CACS 205: Web Technology
XML

PREPARED BY KRISHNA PD. ACHARYA

(MECHI MULTIPLE CAMPUS)

The client tier 10 Hrs

Representing content, Introduction to xml, Elements and attributes

Rules for writing xml, Namespaces(schema)

Simple types and complex types, XSD attributes, default and fixed values, facets, use of patterns,

Order indicators(all, choice, sequence), occurrence indicators (max occurs, min occurs)

DTD:

Internal declaration,

Private external declaration,

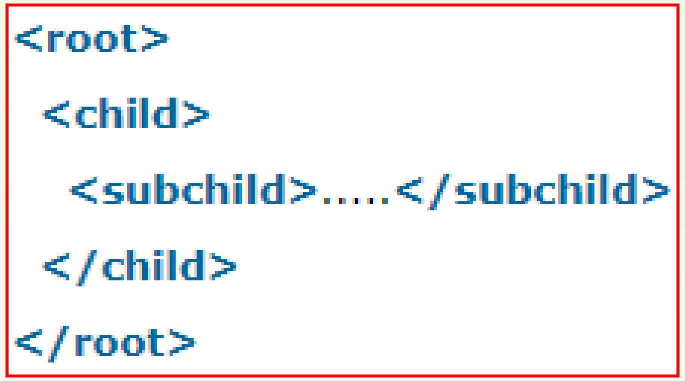
Public external declaration, defining elements and attributes

XSL/XSLT, Xpath, Xquery;

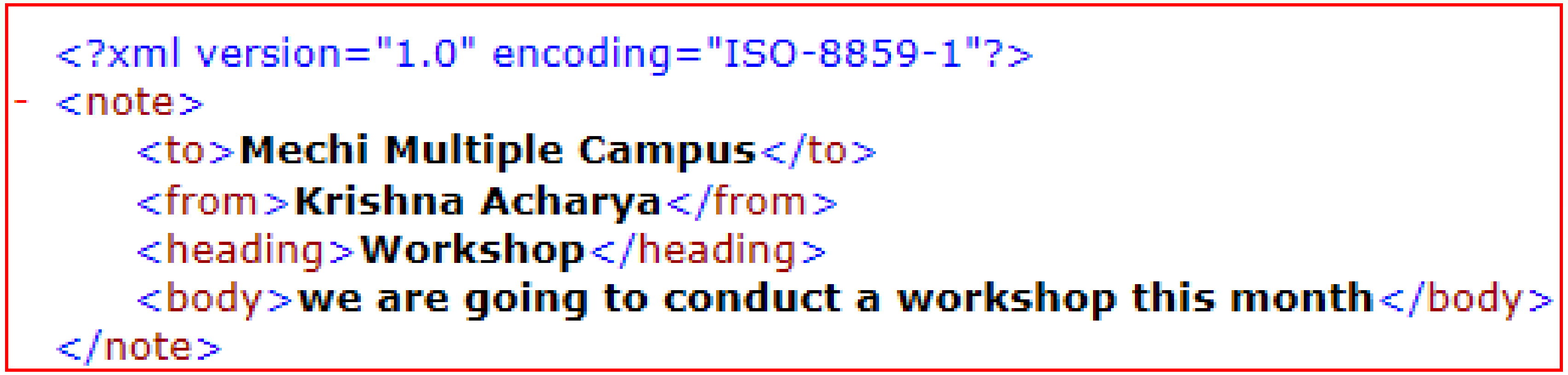
Sax:Dom, Creating XML Parser

Introduction to xml 10 Hrs.

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
 <to>Mechi Multiple Campus</to>
 <from>Krishna Acharya</from>
 <heading>Workshop</heading>
 <body>we are going to conduct a workshop this month</body>
</note>
```



```
<root>
 <child>
  <subchild>.....</subchild>
 </child>
</root>
```



```
  <?xml version="1.0" encoding="ISO-8859-1"?>
- <note>
      <to>Mechi Multiple Campus</to>
      <from>Krishna Acharya</from>
      <heading>Workshop</heading>
      <body>we are going to conduct a workshop this month</body>
  </note>
```

Introduction to xml 10 Hrs.

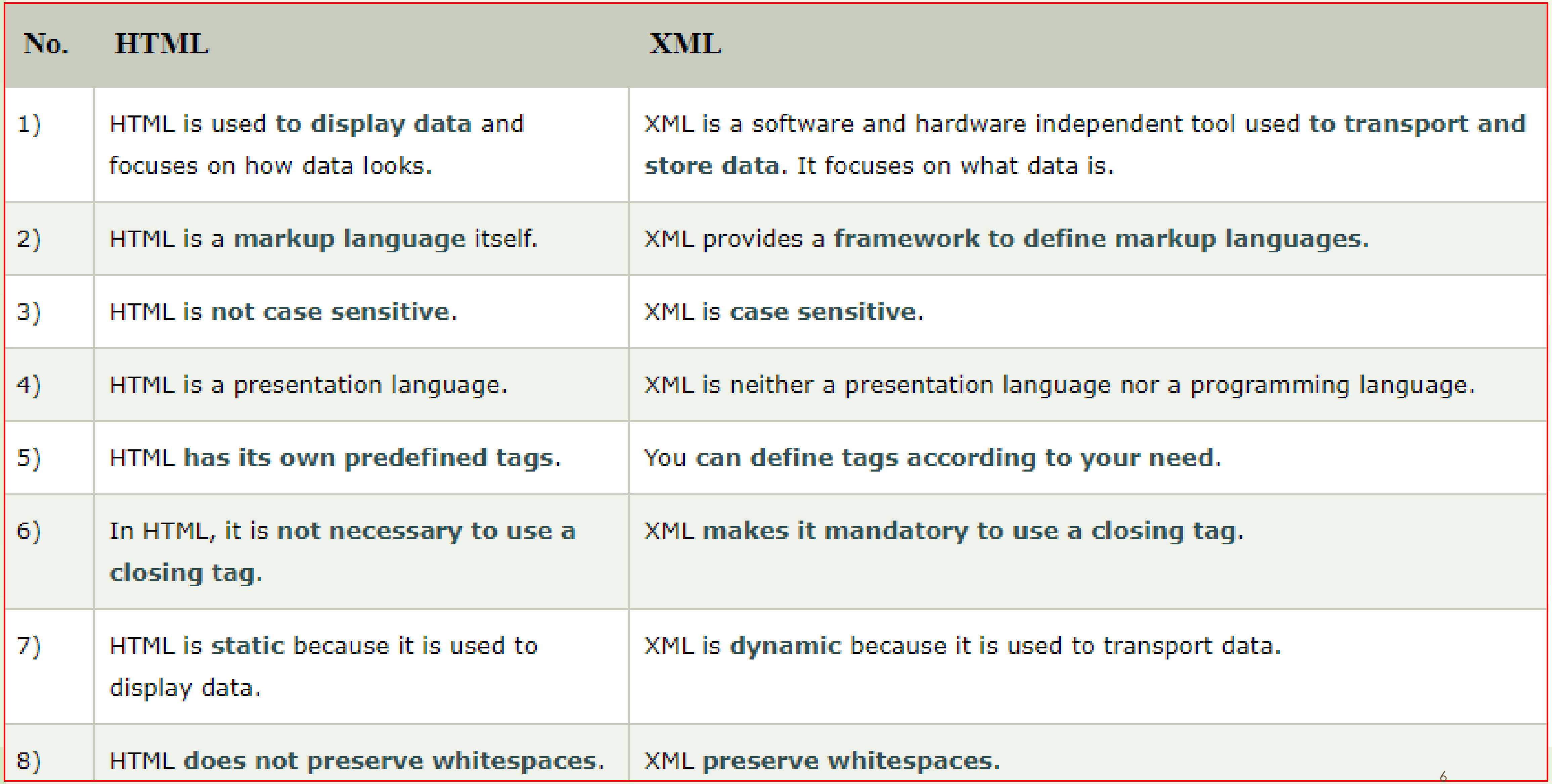
- XML stands for **Extensible Markup Language**. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).
- XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.
- XML was designed to be both human- and machine-readable.
- XML is designed to store and transport data.
- Xml was released in late 90's. it was created to provide an easy to use and store self describing data.
- XML became a W3C Recommendation on February 10, 1998.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.(data that describes both its content and structure.)
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

Introduction to xml 10 Hrs.

Characteristics of XML

- *XML is extensible* − XML allows us to create your own self-descriptive tags, or language, that suits your application.
- *XML carries the data, does not present it* − XML allows us to store the data irrespective of how it will be presented.
- *XML is a public standard* − XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.
- *XML separates data from HTML*
- *XML simplifies data sharing*
- *XML simplifies data transport*
- *XML simplifies Platform change*
- *XML increases data availability*
- *XML can be used to create new internet languages*

HTML vs XML 10 Hrs.



No.	HTML	XML
1)	HTML is used **to display data** and focuses on how data looks.	XML is a software and hardware independent tool used **to transport and store data**. It focuses on what data is.
2)	HTML is a **markup language** itself.	XML provides a **framework to define markup languages.**
3)	HTML is **not case sensitive.**	XML is **case sensitive.**
4)	HTML is a presentation language.	XML is neither a presentation language nor a programming language.
5)	HTML **has its own predefined tags.**	You **can define tags according to your need.**
6)	In HTML, it is **not necessary to use a closing tag.**	XML **makes it mandatory to use a closing tag.**
7)	HTML is **static** because it is used to display data.	XML is **dynamic** because it is used to transport data.
8)	HTML **does not preserve whitespaces.**	XML **preserve whitespaces.**

Introduction to xml 10 Hrs.

Usages of xml in web development

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

Is XML a Programming Language?

- XML does not qualify to be a programming language as it does not perform any computation or algorithms.
- It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

Introduction to xml 10 Hrs.

XML Basics:

- Developed by W3C (World Wide Web Consortium).
- It is the pared--down version of SGML (Standardized Generalized Markup Language).
- Designed especially for web documents
- Allows you to create your own language for displaying documents.

"***XML is not the replacement of HTML.***"

- XML requires a processing application. There is no XML browser in market yet.
- XML isn't about display – it's about Structure.
- XML file should be saved with the extension "**.xml**".
- You can use any text editor to write XML code. And the output can be viewed in any browser.
- Every XML page consists of processing instruction at the very first line.
- There should be a root element in every XML document, which should be unique.

Introduction to xml 10 Hrs.

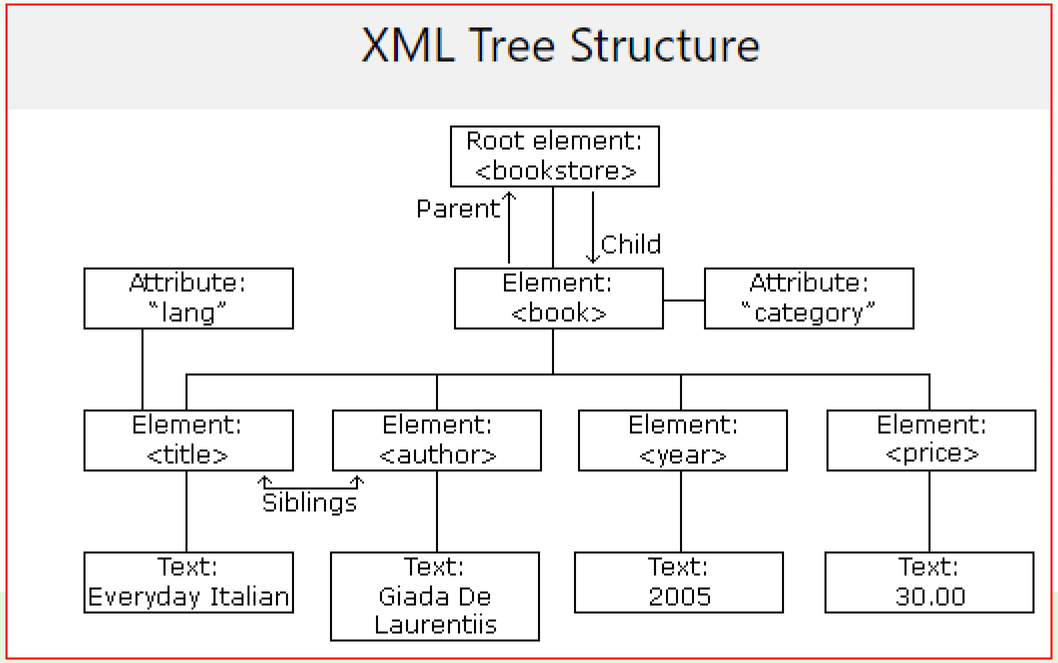
Advantages of XML:

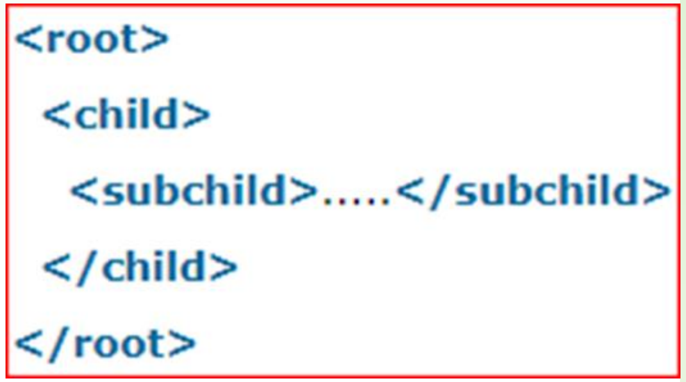
- The first benefit of XML is that because you are writing your own markup language, you are not restricted to a limited set of tags defined by proprietary vendors.
- With XML, you can create your own set of tags at your own pace.
- XML allows every person/organization to build their own tag library which suits their needs perfectly.
- XML allows you to define all sorts of tags with all sorts of rules, such as tags representing business rules or tags representing data description or data relationships.
- With XML, GUI is extracted. Thus, changes to display do not require futzing with the data. Instead separate style-sheet will specify a table display or a list display.
- Searching the data is easy and efficient. Search-engine can simply parse the description- bearing tags rather than muddling in the data. Tags provide the search engines with the intelligence they lack.
- Complex relationships like tree and inheritance can be communicated.
- The code is much more legible to the person coming into the environment with no prior knowledge.

Introduction to xml 10 Hrs.

Disadvantages of XML:

- XML requires a processing application.
- There are no XML browsers on the market yet. Thus, XML documents must either be converted into HTML before distribution or converting it to HTML on-the-fly by middleware.
- Barring translation, developers must code their own processing applications.
- XML isn't about display - it's about structure.
- This has implications that make the browser question secondary. So the whole issue of what is to be displayed and by what means is intentionally left to other applications.





```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Rules for writing XML 10 Hrs.

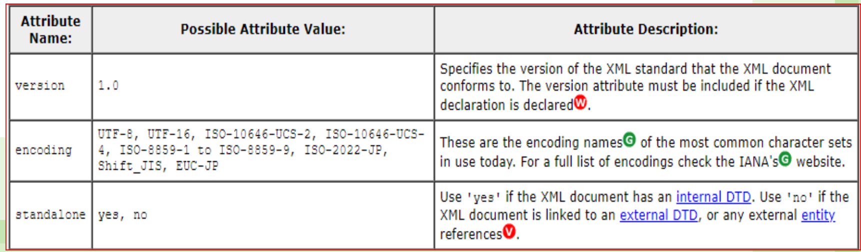
- The first line must be **<?XML version = “1.0” ?>**.
- Tags are enclosed in angle brackets.
- Tags are case sensitive and must have a matching closing tag.
- Tags may contain attributes in the form **name = “value”**.
- Tags may contain text, other tags, or both. Tags content lies between the starting and ending tag.

XML Declaration

- To begin an XML document, it is a good idea to include the XML declaration as the very first line of the document.
- The XML declaration is a processing instruction that notifies the processing agent that the following document has been marked up as an XML document.
- The XML declaration looks as following:
- <?xml version = "1.0“ ?>

The XML declaration might look like the following:

<?xml version = "1.0"? standalone = "yes" encoding = "UTF-8"?>

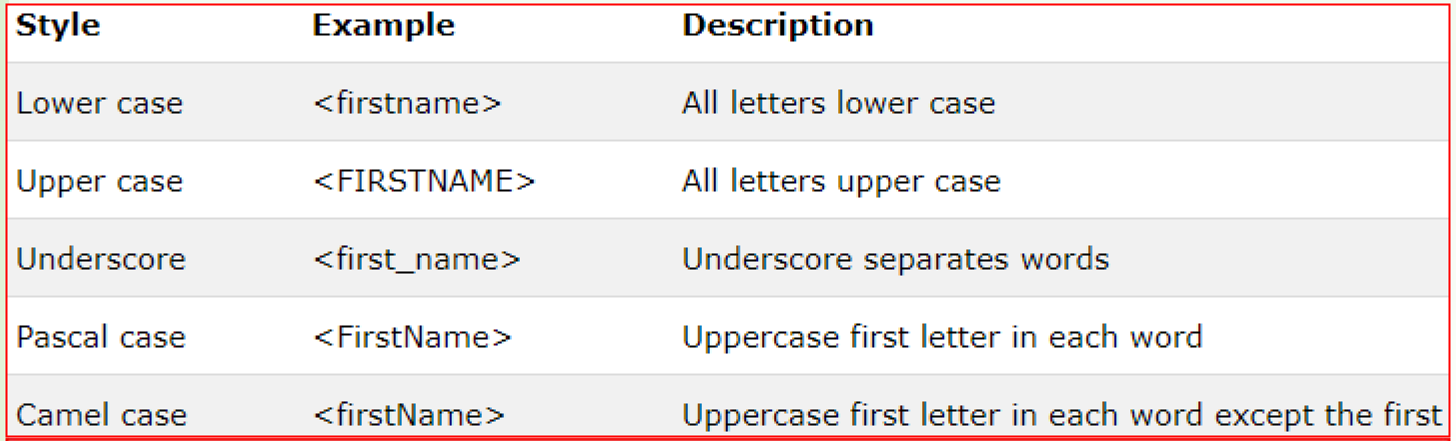


Attribute Name:	Possible Attribute Value:	Attribute Description:
`version`	`1.0`	Specifies the version of the XML standard that the XML document conforms to. The version attribute must be included if the XML declaration is declared.
`encoding`	`UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP`	These are the encoding names of the most common character sets in use today. For a full list of encodings check the IANA's website.
`standalone`	`yes, no`	Use `'yes'` if the XML document has an internal DTD. Use `'no'` if the XML document is linked to an external DTD, or any external entity references.

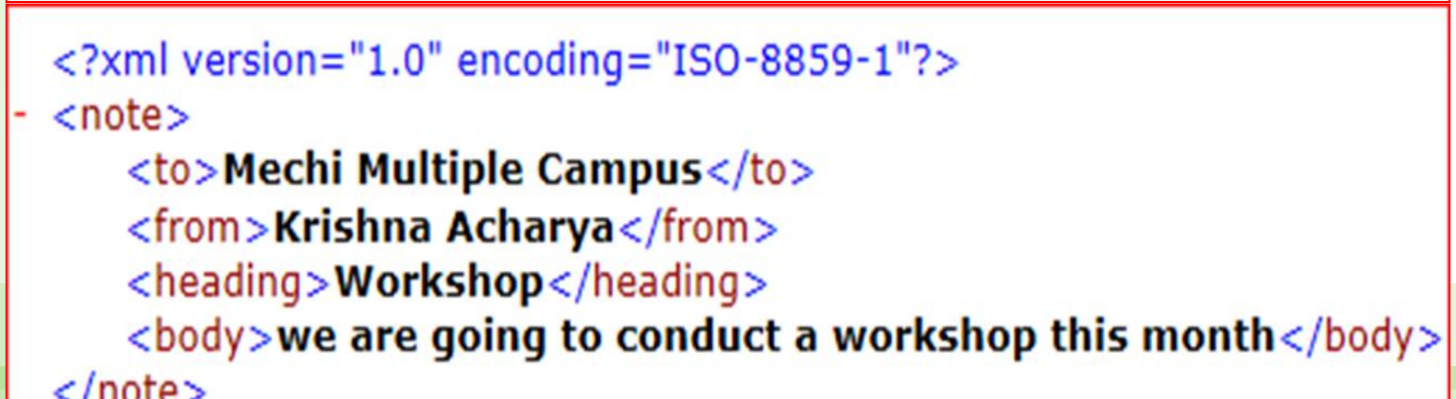
XML 10 Hrs.

Elements naming convention

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces.



Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first



```
  <?xml version="1.0" encoding="ISO-8859-1"?>
- <note>
    <to>Mechi Multiple Campus</to>
    <from>Krishna Acharya</from>
    <heading>Workshop</heading>
    <body>we are going to conduct a workshop this month</body>
  </note>
```

XML 10 Hrs.

Elements

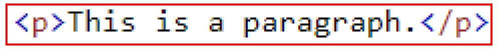
- Elements are the basic unit of XML content.
- Syntactically, an element consists of a start tag, and an end tag, and everything in between.

<u>*Example:*</u>

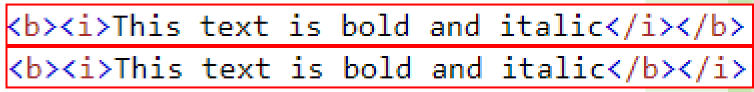
<NAME>Krishna Acharya </NAME>

- All XML documents must have at least one root element to be well formed. The root element, also often called the document tag, must follow the prolog (XML declaration plus DTD) and must be a nonempty tag that encompasses the entire document.
- XML defines the text between the start and end tags to be "character data" and the text within the tags to be "markup".

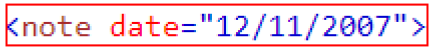
Rules for Elements

- XML documents can have only one root element.
- Every starting tag must have a matching end tag. 

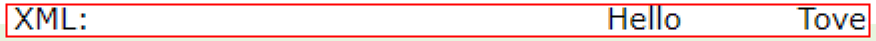
```
<p>This is a paragraph.</p>
```

- XML elements must be properly but must not must not overlapped 

```
<b><i>This text is bold and italic</i></b>
<b><i>This text is bold and italic</b></i>
```

- XML attribute values must always be quoted 

```
<note date="12/11/2007">
```

- XML is case sensitive. The tag <Letter> is different from the tag <letter>.
- white-space is preserved in xml. 

```
XML:                                   Hello        Tove
```

XML 10 Hrs.

Rules for Elements

- Comments and processing instructions may not appear inside tags. `<?xml <!-- version="1.0"--> encoding="UTF-8"?>`

valid `<!-- This is a comment -->` **invalid** `<!-- This is an invalid -- comment -->`

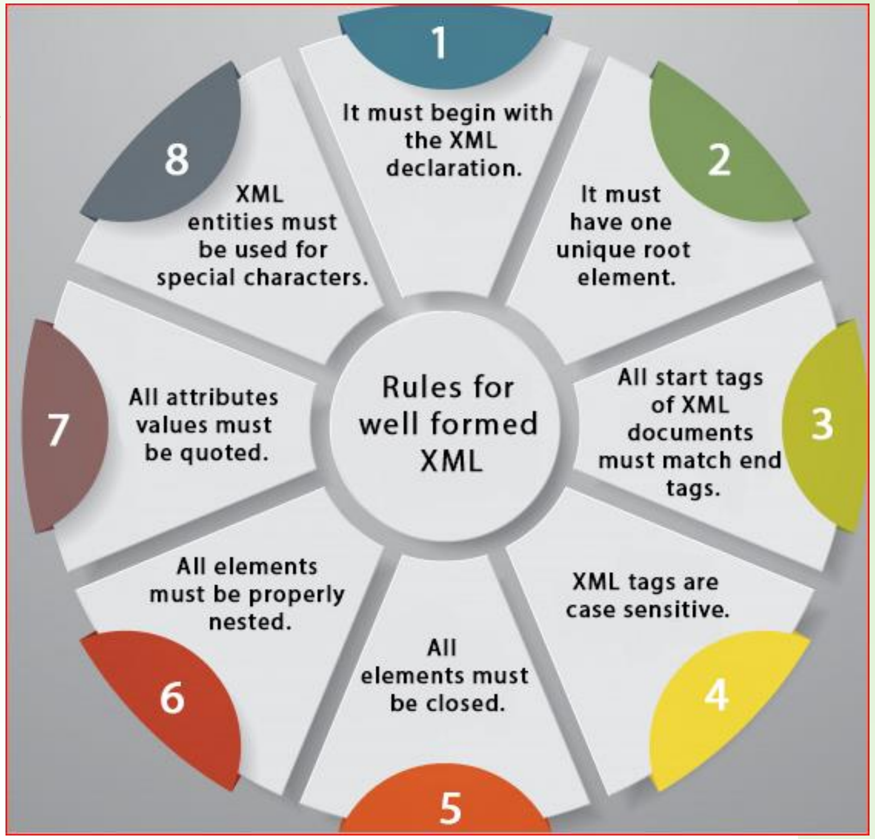
- No unescaped < or & signs may occur inside character data.

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

```
<message>salary < 1000</message>
```

valid

```
<message>salary &lt; 1000</message>
```



Rules for well formed XML

1. It must begin with the XML declaration.
2. It must have one unique root element.
3. All start tags of XML documents must match end tags.
4. XML tags are case sensitive.
5. All elements must be closed.
6. All elements must be properly nested.
7. All attributes values must be quoted.
8. XML entities must be used for special characters.

- XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

XML

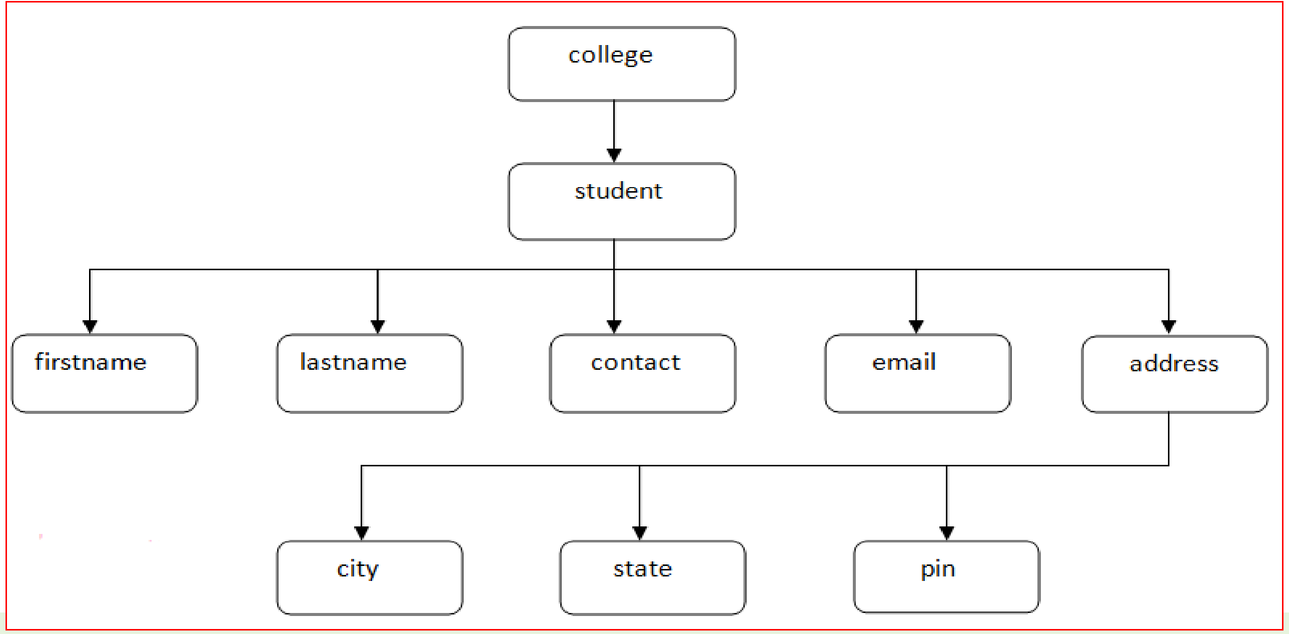
XML Attributes

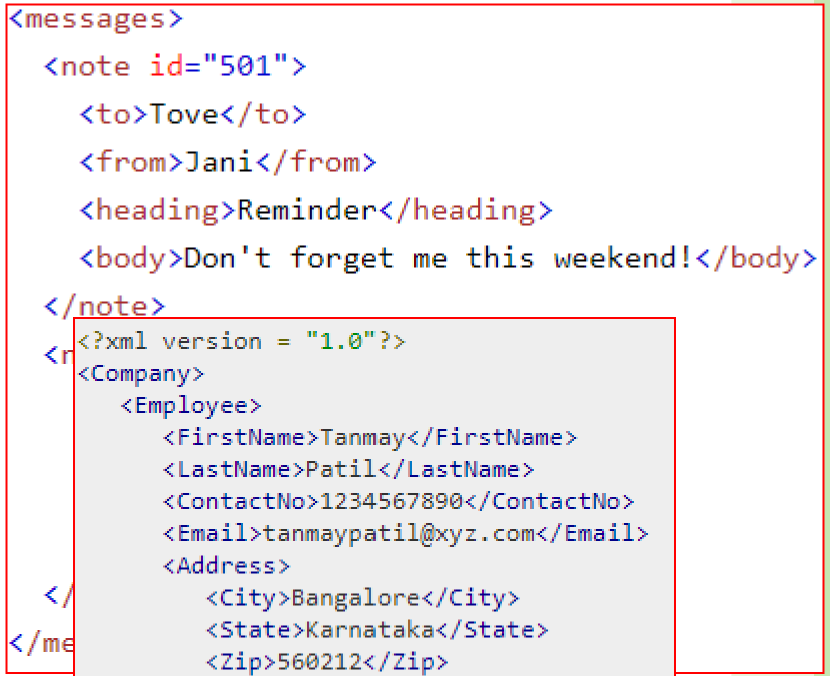
- XML elements can have attributes. By the use of attributes we can add the information about the element. XML attributes enhance the properties of the elements.

Avoid XML Attributes?

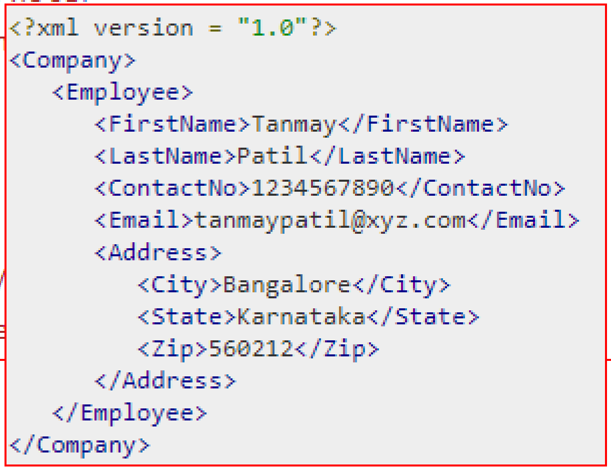
- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

Q1. Create well-form xml document form bellow figure.





```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <n

  </
</me
```



```
<?xml version = "1.0"?>
<Company>
   <Employee>
      <FirstName>Tanmay</FirstName>
      <LastName>Patil</LastName>
      <ContactNo>1234567890</ContactNo>
      <Email>tanmaypatil@xyz.com</Email>
      <Address>
         <City>Bangalore</City>
         <State>Karnataka</State>
         <Zip>560212</Zip>
      </Address>
   </Employee>
</Company>
```

XML

CDATA :

- The term CDATA means, Character Data. CDATA is defined as blocks of text that are not parsed by the parser, but are otherwise recognized as markup.
- The predefined entities such as **&lt;**, **&gt;**, and **&amp;** require typing and are generally difficult to read in the markup.
- In such cases, CDATA section can be used. By using CDATA section, you are commanding the parser that the particular section of the document contains no markup and should be treated as regular text.

```
<?xml version="1.0" encoding="UTF-8"?>
<script>
    <![CDATA[
      <message> Welcome to Mecchi Multiple Campus </message>
   ]]>
</script>
```

- The above markup code shows an example of CDATA. Here, each character written inside the CDATA section is ignored by the parser.
- In the above syntax, everything between <message> and </message> is treated as character data and not as markup.

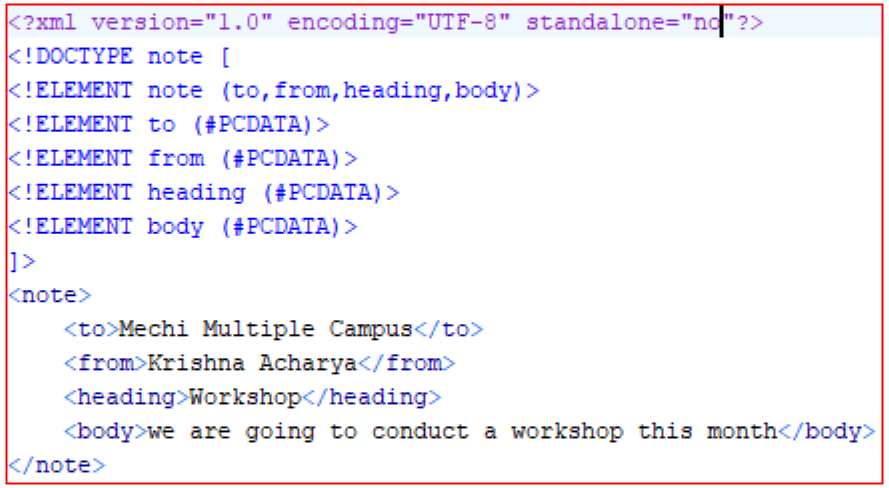
XML

Processing Instructions (PIs)

- Processing instructions (PIs) allow documents to contain instructions for applications. PIs are not part of the character data of the document, but MUST be passed through to the application.
- Processing instructions (PIs) can be used to pass information to applications. PIs can appear anywhere in the document outside the markup. They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document.
- Eg <?xml-stylesheet href = "style.css" type = "text/css"?>

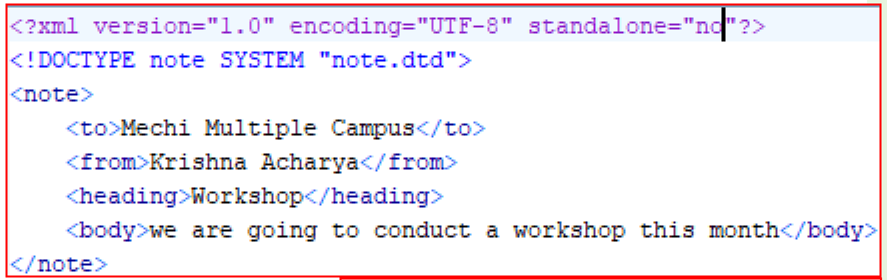
XML DTD

- The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely.
- DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.
- An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.
- with a DTD, independent groups of people can agree on a standard DTD for interchanging data.
- An application can use a DTD to verify that XML data is valid.



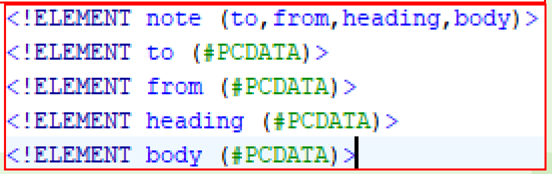
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
    <to>Mechi Multiple Campus</to>
    <from>Krishna Acharya</from>
    <heading>Workshop</heading>
    <body>we are going to conduct a workshop this month</body>
</note>
```

Internal vs External



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
    <to>Mechi Multiple Campus</to>
    <from>Krishna Acharya</from>
    <heading>Workshop</heading>
    <body>we are going to conduct a workshop this month</body>
</note>
```

note.dtd



```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

XML DTD

- Element content Categories

Content Category	Description
ANY	Element type may contain any well formed XML data.
EMPTY	Element type may contain any text or child elements-- only elements attributes are permitted.
Element	Element type contains only child elements no additional text is permitted.
Mixed	Element type may contain text and/or child element.
PCDATA	Element type may contain text (character data) only.

Cardinality

Operators	Description
[none]	The absence of a cardinality operator character indicates that one, and only one, instance of child element is allowed (required).
?	Zero or one element – optional singular element.
*	Zero or more element – optional element(s).
+	One or more child elements – required element(s).

XML DTD

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element must contain four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
- **!ELEMENT body** defines the body element to be of type "#PCDATA"

PCDATA

- #PCDATA means parsed character data.(parse-able text data.)
- Think of character data as the text found between the start tag and the end tag of an XML element.
- **PCDATA is text that WILL be parsed by a parser**. **The text will be examined by the parser for entities and markup**.
- Tags inside the text will be treated as markup and entities will be expanded.
- However, parsed character data should not contain any &, <, or > characters; these need to be represented by the &amp; &lt; and &gt; entities, respectively.

XML DTD

Declaring Elements

<!ELEMENT element-name (element-content)>

Empty Elements

<!ELEMENT br EMPTY>

Elements with Parsed Character Data

<!ELEMENT element-name (#PCDATA)>

<!ELEMENT from (#PCDATA)>

Elements with any Contents(an contain any combination of parsable data**)**

<!ELEMENT element-name ANY>
Example:
<!ELEMENT note ANY>(can contain any combination of parsable data)

Elements with Children (sequences)

<!ELEMENT element-name (child1,child2,...)>
Example:
<!ELEMENT note (to,from,heading,body)>

XML DTD

Declaring Only One Occurrence of an Element

<!ELEMENT element-name (child-name)>
Example:
<!ELEMENT note (message)>(child element "message" must occur once, and only once inside the "note" element.)

Declaring Minimum One Occurrence of an Element

<!ELEMENT element-name (child-name+)>
Example:
<!ELEMENT note (message+)>(the child element "message" must occur one or more times inside the "note" element.)

Declaring Zero or More Occurrences of an Element

<!ELEMENT element-name (child-name*)>
Example:
<!ELEMENT note (message*)>

Declaring either/or Content

<!ELEMENT note (to,from,header,(message|body))>

Declaring Zero or One Occurrences of an Element

<!ELEMENT note (message?)>

XML DTD

Declaring Mixed Content

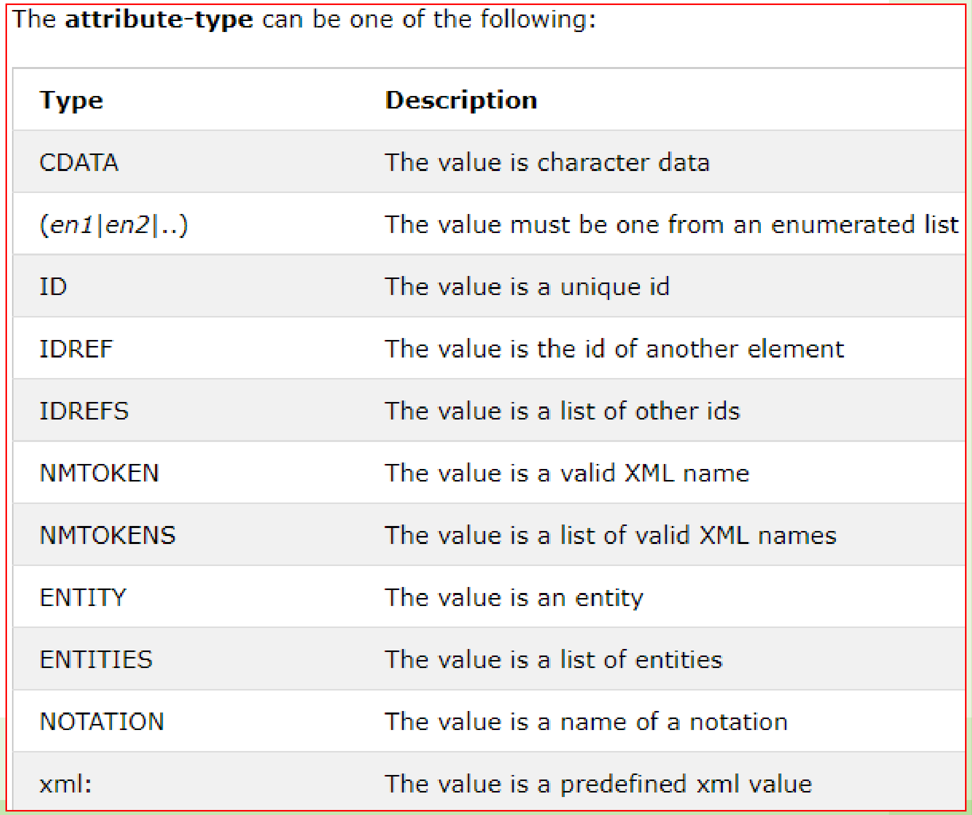
<!ELEMENT note (#PCDATA | to | from | header | message)*>

ATTRIBUTES

In a DTD, attributes are declared with an ATTLIST declaration.

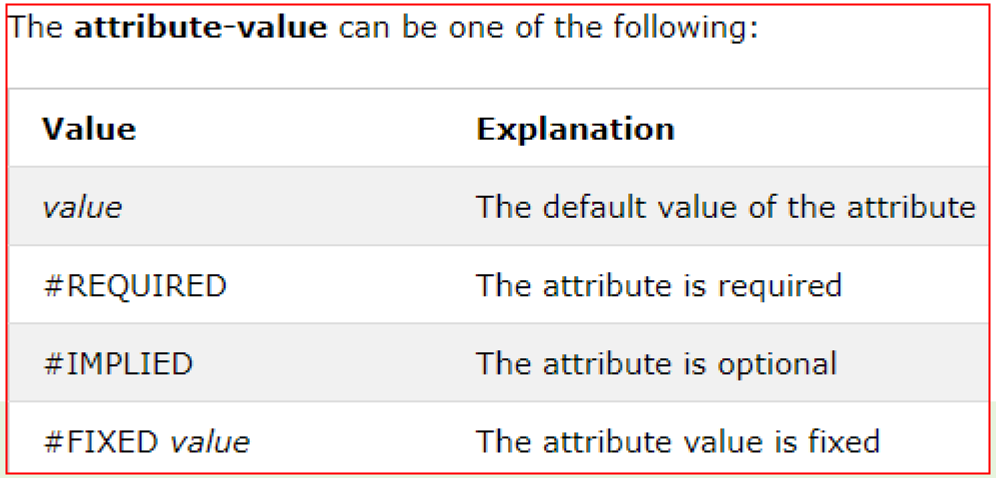
<!ATTLIST element-name attribute-name attribute-type attribute-value>

<!ATTLIST payment type CDATA "check">
XML example:
<payment type="check" />



The **attribute-type** can be one of the following:

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value



The **attribute-value** can be one of the following:

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

XML DTD

A Default Attribute Value

DTD:
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
Valid XML:
<square width="100" />

#REQUIRED

DTD:
<!ATTLIST person number CDATA #REQUIRED>

<!ATTLIS Valid XML:
<person number="5677" />
Invalid XML:
<person /> T element-name attribute-name attribute-type #REQUIRED>

#IMPLIED

<!ATTLIST element-name attribute-name attribute-type #IMPLIED>

DTD:
<!ATTLIST contact fax CDATA #IMPLIED>
Valid XML:
<contact fax="555-667788" />
Valid XML:
<contact />

XML DTD

#FIXED

<!ATTLIST element-name attribute-name attribute-type #FIXED "value">

DTD:
<!ATTLIST sender company CDATA #FIXED "Microsoft">
Valid XML:
<sender company="Microsoft" />
Invalid XML:
<sender company="Mechicampus" />

Enumerated Attribute Values

<!ATTLIST element-name attribute-name (en1|en2|..) default-value>

DTD:
<!ATTLIST payment type (check|cash) "cash">
XML example:
<payment type="check" />
or
<payment type="cash" />

DTD - Entities

Entities are used to define shortcuts to special characters. Entities can be declared internal or external.

<!ENTITY entity-name "entity-value">

DTD Example:

<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">
XML example:
<author>&writer;&copyright;</author>

XML DTD

Avoid using attributes?

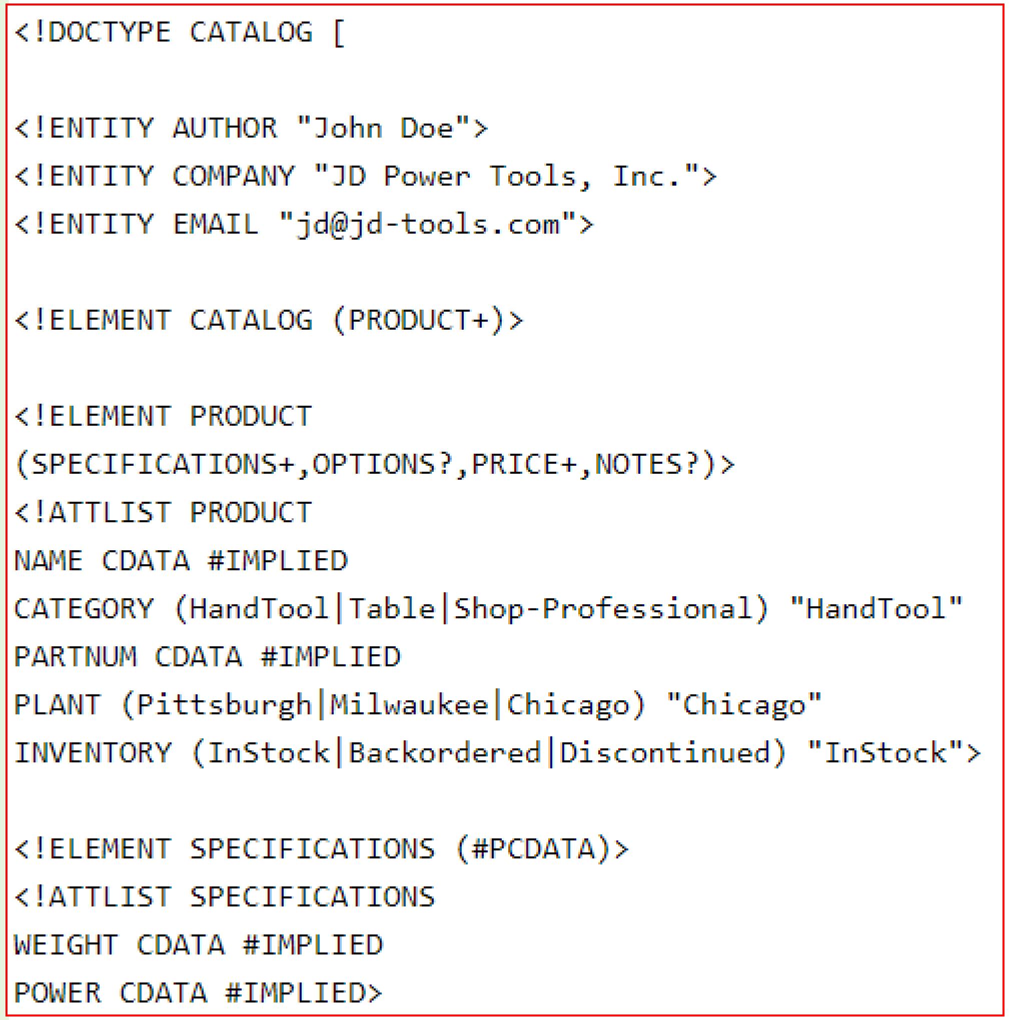
- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- attributes cannot describe structures (child elements can)
- attributes are more difficult to manipulate by program code
- attribute values are not easy to test against a DTD

```
<!DOCTYPE TVSCHEDULE [

<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>

<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

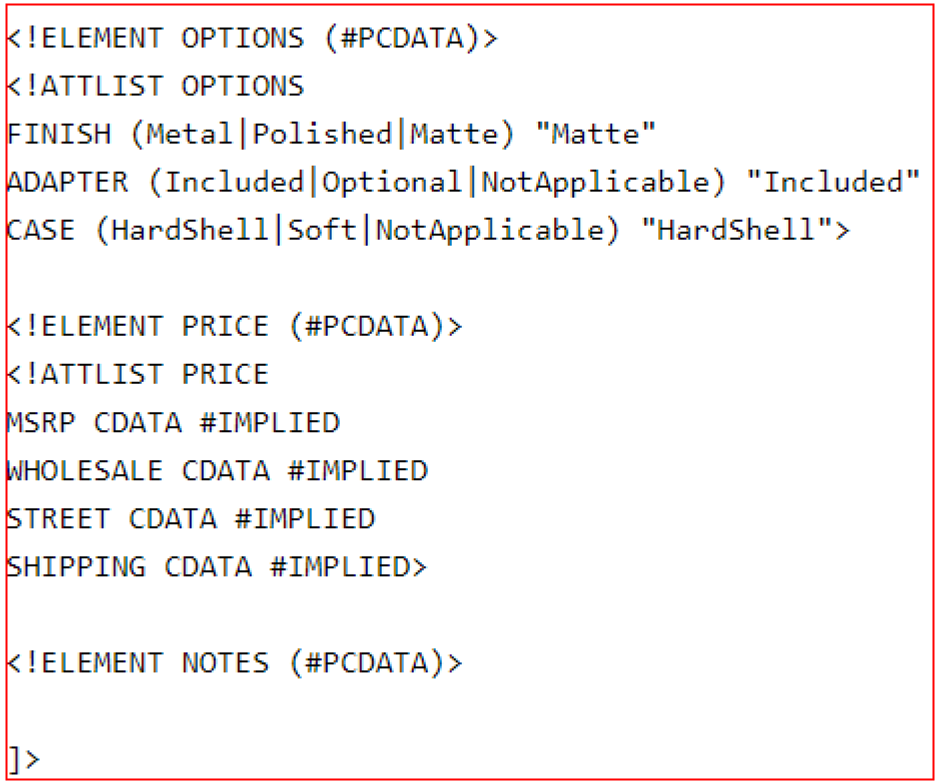
```
<!DOCTYPE NEWSPAPER [

<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

<!ENTITY NEWSPAPER "Vervet Logic Times">
<!ENTITY PUBLISHER "Vervet Logic Press">
<!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">

]>
```

1

XML DTD (Product Catalog DTD)



```
<!DOCTYPE CATALOG [

<!ENTITY AUTHOR "John Doe">
<!ENTITY COMPANY "JD Power Tools, Inc.">
<!ENTITY EMAIL "jd@jd-tools.com">

<!ELEMENT CATALOG (PRODUCT+)>

<!ELEMENT PRODUCT
(SPECIFICATIONS+,OPTIONS?,PRICE+,NOTES?)>
<!ATTLIST PRODUCT
NAME CDATA #IMPLIED
CATEGORY (HandTool|Table|Shop-Professional) "HandTool"
PARTNUM CDATA #IMPLIED
PLANT (Pittsburgh|Milwaukee|Chicago) "Chicago"
INVENTORY (InStock|Backordered|Discontinued) "InStock">

<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ATTLIST SPECIFICATIONS
WEIGHT CDATA #IMPLIED
POWER CDATA #IMPLIED>
```



```
<!ELEMENT OPTIONS (#PCDATA)>
<!ATTLIST OPTIONS
FINISH (Metal|Polished|Matte) "Matte"
ADAPTER (Included|Optional|NotApplicable) "Included"
CASE (HardShell|Soft|NotApplicable) "HardShell">

<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST PRICE
MSRP CDATA #IMPLIED
WHOLESALE CDATA #IMPLIED
STREET CDATA #IMPLIED
SHIPPING CDATA #IMPLIED>

<!ELEMENT NOTES (#PCDATA)>

]>
```

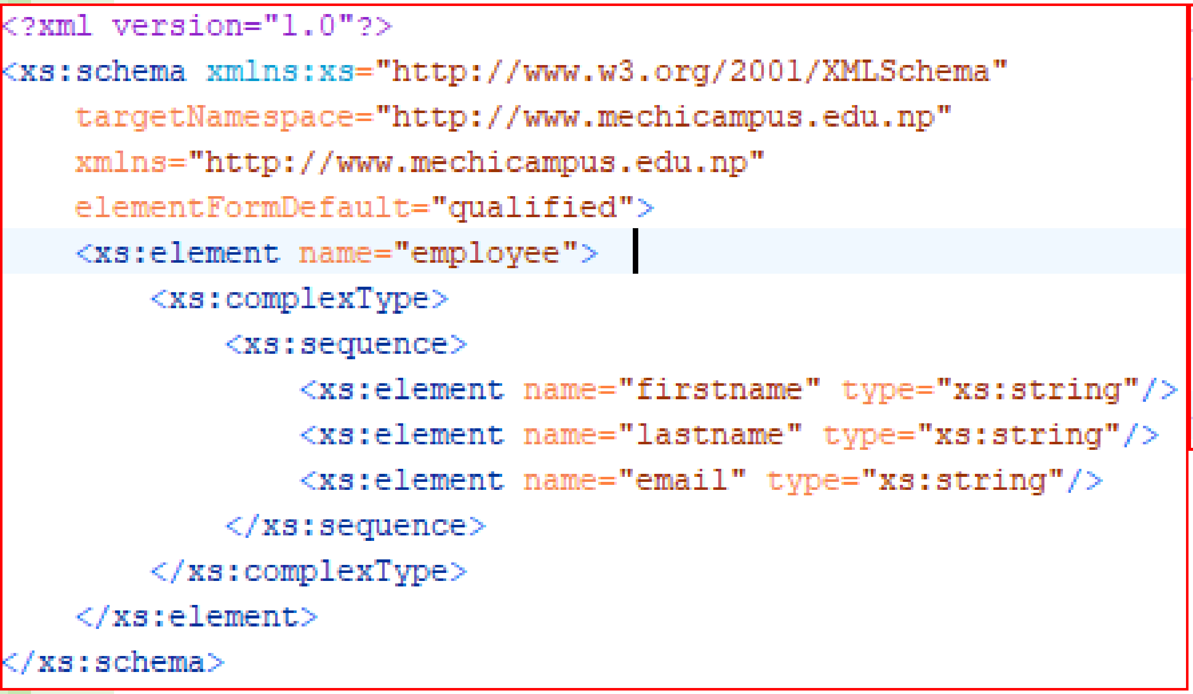
1

Limitation of DTD

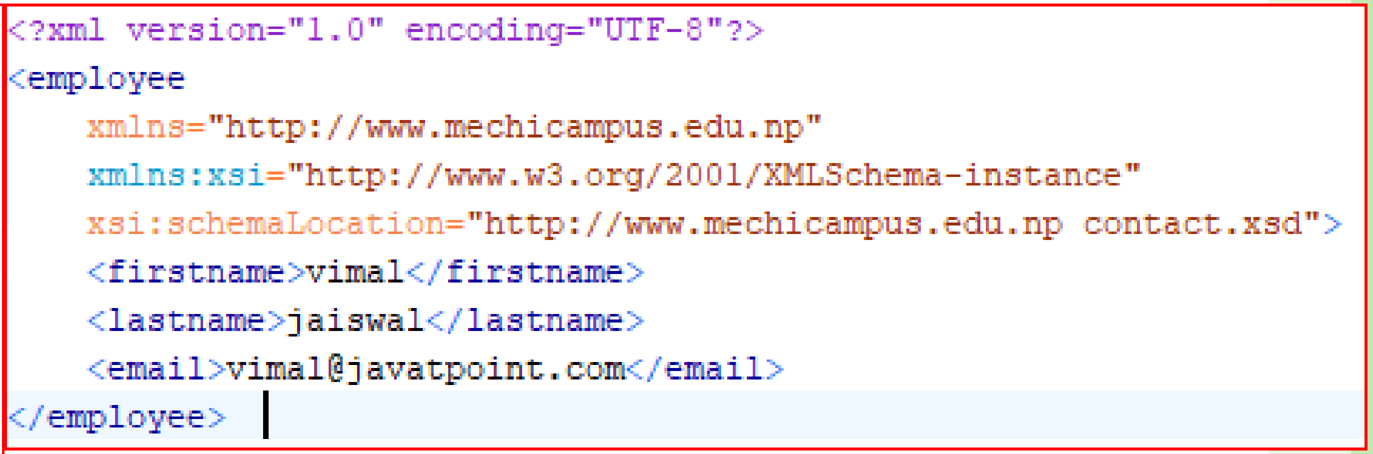
- DTD are not extensible, unlike XML itself.
- Only one DTD may be associated with each XML document.
- DTDs do not work well with XML namespaces.
- Supports very weak data typing.
- Limited content model descriptions.
- No object oriented type object inheritance.
- A document can override / ignore an external DTD using internal subset.
- Non-XML syntax.
- No DOM support.
- Relatively few older, more expensive tools.
- Very limited support to modularity and reuse.
- Too simple ID attribute mechanism (no points to requirements, uniqueness scope, etc)

XML XSD

- XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data.
- XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

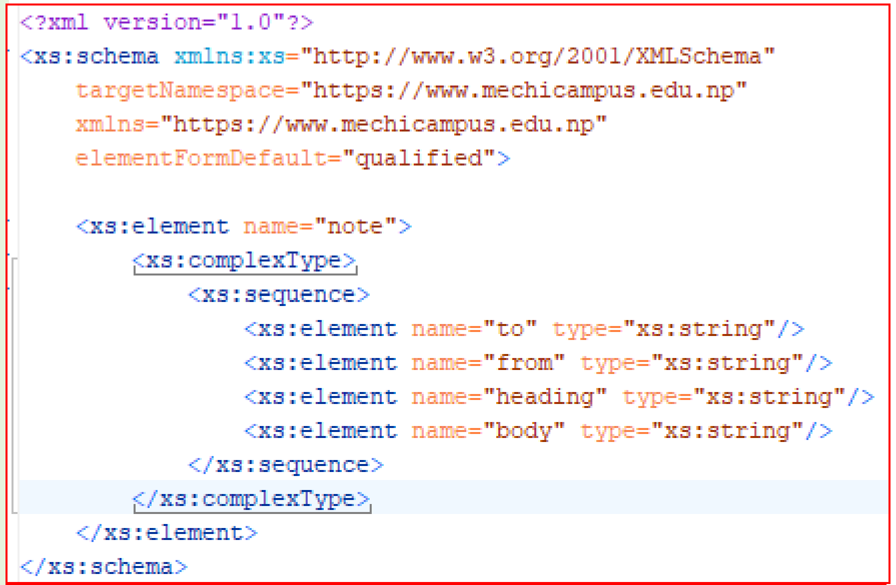


```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.mechicampus.edu.np"
    xmlns="http://www.mechicampus.edu.np"
    elementFormDefault="qualified">
    <xs:element name="employee">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="firstname" type="xs:string"/>
                <xs:element name="lastname" type="xs:string"/>
                <xs:element name="email" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

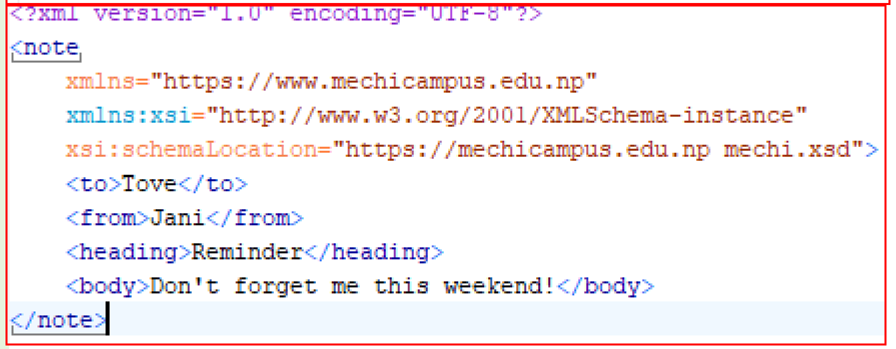


```xml
<?xml version="1.0" encoding="UTF-8"?>
<employee
    xmlns="http://www.mechicampus.edu.np"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.mechicampus.edu.np contact.xsd">
    <firstname>vimal</firstname>
    <lastname>jaiswal</lastname>
    <email>vimal@javatpoint.com</email>
</employee>
```

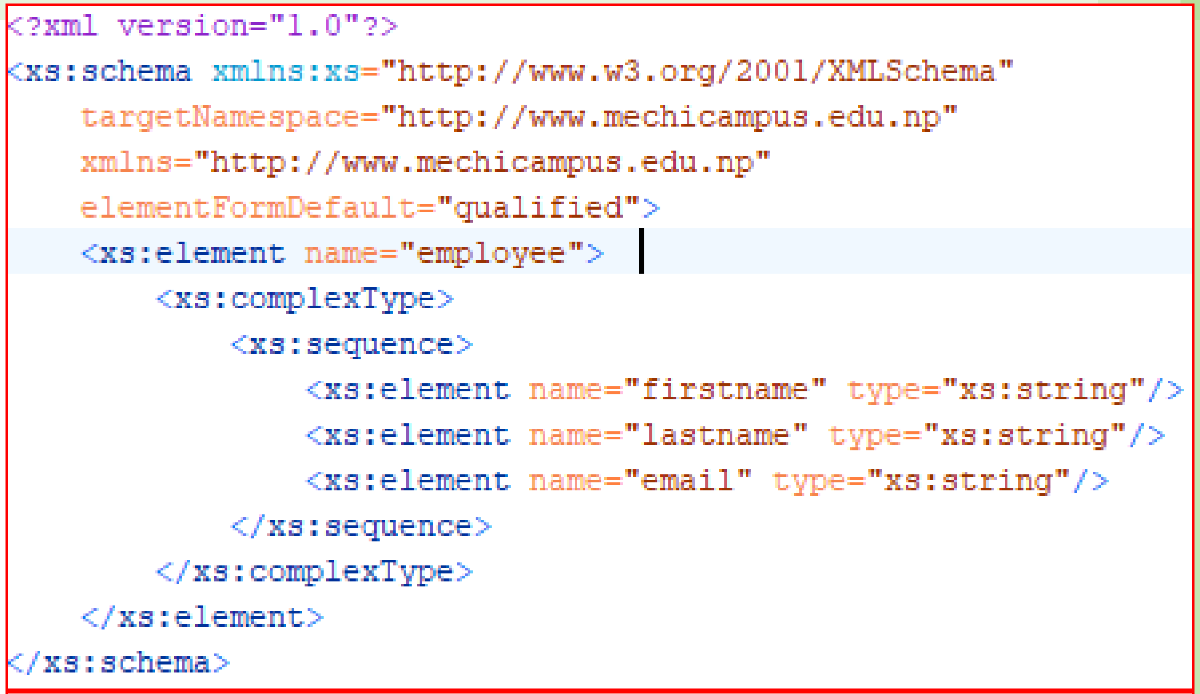
XML XSD



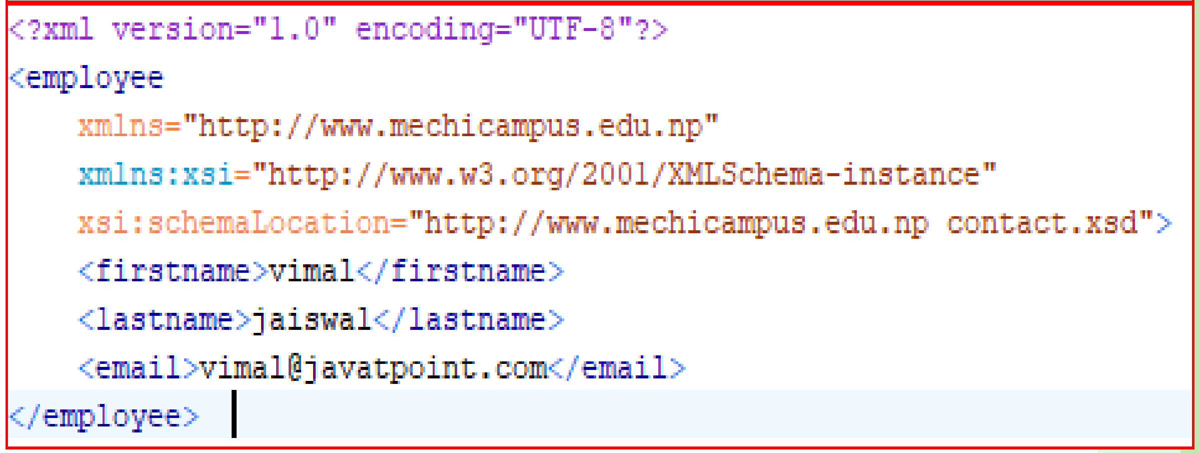
```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="https://www.mechicampus.edu.np"
    xmlns="https://www.mechicampus.edu.np"
    elementFormDefault="qualified">

    <xs:element name="note">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="to" type="xs:string"/>
                <xs:element name="from" type="xs:string"/>
                <xs:element name="heading" type="xs:string"/>
                <xs:element name="body" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<note
    xmlns="https://www.mechicampus.edu.np"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://mechicampus.edu.np mechi.xsd">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

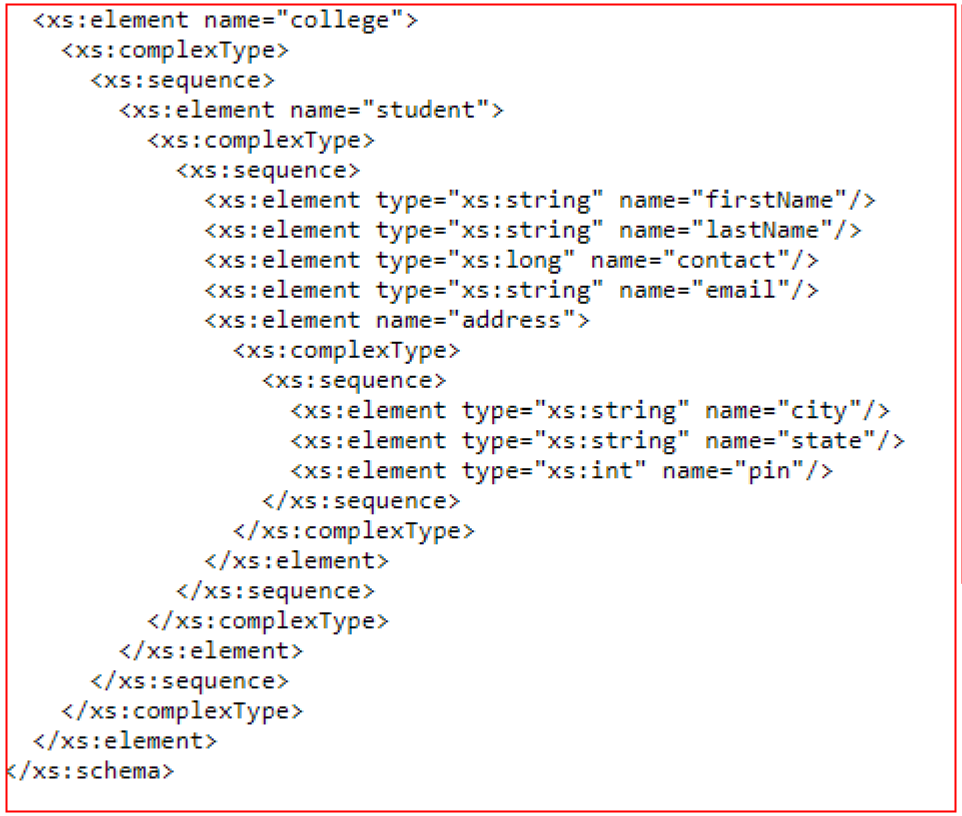


```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.mechicampus.edu.np"
    xmlns="http://www.mechicampus.edu.np"
    elementFormDefault="qualified">
    <xs:element name="employee">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="firstname" type="xs:string"/>
                <xs:element name="lastname" type="xs:string"/>
                <xs:element name="email" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

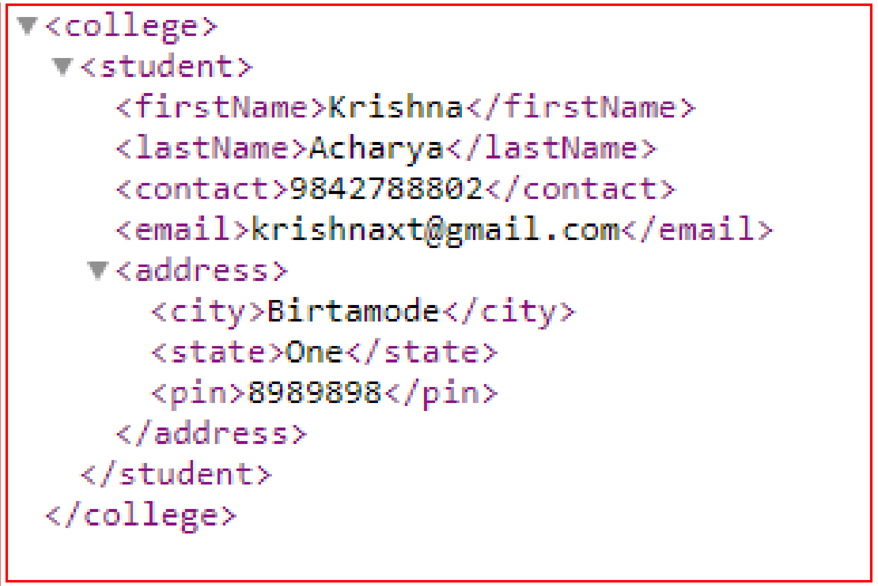


```
<?xml version="1.0" encoding="UTF-8"?>
<employee
    xmlns="http://www.mechicampus.edu.np"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.mechicampus.edu.np contact.xsd">
    <firstname>vimal</firstname>
    <lastname>jaiswal</lastname>
    <email>vimal@javatpoint.com</email>
</employee>
```

XML XSD

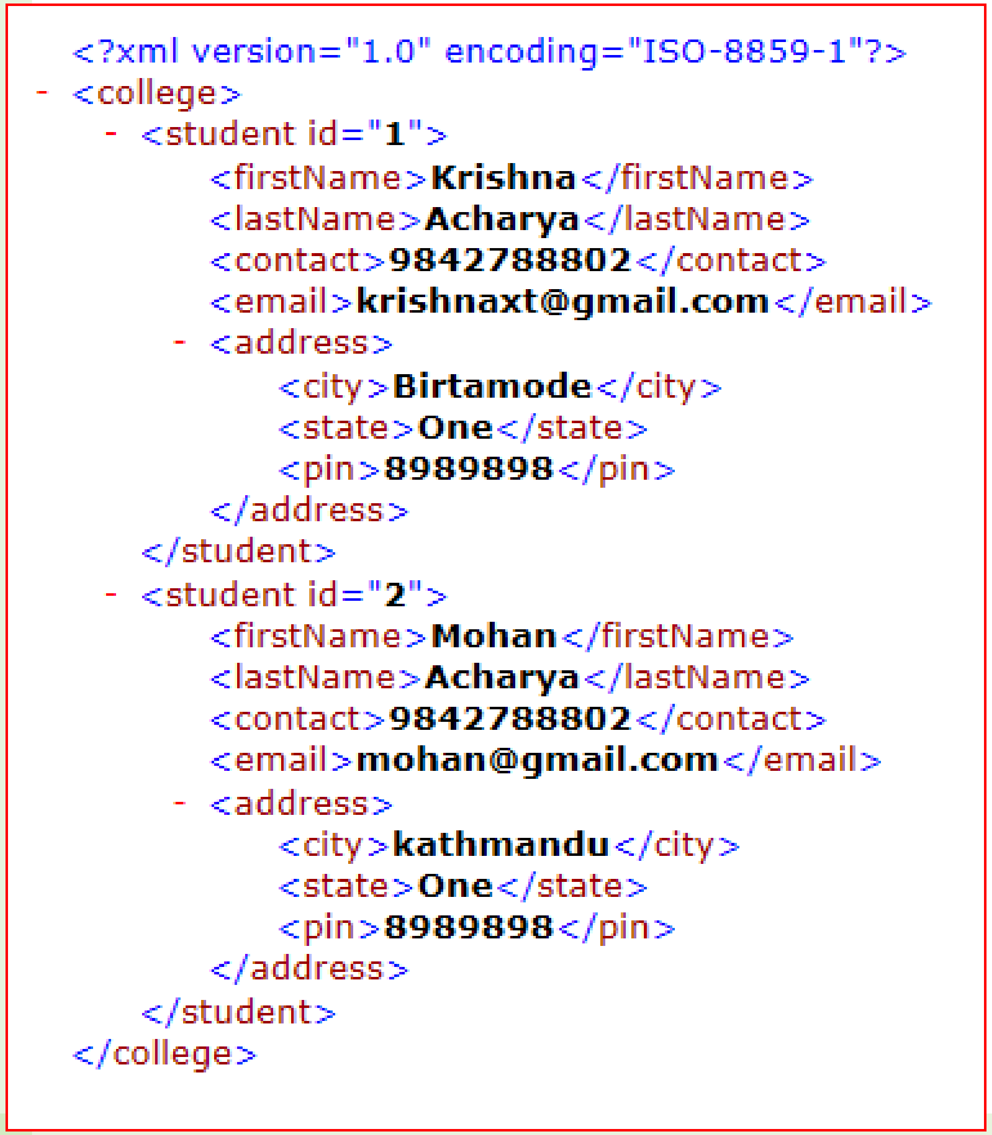


```
  <xs:element name="college">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="student">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="firstName"/>
              <xs:element type="xs:string" name="lastName"/>
              <xs:element type="xs:long" name="contact"/>
              <xs:element type="xs:string" name="email"/>
              <xs:element name="address">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="city"/>
                    <xs:element type="xs:string" name="state"/>
                    <xs:element type="xs:int" name="pin"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

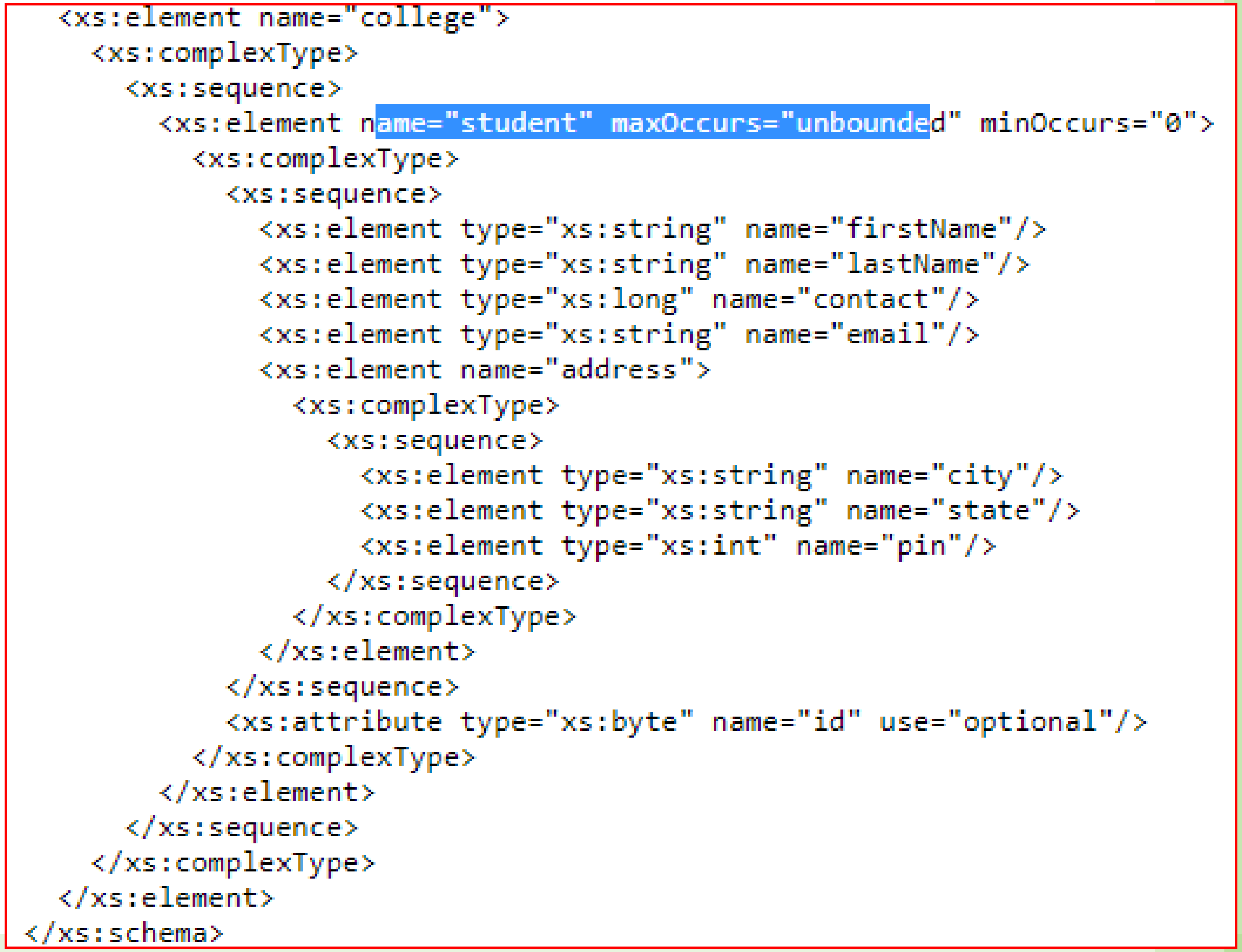


```
<college>
  <student>
    <firstName>Krishna</firstName>
    <lastName>Acharya</lastName>
    <contact>9842788802</contact>
    <email>krishnaxt@gmail.com</email>
    <address>
      <city>Birtamode</city>
      <state>One</state>
      <pin>8989898</pin>
    </address>
  </student>
</college>
```

XML XSD

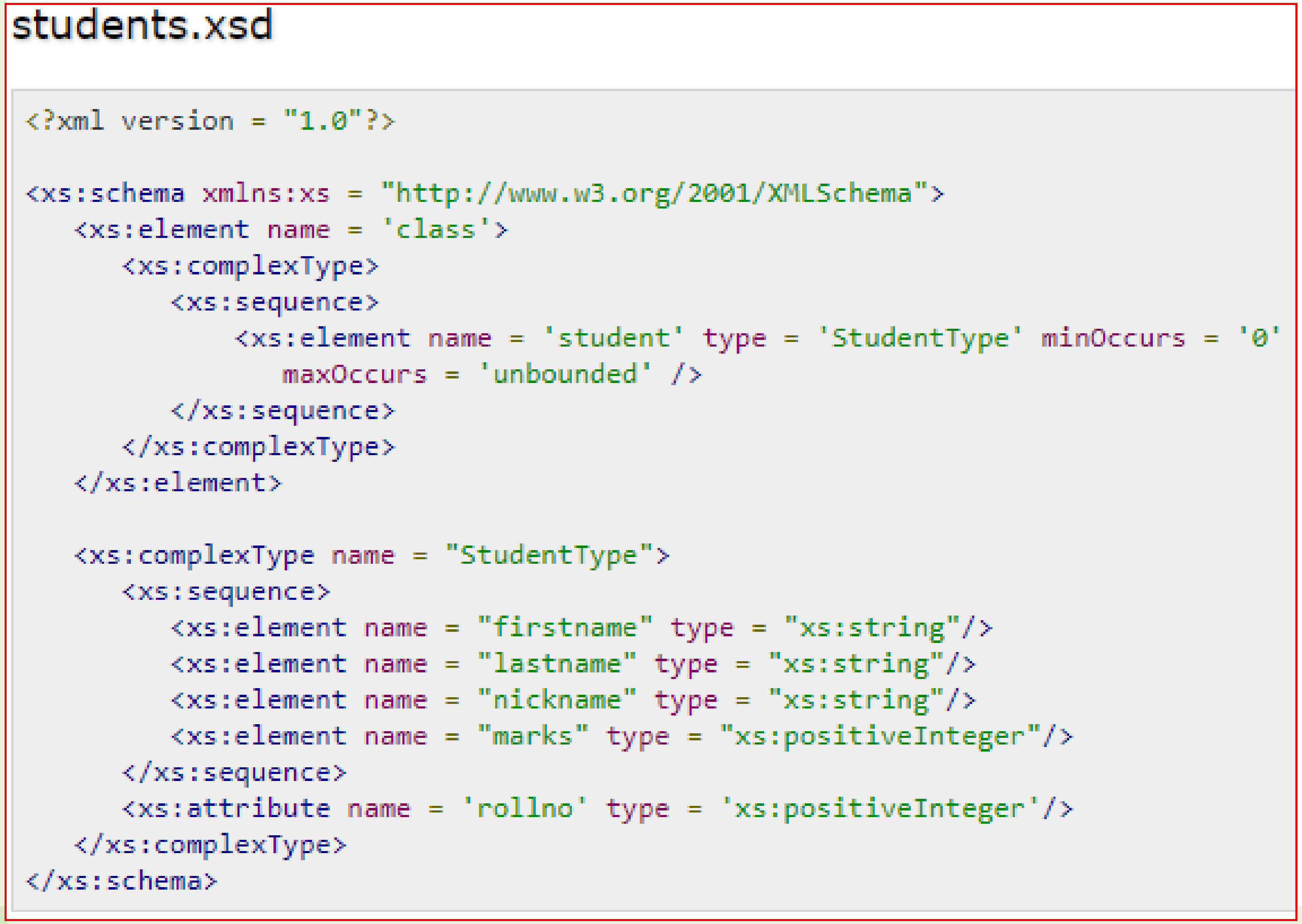


```
  <?xml version="1.0" encoding="ISO-8859-1"?>
- <college>
  - <student id="1">
      <firstName>Krishna</firstName>
      <lastName>Acharya</lastName>
      <contact>9842788802</contact>
      <email>krishnaxt@gmail.com</email>
    - <address>
        <city>Birtamode</city>
        <state>One</state>
        <pin>8989898</pin>
      </address>
    </student>
  - <student id="2">
      <firstName>Mohan</firstName>
      <lastName>Acharya</lastName>
      <contact>9842788802</contact>
      <email>mohan@gmail.com</email>
    - <address>
        <city>kathmandu</city>
        <state>One</state>
        <pin>8989898</pin>
      </address>
    </student>
  </college>
```



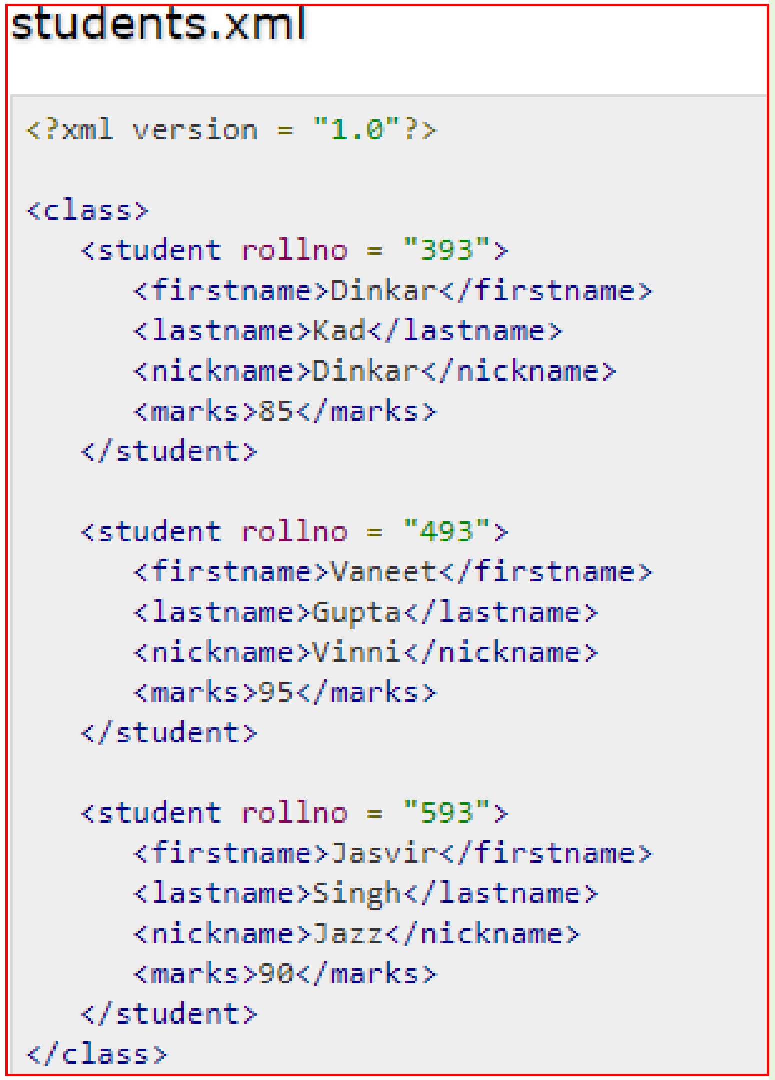
```
  <xs:element name="college">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="student" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="firstName"/>
              <xs:element type="xs:string" name="lastName"/>
              <xs:element type="xs:long" name="contact"/>
              <xs:element type="xs:string" name="email"/>
              <xs:element name="address">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="city"/>
                    <xs:element type="xs:string" name="state"/>
                    <xs:element type="xs:int" name="pin"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute type="xs:byte" name="id" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML XSD



students.xsd

```
<?xml version = "1.0"?>

<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
   <xs:element name = 'class'>
      <xs:complexType>
         <xs:sequence>
             <xs:element name = 'student' type = 'StudentType' minOccurs = '0'
                maxOccurs = 'unbounded' />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:complexType name = "StudentType">
      <xs:sequence>
         <xs:element name = "firstname" type = "xs:string"/>
         <xs:element name = "lastname" type = "xs:string"/>
         <xs:element name = "nickname" type = "xs:string"/>
         <xs:element name = "marks" type = "xs:positiveInteger"/>
      </xs:sequence>
      <xs:attribute name = 'rollno' type = 'xs:positiveInteger'/>
   </xs:complexType>
</xs:schema>
```



students.xml

```
<?xml version = "1.0"?>

<class>
   <student rollno = "393">
      <firstname>Dinkar</firstname>
      <lastname>Kad</lastname>
      <nickname>Dinkar</nickname>
      <marks>85</marks>
   </student>

   <student rollno = "493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>Vinni</nickname>
      <marks>95</marks>
   </student>

   <student rollno = "593">
      <firstname>Jasvir</firstname>
      <lastname>Singh</lastname>
      <nickname>Jazz</nickname>
      <marks>90</marks>
   </student>
</class>
```

XML XSD

Attributes

Attributes in XSD provide extra information within an element. Attributes have name and type property as shown below −

<xs:attribute name = "x" type = "y"/>

Elements

<xs:element name = "x" type = "y"/>

Definition Types

- simple Type

Simple type element is used only in the context of the text. Some of the predefined simple types are: xs:string,xs:decimal,xs:integer,xs:Boolean,xs:date,xs:time

- Complex Type

A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents.

<xs:complexType>

Default and Fixed Values for Attributes

<xs:attribute name="lang" type="xs:string" default="EN"/> you can define new value

<xs:attribute name="lang" type="xs:string" fixed="EN"/> you can not override it

XML XSD

<lastname lang="EN">Smith</lastname> in xml

<xs:attribute name="lang" type="xs:string"/> in xsd

Optional and Required Attributes:[Attributes are optional by default. To specify that the attribute is required, use the "use" attribute**]**

<xs:attribute name="lang" type="xs:string" use="required"/>

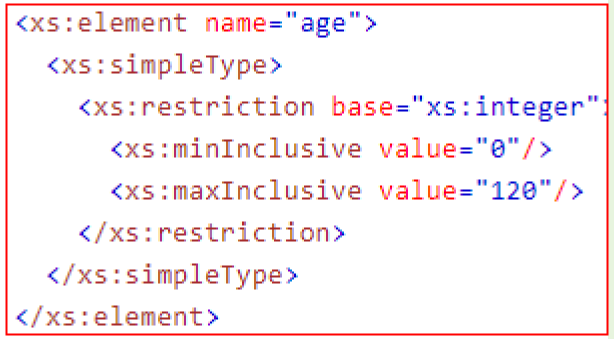
Restrictions on Content

- When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content.
- If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate.
- With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called **facets**.

XML XSD

Restrictions on Values

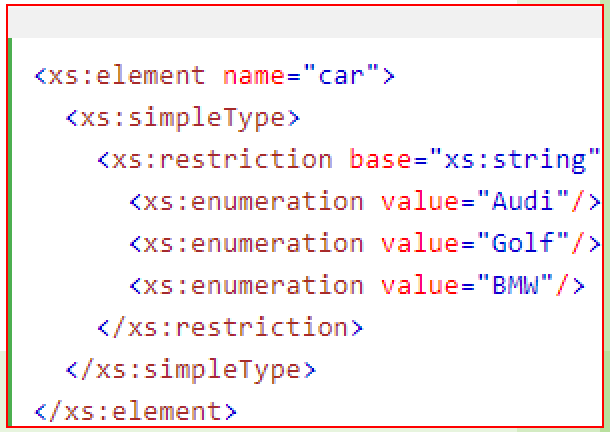
- The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 120:



```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer"
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions on a Set of Values

- To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.
- The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:



```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string"
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML XSD

Restrictions on a Series of Values

- To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.
- The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- defines an element called "initials" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to z:

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML XSD

Restrictions on a Series of Values

- An element called "choice" with a restriction. The only acceptable value is ONE of the following letters: x, y, OR z:

```
<xs:element name="choice">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[xyz]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- An element called "initials" with a restriction. The only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to z:

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML XSD

Restrictions on a Series of Values

- an element called "prodid" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9.

```
<xs:element name="prodid">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- an element called "letter" with a restriction. The acceptable value is zero or more occurrences of lowercase letters from a to z:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML XSD

Restrictions on a Series of Values

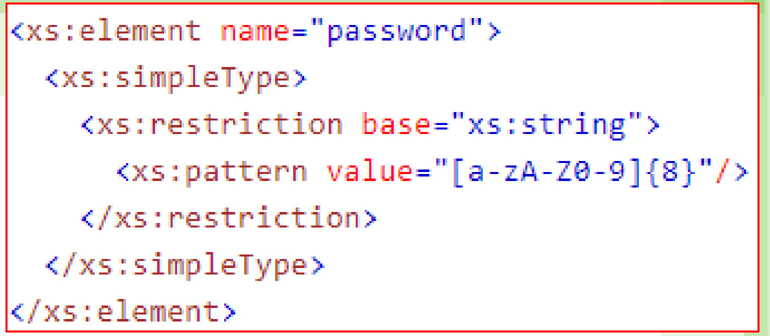
- an element called "letter" with a restriction. The acceptable value is one or more pairs of letters, each pair consisting of a lower case letter followed by an upper case letter. For example, "sToP" will be validated by this pattern, but not "Stop" or "STOP" or "stop":

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z][A-Z])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- defines an element called "gender" with a restriction. The only acceptable value is male OR female

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML XSD

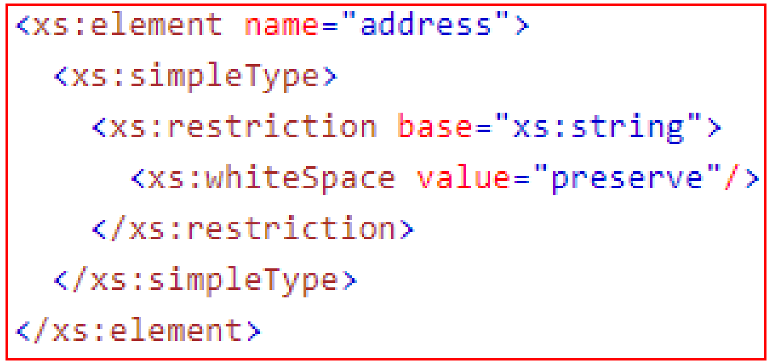


```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

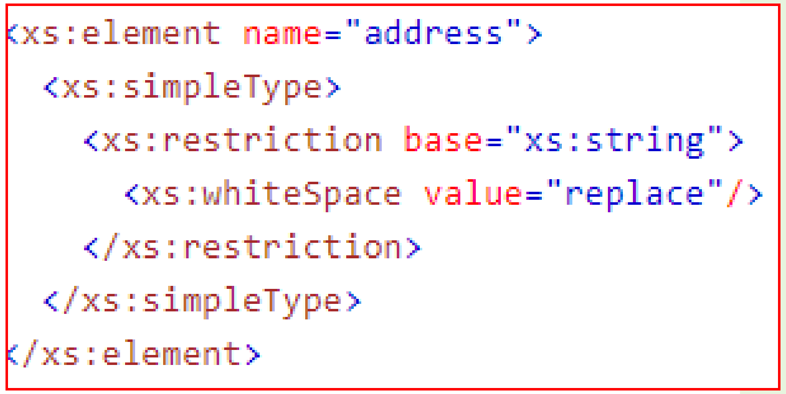
Restrictions on a Series of Values

- defines an element called "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9.

Restrictions on Whitespace Characters



```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

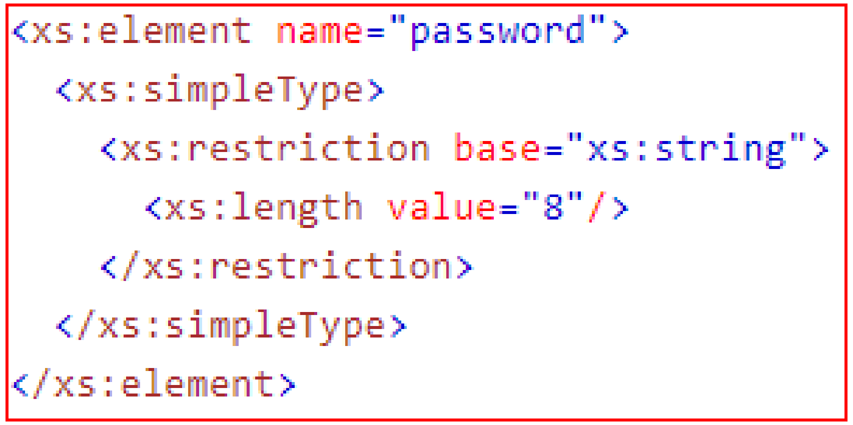


```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="replace"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

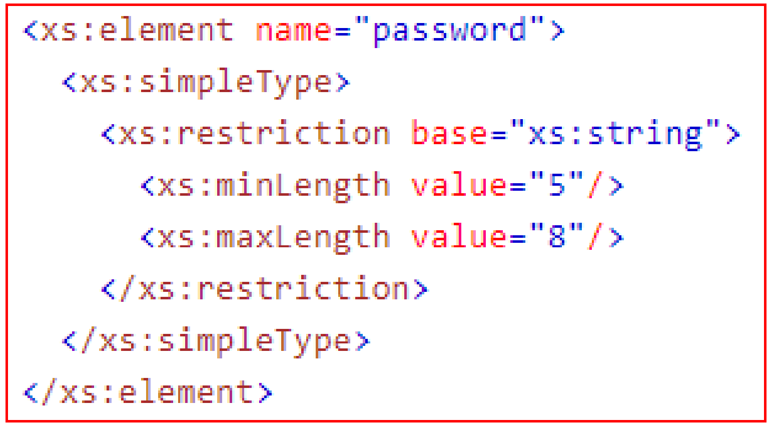
- The white Space constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters
- The white Space constraint is set to “replace", which means that the XML processor WILL REMOVE all white space characters

XML XSD

Restrictions on Length



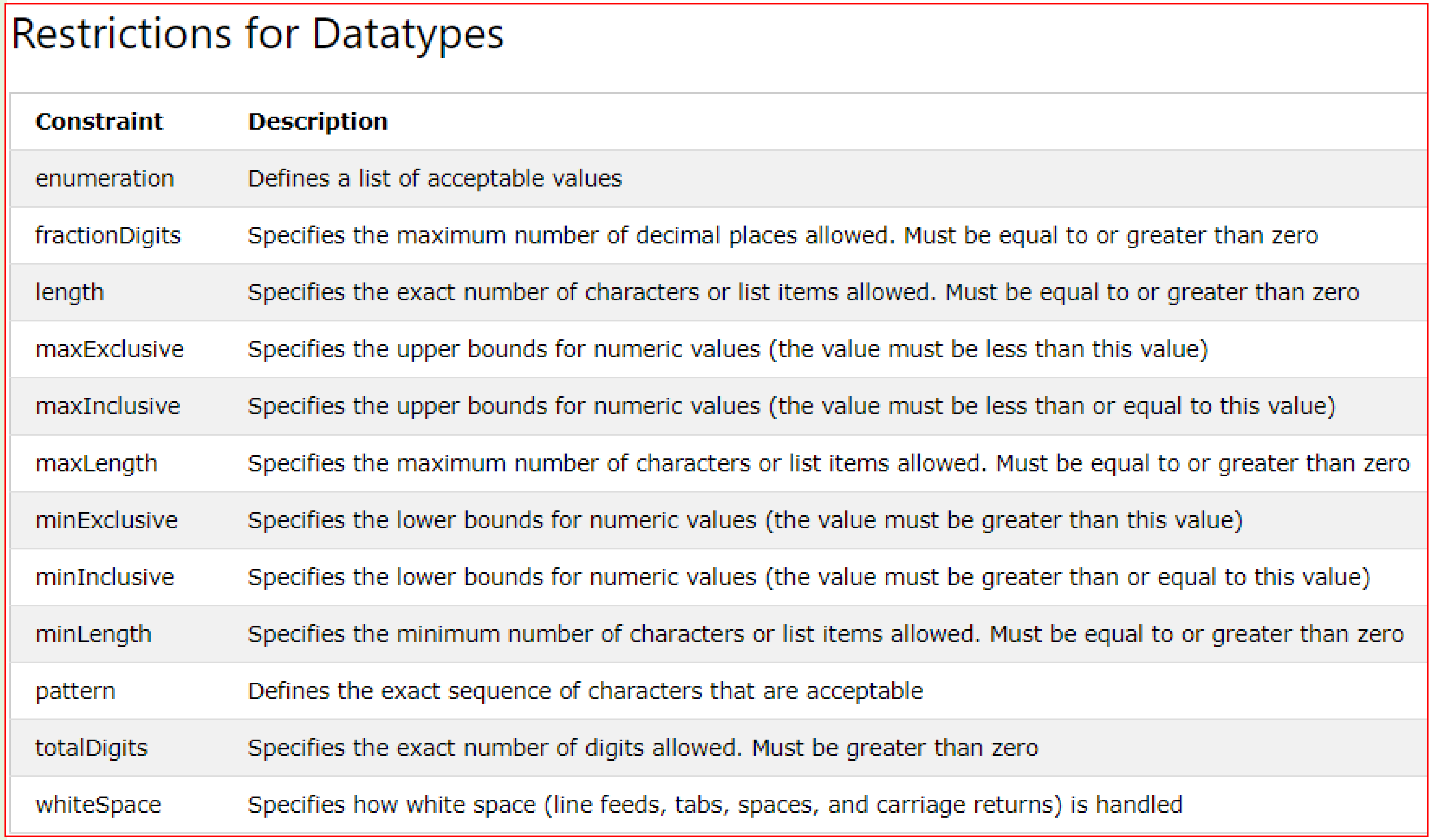
```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- defines an element called "password" with a restriction. The value must be exactly eight characters.
- defines another element called "password" with a restriction. The value must be minimum five characters and maximum eight characters.

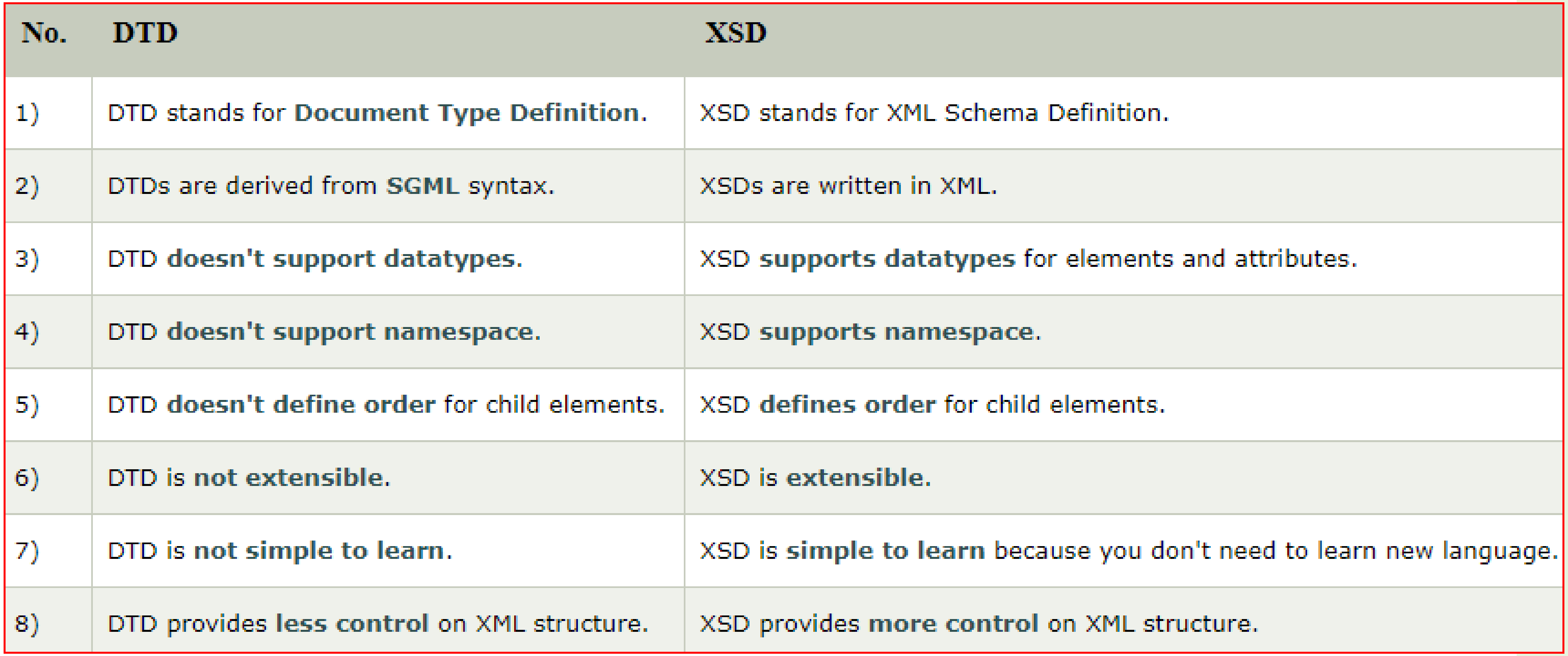
XML XSD



Restrictions for Datatypes

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XML DTD



No.	DTD	XSD
1)	DTD stands for **Document Type Definition.**	XSD stands for XML Schema Definition.
2)	DTDs are derived from **SGML** syntax.	XSDs are written in XML.
3)	DTD **doesn't support datatypes.**	XSD **supports datatypes** for elements and attributes.
4)	DTD **doesn't support namespace.**	XSD **supports namespace.**
5)	DTD **doesn't define order** for child elements.	XSD **defines order** for child elements.
6)	DTD is **not extensible.**	XSD is **extensible.**
7)	DTD is **not simple to learn.**	XSD is **simple to learn** because you don't need to learn new language.
8)	DTD provides **less control** on XML structure.	XSD provides **more control** on XML structure.

XSL Languages

- XSL stands for eXtensible Stylesheet Language. The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML based Style sheet Language.
- CSS = HTML Style Sheets
- HTML uses predefined tags and the meanings of the tags are well understood. The <table> element in HTML defines a table and a browser knows how to display it. Adding styles to HTML elements is simple. Telling a browser to display an element in a special font or color is easy with CSS.
- XSL = XML Style Sheets
- XML does not use predefined tags (we can use any tag--names we like), and the meaning of these tags are not well understood. A <table> element could mean an HTML table, a piece of furniture, or something else and a browser does not know how to display it.
- XSL describes how the XML document should be displayed! XSL is More Than a Style Sheet Language

XSL Languages

XSL consists of three parts:

- XSLT -- a language for transforming XML documents
- XPath -- a language for navigating in XML documents
- XSL-FO -- a language for formatting XML documents

XSLT -- the language for transforming XML documents.

- XSLT is a language for transforming XML documents into XHTML documents or to other XML documents.
- XPath is a language for navigating in XML documents.
- What is XSLT?
- XSLT stands for XSL Transformations
  - XSLT is the most important part of XSL
  - XSLT transforms an XML document into another XML document
  - XSLT uses XPath to navigate in XML documents
  - XSLT is a W3C Recommendation
  - XSLT = XSL Transformations

XSL Languages

- XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X) HTML element.
- With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.
- A common way to describe the transformation process is to say that XSLT transforms an XML source--tree into an XML result--tree.

XSLT Uses XPath

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents

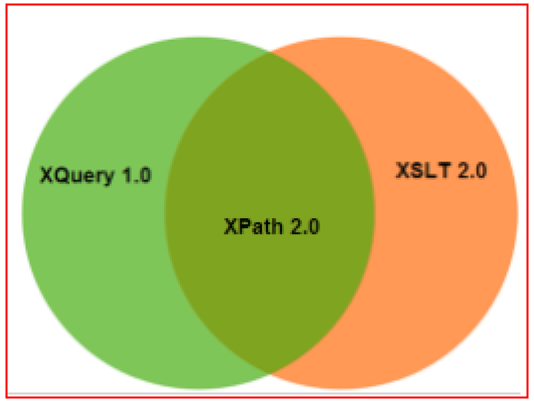
How does it works

In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

- XSLT is a W3C Recommendation
- XSLT became a W3C Recommendation 16. November 1999.

xpath

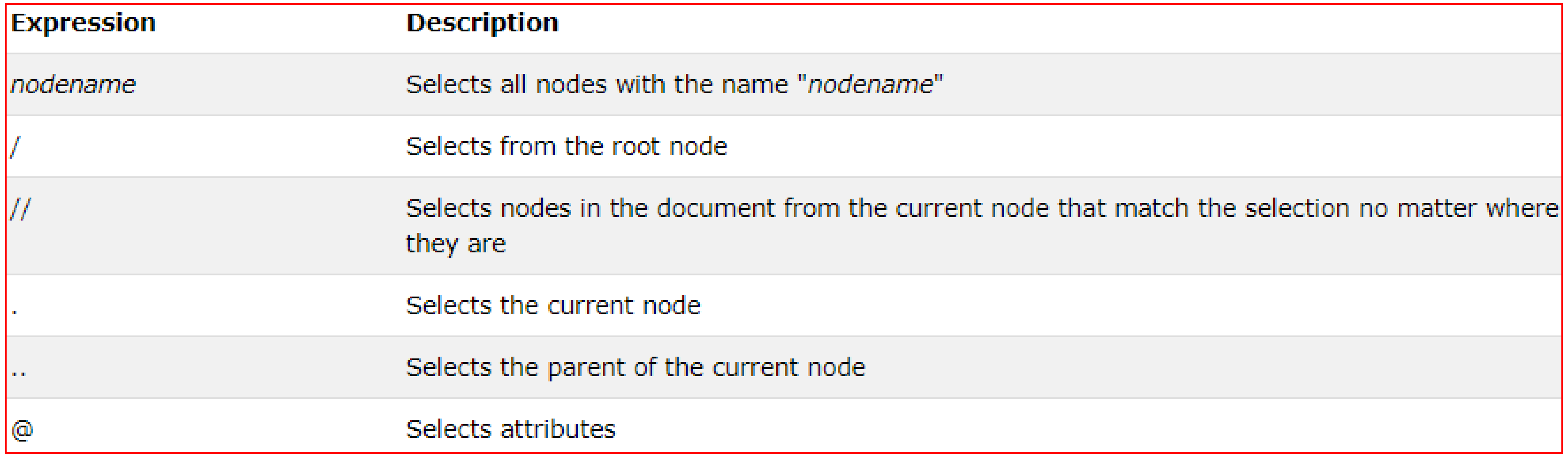
- XPath is a query language that is used for traversing through an XML document. It is used commonly to search particular elements or attributes with matching patterns.
  - XPath stands for XML Path Language
  - XPath uses "path like" syntax to identify and navigate nodes in an XML document
  - XPath contains over 200 built-in functions
  - XPath is a major element in the XSLT standard
  - XPath is a W3C recommendation.



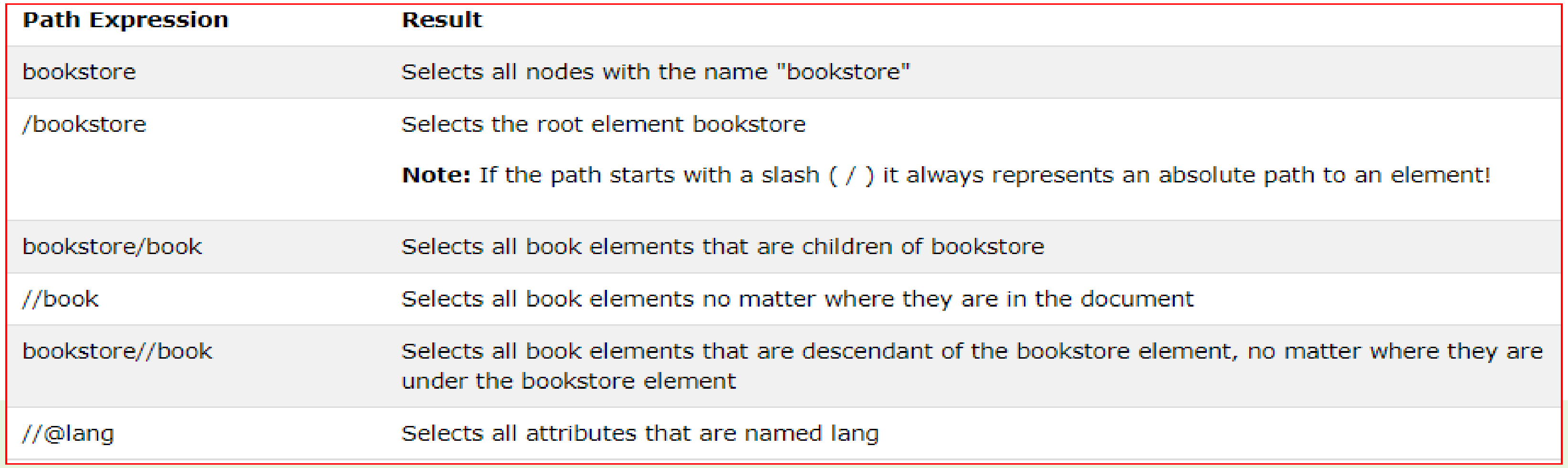
  - XPath defines a pattern or path expression to select nodes or node sets in an XML document. These patterns are used by XSLT to perform transformations.
  - The path expressions look like very similar to the general expressions we used in traditional file system.

xpath

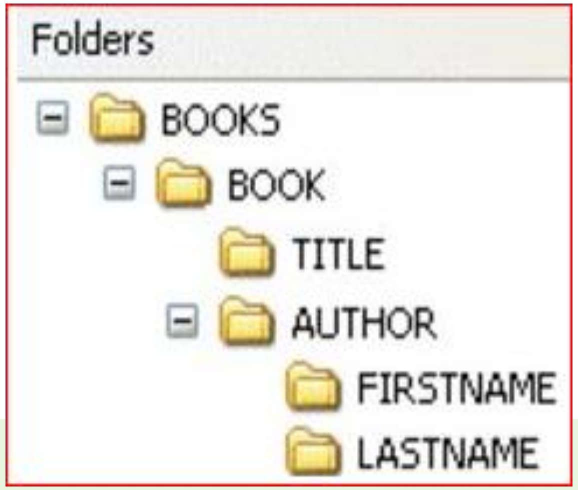
- XPath expression generally defines a pattern in order to select a set of nodes. These patterns are used by XSLT to perform transformations.



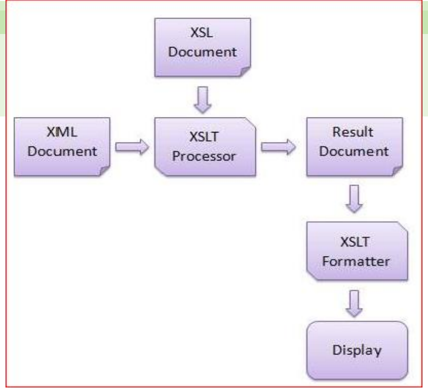
Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes



Path Expression	Result
bookstore	Selects all nodes with the name "bookstore"
/bookstore	Selects the root element bookstore **Note:** If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang



XSL Languages



XSLT Browsers

Nearly all major browsers have support for XML and XSLT. Like Mozilla Firefox, Netscape, Opera, Internet Explorer.

Advantage of XSLT

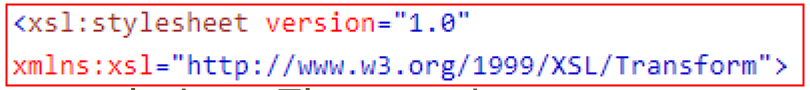
- XSLT provides an easy way to merge XML data into presentation because it applies user defined transformations to an XML document and the output can be HTML, XML, or any other structured document.
- XSLT provides Xpath to locate elements/attribute within an XML document. So it is more convenient way to traverse an XML document rather than a traditional way, by using scripting language.
- XSLT is template based. So it is more resilient to changes in documents than low level DOM and SAX.
- By using XML and XSLT, the application UI script will look clean and will be easier to maintain.
- XSLT templates are based on XPath pattern which is very powerful in terms of performance to process the XML document.
- XSLT can be used as a validation language as it uses tree-pattern-matching approach.
- You an change the output simply modifying the transformations in XSL files.

XSL Languages

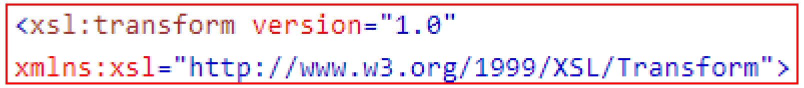
XSLT - Transformation

Correct Style Sheet Declaration

- The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.
- **Note:** <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!



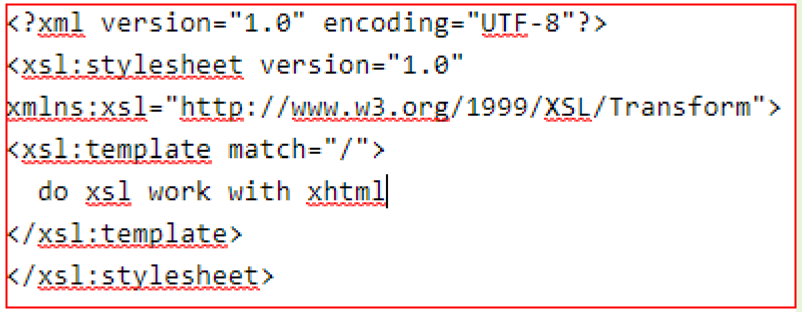
```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```



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

- The <xsl:template> Element

The **match** attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).



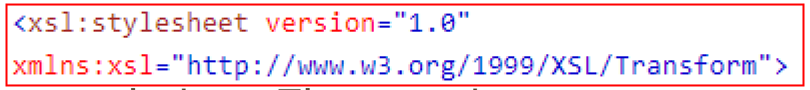
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  do xsl work with xhtml
</xsl:template>
</xsl:stylesheet>
```

XSL Languages

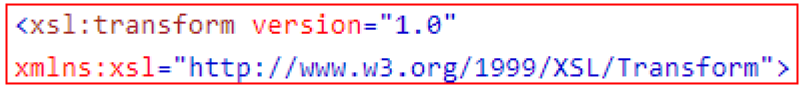
XSLT - Transformation

Correct Style Sheet Declaration

- The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.
- **Note:** <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!



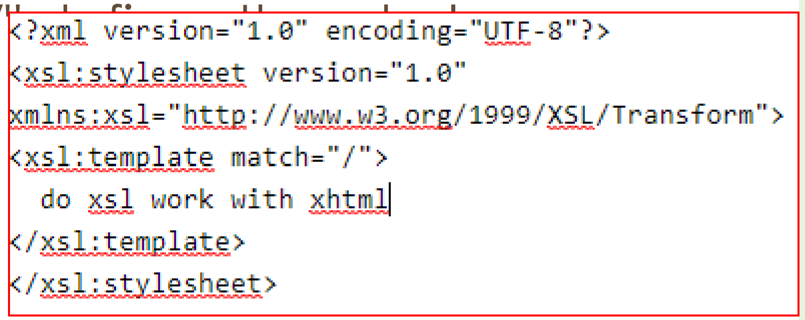
```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```



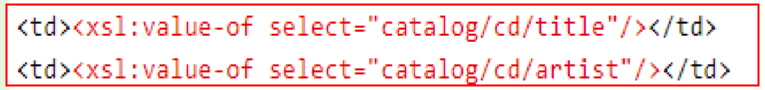
```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- The <xsl:template> Element

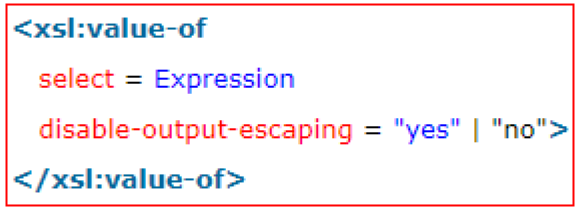
The **match** attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/ document).



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



```
<td><xsl:value-of select="catalog/cd/title"/></td>
<td><xsl:value-of select="catalog/cd/artist"/></td>
```

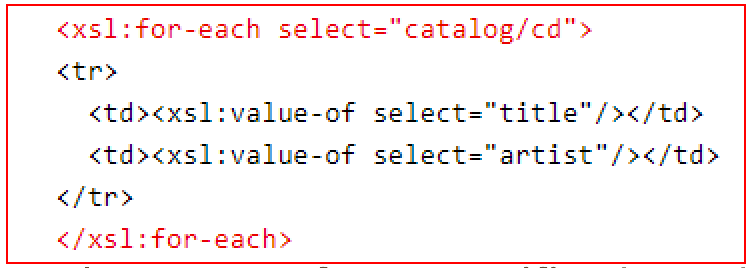


```
<xsl:value-of
  select = Expression
  disable-output-escaping = "yes" | "no">
</xsl:value-of>
```

The <xsl:value-of> Element

can be used to extract the value of an XML element and add it to the output stream of the transformation.

XSL Languages



```
<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>
```

XSLT - Transformation

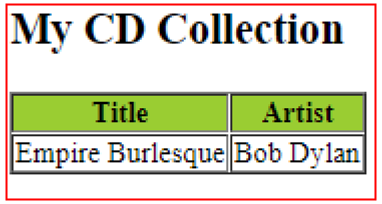
<xsl:for-each> Element

can be used to select every XML element of a specified node-set.

- The **<xsl:sort> element** is used to sort the output.

XSL Languages

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="school.xsl" type="text/xsl"?>
<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>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
</catalog>
```



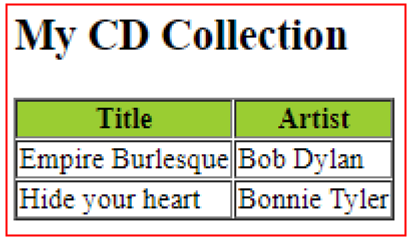
My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan

```
<?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>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td><xsl:value-of select="catalog/cd/title"/></td>
      <td><xsl:value-of select="catalog/cd/artist"/></td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

XSL Languages

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="school.xsl" type="text/xsl"?>
<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>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
</catalog>
---------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```



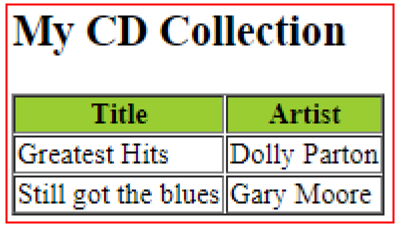
My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <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>
```

XSL Languages

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="school.xsl" type="text/xsl"?>
<catalog>
<cd>
   <title>Still got the blues</title>
   <artist>Gary Moore</artist>
   <country>UK</country>
   <company>Virgin records</company>
   <price>10.20</price>
   <year>1990</year>
  </cd>
   <cd>
   <title>Greatest Hits</title>
   <artist>Dolly Parton</artist>
   <country>USA</country>
   <company>RCA</company>
   <price>9.90</price>
   <year>1982</year>
  </cd>
</catalog>
---------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```



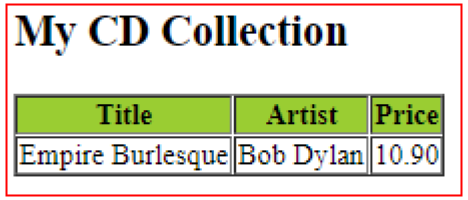
My CD Collection

Title	Artist
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <xsl:sort select="artist"/>
      <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>
```

XSL Languages

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="school.xsl" type="text/xsl"?>
<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>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
</catalog>
----------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```



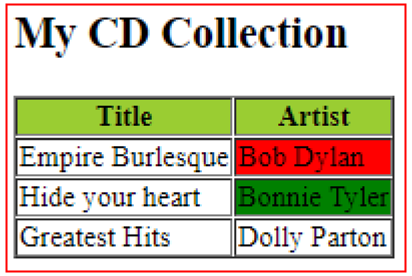
My CD Collection

Title	Artist	Price
Empire Burlesque	Bob Dylan	10.90

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
      <th>Price</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <xsl:if test="price>10">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
        <td><xsl:value-of select="price"/></td>
      </tr>
    </xsl:if>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

XSL Languages

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="school.xsl" type="text/xsl"?>
<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>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
  <cd>
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <country>USA</country>
    <company>RCA</company>
    <price>5.90</price>
    <year>1982</year>
  </cd>
</catalog>
-----------------------------------
<?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>
```



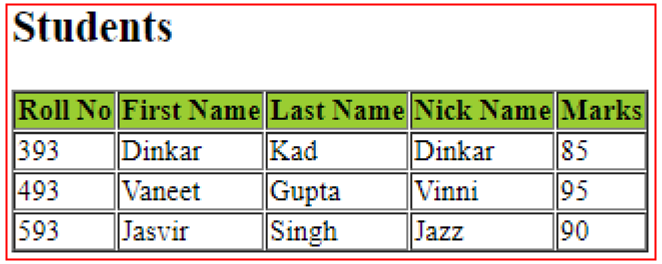
My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton

```
<body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <xsl:choose>
      <xsl:when test="price > 10">
         <td bgcolor="red">
         <xsl:value-of select="artist"/>
         </td>
      </xsl:when>
      <xsl:when test="price > 9">
         <td bgcolor="green">
         <xsl:value-of select="artist"/></td>
      </xsl:when>
      <xsl:otherwise>
         <td><xsl:value-of select="artist"/></td>
      </xsl:otherwise>
      </xsl:choose>
    </tr>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XSL Languages

```
<?xml version = "1.0"?>
<?xml-stylesheet href="school.xsl" type="text/xsl"?>
<class>
  <student rollno = "393">
    <firstname>Dinkar</firstname>
    <lastname>Kad</lastname>
    <nickname>Dinkar</nickname>
    <marks>85</marks>
  </student>
  <student rollno = "493">
    <firstname>Vaneet</firstname>
    <lastname>Gupta</lastname>
    <nickname>Vinni</nickname>
    <marks>95</marks>
  </student>
  <student rollno = "593">
    <firstname>Jasvir</firstname>
    <lastname>Singh</lastname>
    <nickname>Jazz</nickname>
    <marks>90</marks>
  </student>
</class>
-----------------------------------
<?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>
       <body>
         <h2>Students</h2>
         <table border = "1">
           <tr bgcolor = "#9acd32">
             <th>Roll No</th>
             <th>First Name</th>
             <th>Last Name</th>
             <th>Nick Name</th>
             <th>Marks</th>
           </tr>
               <xsl:for-each select="class/student">
             <tr>
               <td>
                  <xsl:value-of select = "@rollno"/>
               </td>
               <td><xsl:value-of select = "firstname"/></td>
               <td><xsl:value-of select = "lastname"/></td>
               <td><xsl:value-of select = "nickname"/></td>
               <td><xsl:value-of select = "marks"/></td>
             </tr>
           </xsl:for-each>
         </table>
       </body>
     </html>
   </xsl:template>
  </xsl:stylesheet>
```



Students

Roll No	First Name	Last Name	Nick Name	Marks
393	Dinkar	Kad	Dinkar	85
493	Vaneet	Gupta	Vinni	95
593	Jasvir	Singh	Jazz	90

Xquery

XQuery is to XML what SQL is to databases. XQuery is designed to query XML data.

- XQuery is the language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is supported by all major databases
- XQuery is a W3C Recommendation

Application of xquery

- XQuery can be used to:
- Extract information to use in a Web Service
- Generate summary reports
- Transform XML data to XHTML
- Search Web documents for relevant information

some basic syntax rules:

- XQuery is case-sensitive
- XQuery elements, attributes, and variables must be valid XML names
- An XQuery string value can be in single or double quotes
- An XQuery variable is defined with a $ followed by a name, e.g. $bookstore
- XQuery comments are delimited by (: and :), e.g. (: XQuery Comment :)

Xquery

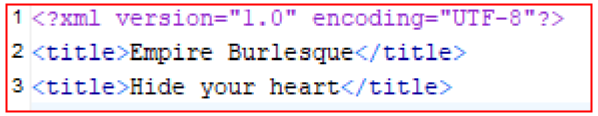
Example

The doc() function is used to open the “school.xml" file:

*doc("school.xml")/**catalog/cd/title***

doc("school.xml")/catalog/cd[price<10]

doc("school.xml")/catalog/cd[price<10]/title



```
<?xml version="1.0" encoding="UTF-8"?>
<title>Empire Burlesque</title>
<title>Hide your heart</title>
```

What is FLWOR?

FLWOR (pronounced "flower") is an acronym for "For, Let, Where, Order by, Return".

For - selects a sequence of nodes

Let - binds a sequence to a variable

Where - filters the nodes

Order by - sorts the nodes

Return - what to return (gets evaluated once for every node)

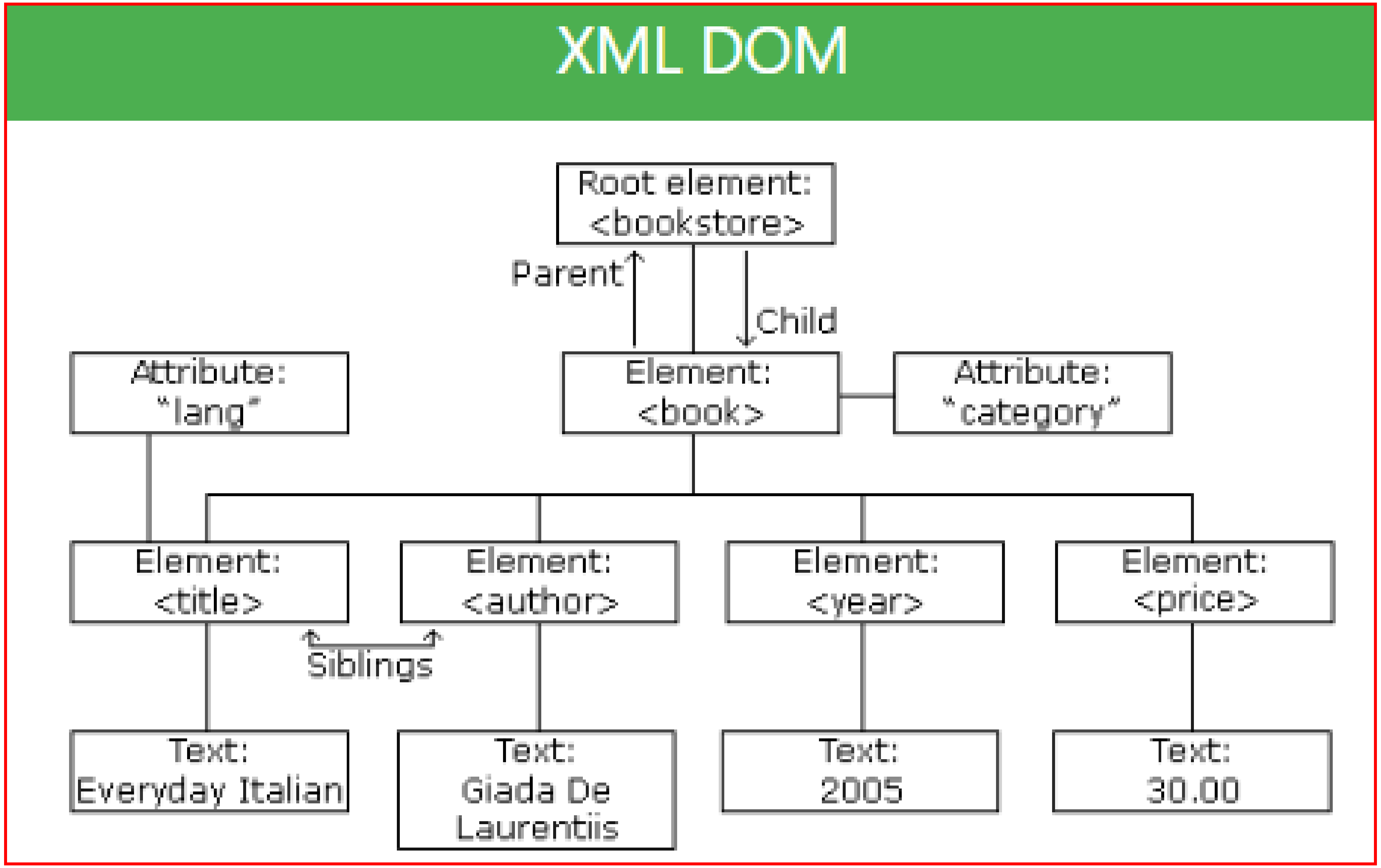
```
for $x in doc("school.xml")/catalog/cd/
where $x/price>10
return $x/title
```

Xquery

Example

```
(:for $x in doc("school.xml")/catalog/cd
where $x/price>10
order by $x/title
return $x/title:)

ul>
{
for $x in doc("school.xml")/catalog/cd/title
order by $x
return <li>{$x}</li>
}
</ul>
```

DOM


XML DOM
Root element: <bookstore>
Parent
Child
Attribute: "lang"
Element: <book>
Attribute: "category"
Element: <title>
Element: <author>
Element: <year>
Element: <price>
Siblings
Text: Everyday Italian
Text: Giada De Laurentiis
Text: 2005
Text: 30.00

DOM

- The DOM defines a standard for accessing and manipulating documents:
- *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
- The HTML DOM defines a standard way for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure.
- The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.
- Understanding the DOM is a must for anyone working with HTML or XML.
  - The XML DOM is:
  - A standard object model for XML
  - A standard programming interface for XML
  - Platform- and language-independent
  - A W3C standard
  - In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

DOM Praser

```
<html>
<body>
<p id="demo"></p>
<script>
var text, parser, xmlDoc;
text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

SAX Praser

- *SAX (Simple API for XML)* is an event-driven algorithm for parsing XML documents. SAX is an alternative to the Document Object Model (DOM). Where the DOM reads the whole document to operate on XML, SAX parsers read XML node by node. SAX processes documents state-independently (the handling of an element does not depend on the elements that came before). SAX parsers are read-only.
- SAX parsers are faster and require less memory. On the other hand, DOM is easier to use and there are tasks, such as sorting elements, rearranging elements or looking up elements, that are faster with DOM.

Assignment