Markup: SGML, XML, DOM e (X)HTML

Angelo Di Iorio

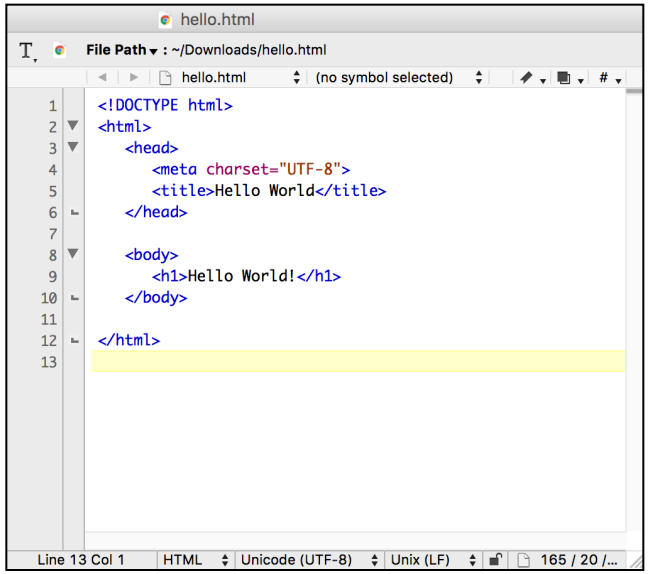
Università di Bologna

HTML

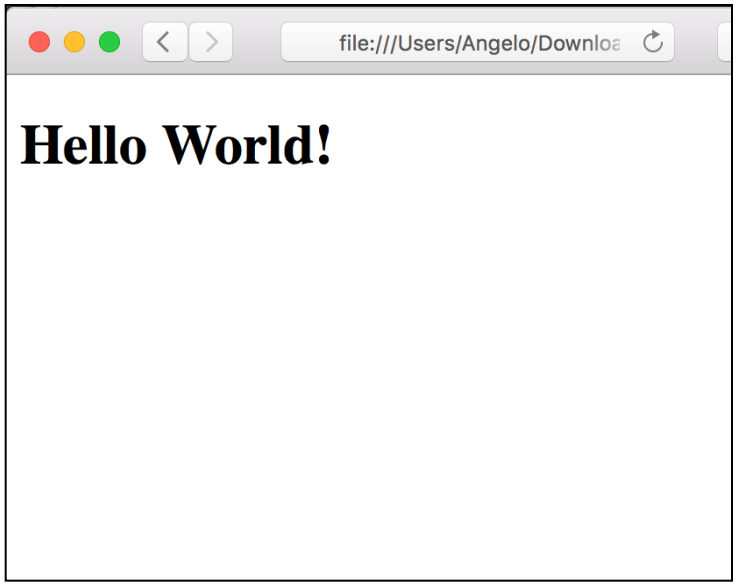
- Il sorgente delle pagine Web è in HTML (HyperText Markup Language), un linguaggio che “marca”:
  - struttura del documento
  - informazioni di presentazione
  - collegamenti ipertestuali (e URL della destinazione)
  - risorse multimediali (e URL)
- Il browser è in grado di interpretare queste informazioni e visualizzare la pagina finale

Sorgenti e rendering

Editor



Browser, read-only (o quasi)



Un passo indietro: il markup

- Definiamo markup ***ogni mezzo per rendere esplicita una particolare interpretazione di un testo***.
- Oltre a rendere il testo più leggibile, il markup permette anche di specificare ulteriori usi del testo.
- Con il markup per sistemi informatici (il nostro caso), specifichiamo le modalità esatte di utilizzo del testo nel sistema stesso.
- Il markup esiste da prima dell'informatica!

Tipi di markup

- Esistono diversi tipi di markup tra cui:
  - ***Presentazionale**:* indica effetti (grafici o altro) per rendere più chiara la presentazione del contenuto
  - ***Procedurale**:* indica ad un sistema automatico che che procedura (serie di istruzioni) eseguire per visualizzare il contenuto
  - ***Descrittivo**:* individua strutturalmente il tipo di ogni elemento del contenuto e ne evidenzia il ruolo
  - **Referenziale**: consiste nel fare riferimento ad entità esterne al documento per fornire significato o effetto grafico ad elementi del document
- Esistono molti linguaggi di markup per ogni tipo o che mescolano diversi tipi di markup

PDF

```
%PDF-1.6
1 0 obj
<<
  /Type /Catalog
  /Pages 2 0 R
>>
endobj
2 0 obj
<<
  /Type /Pages
  /Count 1
  /Kids [3 0 R]
>>
endobj
3 0 obj
<<
  /Type /Page
  /Parent 1 0 R
  /MediaBox [0 0 614 794]
  /Contents 4 0 R
  /Resources 5 0 R
>>
```

LaTeX

```
documentclass{article}

\begin{document}

\section{Titolo}

Questo \`e un paragrafo.

\subsection{Cap 1}

Paragrafo in una
sottosezione.

\end{document}

% Fine
```

Markdown

```
Titolo
=======

Paragrafo.

Paragrafo con testo in
*corsivo* o **grassetto**,
seguito da una lista:

  * A
  * B
```

XML

```
<book>
<title>Titolo</title>
<chapters>
   <chapter>
   <heading>Cap 1</heading>
      <para>Paragrafo</para>
   </chapter>
</chapters>
</book>
```

HTML

```
<html>
 <head>
 </head>
 <body>
  <h1>Titolo</h1>
  <p>Testo con un
  <a href="cap1.html">link al
        capitolo 1</a>.</p>
  <p>Secondo paragrafo</p>
 </body>
</html>
```

Meta-linguaggio di markup

- Un meta-linguaggio è un linguaggio per definire linguaggi, una grammatica di costruzione di linguaggi.
- SGML e XML non sono linguaggi di markup, ma linguaggi con cui definiamo linguaggi di markup, appunto **metalinguaggi di markup**
- SGML e XML non sanno cos'è un paragrafo, una lista, un titolo, ma forniscono grammatiche che ci permettono di definirli.
- HTML nasce come linguaggio di markup definito usando la grammatica SGML

(un pezzo di) HTML4 in SGML

```
…
<!ENTITY % inline "#PCDATA | %fontstyle; | %phrase; | %special;
| %formctrl;">
…
<!ENTITY % block
     "P | %heading; | %list; | %preformatted; | DL | DIV |
CENTER | NOSCRIPT | NOFRAMES | BLOCKQUOTE | FORM | ISINDEX | HR
| TABLE | FIELDSET | ADDRESS">

<!ENTITY % flow "%block; | %inline;">

…

<!ELEMENT BODY O O (%flow;)* +(INS|DEL) >
<!ATTLIST BODY
  %attrs;
  onload          %Script;   #IMPLIED
  onunload        %Script;   #IMPLIED
  …
…
```

Markup SGML

- Il markup in SGML è :
  - **strutturato**: possibile definire una serie di regole affinché il testo sia considerabile strutturalmente corretto
  - **gerarchico**: gli elementi del testo possono essere annidati e specificare la struttura in maniera gerarchica

- Un documento con markup di derivazione SGML - inclusi HTML e XML – contiene:
  - Elementi, attributi e testo
  - Entità
  - Commenti e Processing Instructions

Elementi e attributi e testo

- Gli **elementi** sono le parti di documento dotate di un senso proprio, e sono individuati da **tag iniziale**, contenuto e **tag finale**.
  - Non confondere i tag con gli elementi!
- Gli **attributi** sono informazioni aggiuntive sugli elementi, tipicamente espressi in sintassi **`nome="valore"`**
- Il **testo** è il contenuto vero è proprio. Viene anche detto **#PCDATA**, *Parsed Character DATA*, perché è processato ("parsato") per sostituire le entità

```
<section id="s1">
  <h1>Titolo</h1>
  <p>Testo con un
     <a href="cap1.html">
      link</a>.
  </p>
</section>
```

Entità

- Le entità sono frammenti di documento memorizzati separatamente e richiamabili all'interno del documento
- Sono usate sia nei linguaggi che nei metalinguaggi che li definiscono
- HTML definisce un set di entità utilizzabili, ad esempio, per caratteri non ammessi nella codifica usata (es. lettere accentate in ASCII)

```
<section id="s1">
  <h1>Titolo</h1>
  <p>Testo con un
     <a href="cap1.html">
      link</a>.
  </p>
  <p class="last">Questo
  &egrave; l'ultimo
  paragrafo</p>
<section>
```

Commenti e Processing Intructions

- I documenti di markup possono contenere commenti, non fanno parte del contenuto, ignorati dalle applicazioni di rendering
- Le processing instructions (PI) sono elementi particolari usati per dare indicazioni alle applicazioni. Molto usate in XML, poco in HTML

```
<section id="s1">
  <h1>Titolo</h1>
  <p>Testo con un
     <a href="cap1.html">
      link</a>.
  </p>
  <p class="last">Questo
  &egrave; l'ultimo
  paragrafo</p>
<section>
<!-- fine sezione -->
```

XML

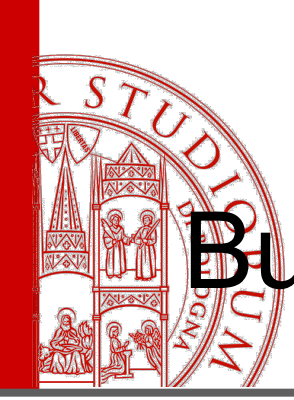
- Una raccomandazione W3C del 10 febbraio 1998
- È definita come un sottoinsieme di SGML ma è molto più formalizzata
- XML distingue due tipi di documenti rilevanti per le applicazioni
  - **ben formato**: se ha una struttura sufficientemente regolare e comprensibile da poter essere controllata
  - un documento è **valido** se presenta uno schema che ne definisce la struttura (chiamato *DTD, Document Type Definition* come per SGML) ed è possibile validarlo usando questo DTD

Documenti XML ben formati

- Un documento XML si dice ben formato se:
  - Tutti i tag di apertura e chiusura corrispondono e sono ben annidati
  - Esiste un elemento radice che contiene tutti gli altri
  - Gli elementi vuoti (senza contenuto) utilizzano un simbolo speciale di fine tag: `<vuoto/>`
  - Tutti gli attributi sono sempre racchiusi tra virgolette
  - Tutte le entità sono definite

- **Se un documento non rispetta queste regole NON può essere processato da un'applicazione XML**

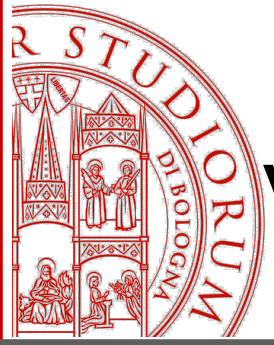
- Questa rigidità garantisce maggiore controllo e interoperabilità tra le applicazioni



Buona forma XML: cosa non va?

```
<?xml version="1.0"
encoding="UTF-8"?>
<book>
 <title>Titolo</title>
 <chapters>
   <chapter id=c1>
      <heading>Cap 1</heading>
      <para>Paragrafo con
       <bold><italic>corsivo
       e grassetto</bold>
       </italic></para>
       <marker>
   </chapter>
</chapters>
</book>
```

```
<?xml version="1.0"
encoding="UTF-8"?>
<book>
 <title>Titolo</title>
 <chapters>
   <chapter id="c1">
      <heading>Cap 1</heading>
      <para>Paragrafo con
       <bold><italic>corsivo
       e grassetto</italic>
       </bold></para>
       <marker/>
   </chapter>
</chapters>
</book>
```



Validazione XML: cosa non va qui?

```
<!ELEMENT book  (title, chapters)>

<!ELEMENT chapters (chapter+)>

<!ELEMENT chapter (heading, para*)>
...
```

\+ almeno 1 elemento
* da 0 a piu elemento
? da 0 a 1 elemento

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
 <title>Titolo</title>
 <chapters>
   <chapter id=c1>
      <heading>Cap 1</heading>
      <para>A paragraph</para>
       <marker/>
   </chapter>
</chapters>
</book>
```

Problema?

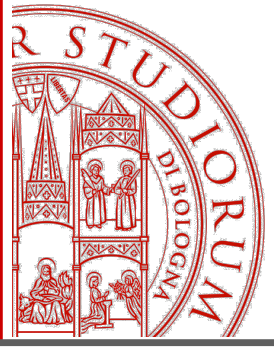
Cosa succede durante il parsing del seguente frammento XML?

```
<?xml version="1.0" encoding="UTF-8"?>
<math>
 <formula>
    Se A < B allora B >= A
 <formula>
</math>
```

Entità carattere e parsing

- XML definisce 5 entità predefinite utili nei casi simili al precedente
- Queste entità sono espanse durante la fase di parsing
- Inoltre è possibile identificare un carattere dato il suo codice numerico
  - `&#60;` `&#62;` `&#x3E;`
- TUTTE le altre entità vanno esplicitamente definite!
- In particolare non esistono le entità di HTML come `&agrave;` o ` `

Entity	Character
<	<
>	>
&	&
'	'
"	"



Problema risolto

```
<?xml version="1.0" encoding="UTF-8"?>
<math>
 <formula>
    Se A &lt; B allora B &gt;= A
 <formula>
</math>
```

+	almeno 1 elemento
*	da 0 a piu elemento
?	da 0 a 1 elemento

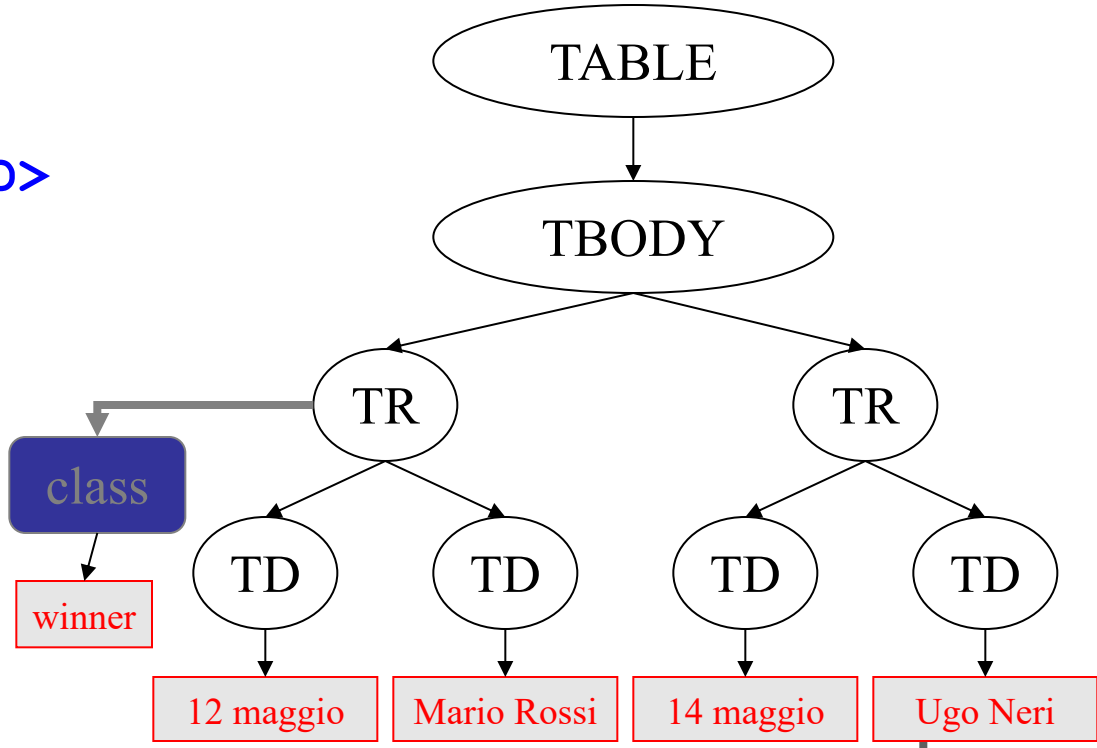
```
<?xml version="1.0" encoding="UTF-8"?>
<math>
 <formula>
    Se A &#60; B allora B &#62;= A
 <formula>
</math>
```

XML e DOM

- Se un documento è ben formato può essere rappresentato in memoria come un albero, chiamato DOM
- Document Object Model è un **interfaccia di programmazione (API)** per documenti sia XML che HTML
- Definisce la struttura logica dei documenti ed il modo in cui si accede e si manipola un documento.
- Utilizzando DOM i programmatori possono costruire documenti, navigare attraverso la loro struttura, e aggiungere, modificare o cancellare elementi.
- Ogni componente di un documento XML (o HTML) può essere *letto, modificato, cancellato o aggiunto* utilizzando il Document Object Model.

Elaborazione DOM

```
<TABLE>
  <TBODY>
    <TR class="winner">
      <TD>12 maggio</TD>
      <TD>Mario Rossi</TD>
    </TR>
    <TR>
      <TD>14 maggio</TD>
      <TD>Ugo Neri</TD>
    </TR>
  </TBODY>
</TABLE>
```



Oggetti del DOM

- L'oggetto principale di DOM è *DOMNode*, che però è una interfaccia (cioè viene solo usata per crearne classi)
- Il core del DOM definisce alcune classi fondamentali per i documenti HTML e XML, e ne specifica proprietà e metodi.
- Gli oggetti principali definiti nel DOM sono:
  - `DOMDocument` : il documento di cui si sta parlando
  - `DOMElement`: ogni singolo elemento del documento
  - `DOMAttr`: ogni singolo attributo del documento
  - `DOMText`: ogni singolo nodo di testo del documento
  - `DOMComment`, `DOMProcessingInstruction`, `DOMCDATASection`, `DOMDocumentType`, ecc.

Un esempio: DOM Node

- **DOMNode** specifica i metodi per accedere a tutti gli elementi di un nodo di un documento, inclusi il nodo radice, il nodo documento, i nodi elemento, i nodi attributo, i nodi testo, ecc. Semplificando:

membri	**metodi**
– `nodeName`	`insertBefore()`
– `nodeValue`	`replaceChild()`
– `nodeType`	`removeChild()`
– `parentNode`	`appendChild()`
– `childNodes`	`hasChildNodes()`
– `attributes`	`hasAttributes()`

Manipolazione del DOM in Javascript (ci torneremo)

```
var root = document.documentElement
var nodes = root.childNodes()
for (var x=0; x<.nodes.length; x++)
  { ... }

root = document.createElement("tr");
item = document.createElement("td");
text = document.createTextNode("testo della
cella")
item.appendChild(text);
root.appendChild(item);
document.documentElement = root
```

E in HTML?

- Le versioni di HTML hanno adottato in modo molto diverso le regole di buona forma e validazione, sotto le diverse spinte del W3C e dei produttori di browser
- Analizziamo i momenti principali dell'evoluzione del linguaggio, che ne hanno poi determinato alcune caratteristiche
- Un punto fondamentale: esistono **differenze** anche **sensibili** tra un documento **HTML *corretto*** e un documento **HTML *visualizzabile*** *da un browser Web*
- **Non è detto che un documento HTML "accettabile" sia "ben fatto"**

HTML 4

- Il linguaggio HTML un tipo di documento SGML (esiste un DTD di HTML, anzi più di uno) progettato per marcare documenti ipertestuali.
- La cosidetta “guerra dei browser” (1994-98) ha portato alla creazione di numerosi elementi proprietari, estensioni, effetti sofisticati e caratteristiche puramente presentazionali
- Dopo diverse versioni intermedie, HTML 4.0 (1997) “chiude i giochi” e aggiunge il supporto l’internazionalizzazione, per gli style sheet, per i frame, tabelle molto più ricche, etc.
- Da allora lo standard è rimasto sostanzialmente invariato fino ai giorni nostri**: HTML 4 è il linguaggio in cui è scritta la stragrande maggioranza delle pagine Web attuali**

Da HTML a “tag soup”

• Molte pagine HTML tuttavia sono diventate “tag soup”, ossia un insieme di elementi non conformi allo standard. I browser infatti si sono preoccupati poco della correttezza sintattica o strutturale dei documenti HTML.

• Ma come fa il browser ad interpretare i documenti sintatticamente non corretti? Prima di HTML 5 ognuno faceva a modo suo!

• Per gestire sia le pagine conformi allo standard che quelle non compatibili (proliferate dopo la “guerra dei browser”), sono stati introdotti due modelli di rendering: ***quirks mode*** e ***strict mode***

• Se la pagina non specifica niente, il browser adotta il modo compatibile (quirks mode), altrimenti se l’autore lo richiede esplicitamente attiva il modo restrittivo e corretto (strict mode)

XHTML

• XHTML 1.0 è una Recommendation W3C del 2000. È una riformulazione di HTML 4 come un'applicazione di XML.

• ***L'elenco e la semantica di elementi e attributi non è assolutamente cambiata rispetto a HTML 4.***

• XHTML è il primo di una serie di DTD che riproducono, limitano ed estendono HTML. Sono fatti per lavorare con user agent basati su XML, ma con un'esplicita strategia di transizione.

• **I documenti debbono essere ben formati, in particolare l'annidamento deve essere corretto.**

• XHTML 2.0 adotta **solo sintassi XML**, rimuove tutti gli elementi presentazionali, riduce l'interattività e **non è compatibile all'indietro.**

• In pratica la proposta non ha successo e il WG chiude

Parsing (X)HTML

HTML 4.x

```
<html>
 <body>
  <h1>Titolo</h1>
  <p>Paragrafo</p>
  <ul id=list1>
   <li>
     <b>A</b>
   </li>
   <li>A</li>
   <li>A
  </ul>
 </body>
</html>
```

Tag soup

```
<body>
  <h1>Titolo</h1>
  <p>Paragrafo
  </p>
  <ul id=list1>
   <li><b>A</li>
   <li>A
   <li>A
  </ul>
</body>
```

XHTML

```
<html>
 <body>
  <h1>Titolo</h1>
  <p>Paragrafo</p>
  <ul id="list1">
   <li>
     <b>A</b>
   </li>
   <li>A</li>
   <li>A</li>
  </ul>
 </body>
</html>
```

(X)HTML 5

- Nel 2004, Firefox e Opera proposero al W3C la riapertura del Working Group su HTML per lo sviluppo di nuove versioni del linguaggio. La proposta, che ignorava volutamente XHTML e la rigida sintassi di XML, venne bocciata dal W3C.
- Venne così creata una comunità aperta chiamata WHATWG (Web Hypertext Application Technology Working Group) finanziata e supportata da Mozilla, Opera e Apple, che ha iniziato a lavorare ad una versione "intermedia" di HTML, "Web Application 1.0" (WA1), meno ambiziosa di XHTML 2.0
- Nel 2007 il W3C dovette ammettere che queste modifiche avevano un impatto innegabile (*"he who ships working code wins"*) riaprì il working group con tutti i membri del WHAT per creare una nuova versione del linguaggio, (X)HTML 5.

~~(X)~~HTML ~~5~~

- WA1 accetta l'amara realtà dei browser odierni che dicono che non è mai esistita una grammatica SGML per HTML, ma che il linguaggio è sempre stato
  - O un'applicazione di XML (XHTML*)
  - Oppure una *tag soup* in cui i browser accettano ogni sorta di "porcheria" e fanno il meglio che possono
- WA1 è più interessato a mettere a posto le cose che certamente non vanno piuttosto che lavorare su un linguaggio completamente nuovo. E il W3C deve adeguarsi.
- Anche perché WA1 ha supporto dichiarato da tutti i browser e di tutta l'industria delle applicazioni Web
- "(X)HTML 5" diventa quindi solo "HTML 5" e successivamente solo "HTML"

HTML Living Standard

- Il “nuovo” HTML è, per scelta del WG, una specifica perennemente in sviluppo (“*living standard*”)

- *We are today announcing two changes:*

  – *The HTML specification will henceforth just be known as "HTML”.*

  – *The WHATWG HTML spec can now be considered a **"living standard"**. It's more mature than any version of the HTML specification to date, so it made no sense for us to keep referring to it as merely a draft.*

    • *Ian Hickson, chair del WHATWG, 19 gennaio 2011, http://blog.whatwg.org/html-is-the-new-html5*

- Cambia completamente il modello di sviluppo del linguaggio che si allontana sensibilmente dall’approccio sistematico e democratico (ma non privo di difetti) di evoluzione degli altri standard W3C

- **E’ la vittoria indiscussa dei produttori di browser: sia per le nuove caratteristiche del linguaggio sia per il modo in cui è sviluppato**

Esistono quindi versioni di HTML?

- Nel Working Group di HTML convivono quindi due “anime” ma con scarsi risultati e nel 2011 i gruppi si dividono
- Lo sviluppo va avanti in parallello: il W3C continua a standardizzare "snapshot" di "HTML Living Standard" e a dargli un'approvazione formale con "HTML 5.x"
- Nel 2019 W3C e WHATCG raggiungono un accordo e si impegnano a sviluppare un'unica versione di HTML, nell'ottica di Web come Open Platform

Document Type HTML

- Nella sintassi SGML e XML, i documenti iniziano con dichiarazione di tipo (*DocType*) che serve a specificare le regole per validare i documenti
- Esistono quindi diversi DocType per HTML:

```
<?DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 2.0//EN"
"http://www.w3.org/MarkUp/DTD/xhtml2.dtd">
```

- HTML(5) semplifica e usa un unico DocType, da indicare all'inizio delle pagine prima dell'elemento radice:

```
<?DOCTYPE html>
```

Un punto chiave: parsing HTML

HTML 4.x

```
<html>
 <body>
  <h1>Titolo</h1>
  <p>Paragrafo</p>
  <ul id=list1>
   <li>
     <b>A</b>
   </li>
   <li>A</li>
   <li>A
  </ul>
 </body>
</html>
```

XHTML

```
<html>
 <body>
  <h1>Titolo</h1>
  <p>Paragrafo</p>
  <ul id="list1">
   <li>
     <b>A</b>
   </li>
   <li>A</li>
   <li>A</li>
  </ul>
 </body>
</html>
```

HTML Living Standard

```
<body>
  <h1>Titolo
  <p>Paragrafo
  </p>
  <ul id=list1>
   <li><b>A</li>
   <li>A
   <li>A
</body>
```

"Buona forma" in HTML 5

- “HTML Living Standard” definisce un **algoritmo** per fare **il parsing di qualunque documento HTML**, anche dei documenti mal formati, sulla base di ciò che i browser già facevano (in quirks mode)
- In realtà dalla prospettiva WHATCG questi documenti non sono propriamente “mal formati” ma **semplicemente “non strict”**. Sono validi a tutti gli effetti, tanto quanto i documenti XHTML!
- Pragmaticamente potremmo dire: “**l’importante è arrivare ad una struttura dati in memoria unica su cui costruire applicazioni**”.
- E' a questo scopo che la vera attenzione da parte del WHATWG è la costruzione di una struttura dati chiamata **Document Object Model (DOM)**, a cui sia possibile arrivare a partire dalla stringa HTML e da cui si possa generare nuovamente una altra stringa HTML.

Un problema risolto?

- Aver uniformato l'algoritmo di parsing non è un deterrente per creare pagine "ben formate", al contrario lascia maggiore libertà e margine di errore agli sviluppatori.
- Più complesso estrarre dati e implementare manipolazioni automatiche dei contenuti, tranne ovviamente il caso il sistema espone un'API di accesso ai dati
- La cosa si complica ancora di con i *sistemi a componenti* (es. AngularJS o React): lo stesso concetto di markup perderà valore a vantaggio di sintassi miste Javascript/Markup/CSS/whatever create ad hoc e mai standardizzate

Non è solo questione di parsing e buona forma…

```
<p>Roma è la
capitale
d'Italia.</p>
<p>Ha 3 milioni
di abitanti.</p>
```

Roma è la capitale d'Italia.

Ha 3 milioni di abitanti.

```
<font>
 <p>Roma è la
   capitale
   d'Italia.</p>
 <p>Ha 3 milioni
   di abitanti.</p>
</font>
```

```
Roma è la
capitale
d'Italia.<br/>
<br/>
Ha 3 milioni di
abitanti.
```

```
<span>Roma è la
capitale d'Italia.
<br/>
</span>
<span>Ha 3 milioni
di abitanti.</span>
```

Museo degli orrori

```
<table border=0>
 <tr><td>Roma è la capitale d'Italia.</td></tr>
 <tr><td>Ha 3 milioni di abitanti.</td></tr>
</table>
```

```
<table border=0>
 <tr><td>Roma è la capitale d'Italia.
 <tr><td>Ha 3 milioni di abitanti.
</table>
```

```
<font>
<p>Roma è la capitale d'Italia.<font>
<p>Ha 3 milioni di abitanti.
</font>
```

Conclusioni

- Eesistono differenze anche sensibili tra un documento **HTML sintatticamente corretto** e un documento **HTML visualizzabile** da un browser Web
- Anche se sintatticamente in buona forma un documento HTML può essere strutturato in modo non corretto
- Separare contenuto e struttura è fondamentale in quest'ottica
- …così come evidenziare il ruolo dei singoli elementi e marcarli correttamente
- Importante usare correttamente gli elementi HTML e CSS