Opera e Safari.

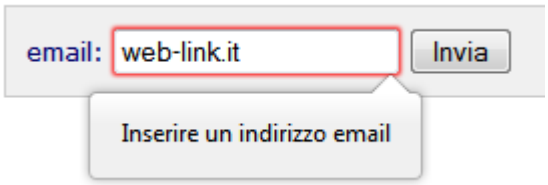
email

Il type **email** permette di inserire un indirizzo email, al suo invio viene eseguita una validazione.

```
<input type="email">
```

Questo il risultato:





Provate ad inserire una qualsiasi cosa priva della chiocciola che contraddistingue un indirizzo email valido.

email è supportato dai browser FireFox Opera Chrome.

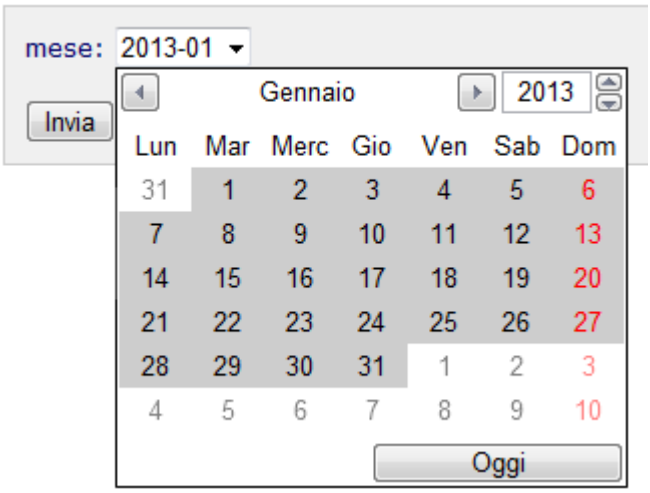
month

Il type **month** permette di selezionare il mese e l'anno direttamente da un calendario grafico generato dal browser.

```
<input type="month">
```

Questo il risultato:

mese:



month è Supportato dai browser Opera e Safari.

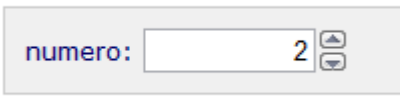
number

Il type **number** permette di selezionare un numero anche tramite pulsantini grafici al lato o all'interno del campo input.

```
<input type="number">
```

Questo il risultato:

numero:



number è Supportato dai browser FireFox Opera Chrome.
Negli smartphone richiama direttamente la tastiera numerica del telefonino.

range

Il type **range** permette di selezionare un valore numerico semplicemente spostando un cursore grafico. Ovviamente non viene considerato importante il suo valore, essendo espresso graficamente risulterà solo indicativo.

```
<input type="range" min="1" max="10">
```

Questo il risultato:

range:

In questo esempio è stato inserito l'attributo min e max per dare un senso alla scala di valori.
range è supportato dai browser FireFox Opera Chrome Safari.

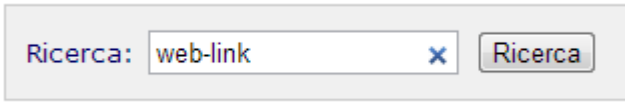
search

Il type **search** identifica un campo di ricerca, interna al sito o esterna in rete. Apparentemente sembra un campo come altri ma il browser lo distingue assegnandogli maggiore accessibilità. Chi ha un browser Chrome potrà notare che non appena si inizia a scrivere qualcosa al suo interno, appare una X azzurra di lato che serve per cancellare quanto già scritto.

```
<input type="search">
```

Questo il risultato:

Ricerca: Ricerca



search è supportato dai browser Chrome e Safari.

tel

Il type **tel** permette di inserire numeri adoperando la tastiera del telefono, ne consegue che questo tipo di input, apparentemente identico ad altri, viene sfruttato a dovere dagli smartphone.

```
<input type="tel">
```

Questo il risultato:

tel:

tel attualmente nessun browser lo supporta. Negli smartphone richiama direttamente la tastiera numerica del telefonino.

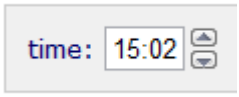
time

Il type **time** permette di selezionare l'ora senza l'informazione del fuso orario.

```
<input type="time">
```

Questo il risultato:

time:



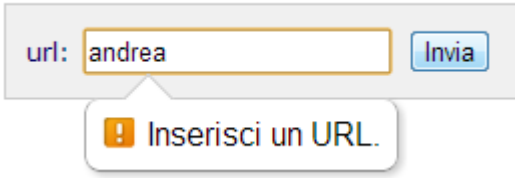
time è supportato dai browser Opera Chrome Safari.

url

Il type **url** permette di inserire un URL valido, al suo invio viene eseguita una validazione.

```
<input type="url">
```

Questo il risultato:



Provate ad inserire una qualsiasi cosa, Opera aggiunge al vostro testo http:// mentre Chrome e FireFox vi segnalano che non è un URL.
url è supportato dai browser FireFox Opera Chrome.

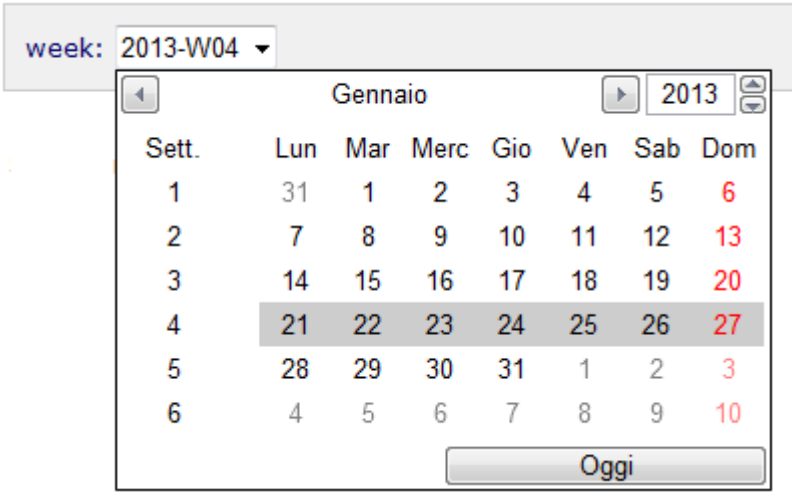
week

Il type week permette di selezionare la settimana e l'anno direttamente da un calendario grafico generato dal browser.

```
<input type="week">
```

Questo il risultato:

week:



week è supportato dai browser Opera e Safari anche se in modalità grafica differente.

Gli input type: number, range, date, datetime, datetime-local, month, time e week accettano come attributi: step, min- max e value spiegati insieme ad altri nuovi attributi nella prossima lezione.

Attributi di HTML5 per i Forms

Nuovi attributi per elementi dei Form: autocomplete - autofocus - form - formaction - formenctype - formmethod - formnovalidate - formtarget - multiple - pattern (regexp) - placeholder - required - step - width e height

Nuovi attributi per i Forms.

HTML5 ha introdotto nuovi attributi che di solito si associano ai campi di tipo input dei moduli stessi, alcuni però sono utilizzabili anche all'interno di elementi come vedremo negli esempi. Vorrei ricordare che in questa guida vengono mostrati soltanto i nuovi attributi per i forms di HTML5, dando per scontato che voi conosciate già gli altri di HTML precedenti, se così non fosse, su web-link ci sono guide per tutti i livelli di conoscenza, principianti compresi, grazie alle quali sarà possibile comprendere al meglio lo scopo ed il significato di questi nuovi introdotti da HTML5.

Di seguito la descrizione con esempio dei nuovi attributi da assegnare al campo input che per default è text, libero testo. Nelle attuali versioni di alcuni browser non tutti sono riconosciuti. Quello che meglio di altri ha dimostrato di essere adeguato è al momento Opera.

autocomplete

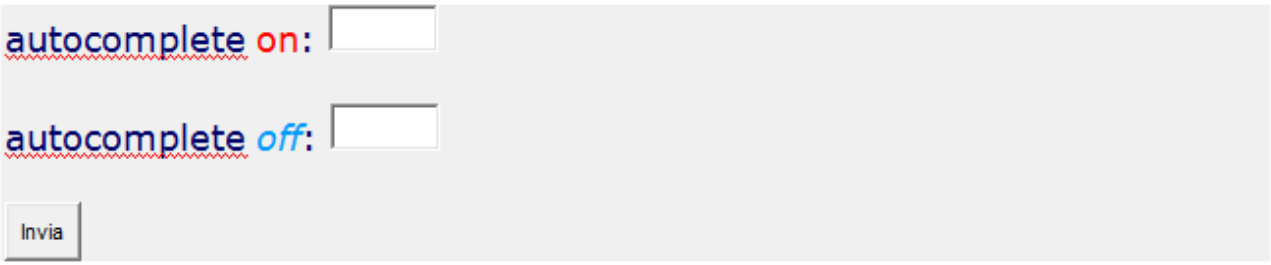
L'attributo **autocomplete** ha due stati logici: on ed off rispettivamente per essere attivato o disattivato, per default è già attivo. Serve per riutilizzare i dati già inseriti su form precedenti, se infatti provate ad inserire il dato, vi viene proposto quanto già inserito precedentemente. Con autocomplete off è possibile disattivarlo e diventa utile in quei campi che per ragioni di privacy e/o di sicurezza non devono essere mostrati dati soltanto perchè già utilizzati in precedenza. Si applica ai vari tipi di input type ma è possibile applicarlo anche all'intero modulo se dichiarato nell'elemento principale <form>.

```
<form action="xxx" autocomplete="off">
```

Sopra applicato ad un solo campo riservato per la email, sotto applicato all'intero form e di conseguenza a tutti i campi in esso contenuti.

```
<input type="email" autocomplete="off">
```

Questo il
risultato:



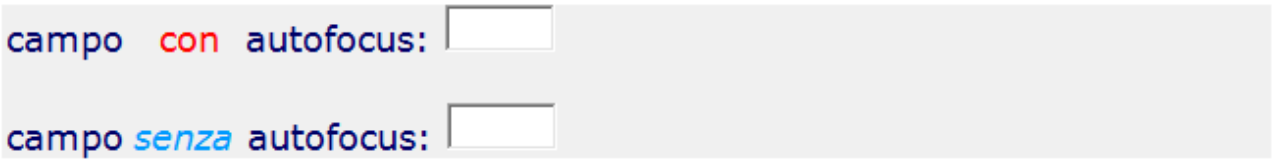
Mettete il vostro nome su entrambi i campi, premete invio, adesso cancellate le caselle e provate a scrivere di nuovo lo stesso nome, su quella impostata *on*appena inserite la prima lettera, o fate un doppio click, vi viene autocompletato il nome inserito precedentemente, su quella impostata *off* invece non succede nulla proprio per via della disabilitazione di questa funzione.

autofocus

L'attributo **autofocus** richiama il focus nella casella di input alla quale è stato applicato, praticamente il cursore è lì lampeggiante che vi aspetta per introdurre il dato, dovete soltanto iniziare a scrivere.

```
<input type="text" autofocus>
```

Questo il risultato:



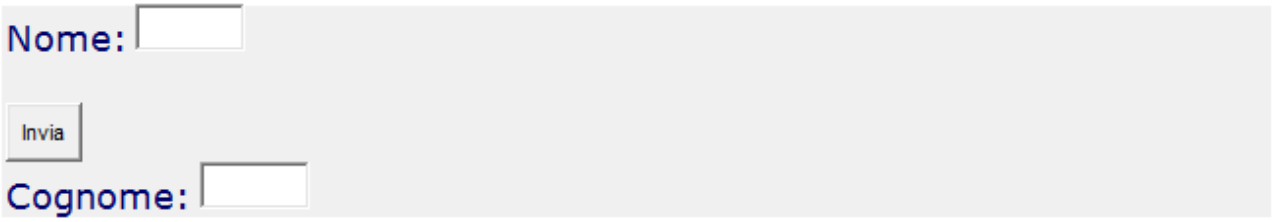
Ho volutamente disabilitato in questo esempio autofocus perchè sareste arrivati qui ogni volta che facevate un accesso a questa pagina, quindi poteva risultare fastidioso, per farvi vedere come funziona cliccate su questo link e noterete il cursore ad inizio campo che lampeggia, proprio come dopo un click al suo interno per iniziare a scrivere.

form

L'attributo **form** assegna ad un form un eventuale campo situato fisicamente al di fuori del form stesso, questo fino ad HTML5 era considerato un grave errore, la chiusura di un form con il suo tag </form> impediva di essere preso in considerazione qualsiasi altro elemento valido di un form che ne fosse al di fuori. Con questo attributo invece, facendo riferimento al nome del form assegnato con ID, è possibile posizionare fisicamente un campo in un punto qualsiasi della pagina web facendo in modo che risulti appartenente al modulo e come tale considerato suo a tutti gli effetti.

```
<form action="xxx" id="weblink">
Nome: <input type="text">
<input type="submit" value="Invia">
</form>

Cognome: <input type="text" form="weblink">
```



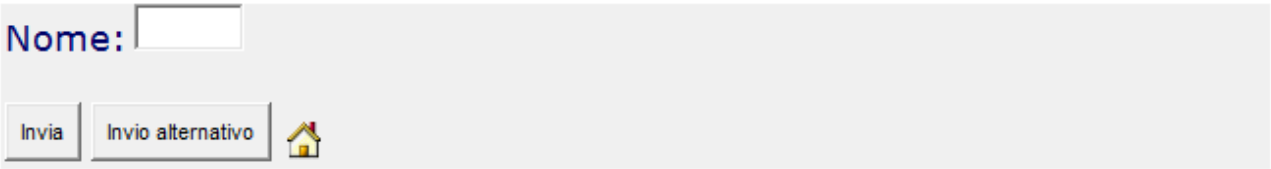
Si noti il nome assegnato con id (weblink) all'elemento iniziale <form>, si noti la chiusura di </form> ed il campo esterno Cognome che grazie all'attributo *form="weblink"* di fatto lo rende un campo facente parte integrante del modulo stesso. proprio come se fosse stato scritto prima della chiusura di </form>.

formaction

L'attributo **formaction** permette di spedire il modulo a destinazioni diverse, praticamente oltre a quanto dichiarato con l'attributo action di form, è possibile avere più di un solo action. Si adopera con gli elementi input type image e submit. Potrebbe risultare comodo nel caso in cui lo stesso modulo dati dovesse o potesse essere spedito a destinazioni differenti a seconda degli eventi, per esempio volendo contattare un membro di una comunità, anzichè creare tanti form uguli ognuno per ogni membro, se ne crea uno solo e si decide con formaction a chi inviarlo.

```
<form action="xxx">
Nome: <input type="text">
<input type="submit" value="Invio">
<input type="submit" formaction="yyy" value="Invio alternativo">
<input type="image" src="nome.gif" formaction="zzz" alt="invio">
</form>
```

Questo il risultato:



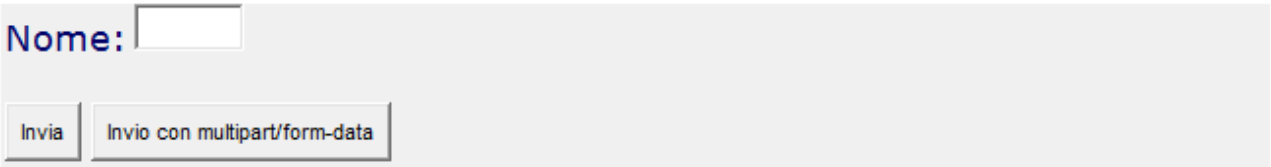
Lo stesso modulo può essere inviato a tre destinazioni diverse, (xxx, yyy, zzz) siano queste pagine web, indirizzi di posta o script per l'elaborazione dei dati. Due sono possibili con il pulsante submit ed una con l'immagine grafica.

formenctype

L'attributo **formenctype** permette di cambiare il tipo di media utilizzato da form per codificare i dati del modulo che di default è MINE, è possibile avere più di un solo enctype ma solo se il method è post. Si adopera con gli elementi **input type image e submit**.
Potrebbe servire avere più codifiche usando lo stesso modulo anzichè creare tanti moduli uguali, ognuno per una diversa codifica, se ne crea uno solo e si decide con formenctype quale usare.

```
<form action="xxx" method="post">
Nome: <input type="text">
<input type="submit" value="Invio">
<input type="submit" formenctype="multipart/form-data" value="Invio con
Multipart/form-data">
</form>
```

Questo il risultato:



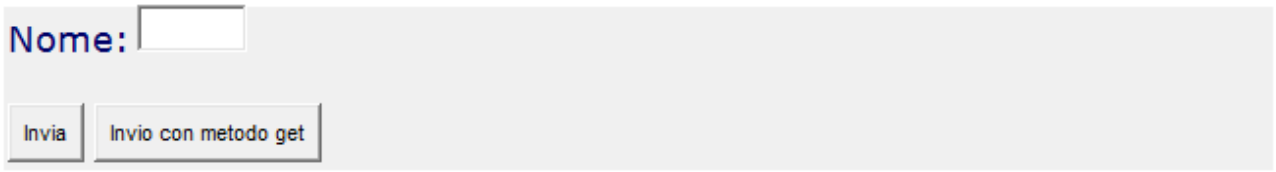
Lo stesso modulo può essere inviato con diversi tipi di media per codificare i dati. Unico requisito obbligatorio, il method che deve essere di tipo *post* e non *get*.

formmethod

L'attributo **formmethod** permette di cambiare il metodo di spedizione post o get. Si adopera con gli elementi input **type image e submit**.
Potrebbe servire avere due diversi script che elaborano i dati ed inviarli con due metodi differenti, pos e get.

```
<form action="xxx" method="post">
Nome: <input type="text">
<input type="submit" value="Invio">
<input type="submit" formmethod="get" formaction="yyy" value="Invio con metodo get">
</form>
```

Questo il risultato:



Lo stesso modulo può essere inviato con i due diversi metodi, tipo *post* o tipo *get*.

formnovalidate e novalidate

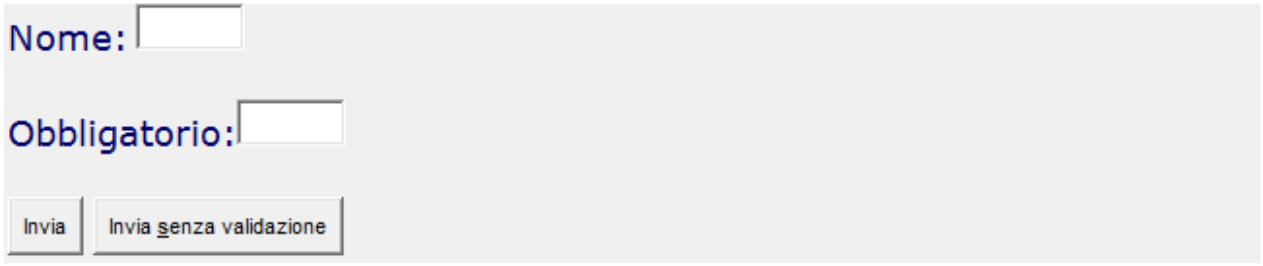
L'attributo **formnovalidate** esclude il modulo da una validazione. Si adopera con il solo elemento input type submit. Potrebbe essere utile spedire il modulo senza validazione, in questo caso o si dichiara l'attributo **novalidate** con <form> oppure si adopera **formnovalidate** con un secondo pulsante di invio (submit) per avere entrambe le possibilità.

```
<form action="xxx" novalidate>
```

Sopra l'atributo novalidate applicato all'intero modulo. Sotto l'attributo formnovalidate applicato ad un secondo pulsante di invio.

```
<form action="xxx">
Nome: <input type="text">
<input type="submit" value="Invio">
<input type="submit" formnovalidate value="Invio senza validazione">
</form>
```

Questo il risultato:



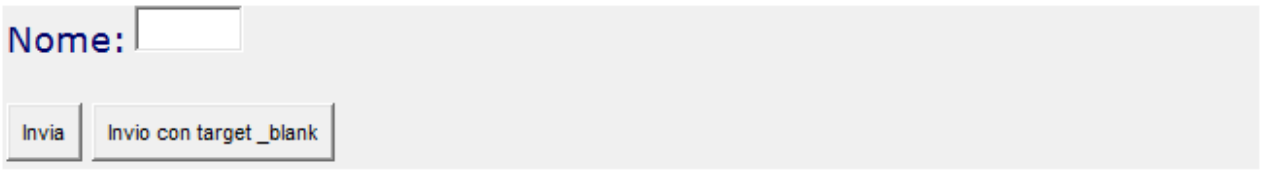
Lo stesso modulo può essere inviato con o senza il controllo di validazione.

formtarget

L'attributo **formtarget** permette di scegliere dove visualizzare la risposta del server all'invio del nostro modulo, simile al target dei links per intenderci meglio. Si adopera con gli elementi input **type image e submit**.
Potrebbe servire avere la risposta su una nuova finestra del browser target="_blank".

```
<form action="xxx">
Nome: <input type="text">
<input type="submit" value="Invio">
<input type="submit" formtarget="_blank" value="Invio con target _blank">
</form>
```

Questo il risultato:



Lo stesso modulo può essere inviato scegliendo magari una nuova finestra del browser per la risposta.

width e height

Gli attributi **width e height** servono per specificare le dimensioni in larghezza ed in altezza delle immagini, di conseguenza sono applicabili al solo elemento input type image. Proprio come per il tag img di html è buona norma usarlo anche se non è obbligatorio, si agevola il lavoro del browser che saprebbe fin da subito quanto spazio riservare alle dimensioni dell'immagine che dovrà essere caricata prima di essere visualizzata.

```
<form action="xxx">
<input type="image" src="nome_immagine.gif" width="xx" height="yy">
</form>
```

Questo il risultato:

list

L'attributo **list** serve per richiamare l'elemento <datalist> visto in precedenza, una lista di dati selezionabili con un click.

```
<input list="nazione">
<datalist id="nazione">
<option value="Italia">
<option value="Inghilterra">
</datalist>
```

Questo il risultato:

Provate un doppio click nel campo vuoto o ad inserire una lettera presente nei nomi Italia ed Inghilterra.

min e max

Gli attributi min e max delimitano valori minimi e valori massimi che un campo può accettare. Si adopera con gli elementi input type number, range, date, datetime, datetime local, month, time e week.

```
<form action="xxx">
Scegli un valore: <input type="number" min="5" max="10">
<input type="submit" value="Invio">
</form>
```

Questo il risultato:

Scegli un valore:

Invia

Avendo usato min 5 e max 10 il campo accetta soltanto i valori corrispondenti, qualsiasi altro valore genera il messaggio di avviso.

multiple

L'attributo **multiple** si adopera quando i dati da inserire in quel campo sono più di uno. Si adopera con gli elementi input type file e email.

```
<form action="xxx">
Invia uno o più files: <input type="file" multiple>
<input type="submit" value="Invio">
</form>
```

Questo il risultato:

Invia uno o più files:

Invia

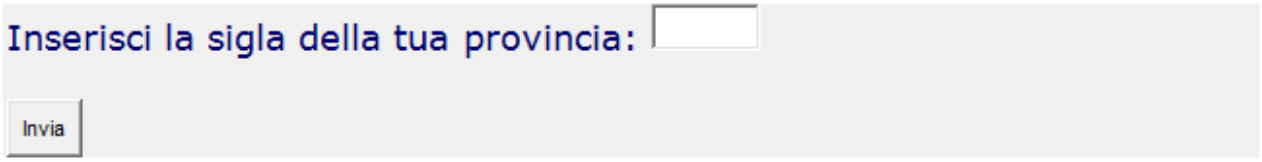
Avendo usato multiple il campo input accetterà più di un solo file.

pattern

L'attributo **pattern** si adopera per specificare una espressione regolare in informatica regular expressions, praticamente è possibile vincolare quantità e tipo di dato da inserire usando questa regular expressions che è conosciuta con qualsiasi linguaggio di programmazione. Si adopera con gli elementi input type text, search, url, tel, email, e password.

```
<form action="xxx">
Inserisci la sigla della tua provincia: <input type="text" pattern="[A-Za-z]{2}"title="solo due lettere per la provincia">
<input type="submit" value="Invio">
</form>
```

Questo il risultato:



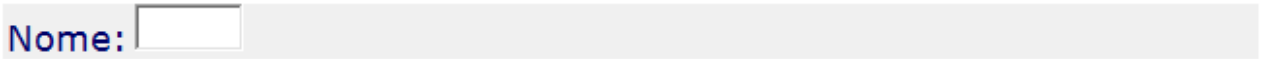
Nell'esempio si accettano solo 2 caratteri alfabetici dalla A alla Z, maiuscoli e/o minuscoli niente numeri o caratteri speciali. La frase nel title è quella che appare nel caso in cui vogliate inviarlo senza aver rispettato la regola imposta.

placeholder

L'attributo **placeholder** inserisce provvisoriamente una indicazione scritta di un colore grigio chiaro nella casella di input che sparisce nel momento in cui l'utente compila il campo stesso, molto utile per ricordare quale sia il tipo di dato o lo scopo di quella casella.

```
<input type="search">
```

Questo il risultato:



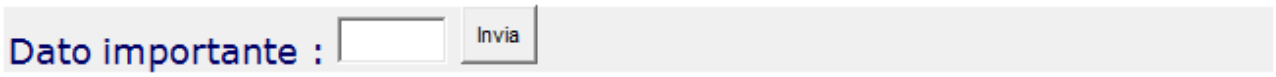
Fate click sul campo e provate a digitare qualcosa, con la prima lettera la frase che c'è al momento sparirà.

required

L'attributo **required** vincola la spedizione del modulo al fatto di aver compilato questo campo, quindi un dato obbligatorio che non può essere omesso.

```
<input type="text" required>
```

Questo il risultato:



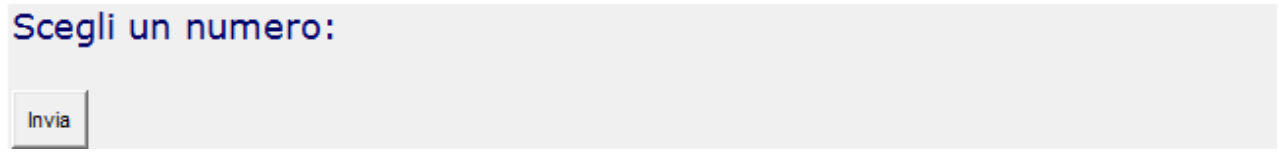
Provate ad inviare il modulo senza aver inserito alcun dato in questo campo con attributo required

step

L'attributo **step**= passo, incremento, così come max e min, cambia il suo valore a seconda del contesto, per esempio se inserito in un type data step 1 consente incrementi di 1 giorno, mentre se inserito in un type time l'incremento è di 1 secondo. Si adopera con gli elementi input type number, range, date, datetime, datetime-local, month, time e week.

```
<form action="xxx">
Scegli un valore: <input type="number" step="2">
<input type="submit" value="Invio">
</form>
```

Questo il risultato:



Avendo usato min 5 e max 10 il campo accetta soltanto i valori corrispondenti, qualsiasi altro valore genera il messaggio di avviso.

Video in HTML5

Elementi Video: <video> <source>

Attributi per Elementi Video: controls - width e height - autoplay - loop - muted - poster - preload - src

Filmati Video.

HTML5 ha introdotto nuovi elementi e nuovi attributi per poter mostrare e gestire un video senza preoccuparsi se nel PC di chi visualizza la pagina web ci siano o meno i programmi e/o i plug-in istallati, Quello che serve è tutto all'interno del browser.
Di seguito la descrizione con esempi dei nuovi elementi usabili per i video. Nelle attuali versioni di alcuni browser si riscontra che non tutti si comportano allo stesso modo ma possiamo dire che a differenza delle versioni precedenti tutti riescono a mostrare il video cambiando soltanto quello che è l'interfaccia grafica di controllo.

Vi segnalo questo sito che vi permette direttamente on line di convertire video e audio da e verso qualsiasi formato.

<video>...</video>

L'elemento **<video>** richiama un file di tipo video facendo uso dell'attributo srcpraticamente allo stesso modo di come si è abituati ad usare il tag img. Al momento sono tre i formati video supportati: MP4, Ogg e WebM.

Potrei quindi usare uno dei tre formati con questa sintassi:

```
<video src="nome_video.MP4">
Il tuo browser non supporta l'elemento video
</video>
```

Questo il risultato:

Non vedete nulla? è probabile perchè non tutti i browser supportano il formato mp4. Ma allora qualcuno penserà che anche con html5 non è cambiato nulla, tranquilli, la soluzione c'è, ho fatto volutamente questo passaggio per farvi capire meglio lo scopo dell'elemento source spiegato subito sotto. Prima però assicuriamoci che il vostro browser riconosca il tag video fate un click sul pulsante Check Video:

Check Video

<source>

L'elemento **<source>** si adopera all'interno dell'elemento <video>, richiama tramite il suo attributo src il tipo di file video. Praticamente la stessa cosa vista sopra ma con la notevole differenza che è possibile dichiarare all'interno di <video> più di un solo source. Così facendo i formati video dichiarati saranno più di uno e sarà il browser a mostrare quello da lui stesso riconosciuto ignorando gli altri. A differenza di src per video che richiamava soltanto il nome del file, con source si deve specificare tramite l'attributo type, anche il tipo di codifica mine:

MP4 = video/mp4

Ogg = video/ogg

WebM = video/webm

Si Potrebbero quindi usare tutti e tre i formati con questa nuova sintassi:

```
<video>
<source src="nome_video.mp4" type="video/mp4">
<source src="nome_video.ogg" type="video/ogg">
<source src="nome_video.webm" type="video/webm">
Il tuo browser non supporta l'elemento video
</video>
```

Questo il risultato, sicuro che anche il vostro browser vi mostrerà il filmato.

Questi sotto i tre formati presi singolarmente, al solo scopo di vedere quali dei tre il tuo browser supporta.

```
<video>
<source src="nome_video.mp4" type="video/mp4">
Il tuo browser non supporta l'elemento video
</video>
```

Formato mp4

```
<video>
<source src="nome_video.ogg" type="video/ogg">
Il tuo browser non supporta l'elemento video
</video>
```

Formato ogg

```
<video>
<source src="nome_video.webm" type="video/webm">
Il tuo browser non supporta l'elemento video
</video>
```

Formato webm

In tutti gli esempi sopra si è fatto uso dell'attributo *controls* anche se nel codice degli esempi non è riportato, lo andremo a vedere subito, ci è servito per mostrare l'interfaccia grafica offrendoci la possibilità di avviare e fermare il filmato stesso.

controls

L'attributo **controls** permette di avere un'interfaccia grafica che cambia a seconda del browser e che serve per fermare/avviare il filmato, controllare il volume dell'audio, passare a schermo intero, avere dei sottotitoli ed altro ancora.

```
<video controls>
<source src="nome_video.mp4" type="video/mp4">
<source src="nome_video.ogg" type="video/ogg">
<source src="nome_video.webm" type="video/webm">
Il tuo browser non supporta l'elemento video
</video>
```

Questo il risultato:

Portate il cursore del mouse sopra l'interfaccia grafica, vedrete la possibilità di controllare il volume, mettere il video in pausa, riavviarlo, passare a tutto schermo ecc. ecc. La qualità e la durata del filmato sono volutamente ridotte per non gravare sulle risorse del mio spazio web, tenete sempre in considerazione il consumo di banda usando filmati di lunga durata e con ottima qualità video.

whidt e hight

Gli attributi **whidt e hight** servono per specificare le misure in pixel per larghezza ed altezza del filmato. Conviene specificarle sempre, anche nel caso in cui non si volesse ingrandire o ridurre le dimensioni originali, questo aiuterebbe il browser che diversamente dovrebbe calcolarselo da solo soltanto a caricamento del filmato avvenuto.

```
<video width="320" height="240" controls>
<source src="nome_video.mp4" type="video/mp4">
<source src="nome_video.ogg" type="video/ogg">
<source src="nome_video.webm" type="video/webm">
Il tuo browser non supporta l'elemento video
</video>
```

Questo il risultato con dimensioni doppie rispetto a quelli visti sopra:

Portate il cursore del mouse sopra l'interfaccia grafica, vedrete la possibilità di controllare il volume, mettere il video in pausa, riavviarlo, passare a tutto schermo ecc. ecc. La qualità e la durata del filmato sono volutamente ridotte per non gravare sulle risorse del mio spazio web, tenete sempre in considerazione il consumo di banda usando filmati di lunga durata e con ottima qualità video.

autoplay

L'attributo **autoplay** inserito nell'elemento <video> indica al browser di avviare automaticamente il video non appena questo è stato caricato.

```
<video autoplay width="320" height="240" controls>
<source src="nome_video.mp4" type="video/mp4">
<source src="nome_video.ogg" type="video/ogg">
<source src="nome_video.webm" type="video/webm">
Il tuo browser non supporta l'elemento video
</video>
```

loop

L'attributo **loop** inserito nell'elemento <video> indica al browser di ripetere la riproduzione del filmato all'infinito.

```
<video loop width="160" height="120" controls>
<source src="nome_video.mp4" type="video/mp4">
<source src="nome_video.ogg" type="video/ogg">
<source src="nome_video.webm" type="video/webm">
Il tuo browser non supporta l'elemento video
</video>
```

Questo il risultato:

Una volta avviato si fermerà soltanto cliccando sul pulsantino dell'interfaccia grafica.

muted

L'attributo **muted** inserito nell'elemento <video> indica al browser di riprodurre il filmato senza audio, muto.

```
<video muted width="160" height="120" controls>
<source src="nome_video.mp4" type="video/mp4">
<source src="nome_video.ogg" type="video/ogg">
<source src="nome_video.webm" type="video/webm">
Il tuo browser non supporta l'elemento video
</video>
```

Questo il risultato:

Una volta avviato si fermerà soltantoi cliccando sul pulsantino dell'interfaccia grafica.

poster

L'attributo **poster** inserito nell'elemento <video> indica quale immagine visualizzare al posto di quella del primo fotogramma del filmato stesso. Questa immagine resta visibile fino a quando non si avvia il filmato o automaticamente o facendo click sul pulsante [play] dell'apposita interfaccia grafica. Per i tipi di immagini che è possibile utilizzare e relativo percorso in cui si trovano, valgono le stesse regole più volte descritte in precedenza.

```
<video poster="immagine.gif" width="160" height="120" controls>
<source src="nome_video.mp4" type="video/mp4">
<source src="nome_video.ogg" type="video/ogg">
<source src="nome_video.webm" type="video/webm">
Il tuo browser non supporta l'elemento video
</video>
```

Questo il risultato:

Ho messo come immagine di copertina il logo di HTML5 che sparirà una volta avviato il filmato.

preload

L'attributo **preload** inserito nell'elemento <video> accetta tre valori: auto, metadate e none se non si specifica nulla, il solo preload assume come valore di default auto. Questi tre valori indicano al browser rispettivamente:

auto = carica il video durante il caricamento della pagina, si avranno così meno attese nella sua esecuzione.

metadate = non caricare il video durante il caricamento della pagina ma solo le sue informazioni compresa una eventuale immagine usata con poster.

none = non caricare il video se non quando sarà premuto il pulsantino [play] dell'interfaccia.

```
<video preload="none" width="160" height="120" controls>
<source src="nome_video.mp4" type="video/mp4">
<source src="nome_video.ogg" type="video/ogg">
<source src="nome_video.webm" type="video/webm">
Il tuo browser non supporta l'elemento video
</video>
```

Questo il risultato:

Facciamo una considerazione, perchè mai dovrei caricare sempre nella memoria cache del browser il filmato, specialmente poi se ha notevoli dimensioni? Chi mi assicura che poi questo sarà visto da chi visita la mia pagina? Avrei uno spreco di banda e di risorse, meglio fare in modo di

consumare queste risorse soltanto per chi è realmente interessato al filmato, quindi personalmente scelgo none.

src

L'attributo **src** inserito nell'elemento <video> o <source> indica il nome del file video completo di estensione ed eventuale percorso per raggiungerlo.

```
<video src="nome_video.MP4">

<video src="filmati/nome_video.MP4">

<video>
<source src="nome_video.mp4" type="video/mp4">
</video>
```

Audio in HTML5

Elementi Audio: <audio><source>

Attributi per Elementi Audio: controls - autoplay - loop - preload - src

Suoni e Musica.

HTML5 ha introdotto nuovi elementi e nuovi attributi per poter ascoltare e gestire un brano audio senza preoccuparsi se nel PC di chi visualizza la pagina web ci siano o meno i programmi e/o i plug-in istallati, Quello che serve è tutto all'interno del browser.
Di seguito la descrizione con esempi degli elementi usabili per l'audio. Nelle attuali versioni di alcuni browser si riscontra che non tutti si comportano allo stesso modo ma possiamo dire che a differenza delle versioni precedenti tutti riescono a riprodurre l'audio cambiando soltanto quello che è la loro interfaccia grafica di controllo.

<audio>...</audio>

L'elemento <audio> richiama un file di tipo audio facendo uso dell'attributo srcpraticamente allo stesso modo di come si è abituati ad usare il tag img. Al momento sono tre i formati audio supportati: MP3, Ogg e Wav.

Potrei quindi usare uno dei tre formati con questa sintassi:

```
<audio src="nome_audio.MP3">
Il tuo browser non supporta l'elemento audio
</audio>
```

Questo il risultato:

Non si sente nulla? è probabile perchè non tutti i browser supportano il formato mp3. Ma allora qualcuno penserà che anche con html5 non è cambiato nulla, tranquilli, la soluzione c'è, ho fatto volutamente questo passaggio per farvi capire meglio lo scopo dell'elemento *source* spiegato subito sotto.

<source>

L'elemento **<source>** si adopera all'interno dell'elemento <audio>, richiama tramite il suo attributo src il tipo di file audio. Praticamente la stessa cosa vista sopra ma con la notevole differenza che è possibile dichiarare all'interno di <audio> più di un solo source. Così facendo i formati audio dichiarati saranno più di uno e sarà il browser a riprodurre quello da lui stesso riconosciuto ignorando gli altri. A differenza di src per audio che richiamava soltanto il nome del file, con source si deve specificare tramite l'attributo type, anche il tipo di codifica mine:
MP3 = audio/mp3
Ogg = audio/ogg
Wav = audio/wav

Potrei quindi usare tutti e tre i formati con questa nuova sintassi:

```
<audio>
<source src="nome_audio.mp3" type="audio/mpeg">
<source src="nome_audio.ogg" type="audio/ogg">
<source src="nome_audio.wav" type="audio/wav">
Il tuo browser non supporta l'elemento audio
</audio>
```

Questo il risultato, sono sicuro che anche il vostro browser vi farà ascoltare il brano.

Questi sotto i tre formati presi singolarmente, al solo scopo di vedere quali dei tre il tuo browser supporta.

```
<audio>
<source src="nome_audio.mp3" type="audio/mp3">
Il tuo browser non supporta l'elemento audio
</audio>
```

Formato mp3

```
<audio>
<source src="nome_audio.ogg" type="audio/ogg">
Il tuo browser non supporta l'elemento audio
</audio>
```

Formato ogg

```
<audio>
<source src="nome_audio.wav" type="audio/wav">
Il tuo browser non supporta l'elemento audio
</audio>
```

Formato wav

In tutti gli esempi sopra ho fatto uso dell'attributo *controls* anche se nel codice degli esempi non è riportato, lo andremo a vedere subito, mi è servito per mostrare l'interfaccia grafica offrendovi la possibilità di avviare e fermare la riproduzione audio.

controls

L'attributo **controls** permette di avere un'interfaccia grafica che cambia a seconda del browser e che serve per fermare/avviare l'audio, controllare il volume, spostarsi all'interno della traccia audio.

```
<audio controls>
<source src="nome_audio.mp3" type="audio/mp3">
<source src="nome_audio.ogg" type="audio/ogg">
<source src="nome_audio.wav" type="audio/wav">
Il tuo browser non supporta l'elemento audio
</audio>
```

Questo il risultato:

Portate il cursore del mouse sopra l'interfaccia grafica, vedrete la possibilità di controllare il volume, mettere l'audio in pausa, riavviarlo, spostarsi in punti diversi del brano. La qualità e la durata del brano sono volutamente ridotte per non gravare sulle risorse del mio spazio web, tenete sempre in considerazione il consumo di banda usando brani di lunga durata e di ottima qualità.

autoplay

L'attributo **autoplay** inserito nell'elemento <audio> indica al browser di avviare automaticamente il brano non appena questo è stato caricato.

```
<audio autoplay controls>
<source src="nome_audio.mp3" type="audio/mp3">
<source src="nome_audio.ogg" type="audio/ogg">
<source src="nome_audio.wav" type="audio/wav">
Il tuo browser non supporta l'elemento audio
</audio>
```

loop

L'attributo **loop** inserito nell'elemento <audio> indica al browser di ripetere la riproduzione del brano all'infinito.

```
<audio loop controls>
<source src="nome_audio.mp3" type="audio/mp3">
<source src="nome_audio.ogg" type="audio/ogg">
<source src="nome_audio.wav" type="audio/wav">
Il tuo browser non supporta l'elemento audio
</audio>
```

Questo il risultato:

Una volta avviato si fermerà soltantoi cliccando sul pulsantino dell'interfaccia grafica.

preload

L'attributo **preload** inserito nell'elemento <audio> accetta tre valori: auto, metadate e none se non si specifica nulla, il solo preload assume come valore di default auto. Questi tre valori indicano al browser rispettivamente:

auto = carica il brano durante il caricamento della pagina, si avranno così meno attese nella sua esecuzione.

metadate = non caricare il brano durante il caricamento della pagina ma solo le sue informazioni.

none = non caricare l'audio se non quando sarà premuto il pulsantino [play] dell'interfaccia.

```
<audio preload="none" controls>
<source src="nome_audio.mp3" type="audio/mp3">
<source src="nome_audio.ogg" type="audio/ogg">
<source src="nome_audio.wav" type="audio/wav">
Il tuo browser non supporta l'elemento audio
</audio>
```

Questo il risultato:

Facciamo una considerazione, perchè mai dovrei caricare sempre nella memoria cache del browser il brano, specialmente poi se ha notevoli dimensioni? Chi mi assicura che poi questo sarà ascoltato volentieri da chi visita la mia pagina? Avrei uno spreco di banda e di risorse, meglio fare in modo di consumare queste risorse soltanto per chi è realmente interessato all'audio, quindi personalmente scelgo none.

src

L'attributo **src** inserito nell'elemento <audio> o <source> indica il nome del file audio completo di estensione ed eventuale percorso per raggiungerlo.

```
<audio src="nome_audio.mp3">

<audio src="musica/nome_audio.mp3">

<audio>
<source src="nome_audio.mp3" type="audio/mp3">
</audio>
```

Canvas il nuovo elemento per disegno 2d in HTML5

Metodi e Proprietà Javascript per canvas: coordinate - getContext("2d") - beginPath() - closePath()

Una delle maggiori novità introdotte da HTML5 è la possibilità di disegnare "al volo" sulla pagina web facendo uso di un linguaggio di scriptting come ad esempio JavaScript. Si tratta di definire una porzione della pagina web sulla quale poter disegnare (in 2d) forme geometriche, font di caratteri e persino immagini grafiche. Questa possibilità prende il nome di canvas che significa "tela".

<canvas>...</canvas>

L'elemento **<canvas>** necessita di almeno due attributi per definire larghezza ed altezza della superficie sulla quale andare a disegnare. Si possono avere più canvas nella stessa pagina, per potersi riferire ad ognuno di essi è obbligatorio usare un ID per identificarli, quindi un tag valido potrebbe essere questo:

```
<canvas id="wl" width="250" height="150"></canvas>
```

Così facendo abbiamo creato solo un rettangolo largo *250* pixel ed alto *150* pixel, al quale abbiamo assegnato come identificatore univoco *wl*, possiamo andare a disegnarci sopra come se fosse una piccola lavagna, per farlo si usa javascript, per cui chi non ha esperienza con questo linguaggio troverà forse difficoltà per comprendere a fondo il meccanismo ma gli esempi usati daranno comunque sufficienti informazioni anche ad un utente con poca esperienza in fatto di linguaggi di programmazione.

Per farvi vedere lo spazio occupato da questo canvas, al momento vuoto, gli ho assegnato un colore ed un sottile bordo usando una dichiarazione di style, aggiungiamo anche una scritta che sarà visibile soltanto nel caso in cui il browser non supportasse canvas:

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

Questo sotto il risultato, a seconda del browser adoperato vedrete un rettangolo vuoto con bordo sottile nero, o la scritta che informa che il vostro browser non supporta canvas:

Per poter disegnare su questo canvas abbiamo detto che useremo javascript, vediamo cosa serve per farlo. Si dichiara una variabile, in questo esempio: weblink, alla quale viene assegnato il canvas definito da html5 con id="wl"

```
<script>
var weblink = document.getElementById("wl");
</script>
```

Fatto questo serve un'altra variabile che sfrutta un oggetto già presente in html5, si tratta di getContext("2D"), vediamo di cosa si tratta.

getContext("2D")

L'oggetto **getContext("2D")** è incorporato in HTML5, ha molte proprietà e metodi per disegnare linee, curve, rettangoli, cerchi, archi, testi, immagini ed altro ancora. Qui viene assegnato alla variabile context con riferimento alla variabile weblink che identifica il canvas creato in precedenza, questa nuova variabile di fatto sarà richiamata ed utilizzata da tutti i possibili comandi.

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
</script>
```

Prima però di passare ai vari metodi e proprietà che si possono utilizzare, è utile una dimostrazione per vedere e capire quali siano i punti a cui si fa riferimento sottoforma di coordinate x e y all'interno del canvas stesso. Infatti per scriverci dentro si deve sempre fare riferimento a questi punti dichiarati come *x* ed *y*, coordinate per l'appunto.

Le *Coordinate* del canvas.

Per disegnare si adoperano metodi e proprietà, per le linee ad esempio: *moveTo* e *lineTo* che servono rispettivamente per posizionare la matita virtuale in un punto preciso del canvas e per disegnare da quel punto ad un altro.

beginPath()

Il metodo **beginPath()** può essere usato ogni volta che si inizia a disegnare un nuovo tracciato, formato da line, archi, cerchi: lineTo (), arcTo (), quadraticCurveTo (), bezierCurveTo ().

```
context.beginPath();
```

closePath()

Il metodo closePath() può essere usato ogni volta che si finisce di disegnare un nuovo tracciato, formato da line, archi, cerchi: lineTo (), arcTo (), quadraticCurveTo (), bezierCurveTo ().

```
context.closePath();
```

Canvas il nuovo elemento per disegno 2d in HTML5

Metodi e Proprietà Javascript per disegnare forme nel canvas: beginPath() - closePath() - moveTo - lineTo - stroke() - strokeStyle - fillStyle - fill() - arc - rect - fillRect

Abbiamo visto che cosa è un canvas e quali sono le istruzioni javascript per poterlo gestire, adesso vediamo come fare per disegnare al suo interno una linea retta che lo attraversa in diagonale, per farlo ci posizioniamo in alto a sinistra moveTo(0,0) e tracciamo una linea fino al punto (250,150) praticamente le dimensioni max del nostro canvas e la rendiamo visibile con stroke().

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.beginPath();
context.moveTo(0,0);
context.lineTo(250,150);
context.closePath();
context.stroke();
</script>
```

Questa la spiegazione dei comandi adoperati:

beginPath()

Il metodo beginPath() può essere usato ogni volta che si inizia a disegnare un nuovo tracciato, formato da line, archi, cerchi: lineTo (), arcTo (), quadraticCurveTo (), bezierCurveTo ().

context.beginPath();

closePath()

Il metodo closePath() può essere usato ogni volta che si finisce di disegnare un nuovo tracciato, formato da line, archi, cerchi: lineTo (), arcTo (), quadraticCurveTo (), bezierCurveTo ().

context.closePath();

moveTo(x,y)

Il metodo moveTo(x,y) posiziona la punta della matita nel punto da cui iniziare il disegno. I valori associabili alle sue variabili: x, y sono espressi in pixel e si riferiscono alla posizione rispetto ai margini del canvas.

```
context.moveTo(0,0);
```

lineTo(x,y)

Il metodo lineTo(x,y) disegna la linea fino al punto specificato. I valori associabili alle sue variabili: x, y sono espressi in pixel e si riferiscono alla posizione di fine linea rispetto ai margini del canvas.

```
context.lineTo(250,150);
```

strokeStyle

La proprietà strokeStyle è il colore dell'inchiostro ed è opzionale, assegna a stroke() un colore diverso da quello di default, nell'esempio il rosso corrispondente al codice esadecimale #ff0000".

```
context.strokeStyle = "#ff0000";
```

stroke()

Il metodo stroke() è l'inchiostro, praticamente rende visibile tutto quello che è stato disegnato, senza questo metodo, risulterebbe invisibile. strokeStyle è opzionale ed assegna un colore diverso da quello di default.

```
context.stroke();
```

fillStyle

La proprietà fillStyle può rappresentare un colore, una sfumatura, un pattern. Il valore predefinito è il colore nero.

```
context.fillStyle="#009900";
```

fill()

Il metodo fill() riempie la forma col colore definito con fillStyle.

```
context.fill();
```

Con la stessa identica tecnica si possono disegnare un sacco di altre cose che il canvas permette di fare, magari un cerchio, pieno o vuoto, un rettangolo, o del testo, ecc.

arc(x,y,r,start,stop)

Il metodo **arc(x,y,r,start,stop)** disegna un arco che se fatto girare tutto intorno diventa un cerchio. Accetta il punto centrale(x,y) il raggio(r), un angolo di partenza(start) ed uno di arrivo(stop) in radianti, ed opzionalmente false per il senso orario o true per quello antiorario. Nell'esempio sotto per calcolare i radianti è stato utilizzato il modulo Math di Javascript.

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.beginPath();
context.arc(50,50,30,0,Math.PI*2,false);
context.closePath();
context.strokeStyle = "#006600";
context.stroke();
</script>
```

Con arc è stato tracciato un cerchio in un punto specificato del canvas e con un determinato raggio, strokeStyle definisce il colore della matita, in questo caso il verde, stroke() ha disegnato il cerchietto verde.

C'è anche la possibilità di riempire la forma disegnata usando fill() e fillStyle descritti sopra. Nell'esempio sotto si adopera un colore a tinta unita ma avrebbe potuto essere uno sfondo o qualsiasi altra cosa ammessa.

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.beginPath();
context.arc(50,50,30,0,Math.PI*2,false);
context.closePath();
context.fillStyle="#ff0000";
context.fill();
context.strokeStyle = "#006600";
context.stroke();
</script>
```

La matita che ha disegnato il cerchio è di colore verde ed il riempimento dello stesso è di colore rosso.

rect(x,y,width,height)

Il metodo **rect(x,y,width,height)** disegna un rettangolo o un quadrato, x e y sono espressi in pixel e si riferiscono alla posizione di inizio (angolo superiore sinistro della forma) rispetto ai margini del canvas, mentre width ed heigth sono le dimensioni in larghezza ed altezza.

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.beginPath();
context.rect(25,25,100,70);
context.closePath();
context.strokeStyle = "#006600";
context.stroke();
</script>
```

Con rect è stato tracciato un rettangolo in un punto specificato del canvas e con determinate dimensioni, strokeStyle definisce il colore della matita, in questo caso il verde, stroke() ha disegnato il rettangolo verde.

C'è anche la possibilità di riempire la forma disegnata usando fill() e fillStyle descritti sopra. Nell'esempio sotto si adopera un colore a tinta unita ma avrebbe potuto essere uno sfondo o qualsiasi altra cosa ammessa.

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.beginPath();
context.rect(25,25,100,70);
context.closePath();
context.fillStyle="#ff0000";
context.fill();
context.strokeStyle = "#006600";
context.stroke();
</script>
```

La matita che ha disegnato il rettangolo è di colore verde ed il riempimento dello stesso è di colore rosso.

fillRect(x, y, width, height)

Il metodo **fillRect(x, y, width, height)** disegna un rettangolo già riempito del colore definito con fillStyle. I valori associabili alle sue variabili: x, y, width ed height sono espressi in pixel e si riferiscono alla sua posizione rispetto al margine del canvas e alle sue dimensioni.

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.beginPath();
context.fillStyle="#99cc66";
context.fillRect(50,35,150,80);
context.closePath();
</script>
```

Si noti la mancanza del tratto della matita vista sopra per il contorno della forma.

Canvas il nuovo elemento per disegno 2d in HTML5

Metodi e Proprietà Javascript per disegnare immagini canvas: createLinearGradient() - createRadialGradient() - addColorStop - createPattern() - drawImage()

Abbiamo visto che cosa è un canvas e quali sono le istruzioni javascript per poterlo gestire, adesso vediamo come fare per disegnare al suo interno delle immagini.

createLinearGradient(x,y iniziali, x,y finali)

Il metodo **createLinearGradient()** serve per creare una sfumatura di colore lineare. Una volta dichiarato il rettangolo e creato il gradiente, per inserire i colori si adopera la proprietà addColorStop. All'interno delle parentesi tonde vanno inserite le coordinate del rettangolo con la consueta formula x e y iniziali e x e y finali.

```
context.rect(0,0,250,150);
context.createLinearGradient(0,0,250,150);
```

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.beginPath();
context.rect(0,0,250,150);
var color = context.createLinearGradient(0,0,250,150);
color.addColorStop(0,'#006600');
color.addColorStop(1,'#66ff66');
context.fillStyle = color;
context.fill();
</script>
```

Viene dichiarata una variabile col nome color alla quale viene associato il metodo createLinearGradient, color viene poi usato da addColorStop per gestire l'effetto.

createRadialGradient(x,y iniziali, inizio rad, x,y finali, fine rad)

Il metodo **createRadialGradient()** serve per creare una sfumatura di colore radiale. Una volta dichiarato il rettangolo e creato il gradiente, per inserire i colori si adopera la proprietà addColorStop. All'interno delle parentesi tonde vanno inserite le coordinate del rettangolo con la consueta formula x e y iniziali, inizioRadiale, x e y finali, fineRadiale.

```
context.rect(0,0,250,150);
context.createRadialGradient(120,80,1,120,80,150);
```

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.beginPath();
context.rect(0,0,250,150);
var color = context.createRadialGradient(120,80,1,120,80,150);
color.addColorStop(0, '#006600');
color.addColorStop(1, '#66ff66');
context.fillStyle = color;
context.fill();
</script>
```

Viene dichiarata una variabile col nome color alla quale viene associato il metodo createRadialGradient, color vien poi usato da addColorStop per gestire l'effetto.

addColorStop(offset, colore)

la proprietà **addColorStop** serve per inserire il colore e stabilire la sua sfumatura, essendo un gradiente, radiale o lineare, necessita di due colori, uno di inizio e l'altro di fine, quindi oltre ai due colori è possibile stabilire da quale iniziare e con quale finire (offset). I valori ammessi sono 0 e 1 per l'offset e qualsiasi valore esadecimale per il colore. Se si inverte l'offset si inverte la direzione della sfumatura.

```
color.addColorStop(0,'#006600');
color.addColorStop(1,'#66ff66');

color.addColorStop(1,'#006600');
color.addColorStop(0,'#66ff66');
```

Gli stessi due colori a tinta verde, è sufficiente scambiare il valore 0, 1 (offset) per avere la sfumatura invertita.

createPattern(oggetto immagine, tipo di repeat)

Il metodo **createPattern()** serve per inserire una immagine che faccia da sfondo, valgono le stesse regole dei css per quanto riguarda il suo repeat: repeat, repeat-x, repeat-y, no-repeat, se si omette il parametro viene assunto per default repeat.

```
context.rect(0,0,250,150);
var pattern = context.createPattern(imageObj, 'repeat');
context.fillStyle = pattern;
```

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
var imageObj = new Image();
context.beginPath();
context.rect(0,0,250,150);
imageObj.onload = function() {
var pattern = context.createPattern(imageObj,"repeat");
context.fillStyle = pattern;
context.fill();};
imageObj.src = "percorso e nome immagine";
```

Viene dichiarata una variabile col nome pattern alla quale viene associato il metodo createPattern, pattern viene poi usato da fillStyle per riempire il canvas.

Questo il piccolo pattern usato per l'esempio:  Per i tipi di immagini valide, valgono le stesse regole di html e css, e cioè: .gif, .png, .jpg.
Il percorso dichiarato per il richiamo dell'immagine grafica può essere di tipo assoluto o relativo.

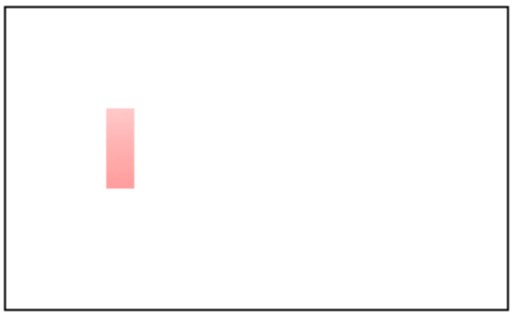
drawImage(oggetto immagine, x, y)

Il metodo **drawImage()** serve per inserire una immagine grafica, i valori x e y servono per stabilire il punto preciso del canvas in cui inserirla facendo riferimento all'angolo superiore sinistro dell'immagine usata.

```
context.drawImage(imageObj,x,y');;
```

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
var imageObj = new Image();
imageObj.onload = function() {
context.drawImage(imageObj,x,y);
};
imageObj.src = "percorso e nome immagine";
```



Questa la piccola immagine usata per l'esempio:  Per i tipi di immagini valide, valgono le stesse regole di html e css e cioè: .gif, .png, .jpg.

Il percorso dichiarato per il richiamo dell'immagine grafica può essere di tipo assoluto o relativo.

Canvas il nuovo elemento per disegno 2d in HTML5

Metodi e Proprietà Javascript per disegnare testo canvas: font - fillText() - fillStyle - strokeStylet - strokeText - lineWidth - textAlign - textBaseline

Abbiamo visto che cosa è un canvas e quali sono le istruzioni javascript per poterlo gestire, adesso vediamo come fare per scrivere al suo interno del testo.

font

La proprietà **font** serve per impostare gli attributi del testo, è possibile specificare lo style del font, la sua dimensione e la famiglia (tipo di font). Questi tre parametri vanno inseriti all'interno di virgolette e separati da uno spazio vuoto. Per style del font valgono: normal, italic e bold, se non si specifica nulla per default viene assunto il normal. Da solo serve a poco, si tratta di attributi letti dal metodo fillText() descritto sotto.

```
context.font = "italic 28px Verdana"';
context.fillText(frase, x, y);
```

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.font = "italic 28px Verdana";
context.fillText("Web-Link", 25, 50);
</script>
```

Ho scritto la parola Web-Link con stile italico usando il font verdana con dimensioni 28 pixel e l'ho posizionato nel canvas a 25px dal margine sinistro e 50 px dal margine superiore.

fillText("testo" x, y)

Il metodo **fillText()** serve per scrivere nel canvas facendo uso della proprietà fonte relativi parametri visti sopra. La frase o il testo devono essere racchiusi da virgolette, una virgola e le coordinate x, y per specificare la posizione esatta di inizio testo all'interno del canvas stesso.

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.font = "italic 28px Verdana";
context.fillText("Web-Link", 25, 50);
</script>
```

Ho scritto la parola Web-Link con stile italico usando il font verdana con dimensioni 28 pixel e l'ho posizionato nel canvas a 25px dal margine sinistro e 50 px dal margine superiore.

fillStyle

La proprietà **fillStyle**, già vista per il canvas forme, rappresenta il colore. Il valore predefinito è il nero, accetta valori come: il nome del colore, il valore in #esadecimale ed il valore in RGB racchiusi da doppie virgolette.

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.font = "italic 28px Verdana";
context.fillStyle="#009900";
context.fillText("Web-Link", 25, 50);
</script>
```

Ho scritto la parola Web-Link con stile italico usando il font verdana con dimensioni 28 pixel e l'ho posizionato nel canvas a 25px dal margine sinistro e 50 px dal margine superiore. Con fillStyle gli ho assegnato il colore verde.

strokeStyle

Come già visto per le forme è possibile impostare un colore per la penna che scrive (stroke). La proprietà **strokeStyle** ha come valore predefinito il nero, accetta valori come: il nome del colore, il codice in #esadecimale o il codice in RGB racchiusi da doppie virgolette. Questa proprietà necessita del metodo strokeText() descritto sotto.

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.font = "italic 28px Verdana";
context.strokeStyle="#ff0000";
context.strokeText("Web-Link", 25, 50);
</script>
```

Ho scritto la parola Web-Link con stile italico usando il font verdana con dimensioni 28 pixel e l'ho posizionato nel canvas a 25 px dal margine sinistro e 50 px dal margine superiore. Ho usato strokeText() per scrivere e selezionato il colore del tratto, rosso in questo esempio, con strokestyle.

strokeText()

Il metodo **strokeText** serve per scrivere testo nel canvas, un po' quello che fa già fillText ma stroke mostra il tratto della penna, grazie alla proprietà strokeStyle vista sopra è possibile scegliere il colore del tratto. La frase, o il testo, devono essere racchiusi da virgolette, una virgola e le coordinate x, y per specificare la posizione esatta di inizio testo all'interno del canvas stesso.

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.font = "italic 50px Verdana";
context.strokeStyle="#0000ff";
context.strokeText("Web-Link", 25, 50);
</script>
```

Ho scritto la parola Web-Link con stile italico usando il font verdana con dimensioni 50 pixel ho scelto di ingrandire il font per meglio mostrare il suo bordo colorato e l'ho posizionato nel canvas a 25 px dal margine sinistro e 80 px dal margine superiore. Ho usato strokeText() ed impostato il colore azzurro con strokeStyle.

E' possibile usare entrambi: strokeText e fillText, in questo caso fillText deve essere dichiarato prima di strokeText per evitare che il riempimento si sovrapponga al margine.

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.font = "italic 50px Verdana";
context.fillStyle="#ffff00";
context.fillText("Web-Link", 5, 80);
```

```
context.strokeStyle="#ff0000";
context.strokeText("Web-Link", 5, 80);
</script>
```

Nell'esempio sopra ho riempito la scritta di giallo con fillStyle usato da FillText poi ho impostato il bordo in rosso con strokeStyle usato da strokeText. Avendo entrambi i metodi le stesse coordinate di posizionamento l'effetto di riempimento è assicurato.

lineWidth

La proprietà **lineWidth**, definisce lo spessore in pixel del tratto usato con strokeText().

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.font = "italic 48px Verdana";
context.lineWidth="3";
context.strokeStyle="#ff0000";
context.strokeText("Web-Link", 5, 80);
</script>
```

textAllign

La proprietà **textAlign**, serve per allineare il testo nel canvas rispetto al punto dichiarato con x ed y di fillText() o strokeText() immaginando una linea verticale. I valori ammessi
sono: start, end, left, right e center. rispettivamente per all'inizio, alla fine, a sinistra, a destra, centrato. E' interessante vedere come il testo cambia punto di riferimento da sinistra verso destra e viceversa a seconda del valore adoperato.

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.font = "italic 18px Verdana";
context.textAlign = "center";
context.fillStyle="#009900";
context.fillText("Web-Link", 125, 75);
</script>
```

Nell'esempio sopra ho usato un allineamento *center* ed ho inserito nelle coordinate x,y la metà esatta delle dimensioni del canvas definito, di conseguenza la scritta risulta perfettamente al centro.

Dal punto centrale usando *end* In quel punto deve finire.

Dal punto centrale usando *start* Da quel punto deve iniziare.

La proprietà textBaseline, serve per allineare il testo nel canvas rispetto al punto dichiarato con x ed y di fillText() o strokeText() immaginando una linea orizzontale. I valori ammessi sono: *top*, *bottom*, *middle*, *alphabetic*, rispettivamente per in alto, in basso, in mezzo. Per default viene assunto alphabetic. Ci sarebbero altri due valori: *ideographic* (simile a bottom) e *hanging*(simile a top) difficilmente adoperati.

```
<canvas id="wl" width="250" height="150" style="border: 1px solid #000;">
Il tuo browser non supporta canvas di html5
</canvas>
```

```
<script>
var weblink = document.getElementById("wl");
var context = weblink.getContext("2d");
context.font = "italic 18px Verdana";
context.textBaseline = "top";
context.fillStyle="#009900";
context.fillText("Web-Link", 125, 75);
</script>
```

Nell'esempio sopra usando allineamento *top*.

Dal punto centrale usando *bottom*

Dal punto centrale usando *middle*

Conclusioni

HTML5 ha introdotto nel web anche molto altro ancora ma come è mia abitudine fare, al momento ho trattato soltanto gli argomenti che personalmente considero realmente utili ai fini pratici. Non è presunzione, considerando anche che la rete è piena di siti che trattano HTML5 in maniera molto esaustiva. Lo stesso sito del W3C dal quale ho attinto molte delle info contenute in questa guida, saprà soddisfare ogni richiesta dei più esigenti.

Questa guida da sola potrebbe non bastare al principiante, ho dato per scontato che giungere fin qui sia da utente medio, se così non fosse ci sono le altre guide che partono proprio da zero e che vi invito a consultare.

A tal potete visionare il sito www.web-link.it da cui anche questa guida è stata rielaborata.

Rispetto ad HTML precedenti HTML5 ha messo fuori uso questi elementi o tags:

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

Tutti gli altri invece possono tranquillamente essere adoperati anche con HTML5.