



HTML & CSS

design and build websites

JON DUCKETT



Sapienza Università di Roma
Corso di laurea in Ingegneria Informatica e Automatica

Linguaggi e tecnologie per il Web

a.a. 2020/2021

Parte 1 HTML

Riccardo Rosati
Dipartimento di Ingegneria informatica, automatica e gestionale
Sapienza Università di Roma

Sommario

1. World Wide Web
2. Ipertesti e HTML
3. HTML di base
4. Form
5. HTML5

1. World Wide Web

Il World Wide Web

- World Wide Web = sistema di accesso a Internet basato sul protocollo HTTP
 - insieme di protocolli e servizi (HTTP, FTP, ...)
 - insieme di tool per l'accesso (es. web browser)
- Basato sulla metafora dell'**ipertesto**
 - linguaggio HTML
- Distinzione tra Internet e WWW
 - Internet = rete
 - WWW = sistema di accesso alla rete

Architettura del web

Architettura client-server:

- Web client (es. web browser)
 - inoltra richieste di **risorse** (documenti, file, ecc.) ad una macchina (web server)
- Web server
 - risponde alle richieste dei web client
- Sia il web server che il web client sono programmi in esecuzione su macchine connesse a Internet
- Web server e web client comunicano in base al protocollo HTTP

Architetture client-server

- Basate sul concetto di richiesta di servizio (client) e di fornitura di servizio (server)
- Enormemente diffuse in informatica, in particolare nelle applicazioni di rete
 - HTTP
 - FTP
 - DNS
 - PPP
 - Proxy
 -

Protocolli

- Protocollo = insieme di convenzioni (o regole) per lo scambio di informazioni
- protocolli di “basso” livello per Internet:
 - determinano le modalità di comunicazione tra i nodi della rete
 - TCP/IP
- protocolli di “alto” livello:
 - determinano il formato dei messaggi e le modalità di scambio dei messaggi
 - HTTP, FTP, SMTP, TELNET...

Il protocollo HTTP

HTTP = HyperText Transfer Protocol

- Client-server
 - ogni interazione è una richiesta (messaggio ASCII) seguita da una risposta (messaggio tipo MIME)
- 7 metodi nativi:
 - GET
 - HEAD
 - PUT
 - POST
 - DELETE
 - LINK
 - UNLINK

Il protocollo HTTP

- Client-server
- HTTP è connectionless = **non** si instaura una connessione prima di richiedere un servizio
- FTP:
 - richiesta connessione
 - instaurazione connessione
 - richiesta servizio 1
 - fornitura servizio 1
 - richiesta servizio 2 ...
 - chiusura connessione
- HTTP:
 - richiesta servizio
 - fornitura servizio

URL

- In HTTP ogni interazione è relativa ad una URL (Uniform Resource Locator)
- La URL è un nome che identifica univocamente ogni risorsa disponibile sul web
- Ogni URL specifica:
 - il protocollo da utilizzare per il trasferimento della URL
 - il dominio, cioè il nome (simbolico) del computer su cui si trova il server (web server o altro server) che gestisce la risorsa
 - il nome, all'interno del dominio, della risorsa che si vuole accedere

Domain Name System (DNS)

- Sistema per introdurre nomi logici (o simbolici) ai computer su Internet
- IP (Internet Protocol):
 - l'indirizzo del computer è una sequenza di 4 numeri da 0 a 255
 - es. 151.100.16.20
- DNS:
 - l'indirizzo del computer è una stringa di caratteri
 - es. www.dis.uniroma1.it
- Il DNS è basato sul concetto di **dominio**

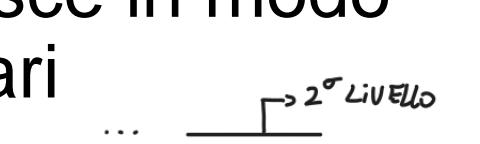
Domini

Domini radice o di primo livello:

- COM, ORG, NET, EDU, MIL, GOV, INT
- biletterale per ogni nazione (es. IT)

Per ogni dominio di primo livello:

- domini di secondo livello (es. IBM.COM,
VIRGILIO.IT)

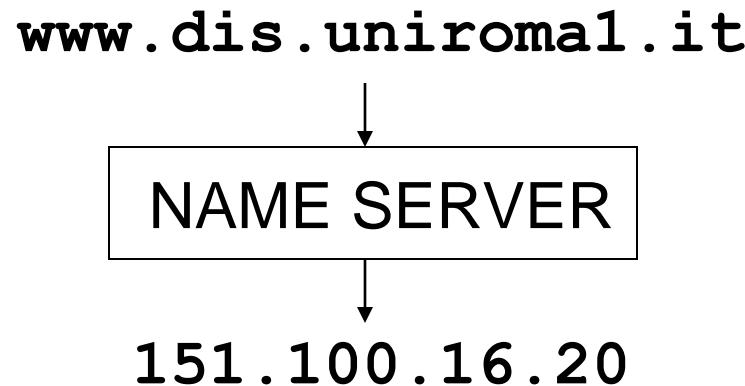
↳ finisce con .com
 - ogni dominio di primo livello gestisce in modo autonomo i propri domini secondari

↳ 2° LIVELLO
...
↳ 1° LIVELLO
 - domini di terzo livello, quarto, ecc.
- i nomi dei domini sono case-insensitive
- 
- ↳ gestione ad albero.

Name Server

- Occorre tradurre il nome simbolico in indirizzo IP
- NAME SERVER (o Domain Name Server)
- Si deve ricorrere ad un name server ogniqualvolta si fa uso di un nome simbolico
- Es. ogni web browser che richiede una URL, deve **prima** richiedere ad un name server l'indirizzo IP corrispondente al dominio nella URL:
- Dominio → **Name Server** → indirizzo IP

URL: esempio

URL: `http://www.dis.uniroma1.it/~rosati/index.htm`



- `~rosati/index.htm` è il nome (completo di percorso) del file corrispondente alla URL

Tipi di URL

Protocolli HTTP e FTP:

- Documenti ipertestuali (file HTML)
- Immagini
- Documenti multimediali (audio e video)
- Programmi
- File di altro genere

Altri protocolli: (es. mailto:)

- indirizzi di email
-

Domini, siti web e pagine web

Distinzione tra domini, web server, siti web e pagine web:

- **dominio** = computer dove gira un web server (“identifica” il web server)
- **sito web** = insieme di URL gestite da un’unica entità e organizzate in modo da essere accedute secondo un ordine logico
 - più siti web possono essere gestiti dallo stesso web server
- **pagina web** = singolo documento HTML
- **home page** = pagina iniziale di un sito web

2. Ipertesti e HTML

Ipertesti

Ipertesto = documento contenente:

- testo
- immagini
- audio
- video
- **collegamenti ipertestuali** = riferimenti ad altri documenti (o parti di documenti) ipertestuali

Collegamenti ipertestuali

- Sul World Wide Web, un collegamento ipertestuale ad un documento può essere specificato tramite la URL che corrisponde al documento
- Si possono usare le tecnologie informatiche per **accedere direttamente** ai documenti corrispondenti ai link mentre si legge un ipertesto
- (esempio: web browser)
- superamento dell'accesso “sequenziale” al testo

Linguaggio per ipertesti: HTML

- HTML = HyperText Markup Language
- Linguaggio standard per la specifica di documenti ipertestuali sul World Wide Web
- Linguaggio a marcatura, “figlio” di SGML (Standard Generalized Markup Language)
- Un documento HTML può contenere:
 - testo
 - link ipertestuali
 - immagini
 - link a risorse (URL) di ogni tipo

Breve storia di HTML (1)

- definito (insieme ad HTTP) da Tim Berners-Lee del CERN di Ginevra nel 1989
- scopo: permettere lo scambio dei dati sperimentali tra i fisici
- 1993: diffusione di Mosaic (web browser sviluppato da NCSA)
- 1995: fondazione del World Wide Web Consortium (W3C)
- 1999: standardizzazione di HTML 4.0
- 2000: standardizzazione di XHTML 1.0 (ridefinizione di HTML basata su XML)

Breve storia di HTML (2)

- 2004: viene fondato il WHATWG, Web Hypertext Application Technology Working Group
 - Apple, Mozilla Foundation, Opera Software, Google
- WHATWG nasce in contrapposizione con il futuro standard XHTML2 del W3C, che sarebbe stato un linguaggio non retrocompatibile con HTML 4
- ottobre 2014: standardizzazione di HTML5
- novembre 2016: standardizzazione di HTML 5.1
- dicembre 2017: standardizzazione di HTML 5.2
- maggio 2019: accordo tra W3C e WHATWG
 - Il «Living standard» di HTML di WHATWG diventa il riferimento comune dei due consorzi

Sintassi di HTML

- Documento HTML = testo ASCII
- contiene:
 - blocchi di testo
 - tag (marcature o “comandi”)
- tag = testo delimitato dai simboli “<” e “>”
- esempio:

`<nome-tag>`

Il concetto di tag

- TAG = “marcatura” (o marcatore)
- Un tag viene usato per **marcare** una parte di testo
- Tag:
 - di formattazione (per cambiare l’aspetto ad una parte di testo) (es.)
 - “semantici” (per dare un “significato” ad una parte di testo) (es. <a>)
- 2 tipi di tag:
 - tag di apertura (marcatore iniziale) <nome-tag>
 - tag di chiusura (marcatore finale) </nome-tag>

Attributi dei tag

- Ogni occorrenza di tag (di apertura) può contenere assegnazioni di **attributi**
- ogni tag ha un diverso insieme di possibili attributi
- assegnazione: nome-attributo = valore
 - es.
- struttura del tag con attributi:

<nome-tag attributo1 = valore1 attributo2 = valore2>

- alcuni attributi del tag sono **obbligatori** (vanno assegnati)

Elementi e tag

- Elemento = insieme formato da tag di apertura, testo e tag di chiusura corrispondente
- Esempio di elemento (font):
``
- In genere si usa il termine “tag” erroneamente, per indicare un elemento (composto da due tag)

Per HTML i tag sono case-insensitive (es.
`` e `` hanno lo stesso significato)

Semantica di HTML

Il “significato” di un documento HTML è dato da due componenti:

- aspetto del documento
- “contenuto” (rispetto ai tag) del documento

La semantica “immediata” di un documento HTML è la sua visualizzazione sul browser

- dipendente dal browser
- perdita (parziale) del significato legato al “contenuto”

Struttura di un documento HTML

```
    ↗ ELEMENTO RADICE
<html>      <-- inizio del documento

<head>

...
...          <-- intestazione del documento

</head>

<body>

...
...          <-- corpo del documento

...
</body>

</html>      <-- fine del documento
```

Informazione e meta-information

- corpo del documento = contiene l'informazione (il documento stesso)
- intestazione del documento = contiene **meta-information** (cioè informazioni **sul** documento)
 - esempi:
 - autore del documento
 - parole chiave
 - “titolo” del documento
 -

Contenuto e presentazione

- Problema: distinguere tra
 - **contenuto** del documento
 - **presentazione** (o aspetto) del documento
- E' molto importante poter individuare il contenuto di un documento indipendentemente dalla formattazione del documento
- Nelle intenzioni, HTML doveva mantenere separati i due aspetti. Nella realtà, non è così:
 - i marcatori sono usati anche per dare attributi di formattazione al testo
 - i vari browser “interpretano” il codice HTML in modo diverso

HTML e browser HTML

I principali web browser, specie in passato, hanno influito sull'evoluzione di HTML:

- imponendo nuovi elementi del linguaggio
- “rifiutando” (cioè non supportando) nuovi elementi del linguaggio
- Problemi principali:
 - differente interpretazione di HTML
 - presenza di vecchie versioni dei browser

Creare documenti HTML

Documento HTML = testo ASCII

(analogo ad un programma sorgente JAVA)

Modalità di creazione di un documento HTML:

- con un editor per testo ASCII (es. blocco note o Wordpad sotto Windows)
- con un “editor WYSIWYG” o “editor HTML” (es. FrontPage, Dreamweaver)
- con un editor di testi “normale” che permette di esportare i documenti in HTML (es. Word)

Editor HTML

- Permette di editare il documento vedendo direttamente come verrà visualizzato dal browser
- Facilità di utilizzo:
 - non è necessario conoscere il linguaggio HTML
- Limiti nella realizzazione:
 - non tutte le potenzialità di HTML possono essere utilizzate
- Editor HTML professionali possono essere utilizzati al massimo solo conoscendo il codice HTML

3. HTML di base

Sommario

- **intestazione**
- formattazione testo
- link
- oggetti e immagini
- tavole
- frame

Parte intestazione

- contiene una serie di informazioni necessarie al browser per una corretta interpretazione del documento, ma non visualizzate all'interno dello stesso:
- tipo di HTML supportato
- titolo della pagina
- parole chiave (per motori di ricerca)
- link base di riferimento
- stili (comandi di formattazione)

Parte intestazione

Elementi principali:

- DOCTYPE
- HTML
- HEAD
- TITLE
- META

Parte intestazione

```
<!DOCTYPE html>

<html>

<head>

<meta name="keywords" Content= "HTML, parte
intestazione, meta-information">

<meta name= "author" Content = "Riccardo Rosati">

<meta name="GENERATOR" content="Blocco note di
Windows 10">    ↴TOOL USATO PER SCRIVERE QUESTO
<title>Pagina web di prova </title>

</head>

.....
</html>
```

Parte intestazione

DOCTYPE:

```
<!DOCTYPE html>
```

- fornisce l'informazione sul tipo di documento
- deve essere il primo elemento ad aprire il documento
- è obbligatorio (in HTML5)
- (nelle versioni precedenti a HTML5 doveva indicare una DTD, dichiarazione di tipo di documento)

Parte intestazione

META:

```
<meta name="keywords" Content= "HTML, parte  
intestazione, meta-information">  
<meta name= "author" Content = "Riccardo Rosati">  
<meta name="GENERATOR" content="Blocco note di  
Windows 10">
```

- fornisce meta-informationi sul contenuto del documento
- usate dai motori di ricerca per classificare il documento
- non è obbligatorio

Parte intestazione

TITLE:

```
<title>Pagina web di prova </title>
```

- Titolo della pagina
- Compare sulla barra del titolo della finestra del browser
- Usato dai motori di ricerca
- Usato nella visualizzazione di “bookmark” (siti preferiti)

HTML di base

- intestazione
- **formattazione testo**
- link
- oggetti e immagini
- tavole
- frame

Corpo del documento

- Memorizza il contenuto del documento (la parte visualizzata all'utente del browser)

```
<body ..... >
```

```
.....
```

```
</body>
```

- gli attributi di `<body>` configurano alcuni parametri di visualizzazione del documento (in realtà il loro utilizzo è «deprecato»)

Header

- <H1>,<H2>,...,<H6>
- Permettono di inserire titoli (o intestazioni) all'interno del documento
- il testo tra <Hn>...</Hn> viene evidenziato dal browser
- 6 livelli di titoli
- 6 differenti livelli di enfatizzazione del testo

Enfatizzare il testo

, <I>, <U>:

- (bold):
 - permette di visualizzare testo in neretto
- <I> (italic):
 - permette di visualizzare testo in corsivo
- <U> (underlined):
 - permette di sottolineare il testo
- definiscono attributi **fisici** di formattazione

Attributi logici e fisici

- tag fisico = ha il compito di formattare il documento
- tag logico = dà una struttura al documento, ed è indipendente dalla visualizzazione
- esempio: elemento `<address>`
 - permette di specificare che una parte di testo è un indirizzo
 - viene visualizzato come testo in corsivo
 - per il browser è come usare `<i>`
 - ma `<ADDRESS>` aggiunge “semantica” al documento

Selezione del font

- Tag fisico
- deprecato da W3C: non dovrebbe mai essere usato (al suo posto vanno usate le proprietà di CSS)
- definisce il modo in cui deve essere visualizzata una parte di testo:
 - tipo di carattere
 - colore
 - grandezza

Attributi del tag

- FACE
 - determina il tipo di carattere (font) usato
 - font disponibili: courier, times, arial, verdana, ...
- SIZE
 - determina la grandezza dei caratteri
 - si esprime con numeri assoluti (da 1 a 7) o relativi
- COLOR determina il colore

esempio:

```
<FONT FACE="verdana" SIZE="+1" COLOR="green">  
testo in verde</FONT>
```

Apici e pedici

- <SUB> (subscript)
 - permette di scrivere testo come “pedici”
- <SUP> (superscript)
 - permette di scrivere “apici”
- esempio:

I₅ = 2⁴

viene visualizzato come

$$I_5 = 2^4$$

Testo preformattato

- Tag <PRE>
- permette di visualizzare il testo esattamente come è scritto (“preformattato”) nel file sorgente HTML
- il testo preformattato non viene “interpretato”

Testo preformatto con <PRE>

Codice HTML:

```
<PRE>
begin
  if
    then
end;
    I<SUB>5</SUB>      =  2<SUP>4</SUP>
</PRE>
```

Visualizzazione:

```
begin
  if
    then
end;
    I5      =  24
```

Stili logici

- <ADDRESS>
 - marcatura usata per indirizzi (mail, email, telefono,...)
- <BLOCKQUOTE>
 - usato per inserire nel testo citazioni da un altro testo o autore
- <CITE>
 - usato per la fonte della citazione
- e
 - “enfatizzano” il testo all’interno del tag
- <VAR> e <CODE>
 - utilizzati per righe di codice di programmazione

Separare e allineare il testo

- <P> (paragraph)
 - crea un paragrafo all'interno del testo
-
 (break)
 - interruzione di riga (“a capo”)
- <DIV>
 - usato per allineare il documento:
 - <DIV align = left> allinea a sinistra il testo
 - <DIV align = center> allinea al centro il testo
 - <DIV align = right> allinea a destra il testo

Righe orizzontali

- <HR> (horizontal row)
 - aggiunge una riga orizzontale al testo
 - usato per separare parti di testo
- attributi di <HR>:
 - ALIGN (left|center|right|) allineamento rispetto alla pagina
 - WIDTH lunghezza orizzontale (in pixel o in percentuale)
 - SIZE altezza della riga in pixel
 - COLOR colore della riga
 - NOSHADE elimina l'effetto 3D

Liste puntate

- **** (unordered list)
 - produce un elenco (lista) di elementi (parti di testo)
 - ogni elemento è evidenziato all'inizio da un simbolo grafico (di solito cerchietto o quadratino)
- **** (list item)
 - per identificare un elemento della lista
- **esempio:**

```
<UL>
<LI>Primo elemento </LI>
<LI>Secondo elemento </LI>
<LI>Terzo elemento </LI>
</UL>
```

Liste numerate

- **** (ordered list)
 - produce un elenco (lista) di elementi (parti di testo)
 - ogni elemento è evidenziato all'inizio dal numero d'ordine all'interno della lista
- **** (list item) (come per liste puntate)
- **esempio:**

```
<OL>
<LI>Primo elemento </LI>
<LI>Secondo elemento </LI>
<LI>Terzo elemento </LI>
</OL>
```

Liste numerate

Oltre al numero progressivo, si possono usare altre indicizzazioni per le liste puntate

Uso dell'attributo TYPE di :

- <OL TYPE=A> indica con lettere alfabetiche maiuscole
- <OL TYPE=a> indica con lettere alfabetiche minuscole
- <OL TYPE=I> indica con numeri romani maiuscoli
- <OL TYPE=i> indica con numeri romani minuscoli

Esempio

Esempio di liste annidate:

```
<ol>
<li>gruppo di nomi:
<ul>
<li>nome a </li>
<li>nome b </li>
</ul></li>
<li>gruppo di nomi:
<ul>
<li>nome c </li>
<li>nome d </li>
<li>nome e </li>
</ul></li>
</ol>
```

HTML di base

- intestazione
- formattazione testo
- link
- oggetti e immagini
- tavelle
- frame

Link ipertestuali

Tag <A> (anchor)

- permette l'inserimento di link ipertestuali all'interno del documento
- attributo principale: HREF (Hypertext reference)
 - specifica la URL che viene associata al link
- il testo tra <A> e viene associato a tale URL
 - cliccando su tale testo il browser accede alla URL
- Esempio:

```
<A HREF="http://www.virgilio.it">Visita  
virgilio.it</A>
```

Attributi del tag <a>

- 2 tipi di tag <a>:
- con attributo HREF:
 - aggiungono un link ipertestuale (esterno o interno al documento)
- con attributo NAME:
 - definiscono uno specifico punto del documento come un link interno
 - tale punto può essere direttamente acceduto attraverso un tag <A HREF...>
- altri attributi: TARGET, TITLE

Esempio

...

```
<a name="zona1"> zona 1 del documento  
raggiungibile direttamente </a>
```

...

```
<a href="#zona1">torna alla zona 1 del  
documento</a>
```

...

**Con l'anchor zona1 si può anche accedere direttamente
dall'esterno del documento:**

```
<a href="www.dis.uniroma1.it/index.html#zona1">  
vai alla zona 1 del documento index.html del sito  
dis.uniroma1.it </a>
```

Attributo target di <a>

Attributo TARGET di <A>:

- permette di specificare dove deve essere visualizzata la URL associata al link

valori principali di TARGET:

- _NEW: visualizza la URL collegata in una nuova finestra (o tab) del browser
- _PARENT: visualizza nella stessa finestra, eliminando tutti i frame presenti (vedere sezione sui frame)

Attributo title di <a>

Attributo TITLE di <A>:

- permette di specificare una informazione associata al link
- es. commento riguardante il link
- i browser visualizzano tale informazione (pop-up) quando il puntatore del mouse passa sopra al link

HTML di base

- intestazione
- formattazione testo
- link
- **oggetti e immagini**
- tavelle
- frame

Immagini

- HTML permette di inserire immagini nel documento
- Tag (singolo)
- permette di inserire nel documento una immagine, memorizzata in un file (o URL) a parte
- i browser riconoscono i principali formati immagine (es. JPG, GIF, BMP)

Attributi del tag

- SRC
 - specifica il nome della URL contenente l'immagine
 - es.
- WIDTH larghezza (in pixel o percentuale)
- HEIGHT altezza (in pixel o percentuale)
 - se WIDTH e HEIGHT mancano, l'immagine viene visualizzata nelle sue dimensioni originali
- ALT
 - permette di aggiungere un commento testuale associato all'immagine

Attributi del tag

- BORDER
 - spessore cornice dell'immagine (in pixel)
- HSPACE e VSPACE
 - distanze minime orizzontali e verticali (in pixel) dell'immagine dagli oggetti più vicini
- ALIGN
 - determina l'allineamento del testo rispetto all'immagine

L'attributo ALT

- Permette di aggiungere una informazione testuale all'immagine
- Il testo viene visualizzato dai browser (popup)
- Necessario per i browser solo testuali
- Oppure per la navigazione con immagini disabilitate

es.

L'attributo ALIGN

- determina l'allineamento del testo rispetto all'immagine
 - **ALIGN=top**: allinea la prima riga di testo sulla sinistra al top dell'immagine
 - **ALIGN=middle**: allinea la prima riga di testo sulla sinistra al centro dell'immagine
 - **ALIGN=bottom**: allinea la prima riga di testo sulla sinistra nella parte più bassa dell'immagine
 - **ALIGN=left**: allinea il testo sulla destra dell'immagine partendo dal top
 - **ALIGN=right**: allinea il testo sulla sinistra dell'immagine partendo dal top

Mappe cliccabili

- Una immagine può essere associata ad un link
- es.
- spesso si vogliono associare due o più link alla stessa immagine
 - associare zone diverse dell'immagine a diversi link
- **mappe cliccabili**
- 2 tipi:
 - lato server (poco diffuse)
 - lato client (USEMAP)

Mappe cliccabili (lato client)

```
<IMG SRC="img1.gif" WIDTH=400 HEIGHT=100  
      BORDER=0 usemap="#immagine1">  
  
<map name="immagine1">  
  <area shape="rect" alt="parte 1 immagine"  
        coords="0,0,200,100" href="doc1.htm"  
        title="parte 1 immagine">  
  <area shape="rect" alt="parte 2 immagine"  
        coords="201,0,400,100" href="doc2.htm"  
        title="parte 2 immagine">  
  <area shape="default" nohref>  
</map>
```

Generare mappe cliccabili

- Tipi di aree definibili con usemap:
 - rect
 - circle
 - poly
- difficili da definire a mano
- uso di programmi (editor di mappe)
 - es. MAPEDIT

Oggetti

- Tag <object> e <embed>
 - Permettono l'inclusione di oggetti (risorse) generici nel documento HTML
 - Esempio:

```
<embed type="text/html"  
src="page1.html" width="600"  
height="400">
```

Simboli speciali

- Come rappresentare simboli non-ASCII e simboli utilizzati da HTML?
- insieme di definizioni di simboli (ogni simbolo è rappresentato da un nome)
- l'invocazione di un simbolo predefinito inizia con “&” e termina con “;”
- esempio: © per rappresentare ©
- le entità si possono anche rappresentare mediante i loro codici unicode (in decimale o esadecimale (es.: { �)

Simboli speciali

- esempi: lettere accentate:
 - à à
 - è è
 - ´ é
 - ì ì
 - ò ò
 - ù ù
- il simbolo “&” si rappresenta con &

Il simbolo “<”

- < è un simbolo particolarmente speciale in HTML (apertura dei tag)
- < si rappresenta con <
- > si rappresenta con >

HTML di base

- intestazione
- formattazione testo
- link
- oggetti e immagini
- **tabelle**
- frame

Tabelle

- Rappresentano informazione in forma tabellare (righe e colonne) nei documenti HTML
- Molto utilizzate anche come strumento di formattazione di testi e/o immagini
- HTML permette una gestione piuttosto potente delle tabelle

Elementi relativi alle tavette

- TABLE
 - definisce la tabella
- TD
 - definisce un campo “dati” all’interno della tabella
- TR
 - suddivide i campi in righe all’interno della tabella
- TH
 - definisce campi intestazione all’interno della tabella
- THEAD, TBODY, TFOOT

NOTA BENE

NOTA: praticamente tutti gli attributi degli elementi delle tavole (tranne rowspan e colspan) presentati nel seguito sono deprecati o eliminati in HTML5 e non dovrebbero essere usati (al loro posto vanno usate opportune proprietà CSS)

L'elemento <table>

Racchiude tutta l'informazione relativa ad una tabella

- Esempio:

```
<TABLE WIDTH=300 HEIGHT=200>
```

.....

```
</TABLE>
```

Gli attributi di <table> settano proprietà globali della tabella

Dimensioni della tabella

Espresso in:

- pixel (punti)
 - es. `<table width=300 height=200>`
 - indipendente dalle dimensioni della finestra di visualizzazione
- percentuale della dimensione della pagina
 - es. `<table width="60%">`
 - dipendente dalle dimensioni della finestra di visualizzazione

La scelta tra i due tipi di dimensioni dipende dalla applicazione

Attributi di <table>

- WIDTH larghezza
- HEIGHT altezza (non dovrebbe essere usato)
- BORDER spessore del bordo (in pixel)
- BGCOLOR colore sfondo tabella
- SUMMARY testo che spiega il contenuto della tabella (per media non visuali)
- CELLSPACING distanza tra i campi (celle) della tabella
- CELLPADDING distanza in pixel tra il contenuto del campo e i margini del campo

L'elemento <TD>

<TD> permette la definizione di un singolo campo (cella)

- va usato per campi di tipo “dati”
- non va usato per campi di tipo intestazione di colonne
- esempio:

```
<TD width=100>prova</TD>
```

definisce una cella con contenuto prova

Attributi di <TD>

- WIDTH, HEIGHT non dovrebbero essere usati
- VALIGN (top|bottom|middle) allineamento verticale
- ALIGN (left|center|right) allineamento orizzontale
- BGCOLOR colore di sfondo della cella
- BACKGROUND motivo di sfondo della cella
- ROWSPAN, COLSPAN

L'elemento <TR>

- Divide le celle in righe
- Esempio:

```
<TABLE border=1 cellpadding=2>  
<TR>  
<TD>cella 1</TD>  
<TD>cella 2</TD>  
<TD>cella 3</TD>  
<TR>  
<TD>cella 1 riga 2</TD>  
<TD>cella 2</TD>  
<TD>cella 3</TD>  
</TABLE>
```

cella 1	cella 2	cella 3
cella 1 riga 2	cella 2	cella 3

Attributi di <TR>

- ALIGN (left|center|right) allineamento orizzontale delle celle che seguono <TR>
- VALIGN (top|middle|bottom) allineamento verticale
- BGCOLOR

Esempio

```
<TABLE WIDTH=300 HEIGHT=200>  
<TD width=100 VALIGN=TOP>  
Prova1</TD>  
<TD WIDTH=100 VALIGN=BOTTOM>  
Prova2</TD>  
<TD WIDTH=100 VALIGN=MIDDLE>  
Prova3</TD>  
</TABLE>
```

Prova1		Prova3
	Prova2	

Esempio

```
<TABLE WIDTH=300 HEIGHT=200 border=1>  
<TD width=100 ALIGN=RIGHT>  
prova1</TD>  
<TD WIDTH=100 ALIGN=CENTER>  
Prova2</TD>  
<TD WIDTH=100 ALIGN=LEFT>  
Prova3</TD>  
</TABLE>
```

Prova1	Prova2	Prova3
--------	--------	--------

L'elemento <TH>

- Come <TD> ma va usato per specificare campi **intestazione**
- permette di dare più “semantica” alla tabella
- da un punto di vista di presentazione, per i browser non c’è differenza
- stessi attributi di <TD>

Esempio

```
<TABLE border=1 cellpadding=2>
<TR>
<TH>nome</TH>
<TH>cognome</TH>
<TH>età</TH>
<TR>
<TD>Mario</TD>
<TD>Rossi</TD>
<TD>36</TD>
<TR><TD>Paola</TD>
<TD>Bianchi</TD>
<TD>32</TD>
</TABLE>
```

nome	cognome	età
Mario	Rossi	36
Paola	Bianchi	32

L'elemento <CAPTION>

- Permette di associare una didascalia alla tabella
- esempio:

```
<TABLE border=1 cellpadding=2>
<CAPTION>Elenco degli impiegati:</CAPTION>
<TR>
<TH>nome</TH>
<TH>cognome</TH>
<TH>età</TH>
<TR>
<TD>Mario</TD>
<TD>Rossi</TD>
<TD>36</TD>
<TD>Paola</TD>
<TD>Bianchi</TD>
<TD>32</TD>
</TABLE>
```

Elenco degli impiegati:

nome	cognome	età
Mario	Rossi	36
Paola	Bianchi	32

Elementi di raggruppamento righe

- <THEAD>, <TBODY>, <TFOOT>
- Permettono di suddividere l'informazione contenuta in una tabella per righe
- Permettono la gestione separata di intestazione e contenuto
- Permettono di raggruppare il contenuto della tabella in più gruppi (più occorrenze di <TBODY>...</TBODY>)
- Struttura non visualizzata dai browser (occorrono fogli di stile)

Esempio

```
<TABLE border=1 cellpadding=2>
<THEAD>
<TR><TH>nome</TH>
<TH>cognome</TH>
<TH>et&agrave;;</TH>
</THEAD>
<TBODY>
<TR>
<TD>Mario</TD>
<TD>Rossi</TD>
<TD>36</TD>
<TR>
<TD>Paola</TD>
<TD>Bianchi</TD>
<TD>32</TD>
</TBODY>
</TABLE>
```

Raggruppamento delle colonne

- <COLGROUP>, <COL>
- Permettono di suddividere l'informazione contenuta in una tabella per colonne
- Struttura non visualizzata dai browser (occorrono fogli di stile)

Celle variabili

- Una cella può occupare più righe o più colonne di una tabella
- Attributi ROWSPAN, COLSPAN di <TD>
- ROWSPAN = numero righe occupate dalla cella
- COLSPAN = numero colonne occupate dalla cella

Esempio

```
<TABLE border=1 cellpadding=2>
<TR>
<TH>nome</TH>
<TH>cognome</TH>
<TH>età</TH>
<TR>
<TD colspan=2>Rossi</TD>
<TD>36</TD>
<TR><TD>Paola</TD>
<TD rowspan=2>Bianchi</TD>
<TD>32</TD>
<TR><TD>Maria</TD>
<TD>36</TD>
</TABLE>
```

Nome	Cognome	Età
Rossi		36
Paola	Bianchi	32
Maria		36

HTML di base

- intestazione
- formattazione testo
- link
- oggetti e immagini
- tavelle
- frame

Frame

- Frame = riquadro (zona rettangolare) della finestra di visualizzazione del browser
- ogni frame è gestito in modo indipendente dal browser (come una finestra a sé stante)
- Nelle versioni precedenti di HTML erano presenti diversi elementi per organizzare la finestra in frame
- Nel seguito vedremo l'unico elemento tuttora supportato, l'elemento iframe

L'elemento <iframe>

- Questo elemento («inline frame») permette di definire un frame (riquadro) in qualsiasi punto di un documento HTML
- struttura:

```
<iframe src="http://www.uniroma1.it"  
width="500" height="300">  
codice HTML alternativo (per i browser  
che non supportano iframe  
</iframe>
```

4. Form

Form (modulo)

- usati per inviare informazioni via WWW
- il modulo viene compilato dall’utente (sul browser)
- quindi viene inviato al server
- un programma apposito sul server elabora il modulo
- in genere tale programma invia una “risposta” all’utente (pagina web, email, ecc.)

Form e CGI

- CGI (Common Gateway Interface): metodo inizialmente più usato per elaborare form
- si possono usare programmi lato server alternativi al CGI usando i linguaggi di scripting lato server (PHP, JSP, ASP, Node.js/JavaScript, ecc.)
- in teoria è possibile evitare l'uso di CGI o di qualunque programma lato server (es. invio diretto di email dal browser)
- in pratica, ciò è possibile solo per form estremamente semplici

L'elemento <form>

Apre e chiude il modulo e raccoglie il contenuto dello stesso

Esempio:

```
<FORM method="post"  
      action="http://www.dis.uniroma1.it/cgi-  
      bin/nome_script.cgi">
```

- **attributo ACTION:** specifica la URL della risorsa (programma) che elabora la form

L'attributo method

method=get

- i dati inseriti nella form vengono spediti al server e separati in due variabili
- le coppie nome-campo-form=valore-inserito compaiono alla fine della URL usata per l'invio del messaggio (i dati sono quindi visibili già nella URL)
- Esempio (Google):
`https://www.google.com/search?q=linguaggi+e+tecnologie+per+il+web&ie=utf-8&oe=utf-8&client=firefox-b`
- per questo metodo il numero massimo di caratteri contenuti nella form è circa 3000

L'attributo method

method=post

- i dati vengono ricevuti direttamente dal programma (lato server) senza un preventivo processo di decodifica
- questa caratteristica fa sì che lo script possa leggere una quantità illimitata di caratteri

Campi editabili del modulo

Vengono definiti tramite gli elementi:

- INPUT (campi editabili, checkbox, radio, ecc.)
- TEXTAREA (area di testo)
- SELECT (creazione di menu di opzioni)

L'elemento INPUT

- Permette l'inserimento di campi editabili e/o modificabili nel modulo
- Attributi principali:
 - TYPE: determina il tipo di campo
 - NAME
 - VALUE
 - MAXLENGTH

Attributo TYPE di <INPUT>

- Attributo TYPE: determina il tipo di campo
- Principali possibili valori di tale attributo:
 - HIDDEN
 - TEXT
 - PASSWORD
 - CHECKBOX
 - RADIO
 - SUBMIT
 - RESET
 - IMAGE

TYPE=“HIDDEN”

- Usato per dare informazioni “nascoste” al CGI (cioè al server)
- Il suo uso dipende dal tipo di CGI associato al modulo

TYPE=“HIDDEN”

Esempi:

```
<INPUT type="HIDDEN" name=MAILFORM_SUBJECT  
value="titolo del form">
```

- Questo codice determina il titolo (subject) del messaggio che verrà ricevuto via e-mail, e che conterrà il contenuto del modulo

```
<INPUT TYPE="HIDDEN" NAME=MAILFORM_URL  
VALUE="http://www.tuosito.it">
```

- dopo aver compilato e spedito correttamente il form, dà in risposta una pagina HTML successiva (es. pagina di conferma di invio modulo avvenuta)

TYPE=“TEXT”

```
<INPUT type="TEXT" name="nome"  
       maxlength="40" size="33"  
       value="inserisci nome">
```

- crea i tipici campi di testo
- usato soprattutto per informazioni non predefinite che variano di volta in volta
- attributi di <INPUT> nel caso TYPE=TEXT:
 - MAXLENGTH (num. max caratteri inseribili)
 - SIZE (larghezza campo all'interno della pagina)
 - VALUE (valore di default che compare nel campo)

TYPE="PASSWORD"

```
<INPUT type="PASSWORD" name="nome"  
       maxlength="40" size="33">
```

- crea i campi di tipo password
- vengono visualizzati asterischi al posto dei caratteri
- i dati NON vengono codificati (problema per la sicurezza)

TYPE="CHECKBOX"

```
<INPUT type="checkbox" name="eta"  
size="3" VALUE="YES" CHECKED>
```

- crea campi booleani (si/no)
- crea delle piccole caselle quadrate da spuntare o da lasciare in bianco
- VALUE impostato su YES significa che di default la casella è spuntata
- CHECKED controlla lo stato iniziale della casella, all'atto del caricamento della pagina

TYPE=“RADIO”

```
<INPUT type="RADIO" name="giudizio"  
value="sufficiente">
```

```
<INPUT type="RADIO" name="giudizio"  
value="buono">
```

```
<INPUT type="RADIO" name="giudizio"  
value="ottimo">
```

- usato per selezionare una tra alcune scelte
- tutte le scelte con lo stesso “name” (altrimenti una scelta non esclude le altre)

TYPE=“SUBMIT”

```
<INPUT type="SUBMIT" value="spedisci">
```

- crea un bottone che invia il modulo al server
- la lunghezza del bottone dipende dalla lunghezza del valore di VALUE

TYPE=“RESET”

```
<INPUT type="RESET" value="reimposta">
```

- crea un bottone che resetta i campi del modulo
- i dati inseriti vengono eliminati
- la lunghezza del bottone dipende dalla lunghezza del valore di VALUE

TYPE=“IMAGE”

```
<INPUT type="IMAGE" SRC="bottone.gif">
```

- come SUBMIT, ma crea un bottone tramite l’immagine (URL) specificata in SRC

L'elemento TEXTAREA

```
<TEXTAREA cols=40 rows=5 WRAP="physical"  
name="commento"> </textarea>
```

- utilizzato per commenti o campi che prevedono l'inserimento di molto testo
- WRAP="physical" stabilisce che qualora il testo inserito superi la larghezza della finestra, venga automaticamente riportato a capo

L'elemento SELECT

```
<SELECT size=1 cols=4 NAME="giudizio">
  <OPTION selected Value=nessuna>
  <OPTION value=buono> Buono
  <OPTION value=sufficiente> Sufficiente
  <OPTION Value=ottimo> Ottimo
</select>
```

- Permette la creazione di menu a tendina con scelte multiple

L'elemento FIELDSET

```
<fieldset name="giudizio">
    <legend>Dati dello studente:</legend>
    <input type="text" name="matricola">
    <input type="text" name="anno-iscrizione">
    <input type="text" name="numero-esami">
</fieldset>
```

- permette la creazione di un gruppo di campi all'interno della form
- l'elemento `<legend>` permette di inserire una descrizione (o titolo) per il gruppo di campi

Esempio di form

cognome e nome:	<input type="text"/>
data di nascita:	<input type="text"/>
CAP:	<input type="text"/> <input type="button" value="..."/>
codice fiscale:	<input type="text"/>
studente lavoratore:	<input type="checkbox"/>
foto:	<input type="button" value="Sfoglia..."/> Nessun file selezionato.
giudizio:	<input type="radio"/> sufficiente <input type="radio"/> buono <input checked="" type="radio"/> ottimo
descrizione lavoro:	<input type="text"/>
<input type="button" value="Invia form"/>	<input type="button" value="Reset form"/>

Codice HTML della form

```
<form action="" method="post" name="eseform"
      enctype="multipart/form-data">
<table border="1">
<tr>
<td align="right">cognome e nome:
<td><input type="text" name="nome" size="50" maxlength="50"
           required>
<tr>
<td align="right">data di nascita:
<td><input type="date" name="ddn">
<tr>
<td align="right">CAP:
<td><input type="number" name="cap" size="5" maxlength="5">
<tr>
<td align="right">codice fiscale:
<td><input type="text" name="cf" size="16" maxlength="16">
<tr>
<td align="right">studente lavoratore:
<td><input type="checkbox" name="lavoratore">
```

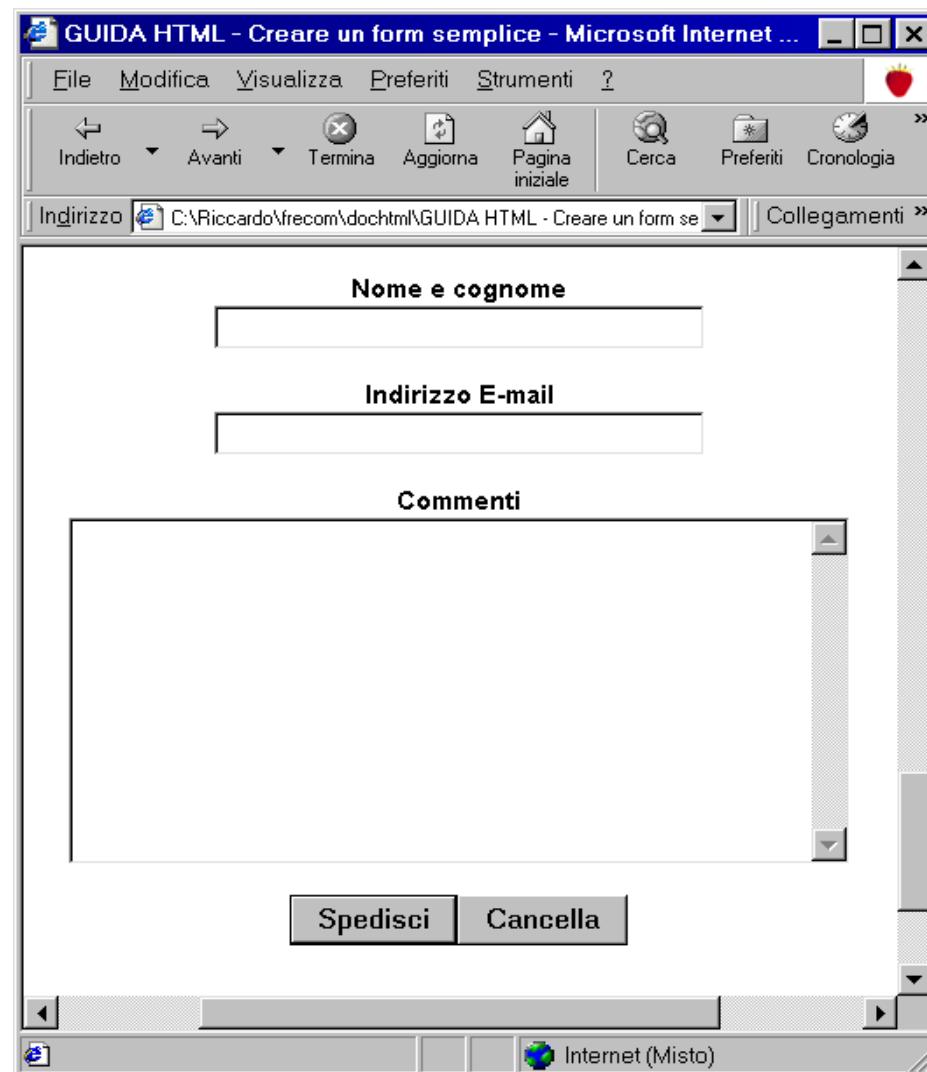
Codice HTML della form (continua)

```
<tr>
<td align="right">foto:
<td><input type="file" name="foto">
<tr>
<td align="right">giudizio:
<td>
<input type="radio" name="giudizio"
       value="sufficiente">sufficiente
<input type="radio" name="giudizio" value="buono">buono
<input type="radio" name="giudizio" value="ottimo">ottimo
<tr>
<td align="right">descrizione lavoro:
<td><textarea name="descr" cols="60" rows="10"></textarea>
<tr>
<td align="center"><input type="submit" value="Invia form">
<td align="center"><input type="reset" value="Reset form">
</table>
</form>
```

Esempio 2

```
<FORM
  ACTION=http://www.coder.com/code/mailform/mailform.pl.
cgi METHOD=POST>
<INPUT TYPE=HIDDEN NAME=MAILFORM_ID VALUE="Val_7743">
<INPUT TYPE=HIDDEN NAME=MAILFORM SUBJECT VALUE="Il mio
  primo FORM">
<INPUT TYPE=HIDDEN NAME=MAILFORM_URL
  VALUE="http://www.html.it/risposta.htm">
<B>Nome e cognome</B><BR>
<input type=text NAME=MAILFORM_NAME size=33><BR><BR>
<B>Indirizzo E-mail</B><BR>
<input type=text NAME=MAILFORM_FROM size=33><BR><BR>
<B>Commenti</B><BR>
<TEXTAREA NAME=MAILFORM_TEXT ROWS=10 COLS=42
  WRAP></TEXTAREA><BR><BR>
<INPUT TYPE=SUBMIT VALUE="Spedisci"><INPUT TYPE=RESET
  VALUE="Cancella">
</FORM>
```

Esempio 2 (cont.)



5. HTML5

HTML5: breve storia

- **HTML5** è il successore di HTML 4 e XHTML 1.0, standardizzati dal W3C nel 1999 e 2000
- Nel 2002 il W3C decide che la futura versione 2.0 di XHTML debba rimpiazzare HTML (questo implica la non-retrocompatibilità del nuovo linguaggio con HTML 4), ed essere *document-oriented* e non *application-oriented*
- Nel 2004 si forma il consorzio WHATWG (Web Hypertext Application Technology Working Group) in opposizione al W3C e alla sua decisione di abbandonare HTML per XHTML
- Slogan di WHATWG: “Don’t break the Web”

HTML5: breve storia (segue)

- WHATWG rilascia nel 2008 il primo draft del suo HTML5
- Nel frattempo il W3C torna sui suoi passi, e finisce per sposare la visione WHATWG. Il progetto XHTML 2.0 viene chiuso, e parte il lavoro per la standardizzazione di HTML5
- Nel 2012 WHATWG rilascia l’“HTML5 living standard”, ovvero una specifica di HTML5 che verrà lasciata evolvere continuamente (senza rilascio di versioni specifiche)

HTML5: breve storia (segue)

- Nel 2014 il W3C rilascia il suo standard HTML5
- Nel 2016 il W3C rilascia HTML 5.1
- Nel 2017 il W3C rilascia HTML 5.2
- Nel 2019 W3C e WHATWG concordano che in futuro l'unico riferimento comune per i due consorzi sarà il “living standard” di WHATWG

HTML5 vs. HTML 4

- Nuove marcature strutturali e «semantiche»
- Altri nuovi tag
- Elementi HTML editabili
- Supporto ai microdati
- Nuovi tag e attributi per le form
- Nuove API

HTML5 vs. HTML4 (segue)

Nuove API create per supportare:

- Multimedialità
- Geolocalizzazione
- Esecuzione asincrona e parallela di script
- Comunicazioni bidirezionali tra web client e web server
- Drag and drop
- Applicazioni web offline
- Grafica
- Cronologia della navigazione
- ...

Tag strutturali e semantici

- <header>
- <footer>
- <section>
- <article>
- <nav>
- <aside>
- <hgroup>
- <mark>
- <time>
- <meter> e <progress>
- <picture>

Altri tag

- <figure>
- <figcaption>
- <ruby>
- <wbr>
- <command>
- <menu>
- <details>
- <summary>
- <keygen>
- <output>

DOCTYPE

In HTML5 il DOCTYPE non fa più riferimento a una DTD, e viene semplificato nel modo seguente:

```
<!DOCTYPE html>
```

Modificare il contenuto di una pagina

I seguenti nuovi attributi globali permettono di dichiarare elementi del documento HTML modificabili da utente:

- contenteditable
- contextmenu
- data-* (attributi definibili da utente)
- draggable
- hidden
- spellcheck

Nuove API

- HTML5 aumenta significativamente le API messe a disposizione degli script
- Gli obiettivi sono:
 - In generale, accrescere le possibilità offerte alla programmazione lato client
 - Standardizzare alcuni tipi più comuni di operazioni effettuate lato client
 - Aggiornare le API alle necessità dei media agent più recenti (smartphone, tablet)

Nuove API

- Gestione di risorse e flussi audio e video
- Accesso off-line alle applicazioni
- Estensione delle capacità di comunicazione, sia verso il web server che verso altre applicazioni
- Esecuzione di azioni in background
- Estensione del concetto di cookie (salvataggio di informazioni sul dispositivo dell'utente)
- Gestione della cronologia della navigazione

Nuove API (segue)

- Text editing
- Gestione del «drag and drop»
- Generazione di oggetti grafici 2D/3D
- Gestione di informazioni multimediali generate dall'utente (ad esempio mediante webcam e microfono)

Offline API

- permettono di salvare copie locali di un insieme di risorse, allo scopo di permettere al browser di eseguire applicazioni anche in modalità offline
- l'elenco delle risorse da salvare è contenuto in un file chiamato **manifest** (MIME type ‘text/cache-manifest’)
- L'attributo manifest del tag html permette di dichiarare il file manifest associato al documento HTML
- La cache costituita dalle risorse elencate nel file manifest è gestita da apposite API (associate all'oggetto corrispondente alla proprietà applicationCache dell'oggetto window)

WebStorage API

- Estendono le capacità di memorizzazione dei cookies
- Oggetti **localStorage** e **sessionStorage** (accessibili come array associativi)
- Possono memorizzare solo stringhe (testo): per memorizzare oggetti arbitrari occorre serializzarli (ad esempio tramite JSON)

WebSocket API

- Permettono di instaurare una connessione dati bidirezionale tra web client e web server
- La creazione di un nuovo oggetto WebSocket crea una connessione con un server (la cui url va specificata nel costruttore)
- La funzione associata all'event handler onmessage viene eseguita quando dal server arriva un messaggio
- Il metodo send(x) invia il testo x al server

Canvas

- elemento <canvas> per dichiarare una zona della pagina su cui è possibile disegnare, tramite nuove API
- si può disegnare una canvas in un contesto 2D o in un contesto 3D
- le API per disegnare (in contesto 2D) si dividono in
 - metodi path (linee, archi,...)
 - metodi modificatori (rotazioni, traslazioni,...)
 - metodo drawImage (disegna un'immagine)
 - metodi per scrivere testo
 - metodi per scrivere singoli pixel

Geolocation API

Servono a gestire dati geospaziali, tipicamente la posizione (anche se questa è effettivamente disponibile solo su alcuni user agent)

La proprietà geolocation dell'oggetto navigator ha due metodi per ottenere la posizione corrente:

- **getCurrentPosition**
- **watchPosition**

(il secondo metodo differisce dal primo perché restituisce una nuova posizione ogni volta che questa cambia)

Audio/Video e nuove API

- HTML5 permette la gestione nativa (ovvero senza ricorrere a plug-in del browser) di contenuti multimediali
- tag **<video>**: permette di inserire un contenuto video
- tag **<audio>**: permette di inserire un contenuto audio
- il formato dei video e degli audio supportati dipende dai browser, tuttavia esistono dei formati di riferimento (mp4, webm, ogg per i video)
- esistono delle nuove API per video e audio che permettono la gestione di questo tipo di risorse da script

Form

- Nuovi attributi ed input type per le form sono stati introdotti in HTML5
- L'obiettivo principale è quello di permettere di definire le più comuni forme di validazione di form lato client direttamente nel documento HTML (cioè senza bisogno di aggiungere codice JavaScript)
- Sono stati inoltre aggiunti tipi di elementi utili soprattutto nei nuovi media agent (smartphone e tablet)

Autofocus, placeholder, form

Nuovi attributi:

- **autofocus** (booleano): all'apertura della form, il focus va sull'elemento che ha dichiarato autofocus
- **placeholder** (per elementi input o textarea): valore che all'apertura della form compare sul campo editabile (N.B.: non corrisponde ad un valore (value) iniziale del campo, viene solo visualizzato all'inizio)
- **form**: attributo che permette di associare un elemento ad una form. E' così possibile, ad esempio, dichiarare un campo che appartiene a più form, o dichiarare un campo all'esterno di un elemento form

Required, autocomplete

- **required** (booleano): rende obbligatoria la compilazione dell'elemento al momento del submit della form a cui l'elemento appartiene
- **autocomplete**: attributo che può assumere due valori:
 - on = il browser può effettuare l'**autocomplete** di questo campo, cioè completare il campo in maniera automatica usando i valori precedentemente inseriti in questo campo
 - off = il browser non può fare l'autocompletion
 - se autocomplete non viene assegnato, viene usato il default del browser (di solito è on)

Multiple, pattern

- **multiple** (booleano): permette al campo di accettare valori multipli
- esempio:

```
<form action="demo_form.asp">  
    Select images: <input type="file" name="img"  
    multiple>  
    <input type="submit">  
</form>
```

- **pattern**: viene assegnato ad una espressione regolare; i valori del campo devono appartenere al linguaggio denotato dall'espressione regolare

Min, max, step

- **min** = valore minimo ammesso
- **max** = valore massimo ammesso
- **step** = intervallo tra un valore ammesso e il successivo
- **novalidate** (booleano): se dichiarato sull'elemento form, indica che **non** verrà effettuata la validazione degli elementi di quella form

Nuovi input type

I seguenti input types permettono di gestire informazioni testuali di tipo specifico:

- tel
- search
- url
- email

Nuovi input type (segue)

- color: permette la selezione di un colore
- number: permette l'inserimento di un numero
- range: permette l'inserimento di un numero attraverso uno slider (cursore orizzontale)

Nuovi input type (segue)

Per gestire le date sono stati introdotti i seguenti input types:

- `datetime`
- `datetime-local`
- `date`
- `month`
- `week`
- `time`

Il tag <datalist>

- Permette di definire un campo testuale sul quale il browser può effettuare **autocomplete** usando un insieme predefinito di valori (l'utente però è libero di scrivere un valore al di fuori dell'elenco predefinito)
- esempio:

```
<input type="text" name="giudiz" list="listagiudizi">
  <datalist id="listagiudizi">
    <option value="insufficiente">
    <option value="sufficiente">
    <option value="discreto">
    <option value="buono">
    <option value="ottimo">
  </datalist>
</input>
```

Supporto ai microdati

- i microdati servono a creare un tagging «semantico» di porzioni del documento
- sono basati vocabolari che definiscono identificatori di proprietà relative ad un (micro-)dominio di interesse
- si utilizzano gli attributi HTML `itemscope`, `itemtype` e `itemprop`

Esempio: il vocabolario Person

Property	Description
name (fn)	Name.
nickname	Nickname.
photo	An image link.
title	The person's title (for example, Financial Manager).
role	The person's role (for example, Accountant).
url	Link to a web page, such as the person's home page.
affiliation (org)	The name of an organization with which the person is associated (for example, an employer). If fn and org have the exact same value, Google will interpret the information as referring to a business or organization, not a person.
friend	Identifies a social relationship between the person described and another person.
contact	Identifies a social relationship between the person described and another person.
acquaintance	Identifies a social relationship between the person described and another person.
address (adr)	The location of the person. Can have the subproperties street-address, locality, region, postal-code, and country-name.

Microdati in HTML5: esempio

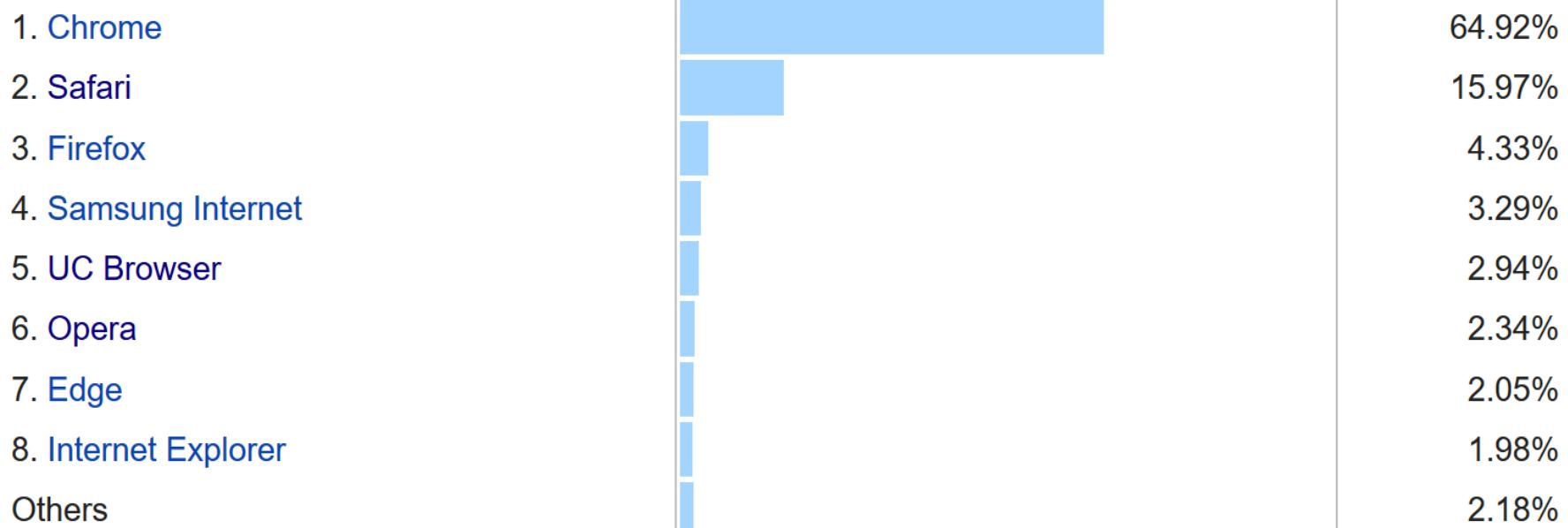
...

```
<div itemscope itemtype="http://datavocabulary.org/Person">  
Benvenuti nella home page di  
<span itemprop="name">Riccardo Rosati</span>,  
<span itemprop="title">professore associato</span> alla  
<span itemprop="affiliation">Sapienza Università di  
Roma</span>.  
</div>
```

...

I web browser più diffusi

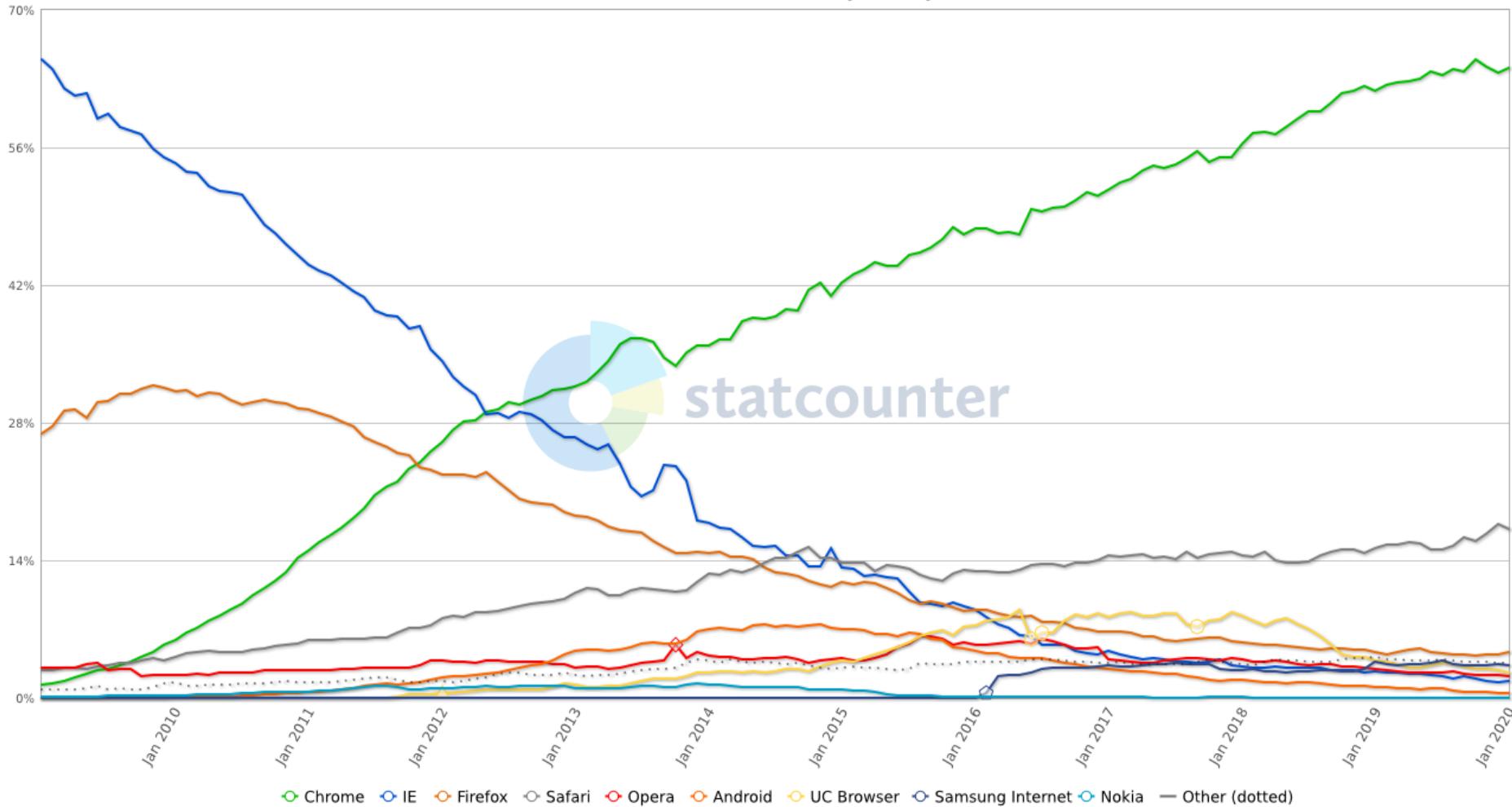
Usage share of all browsers



Web browser market share according to [StatCounter](#) for October 2019.^[25]

Diffusione dei web browser, 2009-2020

StatCounter Global Stats
Browser Market Share Worldwide from Jan 2009 - Jan 2020



Riferimenti

- Raccomandazioni ufficiali W3C su HTML:
 - www.w3.org
 - <http://www.w3.org/TR/html15/>
 - <https://www.w3.org/TR/html151/>
 - <https://www.w3.org/TR/html152/>
- HTML living standard (WHATWG):
 - <https://html.spec.whatwg.org/multipage/>
- Guide (libri) su HTML/HTML5:
 - es.: Jennifer Niederst Robbins: Learning Web Design: A Beginner's Guide to Html, Css, Javascript, and Web Graphics. 5th edition. O'Reilly editore, 2018. ISBN 978-1491960202 (in inglese)

Riferimenti

- Guida online per HTML e altre tecnologie Web:
 - www.w3schools.com
- Sito italiano per HTML e altre tecnologie Web:
 - www.html.it
- Microdati, data vocabulary:
 - <http://www.schema.org>
- Statistiche sull'utilizzo dei web browser:
 - <http://gsa.statcounter.com/global-stats/browser-market-share>

Sapienza Università di Roma
corso di laurea in Ingegneria informatica e automatica

Linguaggi e tecnologie per il Web

a.a. 2020/2021

Parte 2

JavaScript

Luigi Dragone, Riccardo Rosati

Introduzione

- **JavaScript** è un linguaggio di scripting **lato client** utilizzato per rendere dinamico il codice HTML.
- Il documento HTML è generato staticamente.
- Il codice JavaScript è immerso nel documento HTML ma viene **eseguito dinamicamente** solo al momento della richiesta del web client (browser).
- Principali usi:
 - posizionamento dinamico degli oggetti;
 - validazione campi dei moduli;
 - effetti grafici;
 - ...

Introduzione

- JavaScript è un linguaggio di programmazione di tipo script:
 - Non è compilato
 - È ad alto livello
- La sua caratteristica principale è di essere nato espressamente per il WWW (è stato introdotto da Netscape nel 1995) e di essere supportato dalla maggior parte dei browser in uso.

Introduzione

- Non può essere adoperato per costruire programmi complessi o con particolari requisiti di prestazioni, tuttavia può essere impiegato per implementare velocemente piccole procedure e controlli.
- Il suo campo d'impiego naturale è la gestione sul web client (browser) di alcuni elementi delle applicazioni per WWW.

PRIMA PARTE

Il linguaggio JavaScript

Sintassi

- Un programma JavaScript è composto da una sequenza di istruzioni terminate dal punto e virgola (;)
- Un insieme di istruzioni delimitate da parentesi graffe ({ e }) costituiscono un blocco.

Sintassi

- Un **identificatore** è una sequenza di simboli che identifica un elemento all'interno del programma
- Un identificatore può essere composto da lettere e numeri, ma:
 - non deve contenere elementi di punteggiatura o spazi, e
 - deve cominciare con una lettera.

Sintassi

- Questi sono identificatori validi
 - nome
 - Codice_Fiscale
 - a0
- Questi sono identificatori non validi
 - 0a
 - x.y
 - Partita IVA

Sintassi

- Il linguaggio è **case-sensitive**, ovvero distingue le lettere maiuscole dalle minuscole
- I commenti sono delimitati da `/*` e `*/`
- Il simbolo `//` indica che il testo seguente sulla medesima riga è un commento

Variabili

- Una variabile è un'area di memoria contenente informazioni che possono “variare” nel corso dell'esecuzione del programma.
- Una variabile è caratterizzata da un nome identificativo e dal tipo dell'informazione contenuta.

Definizione di variabili

- Prima di essere adoperata una variabile deve essere definita
- La definizione stabilisce il nome della variabile, mentre il tipo dipende dall'assegnazione

```
var eta;
```

```
eta = 35; //intero
```

```
nome = "Mario Rossi"; //stringa
```

Definizione di variabili

- Il tipo di una variabile dipende dall’ultima assegnazione, quindi una variabile può cambiare tipo nel corso del suo ciclo di vita

```
x = 10; //intero
```

...

```
x = "a"; //stringa
```

- JavaScript è un linguaggio “debolmente tipato”

Tipi dato predefiniti

- Number
- Boolean
- Null
- String
- Date
- Array

Tipo Number

- Una variabile di tipo Number assume valori numerici interi o decimali

```
var x = 10;      //valore intero
```

```
var y = -5.3;   //valore decimale
```

- Sono definite le operazioni aritmetiche fondamentali ed una serie di funzioni matematiche di base

Tipo Boolean

- Una variabile di tipo Boolean assume i soli valori della logica booleana vero e falso

```
var pagato = true; //valore logico vero
```

```
var consegnato = false; //valore logico falso
```

- Sono definite le operazioni logiche fondamentali (AND, OR e NOT)

Tipo Boolean

AND:

X	Y	X AND Y
true	true	true
true	false	false
false	true	false
false	false	false

Tipo Boolean

OR:

X	Y	X OR Y
true	true	true
true	false	true
false	true	true
false	false	false

Tipo Boolean

NOT:

X	NOT X
true	false
false	true

Tipo Null

- Si tratta di un tipo che può assumere un unico valore

```
var a = null;
```

- Serve ad indicare che il contenuto della variabile è non significativo

```
var sesso = "f";
```

```
var militesente = null;
```

Tipo String

- Una variabile di tipo String contiene una sequenza arbitraria di caratteri
- Un valore di tipo Stringa è delimitato da apici ('') o doppi-apici ("")

```
var nome = "Mario Rossi";  
var empty = ""; //Stringa vuota  
var empty2 = new String(); //Stringa vuota  
var str = 'Anche questa è una stringa';  
var str2 = new String("Un'altra stringa");
```

Tipo Date

- Una variabile di tipo Date rappresenta un istante temporale (data ed ora)

```
var adesso = new Date();  
var natale2012 = new Date(2012,11,25);  
var capodanno2013 = new Date("Gen 1  
2013");
```

- E' definito l'operatore di sottrazione (-) tra due date che restituisce la differenza con segno espresso in millisecondi

Tipo Array

- Un array è un vettore monodimensionale di elementi di tipo arbitrario

```
var v = new Array(); //Vettore vuoto  
var w = new Array("Qui", "Quo", "Qua");  
var u = new Array("Lun", "Mar", "Mer",  
    "Gio", "Ven", "Sab", "Dom");
```

- Non è necessario specificare la dimensione

Assegnazione

- L'assegnazione è l'operazione fondamentale nei linguaggi di programmazione imperativa
id = expr
- Dapprima viene “valutata” l'espressione **expr**, quindi il risultato viene “assegnato” alla variabile **id**

Assegnazione

- Si consideri il seguente frammento di codice

```
var x = 10;
```

```
var y = 7;
```

```
var z = 3 * (x - y);
```

- Al termine dell'esecuzione la variabile z assume il valore 9

Espressioni

- Un'espressione è composta da
 - identificatori di variabili
x, nome, eta, importo, ...
 - costanti
10, 3.14, "<HTML>", ...
 - operatori
 - parentesi ((e)) per alterare le regole di precedenza tra operatori

Operatori aritmetici

- Operano tra valori (costanti o variabili) numerici
 - Somma ($1+1$)
 - Sottrazione ($3-5$)
 - Moltiplicazione (3×4)
 - Divisione ($6 / 4$)
 - Modulo ($6 \% 5$)
 - Cambiamento di segno (-3)

Operatori di pre/post incremento/decremento

- Derivano dal linguaggio C (e sono presenti in C++ e Java)
- Alterano e restituiscono il valore di una variabile
- Consentono di scrivere codice compatto ed efficiente

Operatori di pre/post incremento/decremento

```
var x = 10;
```

```
var y = x++; // y=10 e x=11
```

```
var z = ++x; // z=12 e x=12
```

```
var w = x--; // w=12 e x=11
```

```
var v = --x; // v=10 e x=10
```

Operatori su stringhe

- L'unica operazione possibile in un'espressione è la concatenazione di stringhe

```
nomeCompleto = titolo + " " + nome + " "  
+ cognome
```

```
indirizzo = recapito + ", " + numCivico +  
" " + CAP + " - " + citta + " (" + prov  
+ ")"
```

Operatori su stringhe

- Esiste una forma più compatta per esprimere l'accodamento
- L'istruzione seguente

```
str = str + newStr;
```

è equivalente a

```
str += newStr;
```

Operatori su vettori

- L'operatore definito sui vettori è l'accesso ad un determinato elemento dato l'indice

```
v[3] = 10;  
a[i] = b[i] * c[i];  
p = v[1];
```

- Il primo elemento di un vettore ha sempre l'indice 0 (come in C/C++ e Java)

Operatori sui vettori

- Se si assegna un valore ad un elemento non presente questo viene creato

```
var v = new Array();  
v[0] = 1;
```

- Se si accede al valore di un elemento non presente si ottiene un valore indefinito

```
var v = new Array();  
var a = v[0];
```

Operatori relazionali

- Confrontano due valori e restituiscono l'esito come valore booleano
 - Uguaglianza (`==`)
 - Disuguaglianza (`!=`)
 - Minore (`<`)
 - Maggiore (`>`)
 - Minore o uguale (`<=`)
 - Maggiore o uguale (`>=`)

Operatori logici

- Operano tra valori (costanti o variabili) booleani
 - AND (`&&`)
 - OR (`||`)
 - NOT (`!`)
- Solitamente gli operandi sono l'esito di un confronto

Condizioni

- L'utilizzo di operatori relazionali e logici consente di formulare delle condizioni che possono essere utilizzate per controllare l'esecuzione del programma

(metodoPagamento=="contrassegno") &&
(!residenteInItalia)

Controllo dell'esecuzione

- L'esecuzione di un programma è generalmente sequenziale
- Tuttavia in determinate “condizioni” può essere necessario eseguire solo alcune istruzioni, ma non altre, oppure ripetere più volte un'operazione

Istruzione if

- L'istruzione **instr** viene eseguita solo se la condizione **cond** risulta vera

```
if (cond)
    instr
```

- L'istruzione **instr** può essere sostituita da un gruppo di istruzioni tra parentesi graffe ({ e })

Istruzione if

- Una costruzione alternativa prevede la presenza di una seconda istruzione da eseguire nel caso la condizione risulti falsa

```
if (cond)
  instr_then
else
  instr_else
```

Istruzione if

```
if(scelta=="NO")  {  
    // Se la scelta è NO  
    ...  
} else {  
    // Altrimenti  
    ...  
}
```

Istruzione for

- Viene dapprima eseguita l'istruzione **init**, quindi l'istruzione **instr** viene ripetuta finché la condizione **cond** risulta vera, dopo ogni ripetizione viene eseguita l'istruzione **next**

```
for(init; cond; next)  
    instr
```

Istruzione for

- Inizializzare a 0 gli n elementi del vettore a

```
for(var i=0; i<n; i++)  
    a[i]=0;
```

- Copiare gli n elementi del vettore a nel vettore b

```
for(var i=0; i<n; i++)  
    b[i]=a[i];
```

Istruzione while

- L'istruzione **instr** viene eseguita finché la condizione **cond** risulta essere verificata

```
while (cond)
```

```
    instr
```

- Un'istruzione for può essere espressa come

```
init;
```

```
while (cond) { instr; next; }
```

Funzioni

- Una funzione è un elemento di un programma che calcola un valore che dipende “funzionalmente” da altri

$$y \leftarrow f(x)$$

$$y \leftarrow \log_{10}(x)$$

- L'utilizzo delle funzioni nella programmazione strutturata aumenta la modularità e favorisce il riutilizzo

Definizione di funzioni

- In JavaScript è possibile definire una o più funzioni all'interno di un programma

```
function name (arg0, arg1, ..., argn-1) {  
    ...  
}
```

- La funzione definita è identificata da **name** e dipende dagli argomenti **arg**₀, **arg**₁, ..., **arg**_{n-1}

Definizione di funzioni

- Somma di due numeri

```
function somma(a, b) {  
    return a+b;  
}
```

- La funzione viene “invocata” all’interno di un’espressione

```
var s = somma(1, 2);
```

Invocazione di funzioni

- Quando una funzione viene invocata gli argomenti sono inizializzati con i valori specificati
- Quindi si procede con l'esecuzione delle istruzioni costituenti la funzione
- L'istruzione `return` restituisce il valore calcolato al chiamante

Invocazione di funzioni

- La chiamata

`x = somma(1, 2)`

inizializza gli argomenti `a` e `b` rispettivamente ai valori 1 e 2

- L'istruzione

`return a+b;`

valuta l'espressione e restituisce il risultato (3) che viene assegnato alla variabile `x`

Variabili locali e globali

- All'interno di una funzione è possibile definire delle variabili “confinite” all'interno della funzione stessa
- Tali variabili, dette **locali**, sono create all'atto dell'invocazione della funzione e sono distrutte al termine dell'esecuzione
- Il loro valore non è accessibile dall'esterno della funzione
- Ogni argomento di una funzione è una variabile locale definita implicitamente

Variabili locali e globali

- Le variabili definite all'esterno di una funzione sono denominate, invece, **globali**
- A differenza delle variabili locali, le variabili globali sono accessibili da qualsiasi punto del programma, anche dall'interno di una funzione, sempre che in quest'ultima non sia stata definita una variabile locale con lo stesso nome

Variabili locali e globali

- Si consideri il seguente frammento di codice

```
function f (...) {  
    ...  
    var x = 1;  
    ...  
}  
var x = -1;  
...f (...);
```

- La variabile globale x continua a valere -1

Procedure

- Una funzione che non restituisce valori viene detta **procedura** (in analogia con il Pascal)
- Una procedura agisce in genere su variabili globali (*side-effect*)

Funzioni predefinite

- In JavaScript sono presenti alcune **funzioni predefinite**
 - isNaN (v) verifica se v non è un numero
 - isFinite (v) verifica se v è finito
 - parseFloat (str) converte str in un numero decimale
 - parseInt (str) converte str in un numero intero

Oggetti

- Un oggetto è un elemento caratterizzato da uno stato rappresentato mediante proprietà e da un insieme di azioni (o metodi) che può eseguire
- Oggetti caratterizzati dagli stessi metodi e dalle stesse proprietà, ma non dallo stesso stato, sono detti della stessa classe

Oggetti

- JavaScript è un linguaggio orientato agli oggetti, tuttavia non possiede il costrutto di classe
- Molti tipi di dato fondamentali sono, in effetti, degli oggetti (String, Date, Array,...)

Proprietà e metodi

- Una proprietà di un oggetto è assimilabile ad una variabile

```
cliente.nome = "Carlo Bianchi";
```

```
x = ordine.aliquotaIVA;
```

- Un metodo, invece, è simile ad una funzione

```
tot = ordine.calcolaTotale();
```

Proprietà e metodi

- Esistono due sintassi alternative per accedere alle proprietà degli oggetti
 - oggetto.proprieta
 - oggetto[“proprieta”]
- La seconda è utile quando il nome della proprietà viene determinato durante l'esecuzione del programma

Oggetti di tipo String

- Proprietà
 - `length` lunghezza della stringa
- Metodi
 - `charAt(pos)` carattere alla posizione `pos`
 - `subString(start, end)` sottostringa dalla posizione `start` alla posizione `end`
 - `toUpperCase()`/`toLowerCase()` converte la stringa in maiuscolo/minuscolo
 - `indexOf(str, pos)` posizione della prima occorrenza della stringa `str` cercata a partire dalla posizione `pos`

Oggetti di tipo Array

- Proprietà
 - `length` lunghezza del vettore
- Metodi
 - `sort()` ordina gli elementi del vettore
 - `reverse()` inverte l'ordine degli elementi del vettore

Oggetti di tipo Date

- Metodi
 - `getXXX()` restituisce il valore della caratteristica `XXX` della data (es. `getFullYear()`).
 - `setXXX(val)` imposta il valore della caratteristica `XXX` della data (es.
`setFullYear(2013,1,1);`
 - `toString()` restituisce la data come stringa formattata

Oggetti di tipo Date

Nome caratteristica	Significato
Date	Giorno del mese
Day	Giorno della settimana
FullYear	Anno
Minutes	Minuti
Hours	Ore
Month	Mese
Seconds	Secondi
Time	Tempo (hh:mm:ss)

Oggetto Math

- Proprietà
 - E costante di Eulero
 - PI pi greco
- Metodi
 - abs (val) valore assoluto
 - ceil (val)/floor (val) troncamento
 - exp (val) esponenziale
 - log (val) logaritmo
 - pow (base, exp) elevamento a potenza
 - sqrt (val) radice quadrata

SECONDA PARTE

Uso di JavaScript per il Wide Web

Integrazione con i browser web

- La caratteristica principale di JavaScript è di essere “integrabile” all’interno delle pagine web
- In particolare consente di aggiungere una logica procedurale alle pagine rendendole “dinamiche”
- A differenza di altre tecnologie, JavaScript funziona completamente sul client

Integrazione con i browser web

- I campi di impiego tradizionali sono
 - Validazione dell'input dell'utente e controllo dell'interazione
 - Effetti visivi di presentazione
- Con l'avvento di HTML5 i potenziali usi di JavaScript sono notevolmente aumentati
- JavaScript è supportato da tutti i browser più diffusi

JavaScript e HTML

- L'integrazione degli script all'interno di una pagina HTML avviene in due modi
 - Associando l'esecuzione di funzioni JavaScript agli **eventi** collegati alla pagina che si intende gestire
 - Accedendo dalle funzioni JavaScript alle proprietà degli oggetti che costituiscono la pagina

Dynamic HTML (DHTML)

- Una pagina “dinamica” (DHTML) è una pagina HTML contenente codice JavaScript associato a determinati eventi che implementa specifiche funzionalità
- Non bisogna confondere una pagina DHTML con una pagina generata dinamicamente dal server

Modello ad oggetti

- Un browser web esporta verso JavaScript un modello ad oggetti della pagina (DOM) e dell’“ambiente” (BOM) in cui la pagina è visualizzata
- Una funzione JavaScript adopera tali oggetti invocando i metodi e accedendo alle proprietà

DOM e BOM

- DOM = Document Object Model
 - Corrisponde in pratica all'oggetto document che vedremo nel seguito
- Il DOM HTML è uno standard (è parte dello standard HTML)
- BOM = Browser Object Model
 - Corrisponde in pratica all'oggetto window che vedremo in seguito
- Il BOM non è uno standard (ma i browser si basano essenzialmente sullo stesso BOM)

DOM: Oggetto document

- Rappresenta il documento HTML che costituisce la pagina visualizzata
- Non è possibile accedere a tutti gli elementi del documento
- Tuttavia è possibile accedere agli elementi dei moduli (form) ed alle proprietà di visualizzazione

Oggetto document

- Inoltre, è possibile costruire on-the-fly il documento prima che questo sia stato completamente caricato e visualizzato

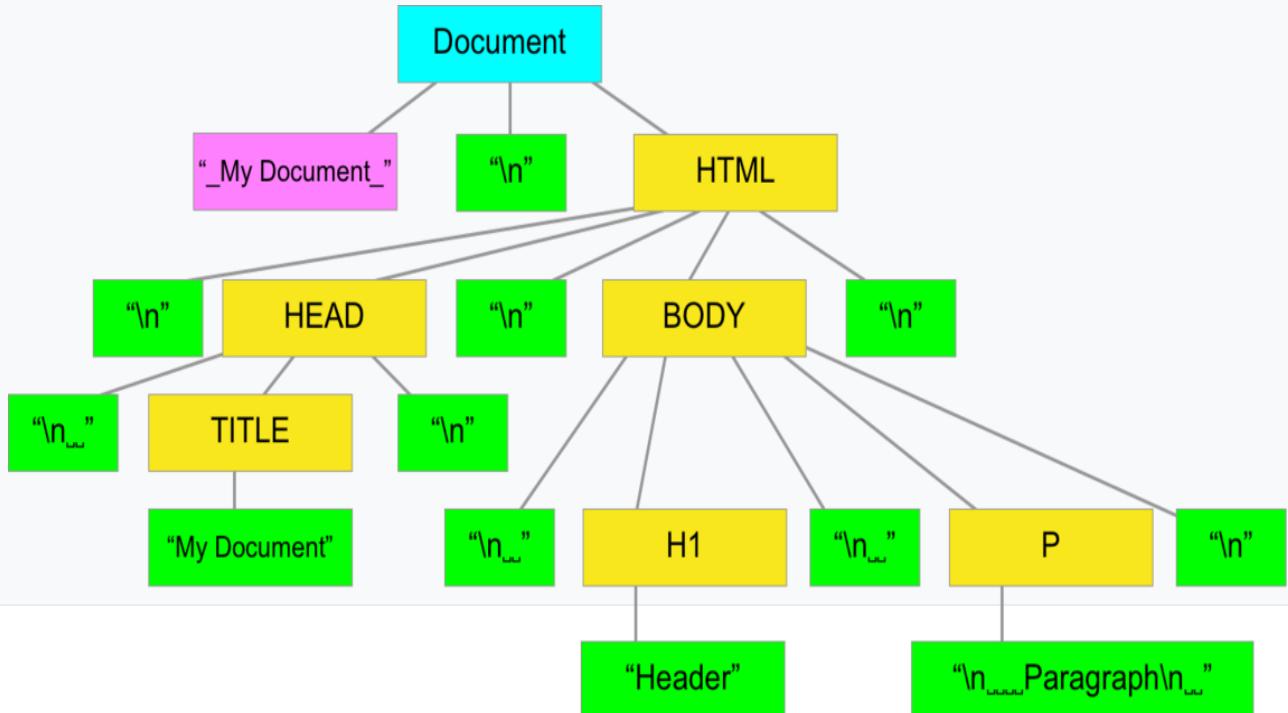
Oggetto document

- Proprietà
 - bgColor colore dello sfondo
 - fgColor colore del testo
 - forms vettore dei moduli presenti nella pagina
 - title titolo del documento
 - URL indirizzo del documento
- Metodi
 - write (string) accoda string al documento, serve per la costruzione on-the-fly

Documento HTML = albero

- Nel DOM il documento HTML è rappresentato come un albero:

```
<!-- My document -->
<HTML>
<HEAD>
  <TITLE>My Document</TITLE>
</HEAD>
<BODY>
  <H1>Header</H1>
  <P>
    Paragraph
  </P>
</BODY>
</HTML>
```



Oggetto document

Principali metodi per selezionare elementi:

- `document.getElementById("titolo");`
Restituisce l'elemento (se esiste) il cui attributo ID vale "titolo"
- `document.getElementsByClassName("c1");`
Restituisce (in un oggetto NodeList) tutti gli elementi il cui attributo CLASS vale "c1"
- `document.getElementsByName("pluto");`
Restituisce (in un oggetto NodeList) tutti gli elementi il cui attributo NAME vale "pluto"
- `document.getElementsByTagName("div");`
Restituisce (in un oggetto NodeList) tutti gli elementi div

Oggetto document

- Supponendo che nel documento HTML sia definito un modulo di nome *modulo*

```
<FORM NAME="modulo"...>
```

...

```
</FORM>
```

Oggetto document

- Si può accedere a tale oggetto in due diversi modi

```
document.forms[“modulo”];
```

```
document.modulo;
```

- Ciò è possibile, in generale, per tutti gli elementi del documento con un attributo NAME

Oggetto document

- Dal momento che la proprietà forms è di tipo Array è possibile accedervi anche tramite l'indice numerico dell'elemento

```
for(var i=0; i<document.forms.length; i++)  
{  
    //Accedi a document.forms[i]  
    ... = document.forms[i];  
    ...  
}
```

Oggetto Form

- Un oggetto di questo tipo corrisponde ad un modulo all'interno di una pagina HTML
- Tramite le proprietà di questo oggetto è possibile accedere ai diversi elementi (o controlli) del modulo (inputbox, listbox, checkbox, ecc.)

Oggetto Form

- Proprietà
 - action valore dell’attributo ACTION
 - elements vettore contenente gli elementi del modulo
 - length numero di elementi del modulo
 - method valore dell’attributo METHOD
 - target valore dell’attributo TARGET

Oggetto Form

- Metodi
 - `reset()` azzerà il modulo reimpostando i valori di default per i vari elementi
 - `submit()` invia il modulo

Oggetto Form

- Supponendo che l’i-esimo elemento di un modulo mod sia denominato nome_i è possibile farvi riferimento in 3 modi diversi

```
document.mod.elements[i-1];
```

```
document.mod.elements["nome_i"];
```

```
document.mod.name_i;
```

- Attenzione l’indice del primo elemento di un vettore è sempre 0 (quindi l’i-esimo elemento ha indice i-1)

Elementi dei moduli

- All'interno di un modulo possono comparire diversi tipi di elementi, corrispondenti ai vari costrutti HTML
- Ogni tipo ha proprietà e metodi specifici, per una trattazione approfondita si rimanda alla guida di riferimento del modello ad oggetti implementato dal browser

Elementi dei moduli

HTML	JavaScript
<INPUT TYPE="text">	Text
<TEXTAREA></TEXTAREA>	Textarea
<SELECT></SELECT>	Select
<INPUT TYPE="checkbox">	Checkbox
<INPUT TYPE="radio">	Radio
<INPUT TYPE="button">	Button

Elementi dei moduli

- Tutti i tipi di elementi possiedono le seguenti proprietà
 - name nome dell'elemento
 - value valore corrente dell'elemento
- Gli elementi di tipo Input possiedono la proprietà defaultValue che contiene il valore predefinito del campo (attributo VALUE del tag HTML)

Elementi dei moduli

- Gli elementi di tipo Radio e Checkbox possiedono la proprietà checked che indica se l'elemento è stato selezionato
- Gli elementi di tipo Select possiedono la proprietà selectedIndex, che contiene l'indice dell'elemento selezionato nella lista, e la proprietà options, che contiene il vettore delle scelte dell'elenco

Elementi dei moduli

- E' possibile modificare i valori contenuti negli elementi dei moduli
- Pertanto è possibile utilizzare questi elementi anche per fornire risultati all'utente
- Se un elemento ha scopi esclusivamente di rappresentazione può essere marcato come READONLY

Esempio: modulo di iscrizione

- Il modulo deve raccogliere i dati su un utente che vuole sottoscrive un certo servizio
- Per ogni utente deve richiedere
 - nominativo, età, sesso
 - se desidera ricevere informazioni commerciali
 - il servizio cui desidera iscriversi

Esempio: modulo di iscrizione

- Il modulo deve suggerire un'età di 18 anni ed il consenso all'invio di informazioni commerciali
- I servizi disponibili sono denominati “Servizio 1”, “Servizio 2” e “Servizio 3”
- Il modulo deve suggerire la sottoscrizione al primo servizio

Esempio: modulo di iscrizione

- Per ogni dato si determina il tipo di elemento da inserire nel modulo
 - nominativo ed età con elementi di tipo Text
 - sesso con un elementi di tipo Radio
 - infoComm con un elemento di tipo Checkbox
 - servizio con un elemento di tipo Select
- Si deve, inoltre, inserire il pulsante di invio del modulo

Esempio: modulo di iscrizione

- Per gestire la presentazione si adopera una tabella HTML che presenta sulla prima colonna il nome del campo e nella seconda gli elementi corrispondenti

Esempio: modulo di iscrizione

```
<HTML>
<BODY>
<FORM NAME="iscrizione" ACTION="" METHOD="POST">
<TABLE>
...
...
```

Esempio: modulo di iscrizione

```
...  
<!-- Nominativo -->  
<TR>  
  <TD>Nominativo</TD>  
  <TD>  
    <INPUT NAME="nominativo" TYPE="text" SIZE="40">  
  </TD>  
</TR>  
...
```

Esempio: modulo di iscrizione

```
...
<!-- Eta' -->
<TR>
  <TD>Et&agrave;;</TD>
  <TD>
    <INPUT NAME="eta" TYPE="text" SIZE="3" VALUE="18">
  </TD>
</TR>
...

```

Esempio: modulo di iscrizione

```
<!-- Sesso -->
<TR>
    <TD>Sesso</TD>
    <TD>
        <INPUT NAME="sesso" TYPE="radio" VALUE="M">M
        <INPUT NAME="sesso" TYPE="radio" VALUE="F">F
    </TD>
</TR>
```

Esempio: modulo di iscrizione

```
...  
<!-- Inform. commerciali -->  
<TR>  
    <TD>Inform. commerciali</TD>  
    <TD>  
        <INPUT NAME="infoComm" TYPE="checkbox" CHECKED>  
    </TD>  
</TR>  
...
```

Esempio: modulo di iscrizione

```
<!-- Servizio -->
<TR>
    <TD>Servizio</TD>
    <TD>
        <SELECT NAME="servizio">
            <OPTION VALUE="s1" SELECTED>Servizio 1</OPTION>
            <OPTION VALUE="s2">Servizio 2</OPTION>
            <OPTION VALUE="s3">Servizio 3</OPTION>
        </SELECT>
    </TD>
</TR>
```

Esempio: modulo di iscrizione

...

```
<!-- Invio del modulo -->  
<TR>  
  <TD>  
    <INPUT TYPE="submit" VALUE="Invia">  
  </TD>  
</TR>
```

...

Esempio: modulo di iscrizione

...

```
</TABLE>  
</FORM>  
</BODY>  
</HTML>
```

Esempio: modulo di iscrizione

The screenshot shows a Mozilla Firefox browser window with a registration form loaded. The form fields are:

- Nominativo: An input field.
- Età: A numeric input field containing "18".
- Sesso: Radio buttons for "M" and "F", with "M" selected.
- Inform. commerciali: A checked checkbox.
- Servizio: A dropdown menu set to "Servizio 1".
- Invia: A blue "Send" button.

The status bar at the bottom of the browser window displays the text "Completato".

Esempio: modulo di iscrizione

- Per accedere al nominativo immesso
`document.iscrizione.nominativo.value`
- Per accedere all'età
`document.iscrizione.eta.value`
- Per accedere al valore numerico dell'età
`parseInt (document.iscrizione.eta.value)`

Esempio: modulo di iscrizione

- Per visualizzare un messaggio relativo alla scelta di ricevere informazioni commerciali:

```
if(document.iscrizione.infoComm.checked)
    alert("Vuoi ricevere informazioni
          commerciali");
else
    alert("Non vuoi ricevere informazioni
          commerciali");
```

Eventi

- Ogni oggetto di un documento HTML “genera” degli **eventi** in risposta alle azioni dell’utente
- Ad esempio, l’evento `click` corrisponde al click del puntatore sull’oggetto
- Per gestire l’interazione con l’utente si associano funzioni JavaScript a particolari eventi

Eventi

- Gli eventi generati da un oggetto dipendono dal tipo di quest'ultimo
- Oggetti Form
 - onSubmit invio del modulo
 - onReset azzeramento del modulo
- Oggetti Button
 - onClick click del puntatore

Eventi

- Oggetti Select, Text e Textarea
 - onChange modifica del contenuto
 - onFocus selezione dell'elemento
- Oggetti Radio e Checkbox
 - onClick click del puntatore
 - onFocus selezione dell'elemento

Intercettazione eventi

- Per intercettare l'evento E di un tag T ed associare l'esecuzione di una funzione $f()$
`<T onE="return f();">`
- Se il risultato della valutazione della funzione è false viene interrotta l'esecuzione del comando corrente, ad esempio l'invio di un modulo

Intercettazione eventi

- Ad esempio, la funzione `valida()` verifica se i dati immessi nel modulo `modulo` sono corretti ed eventualmente procede con l'invio di quest'ultimo

```
<FORM NAME="modulo"  
onSubmit="return valida(); ...>
```

...

```
</FORM>
```

Validazione modulo di iscrizione

- Nel modulo di sottoscrizione del servizio è necessario indicare il nominativo del sottoscrittore
- Quindi il valore del campo corrispondente non deve essere una stringa vuota

Validazione modulo di iscrizione

- Il modulo viene ridefinito come

```
<FORM NAME="iscrizione" ACTION=""  
METHOD="POST" onSubmit="return  
validaIscrizione();">
```
- Nell'intestazione del documento viene definita
una funzione validaIscrizione()

Validazione modulo di iscrizione

```
function validaIscrizione() {  
    if(document.iscrizione.nominativo.value=="")  
    {  
        alert("Nominativo obbligatorio.  
            Impossibile procedere.");  
        return false;  
    }  
    ... //Altri controlli  
    return true;  
}
```

Proprietà di visualizzazione

- Tra le proprietà degli elementi del modello ad oggetti del documento sono presenti alcune specifiche della modalità di presentazione all’utente degli elementi medesimi
- La possibilità di poter accedere e manipolare tali caratteristiche permette di utilizzare JavaScript per ottenere sofisticati effetti di presentazione visiva
- Purtroppo, queste proprietà sono specifiche dei singoli browser

Proprietà di visualizzazione

- Le proprietà di visualizzazione sono connesse con l'uso dei CSS
- Infatti, alcune proprietà degli stili possono essere modificate dinamicamente da funzioni JavaScript
- Tra le proprietà più utili per la creazione di effetti visivi abbiamo la visualizzazione ed il posizionamento degli elementi

BOM: Oggetto window

- Questo oggetto rappresenta la finestra in cui il documento corrente viene visualizzato
- Una funzione può accedere alle proprietà della finestra corrente, ma può creare e manipolare nuove finestre (pop-up)

Oggetto window

- L'oggetto window è in realtà l'oggetto radice di tutto il BOM e DOM
- Infatti i seguenti oggetti (incluso l'oggetto document) sono tutti proprietà dell'oggetto window:
 - document
 - navigator
 - screen
 - location
 - history

Oggetto window

- Altre proprietà dell’oggetto window:
 - title titolo della finestra
 - statusbar testo mostrato sulla barra di stato
 - outerHeight, outerWidth dimensioni esterne
 - innerHeight, innerWidth dimensioni interne

Oggetto window

- Modificare il titolo della finestra corrente

```
window.title = "Questo è il nuovo  
titolo";
```

- Accedere ad un nuovo documento

```
window.location =  
"http://www.yahoo.com/";
```

- Calcolare l'area in pixel della finestra

```
var area = window.innerWidth *  
window.innerHeight;
```

Oggetto window

- Metodi
 - open(location, title) apre una nuova finestra
 - close() chiude la finestra corrente
 - alert(message) visualizza il messaggio in una finestra di dialogo (utile per il debug)
 - confirm(message) visualizza il messaggio e richiede una conferma all'utente
 - moveTo(x, y) sposta la finestra alle coordinate indicate
 - resizeTo(width, height) dimensiona la finestra

Oggetto window

- Creazione e posizionamento di una nuova finestra

```
var w = window.open("http://www.  
google.com/", "Google");  
w.moveTo(0, 0);
```

Oggetto window

- Visualizzazione di un messaggio

```
window.alert("Attenzione si è  
verificato un errore");
```

- Visualizzazione del browser in uso

```
window.alert("Sei connesso con " +  
navigator.appName + " versione " +  
navigator.version);
```

Oggetto window

- Richiesta conferma all'utente

```
if(confirm("Vuoi proseguire con  
l'operazione?")) {  
    //L'utente ha risposto SI  
    ...  
} else {  
    //L'utente ha risposto NO  
    ...  
}
```

Oggetto navigator

- L'oggetto `navigator` è una proprietà dell'oggetto `window` e rappresenta l'istanza del browser in cui lo script è in esecuzione
- Proprietà
 - `appCodeName` codice identificativo del browser
 - `appName` nome del browser
 - `appVersion` numero di versione

Oggetto history

- Proprietà dell’oggetto window
- Rappresenta la sequenza di pagine visitate dall’utente
- Tale sequenza è rappresentata mediante un vettore
- Metodi
 - back () torna alla pagina precedente
 - forward () passa alla pagina successiva

JavaScript in documenti HTML

- Il codice JavaScript viene inserito all'interno di una pagina HTML delimitato dall'elemento `<script>...</script>`
- Oppure è memorizzato in una risorsa (file) a parte e viene linkato dalla pagina web sempre attraverso l'elemento `<script>`:

```
<script type="text/javascript"  
src="myscript.js"></script>
```

JavaScript in documenti HTML

- Generalmente il codice è incluso all'interno del'intestazione (<HEAD>) del documento
- Il codice viene eseguito **prima** della visualizzazione del documento.
- In questa sezione si procede con la definizione delle funzioni e delle variabili globali

JavaScript in documenti HTML

```
<HTML>
<HEAD>
<SCRIPT TYPE="text/javascript">
    function f(...);
    var v=...;

    ...
</SCRIPT>
...
</HEAD>
...
</HTML>
```

JavaScript in documenti HTML

- Il codice JavaScript può risiedere in un documento esterno:

```
<SCRIPT TYPE="text/javascript"  
        SRC="URL.js">  
</SCRIPT>
```

- Ciò è utile per aumentare la modularità e “nascondere” all’utente il codice

JavaScript in documenti HTML

- Adoperando il metodo `write()` dell'oggetto `document` è possibile costruire in fase di caricamento parte del corpo del documento HTML (`<BODY>...</BODY>`)
- Tuttavia questo sistema è poco pratico e se ne sconsiglia l'impiego

IDE per JavaScript

- **Visual Studio Code:**

<https://code.visualstudio.com/>

- **Atom:**

<https://ide.atom.io/>

- **NetBeans:**

<https://netbeans.org/>

- **Komodo:**

<https://www.activestate.com/komodo-ide>

- **WebStorm:**

<https://www.jetbrains.com/webstorm/>

Riferimenti

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>
- <http://msdn.microsoft.com/>
- <http://www.html.it/javascript/>
- <http://www.ecma-international.org>

Sapienza Università di Roma
corso di laurea in Ingegneria informatica e automatica

Linguaggi e tecnologie per il Web

a.a. 2020/2021

Parte 3 CSS

Riccardo Rosati

Fogli di stile

- In HTML non c'è separazione tra **forma e contenuto** del documento
- Fogli di stile o Cascading Style Sheets (CSS) = estensione di HTML introdotta da Internet Explorer 3
- Esempio: tag
- I CSS si occupano di gestire la formattazione del documento esternamente al documento stesso
- Standard W3C (CSS1 e CSS2)

Tipi di fogli di stile

- HTML permette di utilizzare diversi linguaggi per fogli di stile
- Linguaggio standard W3C per i fogli di stile: CSS (Cascading Style Sheets)
- text/css

Tipi di CSS

- CSS in linea
 - agiscono su singole istanze di testo all'interno della pagina
- CSS incorporati
 - agiscono in modo globale su un singolo documento
- CSS esterni
 - agiscono in modo globale su insiemi di documenti

CSS in linea

Agiscono su singole istanze di testo all'interno della pagina

Esempio:

```
<DIV style="font-size:18px;  
          font-family:arial; color:red">  
    Questa parte di testo ha colore rosso  
</DIV>
```

Marcature usate:

- senza “semantica” (<DIV> o)
- marcature con “semantica” (es. <p>)

Limiti dei CSS in linea

- Semplici da usare (si possono considerare una “estensione” di HTML)
- Modificano il contenuto del testo
- Non rispondono all’esigenza di separazione tra forma e contenuto
- È opportuno usarli in modo molto circoscritto

CSS incorporati

- Agiscono in modo globale su un singolo documento HTML
- Vanno scritti nella parte intestazione del documento (tra <HEAD> e </HEAD>)
- Permettono di dare opzioni di formattazione **globale** ai tag (e quindi al documento)

CSS incorporati: esempio

```
<HTML>
<HEAD>
<style type="text/css">
H1 {font-size:17px; color:green}
H2 {font-family:arial; color:red}
</style>
</HEAD>
<BODY>
<H1>Tutti gli H1 sono di colore verde</H1>
<H2>Tutti gli H2 sono di colore rosso</H2>
</BODY>
</HTML>
```

CSS incorporati: sintassi

- Ogni dichiarazione CSS è una coppia **selettore – assegnazione di proprietà**
- Ad esempio:

```
H1 { font-size:17px; color:green }
```

- H1 è il selettore
- { font-size:17px; color:green } è l'assegnazione delle proprietà

CSS incorporati: sintassi

```
H1 { font-size:17px; color:green }  
H2 { font-family:arial; color:red }
```

- gli attributi sono inseriti tra parentesi graffe
- al posto del segno = vengono usati i due punti
- gli attributi composti da più termini sono separati da un trattino
- quando un attributo è considerato proprietà di un oggetto i trattini si eliminano e le iniziali dei termini diventano maiuscole
- esempio: font-style diventa FontStyle

Il tag <style>

```
<style type = "text/css">
```

- L'attributo TYPE del tag <STYLE> definisce il linguaggio in formato MIME del foglio di stile
- TYPE indica al browser il tipo di foglio di stile supportato
- Internet Explorer supporta i CSS solo in formato MIME

Il tag <style>

- Esistono anche i CSS in formato text/jass, cioè accessibili tramite JavaScript
- Se l'attributo TYPE viene omesso, il browser lo identifica di default con text/css

Attenzione: non confondere il **tag** <style> con
l'attributo style usato nei CSS in linea

CSS incorporati vs. CSS in linea

- I CSS incorporati permettono di configurare globalmente la formattazione del testo (cioè l'aspetto dei tag)
- I CSS incorporati sono definiti al di fuori del corpo del documento, e permettono così la separazione tra contenuto e forma
- Tuttavia, i CSS incorporati sono inadatti a trattare in modo uniforme insiemi di documenti (ad esempio un sito web)

CSS esterni

agiscono in modo **globale** su insiemi di documenti:

- gli stili dei singoli marcatori vengono raggruppati in un documento separato (file distinto dai documenti)
- ogni documento “richiama” gli stili (con un opportuno comando)
- una modifica sul file di stili genera automaticamente la stessa modifica su tutti i documenti che lo richiamano

CSS esterni

Esempio: file style.css:

```
H1 {font-size:17px; color:green}  
H2 {font-family:arial; color:red}
```

contiene gli stili che si vogliono definire

Collegamento ad un CSS esterno

Collegamento ad un CSS esterno in un documento:

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

- Il tag `<link>` identifica un file esterno al documento HTML
- L'attributo `rel` indica il tipo di file collegato (`stylesheet`)
- L'attributo `href` richiama il percorso e il nome del file esterno

Vantaggi dei CSS esterni

- Permettono la separazione tra contenuto e forma
- Permettono di gestire insiemi di documenti
- Massima flessibilità di utilizzo
- Riusabilità degli stili
- Possibilità di **fondere** insieme più stili (cascading)

Attributi di stile per il testo

- font-family (serif / sans serif / cursive)
- font-size
 - punti (pt) / pixel (px) / centimetri (cm) / pollici (in) / percentuale (%)
- font-style
 - extra-light / demi-light / light / medium /bold/ demi-bold / extra-bold
- font-variant (normal / small-caps)
- font-weight
- text-decoration (none / underline / italic / line-height)

Esempio

Per i CSS in linea:

```
<A style="text-decoration:none"  
     href="#a1">ciao </a>
```

Per i CSS incorporati:

```
<style>  
A {text-decoration: none}  
</style>
```

Versioni di CSS

- Il W3C ha rilasciato varie versioni di CSS:
 - CSS 1 (1996)
 - CSS 2 (1998)
 - CSS 2.1 (2011)
 - CSS 3 (2011-2014)
- Tra le differenze principali delle varie versioni, si segnala il numero di proprietà definite e quindi assegnabili:
 - CSS1 = 53 proprietà
 - CSS2 = 122 proprietà
 - CSS3 = 339 proprietà

Selettori CSS

- Fino ad ora abbiamo visto selettori CSS corrispondenti a nomi di elementi HTML (che selezionano tutti gli elementi del documento HTML con quel nome)
- In realtà si possono costruire selettori molto più complessi

Selettori CSS

- Selettore universale: *
- Selettore di nome: *nome-elemento* (es. H1)
- Selettore di classe: *.nome-classe*
- Selettore di classi multiple:
.nome-classe1.nome-classe2...
- Selettore di id: *#nome-id*
- Selettore di discendenti (sequenza)
- Selettore di figli (>)
- Selettori di fratelli (+), (~)

Selettori CSS

- Selettori di pseudo-classi:
 - E:link
 - E:visited
 - E:hover
 - E:focus
 - E:first-child
 - E:root
 - E:nth-child()

Selettori CSS

- Selettori di pseudo-classi:
 - E:nth-last-child()
 - E:last-child
 - E:only-child
 - E:nth-of-type()
 - E:nth-last-of-type()
 - E:first-of-type
 - E:last-of-type
 - E:only-of-type

Selettori CSS

- Selettori di pseudo-classi:
 - E:not
 - E:empty
 - E:target
 - E:enabled
 - E:disabled
 - E:checked

Selettori CSS

- Selettori di pseudo-classi:
 - E:default
 - E:valid
 - E:invalid
 - E:in-range
 - E:out-of-range
 - E:required
 - E:optional
 - E:read-only
 - E:read-write

Selettori CSS

- Selettori di pseudo-elementi:
 - ::first-letter
 - ::first-line
 - ::before
 - ::after

Selettori CSS

- Selettori di attributi:
 - $E[attribute]$
 - $E[attribute=value]$
 - $E[attribute\~=value]$
 - $E[attribute|=value]$
 - $E[attribute^=value]$
 - $E[attribute$=value]$
 - $E[attribute*=value]$

Classi

- A tutti gli elementi è possibile assegnare l'attributo CLASS
- Questo attributo permette ai fogli di stile di trattare singole occorrenze o sottosinsiemi di occorrenze dello stesso elemento o di elementi diversi

Classi: esempio

```
<STYLE>
    H2.top { font-family:verdana; font-size:15px;
              font-weight:bold; font-style:normal }
    H2.bottom { font-family:arial; font-size:10px;
                 font-weight:bold; font-style:italic }
</STYLE>

<H2 class=top> Testo della pagina</H2>
<H2 class=top> Altro testo della pagina</H2>
...
<H2 class=bottom> Testo della pagina</H2>
<H2 class=bottom> Altro testo della pagina</H2>
```

Pseudoclassi

- esempio: elementi anchor:
 - quando visitati costituiscono una pseudoclasse `visited`, quando attivi `active` e quando non visitati `link`
- viene specificata all'interno dello stile seguita dai due punti
- esempio:

```
<STYLE>
  A:link { text-decoration: none }
  A:visited { text-decoration: none }
</STYLE>
```

Identifieri

- A tutti gli elementi è possibile assegnare l'attributo ID: il suo valore identifica univocamente quell'elemento all'interno del documento
- Questo attributo permette ai fogli di stile di trattare singole occorrenze di elementi

Identifieri

Esempio:

...

```
<STYLE>
    #mioelem { font-weight:bold }
</STYLE>
```

...

```
<p id="mioelem">testo 1</p>
<p>testo 2</p>
```

...

Il paragrafo "testo 1" verrà visualizzato in neretto

Proprietà CSS nel DOM

- Nel DOM ogni elemento HTML ha la proprietà `style`, che contiene tutte le proprietà CSS dichiarate per quell'elemento
- Il valore di tale proprietà `style` è un oggetto `CSSStyleDeclaration`

Proprietà CSS nel DOM

- Proprietà dell'oggetto `cssStyleDeclaration`:
 - `length`
 - `cssText`
 - `parentRule`
- Metodi dell'oggetto `CSSStyleDeclaration`:
 - `item(n)`
 - restituisce il nome della proprietà CSS di indice n dell'elemento
 - `getPropertyValue(nomeProprietàCSS)`
 - `setProperty(nomeProprietàCSS)`
 - `removeProperty(nomeProprietàCSS)`
 - `getPropertyPriority(nomeProprietàCSS)`

Proprietà CSS nel DOM

- Nel DOM le proprietà CSS possono essere accedute come (sotto)proprietà della proprietà style dell'elemento:
- Esempio:

```
document.getElementById("abc").style.color="white";  
assegna il colore bianco all'elemento con id uguale ad  
"abc"
```

- Altro esempio:

```
document.getElementsByTagName("p")[0].style.text-  
align="right";  
assegna allineamento destro al primo elemento paragraph  
(p) del documento
```

Proprietà CSS nel DOM

- Se non si conoscono a priori le proprietà CSS definite per un elemento HTML, si possono estrarre a runtime
- Esempio:

```
var s=new Array();
var x=document.getElementById("abc");
for (var i=0;i<x.style.length;i++) s[i]=x.style.item(i);
```

memorizza nell'array s l'elenco delle proprietà CSS definite per l'elemento con id uguale ad "abc"

Riferimenti

- Raccomandazioni ufficiali W3C:
<https://www.w3.org/Style/CSS/>
- Tabella riassuntiva delle proprietà CSS rispetto alle varie versioni dello standard:
<http://meiert.com/en/indices/css-properties/>
- Tutorial CSS sul sito W3Schools:
<https://www.w3schools.com/css/default.asp>
- Sito italiano per gli sviluppatori HTML:
www.html.it

Sapienza Università di Roma
corso di laurea in Ingegneria informatica e automatica

Linguaggi e tecnologie per il Web

a.a. 2020/2021

Parte 4

JavaScript: JSON, tipi, oggetti, funzioni

Riccardo Rosati

JSON

- JSON = JavaScript Object Notation
- Formato standard per la serializzazione e de-serializzazione degli oggetti JavaScript
- Serializzazione = trasformazione dell'oggetto JavaScript in stringa
- De-serializzazione = trasformazione della stringa in oggetto JavaScript
- JSON è un formato standard (ECMA 404) per lo scambio dei dati

JSON

- Esempio 1: rappresentazione di un array:

```
[ "ciao", 45, null, true]
```

- Esempio 2: rappresentazione di un oggetto con due proprietà prop1 (stringa), prop2 (booleano):

```
{prop1: "ciao", prop2: true}
```

- Esempio 3: rappresentazione di un oggetto con due proprietà: p1, il cui valore è l'array dell'Esempio 1, e p2, il cui valore è l'oggetto dell'Esempio 2:

```
{p1: [ "ciao", 45, null, true],  
p2: {prop1: "ciao", prop2: true} }
```

JSON: valori

- Un testo JSON è la rappresentazione di un valore
- Un valore può essere:
 - oggetto
 - array
 - numero
 - stringa
 - true
 - false
 - null

valore -> oggetto | array | numero | stringa | "true"
| "false" | "null"

JSON: oggetti

- Un oggetto è una sequenza di coppie nome-valore (separate da virgole) racchiusa tra parentesi graffe

oggetto -> "{" (coppia altreCoppie)? "}"

coppia -> nome ":" valore

altreCoppie -> ("," nome ":" valore)*

JSON: array

- Un array è una sequenza di valori separati da virgole e racchiusi tra parentesi quadre

array -> "[" (valore altriValori)? "]"

altriValori -> ("," valore)*

JSON: numeri e stringhe

- I numeri sono rappresentati nella sintassi usuale dei linguaggi di programmazione (possono essere rappresentati sia numeri interi che in virgola mobile)
- Le stringhe sono sequenze di caratteri delimitate da doppi apici (o singoli apici)

JSON: esempi

```
var mioOggettoStudente = {  
    nome: "Mario",  
    annoDiNascita: 1979,  
    esamiSuperati: ["FI1", "FIS", "LTW"]  
}
```

```
var mioVerbaleEsame = {  
    numero: 123,  
    insegnamento: "LTW",  
    esami: [ { matr:11, voto:30 },  
             { matr:16, voto:27 },  
             { matr:35, voto:28 } ]  
}
```

L'oggetto JSON

- Oggetto JSON contenente due metodi:
 - Metodo per la de-serializzazione:
`JSON.parse(stringa)`
trasforma la stringa in oggetto
 - Metodo per la serializzazione:
`JSON.stringify(oggetto)`
trasforma l'oggetto in stringa

JSON.stringify: esempio

```
JSON.stringify(mioVerbaleEsame)
```

restituisce la stringa

```
"{ \"numero\":123, \"insegnamento\":\"LTW\", \"esami\" : [ { \"matr\":11, \"voto\":30}, { \"matr\":16, \"voto\":27}, { \"matr\":35, \"voto\":28} ] }"
```

JSON.parse: esempio

Assegnazione di mioOggettoStudente tramite
JSON.parse:

```
var mioOggettoStudente =  
JSON.parse('{"nome": "Mario", "annoDiNascita":  
1979, "esamiSuperati": ["FI1", "FIS", "LTW"] }');
```

- Nota: i nomi delle proprietà vanno messi tra doppi apici

L'operatore `typeof`

- L'operatore `typeof` di JavaScript restituisce una stringa che rappresenta il tipo dell'argomento dell'operatore
- Tipi restituiti da `typeof`:
 - `number`
 - `string`
 - `boolean`
 - `undefined`
 - `object`
 - `function`

L'operatore `typeof`: esempi

```
var x1 = 5;  
var x2 = "ciao";  
var x3 = true;  
var x4;  
var x5 = null;  
var x6 = new Array();  
var x7 = function(){var z=1;};
```

```
typeof x1 restituisce "number"  
typeof x2 restituisce "string"  
typeof x3 restituisce "boolean"  
typeof x4 restituisce "undefined"  
typeof x5 restituisce "object"  
typeof x6 restituisce "object"  
typeof x7 restituisce "function"
```

Valori e oggetti in JavaScript

- Le variabili possono essere associate a numeri, stringhe, e valori booleani in due modi:
- Assegnando la variabile al valore stringa/numero/booleano, senza creazione di oggetto:
- Esempio: `var x = "abcd";`
- Dichiarando/assegnando la variabile come oggetto di tipo String (o Number o Boolean):
- Esempio: `var y = new String("abcd");`
- La variabile x ha tipo string, mentre la variabile y ha tipo object:
 - `typeof x` restituisce "string"
 - `typeof y` restituisce "object"

Valori e oggetti in JavaScript

- Nel caso precedente, il confronto di uguaglianza per valore `x==y` è true (perché x e y hanno lo stesso valore)
- mentre il confronto di uguaglianza per tipo e valore `x==y` è false (x e y hanno tipi diversi)
- Se non è necessario, è preferibile assegnare una variabile ad un valore di tipo numero/stringa/booleano, e NON creare un oggetto (perché questo rallenta leggermente l'esecuzione del codice)

Null e undefined

- Abbiamo visto che una variabile non inizializzata ha tipo undefined:

```
var x4;
```

typeof x4 restituisce "undefined"

- mentre una variabile assegnata a null ha tipo object:

```
var x5 = null;
```

typeof x5 restituisce "object"

- Quindi, come nel caso precedente:

- il confronto di uguaglianza per valore $x4==x5$ è true (perché null è considerato un valore indefinito)
- mentre il confronto di uguaglianza per tipo e valore $x4==x5$ è false ($x4$ e $x5$ hanno tipi diversi)

Funzioni in JavaScript

- Anche se `typeof` è in grado di distinguere tra oggetti e funzioni, le funzioni in JavaScript di fatto sono anch'esse **oggetti**
- Possono pertanto, ad esempio, essere assegnate a delle proprietà
- Possono anche essere assegnate a delle variabili:

```
var p = function(a,b){  
    ...  
}
```

- Il precedente è un esempio di **funzione anonima** (o funzione lambda)

Funzioni in JavaScript

- La funzione precedente può essere invocata usando il nome della variabile:

```
var x = p(32,5);
```

oppure può essere invocata in questo modo:

```
(function(a,b){
```

```
...
```

```
}(32,5))
```

Funzioni in JavaScript

- Tutte le funzioni hanno proprietà e metodi definiti
- Proprietà:
 - name (nome della funzione)
 - length (numero di argomenti)
 - ...
- Metodi (per invocare la funzione):
 - call
 - apply
 - bind

Funzioni di callback

- Come gli altri oggetti, le funzioni possono essere passate come parametri nella chiamata ad un'altra funzione (**callback**)
- Esempio:

```
function f(x,y){...}  
function oper(f,a,b){  
return f(a,b); }
```

oppure

```
document.form1.bott.addEventListener("click",f);
```

Funzioni restituite da funzioni

- Come gli altri oggetti, le funzioni possono anche essere restituite come risultato di una funzione
- Esempio:

```
var x = function(y){  
    return function (z){  
        return y*z; };  
};
```

Metodi = funzioni = proprietà

- I metodi degli oggetti sono funzioni
- Ma le funzioni sono oggetti!
- In realtà quindi anche i metodi sono oggetti
- Più precisamente, ogni metodo è una proprietà che ha come valore una funzione
- In questo senso **non c'è più distinzione tra metodi e proprietà** in JavaScript
- Ogni oggetto è un contenitore di proprietà; i metodi dell'oggetto sono quelle proprietà valorizzate ad oggetti di tipo funzione

Il metodo addEventListener

- Un esempio tipico di uso di funzioni di callback avviene per il metodo addEventListener del DOM
- Questo è un metodo sia dell'oggetto document che degli oggetti element
- Ha due parametri:
 - evento (string)
 - gestore dell'evento o event handler (function)
- (c'è un terzo parametro booleano opzionale con cui si può decidere l'ordine di esecuzione degli event handler in elementi annidati)
- Nota: la funzione di callback non può avere argomenti

Il metodo addEventListener

Esempio: aggiungo un event handler che gestisce il click sull'elemento che ha id uguale a d1:

```
function f() {alert("hai fatto click sull'elemento con ID=d1");}
document.getElementById("d1").addEventListener("click",f);
```

Lo stesso esempio si può scrivere usando una funzione anonima:

```
document.getElementById("d1").addEventListener("click",
function() {alert("hai fatto click sull'elemento con ID=d1");});
```

Aggiungiamo un event handler a livello di documento (questo gestirà un qualunque click sul documento):

```
document.addEventListener("click",
function() {alert("hai fatto click sul documento");});
```

Il metodo addEventListener

```
document.getElementById("d1").addEventListener("click",
  function() {alert("hai fatto click sull'elemento con ID=d1");});
document.addEventListener("click",
  function() {alert("hai fatto click sul documento");});
```

- Cosa succede quando sono stati aggiunti entrambi gli event handler e faccio click sull'elemento che ha id uguale a d1?
- Vengono eseguiti **entrambi** gli event handler
- L'ordine di esecuzione dei due event handler dipende dalla **modalità di propagazione** degli eventi

Event bubbling e event capturing

- **Event bubbling:** l'evento è catturato prima dall'elemento più annidato, e poi da quelli più esterni, fino ad arrivare al livello del documento
- **Event capturing:** l'evento è catturato prima a livello di documento, poi di elemento HTML più esterno e via via fino all'elemento più annidato
- Il terzo argomento (opzionale) del metodo `addEventListener` può essere usato per decidere la modalità di propagazione (e quindi l'ordine di esecuzione degli event handler)
- Se il terzo argomento è `false` (o è assente), si sceglie event bubbling
- Se il terzo argomento è `true`, si sceglie event capturing
Es.: `document.addEventListener("click",
 function() {alert("hai fatto click sul documento");},true);`

Sapienza Università di Roma
corso di laurea in Ingegneria informatica e automatica

Linguaggi e tecnologie per il Web

a.a. 2020/2021

Parte 4(b)

ECMAScript, Web Storage API, user script, WebAssembly

Riccardo Rosati

JavaScript e ECMAScript

- Il linguaggio JavaScript fu introdotto da Netscape nel 1995 con il nome di LiveScript (autore: Brendan Eich)
- Nel 1997 fu rilasciata dalla European Computer Manufacturers Association (ECMA) la prima versione standard del linguaggio (ECMAScript 1)
- Seguirono negli anni varie versioni del linguaggio: le più importanti sono state ECMAScript 5 (2009) e ECMAScript 6 (2015)
- L'ultima versione rilasciata è l'undicesima (ECMAScript 2020)
- Esiste però un "living standard" anche per ECMAScript, chiamato ES.Next

JavaScript e ECMAScript

Quelle che abbiamo visto finora sono caratteristiche presenti in ECMAScript almeno dalla versione 5

Principali novità di ECMAScript 6:

- definizioni di classi
- funzioni (a) freccia (arrow functions)
- const
- let
- import e export di moduli
- promises (promesse)
- dati binari, ArrayBuffer, ...
- ...

Classi in JavaScript

Esempio:

```
class miaClasse {  
    constructor(n,c) {  
        this.nome= n;  
        this.cognome = c;  
    }  
}  
var o = new miaClasse("Riccardo","Rosati");
```

- Una classe in realtà è una funzione
- Le funzioni possono essere invocate come costruttori di oggetti (con new)

Classi in JavaScript

Esempio senza l'uso esplicito di class:

```
function miaClasse(n,c) {  
    this.nome= n;  
    this.cognome = c;  
}  
var o = new miaClasse("Riccardo","Rosati");
```

Classi, funzioni e hoisting

- Il costrutto `class` è sostanzialmente "zucchero sintattico"
- C'è però una differenza tra l'uso esplicito di `class` e quello della funzione
- Le classi devono essere definite prima di essere invocate
- Invece le funzioni, così come le variabili, possono essere definite anche dopo essere state invocate
- Infatti l'interprete JavaScript effettua il cosiddetto **hoisting** delle dichiarazioni delle funzioni e delle variabili: esegue tali dichiarazioni all'inizio del blocco in cui si trovano (programma o funzione), indipendentemente dalla posizione di tali dichiarazioni nel codice
- Questo non avviene per le dichiarazioni `class`, `let` e `const`

Arrow functions

Esempi:

```
x = (s,a,b) => { console.log(s); return a + b; };
```

corrisponde all'istruzione

```
x = function(s,a,b) { console.log(s); return a + b; }
```

```
(a,b) => a + b;
```

corrisponde all'istruzione

```
x = function(a,b) { return a + b; }
```

```
x = a => a + 1;
```

corrisponde all'istruzione

```
x = function(a) { return a + 1; }
```

Il simbolo => è chiamato fat arrow

Let

L'uso della parola chiave `let` permette di dichiarare variabili che hanno visibilità solo nel blocco in cui sono dichiarate (block scope).

Esempio:

```
if (test) {  
    var x = 1;  
    ...  
}  
...  
...
```

La variabile `x` è visibile
anche all'esterno del
blocco tra parentesi
graffe

```
if (test) {  
    let y = 1;  
    ...  
}  
...
```

La variabile `y` NON è
visibile all'esterno del
blocco tra parentesi
graffe

Const

L'uso della parola chiave `const` permette di dichiarare costanti, cioè variabili che non possono cambiare valore.

Valgono le stesse regole di visibilità di `let` (block scope).

Esempio:

```
if (test) {  
    const x = 1;  
    ...  
}  
...
```

La costante `x` è NON è visibile all'esterno del blocco tra parentesi graffe

TypeScript

- TypeScript è una estensione di JavaScript (per la precisione di ECMAScript 6) proposta da Microsoft a partire dal 2012
- La principale caratteristica di TypeScript è la **tipizzazione forte**
- È realizzata mediante un sistema di annotazione dei tipi
- Il codice TypeScript può essere compilato in JavaScript (e quindi essere eseguito da qualsiasi web browser)

HTML DOM

- Abbiamo visto finora solo alcuni aspetti del DOM HTML
- In realtà il DOM è una API molto più complessa
 - <https://dom.spec.whatwg.org/>
- Anche la parte degli eventi è molto complessa
- Proposta di standard più recente: UIEvents:
 - <https://w3c.github.io/uievents/>

Memorizzazione persistente

Per la memorizzazione permanente in script lato client:

- Interazione con il web server (l'applicazione lato server memorizza in modo permanente i dati inviati dal client)
- Uso di cookie o file locali (Web Storage API)
- Uso di database locali (IndexedDB API)

Web Storage API

- Nuove API in HTML5: Web Storage API
- Due nuovi oggetti:
 - `localStorage`
 - `sessionStorage`
 - Derivano entrambi dal nuovo oggetto `Storage`
- `localStorage` permette la memorizzazione persistente
- `sessionStorage` permette la memorizzazione a livello di sessione

Memorizzazione persistente

- Entrambi gli oggetti sono utilizzati definendo nuove coppie chiave-valore
- Es. `localStorage.cognome="Rosati"` memorizza una nuova coppia
- I valori assegnabili ad una proprietà dello storage **devono essere di tipo String**
- Per poter memorizzare oggetti arbitrari JavaScript, è necessario serializzarli (ad es. tramite il metodo `JSON.stringify`)

Esempio

```
localStorage.cognome="Rosati";  
localStorage.nome="Riccardo";  
var o = { s1: 5, p2: "ciao" };  
localStorage oggetto=JSON.stringify(o);  
alert(localStorage.cognome + " " +  
      localStorage["nome"]);
```

Altri metodi dello storage

Altri metodi dell'oggetto Storage (e quindi degli oggetti sessionStorage e localStorage):

- `getItem(chiave)`
- `setItem(chiave, valore)`
- `removeItem(chiave)`
- `clear()`

L'esecuzione dei metodi `setItem`, `removeItem` e `clear` genera l'evento `storage` (che può essere gestito come gli tutti altri eventi del DOM)

Gestione dei dati dello storage

Analogie e differenze con i cookie:

- Gli oggetti del Web Storage hanno un limite di dimensione di (almeno) 5 MB (molto maggiore dei cookie)
- I dati contenuti in questi oggetti NON vengono trasmessi automaticamente al server (a differenza dei cookie)
- Ogni dato (proprietà) contenuto negli oggetti storage è visibile solo dagli script provenienti dallo stesso dominio dello script che lo ha creato (same origin policy): questa politica è simile (anche se non esattamente uguale) a quella adottata per i cookie

IndexedDB API

- Le IndexedDB API rispondono a necessità di memorizzazione permanente lato client che localStorage e sessionStorage non possono soddisfare
- Permettono di usare un Object store (database) lato client
- Non è un database relazionale e non si usa il linguaggio SQL
 - una proposta in tal senso, chiamata Web SQL Database API, non è stata adottata come standard W3C
- Proprietà indexedDB dell'oggetto window
- Per i dettagli della API si rimanda alla documentazione ufficiale: <https://www.w3.org/TR/IndexedDB/>

User script

- JavaScript può essere usato per definire i cosiddetti **user script** dei browser
- Gli user script sono script che alcuni browser (tramite opportune estensioni) eseguono automaticamente in corrispondenza al caricamento di certe pagine web
- Ogni user script è definito per una classe di pagine (ad esempio, tutte le pagine provenienti da un dominio o da una porzione di un dominio)

Userscript manager

Le estensioni/plugin dei browser che permettono l'esecuzione di user script sono chiamate **userscript manager**

Principali estensioni di questo tipo:

- Greasemonkey (Firefox)
- Tampermonkey (Chrome, Firefox, Safari, Edge)
- Ace Script (Firefox)
-

Esempio di user script

Vogliamo scrivere uno user script che fa aggiungere automaticamente al browser la visualizzazione (come primo elemento della pagina) dell'immagine del logo Sapienza.

Vogliamo applicare tale script a tutte le pagine web la cui URL inizia con il seguente prefisso:

`http://www.dis.uniroma1.it/~rosati/`

Tuttavia (come eccezione alla precedente regola) non vogliamo applicare tale script alle pagine web la cui URL inizia con il seguente prefisso:

`http://www.dis.uniroma1.it/~rosati/dmdu/`

Esempio di user script (Greasemonkey)

```
// ==UserScript==  
// @version      1.0  
// @name        Esempio2  
// @description Esempio 2 di userscript per LTW  
// @match        http://www.dis.uniroma1.it/~rosati/*  
// @exclude      http://www.dis.uniroma1.it/~rosati/dmdu/*  
// ==/UserScript==  
function aggiungiLogoSapienza(){  
    var body1 = document.getElementsByTagName("body");  
    var body = body1[0];  
    var logoSapienza = document.createElement("img");  
    logoSapienza.setAttribute("src","https://www.uniroma1.it/sites/  
default/files/images/logo/sapienza-big.png");  
    body.insertBefore(logoSapienza,body.firstChild);  
    alert("Fatto!");  
}  
aggiungiLogoSapienza();
```

WebAssembly

- WebAssembly (Wasm o WA) è un linguaggio standardizzato dal World Wide Web Consortium alla fine del 2019
- Come dice il nome, è un linguaggio assembler per le applicazioni Web
- Linguaggio di programmazione di basso livello (per una stacked virtual machine)
- Codice binario
- Permette l'esecuzione efficiente di codice sul lato client
- È supportato da tutti i principali web browser
- È basato sul linguaggio asm proposto da Mozilla

Link

Riferimenti ufficiali:

- ECMAScript:
 - <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>
 - TypeScript:
 - <https://www.typescriptlang.org/>
- HTML DOM:
 - <https://dom.spec.whatwg.org/>
- UIEvents (eventi):
 - <https://w3c.github.io/uievents/>

Link

Web storage API:

- <https://www.w3.org/TR/webstorage/>

Indexed Database API:

- <https://www.w3.org/TR/IndexedDB/>

Greasemonkey:

- <http://www.greasespot.net/>

Tampermonkey:

- <https://tampermonkey.net/>

WebAssembly:

- <https://www.w3.org/TR/wasm-core-1/>
- <https://webassembly.org/>

Il linguaggio PHP

Lorenzo Marconi



Anno accademico 2020/2021

PHP: acronimo ricorsivo che sta per *PHP Hypertext Preprocessor*. Nel corso delle esercitazioni useremo la versione PHP5

- PHP è un preprocessore di ipertesti open source. In particolare, è un linguaggio di scripting interpretato, usato principalmente per la programmazione di pagine web dinamiche
- Per farne il download, su sistemi Windows si consiglia l'installazione di XAMPP, mentre su sistemi Linux è sufficiente lanciare, tramite shell, il seguente comando: *sudo apt-get install php*

- È un linguaggio di scripting, general-purpose ed open source molto utilizzato
- È un linguaggio multiplattaforma, ovvero è possibile sviluppare la propria applicazione web su diversi sistemi operativi (Windows, Linux o Mac)
- Semplice da imparare. Come vedremo la sua sintassi è simile a C, C++ e Java
- Particolarmente indicato per lo sviluppo Web. Infatti è un linguaggio *HTML-embedded*, ovvero permette di inserire codice PHP all'interno di una pagina contenente codice HTML
- Si integra perfettamente con i più diffusi DBMS tra cui gli open source PostgreSQL e MySQL
- È supportato dai più diffusi web server, primi su tutti Apache e nginx

Consideriamo la classica architettura client-server

Il codice PHP viene usato per generare dinamicamente i documenti HTML che il client deve ricevere e visualizzare nel browser.

- In un sito web statico, il documento HTML (non-PHP) viene preso dal file system del web server e inviato direttamente al client
- In un sito web dinamico, i file del web server (ad esempio, file PHP) vengono prima passati all'interprete PHP del web server stesso, e successivamente viene generato il documento HTML da inviare al client

In altre parole, l'output derivato dall'interpretazione di una porzione di codice PHP non è nient'altro che codice lato client (ad esempio, HTML, JavaScript, etc.) eseguibile da un browser

Un esempio banale

Di seguito un semplice file con estensione .php:

```
<html>
  <head>
    <title> Primo script PHP </title>
  </head>
  <body>
    <?php
      echo "<h1> Ciao. Sono un semplice script PHP </h1><br/>";
      echo "<h2> Vengo convertito in codice HTML dall'interprete PHP </h2>";
    ?>
  </body>
</html>
```

Questo è il corrispondente output HTML generato dall'interprete PHP:

```
<html>
  <head>
    <title> Primo script PHP </title>
  </head>
  <body>
    <h1> Ciao. Sono un semplice script PHP </h1><br/>
    <h2> Vengo convertito in codice HTML dall'interprete PHP </h2>
  </body>
</html>
```

PHP Echo è uno dei costrutti basilari di PHP. Esso permette infatti di mostrare a video un valore sotto forma di stringa

Da tenere bene a mente:

- ① Un file PHP può essere composto da vari script PHP mescolati a codice lato client (ad esempio, HTML e JavaScript)
- ② Ogni script PHP è compreso tra un apposito tag di apertura “`<?php`” ed un apposito tag di chiusura “`?>`”. L’interprete PHP esegue solo codice contenuto all’interno di questi due delimitatori
- ③ L’output di un file PHP è un documento HTML
- ④ Quando il client di un sito web richiede una pagina PHP, esso riceve un documento HTML e lo visualizza nel browser,
senza vedere il codice PHP che lo ha originato

Per maggiori dettagli, rimandiamo alla documentazione ufficiale. Di seguito, discuteremo solo di alcuni degli aspetti più importanti.

- La sintassi di PHP è simile a quella di C,C++ e Java (ma senza tipi). Una stessa variabile può essere assegnata a valori di tipo diverso in momenti diversi
- Non esiste un costrutto di dichiarazione di variabili
- Il nome delle variabili sono preceduti dal simbolo \$. Ad, esempio “\$a=5;” è una istruzione PHP che assegna alla variabile di nome a il valore 5
- Si può verificare se una variabile è stata inizializzata tramite la chiamata alla funzione “**isset(\$var)**” che ritorna true se e solo se la variabile var è stata precedentemente inizializzata
- PHP è *case-sensitive* (distingue tra maiuscole e minuscole)
- Supporta commenti multilinea con la stessa sintassi di C++ e Java: /* Questo è un commento in PHP */

- Addizione: $\$a + \b
- Sottrazione: $\$a - \b
- Moltiplicazione: $\$a * \b
- Divisione: $\$a / \b
- Modulo: $\$a \% \b

`$x = 12;`

`$y = 5;`

```
$x + $y;      /* 17 */
$x - $y;      /* 7 */
$x * $y;      /* 60 */
$x / $y;      /* 2.4 */
$x \% $y;    /* 2 */
```

- AND: \$a && \$b
- OR: \$a || \$b
- NOT: !(\$a)

```
$a=true;
```

```
$b=false;
```

```
$a && $b;      /* true */  
$a || $b;     /* false */  
!$a;         /* false */
```

Operatori di confronto

- Uguale a: ==
- Uguale a e dello stesso tipo: ===
- Non uguale a: !=
- Non uguale o di tipo diverso di: !==
- Minore di: <
- Maggiore di: >

`$x = 10;`

`$y = "10";`

```
$x == $y;      /* true */  
$x === $y;     /* false */  
$x != $y;      /* false */  
$x !== $y;     /* true */  
$x < $y;       /* false */  
$x > $y;       /* false */
```

- Selezione:

- **if** (*condizione*) { ... } **else** { ... }
- **switch** (\$var) {
 - **case** "a": ... **break**;
 - **case** "b": ... **break**;
 - ...
 - **default**: ... }

- Iterazione:

- **while** (*condizione*) { ... }
- **do** { ... } **while** (*condizione*)
- **for** (\$i=0; \$i<\$var; \$i=\$i+1) { ... }

In PHP le stringhe possono essere espresse in due maniere:

- ① Delimitata da apici singoli `$a='World';`
- ② Delimitata da doppi apici `$a="World";`

Quando si usano gli apici, la stringa viene interpretata esattamente come è scritta. Quando si usano le virgolette, le eventuali variabili contenute nella stringa vengono sostituite con il loro valore

```
$b='Hello $a'; /*Il valore della variabile b dopo questa istruzione sarà la  
seguente stringa: Hello $a*/
```

```
$c="Hello $a"; /*Il valore della variabile c dopo questa istruzione sarà  
la seguente stringa: Hello World*/
```

L'operazione di concatenazione di stringhe è il punto:

- `$a = "Hello";`
- `$b = "World";`
- `$c = "$a" . " " . "World"; /* Il valore della variabile c dopo questa istruzione sarà la seguente stringa: Hello World */`

Altre operazioni utili su stringhe:

- **strlen(\$c)**, restituisce la lunghezza della stringa `$c` (11, nel caso di sopra)
- **substr(\$c,3,4)**, restituisce la stringa dal terzo carattere di `$c` per i prossimi 4 caratteri ("lo W", nel caso di sopra)
- **strpos(\$c,\$b)**, restituisce la posizione della prima occorrenza della stringa `$b` all'interno di `$c`, oppure false nel caso non venga trovata alcuna occorrenza (6, nel caso di sopra)
- **strrev(\$c)**, restituisce la stringa inversa di `$c` ("dlrow olleH", in questo caso)

PHP gestisce un unico tipo di array, le cui chiavi possono essere numeriche o associative

Esempio di array con chiavi associative:

- `$a = array("fo" => "pippo", "bar" => "pluto");`
- `$a["fo"]` sarà la stringa pippo

Esempio di array con chiavi numeriche:

- `$a = array("pippo", "pluto");`
- `$a[0]` sarà la stringa pippo

Scorrere gli elementi di un array

Per scorrere gli elementi di un array \$a in PHP si può usare il seguente costrutto:

- **foreach** (\$a as \$index => \$var) {
 ...
}

che scorre l'array \$a associando alla variabile \$var i valori contenuti, e alla variabile \$index le chiavi (numeriche o associative a seconda dell'array \$a) corrispondenti

Esempio array

Di seguito un file con estensione .php:

```
<html>
  <head>
    <title> Script Array PHP </title>
  </head>
  <body>
    <?php
      $a = array('foo' => 'pippo', 'bar' => 'pluto', 'altro' => 'paperino');
      foreach ($a as $index => $var){
        echo "Valore associato all'indice " . $index . ":" . $var . "<br/>";
      }
    ?>
  </body>
</html>
```

Questo è il corrispondente output HTML generato dall'interprete PHP:

```
<html>
  <head>
    <title> Script Array PHP </title>
  </head>
  <body>
    Valore associato all'indice foo: pippo<br/>
    Valore associato all'indice bar: pluto<br/>
    Valore associato all'indice altro: paperino<br/>
  </body>
</html>
```

- Contrariamente a molti linguaggi di programmazione, in primis C e Java, PHP non richiede né la dichiarazione del tipo di dato restituito da una funzione, né il tipo dei parametri passati
- Come in JavaScript, una funzione viene dichiarata con la keyword **function**, seguita dal nome della funzione e, tra parentesi tonde, dal nome dei parametri necessari alla funzione. Il codice della funzione è delimitato da due parentesi graffe
- Di default, una funzione non vede le variabili create fuori della funzione stessa. Per poter accedere a una variabile dichiarata al di fuori delle funzione stessa, bisogna usare la dichiarazione **global** dentro la funzione
- Le variabili usate dentro a una funzione sono locali alla funzione stessa
- Le funzioni possono prevedere un valore di ritorno da restituire tramite l'istruzione **return**

Esempio Funzioni

Di seguito un file con estensione .php:

```
<html>
  <head>
    <title> Script Function PHP </title>
  </head>
  <body>
    <?php
      function somma($a , $b){
        return $a+$b;
      }
      function printRandomValue(){
        $firstValue=10;
        $secondValue=20;
        echo "<h1>" . somma( $firstValue , $secondValue ) . "</h1>";
      }
      printRandomValue();
    ?>
  </body>
</html>
```

Questo è il corrispondente output HTML generato dall'interprete PHP:

```
<html>
  <head>
    <title> Script Function PHP </title>
  </head>
  <body>
    <h1>30</h1>
  </body>
</html>
```

Tra le sue molteplici funzionalità, PHP fornisce API (funzioni di interfaccia) per accedere a database

Per connettersi con PHP ad un database, è necessario fornire

- **host**: nome dell'host
- **port**: numero di porto del server dove è attivo il DBMS
- **dbname**: nome del database
- **user**: nome dell'utente per la connessione al database
- **password** relativa password

Di seguito, forniremo alcuni dettagli utili per la connessione di PHP ad un database relazionale PostgreSQL

Un tipico esempio di connessione ad un database relazionale PostgreSQL è il seguente:

- `$dbconn = pg_connect("host=localhost, port=5433,
dbname=EsempioConnessionePHP, user=postgres
password=password") or die('Could not connect: '.
pg_last_error());`

Se la connessione va a buon fine (la funzione `pg_connect()` non restituisce alcun errore), allora potremo interagire con la base di dati "EsempioConnessionePHP" che si trova sulla nostra macchina ("localhost") su un DBMS PostgreSQL che ascolta sul numero di porto "5433" con credenziali di accesso "postgres" e "password". In caso contrario, la funzione `die()` terminerà lo script PHP con il messaggio di errore passatogli come parametro

Interazione con la base di dati

Una volta stabilita la connessione con la base di dati, sarà possibile interagire con essa per creare ed eliminare tabelle, fare operazioni di aggiunta/modifica/inserimento di tuple, così come poter eseguire query.

Continuando con l'esempio precedente, assumiamo che la base di dati "EsempioConnessionePHP" abbia una tabella di nome "organizzazione". Se siamo interessati a tutte le tuple di questa tabella, possiamo lanciare le seguenti istruzioni:

- \$query = 'SELECT * FROM organizzazione';
- \$result = **pg_query(\$query)** or **die('Query failed: ' . pg_last_error());**

La prima istruzione crea una variabile di nome query avente la query che siamo interessati a lanciare. La seconda istruzione lancia la query tramite la funzione **pg_query()** e salva il risultato nella variabile \$result, che sarà un oggetto iterabile corrispondente ad un array per ogni tupla del risultato della query

Nella prossima slide mostreremo come maneggiare le tuple risultanti da una query grazie alla funzione **pg_fetch_array()** che consente di iterare su oggetti iterabili. Nello specifico l'obiettivo sarà quello di creare una tabella HTML dove ogni riga è una tupla della tabella "organizzazione"

Esempio completo

```
<html>
    <title>Esempio in PHP</title>
    <head></head>
    <body>
        <?php
            $dbconn = pg_connect(" host=localhost port=5433
                dbname=EsempioConnesionePHP user=postgres password=password")
            or die('Could not connect: ' . pg_last_error());
            $query = 'SELECT * FROM organizzazione';
            $result = pg_query($query) or die('Query failed: ' .
                pg_last_error());
            // Printing results in HTML
            echo "<table>\n";
            while ($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
                echo "\t<tr>\n";
                foreach ($line as $col_value) {
                    echo "\t\t<td>$col_value</td>";
                }
                echo "\t</tr>\n";
            }
            echo "</table>\n";
            pg_free_result($result);
            pg_close($dbconn);
        ?>
    </body>
</html>
```

AJAX

Riccardo Rosati

Linguaggi e tecnologie per il Web

Corso di laurea in Ingegneria informatica e automatica
Sapienza Università di Roma
a.a. 2020/2021

<http://www.diag.uniroma1.it/rosati/ltw/>



AJAX

- AJAX = Asynchronous Javascript and XML
- Tecnologia basata su JavaScript e sull'interazione **asincrona** tra web client e web server
- Interazione asincrona: il web client (browser) fa una richiesta al server ma, a differenza delle altre forme di interazione previste, NON si interrompe fino all'arrivo della risposta (response) da parte del server
- Inoltre è possibile (ri)caricare solo una piccola parte della pagina attualmente visualizzata dal browser: ciò comporta maggiore velocità di esecuzione del browser e maggiore fluidità nell'interazione con l'utente



AJAX

- XML è il formato inizialmente previsto per lo scambio dei dati tra client e server
- Tuttavia, i dati possono essere scambiati anche in altri formati (JSON o altro)



AJAX

- AJAX è basato sull'interazione asincrona tra server e client
- **XML Http Request Object (XHR)** è l'oggetto che permette la comunicazione asincrona tra client e server
- Per attivare tale tipo di comunicazione, il client deve creare un nuovo oggetto XHR ed usare gli attributi e i metodi di questo oggetto
- XHR non è un oggetto standard del DOM W3C, ma è standard WHATWG ed è attualmente supportato da tutti I browser (ma in modi diversi)



XMLHttpRequest object: attributi

readyState	1 = Open 2 = Sent 3 = Received 4 = Loaded
Status	200 = ok 404 = page not found
statusText	Contiene l'etichetta dello status
responseText	Contiene i dati caricati (stringa di caratteri). Assume il suo valore finale quando readyState diventa uguale a 4
responseXML	Contiene il documento XML caricato (oggetto XML document). È significativo solo se readyState = 4, altrimenti è null
onreadystatechange	Funzione invocata quando cambia l'attributo readyState



XHR object: metodi

abort()	Interrompe tutte le attività create dell'oggetto e lo resetta
getAllResponseHeaders()	Restituisce tutti gli header in una stringa
getResponseHeader(DOMString)	Restituisce gli header dei dati ricevuti dopo l'ultima request
open(mode, url, boolean [, login, password])	mode: tipo della HTTP request (GET, POST, HEAD...) url: indirizzo del file boolean: true (asincrona) / false (sincrona) (login e password opzionali)
send("string")	Invia una stringa. Se la request è GET deve essere vuoto o null. Causa una DOMException (INVALID_STATE_ERR) se readyState è diverso da 1
setRequestHeader(DOMString, DomString)	Gli argomenti sono nome dell'header e valore Causa una DOMException (INVALID_STATE_ERR) se readyState è diverso da 1



AJAX: esempio

```
<!DOCTYPE HTML>
<html>
  <head><title>AJAX: semplice esempio</title></head>
  <body>
    <div>
      <button>Documento_1</button>
      <button>Documento_2</button>
      <button>Documento_3</button>
      <button>Documento_4</button>
    </div>
    <hr/>
    <div id="zonaDinamica">
      Seleziona il documento da visualizzare
    </div>
    <hr/>
    Resto del documento<br/>
    ...
  </body>
</html>
```



AJAX: esempio (segue)

```
<script>
  var documenti = document.getElementsByTagName("button");
  for (var i = 0; i < documenti.length; i++) {
    documenti[i].onclick = caricaDocumento;
  }

  function caricaDocumento(e) {
    var httpRequest = new XMLHttpRequest();
    httpRequest.onreadystatechange = gestisciResponse;
    httpRequest.open("GET", e.target.innerHTML+".htm", true);
    httpRequest.send();
  }

  /* si noti l'uso della proprietà target degli eventi,
  che restituisce l'elemento che ha causato l'evento */

```



AJAX: esempio (segue)

```
function gestisciResponse(e) {  
    if (e.target.readyState == 4 && e.target.status == 200) {  
        document.getElementById("zonaDinamica").innerHTML  
        = e.target.responseText;  
    }  
}  
</script>  
</body>  
</html>
```



Esempio

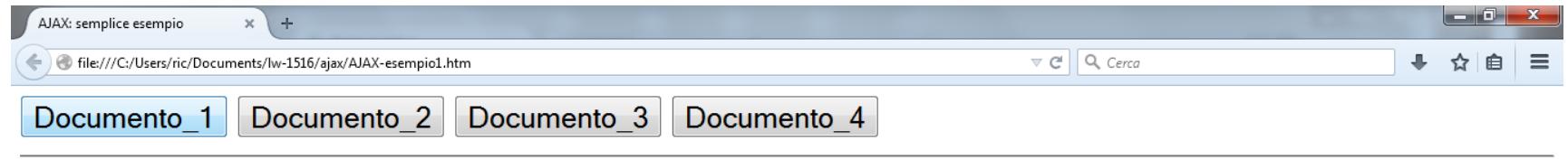
Documento iniziale:

A screenshot of a web browser window titled "AJAX: semplice esempio". The address bar shows the URL "file:///C:/Users/ric/Documents/lw-1516/ajax/AJAX-esempio1.htm". Below the title bar, there are four buttons labeled "Documento_1", "Documento_2", "Documento_3", and "Documento_4". A horizontal line separates this from the main content area. In the content area, the text "Selezione il documento da visualizzare" is displayed. Below it, the text "Resto del documento" is followed by three dots (...).



Esempio (segue)

Cliccando sul bottone Documento_1 si ottiene:



Documento 1

Questo è il documento 1

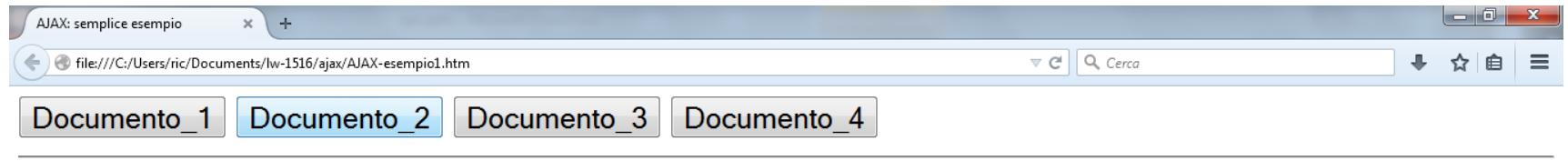
Resto del documento

...



Esempio (segue)

Cliccando sul bottone Documento_2 si ottiene:



Documento 2

Questo è il documento 2

Resto del documento

...



Esempio: commenti

- Nota bene: ad ogni clic su un bottone, non viene ricaricato tutto il documento
- Solo il contenuto del tag `<div id="target">` del documento iniziale viene modificato dall'interazione asincrona con il server
- Ricordiamo che ad ogni invocazione di funzione associata ad un evento viene automaticamente passato un argomento corrispondente all'evento stesso (argomento `e` nelle dichiarazioni delle funzioni dell'esempio)
- L'evento ha l'importante proprietà **target** che contiene l'oggetto che ha causato l'evento



Interazione sincrona

- Nel caso di open sincrone (attributo boolean=false), l'interprete JavaScript si ferma fino a che non viene ricevuta risposta dal server
- per questo le open sincrone vanno in genere evitate o comunque limitate



Caricamento di documenti XML

- AJAX e l'oggetto XMLHttpRequest trattano in modo speciale il caso in cui la risorsa oggetto della comunicazione tra client e server sia un documento XML
- In tal caso, il documento è memorizzato, sottoforma di oggetto XML document, nell'attributo **responseXML** (e non in **responseText**)
- per manipolare tale oggetto si possono usare i metodi del Document Object Model (DOM) di XML



Esempio con risorsa XML

Supponiamo ora che i precedenti documenti da caricare siano file XML.

Supponiamo ad esempio che il file Documento_1.xml sia:

```
<?xml version="1.0"?>
<radiceDocumento>
    <titolo>
        Documento 1
    </titolo>
    <contenuto>
        ...
    </contenuto>
</radiceDocumento>
```



Esempio con risorsa XML (segue)

(stessa struttura per i file Documento_2.xml, Documento_3.xml e Documento_4.xml)

In tal caso tali documenti verrebbero ricevuti nell'attributo responseXML.

La funzione caricaDocumento andrebbe modificata nel seguente modo:

```
function caricaDocumento(e) {  
    var httpRequest = new XMLHttpRequest();  
    httpRequest.onreadystatechange = gestisciResponse;  
    httpRequest.open("GET", e.target.innerHTML + ".xml", true);  
    httpRequest.send();  
}
```



Esempio con risorsa XML (segue)

La funzione gestisciResponse andrebbe modificata come segue:

```
function gestisciResponse(e) {  
    if (e.target.readyState == 4 && e.target.status == 200){  
        var resp = e.target.responseXML;  
        var x = resp.getElementsByTagName("titolo");  
        document.getElementById("zonaDinamica").innerHTML=  
            "<h1>" +x[0].childNodes[0].nodeValue+ "</h1>";  
    }  
}
```

(viene visualizzato il contenuto testuale dell'elemento titolo del file XML)



JQuery

Riccardo Rosati

Linguaggi e tecnologie per il Web

Corso di laurea in Ingegneria informatica e automatica
Sapienza Università di Roma

a.a. 2020/2021

<http://www.diag.uniroma1.it/rosati/ltw/>



Librerie e framework per JavaScript

Libreria:

- insieme di funzioni e strutture dati che possono essere utilizzate da una applicazione
- È il codice dell'applicazione ad accedere alla libreria (tramite collegamenti)
- **JQuery, React, JQueryUI, Dojo** sono esempi di librerie JS

Framework:

- architettura di supporto alla progettazione dell'applicazione
- condiziona la struttura di base e lo sviluppo del codice dell'applicazione
- il codice è inserito nelle strutture del framework (il framework "accede" al codice dell'applicazione)
- **AngularJS, Ember.js, Vue.js** sono esempi di framework per JS



JQuery

Prima versione rilasciata nel gennaio 2006

Obiettivi: creare una libreria in grado di:

- Semplificare la programmazione in JavaScript
- estendere le funzioni native di JavaScript
- garantire il funzionamento cross-browser degli script



JQuery

Il framework JQuery può essere scaricato in due formati:

- development (non compresso)
 - <https://code.jquery.com/jquery-3.5.0.js>
- production (compresso)
 - <https://code.jquery.com/jquery-3.5.0.min.js>

Esistono varie release (versioni) del framework



Includere JQuery in un documento HTML

La libreria JQuery è utilizzabile in vari modi in un documento HTML:

1. Scaricando la libreria in locale (stessa directory del documento HTML, o sua sottodirectory) e creando nel documento un collegamento con la copia locale
2. Creando nel documento un collegamento con la libreria presente nel repository ufficiale di jquery.com (JQuery CDN)
3. Creando nel documento un collegamento con la libreria presente in un altro Content Delivery Network



Includere JQuery in un documento HTML

Assumendo di avere nella cartella locale il file scaricato da jquery.com:

- Collegamento alla versione compressa di JQuery:

```
<script src="jquery-3.5.0.min.js">  
  ...  
</script>
```

- Collegamento alla versione non compressa di JQuery:

```
<script src="jquery-3.5.0.js">  
  ...  
</script>
```



Includere JQuery in un documento HTML

Collegamento al repository di jquery.com:

- Collegamento alla versione compressa di JQuery:

```
<script src="//code.jquery.com/jquery-3.5.0.min.js">  
  ...  
</script>
```

- Collegamento alla versione non compressa di JQuery:

```
<script src="//code.jquery.com/jquery-3.5.0.js">  
  ...  
</script>
```

Nota bene: in questo modo il funzionamento degli script dipende dalla raggiungibilità di queste risorse su un altro server



Uso di un CDN

- In alternativa si può utilizzare una copia del framework presente su un altro content delivery network (CDN)
- Ad esempio, dal CDN di Google:

```
<script src=
    "http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.js">
</script>
```
- Anche in questo caso il funzionamento degli script dipende dalla raggiungibilità di risorse su un altro server
- Se l'utente ha già scaricato questa risorsa nella cache del browser, si ha una diminuzione del tempo di caricamento della pagina
- Inoltre i CDN (ad es. Google) includono molte librerie/framework oltre JQuery e offrono garanzie di compatibilità se si includono più framework



Esempio

Esempio di pagina che usa la libreria JQuery:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Demo</title>
  </head>
  <body>
    <script src="//code.jquery.com/jquery-3.5.0.min.js">
      <script> // script dell'utente... </script>
    ...
  </body>
</html>
```



JQuery: sintassi

- Sintassi di base: `$(selettore).azione()`
- Il selettore seleziona uno o più elementi del documento HTML
 - Usa la sintassi CSS
- L'azione è una funzione JQuery sugli elementi selezionati
- Esempi:
 - `$(this).hide()` nasconde l'elemento corrente
 - `$("#mio-id").fade()` fa il fading dell'elemento con id uguale a mio-id



Selettori

- Oggetto JQuery (denotato anche con il simbolo del dollaro \$)
- `$("nome-elemento")` = seleziona gli elementi HTML con nome nome-elemento
- `$("#valore-id")` = seleziona l'elemento HTML con valore dell'attributo id uguale a valore-id
- `$(".nome-class")` = seleziona l'elemento HTML con valore dell'attributo class uguale a nome-class
- Esempio: `$("p.myclass")` seleziona tutti gli elementi p con valore dell'attributo class uguale a myclass
- Si possono anche usare i metodi del DOM
 - Esempio: `$(getElementById(valore-id))`



Selettori

- `$("*")` = seleziona tutti gli elementi HTML
- `$(this)` = seleziona l'elemento HTML corrente
- `$(".nome-class")` = seleziona l'elemento HTML con valore dell'attributo class uguale a nome-class
- Si possono usare filtri (preceduti da ":")
- Esempi:
 - `$("p:first")` = seleziona il primo elemento p
 - `$("p:last")` = seleziona l'ultimo elemento p
 - `$("p:even")` = seleziona gli elementi p pari
 - `$("p:odd")` = seleziona gli elementi p dispari
 - ...



Selettori

- Si possono scrivere più selettori contemporaneamente:
 - `$("#first, #mio-id")` seleziona sia il primo p che l'elemento con id mio-id
- Si possono scrivere selettori gerarchici (padre-figlio o predecessore-successore)
 - `$("#mio-id p")` seleziona gli elementi p **successori** dell'elemento con id mio-id
 - `("#mio-id > p")` seleziona gli elementi p **figli** dell'elemento con id mio-id
 - `("#mio-id + p")` seleziona l'elemento p che **segue** l'elemento con id mio-id
- Si possono usare condizioni sugli attributi:
 - `("a[title='nuovo'])")` seleziona gli elementi a con attributo title uguale a nuovo



Metodi per il contenuto degli elementi

Metodi per leggere/scrivere il contenuto degli elementi HTML:

- **text**: legge/scrive il contenuto testuale di un elemento
- **html**: legge/scrive il contenuto HTML di un elemento
- **val**: legge/scrive il valore dei campi delle form

Se il metodo è senza argomenti, effettua una lettura (get), se invece ha un argomento, effettua una scrittura (set)

Esiste anche il metodo **attr** per leggere/scrivere i valori degli attributi



Metodi per il contenuto degli elementi

Esempi di set:

- `$(“p”).text(“Nuovo testo dei paragrafi”);`
- `$(“p”).html(“Nuovo testo paragrafi con tag”);`
- `$(“#cognome”).val(“Rossi”)`
- `$(“img”).attr(“width”, “800”);`
- `$(“img”).attr({width: “800”, height: “600”});`

Esempi di get:

- `var testo = $("h1").text();`
- `var codiceHTML = $("p").html();`
- `var cognomeInseritoDaUtente = $("#cognome").val()`
- `var valoreAttributoWidth = $("img").attr("width");`



Metodi per CSS

Il metodo **css** serve per leggere/scrivere le proprietà CSS degli elementi HTML

Esempi:

- `$("#myId).css("background-color")`
legge il valore della proprietà CSS background-color dell'elemento con id myId
- `$("#myId).css("background-color", "red")`
setta a "red" il valore della proprietà CSS background-color dell'elemento con id myId



Metodi per CSS

Altri metodi legati all'uso dei CSS:

- addClass()
- removeClass()
- toggleClass()
- width()
- height()
- innerWidth()
- innerHeight()
- outerWidth()
- outerHeight()



Metodi per navigare il DOM

Metodi che restituiscono il padre o i predecessori dell'elemento selezionato:

- `parent()`
- `parents()`
- `parentsUntil()`

Metodi che restituiscono i figli o i discendenti dell'elemento selezionato:

- `children()`
- `find()`



Metodi per navigare il DOM

Metodi che restituiscono i fratelli dell'elemento selezionato:

- `siblings()`
- `next()`
- `nextAll()`
- `nextUntil()`
- `prev()`
- `prevAll()`
- `prevUntil()`

Altri metodi di navigazione:

- `first()`
- `last()`
- `eq()`
- `filter()`
- `not()`



Metodi per effetti di animazione

- JQuery prevede molti metodi che realizzano effetti di animazione
- Il metodo animate permette di effettuare una animazione di alcune proprietà CSS degli elementi a cui è applicato
- Esempio:

```
$("button").click(function(){
    $("#mioId").animate({left: '300px'});
});
```

- Questo metodo sposta l'elemento con id mioId verso destra, finché la proprietà left non raggiunge il valore di 300px



Passaggio di funzioni come argomenti

- Nel precedente esempio, si noti il passaggio di una funzione come argomento di un'altra funzione
- Questa è una caratteristica di JavaScript: le funzioni possono essere passate come argomento di una funzione, e possono anche essere restituite come risultato di una funzione
- Semplice sempio:

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

- JQuery usa pesantemente questa caratteristica di JavaScript



Metodi per effetti di animazione

Metodo	Descrizione
animate()	Esegue una animazione sugli elementi selezionati
clearQueue()	Cancella i metodi in coda sugli elementi selezionati
delay()	Setta un ritardo per le funzioni in coda sugli elementi selezionati
dequeue()	Toglie la prossima funzione dalla coda e la esegue
fadeIn()	Esegue il fade-in (assolvenza) degli elementi selezionati
fadeOut()	Esegue il fade-out (dissolvenza) degli elementi selezionati
fadeTo()	Setta l'opacità del fade-in e fade-out degli elementi selezionati
fadeToggle()	Altera l'esecuzione dei metodi fadeIn() e fadeOut()



Metodi per effetti di animazione

Metodo	Descrizione
finish()	Interrompe, rimuove e completa tutte le animazioni in coda per gli elementi selezionati
hide()	Nasconde gli elementi selezionati
queue()	Mostra la coda dei metodi sugli elementi selezionati
show()	Mostra gli elementi selezionati
slideDown()	Mostra (scorrendo) gli elementi selezionati
slideToggle()	Altera l'esecuzione dei metodi slideUp() e slideDown()
slideUp()	Nasconde (scorrendo) gli elementi selezionati
stop()	Interrompe l'animazione corrente sugli elementi selezionati
toggle()	Altera l'esecuzione dei metodi hide() e show()



Gestione degli eventi

- La maggior parte degli eventi previsti nel DOM hanno un corrispondente metodo JQuery
- Si può eseguire una funzione (**event handler**) in corrispondenza dell'evento, passando tale funzione come argomento del metodo JQuery corrispondente
- Esempio:

```
$("button").click(function(){  
    // codice per gestire il click sul bottone  
});
```



Event handler multipli

La funzione `on` permette di associare più event handler allo stesso elemento

Esempio:

```
$("li").on({
    mouseenter: function(){
        $(this).css("background-color", "lightblue");
    },
    mouseleave: function(){
        $(this).css("background-color", "lightred");
    },
    click: function(){
        $(this).css("background-color", "lightgreen");
    }
});
```



Evento (document).ready

Molto spesso tutte le funzioni JQuery dello script vengono incapsulate dentro ad un evento “document ready”:

```
$(document).ready(function(){  
    ...  
    // qui ci sono le funzioni jQuery  
    ...  
});
```

- L'evento (document).ready avviene al termine del caricamento del documento da parte del browser
- In questo modo si è sicuri che il codice JQuery verrà eseguito solo dopo la fine del caricamento, evitando così possibili malfunzionamenti
- Inoltre in questo modo diventa possibile mettere gli script nell'intestazione (head) del documento HTML



Evento (document).ready

Esempio:

```
$(document).ready(function(){
    $("button").click(function(){
        $("#myId").css("background-color", "red");
    });
});
```

Notare ancora una volta il passaggio di funzioni come argomento di altre funzioni (anche in modo annidato)



JQuery e AJAX

JQuery rende molto semplice l'attivazione di funzionalità AJAX

Esempio:

```
$("button").click(function(){
    $("#myDiv").load("xyz.htm");
});
```

- la funzione load permette il caricamento (asincrono) della risorsa xyz.htm
- Al verificarsi del click su qualsiasi elemento button, il contenuto di xyz.htm viene caricato nell'elemento con id myDiv



Funzione load

La funzione load può avere, come secondo argomento, una funzione di callback, che viene eseguita al termine del caricamento della risorsa

Esempio:

```
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#myDiv").load("xyx.htm", function(responseTxt, statusTxt, xhr){
            if(statusTxt == "success") alert("Caricamento terminato");
            if(statusTxt == "error") alert("Errore " + xhr.status + ":" +
                xhr.statusText);
        });
    });
});
</script>
```



Metodi get e post

Metodi get e post:

- `$.get(url, callback)`
 - Invia (in modo asincrono) una richiesta http di tipo get al server
- `$.post(url, callback)`
 - Invia (in modo asincrono) una richiesta http di tipo post al server

Esempio get:

```
$( "button" ).click(function(){  
    $.get("programma-server.asp", function(data, status){  
        alert("Dati ricevuti: " + data + "\nStatus: " +  
        status);  
    });  
});
```



Metodi get e post

Esempio di uso della funzione post:

```
$( "button" ).click(function(){
    $.post("programma-server.asp",
    {
        nome: "Mario Rossi",
        matricola: "01234567"
    },
    function(data, status){
        alert("Dati ricevuti: " + data + "\nStatus: " + status);
    });
});
```



Riferimenti

- <http://api.jquery.com/>
- <https://learn.jquery.com/>
- <http://www.w3schools.com/jquery/>
- <http://www.html.it/guide/guida-jquery/>



XML

Parte 1: **Introduzione a XML**

Luigi Dragone, Riccardo Rosati

Introduzione a XML

- Linguaggi di marcatura
 - SGML, HTML ed XML
 - Forma di un documento XML
 - Rappresentazione di caratteri
 - Elementi, Attributi, Entità
 - Sezioni CDATA
 - Commenti, Istruzioni di processamento
-

Requisiti per la rappresentazione delle informazioni sul Web

Serve un formalismo *più flessibile*:

- separazione tra:
 - *contenuto*
 - *presentazione*
 - *navigazione*
- definizione di *domini* o contesti
- indipendenza dalla piattaforma (*media*) e supporto multilingue

XML = eXtensible Markup Language

Esempio

```
<bibliografia>
<pubblicazione id="Ullm82" tipo="libro">
    <titolo>Principles of Database Systems</titolo>
    <autore>Jeffrey D. Ullman</autore>
    <editore>Computer Science Press</editore>
    <anno>1982</anno>
    <commento>Testo storico sulle basi di dati. ... </commento>
</pubblicazione>
```

```
<pubblicazione id="MiSu99" tipo="in-atto-conferenza">
  <titolo>Type Inference for Queries ...</titolo>
  <autore>Tova Milo</autore>
  <autore>Dan Suciu</autore>
  <apparsoin>Proc. of PODS '99</apparsoin>
  <anno>1999</anno>
  <cita pubblicazioni="Ullm89 AbHV95 ...">
    <commento>Fornisce tecniche per l'inferenza di tipo ...
  </commento>
</pubblicazione>
</bibliografia>
```

Marcatura di documenti

Che cosa è una *marcatura*?

- storicamente: annotazioni in un testo che descrivono al tipografo come stampare o comporre una parte del testo
- oggi: qualsiasi tipo di codice inserito in un testo in forma elettronica

Marcatura (o etichettatura) è un qualcosa che permette di rendere esplicita un'interpretazione di un testo.

Linguaggi di marcatura

Un linguaggio di marcatura è un insieme di *convenzioni per la marcatura* di testi.

Deve specificare:

- quali marcature sono permesse
- quali marcature sono obbligatorie
- come vanno composte le marcature
- come distinguere tra marcatura e testo

Può inoltre specificare il significato della marcatura.

Tipi di marcatura

Si distinguono due tipi di marcatura:

- marcatura procedurale
 - descrive come processare il documento
 - Esempi: PostScript, PDF, RTF, formato MS Word
 - marcatura *descrittiva*
 - descrive la struttura logica del documento
 - Esempi: HTML, SGML, XML
-

Il linguaggio XML

XML = eXtensible Markup Language

- linguaggio di marcatura descrittiva
 - in teoria, naturale successore di HTML come linguaggio per il Web (cioè come linguaggio per la specifica di pagine Web)
 - più espressivo e flessibile (eXtensible)
 - più complicato
 - in pratica, l'idea di rimpiazzare HTML con XML (che alla fine degli anni '90 era nei piani del World Wide Web Consortium) è stata abbandonata (ed è stato invece rilasciato HTML5 nel 2014)
 - XML si è imposto come standard de facto per lo scambio di *informazioni semi-strutturate*
-

Tipologie di informazioni

- fortemente strutturare: classi Java, schema base dati relazionale
- destrutturate: testo libero o testo formattato
- semi-strutturate: struttura dati flessibile con porzioni destrutturate

XML si presta ad essere utilizzato anche per informazioni fortemente strutturate, con alcune limitazioni.

Le origini di XML

1969

Charles Goldfarb (IBM) dirige lo sviluppo di *GML*

1974

Charles Goldfarb inventa *SGML*, il padre dei linguaggi di marcatura

1986

SGML diventa uno *standard ISO*

(ISO 8879 "Information Processing - Text and Office Systems - Standard Generalized Markup Language")

1989

Tim-Berners Lee (CERN di Ginevra) inventa *HTML*

1995

Fondazione del World Wide Web Consortium (*W3C*)

1996

Inizio dello sviluppo di *XML* presso il W3C

1998

XML 1.0 diventa una *raccomandazione W3C*
(uno standard di fatto)

1996-oggi

Sviluppo di *standard associati* ad XML
(coordinato da W3C)

2000-oggi

Promulgazione di varie *edizioni* (versioni) di *XML 1.0*

2002

Promulgazione di *XML 1.1*

2006

La seconda edizione di *XML 1.1* diventa una *raccomandazione W3C*

2008

Promulgazione della quinta edizione di *XML 1.0*

SGML - Standard Generalized Markup Language

- È il padre degli attuali linguaggi di marcatura
 - È un linguaggio che permette di definire linguaggi di marcatura (è un *metalinguaggio*):
 - estremamente espressivo e configurabile (troppo)
 - alta espressività rende il processamento complicato
 - utilizzato in grossi progetti di documentazione
 - non studiato espressamente per il Web
 - Manca di alcune caratteristiche fondamentali per il Web:
 - gestione dei link
 - gestione del conflitto sui nomi delle etichette
 - tutti i documenti devono essere ``validi''
(oltre a essere ``ben formati'')
-

HTML - HyperText Markup Language

- è un linguaggio di marcatura
 - definito in termini di SGML
 - insieme di etichette prefissato (RIGIDO)
 - marcatura non denota il significato
 - studiato espressamente per il Web
 - collegamenti ipertestuali
 - immagini
-

XML = (SGML--) + HTML

Obiettivi di sviluppo di XML:

1. XML deve essere usabile direttamente sul Web
 2. XML deve essere compatibile con SGML
 3. deve essere semplice progettare programmi che processano documenti XML
 4. documenti XML devono essere leggibili da umani e sufficientemente chiari
 5. documenti XML devono essere facili da creare
 6. la specifica di XML deve essere formale e concisa
 7. la compattezza nella marcatura è di poca importanza
-

Caratteristiche dei documenti XML

- marcatura *descrittiva* e non procedurale (descrive la struttura logica)
- marcatura è dettata dalla struttura logica del documento
- insieme di etichette può cambiare in base l'applicazione
- viene usato il concetto di *tipo di documento*
 - specificato attraverso una *Document Type Declaration* o *DTD* (è parte dello standard XML)
 - permette di imporre al documento una certa struttura (ovvero come si compongono le sue parti)
 - le parti sono delimitate dalla marcatura

DTD: specifica la struttura della marcatura ammessa

Specifiche aggiuntive

Si rendono necessarie per rendere XML funzionale sul Web

- presentazione del documento (fogli di stile): XSL (XSLT), CSS, FOP
 - significato della marcatura
 - collegamenti ipertestuali : XPath, XLink, XPointer
 - semantica : RDF, OIL, DAML
 - meccanismi più flessibili per la specifica della struttura: XML Schema, XML-Data, DDML
 - linguaggi di interrogazione per XML : XPath, XQuery, XML-QL, XSL
 - supporto diverse tipologie di dispositivo: XHTML, WML, VoiceML, XForms
 - sistemi di cooperazione applicativa (web service): SOAP, WSDL, UDDI
-

Linguaggi di marcatura a confronto

SGML

è un linguaggio molto espressivo per la definizione di linguaggi di marcatura (è un metalinguaggio)

HTML

è un linguaggio di marcatura definito in termini di SGML, ovvero è specificato da una DTD SGML

XML

è sia un linguaggio di marcatura, sia un linguaggio per la definizione di linguaggi di marcatura

Con XML si considera tutto l'insieme di standard associati

Forma di un documento XML

Un documento XML è costituito da:

- un *prologo* (opzionale), costituito da
 - una dichiarazione XML
 - una DTD
- un'*istanza di documento*, contenente
 - testo libero
 - elementi delimitati da etichette
 - attributi associati ad elementi
 - entità
 - sezioni CDATA
 - istruzioni di processamento
 - commenti

Codifica dei caratteri

XML usa la codifica dei caratteri *Unicode* (UTF-16)

- è un codice a 16 bit (65536 simboli)
- è sufficiente a rappresentare i caratteri di tutte le lingue

- ogni carattere è rappresentato da al più quattro cifre esadecimali
Es. ꀟ
- i primi 256 caratteri sono il codice ASCII (UTF-8)
Es.
 (linefeed), (blank), N (N), ...

Un processore XML:

- deve riconoscere almeno i codici Unicode ed ASCII
 - può riconoscere altri codici
-

Documenti XML ben formati

Un documento che rispetta la specifica di XML è detto *ben formato*:

- costituito da un *insieme di elementi* annidati
- un *elemento* è costituito da una coppia di etichette di apertura e di chiusura e da tutto quello che racchiudono

```
<autore>Jeffrey D. Ullman</autore>
<autore><nome>Jeffrey D.</nome> ... </autore>
  ○ nome dell'elemento: autore
  ○ etichetta di apertura dell'elemento: <autore>
  ○ etichetta di chiusura dell'elemento: </autore>
  ○ contenuto dell'elemento:
    Jeffrey D. Ullman
    <nome>Jeffrey D.</nome>
```

Documenti XML ben formati (2)

Il *contenuto* di un elemento è costituito da:

- altri elementi (detti *figli*), delimitati da coppie di etichette
- testo libero (non contenente marcatura), detto #PCDATA
- riferimenti ad entità

```
<esempio>
  Qui inizia il contenuto di un elemento
  <figlio> questo e` un elemento figlio </figlio>
  testo libero con riferimento all'entita` &amp;
  <figlio> un altro elemento figlio </figlio>
</esempio>
```

Documenti XML ben formati (3)

- le etichette devono essere *annidate correttamente*
Esempio di documento non ben formato:

```
<texto>
  <bold><italic>XML</bold></italic>
```

```
</testo>
```

- ci deve essere *un solo elemento radice*

Esempio di documento con più radici:

```
<bibliografia>
  <pubbl><autore>Ullman</autore> ...</pubbl>
  <pubbl><autore>Floyd</autore> ... </pubbl>
  Adesso è ben formato
</bibliografia>
```

Elementi vuoti

Il contenuto di un elemento può essere vuoto.

Due modi di denotare un *elemento vuoto*:

- coppia di etichette di apertura e chiusura: <vuoto></vuoto>
- etichetta di elemento vuoto: <vuoto/>

```
<p></p>
<h1>Bibliografia</h1>
<ul>
  <li><b>Ullman</b> ...
  <li><b>Floyd</b> ...
</ul>
<hr/>
```

Uso di elementi vuoti

Non sono inutili in quanto aggiungono informazione al documento:

- attraverso attributi associati all'elemento

```
<IMG src="colosseo.gif"
      align="left"
      height="50"
      width="30"/>
```

- attraverso la presenza stessa dell'elemento

```
<BR/>
```

Nomi di elemento

XML è *case-sensitive* - esempio non ben formato:

```
<elenco-clienti>
  <cliente><codice>CC128</codice> ... </CLIENTE>
</Elenco-Clienti>
```

Errore comune: dimenticare "/" nell'etichetta di chiusura - es:

```
<elenco-clienti>
  <cliente><codice>CC128</codice> ... <cliente>
</elenco-clienti>
```

Nomi di elemento (2)

Possono contenere solo: lettere, cifre, -, _, .,:.

Carattere ":" usato solo per separare *namespace*.

Nomi che iniziano con XML, xml, xML, ...sono riservati.

```
<nomiPermessi>
  <xsl:template/>
  <Nome_elemento_lungo/>
  <Altro-nome-lungo/>
  <nome.con.punti/>
  <a1233-231-231/>
  <_12/>
</nomiPermessi>

<nomi+Vietati>
  <questiNO@#$%^() +?* ;$=/>
  <Un;nome*2/>
  <XmL-riservato/>
  <8-inizia-con-cifra/>
  <niente spazi/>
  <riservati<&>>
</nomi+Vietati>
```

Caratteri riservati

Il testo non può contenere i caratteri riservati per la marcatura

- carattere "<" (denota l'inizio di un'etichetta)
- carattere "&" (per le entità - più avanti)

Si usano

"<" per "<"
"&" per "&"

```
<editore>Wiley & sons</editore>
<disequazione>x*y &lt; z</disequazione>
```

I caratteri ">", "'''", e "''' possono essere sostituiti da
">", """ e "'".

Attributi

Un elemento può avere nessuno, uno o più *attributi*.

- un attributo ha un *nome* ed associa una *proprietà* ad un elemento
 - i caratteri ammessi nei nomi sono gli stessi che per gli elementi
 - gli attributi vengono specificati nell'etichetta iniziale attraverso coppie *attributo="valore"*
 - il valore deve essere delimitato da una coppia di apici singoli o doppi
 - un elemento non può avere due attributi con lo stesso nome
-

Attributi

```
<attributi-ben-formati>
  <elemento _ok="yes"/>
  <un attr="il suo valore"/>
  <molti primo="1" secondo='2' terzo="333"/>
  <apici-doppi-o-singoli
    doppi="John's"
    singoli='Stampa: "Ciao Mondo!" '/>
</attributi-ben-formati>
```

N.B. In HTML gli apici intorno al valore possono mancare
in XML gli *apici sono obbligatori*.

Attributi - Esempi mal formati

```
<attributi-mal-formati>
  <carattere-non-ammesso a*b="23432"/>
  <separatori-diversi      valore="12' />
  <cambia-separatore      valore="aa"aa"/>
  <cambia-separatore      valore='bb'bb' />
  <parola-riservata       XML-ID="xml234"/>
</attributi-mal-formati>
```

Uso di attributi ed elementi

Per memorizzare dati possiamo usare sia elementi che attributi.

```
<impiegato>
  <nome>A. Rossi</nome>
  <ddn>1/1/1950</ddn>
</impiegato>

<impiegato ddn="1/1/1950">
  <nome>A. Rossi</nome>
</impiegato>
```

La scelta dipende dall'applicazione:

- non si può forzare #PCDATA a non essere vuoto

- gli attributi non si possono strutturare
 - si possono enumerare i valori ammessi per gli attributi
-

Entità

- un'entità è una parte di documento a cui si è dato un *nome* (una specie di macro)
- può essere
 - interna: definita all'interno del documento (o del DTD)
 - esterna: memorizzata su un altro file
- il testo può contenere riferimenti ad entità: &nome-entita;
- quando viene processato il documento, i riferimenti alle entità devono essere espansi
- entità predefinite: <, &, >, ", '

Vedremo più avanti come definire nuove entità.

Sezioni CDATA

- servono ad includere testo contenente caratteri riservati
 - possono comparire ovunque può comparire testo libero
 - tutti i caratteri riservati perdono il loro significato speciale
 - "<" e "&" non vanno sostituiti con "<" e "&"
 - le sequenze <elem>, </elem> e <elem/> non vengono interpretate come marche
 - le sezioni CDATA non possono essere annidate
 - racchiuse tra "<![CDATA[/" e "]]>"
 - una sezione CDATA non può contenere la stringa "]]>"
-

Sezioni CDATA - Esempi

Questo lucido in formato XML

```
<lucido>
  <titolo>Sezioni CDATA - Esempi</titolo>
  Questo lucido in formato XML
  <esempio>
    <! [CDATA[
      <lucido>
        <titolo>Sezioni CDATA - Esempi</titolo>
        Questo lucido in formato XML
        ... caratteri riservati OK: &, <
      </lucido>
    ]]>
  </esempio>
</lucido>
```

Sezioni CDATA - Esempi (2)

La stringa "]]>" non è ammessa

```

<esempio>
<! [CDATA[Un programma C
  if (x[2] + y[i] < 3*z) {
    if (vet[x[i]] > 18) { /* 1 */
      if (a < b) printf("Ciao"); /* 2 */
    }
  }
]]>
sezione CDATA termina su riga 1 e non qui
" a &lt; b" da` errore perche` sono fuori CDATA
</esempio>
```

Al posto di `]]>` bisogna includere `]]>`

Commenti

- servono a escludere una parte di documento dal processamento
 - possono comparire ovunque all'esterno della marcatura
 - un processore XML può o meno rendere disponibili le parti di documento racchiuse tra commenti
 - sono delimitati da `<!-- e -->`
 - possono contenere qualsiasi carattere (inclusi "`<`" e "`&`"), tranne "`-`"
 - non possono essere annidati
-

Commenti - Esempi

```

<bibliografia>
  <!-- voce commentata <pub> ... </pub> -->
  <pub> ... </pub>
  <!-- commento su piu` righe
    <pub> ... </pub>   <- usa "<" 
    <pub> ... </pub>   anche "&" e` ammesso
  -->

  <!-- commento non -- valido -->
  <pub <!-- qui no commento --> id="U11m82"> ... </pub>
  <!-- un commento
    <!-- non puo` contenere commenti annidati -->
  -->
</bibliografia>
```

Commenti in sezioni CDATA

In una sezione CDATA i commenti non vengono riconosciuti come tali

```

<esempio>
Questo e` un <!-- commento -->
<! [CDATA[ Sezione CDATA con "commento"
  <!-- fa parte del testo --> altro testo
]]>
Questo e` di nuovo un <!-- commento -->
```

</esempio>

Istruzioni di processamento (PI)

- permettono di includere nel documento istruzioni da passare alle applicazioni
- fanno parte del prologo (non dei caratteri che compongono il documento)
- devono essere passate alle applicazioni
- hanno la forma <?PITarget ... ?>, dove
 - PITarget denota l'applicazione a cui è diretta la PI
 - ... denota ulteriori dati da passare all'applicazione

Esempio: processamento di uno stylesheet da parte di XSLT

```
<?xml-stylesheet href="mystyle.xsl" type="text/xsl" ?>
```

Dichiarazione XML iniziale

Documenti XML possono (e in realtà dovrebbero) iniziare con una *dichiarazione XML* che specifica la versione di XML utilizzata

```
<?xml version="1.0"?>  
<texto>Questo documento e` conforme alla specifica  
di XML 1.0.</testo>
```

La dichiarazione XML può anche specificare la *codifica dei caratteri* usati:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
```

La codifica dei caratteri specifica il mapping tra i byte che costituiscono la rappresentazione fisica del documento (ad es., file, socket, ...) ed i caratteri.

HTML e XHTML

XHTML è una versione di HTML compatibile con XML:

- i documenti HTML non sono necessariamente ben formati
- i documenti XHTML lo sono

Principali *differenze* tra XHTML e XML

- etichette devono essere annidate correttamente
- tutte le etichette devono essere chiuse
- nomi di elementi ed attributi sono minuscoli
- valori di attributi racchiusi tra apici
- altre differenze dovute alle differenze nei DTD di XML ed SGML

È possibile trasformare un documento HTML in un documento XHTML equivalente, se il documento HTML non è ambiguo.

Riferimenti

Extensible Markup Language (XML) 1.0 (Fifth Edition): <https://www.w3.org/TR/2008/REC-xml-20081126/>

XML

Parte 1: **Introduzione a XML**

Luigi Dragone, Riccardo Rosati

Introduzione a XML

- Linguaggi di marcatura
 - SGML, HTML ed XML
 - Forma di un documento XML
 - Rappresentazione di caratteri
 - Elementi, Attributi, Entità
 - Sezioni CDATA
 - Commenti, Istruzioni di processamento
-

Requisiti per la rappresentazione delle informazioni sul Web

Serve un formalismo *più flessibile*:

- separazione tra:
 - *contenuto*
 - *presentazione*
 - *navigazione*
- definizione di *domini* o contesti
- indipendenza dalla piattaforma (*media*) e supporto multilingue

XML = eXtensible Markup Language

Esempio

```
<bibliografia>
<pubblicazione id="Ullm82" tipo="libro">
    <titolo>Principles of Database Systems</titolo>
    <autore>Jeffrey D. Ullman</autore>
    <editore>Computer Science Press</editore>
    <anno>1982</anno>
    <commento>Testo storico sulle basi di dati. ... </commento>
</pubblicazione>
```

```
<pubblicazione id="MiSu99" tipo="in-atto-conferenza">
  <titolo>Type Inference for Queries ...</titolo>
  <autore>Tova Milo</autore>
  <autore>Dan Suciu</autore>
  <apparsoin>Proc. of PODS '99</apparsoin>
  <anno>1999</anno>
  <cita pubblicazioni="Ullm89 AbHV95 ...">
    <commento>Fornisce tecniche per l'inferenza di tipo ...
  </commento>
</pubblicazione>
</bibliografia>
```

Marcatura di documenti

Che cosa è una *marcatura*?

- storicamente: annotazioni in un testo che descrivono al tipografo come stampare o comporre una parte del testo
- oggi: qualsiasi tipo di codice inserito in un testo in forma elettronica

Marcatura (o etichettatura) è un qualcosa che permette di rendere esplicita un'interpretazione di un testo.

Linguaggi di marcatura

Un linguaggio di marcatura è un insieme di *convenzioni per la marcatura* di testi.

Deve specificare:

- quali marcature sono permesse
- quali marcature sono obbligatorie
- come vanno composte le marcature
- come distinguere tra marcatura e testo

Può inoltre specificare il significato della marcatura.

Tipi di marcatura

Si distinguono due tipi di marcatura:

- marcatura procedurale
 - descrive come processare il documento
 - Esempi: PostScript, PDF, RTF, formato MS Word
 - marcatura *descrittiva*
 - descrive la struttura logica del documento
 - Esempi: HTML, SGML, XML
-

Il linguaggio XML

XML = eXtensible Markup Language

- linguaggio di marcatura descrittiva
 - in teoria, naturale successore di HTML come linguaggio per il Web (cioè come linguaggio per la specifica di pagine Web)
 - più espressivo e flessibile (eXtensible)
 - più complicato
 - in pratica, l'idea di rimpiazzare HTML con XML (che alla fine degli anni '90 era nei piani del World Wide Web Consortium) è stata abbandonata (ed è stato invece rilasciato HTML5 nel 2014)
 - XML si è imposto come standard de facto per lo scambio di *informazioni semi-strutturate*
-

Tipologie di informazioni

- fortemente strutturare: classi Java, schema base dati relazionale
- destrutturate: testo libero o testo formattato
- semi-strutturate: struttura dati flessibile con porzioni destrutturate

XML si presta ad essere utilizzato anche per informazioni fortemente strutturate, con alcune limitazioni.

Le origini di XML

1969

Charles Goldfarb (IBM) dirige lo sviluppo di *GML*

1974

Charles Goldfarb inventa *SGML*, il padre dei linguaggi di marcatura

1986

SGML diventa uno *standard ISO*

(ISO 8879 "Information Processing - Text and Office Systems - Standard Generalized Markup Language")

1989

Tim-Berners Lee (CERN di Ginevra) inventa *HTML*

1995

Fondazione del World Wide Web Consortium (*W3C*)

1996

Inizio dello sviluppo di *XML* presso il W3C

1998

XML 1.0 diventa una *raccomandazione W3C*
(uno standard di fatto)

1996-oggi

Sviluppo di *standard associati* ad XML
(coordinato da W3C)

2000-oggi

Promulgazione di varie *edizioni* (versioni) di *XML 1.0*

2002

Promulgazione di *XML 1.1*

2006

La seconda edizione di *XML 1.1* diventa una *raccomandazione W3C*

2008

Promulgazione della quinta edizione di *XML 1.0*

SGML - Standard Generalized Markup Language

- È il padre degli attuali linguaggi di marcatura
 - È un linguaggio che permette di definire linguaggi di marcatura (è un *metalinguaggio*):
 - estremamente espressivo e configurabile (troppo)
 - alta espressività rende il processamento complicato
 - utilizzato in grossi progetti di documentazione
 - non studiato espressamente per il Web
 - Manca di alcune caratteristiche fondamentali per il Web:
 - gestione dei link
 - gestione del conflitto sui nomi delle etichette
 - tutti i documenti devono essere ``validi''
(oltre a essere ``ben formati'')
-

HTML - HyperText Markup Language

- è un linguaggio di marcatura
 - definito in termini di SGML
 - insieme di etichette prefissato (RIGIDO)
 - marcatura non denota il significato
 - studiato espressamente per il Web
 - collegamenti ipertestuali
 - immagini
-

XML = (SGML--) + HTML

Obiettivi di sviluppo di XML:

1. XML deve essere usabile direttamente sul Web
 2. XML deve essere compatibile con SGML
 3. deve essere semplice progettare programmi che processano documenti XML
 4. documenti XML devono essere leggibili da umani e sufficientemente chiari
 5. documenti XML devono essere facili da creare
 6. la specifica di XML deve essere formale e concisa
 7. la compattezza nella marcatura è di poca importanza
-

Caratteristiche dei documenti XML

- marcatura *descrittiva* e non procedurale (descrive la struttura logica)
- marcatura è dettata dalla struttura logica del documento
- insieme di etichette può cambiare in base l'applicazione
- viene usato il concetto di *tipo di documento*
 - specificato attraverso una *Document Type Declaration* o *DTD* (è parte dello standard XML)
 - permette di imporre al documento una certa struttura (ovvero come si compongono le sue parti)
 - le parti sono delimitate dalla marcatura

DTD: specifica la struttura della marcatura ammessa

Specifiche aggiuntive

Si rendono necessarie per rendere XML funzionale sul Web

- presentazione del documento (fogli di stile): XSL (XSLT), CSS, FOP
 - significato della marcatura
 - collegamenti ipertestuali : XPath, XLink, XPointer
 - semantica : RDF, OIL, DAML
 - meccanismi più flessibili per la specifica della struttura: XML Schema, XML-Data, DDML
 - linguaggi di interrogazione per XML : XPath, XQuery, XML-QL, XSL
 - supporto diverse tipologie di dispositivo: XHTML, WML, VoiceML, XForms
 - sistemi di cooperazione applicativa (web service): SOAP, WSDL, UDDI
-

Linguaggi di marcatura a confronto

SGML

è un linguaggio molto espressivo per la definizione di linguaggi di marcatura (è un metalinguaggio)

HTML

è un linguaggio di marcatura definito in termini di SGML, ovvero è specificato da una DTD SGML

XML

è sia un linguaggio di marcatura, sia un linguaggio per la definizione di linguaggi di marcatura

Con XML si considera tutto l'insieme di standard associati

Forma di un documento XML

Un documento XML è costituito da:

- un *prologo* (opzionale), costituito da
 - una dichiarazione XML
 - una DTD
- un'*istanza di documento*, contenente
 - testo libero
 - elementi delimitati da etichette
 - attributi associati ad elementi
 - entità
 - sezioni CDATA
 - istruzioni di processamento
 - commenti

Codifica dei caratteri

XML usa la codifica dei caratteri *Unicode* (UTF-16)

- è un codice a 16 bit (65536 simboli)
- è sufficiente a rappresentare i caratteri di tutte le lingue

- ogni carattere è rappresentato da al più quattro cifre esadecimali
Es. ꀟ
- i primi 256 caratteri sono il codice ASCII (UTF-8)
Es.
 (linefeed), (blank), N (N), ...

Un processore XML:

- deve riconoscere almeno i codici Unicode ed ASCII
 - può riconoscere altri codici
-

Documenti XML ben formati

Un documento che rispetta la specifica di XML è detto *ben formato*:

- costituito da un *insieme di elementi* annidati
- un *elemento* è costituito da una coppia di etichette di apertura e di chiusura e da tutto quello che racchiudono

```
<autore>Jeffrey D. Ullman</autore>
<autore><nome>Jeffrey D.</nome> ... </autore>
  ○ nome dell'elemento: autore
  ○ etichetta di apertura dell'elemento: <autore>
  ○ etichetta di chiusura dell'elemento: </autore>
  ○ contenuto dell'elemento:
    Jeffrey D. Ullman
    <nome>Jeffrey D.</nome>
```

Documenti XML ben formati (2)

Il *contenuto* di un elemento è costituito da:

- altri elementi (detti *figli*), delimitati da coppie di etichette
- testo libero (non contenente marcatura), detto #PCDATA
- riferimenti ad entità

```
<esempio>
  Qui inizia il contenuto di un elemento
  <figlio> questo e` un elemento figlio </figlio>
  testo libero con riferimento all'entita` &amp;
  <figlio> un altro elemento figlio </figlio>
</esempio>
```

Documenti XML ben formati (3)

- le etichette devono essere *annidate correttamente*
Esempio di documento non ben formato:

```
<texto>
  <bold><italic>XML</bold></italic>
```

```
</testo>
```

- ci deve essere *un solo elemento radice*

Esempio di documento con più radici:

```
<bibliografia>
  <pubbl><autore>Ullman</autore> ...</pubbl>
  <pubbl><autore>Floyd</autore> ... </pubbl>
  Adesso è ben formato
</bibliografia>
```

Elementi vuoti

Il contenuto di un elemento può essere vuoto.

Due modi di denotare un *elemento vuoto*:

- coppia di etichette di apertura e chiusura: <vuoto></vuoto>
- etichetta di elemento vuoto: <vuoto/>

```
<p></p>
<h1>Bibliografia</h1>
<ul>
  <li><b>Ullman</b> ...
  <li><b>Floyd</b> ...
</ul>
<hr/>
```

Uso di elementi vuoti

Non sono inutili in quanto aggiungono informazione al documento:

- attraverso attributi associati all'elemento

```
<IMG src="colosseo.gif"
      align="left"
      height="50"
      width="30"/>
```

- attraverso la presenza stessa dell'elemento

```
<BR/>
```

Nomi di elemento

XML è *case-sensitive* - esempio non ben formato:

```
<elenco-clienti>
  <cliente><codice>CC128</codice> ... </CLIENTE>
</Elenco-Clienti>
```

Errore comune: dimenticare "/" nell'etichetta di chiusura - es:

```
<elenco-clienti>
  <cliente><codice>CC128</codice> ... <cliente>
</elenco-clienti>
```

Nomi di elemento (2)

Possono contenere solo: lettere, cifre, -, _, .,:.

Carattere ":" usato solo per separare *namespace*.

Nomi che iniziano con XML, xml, xML, ...sono riservati.

```
<nomiPermessi>
  <xsl:template/>
  <Nome_elemento_lungo/>
  <Altro-nome-lungo/>
  <nome.con.punti/>
  <a1233-231-231/>
  <_12/>
</nomiPermessi>

<nomi+Vietati>
  <questiNO@#$%^() +?* ;$=/>
  <Un;nome*2/>
  <XmL-riservato/>
  <8-inizia-con-cifra/>
  <niente spazi/>
  <riservati<&>>
</nomi+Vietati>
```

Caratteri riservati

Il testo non può contenere i caratteri riservati per la marcatura

- carattere "<" (denota l'inizio di un'etichetta)
- carattere "&" (per le entità - più avanti)

Si usano

"<" per "<"
"&" per "&"

```
<editore>Wiley & sons</editore>
<disequazione>x*y &lt; z</disequazione>
```

I caratteri ">", "'''", e "''' possono essere sostituiti da
">", """ e "'".

Attributi

Un elemento può avere nessuno, uno o più *attributi*.

- un attributo ha un *nome* ed associa una *proprietà* ad un elemento
 - i caratteri ammessi nei nomi sono gli stessi che per gli elementi
 - gli attributi vengono specificati nell'etichetta iniziale attraverso coppie *attributo="valore"*
 - il valore deve essere delimitato da una coppia di apici singoli o doppi
 - un elemento non può avere due attributi con lo stesso nome
-

Attributi

```
<attributi-ben-formati>
  <elemento _ok="yes"/>
  <un attr="il suo valore"/>
  <molti primo="1" secondo='2' terzo="333"/>
  <apici-doppi-o-singoli
    doppi="John's"
    singoli='Stampa: "Ciao Mondo!" '/>
</attributi-ben-formati>
```

N.B. In HTML gli apici intorno al valore possono mancare
in XML gli *apici sono obbligatori*.

Attributi - Esempi mal formati

```
<attributi-mal-formati>
  <carattere-non-ammesso a*b="23432"/>
  <separatori-diversi      valore="12' />
  <cambia-separatore      valore="aa"aa"/>
  <cambia-separatore      valore='bb'bb' />
  <parola-riservata       XML-ID="xml234"/>
</attributi-mal-formati>
```

Uso di attributi ed elementi

Per memorizzare dati possiamo usare sia elementi che attributi.

```
<impiegato>
  <nome>A. Rossi</nome>
  <ddn>1/1/1950</ddn>
</impiegato>

<impiegato ddn="1/1/1950">
  <nome>A. Rossi</nome>
</impiegato>
```

La scelta dipende dall'applicazione:

- non si può forzare #PCDATA a non essere vuoto

- gli attributi non si possono strutturare
 - si possono enumerare i valori ammessi per gli attributi
-

Entità

- un'entità è una parte di documento a cui si è dato un *nome* (una specie di macro)
- può essere
 - interna: definita all'interno del documento (o del DTD)
 - esterna: memorizzata su un altro file
- il testo può contenere riferimenti ad entità: &nome-entita;
- quando viene processato il documento, i riferimenti alle entità devono essere espansi
- entità predefinite: <, &, >, ", '

Vedremo più avanti come definire nuove entità.

Sezioni CDATA

- servono ad includere testo contenente caratteri riservati
 - possono comparire ovunque può comparire testo libero
 - tutti i caratteri riservati perdono il loro significato speciale
 - "<" e "&" non vanno sostituiti con "<" e "&"
 - le sequenze <elem>, </elem> e <elem/> non vengono interpretate come marche
 - le sezioni CDATA non possono essere annidate
 - racchiuse tra "<![CDATA[/" e "]]>"
 - una sezione CDATA non può contenere la stringa "]]>"
-

Sezioni CDATA - Esempi

Questo lucido in formato XML

```
<lucido>
  <titolo>Sezioni CDATA - Esempi</titolo>
  Questo lucido in formato XML
  <esempio>
    <![CDATA[
      <lucido>
        <titolo>Sezioni CDATA - Esempi</titolo>
        Questo lucido in formato XML
        ... caratteri riservati OK: &, <
      </lucido>
    ]]>
  </esempio>
</lucido>
```

Sezioni CDATA - Esempi (2)

La stringa "]]>" non è ammessa

```

<esempio>
<! [CDATA[Un programma C
  if (x[2] + y[i] < 3*z) {
    if (vet[x[i]] > 18) { /* 1 */
      if (a < b) printf("Ciao"); /* 2 */
    }
  }
]]>
sezione CDATA termina su riga 1 e non qui
" a &lt; b" da` errore perche` sono fuori CDATA
</esempio>
```

Al posto di `]]>` bisogna includere `]]>`

Commenti

- servono a escludere una parte di documento dal processamento
 - possono comparire ovunque all'esterno della marcatura
 - un processore XML può o meno rendere disponibili le parti di documento racchiuse tra commenti
 - sono delimitati da `<!-- e -->`
 - possono contenere qualsiasi carattere (inclusi "`<`" e "`&`"), tranne "`-`"
 - non possono essere annidati
-

Commenti - Esempi

```

<bibliografia>
  <!-- voce commentata <pub> ... </pub> -->
  <pub> ... </pub>
  <!-- commento su piu` righe
    <pub> ... </pub>   <- usa "<" 
    <pub> ... </pub>   anche "&" e` ammesso
  -->

  <!-- commento non -- valido -->
  <pub <!-- qui no commento --> id="U11m82"> ... </pub>
  <!-- un commento
    <!-- non puo` contenere commenti annidati -->
  -->
</bibliografia>
```

Commenti in sezioni CDATA

In una sezione CDATA i commenti non vengono riconosciuti come tali

```

<esempio>
Questo e` un <!-- commento -->
<! [CDATA[ Sezione CDATA con "commento"
  <!-- fa parte del testo --> altro testo
]]>
Questo e` di nuovo un <!-- commento -->
```

</esempio>

Istruzioni di processamento (PI)

- permettono di includere nel documento istruzioni da passare alle applicazioni
- fanno parte del prologo (non dei caratteri che compongono il documento)
- devono essere passate alle applicazioni
- hanno la forma <?PITarget ... ?>, dove
 - PITarget denota l'applicazione a cui è diretta la PI
 - ... denota ulteriori dati da passare all'applicazione

Esempio: processamento di uno stylesheet da parte di XSLT

```
<?xml-stylesheet href="mystyle.xsl" type="text/xsl" ?>
```

Dichiarazione XML iniziale

Documenti XML possono (e in realtà dovrebbero) iniziare con una *dichiarazione XML* che specifica la versione di XML utilizzata

```
<?xml version="1.0"?>  
<texto>Questo documento e` conforme alla specifica  
di XML 1.0.</testo>
```

La dichiarazione XML può anche specificare la *codifica dei caratteri* usati:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
```

La codifica dei caratteri specifica il mapping tra i byte che costituiscono la rappresentazione fisica del documento (ad es., file, socket, ...) ed i caratteri.

HTML e XHTML

XHTML è una versione di HTML compatibile con XML:

- i documenti HTML non sono necessariamente ben formati
- i documenti XHTML lo sono

Principali *differenze* tra XHTML e XML

- etichette devono essere annidate correttamente
- tutte le etichette devono essere chiuse
- nomi di elementi ed attributi sono minuscoli
- valori di attributi racchiusi tra apici
- altre differenze dovute alle differenze nei DTD di XML ed SGML

È possibile trasformare un documento HTML in un documento XHTML equivalente, se il documento HTML non è ambiguo.

Riferimenti

Extensible Markup Language (XML) 1.0 (Fifth Edition): <https://www.w3.org/TR/2008/REC-xml-20081126/>

XML

Parte 2: XML DOM

Luigi Dragone, Riccardo Rosati

DOM: Document Object Model

- Il DOM (Document Object Model) è un modello ad oggetti dei dati contenuti in un documento XML
 - L'API del DOM è un'interfaccia di programmazione applicativa che consente di manipolare opportunamente tali strutture dati
 - Il modello di un documento è assimilabile ad una struttura ad albero
 - Un nodo dell'albero può essere un elemento, un attributo, una sezione #PCDATA o una sezione CDATA del documento
-

Il nucleo DOM

- Interfacce Fondamentali (valide per qualsiasi documento anche non XML, per esempio HTML)
 - Node
 - Document
 - DOMImplementation
 - DocumentFragment
 - NodeList
 - Element
 - Attr
 - CharacterData
 - Text
 - Comments
 - DomException
 - Interfacce Estese (specifiche per i documenti XML)
 - CDATASEction
 - DocumentType
 - Notation
 - EntityReference
 - Entity
 - ProcessingInstruction
-

Interfaccia Node

Il nodo è la struttura fondamentale del DOM, ogni altra struttura è ottenuta come specializzazione dell'interfaccia Node supporta essenzialmente 3 tipi di operazioni:

1. informazioni su un nodo
 2. attraversamento dell'albero
 3. modifica dei nodi
-

Estensioni dell'Interfaccia Node

Interfaccia Document: Estende Node per rappresentare (la radice di) un documento. Un Node non può essere creato se non viene creato un Document

Interfaccia Element: Estende Node per rappresentare un elemento

Interfaccia Attr: Estende Node per rappresentare un attributo

Interfaccia Text : Estende Node per rappresentare una sezione #PCDATA

Interfacce di collegamento

Interfaccia NamedNodeMap: rappresenta una mappa di nodi indicizzata sul nome e consente di accedere ad un nodo dell'insieme attraverso il nome

Interfaccia NodeList: rappresenta una lista di nodi

- int getLength() restituisce il numero dei nodi dell'insieme
 - Node item(int index) restituisce un nodo dato l'indice
-

Interfaccia Node

Il nodo è la struttura fondamentale del DOM, ogni altra struttura è ottenuta come specializzazione dell'interfaccia Node

- Node appendChild(Node newChild) aggiunge un nodo figlio
- NodeList getChildNodes() estrae la lista dei nodi figli
- Node getFirstChild() / Node getLastChild() restituisce il primo/ultimo nodo figlio
- boolean hasChildNodes() verifica la presenza di nodi figli
- NamedNodeMap getAttributes() restituisce la mappa degli attributi
- boolean hasAttributes() verifica la presenza di attributi
- String getNodeName() restituisce il nome del nodo
- short getNodeType() restituisce il tipo del nodo
- Node getNextSibling() / Node getPreviousSibling() restituisce il nodo immediatamente successivo / precedente
- Node getParentNode() restituisce il nodo padre
- Document getOwnerDocument() restituisce il documento contenente il nodo
- void setNodeValue(String nodeValue) / String getNodeValue() imposta /

restituisce il valore del nodo

Interfaccia Document

Estende l'interfaccia Node per rappresentare le caratteristiche tipiche di un documento

- `Element getDocumentElement()` restituisce il root-element del documento
 - `Element getElementById(String elementId)` restituisce l'elemento del documento di cui si è fornito l'ID
 - `Attr createAttribute(String name)` crea un attributo con un dato un nome
 - `Element createElement(String tagName)` crea un elemento con un tag
 - `Text createTextNode(String data)` crea una sezione #PCDATA specificandone il contenuto
-

Interfaccia Element

Estende l'interfaccia Node per rappresentare le caratteristiche tipiche di un elemento

- `String getTagName()` restituisce il tag dell'elemento
 - `NodeList getElementsByTagName(String name)` restituisce i sotto-nodi con un certo tag
 - `String getAttribute(String name)` restituisce il valore di un attributo
 - `Attr getAttributeNode(String name)` restituisce il nodo di un attributo
 - `boolean hasAttribute(String name)` verifica la presenza di un attributo
 - `void setAttribute(String name, String value)` imposta il valore di un attributo
 - `Attr setAttributeNode(Attr newAttr)` aggiunge un attributo
-

Interfaccia Attr

Estende l'interfaccia Node per rappresentare le caratteristiche tipiche di un attributo

- `String getName()` restituisce il nome dell'attributo
 - `Element getOwnerElement()` restituisce l'elemento che contiene l'attributo
 - `boolean getSpecified()` verifica se l'attributo è stato specificato esplicitamente, oppure assume il suo valore di default
 - `String getValue()` restituisce il valore dell'attributo
 - `void setValue(String value)` imposta il valore dell'attributo
-

Interfaccia Text

Estende l'interfaccia Node per rappresentare le caratteristiche tipiche di una sezione #PCDATA

- `void setData(String data)` imposta il contenuto del nodo
 - `String getData()` restituisce il contenuto del nodo
-

Interfaccia NamedNodeMap

Questa interfaccia rappresenta una mappa di nodi indicizzata sul nome e consente di accedere ad un nodo dell'insieme attraverso il nome

- int `getLength()` restituisce il numero dei nodi dell'insieme
 - Node `item(int index)` restituisce un nodo dato l'indice
 - Node `getNamedItem(String name)` restituisce un nodo dato il nome
 - Node `removeNamedItem(String name)` elimina un nodo dall'insieme dato il nome
 - Node `setNamedItem(Node arg)` aggiunge, o sostituisce se presente, un nodo all'insieme
-

Interfaccia NodeList

Questa interfaccia rappresenta una lista di nodi

- int `getLength()` restituisce il numero dei nodi dell'insieme
 - Node `item(int index)` restituisce un nodo dato l'indice
-

Riferimenti

<https://dom.spec.whatwg.org/>

XML

Parte 3: **Fogli di stile per XML: XSL e XSLT**

Luigi Dragone, Riccardo Rosati

Fogli di stile per XML: XSL e XSLT

- XSL ed XSLT
 - Namespace
 - XPath
 - Programmazione in XSLT
 - Ricorsione strutturale
 - Template e regole
 - Ristrutturazione profonda
 - XSL Formatting Objects (Cenni)
-

XSL e XSLT

XSL 1.0: W3C Recommendation del 21/11/2000

XSL 1.1: W3C Recommendation del 5/12/2006

XSLT 1.0: W3C Recommendation del 16/11/1999

XML Stylesheet Language (XSL) è composto di due parti:

1. *XSL Transformations*: linguaggio per la specifica di trasformazioni di documenti XML
2. *XSL Formatting Objects (XSLFO)*: linguaggio per la descrizione della formattazione e disposizione di testo in un documento

XML Stylesheet Language (XSL) utilizza altri due standard:

1. Namespaces: servono per definire il campo di azione degli identificatori
 2. *XPath*: linguaggio per identificare gli elementi di un documento
-

Visualizzazione con XSL

La visualizzazione viene realizzata in due passi:

- usando XSLT si trasforma il documento in un *formato di visualizzazione* (si sostituiscono le etichette con *etichette di visualizzazione*)
- usando i Formatting Objects viene specificato come visualizzare il formato di

visualizzazione.

Attualmente si usa principalmente HTML come formato di visualizzazione.

Collegamento del foglio di stile al documento:

```
<?xml-stylesheet type="text/xsl" href="URL"?>
```

XSL Formatting Objects (XSLFO)

- linguaggio basato su un modello per il layout visuale più sofisticato di quello di (X)HTML+CSS
- pensato non solo per il layout sul web, ma per scopi più generali (ad es. la stampa di un libro)
- si basa su un modello a zone rettangolari che possono contenere testo, spazio, ..., oppure altre zone
- gli elementi dei formatting objects hanno un proprio namespace

```
<xsl:stylesheet  
    xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"  
    xmlns:fo="http://www.w3.org/XSL/Format/1.0">
```

Altri usi di XSLT

Oltre alla visualizzazione dei documenti:

- Scambio di dati in formati diversi da XML ad XML (linguaggio di manipolazione)
- Creazione di reportistica con funzioni di selezione ed aggregazione dei dati (linguaggio di interrogazione XPath)

In pratica XSLT si può considerare un linguaggio di programmazione per documenti (attraverso la struttura ad albero del documento costruita dal parser XML, rappresentata tramite insiemi di nodi [node-set])

Linguaggio *dichiarativo* (funzionale, basato su regole) alternativo a DOM (SAX) per gestire le operazioni sull'albero di sintassi di un documento

Namespace

W3C Recommendation del 14/1/1999

<http://www.w3.org/TR-REC-xml-names/>

Permettono di:

- includere in un documento etichette provenienti da DTD diverse
- evitano che vi siano conflitti quando le DTD usano le stesse etichette (con significati diversi)
`<titolo> accademico`
`<titolo> di una pubblicazione`

Si usa un *namespace* diverso per ogni DTD da integrare

<persona:titolo> accademico

<libro:titolo> di una pubblicazione

Dichiarazioni di namespace

Fatte nell'etichetta iniziale di un elemento usando l'attributo speciale `xmlns:prefisso`:

`<nome-el xmlns:prefisso="URI" ... >`

- *prefisso* è il nome dato al namespace, da anteporre ai nomi che fanno riferimento al namespace
- *URI* ha un ruolo formale (per riconoscere, e non per scaricare il namespace)

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/...">
  <xsl:template match="ATOM">
    <P> <xsl:apply-templates/> </P>
  </xsl:template>
</xsl:stylesheet>
```

Namespace - Osservazioni

- una dichiarazione di namespace ha effetto sull'elemento e tutti i suoi sottoelementi
- si usa mettere le dichiarazioni di namespace nell'etichetta dell'elemento radice
- gli elementi senza *prefisso*: si riferiscono al namespace dell'elemento radice (quello di default)
- il valore dell'attributo `xmlns:prefisso` identifica il namespace e non il prefisso
- l'attributo `xmlns:prefisso` dovrebbe essere dichiarato nella DTD come tutti gli altri attributi

XPath

W3C Recommendation del 16/11/1999

<http://www.w3.org/XPath/>

Si basa sul DOM e sulla struttura ad albero di un documento:

- i *nodi* sono elementi, testo, attributi, ...
- i *figli* di un elemento sono i suoi sotto-elementi ed attributi

Describe come *selezionare (insiemi di) nodi* all'interno di un documento:

- navigazione simile a quella in una struttura di directory
- distinzione tra elementi ed attributi

La manipolazione delle informazioni individuate attraverso XPath può avvenire tramite XSLT o XPointer

Espressioni XPath

- Stringhe (indicate con '' oppure "")
 - Numeri (interi e reali solo in notazione decimale)
 - Percorsi - Location paths (specificati più avanti)
 - Variabili (*\$nome-variabile*)
 - Operatori predefiniti (specificati più avanti)
 - Chiamate a funzioni (*nomefunzione(argomenti)*)
 - Espressioni con parentesi
-

Percorsi-Location Paths

XPath	node-set
libro	ogni el. libro (relativo)
*	ogni el. (relativo)
@prezzo	l'attributo prezzo
/bib	l'elemento radice bib
/bib/libro	ogni libro in bib
/bib//libro	ogni libro in bib a qualsiasi livello
//libro	ogni libro (ad ogni livello)
/bib/libro/@prezzo	l'attributo prezzo di un libro in bib
../libro	elementi libro figli del padre (rel)
capitolo[2]/sezione[1]	figlio 1 sezione del figlio 2 capitolo (rel)
libro[<i>cond</i>]	ogni libro che verifica <i>cond</i>
//libro[@prezzo]	ogni libro che ha un prezzo
//libro[anno]	ogni libro che ha un sottoel. anno
libro articolo	ogni libro o articolo

Operatori e funzioni predefiniti in XPath

- operatori per i dati di tipo numerico (+,-,*,div,mod)
 - operatori di confronto applicabili a numeri stringhe e node-set (>,<,=,>=,<=,!=)
 - operatori di tipo booleano per le condizioni (and, or)
 - funzioni di selezione degli elementi dell'albero del documento
 - funzioni di uso generale
-

Documento XSLT

XSLT ha un proprio namespace con URI:

<http://www.w3.org/1999/XSL/Transform>

Elemento radice: xsl:stylesheet

```
<xsl:stylesheet  
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
    version="1.0">
```

XSLT come linguaggio di trasformazione

Permette di trasformare:

- XML >> XML
- XML >> HTML
- XHTML >> XML
- XML >> testo

Entrambi i documenti vengono visti come *alberi* (XSLT è un linguaggio di trasformazione tra alberi).

Per processare documenti HTML è necessario trasformarli in XHTML (ad es., utilizzando HTMLTidy).

Non permette trasformazioni tra formati generici (PDF, PostScript, RTF, TEX , ...) anche se l'utilizzo di uno specifico processore FO permette di produrre output in tali formati

Ricorsione strutturale

Da un punto di vista astratto, XSLT si basa sulla *ricorsione strutturale* come modello di computazione.

- il documento viene visto come un albero (una struttura induttiva)
- abbiamo un insieme di funzioni che *attraversano l'albero* chiamandosi a vicenda (è una forma ristretta di ricorsione)
- XSLT può accedere a nodi (ovvero elementi) a profondità arbitraria nell'albero

XSLT è un linguaggio di trasformazione molto espressivo.

Ricorsione strutturale - Esempio

```
<DOC>  
    <A>3</A>  
    <A> <B>xxx</B> <C>5</C> </A>  
        <B>4</B>  
</DOC>
```

Vogliamo estrarre da questo documento tutti gli elementi il cui testo è costituito da un numero, cioè ottenere:

```
<RIS><I>3</I> <I>5</I> <I>4</I></RIS>
```

La specifica dell'operazione è ricorsiva:

$$f(<\text{DOC}>cont</\text{DOC}>) = <\text{RIS}>f(cont)</\text{RIS}>$$

```
f(e1 ... en) = f(e1)...f(en)
f(<e>cont</e>) = f(cont)
f(text) = if (isInt(text)) <I>text</I> else StringaVuota
f(StringaVuota) = StringaVuota
```

XSLT - Template e regole

XSLT realizza la ricorsione strutturale attraverso *regole* e *template*:

- una *specifica XSLT* è un insieme di regole template
 - una *regola template* ha un match pattern e un template
 - il *match pattern* è specificato in *XPath* e seleziona un insieme di nodi su cui istanziare il template
 - il *template* è costituito da:
 - testo che viene copiato letteralmente in output
 - istruzioni XSL che copiano dati da input ad output
 - istruzioni XSL che specificano come continuare il processamento
-

Applicazione delle regole

- l'applicazione delle regole parte dall'elemento radice ed è guidata dalla struttura del documento e dai match pattern
 - il processore XSL confronta ogni nodo selezionato con i match pattern delle regole
 - quando un nodo corrisponde ad un match pattern viene applicato il corrispondente template
 - il template può anche specificare di continuare il processamento selezionando nuovi nodi (normalmente i figli del nodo corrente)
 - si possono associare delle priorità alle regole, per evitare ambiguità nella selezione della regola da applicare
-

XSLT - Esempio

Estrai i titoli di libri ed articoli dall'insieme di voci bibliografiche

```
<xsl:template match="/bib//(libro|art)/tit">
  <titolo>
    <xsl:value-of select=". "/>
  </titolo>
</xsl:template>
<xsl:template match="* | /">
  <xsl:apply-templates/>
</xsl:template>
```

XSLT mette a disposizione un insieme di elementi ed attributi con significato particolare per selezionare nodi e generare output.

Elementi ed attributi di XSLT

1. Elemento iniziale: `<xsl:stylesheet>`

2. Elementi di primo livello (Es. output)
 3. Elementi template
 4. Istruzioni XSLT
-

Elemento iniziale

Consente di specificare il namespace, la versione ed altri attributi dello stylesheet

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL  
/Transform">  
  
...  
</xsl:stylesheet>
```

Elemento xsl:output

Output è un elemento di primo livello: attraverso l'attributo method di xsl:output è possibile specificare la *forma dell'output*:

- xml: come documento XML ben formato
- html: come documento HTML
(nessuna differenza tra maiuscole e minuscole,
no etichetta di chiusura per elementi vuoti, ...)
- text: output di nodi testo senza escape

Altri attributi di xsl:output: -

- version
- omit-xml-declaration, standalone: valori yes o no
- doctype-public e doctype-system

```
<xsl:output method="html"/>
```

Elementi template

```
<xsl:template>
```

- specifica una regola template
- attributo match seleziona (usando un XPath) i nodi a cui applicare la regola

```
<xsl:apply-templates/>
```

- applica ricorsivamente i template
- attributo select permette di selezionare (usando un XPath) i nodi su cui continuare l'applicazione (per default i figli)

```
<xsl:call-template>
```

- specifica l'attivazione di uno specifico template (come una procedura convenzionale)
 - può contenere la specifica dei parametri (anche apply-templates)
-

XSLT - Esempio (2)

```
<TAVOLA-PERIODICA>
    <ATOMO> <NOME>Idrogeno</NOME> ... </ATOMO>
    <ATOMO> <NOME>Elio</NOME> ... </ATOMO>
    ...
</TAVOLA-PERIODICA>

<xsl:template match="TAVOLA-PERIODICA">
    <xsl:apply-templates select="ATOMO"/>
</xsl:template>

<xsl:template match="ATOMO">
    <xsl:value-of select="NOME"/>
</xsl:template>
```

Produce in output la lista dei nomi di atomi.

Istruzioni XSLT

Chiamiamo "istruzioni XSLT" gli elementi XSLT che si usano all'interno dei template

Esempio:

```
<xsl:value-of>
```

- copia il valore di un nodo del documento in ingresso nel documento in uscita
- se il nodo selezionato non è unico prende solo il primo

Tutti gli elementi (non-istruzioni) che si incontrano nel corpo di un template vengono considerati "*Elementi letterali del risultato*" e riportati testualmente in uscita.

Istruzione `<xsl:for-each>`

- processa in sequenza tutti i nodi selezionati
- può essere inserito direttamente in `<xsl:template>`

```
<TAVOLA-PERIODICA>
    <ATOMO> <NOME>Idrogeno</NOME> ... </ATOMO>
    <ATOMO> <NOME>Elio</NOME> ... </ATOMO>
    ...
</TAVOLA-PERIODICA>

<xsl:template match="TAVOLA-PERIODICA">
    <xsl:for-each select="ATOMO">
        <xsl:value-of select="NOME"/>
    </xsl:for-each>
</xsl:template>
```

Istruzione <xsl:element>

- genera in output un elemento
- il nome dell'elemento è specificato dall'attributo name

```
<ATOMO STATO="GAS"> <NOME>Elio</NOME> </ATOMO>
<ATOMO STATO="SOLIDO"> <NOME>Oro</NOME> </ATOMO>
```

```
<xsl:template match="ATOMO">
  <xsl:element name="@STATO">
    <NOME> <xsl:value-of select="NOME"/> </NOME>
  </xsl:element>
</xsl:template>
```

Sostituisce ATOMO con il valore dell'attributo STATO:

```
<GAS> <NOME>Elio</NOME> </GAS>
<SOLIDO> <NOME>Oro</NOME> </SOLIDO>
```

Ristrutturazione profonda

L'elemento a viene sostituito con un elemento A e si continua a processare il contenuto:

```
<xsl:template match="a">
  <A><xsl:apply-templates/></A>
</xsl:template>
```

L'elemento b viene sostituito con un elemento B e si copia il contenuto

```
<xsl:template match="b">
  <B><xsl:value-of select=". "/></B>
</xsl:template>
```

L'elemento c viene cancellato e si continua a processare il contenuto

```
<xsl:template match="c">
  <xsl:apply-templates/>
</xsl:template>
```

Per tutti gli altri elementi (*) si costruisce un elemento con lo stesso nome e si processa il contenuto:

```
<xsl:template match="* ">
  <xsl:element name="{name()}"><xsl:apply-templates/></xsl:element>
</xsl:template>
```

Istruzione <xsl:text>

Normalmente, per generare del testo in output è sufficiente includerlo nella regola template:

```
<xsl:template match="TAVOLA-PERIODICA">
  Atomi della tavola periodica:
  <xsl:apply-templates select="ATOMO"/>
```

```
</xsl:template>
```

Per preservare gli spazi bianchi si può usare <xsl:text>

```
<xsl:template match="TAVOLA-PERIODICA">
  <xsl:text>Atomi della tavola periodica:
  </xsl:text>
  <xsl:apply-templates select="ATOMO" />
</xsl:template>
```

Istruzione <xsl:copy>

Il nodo selezionato viene copiato dall'input all'output.

Per copiare anche nodi figlio, attributi, ...si può usare <xsl:apply-templates> all'interno di <xsl:copy>.

Esempio: trasformazione identità

```
<xsl:template match="* | @* | comment() | pi() | text()">
  <xsl:copy>
    <xsl:apply-templates
      select="* | @* | comment() | pi() | text()" />
  </xsl:copy>
</xsl:template>
```

Può essere adattata opportunamente per effettuare trasformazioni tra due formati molto simili.

Variabili

XSLT è un linguaggio funzionale puro, ovvero il concetto di variabile dei linguaggi imperativi o ad oggetti non è definito (non esiste nessuno stato). Esiste, tuttavia la possibilità di definire delle variabili o parametri come "puntatori" a porzioni del documento XML da processare ovvero risultati di query XPath (result tree-fragment).

Le variabili XSLT consentono di compattare la codifica della trasformazione o ad esprimere interrogazioni complesse.

A differenza delle variabili proprie, le variabili XSLT non possono essere riassegnate.

Lo scope di una variable può essere globale all'intero foglio di stile oppure locale ad un template.

I parametri sono particolari variabili da utilizzarsi per passare valori o dati ad un template. I parametri globali sono definiti per l'intero foglio di stile e devono essere valorizzati dal processo che invoca la trasformazione.

Istruzione <xsl:variable>

Assegna ad una variabile il nodeset derivante dalla valutazione di un'espressione XPath

```
<xsl:variable name="v" select="//libro[@titolo = 'XML']/autore"/>
```

La variabile può essere utilizzata nelle successive espressioni XPath dello stesso template (visibilità locale)

```
<xsl:value select="//libro[./autore = $v]"/>
```

Parametri

Un template dichiara la presenza di un parametro (argomento formale) con il costrutto `<xsl:param>`

```
<xsl:template name="t">
  <xsl:param name="p"/>
  ...

```

La valorizzazione di un parametro (argomento attuale) con il costrutto `<xsl:with-param>`

```
<xsl:call-template name="t">
  <xsl:with-param name="p" select="/e[@id = $v]"/>
</xsl:call-template>
```

I parametri vengono referenziati con la stessa sintassi delle variabili

Regole di default

XSLT definisce due regole che sono attive in tutti gli stylesheet (a meno che non vengano mascherate esplicitamente)

Regola di default per elementi: discende ricorsivamente

```
<xsl:template match="* | /">
  <xsl:apply-templates/>
</xsl:template>
```

Regola di default per nodi testo / nodi attributo: copia il testo / valore dell'attributo da input ad output

```
<xsl:template match="text() | @*">
  <xsl:value-of select=". "/>
</xsl:template>
```

XSLFO: esempio

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master
      master-name="simple" page-height="29.7cm"
      page-width="21.0cm" margin-bottom="2cm"
```

```
margin-left="2cm" margin-right="2cm"
margin-top="2cm">
<fo:region-body margin-top="1cm" margin-bottom="1cm"/>
<fo:region-before extent="1cm"/>
<fo:region-after extent="1cm"/>
</fo:simple-page-master>
</fo:layout-master-set>
<fo:page-sequence master-reference="simple">
<fo:flow flow-name="xsl-region-body">
<fo:block space-before="6pt" space-after="4pt" text-align="justify">
Testo <fo:inline font-size="14pt" font-color="red">formattato</fo:inline> contenuto nell'area
</fo:block>
<fo:block><fo:external-graphic src="url(immagine.gif)" /></fo:block>
</fo:flow>
</fo:page-sequence>
</fo:root>
```

Riferimenti

[Extensible Stylesheet Language \(XSL\) Version 1.1](#) W3C Recommendation 5 dicembre 2006

[XSL Transformations \(XSLT\) Version 2.0](#) W3C Recommendation 23 gennaio 2007

[XML Path Language \(XPath\) Version 1.0](#) W3C Recommendation 16 novembre 1999

[XML Path Language \(XPath\) Version 3.1](#) W3C Recommendation 21 marzo 2017

[Namespaces in XML](#) W3C Recommendation 8 dicembre 2009

[Chapter 17: XSL Transformations](#) The XML Bible (Second Edition) 2001, Elliotte Rusty Harold

Sapienza Università di Roma
corso di laurea in Ingegneria informatica e automatica

Linguaggi e tecnologie per il Web

a.a. 2020/2021

Progetti pratici

Riccardo Rosati

Progetto pratico: regole

- Modalità alternativa all'esame (totalmente o parzialmente)
- **Facoltativo** per gli studenti
- **Deve** essere svolto entro la **fine di questo semestre**
(ultima settimana di maggio 2021)
- **Deve** essere svolto da un **gruppo** di studenti
- I gruppi possono essere solo di **due studenti** o di **tre studenti**
- Ogni gruppo dovrà scrivere una email al prof. Rosati entro il
22 aprile 2021
 - oggetto: **LTW: progetto pratico**
 - nel messaggio scrivere **cognome, nome e matricola** di
ogni studente del gruppo, nonché il **titolo** del progetto

+ descrizione.

Progetto pratico: contenuti

- Progetto di una applicazione web, con maggiore enfasi sulla parte dinamica del lato client dell'applicazione (ma anche con un lato server)
- Ogni progetto **dovrà** usare i seguenti linguaggi, librerie e framework:
 - HTML, Bootstrap, CSS, JavaScript, e JQuery e/o Vue.js sul lato client
 - PHP (o altri linguaggi di scripting lato server) e un database relazionale sul lato server

Progetto pratico: presentazione

- I progetti verranno presentati nell'ultima settimana di maggio (e, se necessario, nella prima settimana di giugno)
- Ogni gruppo effettuerà una presentazione del lavoro svolto sul proprio computer, eseguendo l'applicazione creata (ed eventualmente mostrando porzioni del codice sorgente)
- Il tempo della presentazione sarà di circa 15 minuti
- Il tempo della presentazione dovrà essere equamente diviso tra tutti i componenti del gruppo

Progetto pratico: valutazione

- Durante la presentazione i docenti potranno fare domande sia sugli aspetti tecnici del progetto che più in generale sui linguaggi, metodi e tecniche utilizzati
- Tutti gli studenti del gruppo dovranno dimostrare di conoscere i contenuti del proprio progetto e dei linguaggi, metodi e tecniche utilizzati
- I docenti decideranno se il progetto svolto esonera **totalmente, parzialmente, o per nulla** dall'esame, e assegneranno un voto ad ogni studente

Progetto pratico: valutazione

Esonero totale dall'esame:

- Il progetto svolto soddisfa pienamente i requisiti (tutti i linguaggi, librerie e framework sono stati utilizzati)
- Lo studente ha dimostrato conoscenza dei contenuti del progetto e dei linguaggi, metodi e tecniche utilizzati

Progetto pratico: valutazione

Esonero parziale dall'esame:

- Il progetto svolto non soddisfa pienamente i requisiti e/o:
- Lo studente non ha dimostrato sufficiente conoscenza dei contenuti del progetto e/o dei linguaggi, metodi e tecniche utilizzati

Progetto pratico: valutazione

Nessun esonero dall'esame:

- Il voto assegnato allo studente è insufficiente