



XSLT

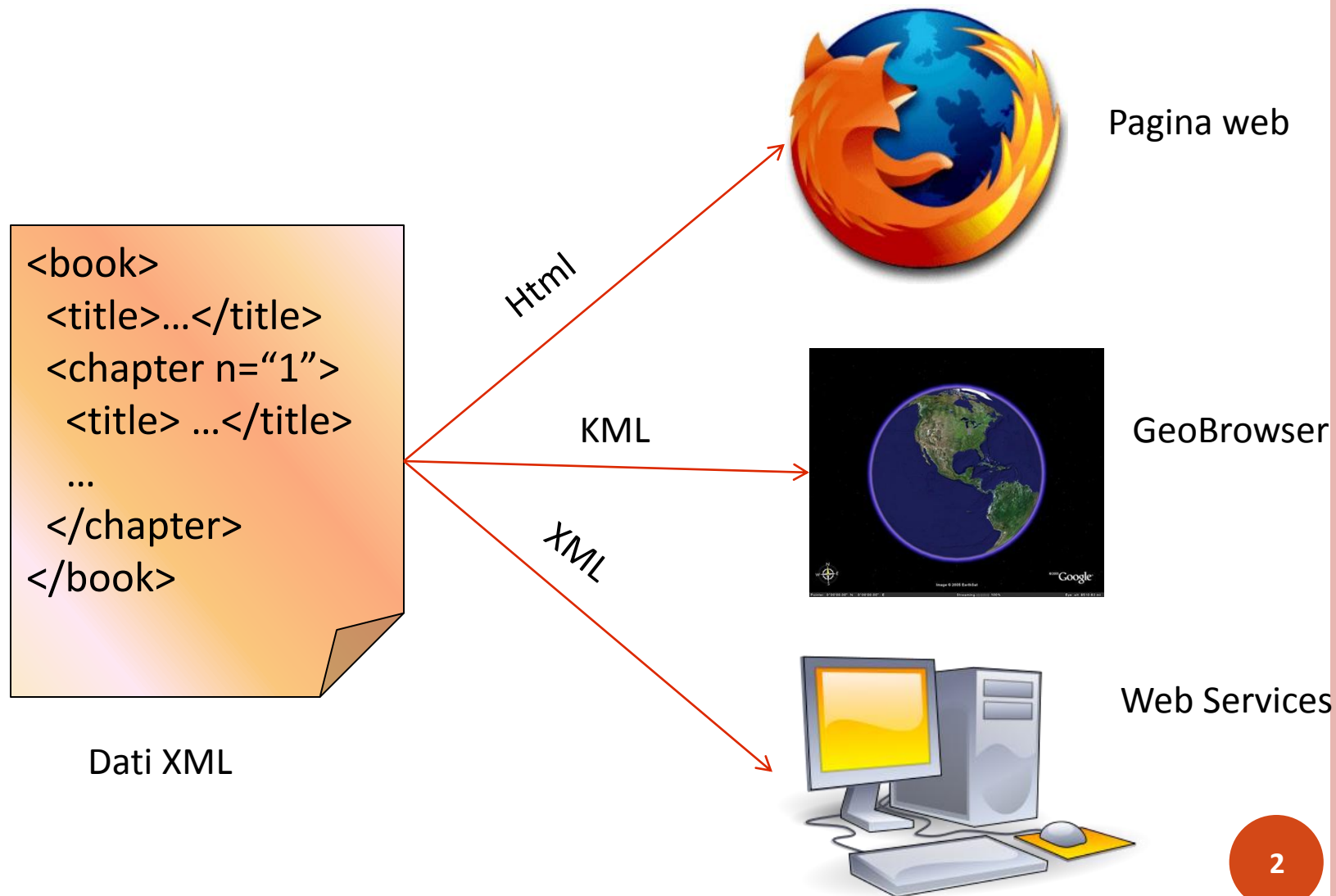
EXTENSIBLE STYLESHEET LANGUAGE

TRANSFORMATION

Dott.sa Vincenza Anna Leano
email: vincenzaanna.leano@unina.it
site: www.leano.it

Basi di Dati II mod. B
Prof. F. Cutugno
A.A. 2011/2012

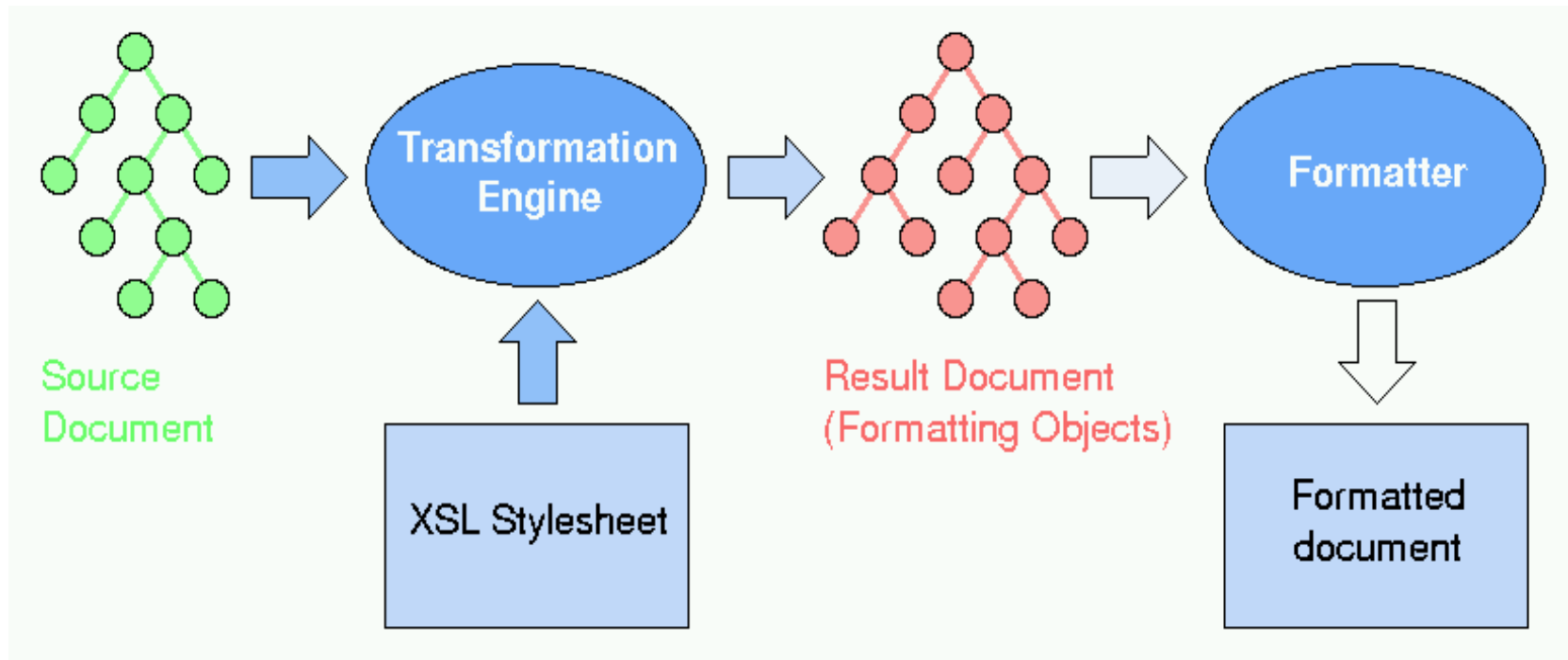
INTRODUZIONE



XSL :TRASFORMAZIONE DOCUMENTI XML

- **XSL (eXtensible Stylesheet Language Family)** è una famiglia di raccomandazioni W3C per definire la trasformazione e la presentazione di documenti XML. Consiste di 3 linguaggi:
 1. XSL Transformations (XSLT): linguaggio che definisce le regole di trasformazione;
 2. XML Path Language (XPath): linguaggio di espressioni utilizzato da XSLT per accedere al (o a parti del) documento
 3. XSL Formatting Objects (XSL-FO): linguaggio che formatta l'output della trasformazione.

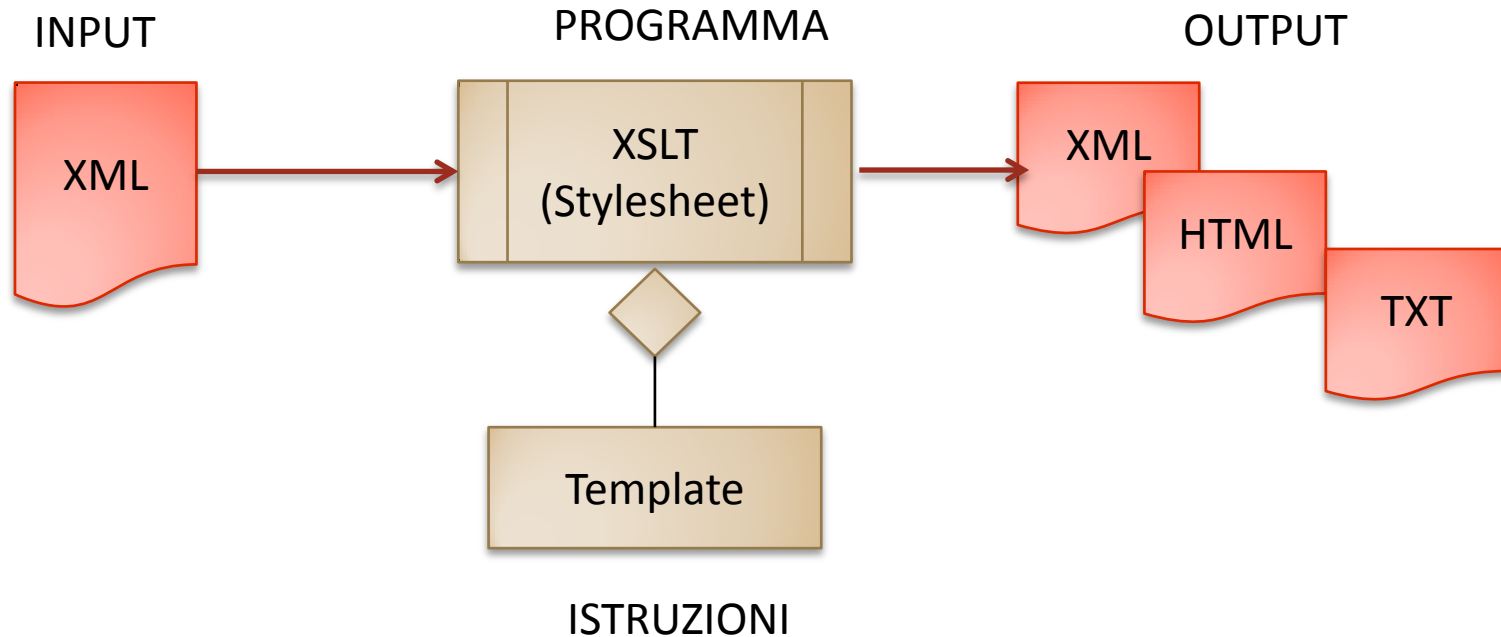
ARCHITETTURA XSL



XSL - FO

- XSL-FO (Formatting Objects) è un linguaggio XML per descrivere il layout fisico di testi
- E' stato pensato originariamente per interagire con l'output di XSLT
- Ad oggi non è supportato da nessun browser
- E' molto utilizzato nell'industria grafica come standard di scambio tra i diversi formati grafici

XSL TRANSFORMATION (XSLT)



- Un programma XSLT prende in input un documento XML e, in base alle regole di template di cui è composto, produce in output un documento XML, HTML, o TXT

XSLT TRANSFORMATION

- Una trasformazione in linguaggio XSLT è espressa come un documento XML ben formato.
- Esprime regole per trasformare un source tree in un result tree.
- Uno stylesheet è composto da una serie di template rules. Ognuna di queste regole ha due parti:
 - **Pattern:** identifica i nodi del documento di origine a cui si applica la regola di template
 - **Body (regola di template):** un frammento di *XML ben formato* che rappresenta come trasformare nel documento di output i nodi su cui è *istanziato* il template.

ESEMPIO «HELLO WORLD»

Prologo → `<?xml version=1.0 ?>`

Radice → `<xsl:stylesheet version="2.0"`

Dichiarazione
di
namespaces → `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
`xmlns="http://www.w3.org/1999/xhtml">`

Output
format → `<xsl:output method="xml" encoding="utf-8"/>`

Template → `<xsl:template match="/">`
`Hello World`
`</xsl:template>`

`</xsl:stylesheet>`

XSL:TEMPLATE

```
<xsl:template
  match = pattern
  name = qname
  priority = number
  mode = qname>
  <!-- Template Rules: (xsl:param*, template) -->
</xsl:template>
```

- `match`: pattern del template
- `name`: usato per chiamare i template come delle funzioni
- `mode`: usato per restringere i template candidati, permesso solo se `match` è presente
- `priority`: priorità della regola

PATTERN

- Il pattern è l'insieme di nodi a cui si applica la regola
- Per indicare tale insieme si usa XPATH, però:
 - L'istanziamento di un pattern deve essere sempre un insieme di nodi
 - Deve essere semplice decidere se un nodo è istanziato o meno da un pattern.
- Un *pattern* è un'espressione XPath ristretta
 - È un'unione di path expressions (*patExp1 | patExp2 | ...*)
 - ogni path expression contiene un numero di passi separati da / o //
 - Ogni passo può usare solo l'axis child o l'axis attribute
 - È possibile utilizzare funzioni XPath (*text()*...)

TEMPLATE BODY/RULES

- Il corpo di un template detta le regole di trasformazione dei nodi di origine nei nodi di destinazione
- E' un frammento di xml ben formato
- Si applica a partire da un contesto corrente
- Può contenere:
 - Tag xml
 - Testo
 - Costrutti di xsl:
 - `<xsl:apply-templates select="" />`: invoca altre regole di template sul risultato di select
 - `<xsl:value-of select="" />`: stampa il contenuto di select
 - altri...

ESECUZIONE DI UNO STYLESHEET (1/2)

- L'ordine di esecuzione non è da quello delle «istruzioni» come nei linguaggi di programmazione comuni.
- Il parser parte ad analizzare i nodi dell'albero del documento di input .
- Si eseguono i template i cui pattern sono il match del nodo corrente (context node)
- Formalmente:

Dato un albero XML T , un pattern P è il match di un nodo N se esiste un nodo C (il context node) in T tale per cui

$$N \in P(T, C)$$

ESECUZIONE DI UNO STYLESHEET (2/2)

- Le valutazioni delle regole in uno stylesheet vengono fatte a partire da un contesto corrente.
- Il parser costruisce una lista di nodi correnti e per default inizia da una lista contenente il nodo radice.
- Seleziona tutti i template applicabili al contesto corrente
 - Se non sono presenti match, si applica una regola di default
 - Se più template sono il match di un nodo, viene selezionato il più specifico.
- Il template viene applicato al nodo corrente creando frammenti dell'albero di destinazione, invocando ricorsivamente altri template e inserendo altri nodi nella lista di nodi correnti.
- Il ciclo si ripete fino ad esaurimento della lista di nodi correnti.

RISOLUZIONI DI CONFLITTI TRA REGOLE

- E' possibile che più regole siano il match di un nodo:
 - XSLT processor seleziona il **best match**
- Regole di precedenza:
 - Se il pattern contiene più alternative separate da |, ognuna di esse è trattata alla pari, come se ci fosse una regola separata per ogni alternativa.
 - Un pattern specifico ha una priorità più alta di uno che contiene informazioni generali.
 - `quotelist/quote/body` è più specifico di `body`, e ha precedenza
 - Un pattern che contiene un `*` è più generale e ha una priorità minore rispetto a un pattern specifico.
 - Il pattern `stuff/cruft` batte il pattern `stuff/*`
 - Un pattern con una test expression racchiusa tra [] sovrascrive un pattern senza test expression
 - `bobo[@role="clown"]` ha maggiore priorità di `bobo`

DEFAULT RULES

- Le default rules sono applicate a quei nodi per i quali lo stylesheet non ha previsto una regola di template
- Le default rules hanno la priorità più bassa.
- Ogni tipo di nodo ha una sua regola di default

DEFAULT RULES

○ Root ed elementi:

```
<xsl:template match=" * | / ">
    <xsl:apply-templates/>
</xsl:template>
```

invoca il template che è un match del figlio del nodo corrente.

○ Nodi testo e attributo

```
<xsl:template match=" text() | @* ">
    <xsl:value-of select="."/>
</xsl:template>
```

Si limita a copiarne il valore

○ Processing instruction o commenti:

```
<xsl:template match="processing-instruction()
|comment()"/>
```

Nessuna azione

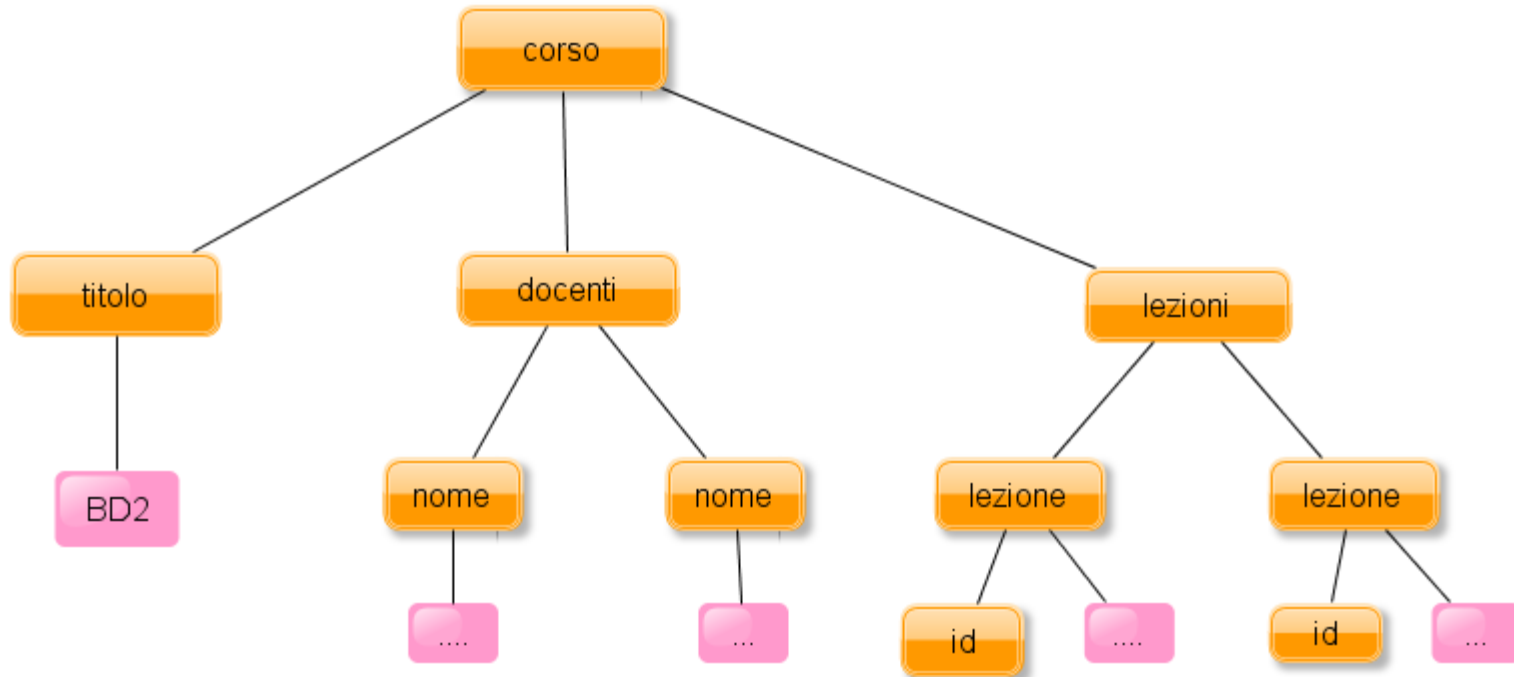
○ Nodi di namespace:

- La regola di default è quella di non eseguire nessuna azione
- Poiché non ci sono pattern che sono il match di un nodo di namespace, la regola di default è l'unica ad essere applicata

ESEMPIO DOCUMENTO DI ORIGINE

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="stile.xsl" type="text/xsl" ?>
<corso>
  <titolo> Basi di Dati e Sistemi Informativi 2 mod B </titolo>
  <docenti>
    <nome> F. Cutugno </nome>
    <nome> V. A. Leano </nome>
  </docenti>
  <lezioni>
    <lezione id="1"> Basi di dati semistrutturate </lezione>
    <lezione id="2"> XML Well Formedness </lezione>
    <lezione id="3"> Xml Schema, DTD </lezione>
    <lezione id="4"> Document Object Model </lezione>
    <lezione id="5"> XPath </lezione>
    <lezione id="6"> XSLT </lezione>
  </lezioni>
</corso>
```

ESEMPIO: ALBERO



ESEMPIO 2: ORDINE ESECUZIONE ISTRUZIONI

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="lezione">
    <xsl:value-of select="."/>
  </xsl:template>
  <xsl:template match="titolo">
    <xsl:value-of select="."/>
  </xsl:template>
  <xsl:template match="nome">
    <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

ESEMPIO 2: RISULTATO

Basi di Dati e Sistemi Informativi 2 mod B

F. Cutugno

V. A. Leano

Basi di dati semistrutturate

XML Well Formedness

Xml Schema, DTD

Document Object Model

XPath

XSLT

- L'ordine è dato dalla visita in profondità dei nodi del nodo di origine

ESEMPIO 2: INSERIMENTO NODI NELLA CURRENT NODE LIST

T1

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

T2

```
<xsl:template match="corso">  
  Titolo: <xsl:value-of select="corso/titolo"/>  
  Docenti :<xsl:apply-templates select="docenti/nome"/>  
</xsl:template>
```

T2a

T3

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

T4

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

ESEMPIO 2: RISULTATO

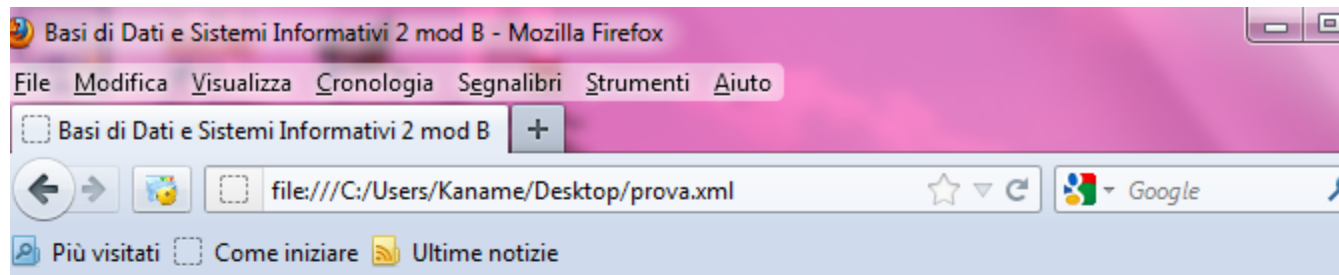
Basi di Dati e Sistemi Informativi 2 mod B

F. Cutugno

V. A. Leano

- Inizializzazione: / -> istanza T1
- T1 aggiunge corso -> istanza T2
- T2 aggiunge titolo, docenti/nome[1] e docenti/nome[2] -> istanza t2a, T3
- T2a stampa titolo
- T3 stampa nome (2 volte)
- **T4 non risulta mai essere un match e non viene invocato.**

RISULTATO CHE SI VUOLE OTTENERE



Basi di Dati e Sistemi Informativi 2 mod B

Docenti

- F. Cutugno
- V. A. Leano

Elenco Lezioni

- Basi di dati semistrutturate
- XML Well Formedness
- Xml Schema, DTD
- Document Object Model
- XPath
- XSLT

ESEMPIO XSLT

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

  <xsl:template match="/">
    <html> <head>
      <title> <xsl:value-of select="corso/titolo"/> </title>
    </head>
    <body>
      <xsl:apply-templates select="corso"/>
    </body>
  </html>
</xsl:template>
```


ESEMPIO XSLT

```
<xsl:template match="corso">
  <h1><xsl:value-of select="titolo"/></h1>
  <h2> Docenti </h2>
  <ul><xsl:apply-templates select="docenti/nome"/></ul>
  <h2> Elenco Lezioni </h2>
  <ul><xsl:apply-templates select="lezioni/lezione"/> </ul>
</xsl:template>
<xsl:template match="nome">
  <li> <xsl:value-of select="."/></li>
</xsl:template>
<xsl:template match="lezione">
  <li> <xsl:value-of select="."/></li>
</xsl:template>
/xsl:stylesheet>
```

CORPO DI UN TEMPLATE

- Il corpo di un template puo' contenere due tipi di istruzioni:

- Istruzioni che creano e modificano l'albero di destinazione

elementi letterali	<xsl:value-of>
<xsl:element>	<xsl:attribute>
<xsl:text>	<xsl:processing-instruction>
<xsl:comment>	<xsl:namespace-alias>
<xsl:copy>	<xsl:number>

- Istruzioni che modificano la lista dei nodi correnti

<xsl:apply-templates>	<xsl:for-each>
<xsl:if>	<xsl:choose>
<xsl:sort>	



ISTRUZIONI DI COSTRUZIONE ALBERO DI DESTINAZIONE

27

COSTRUZIONE DELL'ALBERO DI DESTINAZIONE

1. Costruttori letterali

- Sequenza di nodi che contengono dati carattere e nodi elemento che non appartengono al namespace XSLT

2. Costruttori specifici

- `<xsl:element name="">`
- `<xsl:attribute name="">`: l'attributo creato fa riferimento all'elemento padre del tag (un `xsl:element` o un tag non appartenente al namespace `xslt`)
- `<xsl:text>`: gli elementi all'interno del tag preservano gli spazi.
- `<xsl:processing-instruction name="">`
- `<xsl:comment select="">`

ESEMPIO: COSTRUTTORI LETTERALI

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
    <html>
      <head>
        <title>Hello World</title>
      </head>
      <body>
        <xsl:attribute name="bgcolor" select="'green'"/>
        <b>Hello World</b>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

ESEMPIO COSTRUTTORI SPECIFICI

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
    <xsl:element name="html">
      <xsl:element name="head">
        <xsl:element name="title">Hello World</xsl:element>
      </xsl:element>
      <xsl:element name="body">
        <xsl:attribute name="bgcolor" select="'green'"/>
        <xsl:element name="b">Hello World</xsl:element>
      </xsl:element>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

XSLT VALUE-OF ELEMENT

```
<xsl:value-of  
  select = string-expression  
  disable-output-escaping= "yes"|"no"/>
```

- Valuta l'espressione e mostra il risultato
- Se applicata a una nodelist l'espressione viene applicata al primo nodo
- "." seleziona il text value del nodo corrente
- `disable-output-escaping`: se vale `yes` sono consentiti i valori "<" e "&"

I VALORI PER OGNI TIPO DI NODO

- Root:
 - Il nodo radice eredita il valore del document element
- Element
 - Il testo dell'elemento insieme a tutti i suoi figli di tipo testo o elemento
- Attribute
 - Il valore dell'attributo con gli spazi normalizzati
- Text
 - Tutto il testo nel nodo
- Processing instruction
 - Tutto all'interno del delimitatore eccetto il nome
- Comment
 - Testo all'interno del delimitatore del commento
- Namespace
 - L'URI del namespace

ESEMPIO

○ Input:

```
...  
<name prefix="Mr.">John Doe</name>  
...
```

○ Xslt:

```
<xsl:template match="name">  
    Your name is  
    <xsl:value-of select="@prefix"/>  
    <xsl:value-of select="."/>  
</xsl:template>
```

○ Output:

- Your Name is Mr. John Doe

COPIARE NODI

- Copy-of inserisce il frammento di source tree derivante dalla select nel result tree senza trasformare prima il valore in testo come quando utilizziamo value-of

```
<xsl:copy-of  
  select = expression />
```

- Crea una copia locale del nodo corrente, senza attributi nè figli.

```
<xsl:copy>  
  <!-- Content: template -->  
</xsl:copy>
```

ESEMPIO COPY

- Cambiare il punto degli elenchi di html in un quadrato:

```
<UL>
  <li>...</li>
  ...
  <li>...<li>
</UL>
```

```
<xsl:template match="ol|ul">
  <xsl:copy>
    <xsl:attribute name="style"
      select="'list-style-type:square;'" />
    <xsl:copy-of select="*" />
  </xsl:copy>
</xsl:template>
```



ISTRUZIONI PER MODIFICARE LA NODLIST

36

APPLICAZIONE RICORSIVA

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

Applica ricorsivamente l'intero foglio di stile ai nodi specificati dall'attributo select.

- Concatena la sequenza risultante
- Il valore di default di select è child::node()
- Utilizzando l'attributo mode vengono prese in considerazione solo le trasformazioni che hanno un mode corrispondente.

STUDENT DATA

```
<students>
  <student id="100026">
    <name">Joe Average</name>
    <age>21</age>
    <major>Biology</major>
    <results>
      <result course="Math 101" grade="C-"/>
      <result course="Biology 101" grade="C+"/>
      <result course="Statistics 101" grade="D"/>
    </results>
  </student>
  <student id="100078">
    <name>Jack Doe</name>
    <age>18</age>
    <major>Physics</major>
    <major>XML Science</major>
    <results>
      <result course="Math 101" grade="A"/>
      <result course="XML 101" grade="A-"/>
      <result course="Physics 101" grade="B+"/>
      <result course="XML 102" grade="A"/>
    </results>
  </student>
</students>
```

```
<summary>
  <name id="100026">Joe Average</name>
  <name id="100078">Jack Doe</name>
  <grades id="100026">
    <grade>C-</grade>
    <grade>C+</grade>
    <grade>D</grade>
  </grades>
  <grades id="100078">
    <grade>A</grade>
    <grade>A-</grade>
    <grade>B+</grade>
    <grade>A</grade>
  </grades>
</summary>
```

STYLESHEET PER IL SUMMARY (1/2)

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="students">
    <summary>
      <xsl:apply-templates mode="names" select="student"/>
      <xsl:apply-templates mode="grades" select="student"/>
    </summary>
  </xsl:template>

  <xsl:template mode="names" match="student">
    <name>
      <xsl:attribute name="id" select="@id"/>
      <xsl:value-of select="name"/>
    </name>
  </xsl:template>
```

STYLESHEET PER IL SUMMARY (2/2)

```
<xsl:template mode="grades" match="student">
  <grades>
    <xsl:attribute name="id" select="@id"/>
    <xsl:apply-templates select="./@grade"/>
  </grades>
</xsl:template>

<xsl:template match="@grade">
  <grade>
    <xsl:value-of select="."/>
  </grade>
</xsl:template>
</xsl:stylesheet>
```


INVOCAZIONE DI TEMPLATE

- Se un template ha l'attributo name, può essere invocato all'interno dello stylesheet attraverso l'istruzione

```
<xsl:call-template name="">
```

- Utile per sfruttare il principio di modularità.
- Applica una regola al contesto corrente
- Utile per superare il meccanismo del pattern matching.

ITERAZIONE

- E' possibile iterare sui nodi di una nodelist con
<xsl:for-each select="">
- Calcola la sequenza specificata dall'attributo select e applica la regola specificata nel suo corpo ad ogni elemento della selezione

```
<xsl:template match="student">
  <grades>
    <xsl:attribute name="id" select="@id"/>
    <xsl:for-each select=" .//@grade">
      <xsl:call-template name="listgrade"/>
    </xsl:for-each>
  </grades>
</xsl:template>

<xsl:template name="listgrade">
  <grade>
    <xsl:value-of select="."/>
  </grade>
</xsl:template>
```

ISTRUZIONI CONDIZIONALI: XSLT:IF

```
<xsl:if test="expression">  
<!-- context --!>  
</xsl:if>
```

- Valuta l'espressione se è vera applica la regola
- Se è falsa non produce output

```
<xsl:template match="class">  
<!-- Select the first node in the set. -->  
  <xsl:if test="position() = first()">  
    <b><xsl:value-of select="."/></b>  
  </xsl:if>  
</xsl:template>
```

ISTRUZIONI CONDIZIONALI: XSLT:CHOOSE

```
<xsl:choose>
  <xsl:when test = boolean-expression>
    <!-- Content: template -->
  </xsl:when>
  ...
  <xsl:otherwise>
    <!-- Content: template -->
  </xsl:otherwise>
</xsl:choose>
```

- Seleziona tra un numero di alternative.
- Viene eseguito il primo `When` il cui test risulta vero
- Se nessun `when` risulta vero viene eseguito `otherwise` se presente

ORDINAMENTO

```
<xsl:sort  
  select = string-expression  
  lang = { nmtoken }  
  data-type = { "text" | "number" | .. }  
  order = { "ascending" | "descending" }  
  case-order = { "upper-first" |  
                 "lower-first" } />
```

- Ordina le sequenze selezionate dall'attributo select in base ai valori dei parametri
- L'ordine delle sort nel documento definisce la priorità dei principi di ordinamento.

ESEMPIO DI ORDINAMENTO

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="students">
    <enrolled>
      <xsl:apply-templates select="student">
        <xsl:sort select="age" data-type="number"
          order="descending"/>
        <xsl:sort select="name"/>
      </xsl:apply-templates>
    </enrolled>
  </xsl:template>

  <xsl:template match="student">
    <student name="{name}" age="{age}"/>
  </xsl:template>
</xsl:stylesheet>
```

RAGGRUPPAMENTI

```
<xsl:for-each-group select="" group-by="">  
  <!-- context --!>  
</xsl:for-each group>
```

- Il contesto può contenere due funzioni:
 - `current-grouping-key()`: restituisce la chiave di raggruppamento corrente
 - `current-group()`: restituisce la sequenza di componenti con quella data chiave
- Valutazione:
 - Raggruppa gli elementi di una selezione in base a una chiave.
 - L'espressione di `select` viene valutata-> popolazione
 - Valutazione per ogni elemento della popolazione dell'espressione di `group-by`
 - Confronto dei valori risultanti-> chiavi di raggruppamento distinte
 - Valutazione del contesto della regola per ogni chiave
 - Concatenazione risultati

ESEMPIO

```
<xsl:stylesheet version="2.0"
  xmlns:rcp="http://www.brics.dk/ixwt/recipes"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="rcp:collection">
    <uses>
      <xsl:for-each-group select="//rcp:ingredient"
        group-by="@name">
        <use name="{current-grouping-key}"
          count="{count(current-group())}"/>
      </xsl:for-each-group>
    </uses>
  </xsl:template>
</xsl:stylesheet>
```


OUTPUT

```
<uses>
  <use name="beef cube steak" count="1"/>
  <use name="onion, sliced into thin rings" count="1"/>
  <use name="green bell pepper, sliced in rings" count="1"/>
  <use name="Italian seasoned bread crumbs" count="1"/>
  <use name="grated Parmesan cheese" count="1"/>
  <use name="olive oil" count="2"/>
  <use name="spaghetti sauce" count="1"/>
  <use name="shredded mozzarella cheese" count="1"/>
  <use name="angel hair pasta" count="1"/>
  <use name="minced garlic" count="3"/>
  ...
</uses>
```

FUNZIONI

```
<xsl:function name="">  
  <!-- context --!>  
</xsl:function>
```

- Facilitano il riuso e la modularizzazione
- Riceve i parametri attraverso la dichiarazione di elementi param
- Una volta definita la funzione può essere richiamata all'interno di funzioni XPath

ESEMPIO

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:local="http://www.w3.org/2004/07/xquery-local-functions">
  <xsl:function name="local:fib">
    <xsl:param name="n"/>
    <xsl:value-of select="if ($n le 1)
      then 1
      else local:fib($n -1)+local:fib($n -2)"/>
  </xsl:function>

  <xsl:template match="/">
    <xsl:value-of select="local:fib(10)"/>
  </xsl:template>
</xsl:stylesheet>
```



STRUTTURA DI UN DOCUMENTO XSLT

52

STRUTTURA DOCUMENTO XSLT

○ Root:

- `<xsl:stylesheet`
 `xmlns:xsl=http://www.w3.org/1999/XSL/Transform`
 version = *number*
 `<!-- Content: (xsl:import*, top-level-`
 `elements) -->`
 `</xsl:stylesheet>`
- `<xsl:transform`
 `xmlns:xsl=http://www.w3.org/1999/XSL/Transform`
 version = *number*
 `<!-- Content: (xsl:import*, top-level-`
 `elements) -->`
 `</xsl:transform>`

TOP LEVEL ELEMENTS

- `xsl:import`
- `xsl:include`
- `xsl:strip-space`
- `xsl:preserve-space`
- `xsl:output`
- `xsl:key`
- `xsl:decimal-format`
- `xsl:namespace-alias`
- `xsl:attribute-set`
- `xsl:variable`
- `xsl:param`
- `xsl:template`

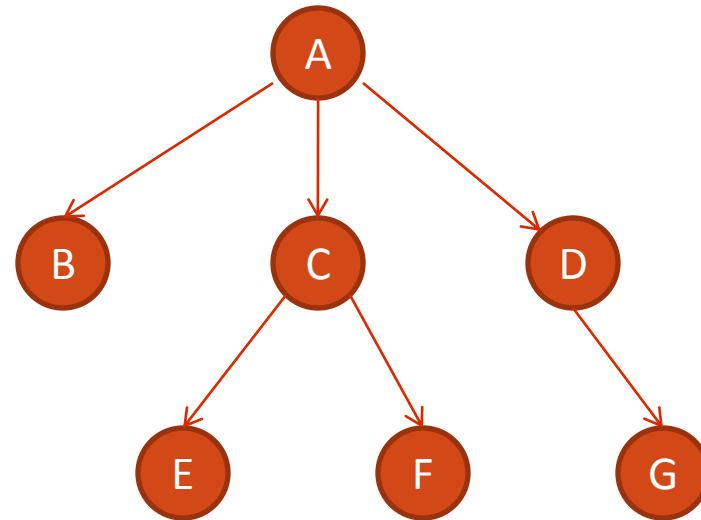
IMPORT E INCLUDE

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

- Servono per la modularizzazione dei file xsl
- Effetto: i file dichiarati vengono inseriti nel file di origine al posto del tag
- Include:
 - errore se ci sono riferimenti circolari o se si tenta di ridefinire una regola.
- Import:
 - deve comparire prima di ogni altra regola
 - consente l'overriding delle regole importate

IMPORT TREE

- Gli elementi incontrati mentre si processa un'istruzione di import vanno a formare l'import tree.
- Ogni include viene risolto prima di costruire l'albero
- Ogni import incontrato viene definito come figlio di stylesheet
- Se i fogli di stile importati al loro interno hanno altri import, saranno figli di quel nodo
- Le priorità delle regole, dalla più bassa alla più alta, è data dalla visita post order dell'albero.



- Priorità (dalla più bassa alla più alta): B,E,F,C,G,D,A

STRIP AND PRESERVE SPACE

```
<xsl:strip-space elements="element list" />  
<xsl:preserve-space elements="element list" />
```

- Gestione degli spazi
- Con strip vengono ignorati gli spazi finali, con preserve vengono conservati

OUTPUT

- Determina il formato di output del documento di destinazione

<xsl:output

```
method = "xml" | "html" | "text" | qname-but-not-ncname  
version = nmtoken  
encoding = string  
omit-xml-declaration = "yes" | "no"  
standalone = "yes" | "no"  
doctype-public = string  
doctype-system = string  
cdata-section-elements = qnames  
indent = "yes" | "no"  
media-type = string />
```

KEY

- Dichiarazione di una chiave

```
<xsl:key  name = qname    match = pattern  
        use = expression />
```

- L'istruzione definisce una corrispondenza tra il nome di una chiave e il valore di un nodo
 - **name**: nome della chiave
 - **match**: nodi obiettivo
 - **Use**: valore da associare alla chiave, viene valutato per ogni nodo che soddisfa il pattern
- Funzione `Key()`: prende in input il nome e il valore di una chiave e ne restituisce gli elementi corrispondenti.

VARIABILI E PARAMETRI

```
<xsl:param name = qname select = expression>
  <!-- Content: template -->
</xsl:param>

<xsl:variable name = qname select = expression>
  <!-- Content: template -->
</xsl:variable>
```

- **Variable** definisce una variabile di sola lettura il cui valore può essere assegnato con un'espressione Xpath
- Il campo d'azione della variabile è la parte restante della sequenza in cui è avvenuta la dichiarazione, se è un top element la variabile sarà globale
- **Param** viene utilizzato per definire parametri formali da utilizzare per esempio nelle istruzioni apply-template o call-template.

ESEMPIO VARIABILI E PARAMETRI

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template name="fib">
    <xsl:param name="n"/>
    <xsl:choose>
      <xsl:when test="$n le 1">
        <xsl:value-of select="1"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:variable name="f1">
          <xsl:call-template name="fib">
            <xsl:with-param name="n" select="$n -1"/>
          </xsl:call-template>
        </xsl:variable>
        <xsl:variable name="f2">
          <xsl:call-template name="fib">
            <xsl:with-param name="n" select="$n -2"/>
          </xsl:call-template>
        </xsl:variable>
        <xsl:value-of select="$f1+$f2"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

```
<xsl:template match="/">
  <xsl:call-template name="fib">
    <xsl:with-param name="n"
      select="10"/>
  </xsl:call-template>
</xsl:template>
</xsl:stylesheet>
```

NAMESPACE-ALIAS

```
<xsl:namespace-alias  
  stylesheet-prefix = prefix  
  result-prefix = prefix />
```

- Crea un alias del namespace result prefix
- Utilizzato per esempio per creare uno stylesheet attraverso xsl

ESEMPIO NAMESPACE-ALIAS: DOCUMENTO DI ORIGINE

```
<elements>  
  <block>p</block>  
  <block>h1</block>  
  <block>h2</block>  
  <block>h3</block>  
</elements>
```

ESEMPIO NAMESPACE ALIAS (1/2)

```
o <xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:axsl="file://namespace.alias">
  <xsl:namespace-alias stylesheet-prefix="axsl"
  result-prefix="xsl"/>
  <xsl:template match="/">
    <axsl:stylesheet version="2.0">
      <xsl:apply-templates/>
    </axsl:stylesheet>
  </xsl:template>
```


ESEMPIO NAMESPACE-ALIAS (2/2)

```
<xsl:template match="elements">
  <axsl:template match="/">
    <axsl:comment
      select="system-property('xsl:version')"/>
    <axsl:apply-templates/>
  </axsl:template>
</xsl:template>
<xsl:template match="block">
  <axsl:template match="{.}">
    <fo:block>
      <axsl:apply-templates/>
    </fo:block>
  </axsl:template>
</xsl:template>
</xsl:stylesheet>
```

STYLESHEET RISULTATO (1/2)

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:template match="/">
    <xsl:comment select="system-
      property('xsl:version')"/>
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="p">
    <fo:block>
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
```

STYLESHEET RISULTATO (2/2)

```
<xsl:template match="h1">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="h2">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="h3">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
</xsl:stylesheet>
```

PIÙ FILE DI INPUT

```
<xsl:stylesheet version="2.0"
  xmlns:rcp="http://www.brics.dk/ixwt/recipes"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="rcp:collection">
    <rcp:collection>
      <rcp:title>Selected Recipes</rcp:title>
      <xsl:apply-templates select="rcp:recipe"/>
    </rcp:collection>
  </xsl:template>

  <xsl:template match="rcp:recipe">
    <xsl:variable name="t" select="rcp:title/text()"/>
    <xsl:if test="not(doc('dislikes.xml'))//
      rcp:recipe[rcp:title eq $t]">
      <xsl:copy-of select="."/>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

GENERARE PIÙ FILE DI OUTPUT (1/2)

```
<xsl:stylesheet version="2.0"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="students">
    <xsl:result-document href="names.html">
      <html>
        <head><title>Students</title></head>
        <body>
          <xsl:apply-templates select="student" mode="name"/>
        </body>
      </html>
    </xsl:result-document>
    <xsl:result-document href="grades.html">
      <html>
        <head><title>Grades</title></head>
        <body>
          <xsl:apply-templates select="student" mode="grade"/>
        </body>
      </html>
    </xsl:result-document>
  </xsl:template>
```

GENERARE PIÙ FILE DI OUTPUT (2/2)

```
<xsl:template match="student" mode="name">
  <a href="grades.html#{@id}"><xsl:value-of select="name"/></a>
  <br/>
</xsl:template>

<xsl:template match="student" mode="grade">
  <a name="{@id}"/>
  <xsl:value-of select="name"/>
  <ul>
    <xsl:apply-templates select="results/result"/>
  </ul>
</xsl:template>

<xsl:template match="result">
  <li>
    <xsl:value-of select="@course"/>:
    <xsl:text> </xsl:text>
    <xsl:value-of select="@grade"/>
  </li>
</xsl:template>
</xsl:stylesheet>
```



TRASFORMARE I DOCUMENTI XML TRAMITE XSLT

71

TRASFORMAZIONE TRAMITE BROWSER

- Le trasformazioni XSL sono supportate da tutti i browser moderni.
- Un documento xml può contenere una process-instruction che punta a un documento XSLT
- Il browser carica il file ed esegue la trasformazione.

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


TRASFORMAZIONE TRAMITE JAVA

- La JDK 1.4 contiene tutti i package necessari alla trasformazione:
 - `javax.xml.transform package`.
- Le versioni precedenti della JDK richiedevano il download di un processore XSLT e di un parser SAX

XSLTMANAGER.JAVA

```
public static void main(String[] args) {
    try {
        TransformerFactory tFactory =
TransformerFactory.newInstance();

        Transformer transformer = tFactory.newTransformer
            (new
javax.xml.transform.stream.StreamSource(xslFileName));

        transformer.transform
            (new javax.xml.transform.stream.StreamSource
                (xmlFileName),
            new javax.xml.transform.stream.StreamResult
                ( new FileOutputStream(outputFileName)));
    }
    catch (Exception e) {
        e.printStackTrace( );
    }
}
```

ESERCIZIO

XML

```
<catalog>
  <cd>
    <title>Empire
      Burlesque</title>
    <artist>Bob
      Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>...</cd>
</catalog>
```

HTML

```
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th><th>Artist</th>
</tr>
<tr>
<td>Empire
  Burlesque</td><td>Bob
  Dylan</td>
</tr>
<tr>...</tr>
</body>
</html>
```

SOLUZIONE (1/2)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
```

SOLUZIONE (2/2)

```
<xsl:for-each select="catalog/cd">
  <tr>
    <td><xsl:value-of select="title"/></td>
    <td><xsl:value-of select="artist"/></td>
  </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

RIFERIMENTI

- Introduzione a XML – A. Moller - Cap 5
- Web Data management and Distribution – Abitebuol – Additional Chapter:
<http://webdam.inria.fr/Jorge/files/wdm-xslt.pdf>
- Specifiche W3C XSLT 1.0: <http://www.w3.org/TR/xslt>
- Specifiche W3C XSLT 2.0: <http://www.w3.org/TR/xslt20/>