

---

# XML

## Corso di ***Applicazioni Telematiche***

A.A. 2005-06 – Lezione n.8

Prof. Roberto Canonico

Università degli Studi di Napoli Federico II

Facoltà di Ingegneria

---

---

## Argomenti

- XML: usi e scopi
  - XML: sintassi
  - XML ed i namespace
  - XML ed XSLT
-

---

## Cosa è XML

- XML è un linguaggio di *markup* per documenti di tipo testuale
  - Grazie all'uso dei tag, in un documento XML sono contenuti sia dati che *metadati*, ovvero *dati sui dati*, che possono servire a descriverne la semantica
  - Come HTML, anche XML deriva dal linguaggio SGML (*Standard Generalization Markup Language*)
  - A differenza di HTML, un documento XML non contiene informazioni sulla *presentazione* dei dati
  - XML è uno standard W3C
  - XML è flessibile
  - XML è portabile
- 

---

## Cosa non è XML

- XML non è un linguaggio di programmazione
  - XML non è un protocollo di rete
  - XML non è un database
-

## Usi e scopi di XML

- La genericità di XML ne ha consentito l'uso in svariati campi applicativi, come:
  - Formato di interscambio dei dati tra applicazioni diverse
  - Formato di codifica platform-independent di informazioni strutturate complesse (es. presentazioni multimediali, immagini grafiche vettoriali, ecc.)
  - Formato per lo scambio di messaggi tra applicazioni
    - XML-RPC
    - SOAP
  - ...
- L'uso di XML è sempre più diffuso, anche grazie all'esistenza di potenti librerie che facilitano la manipolazione di documenti XML

## XML e il Web

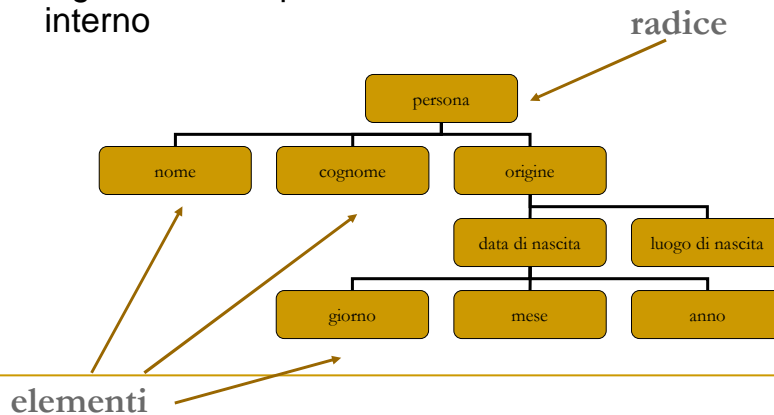
- XML è un ottimo strumento se usato per il web
- XML può essere usato per
  - sostituire il linguaggio HTML con un linguaggio più flessibile (XHTML)
  - definire il contenuto di una pagina Web (XML)
  - definire lo stile con il quale visualizzare una pagina web (XSL)

## Alcune applicazioni di XML

- SVG (Scalable Vector Graphics)
- MathML
- CML (Chemical Markup Language)
- RDF (Resource Description Language)
- XHTML
- SMIL
- MMS
- ...

## Struttura ad albero del contenuto XML

- Il contenuto di un documento XML può essere descritto mediante un albero gerarchico
- Un documento può avere una sola radice
- Ogni elemento può contenere altri elementi al suo interno



## Un esempio di file XML

```
<?xml version="1.0" encoding="UTF-8"?>
<persona>
  <nome>Mario</nome>
  <cognome>Bianchi</cognome>
  <origine>
    <data_di_nascita>
      <giorno>13</giorno>
      <mese>giugno</mese>
      <anno>1967</anno>
    </data_di_nascita>
    <luogo_di_nascita>Milano</luogo_di_nascita>
  </origine>
</persona>
```

Diagram annotations:

- prologo**: points to the XML declaration line.
- tag**: points to the opening <nome> tag.
- dato**: points to the value 13 inside the <giorno> tag.

## Prologo

- Il prologo di un documento XML contiene la dichiarazione di documento XML:
  - <?xml version="1.0" encoding="UTF-8"?>
- Opzionalmente, un documento può contenere la definizione della sua struttura, mediante una DTD (descritta in seguito), che può essere inserita nel documento stesso o salvata in un file esterno
- Esempio di riferimento a DTD esterna:
  - <!DOCTYPE slideshow SYSTEM "slideshow.dtd">

## I tag

- I tag XML hanno lo stesso aspetto dei tag HTML
  - <persona>
  - <nome>
  - <data\_di\_nascita>

## Nomi di tag XML

- Non possono essere usati:
  - spazi
  - virgolette “, apostrofi ‘, il simbolo \$
  - i simboli < e >
  - il simbolo % e il punto e virgola ;
- Il nome usato per un tag è sensibile alla differenza tra maiuscole e minuscole
  - Il tag <Persona> è diverso da <persona>

## I dati

- Tra un tag di apertura e un tag di chiusura possono essere inseriti:
  - altri tag
  - dei dati testuali
  - contenuto misto

```
<data_di_nascita>  
  <giorno>13</giorno>  
  <mese>giugno</mese>  
  <anno>1967</anno>  
</data_di_nascita>
```

```
<p>  
  lezione di <b>XML</b>  
</p>
```

## Attributi

- Ad un tag possono essere associati degli attributi
- Un attributo consiste in una coppia nome-valore:
  - `<persona nome='Mario' cognome='Bianchi'>`
  - `<persona nome="Mario" cognome="Bianchi">`

## Commenti

- I documenti XML possono contenere dei commenti e delle note.
- Un commento si può inserire usando la sintassi:
  - `<!-- Commento -->`

## La dichiarazione XML

- I documenti XML dovrebbero iniziare con una dichiarazione XML:
  - `<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>`
- dove:
  - version identifica la versione di XML in uso. Attualmente la versione in uso è la 1.0
  - encoding specifica quale tipo di codifica deve essere usata all'interno del documento: XML supporta un grande quantità di set di caratteri
  - standalone indica se il documento contiene ("no") o non ("yes") riferimenti a documenti esterni o a definizioni DTD esterne
- In gergo XML, si tratta di una particolare "processing instruction" (PI)



## Documenti ben formati

- Affinché un documento XML sia **ben formato** è necessario rispettare alcune regole sintattiche:
  1. ad ogni tag iniziale deve corrispondere un tag finale
  2. gli elementi non possono sovrapporsi
  3. deve esistere uno ed un solo elemento radice
  4. i valori degli attributi devono essere specificati tra apici
  5. i commenti e le istruzioni di elaborazione non possono apparire all'interno dei tag
  6. non deve apparire nessun carattere < o & all'interno dei caratteri di un elemento o di un attributo

## Regola 1

- Ad ogni tag iniziale deve corrispondere un tag finale
  - <giorno>13</giorno>
  - <mese>giugno</mese>
  - <anno>1967</anno>

## Regola 2

- Gli elementi non possono sovrapporsi
- Non valido:
  - `<giorno>13<mese>giugno</giorno></mese>`
- Valido:
  - `<giorno>13</giorno>`
  - `<mese>giugno</mese>`

## Regola 3

- Deve esistere uno ed un solo elemento radice

```
<?xml version="1.0" ?>  
<persona>  
  <nome>Luigi</nome>  
</persona>  
<persona>  
  <nome>Mario</nome>  
</persona>  
<persona>  
  <nome>Valeria</nome>  
</persona>
```

```
<?xml version="1.0" ?>  
<elenco_persone>  
  <persona>  
    <nome>Luigi</nome>  
  </persona>  
  <persona>  
    <nome>Mario</nome>  
  </persona>  
  <persona>  
    <nome>Valeria</nome>  
  </persona>  
</elenco_persone>
```

## Regola 4

- I valori degli attributi devono essere specificati tra apici
- Non valido:
  - `<persona nome=Mario cognome=Rossi>`
- Valido:
  - `<persona nome='Mario' cognome='Rossi'>`

## Regola 5

- I commenti e le istruzioni di elaborazione non possono apparire all'interno dei tag
- Non valido:
  - `<persona <!-- una persona > >`
- Valido:
  - `<persona> <!-- una persona >`

## Regola 6

- Non deve apparire nessun carattere < o & all'interno dei caratteri di un elemento o di un attributo
- Non valido:
  - <espressione> a > b </espressione>
- Valido:
  - <espressione> a &gt; b </espressione>

## Riferimenti ad entità

- Per inserire alcuni caratteri speciali come testo interno ai tag si usano dei codici chiamati *riferimenti ad entità*

| referenza | carattere  |   |
|-----------|------------|---|
| &lt;      | Minore     | < |
| &gt;      | Maggiore   | > |
| &amp;     | Ampersand  | & |
| &quot;    | Quote      | " |
| &apos;    | Apostrophe | ' |

## Entità

- In generale, una entità è un testo a cui ci si può riferire tramite un riferimento simbolico
- Alcuni riferimenti ad entità sono predefiniti (es. &gt; ...), altri possono essere definiti esplicitamente nella DTD (vedi avanti):
  - <!ENTITY copyright "Copyright 2006 – ACME spa">
- e poi riferiti simbolicamente nel documento XML:
  - &copyright;
- I riferimenti ad entità non vanno confusi con i codici di riferimento a carattere che consentono di inserire nel testo caratteri speciali mediante il loro codice (es. ASCII)
  - &#233; oppure &#xE9; per indicare il carattere 'è'

## Sezioni CDATA

- Quando un documento XML contiene delle grosse porzioni di testo, usare le referenze ad entità può risultare fastidioso
- Per semplificare la scrittura di documenti XML si introducono le sezioni CDATA
- All'interno di una sezione CDATA può essere usato qualunque carattere, senza dover far uso delle referenze ad entità

## Sezione CDATA

- Una sezione CDATA inizia con la sequenza
  - `<![CDATA[`
- e termina con la sequenza
  - `]]>`
- Ad esempio:
  - `<p> <![CDATA[` qui dentro posso usare tutti i caratteri che desidero compreso `>` `<` `%` `"` `'` oppure i caratteri accentati `à` `è` `ì` `ò` `ù` `]]>` `</p>`

## Elementi vuoti

- Quando un elemento non deve avere nessun contenuto esso può apparire in due forme
- Nella forma estesa:
  - `<elemento></elemento>`
- Oppure nella forma contratta:
  - `<elemento/>`

---

## Documenti validi

- La flessibilità di XML ne consente l'utilizzo in campi completamente diversi tra loro
  - Ogni applicazione XML ha quindi scopi diversi e di conseguenza è necessario usare elementi differenti
  - Occorre quindi uno strumento per imporre una determinata struttura ed un determinato vocabolario di tag ad un file XML
- 

---

## DTD

- Il primo strumento elaborato per fornire delle regole di composizione strutturale è DTD
  - Una DTD (*Document Type Definition*) descrive con precisione quali elementi ed entità possono apparire all'interno di un documento, quali possono essere i contenuti e gli attributi di tali elementi
  - Una DTD può essere inserita nel prologo di un documento XML (*inline*) o in un file esterno con estensione .dtd
-

---

## Validazione

- Quando un documento XML rispetta le regole imposte da un determinato DTD si dice che risulta **valido** per l'applicazione specifica
  - Rispetto alle regole di ben formazione, le regole di validazione sono opzionali
  - Lo scopo di tali regole è quello di generare documenti che possono essere capiti dalle applicazioni
- 

---

## DTD: principio di base

- Un DTD opera sul principio che tutto ciò che non è esplicitamente dichiarato come permesso allora deve essere considerato vietato
-



## Esempio di DTD

```
<!ELEMENT persona (nome, cognome, origine)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT origine (data_di_nascita, luogo_di_nascita)>
<!ELEMENT data_di_nascita (giorno, mese, anno)>
<!ELEMENT luogo_di_nascita (#PCDATA)>
<!ELEMENT giorno (#PCDATA)>
<!ELEMENT mese (#PCDATA)>
<!ELEMENT anno (#PCDATA)>
```

dichiarazione di elemento



## Dichiarazione di elementi

- Ogni elemento utilizzato in un documento valido deve essere dichiarato nel DTD
- La sintassi è la seguente:
  - <!ELEMENT nome\_elemento (modello\_di\_contenuto) >
- Dove:
  - il nome di elemento può essere un qualunque nome XML legale

## Modello di contenuto

- Il modello di contenuto dichiara, per ciascun elemento, quali figli possa o debba avere l'elemento in questione e in quale ordine
- I modelli di contenuto possono essere scelti tra sette diversi tipi di elementi:
  - #PCDATA
  - Elementi figli
  - Sequenze
  - Scelte
  - Elementi vuoti
  - ANY
  - Contenuto misto

## #PCDATA

- E' il più semplice modello di contenuto
- Specifica che l'elemento può contenere solamente dati di tipo carattere
- Esempio:
  - `<!ELEMENT cognome (#PCDATA)>`
- Un elemento valido:
  - `<cognome>Bianchi</cognome>`

## Elementi figli

- Specifica che un elemento deve contenere esattamente un elemento figlio di un determinato tipo
- Esempio:
  - `<!ELEMENT fax (numero_di_telefono)>`
- Corrisponde a:  
`<fax>`  
`<numero_di_telefono>091123456</numero_di_telefono>`  
`</fax>`

## Sequenze

- Generalmente gli elementi contengono più di un elemento figlio
- Per specificare quali elementi devono essere contenuti è sufficiente elencarli separandoli con delle virgole
- Esempio:
  - `<!ELEMENT persona (nome, cognome, origine)>`

## Ordine

- Una sequenza specifica che gli elementi devono apparire nell'ordine specificato
- Qualora l'ordine non sia rispettato il file XML non risulta valido rispetto al DTD

## Numero di figli

- E' possibile aggiungere un suffisso per specificare quante istanze di figli sono permessi per un dato elemento.
- Suffissi
  - ? corrisponde a "zero o un elemento"
  - \* corrisponde a "zero o più elementi"
  - + corrisponde a "uno o più elementi"

## Esempi

```
<!ELEMENT name (first_name, middle_name?, last_name )>
```

```
<!ELEMENT name (first_name+, last_name )>
```

```
<!ELEMENT persona (nome, cognome, professione* )>
```

## Scelte

- Può capitare che il contenuto di un elemento debba essere selezionato tra un elenco di possibilità (come per un *tipo enumerativo*)
- Una scelta è un elenco di nomi separati da barre verticali |
- Esempio:  

```
<!ELEMENT mese ( gennaio | febbraio | marzo | aprile  
| maggio | giugno | luglio | agosto | settembre |  
ottobre | novembre | dicembre )>
```

## Uso delle parentesi

- E' possibile combinare scelte e sequenze in modo anche complessi usando le parentesi
- Esempio:
  - `<!ELEMENT cerchio (colore, (raggio | diametro) )>`
- Documenti validi (rispetto alla regola di esempio):

```
<cerchio>
  <colore>blu</centro>
  <raggio>24</raggio>
</cerchio>
```

```
<cerchio>
  <colore>blu</colore>
  <diametro>48</diametro>
</cerchio>
```

## Elementi vuoti

- Alcuni elementi, chiamati **elementi vuoti**, non hanno contenuto
- Gli elementi vuoti vengono dichiarati usando la parola `EMPTY`
- Esempio:
  - `<!ELEMENT sposato EMPTY>`
- Esempio di documento con elemento vuoto:

```
<persona>
  <nome>Mario</nome>
  <cognome>Bianchi</cognome>
  <sposato/>
</persona>
```

---

## ANY

- Alcuni DTD particolarmente liberi permettono di specificare un contenuto arbitrario
  - Tale modello viene dichiarato usando la parola ANY
    - `<!ELEMENT page ANY>`
  - I figli di page possono essere di qualunque tipo: qualsiasi elemento (compreso page), testo, oppure contenuto misto
- 

---

## Contenuto misto

- Per definire che il contenuto di un certo elemento può essere misto si usa la seguente sintassi:
    - `<!ELEMENT page (#PCDATA | elemento_figlio)* >`
    - Questa dichiarazione indica che l'elemento page può contenere qualunque combinazione di caratteri e di elementi di tipo elemento\_figlio
-

---

## Dichiarazione di attributi

- Oltre a dichiarare gli elementi di un documento, si devono dichiarare anche gli attributi
  - La dichiarazione degli attributi ha la seguente forma:
    - `<!ATTLIST elemento attributo caratteristiche >`
  - Una dichiarazione ATTLIST può essere usata per dichiarare più di un attributo
- 

---

## Dichiarazione di attributi

- `<!ATTLIST elemento attributo caratteristiche >`
- **elemento** è il nome dell'elemento che deve contenere l'attributo
  - **attributo** è il nome dell'attributo
  - **caratteristiche** è un insieme di parametri che specificano
    - il tipo di attributo
    - se l'attributo è obbligatorio oppure opzionale
-



## Tipi di attributo

- CDATA: tipo “sequenza di caratteri”
- Tipo “scelta”: (pari|dispari)
- NMTOKEN: tipo “nome di elemento XML”
- NMTOKENS: una lista di nomi di elementi
- ID: tipo “nome di elemento XML” e deve essere un unico nel documento
- IDREF: è l’ID di un altro elemento
- ENTITY: una entità XML
- ENTITIES: una lista di entità XML
- NOTATION
- xml

## Attributo CDATA

- Un attributo CDATA può contenere una qualsiasi stringa di testo accettabile da XML
- Esempio:
  - `<persona nome=“Mario Bianchi”>`

## Valore di presenza

- Ogni dichiarazione ATTLIST può includere un valore di presenza dell'attributo
  - #IMPLIED
    - Indica che l'attributo è opzionale
  - #REQUIRED
    - Indica che l'attributo è obbligatorio
  - #FIXED
    - Indica che l'attributo è costante e non può essere variato

## Esempio di dichiarazione di attributi

```
<!ATTLIST persona    nome CDATA #REQUIRED
                    cognome CDATA #REQUIRED
                    eta CDATA #IMPLIED
                    nazionalita CDATA #FIXED "Italiana">
```

---

## XML Schema: cenni

- Uno XML Schema definisce:
    - gli elementi e gli attributi che possono apparire in un documento
    - per ciascun elemento, quali sono gli elementi figli ed in quale ordine devono apparire
    - se un elemento deve essere vuoto o può contenere testo
    - il tipo di ciascun elemento e ciascun attributo
    - il valore di default per tutti gli elementi ed attributi
  - Vantaggi:
    - sono essi stessi scritti in XML
    - supportano i namespace
    - supportano i tipi di dato
    - sono uno standard W3C
- 

---

## Namespace

- I Namespace servono a:
    - distinguere tra elementi e attributi con lo stesso nome appartenenti ad applicazioni differenti
    - raggruppare assieme tutti gli elementi e gli attributi di una applicazione XML correlati tra loro in modo che il software li possa distinguere facilmente
  - Questa necessità nasce dal fatto che spesso accade che più applicazioni XML risiedono nello stesso documento
-

## Esempi

---

- Documento XHTML con immagini SVG
  - Documento XHTML con equazioni MathML
  - RDF + applicazioni risorsa
  - XSL + documento origine
  - Documento DocBook con elementi XHTML
- 

## Implementazione

---

- I Namespace vengono implementati aggiungendo un prefisso al nome di ogni elemento e attributo
  - Definiscono degli *scope* all'interno del documento
  - In questo modo all'interno di un documento XML si possono avere elementi riguardanti più applicazioni XML
-

## Prefisso

- La sintassi usata per introdurre un prefisso Namespace è la seguente:  
<prefisso:nome\_elemento>
- Per cui si può distinguere tra:
  - rdf:description
  - ex:description
  - description

## Uso degli URL

- I prefissi eliminano l'ambiguità tra elementi con lo stesso nome
- E' obbligatorio assegnare ad ogni prefisso un URL che identifica la particolare applicazione di riferimento
- I prefissi vengono legati agli URL dei Namespace aggiungendo un attributo xmlns:prefisso all'elemento base della gerarchia

## Esempi

- `<rdf:RDF xmlns:rdf="http://www.w3.org/TR/REC-rdf-syntax#">`
- `<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`

## L'URL

- La stringa specificata come valore dell'attributo `xmlns:prefisso` ha la forma di un URL internet.
- In realtà gli URL sono degli identificatori puramente formali
- Non è necessario che la pagina specificata dall'URL esista realmente
- Tuttavia è buona norma quella di inserire all'URL specificato la descrizione della grammatica o il DTD dell'applicazione di riferimento

---

## Namespace e DTD

- I namespace sono completamente indipendenti dai DTD
  - Un documento può essere dotato di DTD ma non utilizzare i namespace
  - Oppure può usare i namespace ma non avere un DTD
  - I namespace non modificano la sintassi dei DTD in nessuna maniera
- 

---

## XSLT

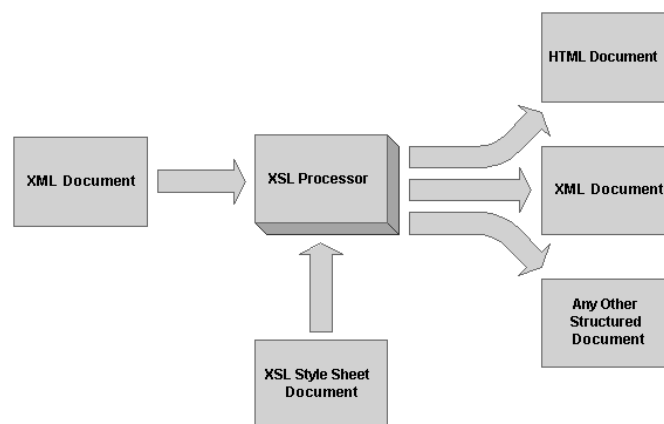
- Uno dei vantaggi nell'utilizzo del XML è la facilità con cui si possono trasformare i documenti in altri formati e strutture.
  - Lo strumento utilizzato per definire queste trasformazioni è denominato Extensible Stylesheet Language Transformations (XSLT).
-

## Foglio di stile

- Un insieme di regole che permettono di trasformare un documento in un altro documento si chiama **foglio di stile**
- Un foglio di stile si definisce mediante delle regole XSL
- XSL è una applicazione XML

## Processo di trasformazione

- La trasformazione avviene tramite un processore chiamato elaboratore XSLT





## Applicazioni pratiche

- Web
  - Un documento XML può essere trasformato in una pagina HTML in modo da essere visualizzato tramite browser Web
- XSL-FO
  - Un documento può essere trasformato in formato FO (Formatting Object) che rappresenta una applicazione di formattazione per documenti testuali.

```
<CatalogoLibri>
  <Libro lingua="Inglese">
    <Autore>Antwone Quenton Fisher</Autore>
    <Titolo>Who Will Cry for the Little Boy</Titolo>
    <Editore>Hardcover</Editore>
    <Prezzo valuta="euro">9.95</Prezzo>
    <Disponibilita>
      <immediata/>
    </Disponibilita>
  </Libro>
  <Libro lingua="Inglese">
    <Autore>Kahlil Gibran</Autore>
    <Titolo>The Prophet </Titolo>
    <Editore>Hardcover</Editore>
    <Prezzo valuta="euro">10.50</Prezzo>
    <Disponibilita>
      <giorno/>
    </Disponibilita>
  </Libro>
  <Libro lingua="inglese">
    <Autore>Samuel Beckett</Autore>
    <Titolo>Waiting for Godot</Titolo>
```

```

<xsl:template match="CatalogoLibri">
  <html>
    <head>
      <title>Catalogo Web Semplice</title>
    </head>
    <body>
      <h1>Catalogo di Libri</h1>
      <ul type="square">
        <xsl:apply-templates select="Libro"/>
      </ul>
    </body>
  </html>
</xsl:template>
<xsl:template match="Libro">
  <li><p>
    Titolo: <xsl:apply-templates select="Titolo"/><br/>
    Autore: <xsl:apply-templates select="Autore"/><br/>
    Editore: <xsl:apply-templates select="Editore"/><br/>
    <b>Prezzo:</b> <xsl:apply-templates select="Prezzo"/>
  </p></li>
</xsl:template>
<xsl:template match="Titolo">
  <b><xsl:value-of select="text()"/></b>
</xsl:template>

```

```

<xsl:template match="CatalogoLibri">
  <html>
    <head>
      <title>Catalogo Web con Tabelle</title>
    </head>
    <body>
      <h1>Catalogo di Libri</h1>
      <table border="1" width="50%">
        <tbody>
          <tr>
            <xsl:apply-templates select="Libro"/>
          </tr>
        </tbody>
      </table>
    </body>
  </html>
</xsl:template>
<xsl:template match="Libro">
  <tr><xsl:apply-templates select="Titolo"/></tr>
  <tr><xsl:apply-templates select="Autore"/>
  <xsl:apply-templates select="Editore"/></tr>
  <tr><xsl:apply-templates select="Prezzo"/>
  <xsl:apply-templates select="Disponibilita"/></tr>
</xsl:template>

```

## Namespace XSL

- XSL è una applicazione XML
- XSL definisce un proprio DTD e un Namespace
- L'URL da usare per il namespace è  
`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
- Il namespace viene generalmente associato al prefisso 'xsl'

## xsl:stylesheet

- L'elemento radice di un documento XSL è `xsl:stylesheet`
- All'interno dell'elemento `xsl:stylesheet` si mettono le regole di trasformazione
- Il più semplice foglio di stile XSL prevede che non vi siano regole di trasformazione

## Catalogo persone

```
<catalogo>
  <persona>
    <nome>Mario</nome>
    <cognome>Bianchi</cognome>
    <origine>
      <data_di_nascita>
        <giorno>13</giorno>
        <mese>6</mese>
        <anno>1976</anno>
      </data_di_nascita>
      <luogo_di_nascita>Milano</luogo_di_nascita>
    </origine>
    <professione da="Luglio 2004">Ingegnere</professione>
  </persona>
  <persona>
    <nome>Marco</nome>
    <cognome>Rossi</cognome>
    <origine>
      <data_di_nascita>
        <giorno>4</giorno>
        <mese>7</mese>
        <anno>1982</anno>
      </data_di_nascita>
    </origine>
  </persona>
  . . . . .
```

## Foglio di stile vuoto

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

- In assenza di regole la trasformazione è la seguente:
  - Viene prelevato il testo interno agli elementi del documento di input
  - Viene quindi riprodotto il contenuto ma non il markup
  - In questo caso il risultato non è un documento XML

## Modelli e regole

- Per controllare il modo in cui viene generato l'output in base all'input bisogna aggiungere dei modelli all'interno del foglio di stile
- Ogni modello è rappresentato da un elemento `xsl:template`, dotato di un attributo `match`
  - `xsl:template` definisce una regola di trasformazione
  - `match` identifica il tipo di input che attiva la regola
- esempio: `persona` → "Una persona"

## Esempio

- Di seguito viene mostrata la regola corrispondente a:
  - `persona` → "Una persona"

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="persona">Una persona
</xsl:template>
</xsl:stylesheet>
```

**Output:**  
Una personaUna persona

## Associamo uno stile HTML

- E' possibile fare in modo che l'output sia un documento XML ben formato

```
<xsl:stylesheet version="1.0" xmlns:xsl="">
  <xsl:template match="persona">
    <p>Una persona</p>
  </xsl:template>
</xsl:stylesheet>
```

### Output:

```
<p>Una persona</p><p>Una persona</p>
```

## Valore di un elemento

- L'elemento xsl:value-of permette di selezionare il contenuto di un elemento di input e inserirlo come output
- xsl:value-of si può trovare soltanto all'interno di un elemento xsl:template
- L'attributo select serve a selezionare il valore che deve essere prelevato

## Esempio

- Di seguito viene riportato un esempio di elemento `xsl:value-of`:

```
<xsl:stylesheet version="1.0" xmlns:xsl="">
<xsl:template match="persona">
  <p>Una persona: <xsl:value-of select="nome"/></p>
</xsl:template>
</xsl:stylesheet>
```

### Output:

```
<p>Una persona: Mario</p><p>Una persona: Marco</p>
```

## select

- Nell'esempio mostrato l'attributo `select` di `xsl:value-of` era impostato come di seguito:
  - `select="nome"`
  - Dove "nome" è l'attributo del quale si vuole estrarre l'informazione.
- Più in generale il valore di `select` deve essere una espressione XPath

## Applicazione forzata di modelli

- L'utilizzo della regola `xsl:apply-templates` permette di stabilire quali modelli devono essere applicati e in che ordine
- `xsl:apply-templates` possiede un attributo `select` che specifica una regola di tipo `xsl:template`

## Esempio

```
<xsl:stylesheet version="1.0" xmlns:xsl="...">
  <xsl:template match="persona">
    <p>
      <xsl:apply-templates select="cognome"/>
      <xsl:apply-templates select="nome"/>
    </p>
  </xsl:template>
  <xsl:template match="nome">
    Nome: <xsl:value-of select="text()"/><br/>
  </xsl:template>
  <xsl:template match="cognome">
    Cognome: <xsl:value-of select="text()"/><br/>
  </xsl:template>
</xsl:stylesheet>
```



## select

- A differenza del primo esempio in cui l'attributo `select` di `xsl:value-of` era impostato come:
  - `select="nome"`
- In questo esempio si ha:
  - `select="text()"`
  - Dove `"text()"` è una espressione XPath che sta ad indicare che dell'elemento corrente (nell'esempio si trattava del nome o del cognome) deve essere prelevato il testo in esso contenuto.

## Output

```
<p>
    Cognome: Bianchi<br/>
    Nome: Mario<br/>
</p>
<p>
    Cognome: Rossi<br/>
    Nome: Marco<br/>
</p>
```

## Intestazione HTML

```
<xsl:template match="catalogo">
  <html>
    <head>
      <title>Catalogo persone</title>
    </head>
    <body>
      <xsl:apply-templates select="persona"/>
    </body>
  </html>
</xsl:template>
```

## Output

```
<html>
  <head>
    <title>Catalogo persone</title>
  </head>
  <body>
    <p>
      Cognome: Bianchi<br/>
      Nome: Mario<br/>
    </p>
    <p>
      Cognome: Rossi<br/>
      Nome: Marco<br/>
    </p>
  </body>
</html>
```

Domande?

