Consiglio Nazionale delle Ricerche

ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

# XSL

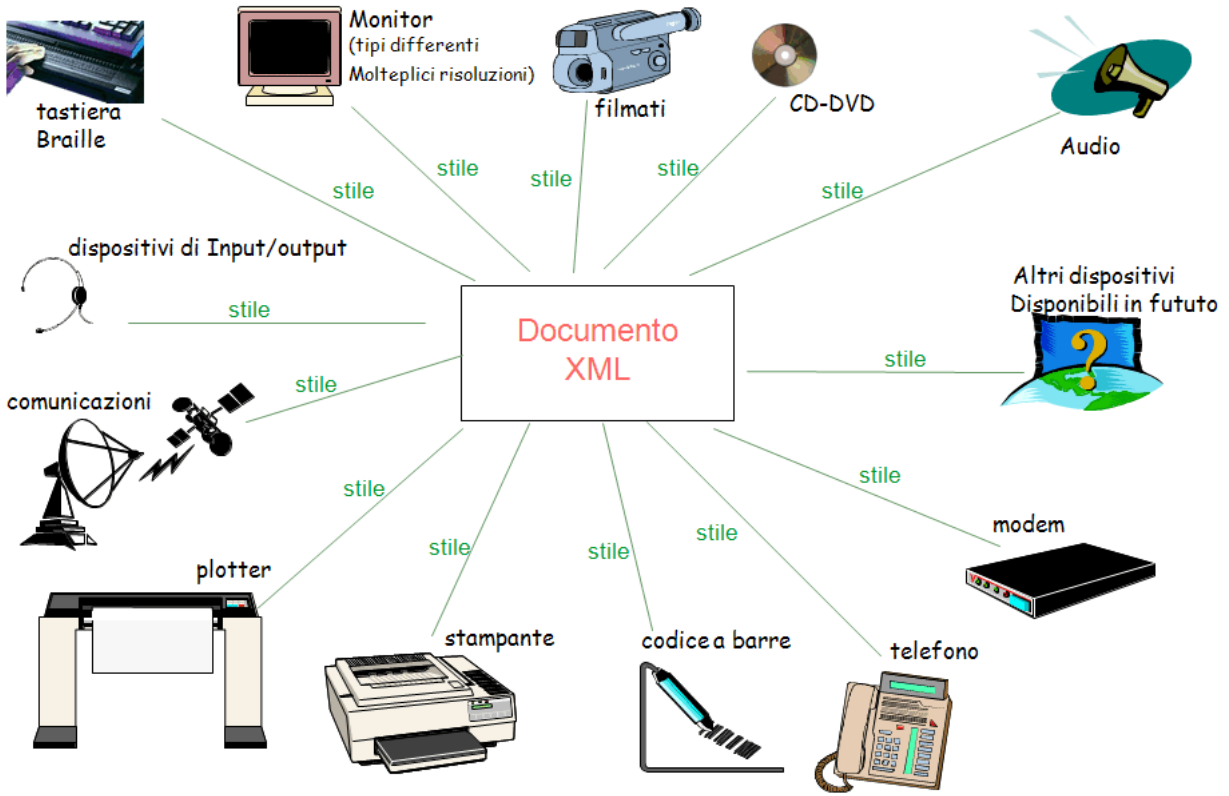
Massimo Martinelli
massimo.martinelli AT isti.cnr.it
Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche
via G. Moruzzi, 1 - 56124 Pisa

Corso InformaticaUmanistica CODIFICA DI TESTI

Pisa 10 Dicembre 2018

# Le componenti di XML

# Visualizzazione di markup XML



# Esempio d'uso di CSS

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="biblioteca.css"?>
<biblioteca>
  <libro codice="R414">
    <titolo>2001: Odissea nello spazio</titolo>
    <autore>
      <cognome>Clarke</cognome>
      <nome>Arthur Charles</nome>
    </autore>
    <editore>Rizzoli</editore>
    <parola_chiave>romanzo</parola_chiave>
    <parola_chiave>fantascienza</parola_chiave>
  </libro>
</biblioteca>
```


```
titolo { display: block;
         text-align: center;
         background: blue;
         color: white;
         font-family: Arial;
         font-size: 20pt
       }
autore { display: block;
         margin-left: 10%;
         text-align: left;
         color: red;
         font-family: Arial;
         font-style: italic;
         font-size: 14pt
       }
cognome, nome { display: inline; }
editore { display: block;
          margin-left: 15%;
          color: green;
          font-family: Arial;
          font-size: 14pt
        }
parola_chiave { display: block;
                margin-left: 5%;
                color: black;
                font-family: "Times New Roman";
                text-align: justify;
                font-size: 14pt
              }
```

# CSS non è sufficiente

- Anche se ogni nuova versione di CSS aggiunge funzionalità
- questi Stylesheet sono comunque limitati:
  - *non consentono modifiche al documento!*
- eXtensible Stylesheet Language (XSL), aggiunge questa ed altre funzionalità

# eXtensible Stylesheet Language (XSL)

## Costituito da 3 parti:

- XSL-T(ransformations) linguaggio di trasformazione
- XPath linguaggio per indirizzare parti di documenti XML
- Formatting Objects vocabolario di formattazione

## W3C Recommendation

- XSL Transformations (XSLT) - *Version 3.0 W3C Recommendation 8 June 2017*
- XML Path Language (XPath) - *Version 3.1 W3C Recommendation 21 March 2017*
- XSL Formatting Objects (FO) - *dizionario*

World Wide Web Consortium Process Document

# A cosa serve XSLT

## Consente di trasformare documenti XML in un altro formato:

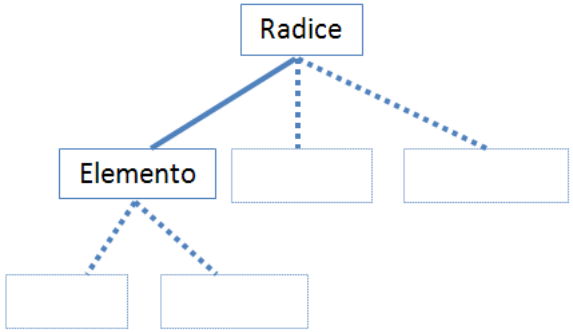
- da XML a XML (HTML, XHTML, Formatting Objects, SVG, ...)
- da XML a testo (privo di markup XML, ma non di altro markup: es PDF, RTF, ...)

### Applicazioni

- che producono documenti XML:
  gestione di documenti XML che contengono lo stesso tipo di informazioni ma in un formato differente
- che ricevono (messaggi) XML (ad esempio un Web Service)
  trasformano in un formato unico (merge) e registrano/visualizzano/inviano ad un altra applicazione

# Premessa: Alberi

- Ogni documento XML ben formato può essere visto come un albero

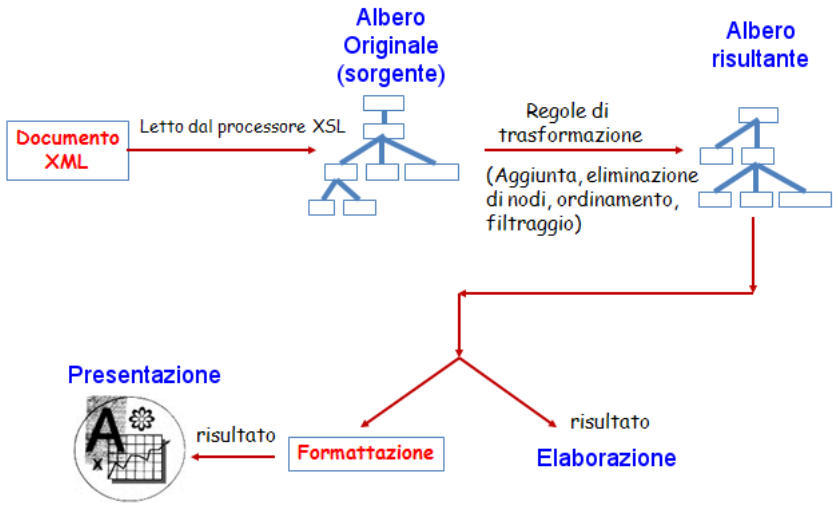



## Nodi dell'albero:

- Radice
- Elementi
- Testo
- Attributi
- Namespace
- Istruzioni di processo
- Commenti
- Utile proprietà degli alberi:
  - *ogni elemento con i propri figli costituisce un albero*

# Il processo di trasformazione

- Una trasformazione espressa in XSLT descrive regole per trasformare zero o più alberi sorgente in uno o più alberi risultanti




- Il processo può prevedere degli alberi temporanei per gestire risultati intermedi
- può inoltre produrre più alberi finali come risultato

# Trasformazioni

## Lato client

- Browser applica lo stylesheet

## Lato server

- servlet o cgi applicano lo stylesheet

## Programma

- La trasformazione avviene prima che il file venga messo a disposizione

# Due esempi

- province.xml - province.xsl
- province.pdf - province.xml - trasformainpdf.xsl

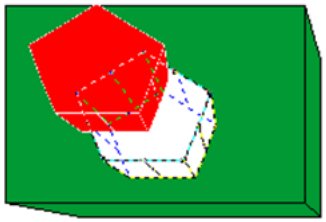
# Funzionamento

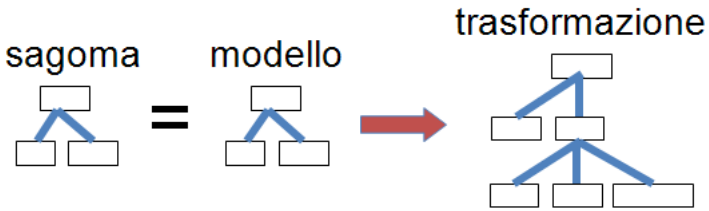
## Regole XSL-T

# Template (sagoma)

- Il processore scorre i nodi di cui è composto il documento XML
- Quando un albero (nodo o insieme di nodi) combacia (match) con il modello (pattern) di una regola di template (sagoma) viene applicata la trasformazione specificata

```
<xsl:template match="modello">
... trasformazione ...
</xsl:template>
```





# Esempio di template

```
    <xsl:template match="nodo">
         testo <tag> <xsl:value-of select="." /> </tag>
    </xsl:template>
```

Se un nodo del documento XML combacia con quello della regola dell'esempio viene prodotto come risultato

```
testo  <tag>  valore (contenuto) dell'elemento corrente </tag>
```

---

Dato il seguente frammento XML

```
<nominativo>Mario Rossi</nominativo>
```

e il il seguente frammento XSLT

```
<xsl:template match="nominativo">
       Nominativo: <strong><xsl:value-of select="."/></strong>
</xsl:template>
```

Risultato trasformazione:

```
Nominativo:  <strong>Mario Rossi</strong>
```

Visualizzato da un browser in questo modo:

```
Nominativo:  Mario Rossi
```

# i nodi?

Abbiamo visto che una regola di template specifica un nodo come sagoma

ma come si specificano i nodi ? ...

# XPath

- Linguaggio che tramite una espressione (una stringa) specifica un insieme di nodi,
  *un percorso per localizzare* (location-path) questi nodi
- Utilizzato da XSL-T per selezionare i nodi da elaborare
- Parte della sintassi simile a quella adottata dai sistemi operativi per individuare un file all'interno di una gerarchia di directory
  **`/Corsi/XSL/index.xml`**
- Le espressioni possono restituire i segg. tipi di oggetti:
  - *Insiemi di nodi*
  - *booleani (true, false)*
  - *numeri (interi e decimali positivi e negativi)*
  - *stringhe ("questa è una stringa", 'anche questa')*
  - *frammenti dell'albero risultante*
- XPath fornisce alcune funzioni di base per manipolare i tipi di oggetti sopra elencati

# Contesto

- In XPath ogni oggetto è interpretato in riferimento ad un *contesto*
- Il contesto:
  - *in generale corrisponde al nodo dell'albero sorgente a partire dal quale viene valutata una espressione*
  - *costituito da 6 parti*
    1. il nodo di contesto (una similitudine: la directory corrente)
    2. la posizione di contesto (es: dato un elenco `<li>` quale oggetto nell'elenco, il 1°, o il 2°, ...)
    3. la dimensione di contesto (es: dato un elenco quanti oggetti ne fanno parte)
    4. l'insieme di variabili visibili
    5. le funzioni disponibili
    6. l'insieme di namespace visibili

# Passi di un percorso di localizzazione

- Percorso: serve a localizzare insiemi di nodi (node-sets) contenuti all'interno di documenti XML
- Un *percorso di localizzazione* può essere scomposto in uno o più *passi*

```
esempio:  biblioteca/libro/titolo              3 passi separati dal carattere "/"
```

- Può essere un assoluto o relativo:
  - *Assoluto: inizia con il simbolo "/"* **`/biblioteca/libro`**
  - *Relativo: non inizia con il simbolo "/ "* **`libro/titolo`**
- Un passo di un percorso può essere scomposto in 3 ulteriori parti

```
        asse::nodo di test [predicati]                  (zero o più predicati)
```

- Esempio:
  child::anagrafe [ position() = 3 ]
  asse node-test predicato
  Individua il 3° elemento "anagrafe", figlio del nodo contesto

# Esempi (sintassi non abbreviata) #1

- **`child::para`** seleziona tutti gli elementi para figli del nodo di contesto (NDC)
- **`child::*`** tutti gli elementi figli del NDC
- **`child::text()`** tutti i nodi text figli del NDC
- **`child::node()`** tutti i figli del NDC, qualsiasi tipo di nodo siano
- **`attribute::name`** il nome dell'attributo del NDC
- **`attribute::*`** tutti gli attributi del NDC
- **`descendant::para`** gli elementi para discendenti del NDC
- **`ancestor::div`** tutti gli elementi div avi del NDC
- **`ancestor-or-self::div`** come precedente più il NDC stesso se è un elemento div
- **`descendant-or-self::para`** tutti gli elementi para discendenti compreso il NDC se è un elemento para
- **`self::para`** seleziona il NDC se è un elemento para (niente altrimenti)
- **`child::capitolo/descendant::para`** gli elementi para discendenti dell'elemento capitolo figlio del NDC
- **`child::*/child::para`** gli elementi para nipoti del NDC

# Esempi (sintassi non abbreviata) #2

- `/` l'elemento radice del documento (sempre genitore del "document element")
- `/descendant::olist/child::item` tutti gli elementi item che hanno un padre olist e che sono nello stesso documento del NDC
- `child::para[position()=1]` il primo figlio para del NDC
- `child::para[position()=last()-1]` il penultimo elemento para figlio del NDC
- `child::para[position()>1]` tutti i figli para del NDC eccetto il primo
- `following-sibling::capitolo[position()=1]` il primo elemento capitolo fratello successivo del NDC
- `preceding-sibling::capitolo[position()=1]` il primo elemento capitolo fratello precedente del NDC
- `/child::doc/child::capitolo[position()=5]/child::sezione[position()=2]` il secondo elemento del 5° capitolo dell'elemento doc figlio dell'elemento radice
- `child::para[attribute::tipo='attenzione'][position()=5]` il 5° figlio para del NDC che ha un attributo tipo con il valore attenzione
- `child::para[position()=5][attribute::type="warning"]` identico al precedente
- `child::capitolo[child::titolo='Introduzione']` i figli capitolo del NDC che hanno uno o più figli titolo con valore uguale a Introduzione
- `child::capitolo[child::titolo]` i figli capitolo del NDC che hanno uno o più figli titolo
- `child::*[self::capitolo or self::appendice][position()=last()]` l'ultimo capitolo o appendice figli di NDC

# Esempi (sintassi abbreviata)

- `para` seleziona gli elementi para figli del NDC
- `*` tutti gli elementi figli del NDC
- `text()` tutti i nodi testo figli del NDC
- `@nome` l'attributo nome del NDC
- `@*` tutti gli attributi del NDC
- `para[1]` il primo elemento para figlio del NDC
- `para[last()]` l'ultimo nodo
- `*/para` tutti gli elementi para nipoti del NDC
- `/doc/capitolo[5]/sezione[2]` il 2° figlio sezione del 5° figlio capitolo dell'elemento doc figlio della radice
- `chapter//para` tutti gli elementi para discendenti dell'elemento capitolo figlio del NDC
- `//para` tutti gli elementi para discendenti dell'elemento radice (stesso documento del NDC)
- `//olist/item` tutti gli elementi item figli di olist discendenti della radice item
- `.` il nodo di contesto
- `.//para` gli elementi para discendenti del NDC
- `..` il genitore del NDC
- `../@lang` l'attributo lang del genitore del NDC
- `para[@tipo="attenzione"]` tutti i figli para che hanno un attributo tipo con valore attenzione
- `para[@tipo="attenzione"][5]` il 5° figlio para del NDC con un attributo tipo con valore attenzione
- `para[5][@tipo="attenzione"]` identico al precedente
- `capitolo[titolo="Introduzione"]` il figlio capitolo figlio del NDC che ha almeno un elemento figlio titolo con valore uguale a Introduzione
- `capitolo[titolo]` tutti i figli del NDC capitolo con almeno un figlio titolo
- `impiegato[@codice and @tipo]` tutti i figli del NDC impiegato con un attributo codice e uno tipo

# Asse

Specifica la *direzione* verso cui spostarsi a partire dal nodo di contesto

- **`ancestor`** corrisponde agli elementi avi del nodo corrente
- **`ancestor-or-self`** avi e nodo corrente
- **`attribute`** attributo del nodo corrente (abbreviato con "@")
- **`child`** figli del nodo corrente (asse di default)
- **`descendant`** tutti i discendenti del nodo corrente
- **`descendant-or-self`** discendenti e nodo corrente
- **`following`** tutti i nodi che seguono quello corrente (in ordine di apparizione all'interno del documento)
- **`following-sibling`** nodi fratelli successivi al nodo corrente
- **`namespace`** i nodi namespace del nodo corrente
- **`parent`** il genitore del nodo corrente (abbreviato con "..")
- **`preceding`** i nodi che precedono quello corrente (in ordine di apparizione all'interno del documento)
- **`preceding-sibling`** nodi fratelli precedenti al nodo corrente
- **`self`** il nodo corrente (abbreviato con ".")

# Test sui nodi

Il *Node-Test* determina il tipo di nodo da selezionare

- **`nome`** combacia con i nodi elementi <nome>
- **`*`** qualsiasi nodo di tipo elemento
- **`namespace:nome`** ogni elemento <nome> con il namespace specificato
- **`namespace:*`** qualsiasi elemento con il namespace specificato
- **`comment()`** i nodi di tipo commento
- **`node()`** qualsiasi nodo
- **`text()`** I nodi di tipo testo
- **`processing-instruction()`** le istruzioni di processo

# XPath: Predicati

- Un predicato *filtra* un insieme di nodi rispetto ad un asse per produrre un nuovo insieme di nodi
  - *Se l'espressione è true il nodo viene inserito nel nuovo insieme di nodi*
  - *Se l'espressione restituisce un numero e questo numero corrisponde alla posizione del nodo di contesto nella lista dei nodi questo nodo viene inserito nel nuovo insieme*
  - *Se restituisce una stringa non vuota il nodo di contesto viene inserito nel nuovo insieme*

Esempi

- **`nodetest[1]`** combacia con il primo nodo
  I test sui nodi restituiscono i nodi secondo l'ordine nel documento, ma nel caso degli avi (ancestor) o precedenti (preceding) l'ordine è ovviamente inverso. Il primo nodo è sempre quello più vicino al nodo di contesto
- **`nodetest[position()=last()]`** l'ultimo nodo
- **`nodetest[position() mod 2 = 0]`** nodo pari
- **`element[@id='val']`** elementi con un attributo id con valore "val"
- **`element[not(@id)]`** elementi che non hanno un attributo id
- **`persona[nome="Mario"]`** elementi <persona> che hanno un elemento figlio nome con valore "Mario".
- **`persona[normalize-space(nome)="Mario"]`** identico al precedente senza tenere di conto gli spazi

# Predicati - Insiemi di nodi - Booleani

Insiemi di nodi

- **`last()`** restituisce il numero di nodi in un insieme di nodi
- **`position()`** posizione del nodo di contesto
- **`count(node-set)`** il numero di nodi
- **`local-name(node-set)`** il nome locale (senza namespace) del primo nodo
- **`namespace-uri(node-set)`** URI del namespace del primo nodo
- **`name(node-set)`** il nome qualificato (con namespace) del primo nodo

Booleani

- **`!=`**
- **`<`**
- **`<=`**
- **`=`**
- **`>`**
- **`>=`**
- **`and`**
- **`or`**
- **`true()`**
- **`false()`**

Esempio:

```
<xsl:template match="libro[position() > 2]">
...
</xsl:template>
```

# Predicati: numeri e stringhe

Numeri

- `+` Addizione
- `-` Sottrazione
- `*` Moltiplicazione
- `div` Divisione
- `mod` Modulo
- `ceiling()` Intero più grande
- `floor()` Intero più piccolo
- `round()` Intero più vicino (arrotondamento)
- `sum()` Somma di numeri

Stringhe

- `starts-with(string string1, string string2)` - Boolean
- `contains(string string1, string string2)` - Boolean
- `substring(string string1, number offset, number length)` - String
- `substring-before(string string1, string string2)` - String
- `substring-after(string string1, string string2)` - String
- `string-length(string string1)` - Number
- `normalize-space(string string1)` - String
- `translate(string string1, string string2,string string3)` - String
- `concat(string string1, string string2,...)` - String

# Esempio di trasformazione di XML in HTML

Documento XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<saluto>
        Benvenuto nel mondo di XSL-T
</saluto>
```

Documento XSL-T

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

<xsl:template match="/">
   <html>
     <head><title>Saluto</title></head>
     <body>
          <xsl:apply-templates select="saluto"/>
     </body>
   </html>
</xsl:template>

<xsl:template match="saluto">
     <xsl:value-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

# Risultato della trasformazione

*Associare lo stile aggiungendo nel file xml alla 2a riga*

```
<?xml-stylesheet href="esempio3_documento.xsl" title="stile" type="text/xsl"?>
```

*e aprire il file per esempio con Firefox e osservare con firebug oppure con Opera*

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>  <-- aggiunta dal processore
<title>Saluto</title>
</head>
<body>
   Benvenuto nel mondo di XSL-T
</body>
</html>
```

# Elementi di XSL-T

- **`<xsl:template match="/">`**
  - *Template per l'elemento radice*
  - *indica come trasformare l'elemento radice*
- **`<xsl:template match="x">`**
  - *Template per tutti gli elementi "x"*
  - *indica come trasformare l'elemento "x"*

Nel successivo esempio vedremo questo template

- **`<xsl:template match="n/x">`**
  - *Template per tutti gli elementi "x" figli di elementi "n"*
  - *indica come trasformare gli elementi n/x*

# Esempio di trasformazione di XML in HTML

Documento XML

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<documento>
        <autore>Mario Rossi</autore>
</documento>
```

Documento XSL-T

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

<xsl:template match="/">
   <html>
     <head><title>Autore</title></head>
     <body>
          <xsl:apply-templates select="documento/autore"/>
     </body>
   </html>
</xsl:template>

<xsl:template match="autore">
     <xsl:value-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

# Risultato della trasformazione

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Autore</title>
</head>
<body>Mario Rossi</body>
</html>
```

# Elementi di XSL-T

- **`<xsl:apply-template select="x">`**
  - *Applica le regole di trasformazione relative a tutti gli elementi "x" contenuti nell'elemento corrente*
- **`<xsl:value-of select=".">`**
  - *Restituisce il valore dell'elemento corrente*

# Un altro esempio

Documento XML

```
<?xml version="1.0" encoding="utf-8" ?>
<documento>
        <autore>Mario Rossi</autore>
</documento>
```

Documento XSLT

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">
        <xsl:template match="autore">
                <div>Autore: <strong> <xsl:apply-templates /> </strong></div>
        </xsl:template>
</xsl:stylesheet>
```

Applica i template relativi agli elementi figli (ove ve ne siano)
...e non solo...

# Il risultato della trasformazione

./esempiCorso2010/esempio3_risultato.xml

```
<div>
Autore:
<strong>Mario Rossi</strong>

</div>
```

Attenzione: non funziona come ci aspetteremmo perché esistono delle regole predefinite...

p.s. togliamo i due tag ** ** e ** ** e controlliamo cosa fanno firefox e opera:

- il primo aggiunge un elemento **<result>** che include il risultato
- il secondo riporta un errore interpretando il codice come XHTML e non avendo incluso il risultato in un elemento block)

# Esempio di template

- Perché il processore ha prodotto in output il contenuto dell'elemento autore ?
- Perché non avendo specificato un template per il contenuto degli elementi viene usato un template di default
- Anche apply-templates senza select fa sì che vengano applicati i template di default:
  applica i template per gli elementi, testi, commenti e istruzioni di processo contenuti nell'elemento corrente (tutti i nodi che sono figli dell'elemento corrente)

```
<xsl:template match="nodo">
        testo <tag> <xsl:apply-templates /> </tag>
</xsl:template>
```

- Se un nodo del documento XML combacia con quello della regola dell'esempio viene prodotto come risultato

```
 testo  <tag>
   valore dell'elemento corrente                        <--regola di default
   applicazione delle altre regole di template
</tag>
```

# Regole XSLT precostituite, per difetto (default)

- XSL-T usa i template per specificare come devono essere trasformati gli elementi di un documento XML
- Il processore inizia l'elaborazione partendo dal template dell'elemento radice
- Ogni elemento riferito dal template dell'elemento radice sarà elaborato a sua volta
- Esistono 5 template di default, uno per ogni tipo di nodo, nel caso nessun template combaci un qualsiasi tipo di nodo viene applicata la regola di default, o anche se usiamo apply-templates senza attributo "select"

```
1.   <xsl:template match="/ | *">                    Applica le regole per qualsiasi elemento
         <xsl:apply-templates/>
      </xsl:template>

2.   <xsl:template match="text()">                   per qualsiasi testo
          <xsl:value-of select="."/>
     </xsl:template>

3.  <xsl:template match="@*">                        La regola per gli attributi è "ingannevole"
        <xsl:value-of select="."/>                    Se non specifichiamo il template per gli attributi
    </xsl:template>                                  non vengono prodotti in output i valori

4.   <xsl:template match="comment()"/>                Non applicare le regole per i  commenti

5.   <xsl:template match="processing-instruction()"/>  Ne per le istruzioni di processo
```

# Esempio minimale #1

Dato il seguente file XML ...

```
<?xml version="1.0"?>
<biblioteca>
   <libro codice="R415">
        <titolo>Destinazione cervello</titolo>
        <autore>
                <cognome>Asimov</cognome>
                <nome>Isaac</nome>
        </autore>
        <editore>Mondadori</editore>
        <parola_chiave>romanzo</parola_chiave>
        <parola_chiave>fantascienza</parola_chiave>
   </libro>
      <libro codice="N516">
        <titolo>Sostiene Pereira</titolo>
        <autore>
                <cognome>Tabucchi</cognome>
                <nome>Antonio</nome>
        </autore>
        <editore>Feltrinelli</editore>
        <parola_chiave>romanzo</parola_chiave>
        <parola_chiave>narrativa</parola_chiave>
   </libro>
</biblioteca>
```

# Esempio minimale #2

... e il seguente stylesheet minimale

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

</xsl:stylesheet>
```

# Esempio minimale #3

il risultato della trasformazione:

```
<?xml version="1.0" encoding="UTF-8"?>   <--Prologo XML



        Destinazione cervello

                Asimov
                Isaac

        Mondadori                   <-- contenuto degli elementi senza i relativi tag
        romanzo
        fantascienza


        Sostiene Pereira

                Tabucchi
                Antonio

        Feltrinelli                 <-- Attenzione agli spazi prodotti!!!
```

```
romanzo                    Il secondo template di default ha
narrativa                  prodotto in output anche gli spazi
```

# Sostituzione delle regole precostituite

```
<xsl:template match="nodo">
</xsl:template>
```

Non applica alcuna trasformazione per il nodo specificato

```
<xsl:template match="*|text()" />
```

Non applica alcuna per gli elementi e per i testi (a meno di regole più specifiche)

# Tipo di Output

- XML per difetto
  ```
  <xsl:output method="xml"/>
  ```
- HTML
  ```
  <xsl:output method="html"/>
  ```
- Testo
  ```
  <xsl:output method="text"/>
  ```
- Inserimento di spazi e indentazione
  ```
  <xsl:output indent="yes"/>
  ```
- Specifica del DOCTYPE nel documento prodotto
  con
  ```
  <xsl:output doctype-system="nomefile.dtd"/>
  ```
  si ottiene
  ```
  <!DOCTYPE elementoradice SYSTEM "nomefile.dtd">
  ```

# apply-templates

```
<xsl:apply-templates/>
```

- Indica di avviare l'elaborazione in modo ricorsivo su tutti i figli di un nodo, controllando ogni regola che possa essere applicata.

```
<xsl:apply-templates select="nodo" />
```

- L'attributo select permette di selezionare il nodo di partenza per l'elaborazione ricorsiva

# Output XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"
    doctype-public="-//W3C//DTD XHTML 1.1//EN"
    doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>

<xsl:template match="/">
   <html xmlns="http://www.w3.org/1999/xhtml">
     <head>
       <title>esempio di output XHTML</title>
     </head>
     <body>
        <xsl:apply-templates />
     </body>
     </html>
</xsl:template>
  ...
</xsl:stylesheet>
```

# Esempio

Concorso CNR 2017

Problema: ottenere elenco pubblicazioni seguendo le regole del concorso

2018: Martinelli M., Tampucci M. - Ottenere l'elenco delle pubblicazioni CNR con 'stile' - Nota tecnica aperta.

file pubblicazioni scaricato dal sito People CNR

file XSLT

# Esercizio

Dato il seguente documento XML:

```
<?xml version="1.0"?>
<brano id="5">
        <titolo>Asturias<titolo/>
        <autore>Andres Segovia</autore>
        <incisione>Deutsche Grammophon</incisione>
</brano>
```

Trasformare in XHTML 1 (strict) in modo da produrre questo risultato:

```
Titolo brano: Asturias
Identificatore: 5
Incisione: Deutsche Grammophon
```

# Risoluzione dei Conflitti - Precedenza tra template

- Se esistono più template che combaciano con lo stesso nodo, prevale quello più specifico
- Questo: <xsl:template match="libro/autore/cognome">
  è più specifico di <xsl:template match="cognome">

La priorità di default è determinata come segue:

- Elementi figli o nomi di attributi non qualificati da namespace hanno priorità 0.
- Istruzioni di processo con un target hanno priorità 0.
- Elementi figli qualificati o nomi di attributi qualificati hanno priorità -0.25.
- "**`*`**" non qualificato ha priorità -0.5
- Qualsiasi altro template ha priorità di default 0.5
- La priorità dei template possono essere specificate esplicitamente con l'attributo priority su <xsl:template>

Esempi

- "**`elemento`**", "**`html:p`**", e "**`@attributo`**" hanno priorità 0
- "**`html:*`**" ha priorità -0.25
- "**`*`**" ha priorita -0.5
- "**`elemento1/elemento2`**" ha priorità 0.5
- "**`elemento[@attributo]`**" ha priorità 0.5

# Ricorsività delle regole di template

- L'albero in output viene creato appendendo i nodi elaborati partendo dal nodo radice
- Un nodo viene elaborato cercando tutti le regole di template con il modello corrispondente
- Un template contiene, tipicamente istruzioni che selezionano una nuova lista di nodi dell'albero sorgente da elaborare
- Il processo di matching, l'istanziazione e la selezione di nodi continuano ricorsivamente fino a quando non esistono più nodi dell'albero sorgente da selezionare
- Ci sono due problemi da superare quando si usa il modello ricorsivo, come risolvere il processo quando più modelli combaciano con i nodi e come elaborare gli stessi nodi in contesti differenti

i

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="doc">
  <html><head><title>Titolo del documento</title></head>
  <body><xsl:apply-templates/></body></html>
</xsl:template>

<xsl:template match="para">
  <p><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="em">
  <em><xsl:apply-templates/></em>
</xsl:template>

<xsl:template match="em/em">
  <span><xsl:apply-templates/></span>
</xsl:template>
</xsl:stylesheet>
```

```
<doc>
<para>Questo è un <em>esempio</em>.
<em>Contenuto  <emphasis>nidificato</emphasis></em>.</para>
</doc>
```

```
<html>
<head>
<title>titolo del documento</title>
</head>
<body>
<p>Questo è un <em>esempio</em>.
<em>Contenuto <span>nidificato</span></em>.</p>
</body>
</html>
```

# Modi

A volte può essere necessario includere più volte lo stesso nodo del documento sorgente in quello finale, ovvero applicare più trasformazioni sullo stesso nodo (ad esempio per fare un indice di oggetti presenti nel documento).
I modi permettono di applicare più regole per lo stesso nodo

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

<xsl:output method="html"/>

<xsl:template match="capitolo/titolo">
  <h2><xsl:apply-templates/></h2>
</xsl:template>

<xsl:template match="capitolo/titolo" mode="crossref">
  <em><xsl:apply-templates/></em>                    \
</xsl:template>                                        \
                                                        \
                                                    applica questo
                                                              \
<xsl:template match="xref">                                    \
  <xsl:variable name="linkend" select="@linkend"/>              \
  <xsl:apply-templates select="//*[@id=$linkend]/titolo" mode="crossref"/>
</xsl:template>

<xsl:template match="para">
  <p><xsl:apply-templates/></p>
</xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0"?>
<capitolo id="c1"> <titolo>Titolo capitolo</titolo>
<para>Questo capitolo si auto referenzia:
<xref linkend="c1" /> . </para>
</capitolo>
```

```
<h2>Titolo capitolo</h2>
<p>Questo capitolo si auto referenzia:
<em>Titolo capitolo</em>.</p>
```

# xsl:text

```
<xsl:text>xyz</xsl:text>
```

Restituisce in output il testo xyz

Scrivere xyz all'interno di un template è equivalente

```
<xsl:template match="nome_elemento">
  xyz
</xsl:template>
```

# xsl:comment

Commenti in XSL-T

```
<xsl:comment>commento</xsl:comment>
```

# xsl:for-each

Come abbiamo visto XSL-T permette di elaborare elementi multipli contenuti all'interno di altri elementi mediante l'applicazione di due template:
il primo fa eseguire il secondo al suo interno.

```
<xsl:template match="persona">
      <xsl:apply-templates select="cognome"/>
  </xsl:template>

  <xsl:template match="cognome">
       <xsl:value-of select="."/>
  </xsl:template>
```

Esiste un'altra regola, molto più leggibile, che permette di ottenere lo stesso risultato:
**<xsl:for-each select="el">** Cicla su tutti gli elementi el all'interno dell'elemento corrente

```
<xsl:template match="persona">
     <xsl:for-each select="cognome">
          <xsl:value-of select="."/>
     </xsl:for-each>
</xsl:template>
```

select="." all'interno del for-each seleziona l'elemento selezionato dal for-each (cognome)

# Elementi di controllo:

**condizioni**

# xsl:if (valori booleani) - Elaborazione condizionale ("if" senza "else")

```
<xsl:if test="$condizione">
    questo messaggio apparirà solo se $condizione è true
</xsl:if>
```

- **<xsl:if test="nome">** esiste l'elemento "nome", figlio dell'elemento corrente ?
- **<xsl:if test="count(x)>1">** esiste più di un elemento "x" figlio dell'elemento corrente ?
- **<xsl:if test="$x">** il valore della variabile "x" è true ?

Dividiamo gli autori con una virgola:

```
<xsl:template match="autore">
   <xsl:value-of select="./cognome" />
   <xsl:text>  </xsl:text>
   <xsl:value-of select="./nome" />
   <xsl:apply-templates />
   <xsl:if test="not(position()=last())">, </xsl:if>
</xsl:template>
```

# Dichiarazione di variabili - xsl:variable

- <xsl:variable> permette di associare una variable con una stringa, un elenco di nodi o un frammento dell'albero risultante.
- Le variabili sono ad assegnamento singolo (non posso fare $a=$a+1 posso ovviare a questo problema facendo chiamate ricorsive )
- Le variabili sono visibili all'interno di un campo di visibilità (all'interno dell'elemento padre)

```
<xsl:variable name="var" expr="subelement[position() mod 3]"/>
```

* Notare la dichiarazione di una variabile il cui valore è una espressione matematica

- Generalmente si inserisce la condizione all'interno della variabile:

```
<xsl:variable name="persona">
  <xsl:if test="$nome">...</xsl:if>
</xsl:variable>
```

# apply-template multipli

- E' possibile specificare più di una regola *xsl:apply-templates* all'interno di un template
- Supponiamo di dover fare due elenchi separati per un negozio di noleggio film creando due tabelle: una di dvd, l'altra di videocassette

```
<xsl:template match="film">
  <h1> elenco dvd</h1>
  <table>
    <xsl:apply-templates select="dvd"/>
  </table>
  <h1> elenco videocassette</h1>
  <table>
        <xsl:apply-templates select="cassetta"/>
  </table>
</xsl:template>
```

# xsl:choose - alternative

Supponiamo di dover gestire la prenotazione di lezioni che possono essere di gruppo o individuali

```
<xsl:template match="lezione">
    <h3><xsl:value-of select="titoloLezione"/></h3>
    <h4>Aula: <xsl:value-of select="../../@name"/></h4>
    <h4>Prenotata da : <xsl:value-of select='prenotataDa'/></h4>
    <h4>Allievi:</h4>
    <xsl:for-each select='allievi'>
      <xsl:choose>
        <xsl:when test="gruppo">   <-- Esiste un elemento gruppo figlio dell'elemento corrente ?
            Gruppo: <em><xsl:value-of select="gruppo"/></em><br />
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="individuo"/><br />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
    <p> dalle ore <xsl:value-of select="oriario/da"/>
             alle ore <xsl:value-of select="orario/alle"/></p>
     . . .
</xsl:template>                      ==  
```

# xsl:choose #2

```
<xsl:template match="elemento">
     <xsl:variable name="var" expr="sottoelemento[position() mod 3]"/>
      <xsl:choose>
        <xsl:when test='$var=1'>
          ... fai qualcosa...
        </xsl:when>
      <xsl:when test='$var=2'>
            ... fai un'altra cosa ...
        </xsl:when>
        <xsl:otherwise>
          ... fai qualcosaltro ...
        </xsl:otherwise>
      </xsl:choose>
  <xsl:apply-templates/>
</xsl:template>
```

# xsl:sort

Modificare l'ordine dei libri per autore

```
<xsl:template match="biblioteca">
        <xsl:apply-templates select="libro[code >= 'M']" >
                <xsl:sort select="libro/autore/cognome" />                    Ordina per cognome
                <xsl:sort select="libro//nome" />                                   poi per nome
        </xsl:apply-templates>
</xsl:template>
```

Supponendo di voler modificare l'ordine per prezzo (numerico)

```
<xsl:sort select="libro/prezzo"
                order="descending"          Specifica l'ordine discendente (per difetto = ascending)
                data-type="number" />                Specifica il tipo di dato (per difetto = text)
```

# xsl:number

Numerazione

```
<xsl:template match="libro">
    <xsl:number />
    <xsl:text> </xsl:text>                Inserisce uno spazio tra il numero e il titolo
        <xsl:value-of select="titolo" />
</xsl:template>
```

**xsl:number** inserisce una sequenza di numeri contanto il numero dei fratelli che precedono l'elemento e aggiunge 1

1 Destinazione cervello
2 Sostiene Pereira

# xsl:number format

Indica il normato della numerazione

```
<xsl:number format="tipo_formato">
```

tipo_formato

- "1" genera la sequenza 1 2 ... 10 11 12 ...,
- "01" genera la sequenza 01 02 ... 09 10 11 12 ... 99 100 101.
- "A" genera la sequenza A B C ... Z AA AB AC....
- "a" genera la sequenza a b c ... z aa ab ac....
- "i" genera la sequenza i ii iii iv v vi vii viii ix x ....
- "I" genera la sequenza I II III IV V VI VII VIII IX X ....
- "1." genera la sequenza 1. 2. 3. ...
- "(1)" genera la sequenza (1) (2) ...
- "ア" (#x30A2) specifica la numerazione Katakana
- "イ" (#x30A4) specifica la numerazione Katakana con ordine "iroha"
- "๑" (#x0E51) specifica la numerazione con cifre Thai
- "א" (#x05D0) letter-value="traditional" specifies la numerazione "traditional" Hebrew

- "ა" (#x10D0) letter-value="traditional" specifies la numerazione Georgian
- "α" (#x03B1) letter-value="traditional" specifica la numerazione "classical" Greek
- "а" (#x0430) letter-value="traditional" specifica la numerazione Old Slavic

# Numerazione multipla (xsl:number multiple)

```
<xsl:template match="titolo">
    <xsl:number level="multiple"
      count="capitolo|sezione|sottosezione"
      format="1.1.a" />
        <xsl:apply-templates />
</xsl:template>
```

```
1               Titolo del capitolo
1.1     Titolo della Sezione
1.1.a   Titolo della Sottosezione
1.2     ...
```

Se aggiungo questo:

```
<xsl:template match="capitolo/titolo">
    <xsl:number level="multiple"
      count="capitolo"
      format="1:" />
        <xsl:apply-templates />
</xsl:template>
```

```
Capitolo 1  Titolo del capitolo
1.1     Titolo della Sezione
1.1.a   Titolo della Sottosezione
1.2     ...
```

# xsl:element

**<xsl:element name="nome_elemento">**

Crea un elemento <nome_elemento> e lo restituisce in uscita

```
<xsl:element name="cognome">
        <xsl:value-of select=".">
</xsl:element>
```

Crea un elemento **<cognome>** contenente il valore dell'elemento corrente

# xsl:attribute

**<xsl:attribute name="nome_attributo">**

Crea un attributo nome_attributo su **<xsl:element>** corrente.

Il contenuto di **<xsl:attribute>** diventa il valore dell'attributo creato

```
<indirizzo_sito_web>http://www.cnr.it</indirizzo_sito_web>


<xsl:element name="a">
        <xsl:attribute name="href">
                <xsl:value-of select="indirizzo_sito_web">
        </xsl:attribute>
   il sito web del CNR
</xsl:element>
```

```
<a href="http://www.cnr.it">il sito web del CNR</a>
```

# Id e key

Riferimenti incrociati e funzione id()

```
<xsl:value-of select="id(@refid)/@reftext" />
```

restituisce l'attributo @reftext del nodo che ha un id con un valore equivalente al valore @refid del nodo corrente

solo un attributo deve essere dichiarato ID nella DTD/Schema

*Key* supera questo limite

Ottimo esempio capitolo 5 del libro "XSL-T" O'Reilly

# Invocazione di template ("procedure")

XSL-T mette a disposizione la possibilita di invocare una regola template per nome passandogli dei parametri

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

<xsl:param name="nome" />               // parametro con visibilità globale

<xsl:template name="calcoloArea">
        <xsl:param name="larghezza" />
        <xsl:param name="altezza" />

        <xsl:value-of select="$larghezza * $altezza" />
</xsl:template>
```

```
<xsl:call-template name="calcolaArea">          // invoca il template
        <xsl:with-param  name="larghezza" select="10" />        // passaggio di parametro
        <xsl:with-param  name="altezza" select="20" />  // passaggio di parametro
<xsl:call-template>

</xsl:stylesheet>
```

I template si possono invocare ricorsivamente (call template all'interno dello stesso template che chiama)
Un ottimo esempio è il template Somma riportato nel capitolo 7 sezione 2 del libro "XSL-T" O'Reilly
http://oreilly.com/catalog/orxmlapp/chapter/ch07.html

# xsl:import - xsl:include

XSL-T mette a disposizione due metodi per combinare regole template contenute in documenti distinti: import e include

```
<xsl:import href = "uri-reference" />   href = URI del foglio di stile da importare
```

```
<xsl:import>    importa uno stylesheet esterno
```

Le regole template importate da un'altro documento hanno una priorità inferiore.

```
<xsl:include href = "uri-reference" />
```

**<xsl:include>** include le regole da un altro documento:
Le regole template incluse da un altro documento hanno la stessa priorità di quelle contenute nel primo documento.

Questi due metodi vanno specificati subito dopo il tag **<xsl:stylesheet>**

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="nomefile.xsl"/>
<xsl:include href="nomealtrofile.xsl"/>
...
</xsl:stylesheet>
```

# document

Un foglio di stile può elaborare più di un documento XML caricando altri documenti con la funzione document()

prodotti.xml

```
<?xml version="1.0"?>
<prodotti>
    <prodotto>
         <nome>Monitor<nome>
         <prezzo valuta="eu">900</prezzo>
         <descrizione>Display LCD ... </descrizione>
    </prodotto>
    ...
</prodotti>
```

valute.xml

```
<?xml version="1.0"?>
<valute>
    <valuta>
         <codice>eu<codice>
         <nome>Euro<nome>
    </valuta>
    ...
</valute>
...
<xsl:variable
   nome=valute
   select="document(valute.xml)/valute" />
...
<xsl:template match="prodotto">
     <xsl:variable name="valuta" select="prezzo/@valuta" />
     <xsl:value-of select="prezzo" />
      <xsl:value-of select="$valute/valuta[codice="$valuta"]/nome" />
     <xsl:value-of select="descrizione" />
</xsl:template>
```

# Estensioni

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/XSL/Transform"
xmlns:myxslt="http://www.apache.org/xslt" exclude-result-prefixes="myxslt">
...
<xsl:variable name="angolo" select="javascript-extension:cos(90)" />
...
<myxslt:component prefix="javascript-extension" functions=cos sin">
   <myxslt:script lang="javascript">
  function cos(a)
  {
   return Math.cos(a);
   }

  function sin(a)
  {
   return Math.cos(a);
   }
 </myxslt:script>
</myxslt:component>
...
</xsl:stylesheet>
```

# Processori XSL-T

- Altova RaptorXML: processore XSLT 3.0
- IBM XSLT processor
- libxslt - free library MIT License per XSLT 1.0
- vari per i diversi linguaggi di programmazione
- Saxon processore XSLT 3.0 e XQuery 3.1
- Xalan processore XSLT 1.0
- Broweser Web: Firefox, Chrome(*), Safari, Opera, Edge per XSLT 1.0

Linux:

```
sudo apt-get install xsltproc

xsltproc --output risultato.xml directorystile/stile.xsl directorysorgente/sorgente.xml
```

# Programmazione

Da una applicazione si può richiamare un processore XSL per eseguire una trasformazione di un documento XML

```
XSLTProcessor processor = XSLTProcessorFactory.getProcessor();
processor.process(new XSLTInputSource(XMLFile),
new XSLTInputSource(XSLFile),
new XSLTResultTarget(NewXMLFile));
```

Apache Xalan (XSL 1.0)

Saxonica saxon (XSL 1.0 e 2.0)

Apache FOP (XSL 1.0 e XSL-FO)

# Preparazione all'esercizio (Trasformazione lato server o da linea di comando)

- Saxon: scaricare da www.saxon.com la versione "he"
- set CLASSPATH=%CLASSPATH%;G:\java-pgm\saxonb9-1-0-2j\saxon9.jar
- java net.sf.saxon.Transform -s:directory\file.xml -xsl:directory\file.xsl -o:directory\output(.html)

# Esercizio

Dato il seguente documento XML

```
<?xml version="1.0"?>
<anagrafe>
        <persona>
                <cognome>Rossi</cognome>
                <nome>Mario</nome>
                <datanascita>
                        <giorno>15</giorno>
                        <mese>05<mese>
                        <anno>2003<anno>
                </datanascita>
        <persona>
        <persona>
                ...
        <persona>
</anagrafe>
```

Trasformare in XHTML con una tabella come questa:

```
<table summary="...." >
    <caption>
    <thead>
        <tr>
        <th>cognome</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Rossi</td><td>Mario</td><td>15/05/2003</td>
        <tr>
    </tbody>
</table>
```

I valori devono essere ordinati per anno, mese, giorno di nascita

# Esempi

Analizziamo gli esempi iniziali
trasformazione di un stesso documento XML in SVG, HTML e PDF

- province.xml - province.xsl
- province.pdf - province.xml - trasformainpdf.xsl

# Risorse su XSLT

- "XSL Transformations (XSLT)" - http://www.w3.org/TR/xslt
- "XML Path Language (XPath)" - http://www.w3.org/TR/xpath
- "Extensible Stylesheet Language (XSL)" - http://www.w3.org/TR/xsl
- "Xalan" http://xml.apache.org/xalan-j
- "Xalan extensions" http://xml.apache.org/xalan-j/extensions.html
- XML Bible 2nd Edition - Capitolo 17 : "XSL Transformations" http://metalab.unc.edu/xml/books/bible/updates/17.html
- "XSL Concepts and Practical Usage" Paul Grosso, Norman Walsh http://www.nwalsh.com/docs/tutorials/xsl/xsl/slides.html
- "XSL by Example" Marc Colan ibm.com/developerWorks/speakers/colan
- G. Ken Holman tutorial "Practical Trasformations with XSL-T and XPath" http://www.CraneSoftwrights.com
- Improve your XSLT coding five ways. Tips to make you a better XSLT programmer. Benoit Marchal - IBM DeveloperWorls
- "XSLT Working with XML and HTML" Fung. Addison Wesley
- XSLT - Doug TidWell - O'Reilly
- Beginning XSLT 2.0 - From Novice to Professional - JENI TENNISON - Apress
- What’s new in XSLT 3.0 and XPath 3.1? - David J. Birnbaum - http://dh.obdurodon.org/xslt3.xhtml

# Una parentesi: XQuery (Recommendation W3C)

Linguaggio di interrogazione
SQL : database relazionali = XQuery : documenti XML su Web

"Trasformazione" da XML a XML

Si avvaleva di un XPath modificato

Forti implicazioni con XSL (XPath2 uniforma l'uso nei due linguaggi XSL e XQuery)

# Esempio XQuery FLWOR

```
<utenti>
  <utente>
    <userid>U01</userid>
    <nome>Mario Rossi</nome>
  </utente>
  ...
```

Elenca i nomi degli utenti che hanno fatto un acquisto singolo di almeno 100 euro

```
<risultato>
  {
    for $u in document("utenti.xml")//utente
    let $s := document("acquisti.xml")//acquisto[userid=$u/userid and costo>=100]
    where count($s) > 1                                                    Se c'è n'è almeno uno
    order by $u/utente/nome descending
    return
        <spendaccione>{ $u/nome/text() }</spendaccione>
  }
</risultato>

Risultato effettivo :
<risultato>
    <spendaccione>Maria Bianchi</spendaccione>
    <spendaccione>Anna Verdi</spendaccione>
    <spendaccione>Giuseppe Gialli</spendaccione>
</risultato>
```

# Grazie per l' attenzione

Domande?

---

... troverete le *slide* all'indirizzo (http://www1.isti.cnr.it/~Martinelli/XML/doc/xsl/)