

Applicazioni Web 2013/14

Lezione 2: HTML

Matteo Vaccari

<http://matteo.vaccari.name/>
matteo.vaccari@uninsubria.it



(cc) Matteo Vaccari. Published in Italy.
Attribution – Non commercial – Share alike 2.5

HTML: HyperText Markup Language

Tim Berners-Lee e Robert Caillau al CERN nel 1991

HTML descrive la **struttura logica** di un documento

- il browser è libero di presentare i tag come preferisce
- il documento ha senso anche se si ignorano i tag

HTML combina idee preesistenti:

- ipertesti (Vannevar Bush, **As We May Think**, 1945)
- linguaggi di marcatura (SGML, 1970)

HTML è compatto, leggibile e portabile

Molti altri formati di documenti sono voluminosi:

- l'autore controlla il layout preciso
- tutti i dettagli di layout, compresi i font, sono salvati con il documento
- il documento può essere presentato solo su un medium specifico (es. foglio A4)

In confronto, HTML è magro:

- l'autore rinuncia al controllo per avere in cambio la portabilità
- HTML rappresenta solo il contenuto e la sua struttura logica

Marcatura: testo + marcatori

sorgente:

```
Questo è il <em>primo</em> esempio
```

il browser mostra:

Questo è il *primo* esempio

em sta per **emphasis**

Elementi e tag

Questo è il $\overbrace{\langle \text{em} \rangle \text{primo} \langle / \text{em} \rangle}$ elemento esempio

``: start tag

``: end tag

Elementi e tag (ii)

Gli elementi possono **racchiudere** del testo (ed altri elementi)

È il *primo giorno di primavera...*

stronger emphasis

emphasis

(È il *primo **giorno** di primavera...*)

Oppure possono essere *vuoti*

Con questo vado a capo.
 Ora sono su una nuova riga

(Con questo vado a capo.

Ora sono su una nuova riga)

Elementi vuoti

Due maniere equivalenti di esprimere un elemento senza contenuto:

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

è equivalente a

```
<div/>
```


Scheletro minimo di un documento html

```
<html>
  <head>
    <title>Titolo</title>
  </head>
  <body>
    Qui va il mio contenuto
  </body>
</html>
```

- tutto è racchiuso nell'elemento **html**
- il quale contiene l'elemento **head** e l'el. **body**
- ogni pagina deve avere un titolo

Provate! Copiate questo html in un file con estensione “.html” e visitate quel file in un browser

Intestazioni e paragrafi

Sei livelli di importanza, da **h1** a **h6**

```
<h1>Un'intestazione importante</h1>  
<h2>E una un po' meno importante</h2>
```

Ogni paragrafo dovrebbe iniziare con il tag `<p>`

```
<p>Bla bla, il primo paragrafo</p>  
<p>Bla bla, il secondo</p>
```

Html non rispetta lo spazio bianco

Link

```
<a href='http://www.disney.it/pippo.html'>Tutto su Pippo</a>
```

target *label*

Posso usare url **relative**

Il `diario` delle lezioni

Lo start tag `<a>` contiene un **attributo**

- **nome**: "href"
- **valore**: la url (assoluta o relativa)

Sintassi di un tag

```
< name attribute0 attribute1 ... >
```

```
<html lang='it'>
```

```

```

Gli attributi specificano informazioni ausiliarie, oppure **meta**informazioni

Altri attributi che hanno senso in un link

```
<a accesskey='h' tabindex='1' title="Return to first page"  
  href='/'>Home page</a>
```

Accesskey: rende il link attivabile con *a/t-h*

Tabindex: rende il link selezionabile con *tab*

Title: descrive cosa trovo se seguo il link

Tre tipi di liste (i)

lista “coi pallini” (**unordered list**)

```
<ul>
  <li>the first list item</li>
  <li>the second list item</li>
  <li>the third list item</li>
</ul>
```

- the first list item
- the second list item
- the third list item

Tre tipi di liste (ii)

lista “coi numeri” (**ordered list**)

```
<ol>  
  <li>the first list item</li>  
  <li>the second list item</li>  
  <li>the third list item</li>  
</ol>
```

1. the first list item
2. the second list item
3. the third list item

Tre tipi di liste (iii)

description list

```
<dl>
  <dt>the first term</dt>
  <dd>its description</dd>

  <dt>the second term</dt>
  <dd>its description</dd>
</dl>
```

the first term

its definition

the second term

its definition

A capo e non a capo

Come forzare un “a capo”

Si usa l'elemento `
`

```
Nocturnis ego somniis<br/>  
iam captum teneo! Iam volucrem sequor<br/>  
te per gramina Martii<br/>  
campi, te per aquas, dure, volubiles!<br/>
```

A capo e non a capo

Come forzare un “a capo”

Si usa l'elemento `
`

```
Nocturnis ego somniis<br/>  
iam captum teneo! Iam volucrem sequor<br/>  
te per gramina Martii<br/>  
campi, te per aquas, dure, volubiles!<br/>
```

Come **non** andare a capo

Il browser può andare a capo dopo ogni spazio; per impedirlo si usa il Non-Breaking Space

...la Coca** **Cola è una bevanda gassata...

HTML Entities

Le **entità** html sono simboli introdotti per nome, come ` `;

less than	<code>&lt;</code>	<
greater than	<code>&gt;</code>	>
ampersand	<code>&amp;</code>	&
em dash	<code>&mdash;</code>	—
euro	<code>&euro;</code>	€
a grave	<code>&agrave;</code>	à
a acute	<code>&aacute;</code>	á
a umlaut	<code>&auml;</code>	ä
⋮	⋮	⋮

Vedi <http://www.w3.org/MarkUp/Guide/Advanced.html> per una lista più completa

Ancora su html

I commenti non vengono mostrati dal browser

```
<!-- questo e' un commento -->
```

L'elemento `pre` è per testo `pre`formattato

```
<pre>
#include <stdio.h>
int main() {
    printf("Hello, world!\n");
    return 0;
}
</pre>
```

Inserire le immagini

```
<img src='pippo.png'  
  alt='Effige di Pippo'  
  title='Pippo è il mio personaggio preferito'  
  width='120' height='100' />
```

- src: url o pathname del file grafico; obbligatorio
- alt: testo **alternativo**: obbligatorio
- title: testo esplicativo, facoltativo
- width, height: facoltativi ma fortemente consigliati

Un “bottone” è semplicemente un elemento **a** con dentro un elemento **img**

```
<a href='http://www.disney.it/pippo.html'><img  
src='pippo.png' alt='' /></a>
```

Tabelle in html

Sono usate per presentare dati

Year	Sales
2000	€18M
2001	€25M
2002	€36M

in html:

```
<table>
<tr> <th>Year</th> <th>Sales</th> </tr>
<tr> <td>2000</td> <td>&euro; 18M</td> </tr>
<tr> <td>2001</td> <td>&euro; 25M</td> </tr>
<tr> <td>2002</td> <td>&euro; 36M</td> </tr>
</table>
```

tr table row

td table data

th table heading

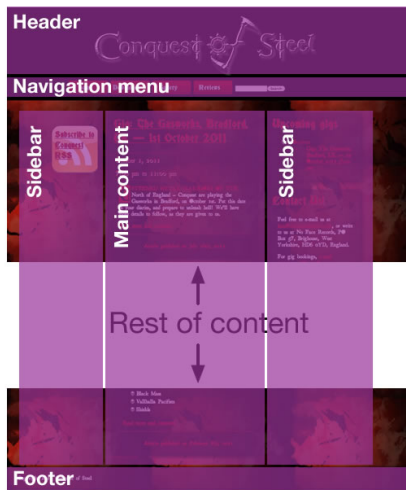
Tabelle in html, cont.

Spesso sono state usate non per rappresentare dati in forma tabellare, ma per **impaginare**

- testo su più colonne
- “box” di testo incorniciati
- layout della pagina
- effetti grafici di ogni tipo

Questo è considerato da cavernicoli e **da evitare!!!**

Struttura della pagina

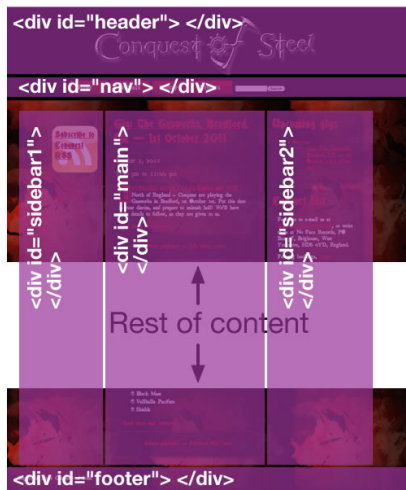


Struttura della pagina in HTML4

```
<body>
  <div id='header'>
    <!-- header content goes in here -->
  </div>
  <div id='nav'>
    <!-- navigation menu goes in here -->
  </div>
  <div id='sidebar1'>
    <!-- sidebar content goes in here -->
  </div>
  <div id='main'>
    <!-- main page content goes in here -->
  </div>
  <div id='sidebar2'>
    <!-- sidebar content goes in here -->
  </div>
  <div id='footer'>
    <!-- footer content goes in here -->
  </div>
</body>
```

Struttura della pagina in HTML4

(Assumendo di avere applicato gli stili CSS appropriati)



Struttura della pagina in HTML5


```
<body>
  <header>
    <!-- header content goes in here -->
  </header>
  <nav>
    <!-- navigation menu goes in here -->
  </nav>
  <section id='sidebar1'>
    <!-- sidebar content goes in here -->
  </section>
  <section id='main'>
    <!-- main page content goes in here -->
  </section>
  <aside>
    <!-- aside content goes in here -->
  </aside>
  <footer>
    <!-- footer content goes in here -->
  </footer>
</body>
```

Struttura logica e struttura fisica

In origine, gli elementi HTML descrivevano solo la struttura

- h2: “questa è una **intestazione** di livello 2”
- em: “questo testo deve avere **enfasi**”
- ul: “questa è una **lista**”

Ben presto, gli utenti desideravano un maggiore controllo:

- “questa intestazione è **centrata** e in **Times-Roman** dimensione **28pt**”
- “questo testo è in **corsivo**”
- “questi elementi di lista sono indentati **7mm** e usano  come pallini”

Elementi orientati al layout

Gli inventori di HTML risposero con **nuovi elementi** orientati al layout fisico

```
<center>
  <font size='2' family='Times-Roman' color='#112233'>
    Ma <i>che <blink>cosa</blink> abbiamo</i> combinato?!?
  </font>
</center>
```

Male!

Markup: semantic or presentation?

Presentation markup (Avoid!)

```
<font size='7'>Introduzione</font><br/>  
Testo introduttivo<br/><br/>  
<font size='6'>Secondariamente</font><br/>  
Bla bla<br/>
```

Semantic markup (Good!)

```
<h1>Introduzione</h1>  
<p>Testo introduttivo</p>  
<h2>Secondariamente</h2>  
<p>Bla bla</p>
```

Markup: semantic or presentation? (ii)

```
un punto<br />
```

```
un altro<br />
```

```
e un altro ancora<br />
```

```
<ul>
```

```
  <li>un punto</li>
```

```
  <li>un altro</li>
```

```
  <li>e un altro ancora</li>
```

```
</ul>
```

Esempi di marcatura semantica

Un sito di ricette

```
<h2>Patate fritte</h2>
```

```
<p>Serve: 3 persone</p>
```

```
<p>Difficoltà: facile</p>
```

```
<h3>Ingredienti</h3>
```

```
<ul>
```

```
  <li>1Kg patate</li>
```

```
  <li>1l olio extra-vergine di oliva</li>
```

```
</ul>
```

```
<h3>Procedimento</h3>
```

```
<p>Scaldare l'olio in una padella...</p>
```

Un po' più di semantica

```
<h2 id='title'>Patate fritte</h2>
<p class='servings'>Serve: 3 persone</p>
<p class='difficulty'>Difficoltà: facile</p>
<h3>Ingredienti</h3>
<ul id='ingredients-list'>
  <li>1Kg patate</li>
  <li>1l olio extra-vergine di oliva</li>
</ul>
<h3 id='process'>Procedimento</h3>
<p>Scaldare l'olio in una padella...</p>
```

id e class

id: identifica **un** elemento

class: etichetta una **classe** di elementi

Ancora più semantica

```
<div class='recipe'>
  <h2 id='title'>Patate fritte</h2>
  <p id='servings'>Serve: 3 persone</p>
  <p id='difficulty'>Difficoltà: facile</p>
  <div class='ingredients'>
    <h3>Ingredienti</h3>
    <ul>
      <li>1Kg patate</li>
      <li>1l olio extra-vergine di oliva</li>
    </ul>
  </div>
  <div id='process'>
    <h3>Procedimento</h3>
    <p>Scaldare l'olio in una padella...</p>
  </div>
</div>
```

XML: marcatura ad hoc

```
<?xml version='1.0' encoding='utf-8' ?>
<recipe>
  <title>Patate fritte</title>
  <servings>Serve: 3 persone</servings>
  <difficulty>Difficoltà: facile</difficulty>
  <ingredients>
    <ingredient>1Kg patate</ingredient>
    <ingredient>1l olio extra-vergine di oliva</ingredient>
  </ingredients>
  <process>
    <p>Scaldare l'olio in una padella...</p>
  </process>
</recipe>
```

La guerra dei browser

La metà degli anni '90 vede Microsoft e Netscape combattere per il predominio dei rispettivi browser

Una parte della strategia: estensioni proprietarie a HTML: nuovi elementi, nuovi attributi

La **portabilità** venne distrutta

Molti siti erano realizzati in due versioni: IE e Firefox

Problemi:

- il costo aumenta enormemente
- ... e tutti gli altri browser?
- ... presenti e futuri?
- il markup fisico aumenta il “peso” delle pagine

Regola d'oro

Una pagina web ben scritta funziona bene su tutti i browser
ragionevolmente recenti

“Questa pagina è ottimizzata per XYZ”

Regola d'oro

Una pagina web ben scritta funziona bene su tutti i browser
ragionevolmente recenti

“Questa pagina è ottimizzata per XYZ”

= “Questa pagina l’ho testata solo con XYZ”

Regola d'oro

Una pagina web ben scritta funziona bene su tutti i browser
ragionevolmente recenti

“Questa pagina è ottimizzata per XYZ”

= “Questa pagina l’ho testata solo con XYZ”

“Si consiglia una risoluzione di 1024x768”

Regola d'oro

Una pagina web ben scritta funziona bene su tutti i browser
ragionevolmente recenti

“Questa pagina è ottimizzata per XYZ”

= “Questa pagina l’ho testata solo con XYZ”

“Si consiglia una risoluzione di 1024x768”

= “Il mio schermo ha una risoluzione di 1024x768”

La reazione del World-Wide Web Consortium (W3C)

Separazione struttura logica (HTML) da presentazione (Cascading Style Sheets, CSS)

Introduzione di standard: HTML 4.0, XHTML 1.0,...

Ma gli standard sono inutili se sono “lettera morta”

Verso il 2000, appaiono i primi **browser che implementano gli standard:**

- Internet Explorer 5 per Macintosh
- Internet Explorer 6 per Windows
- Opera 7
- Mozilla, Netscape 7
- Safari

Diverse versioni di HTML

Gli standard **correnti** di HTML sono:

- HTML 4.01 strict, transitional, frameset
- XHTML 1.0 strict, transitional, frameset
- HTML5

Quale versione di HTML scegliere?

- HTML 4.01
- XHTML 1.0 strict, transitional
- XHTML 1.0 frameset
- **HTML5: OK**

Sintassi e validazione

La sintassi di HTML è definita in maniera precisa e formale.

- ogni documento HTML deve soddisfare questa definizione
- la qual cosa si può validare automaticamente
- i documenti non validi producono una lista di messaggi di errore

Validare una singola pagina è facile - essere certi che l'output di un app web sia sempre valido è più difficile

`http://validator.w3.org/`

`http://html5.validator.nu/`

I browser sono lassisti

La maggior parte dei documenti in realtà **non** sono validi:

- gli autori sono negligenti
- i documenti sono validati guardandoli in un browser
- HTML generato in automatico spesso non è valido

Ciononostante, molte pagine funzionano

- i browser fanno tutto il possibile
- non danno **mai** un messaggio di errore

Pessimo HTML

```
<h2>Pessimo HTML</h1>
<li><a>No, non va tanto</b> bene.
<li><i>Anzi, va abbastanza
    <g>male</g></em>
</ul>
Ma il browser si <a naem=lkjh>arrangia
```

Pessimo HTML

- No, non va tanto bene.
- *Anzi, va abbastanza male* Ma il browser si *arrangia*

Qual'è il problema?

- si promuove cattivo HTML
- browser diversi sono “astuti” in maniera diversa
- è difficile usare documenti non validi per processarli automaticamente con altri strumenti

Un appropriato DOCTYPE

Per poter validare un documento occorre **dichiarare** per quale standard è scritto

La dichiarazione **doctype** serve a questo

Ad esempio, per un documento HTML5

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>
```

Per fortuna il doctype di HTML5 è più semplice:

```
<!DOCTYPE html>
```

La presenza di un doctype corretto fa eseguire i browser in **standards mode** (più affidabile); altrimenti usano il **quirks mode**

Scheletro di documento HTML5

```
<!DOCTYPE html>
<html lang='it'>
<head>
  <meta charset="UTF-8" />
  <title>Il mio titolo</title>
</head>
<body>
  <p>Qui ci va il mio documento</p>
</body>
</html>
```

Perché aderire agli standard W3C?

- Ben supportati oggi su Firefox, IE 6+, Safari, Opera
- A prova di futuro
- Markup leggero → più veloce
- Compatibili con i client più strani
- Accessibile

Leggi “99.9% of Websites Are Obsolete” di Zeldman

Character sets and encodings

Character sets: ASCII

ASCII: 128 simboli

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	BS	RS	US
2	SPC	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Non comprende le accentate....

Character sets: IBM OEM

0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	☺	☹	♠	♣	♠	♣	♠	♣	↑	↓	→	←	♀	♂	☾	☼
1	▶	◀	↕	!!	¶	§	=	±	↑	↓	→	←	♀	♂	▲	▼
2		!	"	#	\$	%	&	'	<	>	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	Δ
8	ç	ü	é	â	ä	à	â	ç	ê	ë	è	ï	î	ÿ	Ä	Å
9	É	á	í	ó	ô	ö	ò	û	ü	ö	ü	½	¼	¾	¼	¾
A	á	í	ó	ú	ñ	ñ	ñ	ó	ü	ü	ü	½	¼	¾	¼	¾
B	▨	▩	▪													
C	▨	▩	▪													
D	μ	τ	π	π	ε	σ	μ	τ	π	π	ε	σ	μ	τ	π	π
E	α	β	γ	π	ε	σ	μ	τ	π	π	ε	σ	μ	τ	π	π
F	≡	±	≥	≤	ρ	σ	μ	÷	≈	°	·	√	∞	∅	€	∩

Estende ASCII con un arbitrario mix di caratteri occidentali e grafici

Character sets: UNICODE

Tentativo di costruire **singolo char set per tutte le lingue**

Associa ad ogni carattere un **code point** (un numero)

“A” corrisponde a U+0065 (decimale)

I primi 127 code points coincidono con ASCII

Encodings

Un **character set encoding** è un codice per mappare una sequenza di **byte** su una sequenza di **code points**

UTF-16: assegnamo 16 bit per ogni carattere (2 versioni: little e big endian)

UTF-8: usa 1 byte per i caratteri ascii

ISO-8859-1 (latin-1): usa 1 byte, codifica ASCII e caratteri occidentali

ISO-8859-15: come latin-1, con l'aggiunta di "€"

Se non conosco l'encoding...

... non so come interpretare un file!

Alcuni esempi

Perché

```

                P e r c h ? ?
UTF-8          50 65 72 63 68 c3 a9

```

```

                P e r c h ?
ISO-8859-1     50 65 72 63 68 e9

```

```

                P     e     r     c     h     ?
UTF-16 little endian  ff fe 50 00 65 00 72 00 63 00 68 00 e9 00

```

```

                P     e     r     c     h     ?
UTF-16 big endian    fe ff 00 50 00 65 00 72 00 63 00 68 00 e9

```

Altro esempio

Specificare l'encoding

Attraverso HTTP:

```
Content-Type: text/html; charset=utf-8
```

Nel documento HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Page Title</title>
  </head>
  <body>
    Ora posso scrivere direttamente “è” senza bisogno di “&egrave;”
  </body>
</html>
```

Inserire caratteri Unicode arbitrari

A equivale ad “A”

Alcuni caratteri utili:

’ ’ apice singolo sinistro (o apostrofo)

“ “ apici doppi sinistri

” ” apici doppi destri

Esercizi

- Sperimenta con tutto quanto visto finora
- Visita il tuo sito preferito e esamina il codice HTML
- Copia un articolo in HTML
- Scarica HTML Tidy e prova ad usarlo

Altri esercizi: trova il tuo posto nel mondo

Procurati accesso su un computer dotato di un web server (es. Apache)

Trova la **root** del tuo web server

Metti un file **index.html** nella root con il tuo nome

Connettiti con un browser

Verifica il tuo HTML con un validatore (es. validator.w3.org)

Trova nell'**access log** la traccia dei tuoi accessi

Html Forms

```
<form action='/search' method='get'>  
  <input type='text' name='q' />  
  <input type='submit' />  
</form>
```


Html Forms

```
<form action='/search' method='get'>  
  <input type='text' name='q' />  
  <input type='submit' />  
</form>
```



```
GET /search?q=ornitorinco HTTP/1.1  
Host: www.example.com
```

Html Forms

Content and controls

```
<form action='/login' method='post'>
  <p>Login:   <input type='text'   name='login'   value='' /></p>
  <p>Password: <input type='password' name='password' value='' /></p>
  <p><input type='submit'   value='Dite amici ed entrate' /></p>
</form>
```

Login:

Password:

Html Forms

Content and controls

```
<form action='/login' method='post'>
  <p>Login:   <input type='text'   name='login'   value='' /></p>
  <p>Password: <input type='password' name='password' value='' /></p>
  <p><input type='submit'   value='Dite amici ed entrate' /></p>
</form>
```

Login:

Password:

POST /login HTTP/1.1

Host: www.example.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 26

login=amici&password=amici

Un esempio di controllo

```
<input type='text' name='login' value='' />
```

`type` vari tipi di controlli: text, password, hidden...

`name` identifica il controllo

`value` il valore *iniziale*

Ogni controllo ha un valore *iniziale* e uno *corrente*

Aggiungere *label* e *id*

```
<form id='login-form' action='/login' method='post'>
  <p><label for='login'>Login:</label>
    <input type='text' id='login' name='login' value='' /></p>
  <p><label for='password'>Password:</label>
    <input type='password' id='password' name='password' value='' /></p>
  <p><input type='submit' value='Dite amici ed entrate' /></p>
</form>
```

Aggiungere un *fieldset*

```
<form id='login-form' action='/login' method='post'>
  <fieldset>
    <legend>Dite amici ed entrate</legend>
    <p><label for='login'>Login:</label>
      <input type='text' id='login' name='login' value='' /></p>
    <p><label for='password'>Password:</label>
      <input type='password' id='password' name='password' value='' /></p>
    <p><input type='submit' value='Submit' /></p>
  </fieldset>
</form>
```

Dite amici ed entrate

Login:

Password:

Control types: buttons

Tre tipi di bottoni

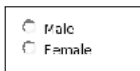
- submit buttons: submits a form
 - viene invocata la *action* della form
 - si carica una nuova pagina
- push buttons: no default behavior; => Client-side scripts

```
<input type='submit' value='Dite amici ed entrate' />  
<button value='cliccami' onclick='alert("zot!")' />
```

Checkboxes, radio buttons

Radio buttons: alternative

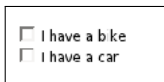
```
<form>
<input type='radio' name='sex' id='sex-male' value='male' />
<label for='sex-male'>Male</label> <br />
<input type='radio' name='sex' id='sex-female' value='female' /> Female
<label for='sex-female'>Female</label> <br />
</form>
```



A rectangular box containing two radio button inputs. The first input is selected (filled) and is labeled "Male". The second input is unselected (empty) and is labeled "Female".

Check boxes: opzioni

```
<form>
<input type='checkbox' name='bike' id='have-bike' />
<label for='have-bike'>I Have a bike</label> <br />
<input type='checkbox' name='car' id='have-car' />
<label for='have-car'>I Have a car</label> <br />
</form>
```



A rectangular box containing two checkbox inputs. Both inputs are unselected (empty) and are labeled "I have a bike" and "I have a car" respectively.

Menu a discesa

```
<select name='cars'>  
  <option value='volvo' checked='checked'>Volvo</option>  
  <option value='saab'>Saab</option>  
  <option value='fiat'>Fiat</option>  
  <option value='audi'>Audi</option>  
</select>
```



Editor di testo

Singola riga: `<input type='text' ... />`

Più righe:

```
<textarea rows='10' cols='30'>
The cat was playing in the garden.
</textarea>
```



File select

Permette di passare file all'applicazione

```
<input type='file' name='f' />
```

Campi invisibili

```
<input name='invisibile'  
      type='hidden'  
      value='un valore nascosto!' />
```

Cosa succede quando premo il bottone?

```
<form action='autentica' method='post'>  
  ...  
  <input type='submit' value='Dite amici ed entrate' />  
</form>
```

input type submit: crea un bottone che manda i dati... dove?

- action: url che indica dove verranno spediti i dati (default: la url della form stessa)
- method: il metodo HTTP con cui li spediremo (default: GET)

Due metodi: “get” e “post”

Con “**get**”:

- i dati sono visibili sulla url
/pippo/autentica?login=amici&password=xyz
- non posso mandare grandi quantità di dati

Con “**post**”:

- i dati non sono visibili sulla url
(ma sono lo stesso visibili con packet sniffing!)
- non c'è limite alla quantità di dati
- non posso fare un bookmark del risultato

“get” e “post”, cont.

Semanticamente:

GET: operazione *idempotente*
(cerco un libro in un catalogo, ottengo sempre la stessa risposta)

POST: operazione *NON idempotente*
(ordino un libro; se eseguo la query 10 volte mando 10 ordini)