

HTML (Hyper Text Markup Language)

- E' un linguaggio per la descrizione formale di **ipertesti multimediali** (e, in generale, interfacce utente grafiche)
- Utilizzato nel WWW come rappresentazione delle pagine (tipo MIME text/html)
- Portabile e leggibile (basato su testo)
- Principali versioni:
 - HTML 2.0 (rfc 1866) Novembre 1995
 - HTML 3.2 (w3c) Gennaio 1997
 - HTML 4.0 (w3c) Dicembre 1997
 - HTML 4.01 (w3c) Dicembre 1999 – vers. finale

Esempio di file HTML

```
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//IT>
<HTML>
<HEAD>
<TITLE>Universita` e Ricerca sul WWW</TITLE>
</HEAD>

<BODY bgcolor="#FFFFFF">

<H1 ALIGN=center> UNIVERSIT&Agrave; E RICERCA SUL WWW</H1>
<HR SIZE=4 WIDTH=60% ALIGN=CENTER>

<P>
<UL>
<LI><A HREF="http://cru1.dsi.uniroma1.it/">CRUI</A> Conferenza
dei Rettori delle Universit&grave; Italiane
<LI><A HREF="http://www.murst.it/">MURST</A> Ministero
dell'Universita` e della Ricerca Scientifica e Tecnologica
</UL>

</BODY>
</HTML>
```

HTML come applicazione SGML

- SGML (Standard Generalized Markup Language) è un sistema per la descrizione di **linguaggi di markup**:
 - un linguaggio di markup è un linguaggio per la descrizione di documenti, basato su annotazioni (markup) che servono per rappresentare elementi strutturali, di presentazione, ecc.
- HTML è un'applicazione SGML (cioè un linguaggio descrivibile tramite SGML)
- Come tutte le applicazioni SGML, HTML è definito da:
 - SGML Declaration, che specifica i caratteri e delimitatori validi
 - Document Type Definition (DTD), che definisce la sintassi dei costrutti di markup
 - La specifica della semantica dei costrutti di markup

Caratteri

- **Character Set** (insieme ordinato di caratteri astratti)
 - Vista l'universalità del web, un set che contenga solo i caratteri ASCII non è sufficiente.
 - HTML usa UCS (Universal Character Set – ISO 10646), che contiene esattamente tutti i caratteri Unicode.
- **Character Encoding** (corrispondenza tra sequenze di caratteri e sequenze di byte)
 - In HTML possono essere usate diverse codifiche (esempio: ISO-8859-1, noto anche come Latin1)
 - Esistono diverse modalità per specificare la codifica usata in un documento HTML

Character References

- Una codifica può rappresentare direttamente solo un sottoinsieme del set di caratteri UCS
- I caratteri che non hanno rappresentazione diretta possono essere indicati tramite i *character references* (rappresentazioni basate su sequenze di escape)
- Esempi:

Carattere	Sequenze di escape	
&	<code>&amp;</code>	<code>&#38;</code>
<	<code>&lt;</code>	<code>&#60;</code>
>	<code>&gt;</code>	<code>&#62;</code>

ELEMENTI e TAG

- Un documento contiene una serie di ELEMENTI (titoli, paragrafi, liste, ecc.)
- Un tag è un simbolo usato per rappresentare e delimitare gli elementi.
- Esempi:

```
<P>Questo e' un paragrafo</P>
```

```
<P>Paragrafo contenente del <EM>testo  
enfaticizzato</EM></P>
```

```
<BR>
```

Sintassi generale degli elementi

<nome_elemento attributi > contenuto </ nome_elemento >

Tag di inizio

Tag di fine

- Per alcuni elementi il tag di fine è opzionale (finiscono col tag di fine dell'elemento successivo)
- Per gli elementi HEAD e BODY anche il tag di inizio è opzionale
- Per alcuni elementi il contenuto è vuoto

TAG con attributi

- Alcuni elementi ammettono attributi, che vengono specificati nel TAG di inizio con una sintassi del tipo:

nome = valore

- Il valore deve essere racchiuso tra apici singoli o doppi (a meno che non sia costituito solo da caratteri base)
- Esempi:

```
<img src='http://home/mydir/xxx.gif'>
```

```
<input type=image src="map.gif">
```

Commenti

- Vengono delimitati da:

`<!--` (inizio)

`-->` (fine)

- Possono essere inseriti dovunque

- Esempi:

```
<!--Ecco un commento-->
```

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

Alcuni Tipi di dato base HTML

- **Stringhe di testo**
- **Identificatori** (iniziano per lettera, possono contenere caratteri alfanumerici e i caratteri speciali - _ : .)
- **URI**
- **Colori:** possono assumere due forme
 - il nome del colore (Es. **Black**, **Silver**, ecc.)
 - un carattere # seguito da un numero esadecimale su 6 cifre

Es: Black = "#000000"

 Gray = "#808080"

Lunghezze

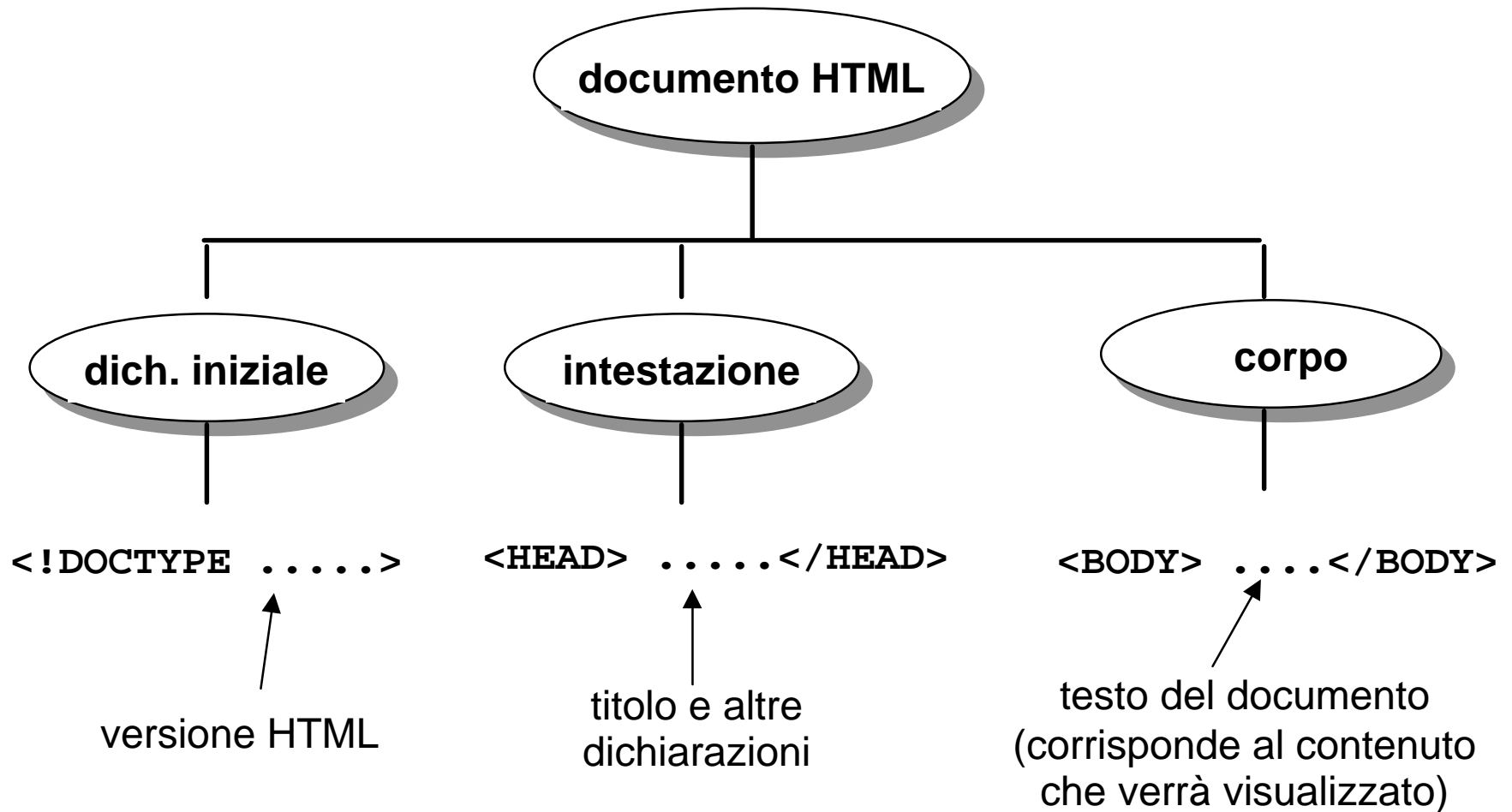
- Possono assumere diverse forme:
 - 50 (un numero di pixel)
 - 50% (una percentuale dello spazio disponibile)
 - 50* (una lunghezza relativa)

I 3 tipi di lunghezza possono convivere: prima vengono allocati i pixel e le percentuali. Lo spazio rimasto viene diviso proporzionalmente alle lunghezze relative.

Esempio: uno spazio di 200 pixel deve essere diviso in 4 campi le cui lunghezze sono descritte da: 20,3*,30%,5*

=> Le lunghezze risultanti saranno (in pixel): 20,45,60,75

Struttura di un documento HTML



L'elemento iniziale

- Specifica
 - qual è il DTD della versione HTML usata
 - l'URI del DTD
- Esempi:

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

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

Gli Elementi dell'Intestazione

- **Titolo** (ogni documento HTML deve contenerne uno):

```
<TITLE>Documento di prova</TITLE>
```

- **Indirizzo base per gli URL relativi**

```
<BASE HREF="http://myhost/mydir/index">
```

- **Meta-informazioni sul documento**

- sono coppie nome-valore (NAME-CONTENT) che possono essere interpretate dal server o dal browser:

```
<META HTTP-EQUIV="Keywords" CONTENT="WWW">
```

```
<META NAME="AUTHOR" CONTENT="R. Sisto">
```

Testo Formattato

- **Titoli di capitolo** (section headings)

Sono previsti 6 livelli di titoli denominati H1 ... H6

```
<H1>Gli animali</H1>
```

```
<H2>I mammiferi</H2>
```

- **Paragrafi**

```
<P>Testo del paragrafo:  
puo' estendersi su piu' linee</P>
```

- **Testo pre-formatto**

```
<PRE>
```

```
xxxx
```

```
    yyyy
```

```
      zzzz
```

```
</PRE>
```

Testo Formattato (II)

- **Interruzioni di linea**

- Non corrispondono alle interruzioni del testo HTML
- Vengono inserite con un apposito TAG:

Essere o non essere.
Questo e' ...

- **Barre orizzontali**

<HR>

<HR SIZE=4 WIDTH=60% ALIGN=CENTER>

Uso dei Font

- Testo *enfattizzato*

Testo `enfattizzato`

- Testo *in corsivo*

Testo `<I>in corsivo</I>`

- Testo **in grassetto**

Testo `in grassetto`

- Testo tipo macchina da scrivere

Testo `<TT>tipo macchina da scrivere</TT>`

- Testo di diverse **dimensioni e colori**

Testo di diverse `dimensioni e colori `

Liste

- Esempio di Lista ordinata (numerata): ordered list (OL)

```
<OL>
```

```
<LI>Primo elemento
```

```
<LI>Secondo elemento
```

```
</OL>
```

- Cambiando i TAG di inizio e fine si ottengono altri tipi di lista:

 lista non ordinata

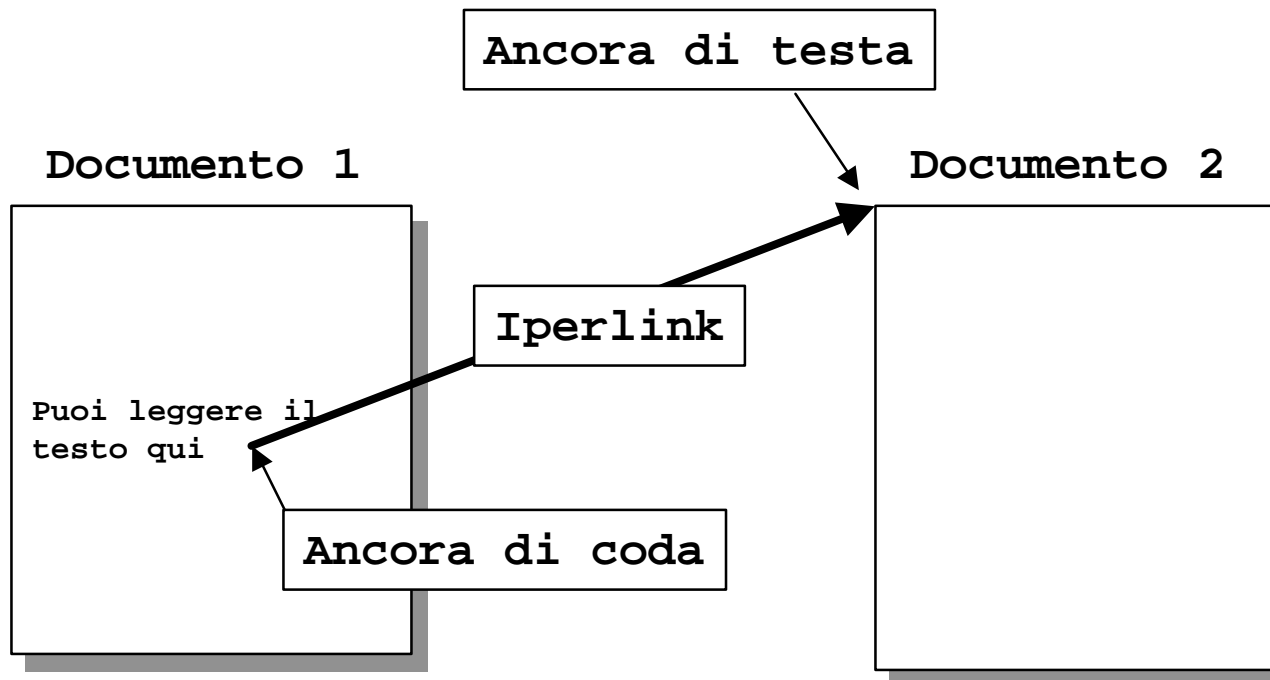
<DIR> lista a elenco (può essere multicolonna)

<MENU> lista a menù

<DL> lista a definizione. Invece di usa i tag <DT> (definition term) e <DD> (definition description)

Iperlink

- Sono relazioni tra due **ancore**: quella di testa e quella di coda
- Permettono la navigazione



Ancore di testa

- Vi si fa riferimento usando un URL, opzionalmente seguito da un indicatore di frammento, che specifica una parte o vista della risorsa

<http://www.polito.it/corsi.html#p2>

URL

Frammento

- Per definire l'inizio di un frammento all'interno di un documento HTML si usa un elemento ancora:

```
<A NAME=p2>
```

Inserimento di Iperlink

- Per inserire un iperlink navigabile con un click del mouse si inserisce un elemento ancora contenente la specificazione dell'ancora di testa cui si vuole puntare:

```
<A HREF="http://www.polito.it/corsi.html#p2">
```

- Le **immagini** sono iperlink la cui risorsa (ancora di testa) viene caricata automaticamente (senza click del mouse):

```
<IMG SRC="aircraft.gif" ALIGN=TOP>
```

```
<IMG SRC="triangle.xbm" ALT="Warning:"> Be sure  
to..
```

Tabelle

- Una tabella è una struttura per allineare elementi in righe e colonne
- Ogni tabella può avere una didascalia (caption)
- Sono configurabili vari attributi. Per esempio:
 - `width` (larghezza tabella)
 - `border` (larghezza bordi)
- Si possono formattare in vario modo le celle
- Si possono distinguere celle di tipo header (`<TH>`) e celle di tipo data (`<TD>`).

Esempio di Tabella

```
<TABLE border="1"
summary="This table gives some statistics about fruit
flies: average height and weight, and percentage
with red eyes (for both males and females).">
<CAPTION><EM>A test table with merged cells</EM></CAPTION>
<TR><TH rowspan="2"><TH colspan="2">Average
<TH rowspan="2">Red<BR>eyes
<TR><TH>height<TH>weight
<TR><TH>Males<TD>1.9<TD>0.003<TD>40%
<TR><TH>Females<TD>1.7<TD>0.002<TD>43%
</TABLE>
```

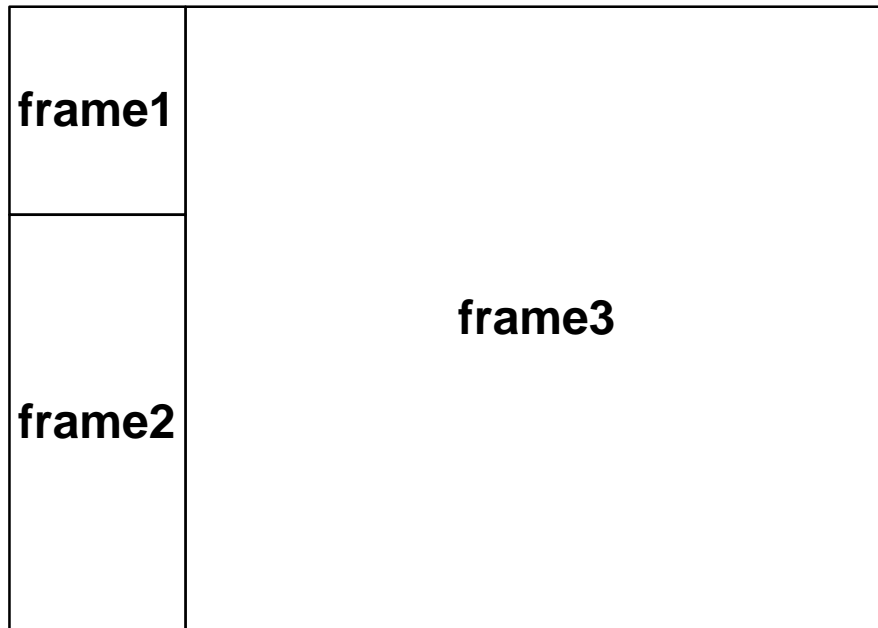
A test table with merged cells

	Average		Red eyes
	height	weight	
Males	1.9	0.003	40%
Females	1.7	0.002	43%

Frame

- Permettono di presentare documenti costituiti da viste multiple in frame indipendenti
- Il layout dei frame viene specificato da un elemento FRAMESET che *sostituisce* l'elemento HTML
- Un singolo FRAMESET definisce un layout a matrice o griglia rettangolare, in cui ogni casella contiene un elemento FRAME o un FRAMESET annidato.
- Annidando i FRAMESET si possono formare strutture complesse

Esempio



```
<FRAMESET cols="20%, 80%">  
  <FRAMESET rows="100, 200">  
    <FRAME src="frame1.html">  
    <FRAME src="frame2.html">  
  </FRAMESET>  
  <FRAME src="frame3.html">  
</FRAMESET>
```

- Matrice 1X2 il cui primo elemento è una matrice 2X1

Principali Attributi

- FRAMESET

- rows= *lista delle lunghezze delle righe* (se assente, 1 sola riga)
- cols= *lista delle larghezze delle colonne* (se assente, 1 sola c.)

- FRAME

- src= *URI del contenuto iniziale del frame*
- noresize
- scrolling= “yes”, oppure “no”, oppure “auto”
- frameborder= “1” (frame con bordo), oppure “0” (senza bordo)
- marginwidth= *larghezza del margine interno*
- marginheight= *altezza del margine interno*

Collegamento tra frame diversi

- Negli elementi che causano l'apertura di un nuovo documento (A, LINK, AREA, ecc.) è possibile specificare il frame dove esso deve essere aperto, con l'attributo

`target= nome del frame`

- Esempio:

```
<A HREF="nuovo.html" target="left"> apertura </A>
```

- Per aprire tutti i nuovi documenti in uno stesso altro frame si pone l'attributo target nell'elemento BASE.

Identificazione dei frame

- Un frame può essere identificato con il proprio nome, assegnatogli nel tag di inizio con l'attributo name
- In alternativa, valgono i seguenti nomi predefiniti:

<code>_self</code>	Il frame stesso in cui si trova il riferimento
<code>_parent</code>	Il frame genitore nella gerarchia di annidamento
<code>_top</code>	Il frame di primo livello nella gerarchia di annidamento (la finestra principale)
<code>_blank</code>	Una nuova finestra

Contenuto alternativo

- Un elemento FRAMESET può contenere un elemento NOFRAMES per specificare cosa deve essere visualizzato nel caso in cui non siano supportati i frame
- Esempio:

```
<NOFRAMES>  
<P>This frameset document contains:  
<UL>  
<LI><A href="contents_of_frame1.html">Some neat contents</A>  
<LI><IMG src="contents_of_frame2.gif" alt="A neat image">  
<LI><A href="contents_of_frame3.html">Some other neat  
  contents</A>  
</UL>  
</NOFRAMES>
```

Inclusione di Oggetti Multimediali

- Fino alla versione 3, gli unici oggetti multimediali che potevano essere inclusi in un documento HTML erano le immagini (elementi IMG) e gli applet Java (elementi APPLET)
- Dalla versione 4 è stato introdotto l'elemento OBJECT, che consente maggiore flessibilità:
 - Permette di trattare uniformemente tutti gli oggetti esterni multimediali (compresi immagini e applet)
 - Permette di includere applet scritti in altri linguaggi (non Java)
 - Si presta ad includere anche tipi di oggetti che saranno definiti in futuro

L'Elemento OBJECT

- Permette di specificare:
 - L'implementazione dell'oggetto (il codice)
 - » **classid** **URI del codice**
 - » **codetype** **content type del codice**
 - I dati associati all'oggetto (il file dell'immagine, la serializzazione dell'applet Java da ricostruire, ecc.)
 - » **data** **URI dei dati**
 - » **type** **content type dei dati**
 - Eventuali dati necessari a run time (tipicamente dati per l'inizializzazione degli applet)
 - » **Elementi PARAM annidati**
 - La Modalità di visualizzazione
 - » **height, width, tabindex**

Esempio: Inclusione di Immagini

```
<P>Here's a photo of my family at the lake:  
<IMG  
  src="http://www.somedomain.com/Ian/vacation/family.png"  
  alt="A photo of my family at the lake.">
```

Dati Immagine

Testo alternativo

```
<P>Here's a photo of my family at the lake:  
<OBJECT  
  data="http://www.somedomain.com/Ian/vacation/family.png"  
  type="image/png">  
  A photo of my family at the lake.  
</OBJECT>
```

Esempio: Inclusione di Applet Java

```
<APPLET code="Bubbles.class" width="500" height="500">  
Java applet that draws animated bubbles.  
</APPLET>
```

Implementazione Applet

Testo alternativo

```
<OBJECT codetype="application/java"  
classid="java:Bubbles.class"  
width="500" height="500">  
Java applet that draws animated bubbles.  
</OBJECT>
```

Esempio: Inclusione di Applet Java con Parametri

```
<APPLET code="AudioItem" width="15" height="15">  
<PARAM name="snd" value="Hello.au|Welcome.au">  
Java applet that plays a welcoming sound.  
</APPLET>
```

```
<OBJECT codetype="application/java"  
classid="AudioItem"  
width="15" height="15">  
<PARAM name="snd" value="Hello.au|Welcome.au">  
Java applet that plays a welcoming sound.  
</OBJECT>
```

Form

- Un FORM è un modulo che l'utente può compilare tramite il browser
- A compilazione terminata il browser invia il modulo compilato al server, che accede ad una risorsa e restituisce come risposta una pagina HTML
- Struttura di un form:

```
<FORM ACTION="http://www.org.sample" METHOD=GET>
```

...

...

```
</FORM>
```

Metodo HTTP usato

Risorsa cui accedere

Campi del FORM ("controls")

Campi INPUT

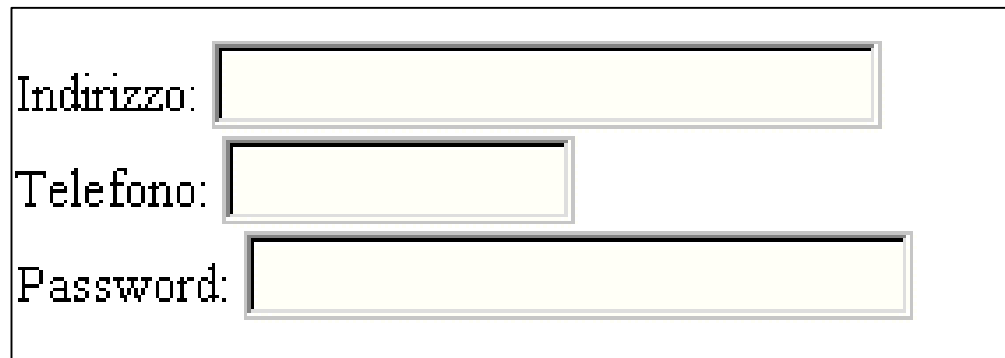
- Sono generici campi per l'input di dati.
- Vengono creati con elementi INPUT (senza contenuto)
- Possono assumere diverse forme:
 - Text Field/Password Field
 - Check Box
 - Radio Button
 - Image Pixel
 - Hidden Field
 - Button (Submit/Reset/Generic)
 - File Selection

Text Field e Password Field

Indirizzo: `<INPUT TYPE=TEXT NAME=indirizzo>
`

Telefono: `<INPUT NAME=telefono SIZE=10 MAXLENGTH=10>`

Password: `<INPUT TYPE=PASSWORD NAME=passwd>`



Indirizzo:

Telefono:

Password:

Check Box

`<p>A quali generi sei interessato?
`

`<INPUT TYPE=CHECKBOX NAME=genere VALUE=west>western
`

`<INPUT TYPE=CHECKBOX NAME=genere VALUE=thr>thriller
`

`<INPUT TYPE=CHECKBOX NAME=genere VALUE=fant>fantasy
`

A quali generi sei interessato?

western

thriller

fantasy

Radio Button

```
<p>Corso di appartenenza:<BR>
```

```
<INPUT TYPE=RADIO NAME=corso VALUE=inf>INF<BR>
```

```
<INPUT TYPE=RADIO NAME=corso VALUE=eln>ELN<BR>
```

```
<INPUT TYPE=RADIO NAME=corso VALUE=alt CHECKED>Altro<BR>
```

Corso di appartenenza:

INF

ELN

Altro

Button

```
<INPUT TYPE=SUBMIT VALUE=INVIA>
```

```
<INPUT TYPE=RESET VALUE=AZZERA>
```

```
<INPUT TYPE=BUTTON VALUE=OK>
```



File Selection

<p>Upload del file:

```
<INPUT TYPE=FILE NAME=filename VALUE="pippo.txt">
```

Upload del file:

Altri campi

- Oltre agli elementi INPUT sono previsti altri elementi per realizzare altri tipi di campi:
 - Menù di selezione
 - Text Area
 - Altri tipi di Bottoni

Menù di selezione

<p>Corso di appartenenza:

```
<SELECT NAME=corso SIZE=3>
```

```
<OPTION VALUE=inf>INF
```


```
<OPTION VALUE=eln>ELN
```

```
<OPTION VALUE=alt SELECTED>Altro
```

```
</SELECT>
```

Con size=1 (default)
diventa un menù a tendina

Corso di appartenenza: 

Corso di appartenenza: 

Text Area

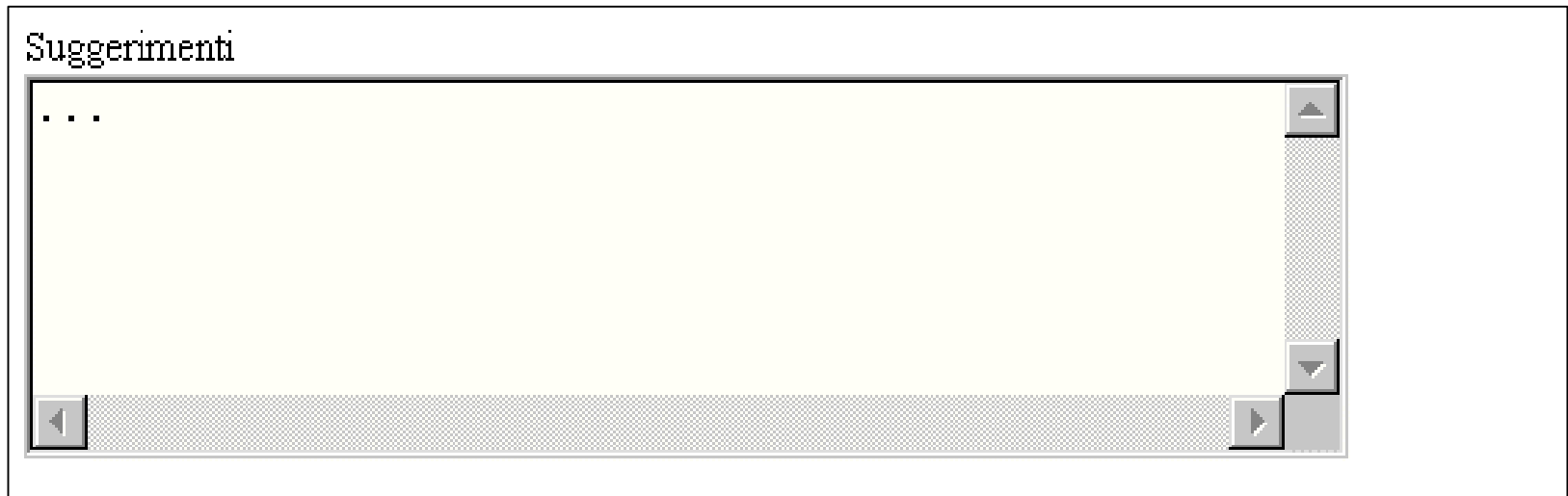
```
<p>Suggerimenti
```

```
<TEXTAREA NAME=suggerimenti ROWS=6 COLS=50>
```

```
...
```

```
</TEXTAREA>
```

Il testo eventualmente inserito qui
è il contenuto iniziale della text area



Suggerimenti

...

Campi disabilitati e read-only

- E' possibile disabilitare e rendere read-only un campo usando gli attributi binari disabled e readonly

```
<INPUT disabled type=TEXT name="bevanda" value="acqua">
```

```
<INPUT readonly type=TEXT name="bevanda" value="acqua">
```

Form Strutturati

- I campi dei FORM possono essere raggruppati usando l'elemento `<FIELDSET>`
- Ogni fieldset può avere una didascalia, assegnata con un elemento `LEGEND`

```
<FIELDSET>  
  <LEGEND> Dati Anagrafici </LEGEND>  
  ...  
</FIELDSET>
```

Codifica e Invio di un FORM

- Quando l'utente sottomette il FORM (tramite un bottone submit) si svolgono le seguenti operazioni:
 - Vengono identificati i cosiddetti “successful controls”
 - Viene costruito il Form Data Set (sequenza di coppie nome-valore, una per ciascun successful control)
 - Il Form Data Set viene codificato
 - Il Form Data Set codificato viene inviato tramite il metodo HTTP specificato
- La specifica HTML 4 definisce un insieme minimo di codifiche e metodi di invio, ma non ne esclude altri.

Codifica Standard

- La codifica standard (default) è descritta dal tipo MIME **application/x-www-form-urlencoded**
- Il Form Data Set viene convertito in una stringa con una sintassi che la rende inseribile in un URL:
nome1=valore1&nome2=valore2& . . .
- Per permettere l'uso dei caratteri speciali e dei caratteri '=' e '&', questi vengono trasformati con sequenze di escape:
 - lo spazio viene trasformato in '+'
 - i caratteri speciali vengono trasformati in '%xx'

Altre Codifiche

- L'attributo `enctype` specifica la codifica da usare
- Lo standard prevede esplicitamente che possa essere usata la codifica **multipart/form-data** (rfc2388):
 - Risulta più efficiente nel caso di dati binari o non ASCII di grosse dimensioni
 - Si basa sul tipo MIME multipart (sequenza di parti, ciascuna con propri attributi: Content-type, Charset, ecc.)
 - Ogni parte rappresenta un successful control e deve avere un attributo

Content-Disposition: form-data; name="*nome_campo*"

Esempio: Form

```
<FORM action="http://server.com/cgi/handle"
  enctype="multipart/form-data" method="post">
<P>
What is your name? <INPUT type="text" name="submit-name"><BR>
What files are you sending? <INPUT type="file"
  name="files"><BR>
<INPUT type="submit" value="Send"> <INPUT type="reset">
</FORM>
```

Esempio: Codifica del Data Set

```
Content-Type: multipart/form-data; boundary=AaB03x
```

```
--AaB03x
```

```
Content-Disposition: form-data; name="submit-name"
```

```
Larry
```

```
--AaB03x
```

```
Content-Disposition: form-data; name="files"; filename="file1.txt"
```

```
Content-Type: text/plain
```

```
... contents of file1.txt ...
```

```
--AaB03x--
```

Metodi di Invio

- **Metodo Get**

- Viene eseguito il metodo GET, inviando come URI la concatenazione dell'attributo action con il Data Set codificato, separati da un carattere '?'
- L'unica codifica ammessa in questo caso è `application/x-www-form-urlencoded`

- **Metodo Post**

- Viene eseguita una transazione POST usando il valore dell'attributo action e inviando un messaggio che contiene la codifica del Data Set

Inclusione di Script

- Un documento HTML può contenere codice
 - scritto in un qualunque linguaggio di scripting (ovviamente il browser deve essere in grado di interpretarlo)
 - in grado di accedere agli elementi HTML del documento e di intercettare eventi (mouse, tastiera, ...)
- Gli script sono tipicamente usati:
 - come meccanismi di Client-Side Programming (controllo dei valori inseriti in un form prima dell'invio, realizzazione di vere e proprie GUI interattive)
 - per modificare “al volo” il contenuto del documento HTML quando questo viene caricato (pagine dinamiche)

Script Eseguiti al Caricamento

- Vengono eseguiti una sola volta, al caricamento del documento
- Vengono inseriti in elementi SCRIPT
- Possono avere un contenuto alternativo, specificato in un corrispondente elemento NOSCRIPT

```
<SCRIPT type="text/javascript">
function my_onload() {
. . .
}
var win = window.open("http://...")
if (win) win.onload = my_onload
</SCRIPT>
<NOSCRIPT>
<P><A href="http://..."> click </A>
</NOSCRIPT>
```

```
<SCRIPT type="text/vbscript"
src="http://someplace.com/vbcalc">
</SCRIPT>
```

Supporto per Pagine Dinamiche

- Se uno script eseguito al caricamento produce come output un testo HTML,
 - l'elemento SCRIPT contenente lo script viene sostituito dal testo prodotto durante il caricamento
 - Il nuovo testo viene interpretato (può a sua volta contenere elementi script)
- Il caricamento può quindi dar luogo a sostituzioni recursive

```
<TITLE>Test Document</TITLE>  
<SCRIPT type="text/javascript">  
document.write("<p><b>Hello World!</b>")  
</SCRIPT>
```

Script Eseguiti al Verificarsi di Eventi

- Vengono inseriti come valori di particolari attributi degli elementi HTML ai quali sono associati gli eventi.
- Esempi di attributi:
 - onload (termine del caricamento di un frameset o di una finestra)
 - onunload
 - onclick (click del mouse sull'elemento)
 - ondblclick (doppio click del mouse sull'elemento)

```
<INPUT NAME="num"  
onchange="if (!checkNum(this.value, 1, 10))  
{this.focus();this.select();} else {thanks()}"  
VALUE="0">
```

La Specifica del Linguaggio

- Negli elementi SCRIPT
 - E' obbligatorio inserire un attributo type che specifica il linguaggio usato
- Se vengono usati script attivati in corrispondenza di eventi
 - E' necessario che il documento abbia un linguaggio di scripting di default, che può essere specificato

» **come meta-informazione:**

```
<META http-equiv="Content-Script-Type" content="text/tcl">
```

» **con un attributo dell'header HTTP**

```
Content-Script-Type: text/javascript
```

Accesso agli Elementi HTML

- Ogni linguaggio di scripting abilitato per l'inclusione in HTML ha meccanismi propri di accesso agli elementi del documento
- In generale, il riferimento ad un elemento avviene sempre tramite il suo attributo name

Il Problema dello Stile di Presentazione

- I markup di HTML consentono di specificare
 - la struttura logica del documento (paragrafi, titoli, ecc.)
 - alcuni aspetti della presentazione (colore, font del testo, ecc.)
- Tutte le altre scelte sulla presentazione (interlinea, spaziatura tra i caratteri, ecc.) sono lasciati al browser
- La necessità di curare meglio la presentazione ha portato a situazioni di incompatibilità e complessità:
 - estensioni proprietarie di HTML
 - uso massiccio di tabelle, immagini-spaziatrici, codice

Style Sheet

- Sono stati introdotti con HTML 4 per
 - potenziare la descrizione degli aspetti di presentazione/stile
 - permettere la separazione tra presentazione e contenuto
- Le specifiche di presentazione possono essere scritte
 - usando diversi linguaggi (quello specificato dal w3c è CSS: Cascading Style Sheet)
 - sia nel documento HTML che in file separati. Se il linguaggio lo prevede, possono essere create gerarchie di file di stile (cascading)
- Può essere lasciata la scelta all'utente di selezionare uno tra diversi stili di presentazione

Specifica del Linguaggio

- Per usare gli style sheet è necessario specificare il linguaggio adottato
- La specifica può avvenire in un elemento META del documento o nell'intestazione HTTP

```
<META http-equiv="Content-Style-Type" content="text/css">
```

```
Content-Style-Type: text/css
```

Specifiche di Stile Interne

- Possono essere annidate negli elementi, tramite l'**attributo** style:

```
<P style="font-size: 12pt; color: red"> paragrafo con  
testo 12 punti rosso
```

- Possono essere incluse nell'intestazione (HEAD) tramite l'**elemento** style:

```
<HEAD>  
  <STYLE type="text/css">  
    H1 {border-width: 1; border: solid; text-align:  
      center}  
  </STYLE>  
</HEAD>
```

Specifiche di Stile Interne

- Un elemento style può essere applicato selettivamente:

```
<HEAD>
  <STYLE type="text/css">
    H1.myclass {border-width: 1; border: solid; text-align: center}
  </STYLE>
</HEAD>
<BODY>
  <H1 class="myclass"> Titolo cui applicare lo stile </H1>
  <H1> Titolo cui non si applica lo stile </H1>
</BODY>
```

Specifiche di Stile Esterne

- Vengono scritte in file esterni. Esempio: mystyle.css

```
H1.myclass {border-width: 1; border: solid; text-align: center}
```

```
P.special {color: green; border: solid red}
```

- Vengono associate al documento tramite elementi LINK posti nell'intestazione:

```
<HEAD>
```

```
<LINK href="mystyle.css" rel=stylesheet type="text/css">
```

```
</HEAD>
```

```
<BODY>
```

```
  <H1 class="myclass"> Titolo cui applicare lo stile </H1>
```

```
  <P class="special"> Paragrafo cui applicare lo stile
```

```
</BODY>
```