Web Technology – CACS 205

The Client Tier-Unit3

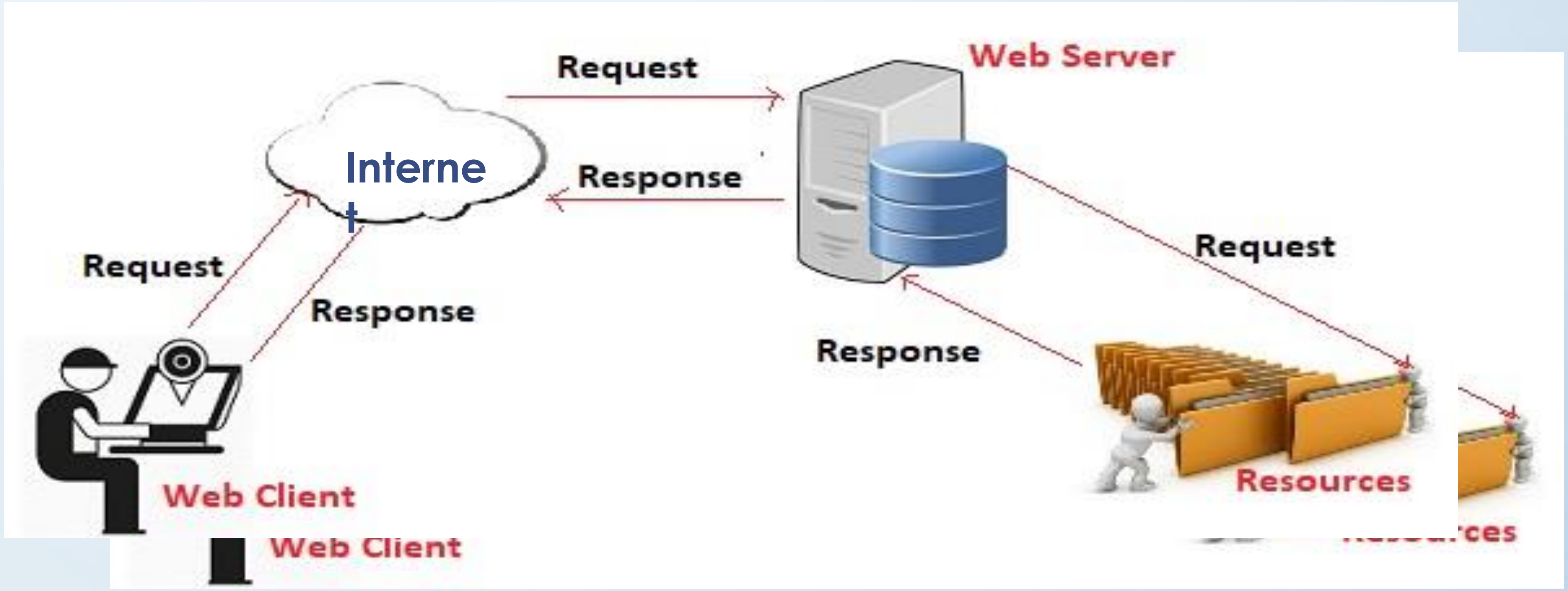
Unit 3: The Client Tier (10 hrs.)

- Representing content,
- Introduction to XML,
- Elements and Attributes,
- Rules for Writing XML,
- Namespace,
- DTD:
  - Internal Declaration,
  - Private External Declaration,
  - Public External Declaration,
  - Defining Elements and Attributes
- Schema: Simple Types and Complex Types,
- XSD attributes: Default and Fixed values,
- Facets,
- Use of Patterns,
- Order Indicators (All, Choice, Sequences),
- Occurrence Indicators
  - MaxOccurs,
  - MinOccurs
- XSL/XSLT,
- XPath,
- XQuery,
- SAX,
- DOM, Creating XML Parser.

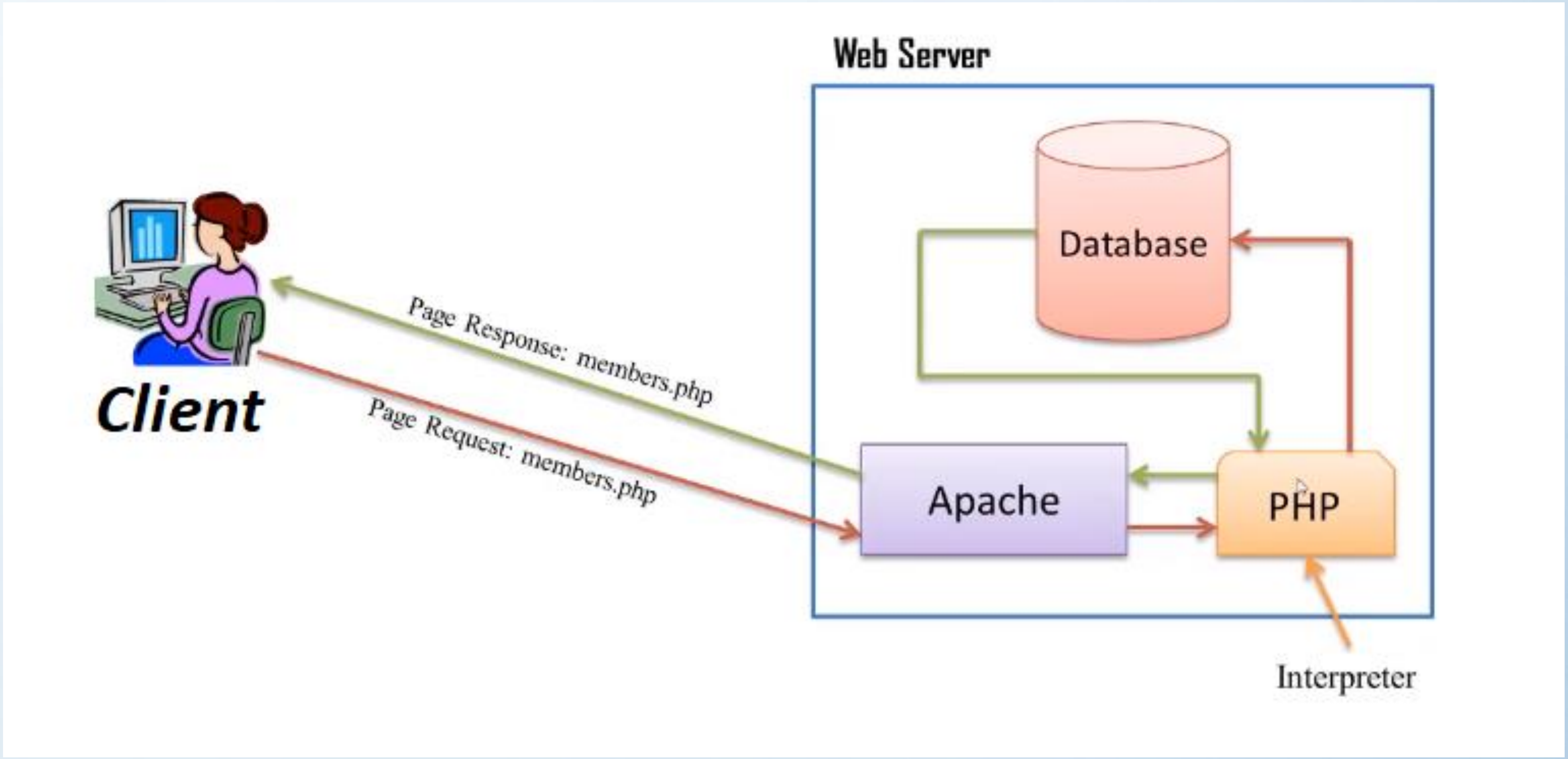
Server-Client Application Concept

The **client-server programming** model is a distributed computing architecture that segregates information users (**clients**) from information providers (**servers**).

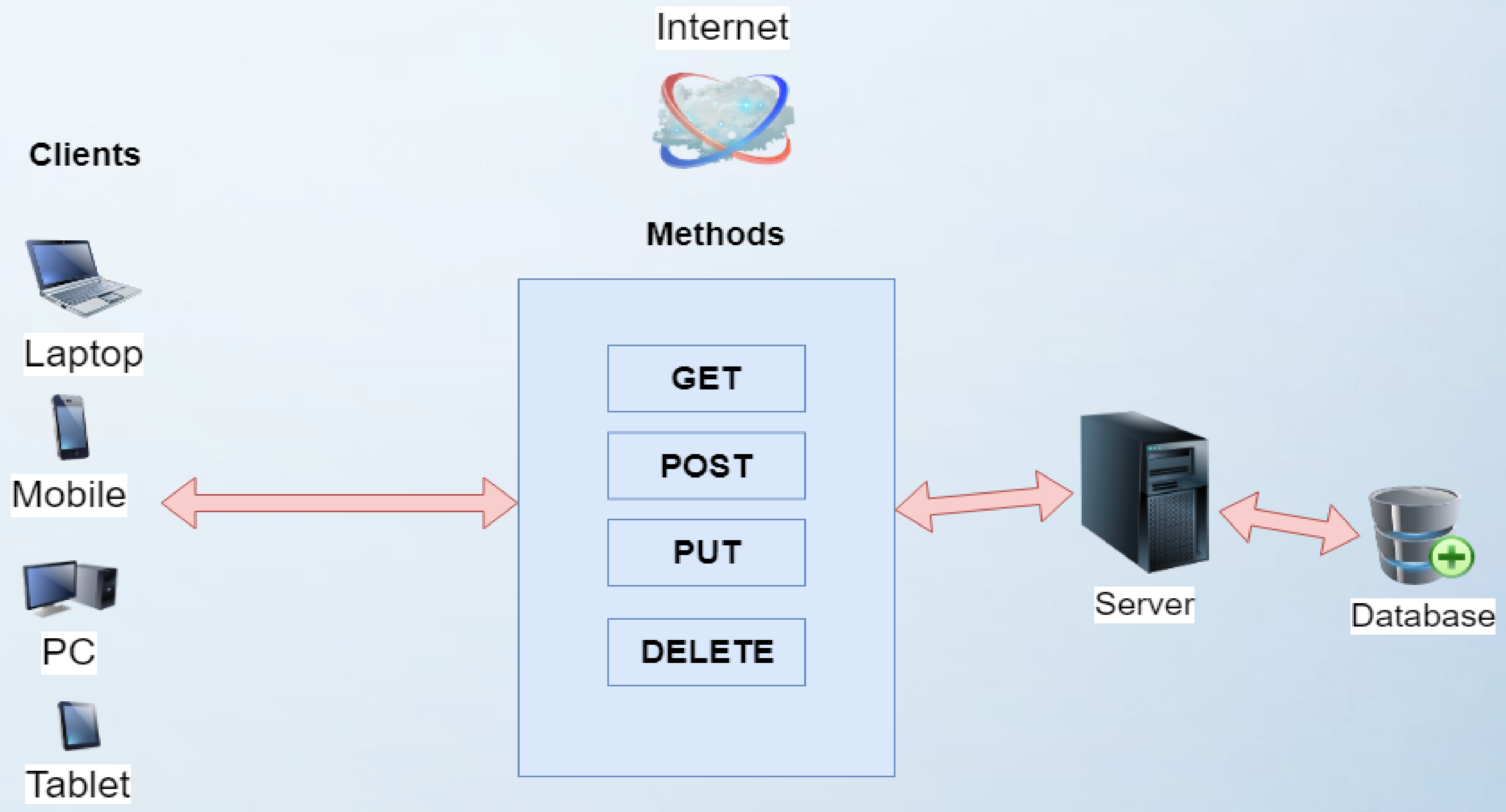
Study detail: *https://microchipdeveloper.com/tcpip:client-server-programming-model*



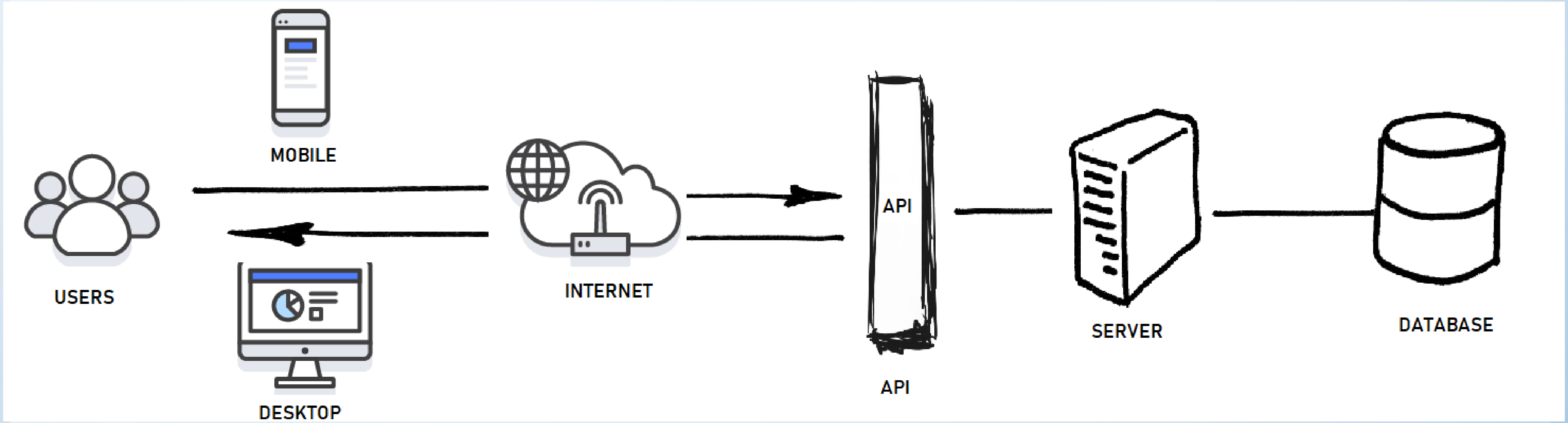
Basic Server-Client



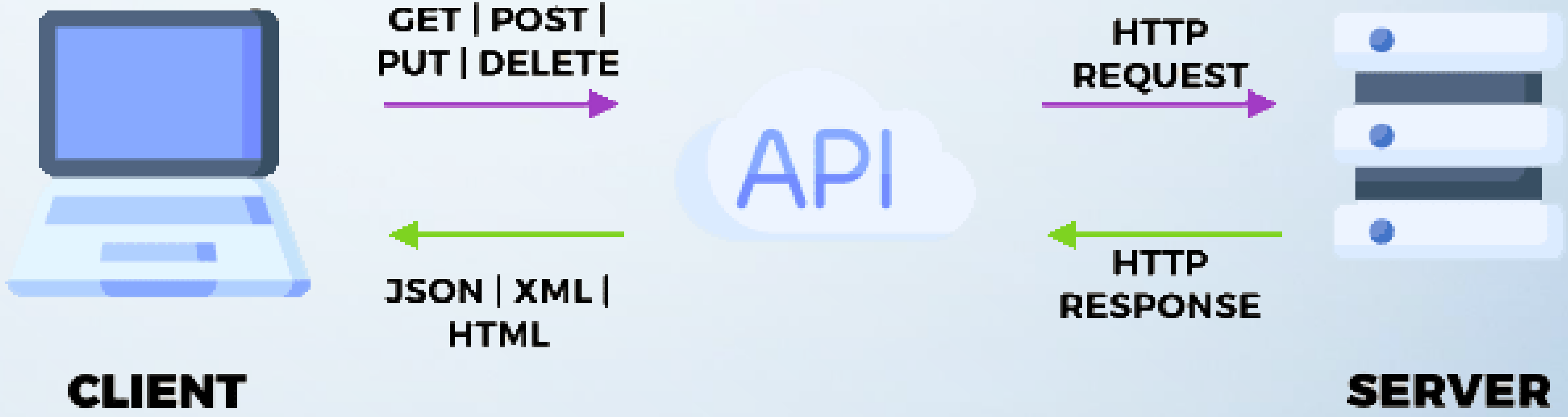
Basic Server-Client Request Methods



API based Server-Client



API based Server-Client Request Methods



Application Program Interface (API)

APIs are central to today's vast software ecosystem. There are virtually endless ways to connect diverse web applications, and APIs power these integrations behind the scenes. So, if you want to connect your app or service to the digital world, it's worth understanding how APIs work.

We can divide Web APIs into groups by intended level of access and scope of use. There are four widely agreed-upon types of web APIs: open APIs, partner APIs, internal APIs, and composite APIs.

- **Open APIs** (*public APIs* or *external APIs,* which any developer can access.)
- **Partner APIs** (which only authorized developers may access.)
- **Internal APIs** (also called private APIs, which only internal teams may access.)
- **Composite APIs** (Combine multiple APIs, allowing developers to bundle calls or requests and receive one unified response from different servers.)

API Architectures

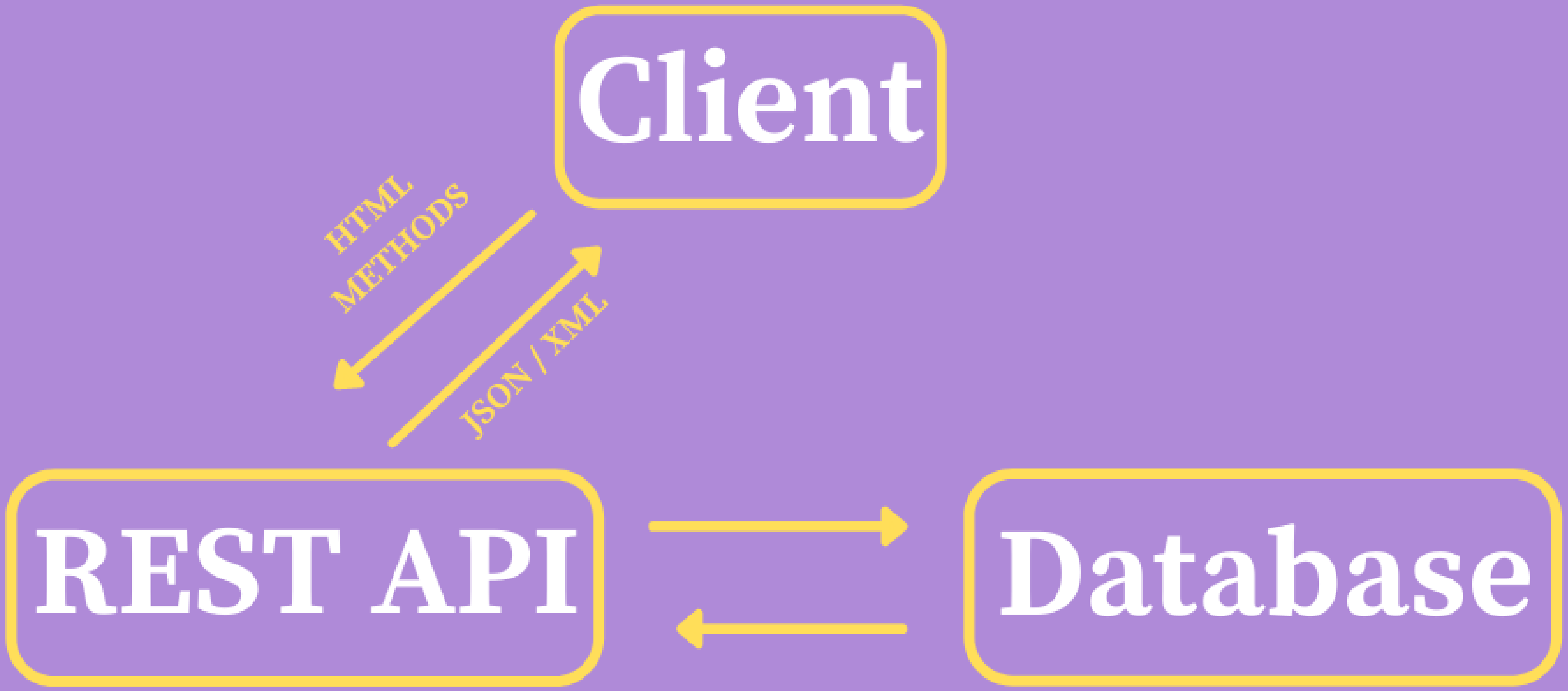
We can also understand APIs in terms of their architecture. An API's architecture consists of the rules that guide what information an API can share with clients and how it shares the data. REST, SOAP, and RPC are the most popular API architectures in use today

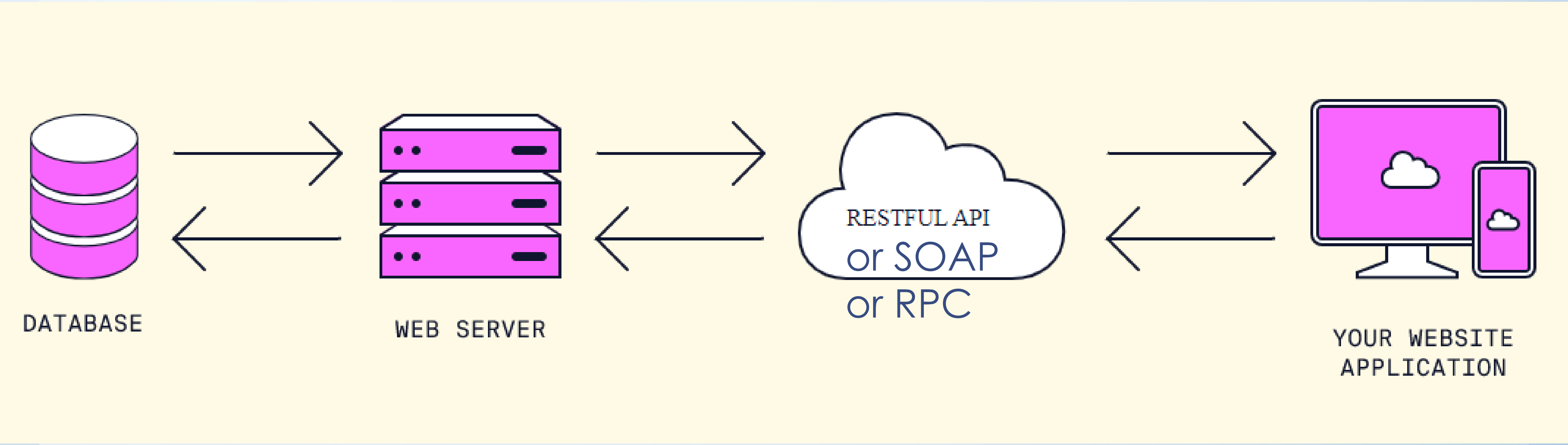
- **REST** (RESTFul), REST stands for representational state transfer and was created by computer scientist **Roy Fielding**. Today, the majority of web APIs are built on REST. a collection of guidelines for lightweight, scalable web APIs. Server responses are formatted in JavaScript Object Notation (JSON).

- **SOAP**, a stricter protocol for more secure APIs. SOAP is standardized by the World Wide Web Consortium (W3C) and utilizes XML to encode information.

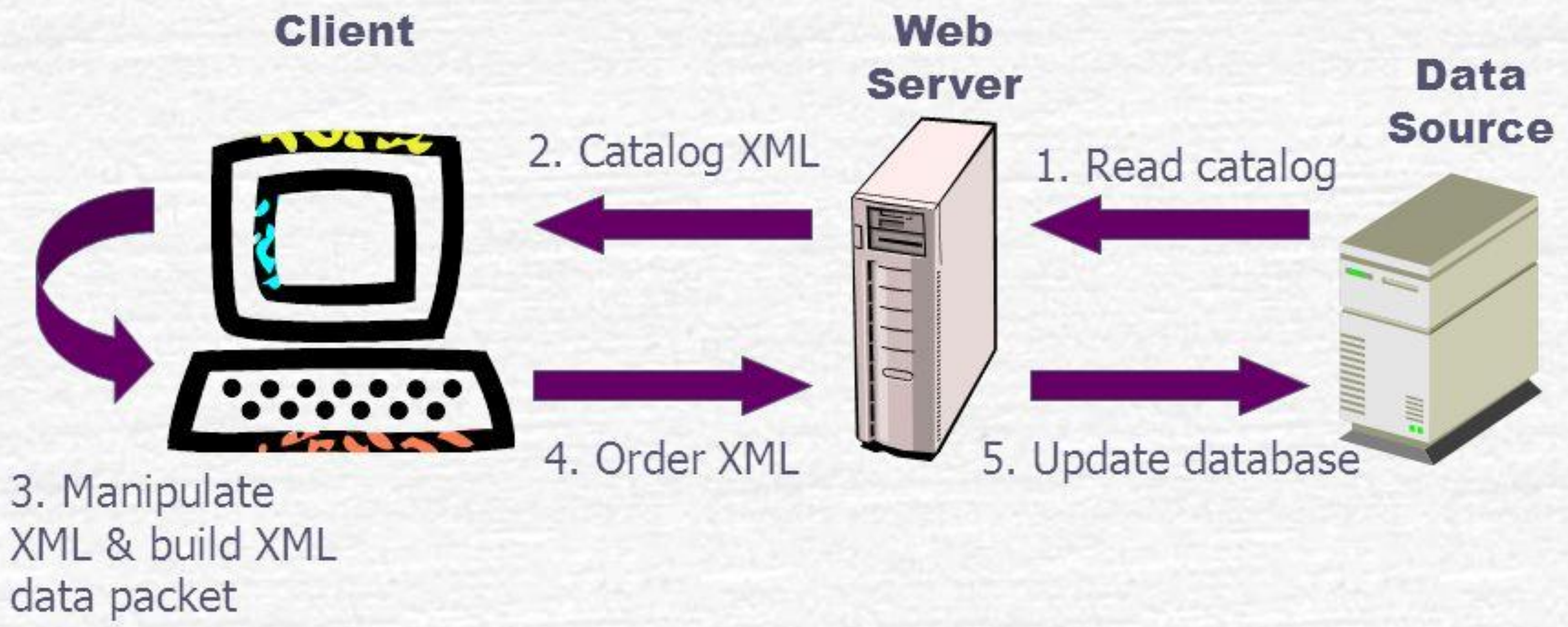
- **RPC**, RPC APIs may employ either JSON (a JSON-RPC protocol) or XML (an XML-RPC protocol) in their calls. XML is more secure and more accommodating than JSON, but these two protocols are otherwise similar.

Study more: https://blog.hubspot.com/website/types-of-apis


Client
HTML METHODS
JSON / XML
REST API
Database


DATABASE
WEB SERVER
RESTFUL API
or SOAP
or RPC
YOUR WEBSITE APPLICATION

Client Server XML Example



API Architectures

We can also understand APIs in terms of their architecture. An API's architecture consists of the rules that guide what information an API can share with clients and how it shares the data. REST, SOAP, and RPC are the most popular API architectures in use today

- **REST** (RESTFul), REST stands for representational state transfer and was created by computer scientist **Roy Fielding**. Today, the majority of web APIs are built on REST. a collection of guidelines for lightweight, scalable web APIs. Server responses are formatted in JavaScript Object Notation (JSON).

- **SOAP**, a stricter protocol for more secure APIs. SOAP is standardized by the World Wide Web Consortium (W3C) and utilizes XML to encode information.

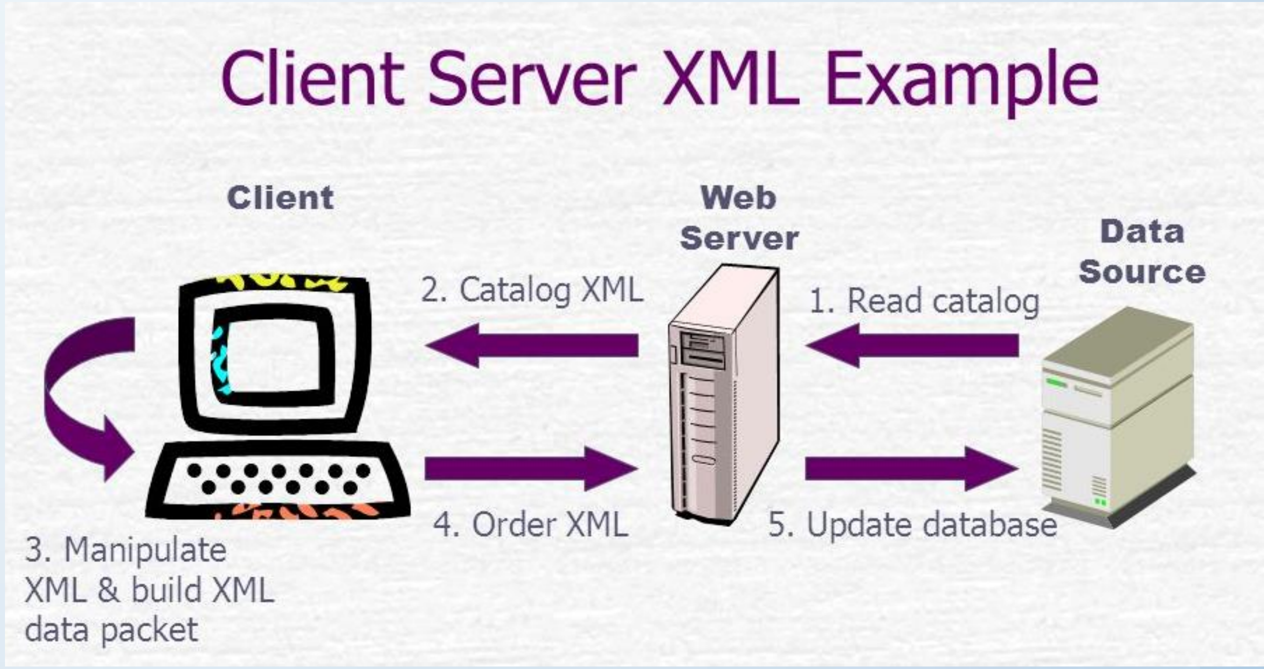
- **RPC**, RPC APIs may employ either JSON (a JSON-RPC protocol) or XML (an XML-RPC protocol) in their calls. XML is more secure and more accommodating than JSON, but these two protocols are otherwise similar.

Study more: https://blog.hubspot.com/website/types-of-apis

Introduction of XML

XML - eXtensible Markup Language.

- Used to store and transport data
- Self description
- Used to carry data (Not used to display data)
- Self-defined tags
- Platform and language independent
- Helps in easy communication between two different platforms. Like between **Java App Server Data** and **PHP App Server Data** sharing.



Introduction of XML

- XML stands for eXtensible Markup Language based on SGML (Standard Generalized Markup Language).
- It is defined in the XML 1.0 specification, which is developed by the W3C (World Wide Web Consortium) in 1996
- While XML is a markup language that is used to **transfer** data and text between **driver hardware**, **operating system**s and **applications**, SGML is an ISO standard for specifying a document markup language or a set of tags.
- XML is mostly self-descriptive like it has the information of the sender, has the information from the receiver, has a heading and also contains a body with message but still, nothing is done by XML, it is just information covered in tags, it does not contain any predefined tags and it is expected to work in the same way even when there is change of data that is even if new data is added or existing data is removed.

Introduction of XML

XML (Extensible Markup Language) is a text-based format used to share data on the World Wide Web, intranets and more.

Posting XML with Correct Content-Type Header. To post XML to the server, you need to make an HTTP POST request, include the XML data in the body of the POST request message, and set the correct MIME type for the XML using the "Content-Type: application/xml".

XML's primary function is to provide a "simple text-based format for representing structured information," according to the World Wide Web Consortium (W3C), the standards body for the web.

XML enables sharing of structured information among and between the following:

- programs and programs;
- programs and people; and
- locally and across networks.

XML vs HTML

XML	HTML
The full form is eXtensible Markup Language	The full form is Hypertext Markup Language
The main purpose is to focus on the transport of data and saving the data	Focusses on the appearance of data. Enhances the appearance of text
XML is dynamic because it is used in the transport of data	HTML is static because its main function is in the display of data
It is case-sensitive. The upper and lower case needs to be kept in mind while coding	It is not case-sensitive. Upper and lower case are of not much importance in HTML
You can define tags as per your requirement, but closing tags are mandatory	It has its own pre-defined tags, and it is not necessary to have closing tags
XML can preserve white spaces	White spaces are not preserved in HTML
eXtensible Markup Language is content-driven, and not many formatting features are available	Hypertext Markup Language, on the other hand, is presentation driven. How the text appears is of utmost importance
Any error in the code shall not give the final outcome	Small errors in the coding can be ignored and the outcome can be achieved
The size of the document may be large	No lengthy documents. Only the syntax needs to be added for best-formatted output

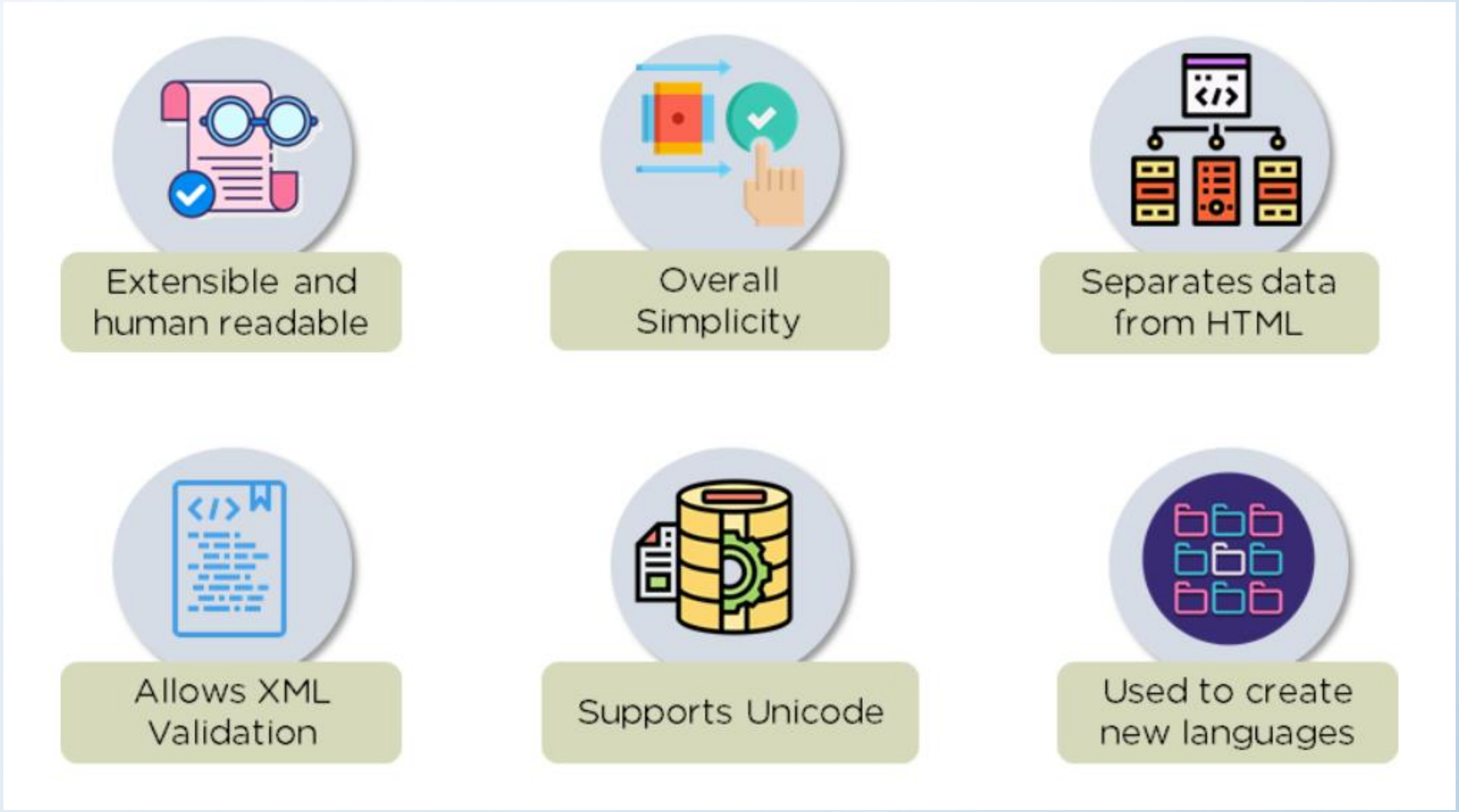
XML is not replacement of HTML

- HTML is an excellent tool for displaying hypermedia documents across a network and XML is designed for electronic information provides who want to do things that HTML is not designed for.
- Both are the markup languages, HTML is Hyper Text mark Up language where XML is Extensible Mark Up languages.
- You can see both language as, where HTML specifies how to displaying data in browser. On Where XML defines content.
- Another thing that is need to consider, that both has tags but use of these are different ,In HTML we use Tags to tell the browser to displaying data and in XML we use tags to describe data.

XML is not replacement of HTML

- Basically XML provides a structural representation of data and HTML is a set of special codes that can be embedded in text to add formatting and linking information.
- **So both are not same.**
  It is important to understand why XML was created. XML was created so that richly structured documents could be used over the web (HTML is not best use for this purpose).
- So, XML itself does not replace HTML. Instead, it provides an alternative which allows you to define your own set of markup elements. HTML is expected to remain in common use on the web, and the current versions of HTML (XHTML and HTML5) are in XML syntax.

Feature of XML



XML Syntax

- The following is the basic XML syntax -

```
<?xml version = "1.0" encoding = "UTF-8" ?
<root>
   <child>
      <subchild>.....</subchild>
   </child>
</root>
```

XML Declaration

- The XML Declaration provides basic information about the format for the rest of the XML document.
- It takes the form of a Processing Instruction and can have the attributes **version**, **encoding** and **standalone**.

The following code shows the syntax for XML declaration :

```
<?xml version="version_number," encoding="character_encoding"
standalone="yes_or_no" ?>
```

Example

```
<?xml version="1.0" encoding="utf-8"?>
```

XML Declaration Examples

- XML declaration with no parameter:

```
<?xml>
```

- XML declaration with version definition:

```
<?xml version=”1”>
```

- XML declaration with all parameters defined:

```
<?xml version=”1” encoding=”UTF-8” standalone=”no” ?>
```

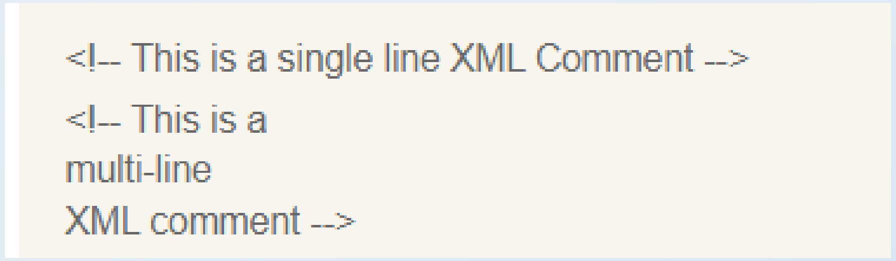
- XML declaration with all parameters defined in single quotes:

```
<?xml version=’1’ encoding=’iso-8859-1’ standalone=’no’ ?>
```

XML Comments

- **XML comments** are similar to HTML comments. The comments are added as notes for understanding the XML code.
- XML comments briefs about the code and help to edit the source code at a later time.
- XML comments are visible only in the source code and not in the XML code.
- Comments are optional. Adding comments to a document can help you comprehend it better.

<?xml version="1.0" encoding="utf-8"?> **<!-- This is XML Declaration-->**



```
<!-- This is a single line XML Comment -->
<!-- This is a
multi-line
XML comment -->
```

XML Tags and Elements

- Except for declarations, **tags** work in **pairs**. An opening tag and a closing tag make up each tag pair
- <> is used to enclose tag names. The start and end tags for a tag pair must be identical, except that the end tag must have / after the <.

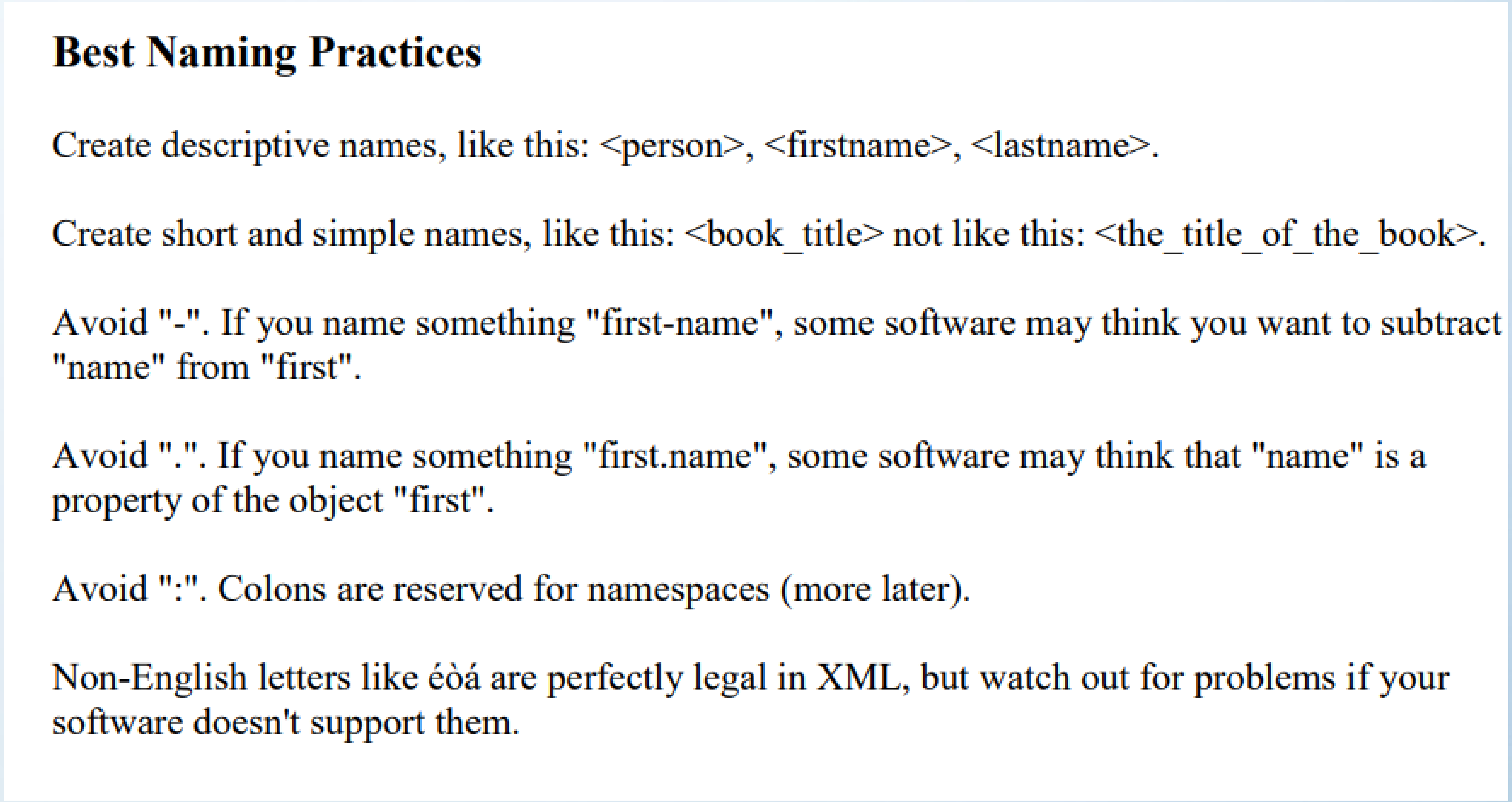
```
<employee>
    <firstname>John</firstname>
    <lastname>Doe</lastname>
    <title>Engineer</title>
    <division>Materials</division>
</employee>
```

- In the above example, employee, firstname, lastname, title and division are tags. Tag names are also referred to as elements.

XML Element Naming Rules

- XML elements must follow these naming rules:
  - 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 Any name can be used, no words are reserved (except xml).

XML Naming Rules : Best Naming Practices



Best Naming Practices

Create descriptive names, like this: <person>, <firstname>, <lastname>.

Create short and simple names, like this: <book_title> not like this: <the_title_of_the_book>.

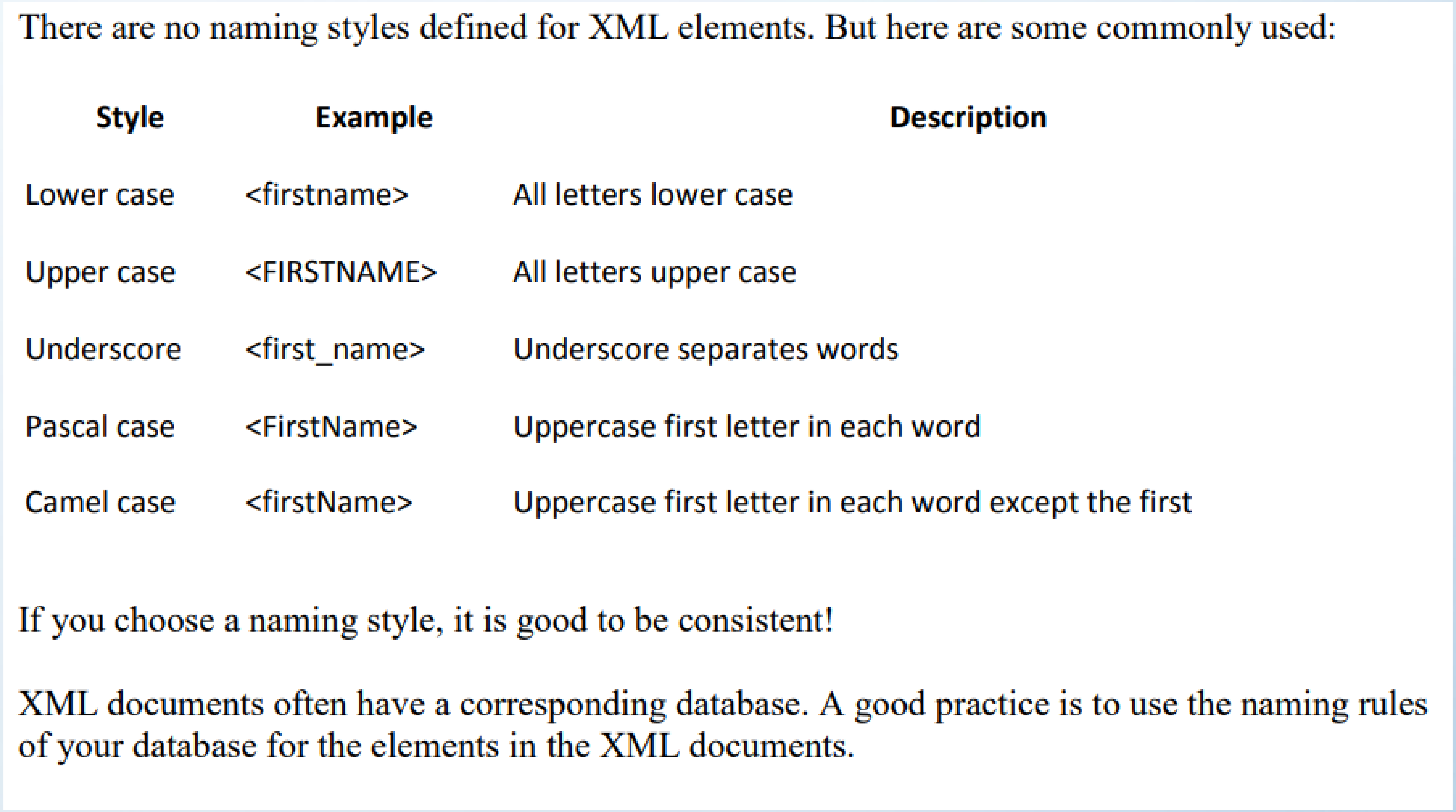
Avoid "-". If you name something "first-name", some software may think you want to subtract "name" from "first".

Avoid ".". If you name something "first.name", some software may think that "name" is a property of the object "first".

Avoid ":". Colons are reserved for namespaces (more later).

Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them.

Element Naming Styles



There are no naming styles defined for XML elements. But here are some commonly used:

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

If you choose a naming style, it is good to be consistent!

XML documents often have a corresponding database. A good practice is to use the naming rules of your database for the elements in the XML documents.

Single Item XML

```
<?xml version="1.0" encoding="UTF-8"?>
<PersonData>  <!-- root element / main container -->
    <User>
        <fname>James</fname>
        <lname>Born</lname>
        <email>james@gmail.com</email>
    </User>
</PersonData>
```

Multiple Item XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PersonData> <!-- root element / main container -->
    <User id="1">
        <fname>James</fname>
        <lname>Born</lname>
        <email>james@gmail.com</email>
    </User>
    <User id="2">
        <fname>John</fname>
        <lname>Smith</lname>
        <email>john@gmail.com</email>
    </User>
    <User id="3">
        <fname>Rodger</fname>
        <lname>Harper</lname>
        <email>rodger@gmail.com</email>
    </User>
</PersonData>
```

XML Attributes

- In the start tag, the attribute for an element is inserted after the tag name. For a single element, you can add many attributes with distinct names.

There are two main rules to define an attribute -

- Attribute values must be within quotes.
- An element cannot contain several attributes with the same name.

```
< tagname attribute-name1="attribute value 1" attribute-name2="attribute value 2"… >
        Content/Data should be inserted in this section..
</ tagname >
```

XML Attributes

- Consider the same example as earlier example

```
<employee id="be129">
        <firstname>John</firstname>
        <lastname>Doe</lastname>
        <title>Engineer</title>
        <division>Materials</division>
</employee>
```

- Here the “id” is an attribute specific to that employee.

Why Avoid XML Attributes?

- Some of the problems with using attributes are:
  - attributes cannot contain multiple values (elements can)
  - attributes cannot contain tree structures (elements can)
  - attributes are not easily expandable (for future changes) Attributes are difficult to read and maintain. Use elements for data.
  - Use attributes for information that is not relevant to the data.

  Don't end up like this:

Why Avoid XML Attributes?

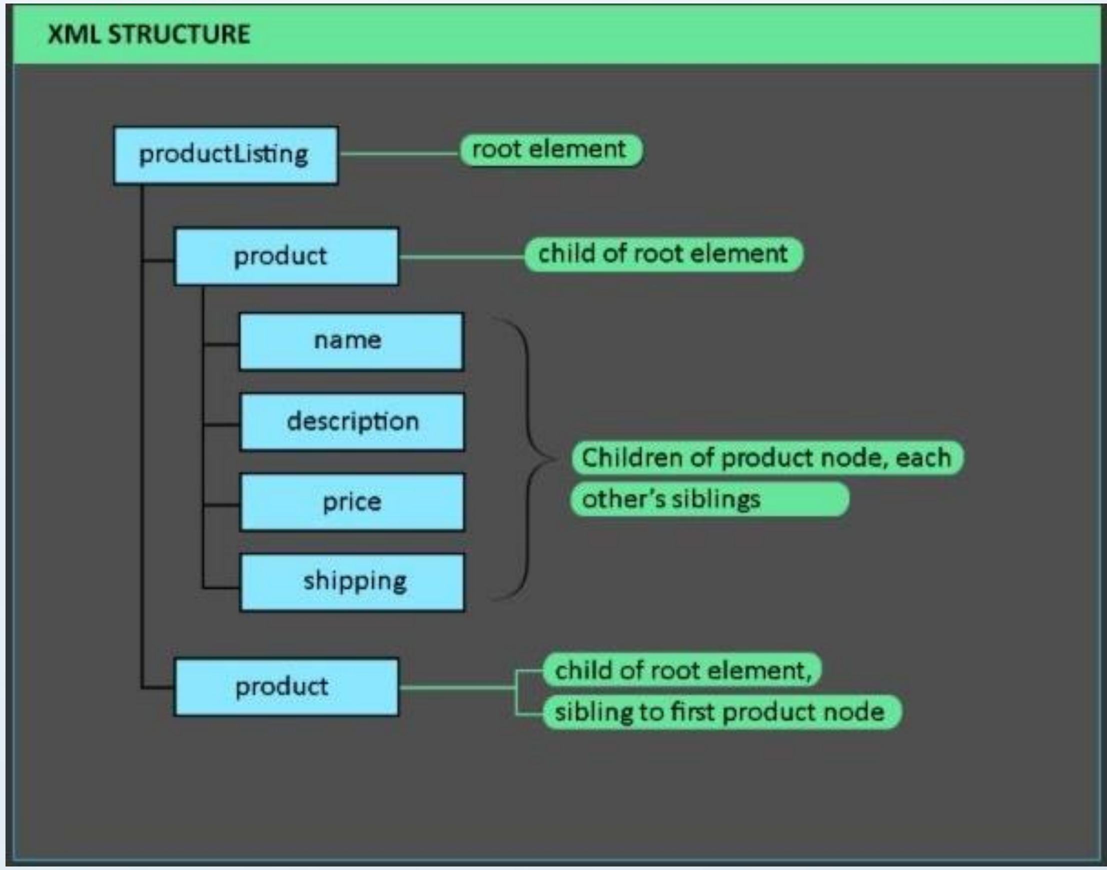
Don’t

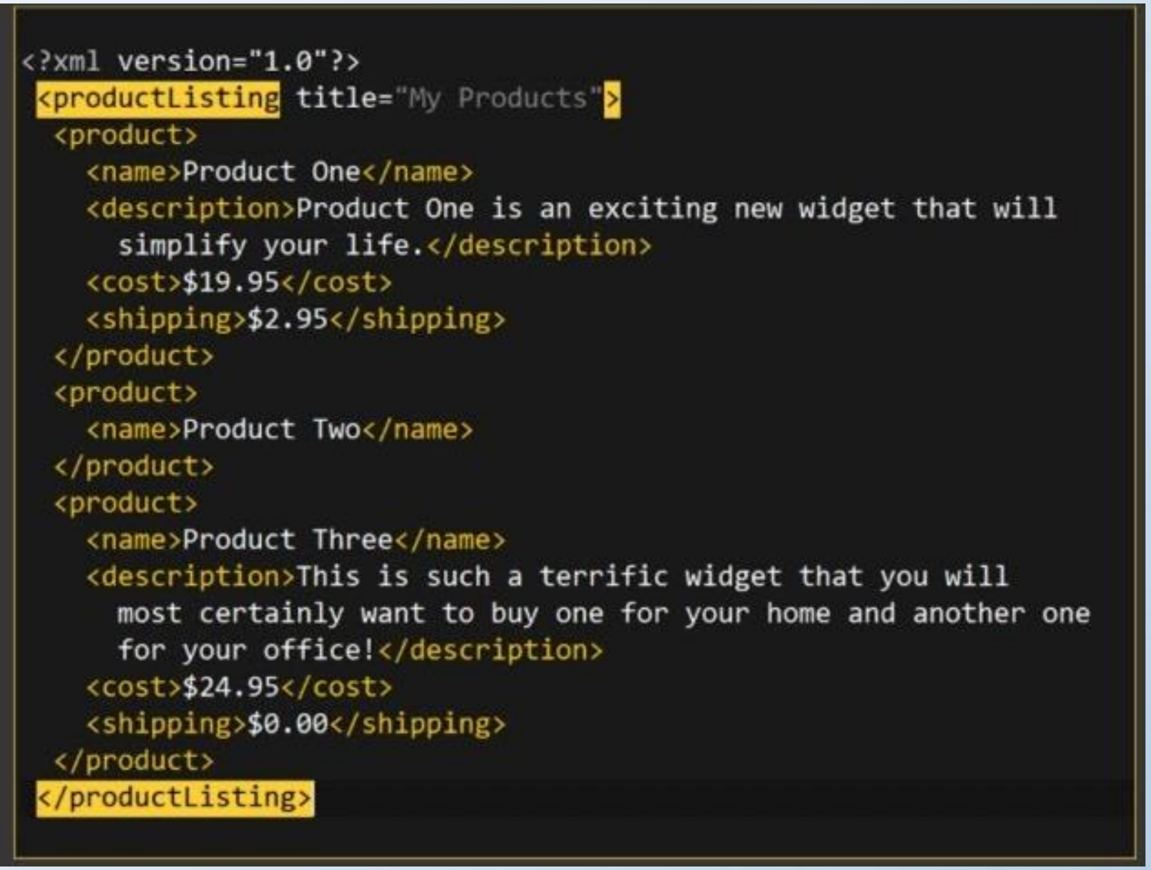
```
<note day="10" month="01"
year="2008"
to="Tove" from="Jani"
heading="Reminder"
body="Don't forget me this
weekend!">
</note>
```

DO:

```
<note>

<date>

 <year>2008</year>

 <month>01</month>

 <day>10</day>

</date>

<to>Tove</to>

<from>Jani</from>

16

<heading>Reminder</heading>

<body>Don't forget me this weekend!</body>

</note>
```

Elements and Attributes





```
<?xml version="1.0"?>
 <productListing title="My Products">
  <product>
    <name>Product One</name>
    <description>Product One is an exciting new widget that will
      simplify your life.</description>
    <cost>$19.95</cost>
    <shipping>$2.95</shipping>
  </product>
  <product>
    <name>Product Two</name>
  </product>
  <product>
    <name>Product Three</name>
    <description>This is such a terrific widget that you will
      most certainly want to buy one for your home and another one
      for your office!</description>
    <cost>$24.95</cost>
    <shipping>$0.00</shipping>
  </product>
 </productListing>
```

Advantage of XML

Here are some key advantages of utilizing XML:

- Documents can now be moved between systems and applications. We can swiftly communicate data between platforms with the help of XML.
- XML decouples data from HTML.
- The platform switching procedure is speeded up with the help of XML.
- User-defined tags / Customised tags can be created.

Disadvantage of XML

Here are some disadvantages of utilizing XML:

- The use of a processing application is necessary for XML
- The XML syntax is quite similar to other 'text-based' data transfer protocols, which might be perplexing at times.
- There is no intrinsic data type support.
- The XML syntax is redundant.

Tree of XML

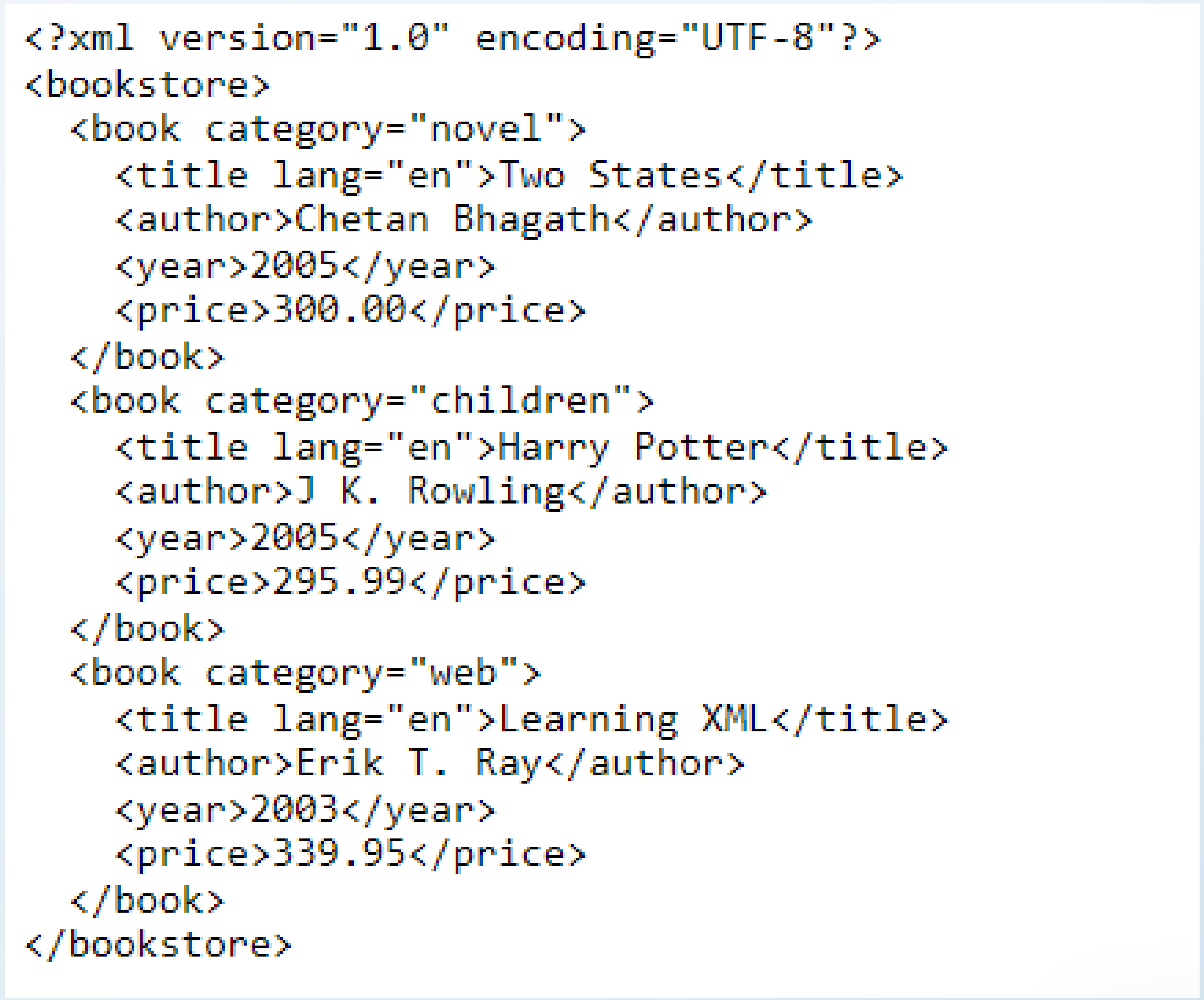
- Elements trees are used to create XML documents.
- An XML tree begins with a root element and branches to child elements.
- All elements can have child elements (sub-elements):

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

Tree of XML

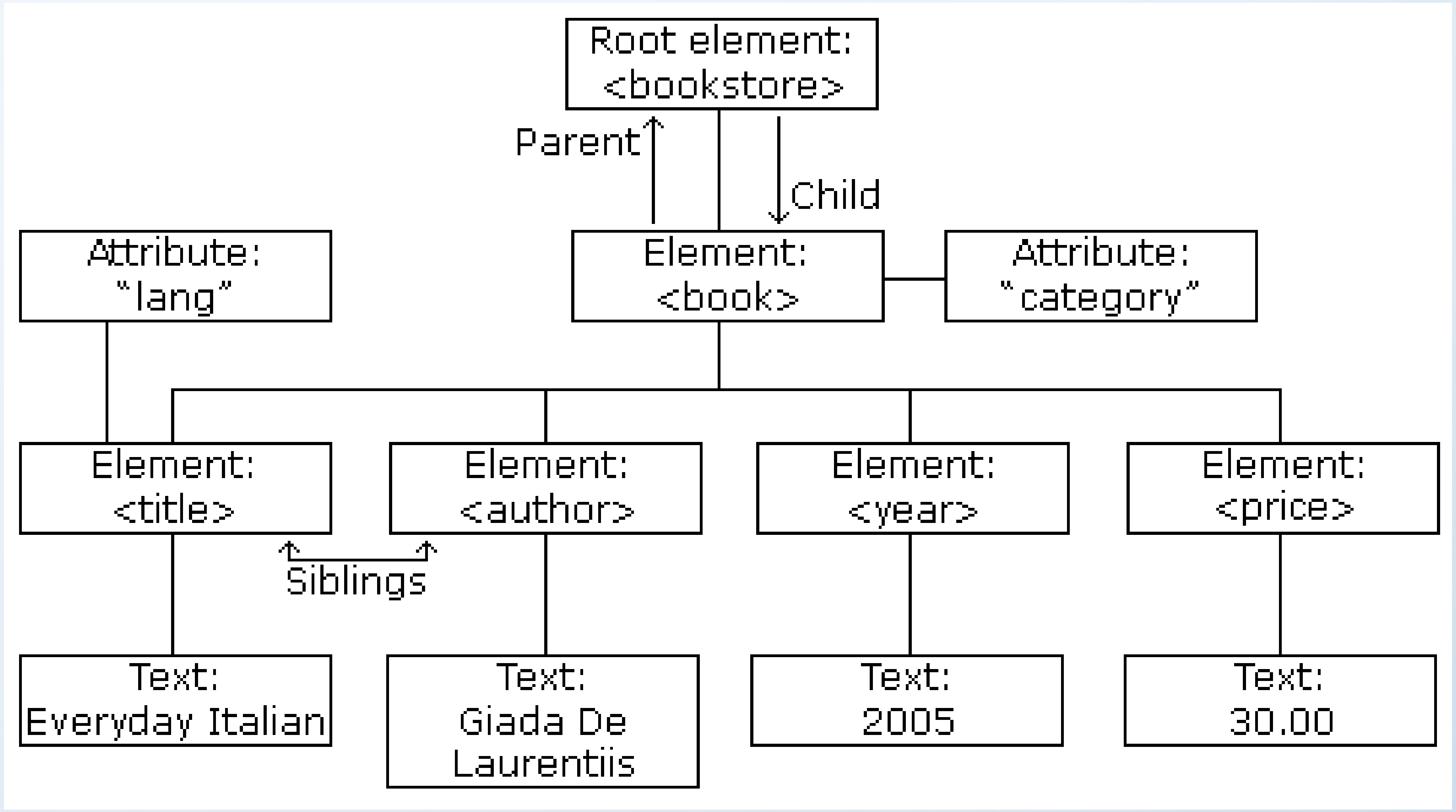
- To describe the relationships between elements, the terms parent, child, and sibling are utilized.
- Parents have kids. Parents exist for children. Siblings are children who are on the same grade level (brothers and sisters).
- Text content (Harry Potter) and attributes (category=”cooking”) are allowed for all elements.

Tree of XML: Code Version



```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="novel">
    <title lang="en">Two States</title>
    <author>Chetan Bhagath</author>
    <year>2005</year>
    <price>300.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>295.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>339.95</price>
  </book>
</bookstore>
```

Representing Content using XML Tree



Tree of XML

A prologue specifies the XML version as well as the character encoding:

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

The following line is the document's main component (root)

```
<bookstore>
```

The following line begins a <book> element:

```
<book category="cooking">
```

The elements <book> have four child elements: <title>, <author>, <year>, and <price>

```
<title lang="en">Two States</title>
<author>Chethan Bhagath</author>
<year>2003</year>
<price>200.00</price>
```

The following line brings the book element to a close:

```
</book>
```

XML Document

There are three types of XML documents:

- **Invalid Document**: Invalid documents don’t keep the syntax structure rules characterized by the XML particular. Once a developer characterizes some certain rules for what a document may contain in a DTD or schema, and the document doest observe those rules of a developer, then, that document remains invalid.
- **Well-formed** documents keep the XML syntax structure rules yet don’t have a DTD or pattern (schema).
- **Valid Document**: Valid documents observe both the XML syntax structure rules and the standards characterized in their DTD or composition (schema).

Rules for Writing XML Syntax

Following basic rules for building good XML:

- All XML must have a root element.
- All tags must be closed.
- All tags must be properly nested.
- Tag names have strict limits.(length)
- Tag names are case sensitive.
- Tag names cannot contain spaces.
- Attribute values must appear within quotes ("").
- White space is preserved.
- HTML tags should be avoided (optional).

XML Character Entities

- Anyplace the XML processor finds the string &dw;, it replaces the entity with the string. The XML-spec additionally characterizes five entities you can use instead of different special characters.
- An entity reference **must not** contain the name of an unparsed entity. Unparsed entities maybe referred to just in attribute values declared to be of type **entity or entities**.

<u>The entities are:</u>

- &lt; for the less than sign
- &gt; for the greater than sign
- &quot; for a double-quote
- &apos; for a single quote (or apostrophe)
- •&amp; for an ampersand.

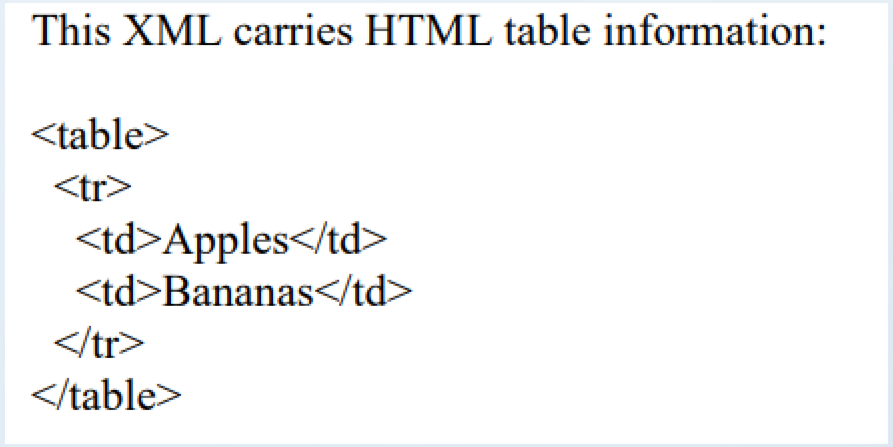
Namespaces

XML Namespaces

- Used to avoid element name conflict in XML document.
- It is a set of unique names
- Identified by URI (Uniform Resource Identifier)
- Attributes name must start with "**xmlns**"

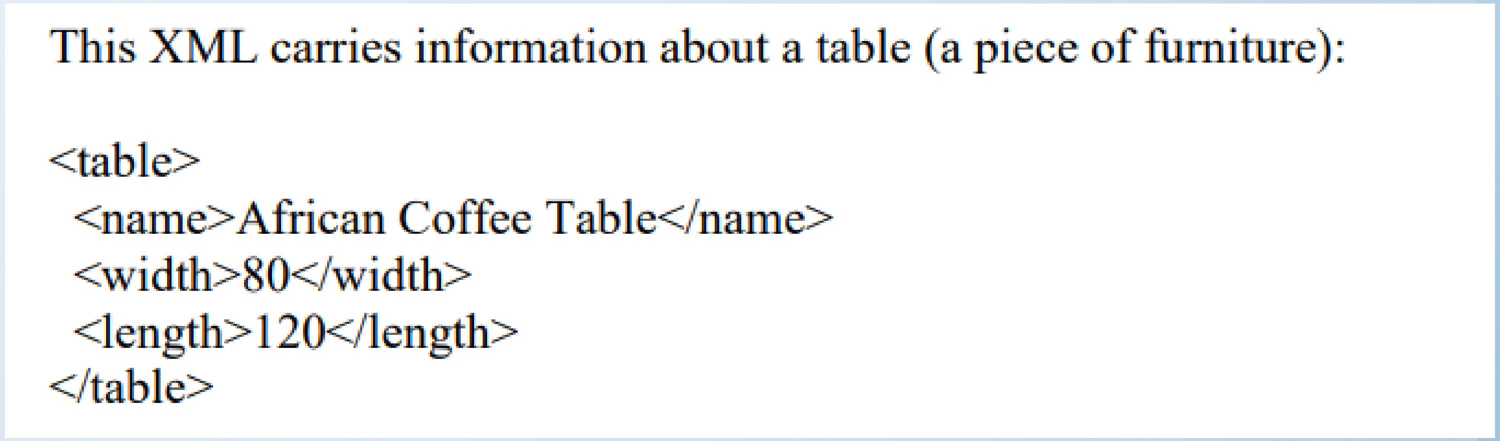
Namespaces: **Name Conflicts**

- In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.
- If these XML fragments were added together, there would be a name conflict. Both contain a
- <table> element, but the elements have different content and meaning.
- A user or an XML application will not know how to handle these differences.



This XML carries HTML table information:

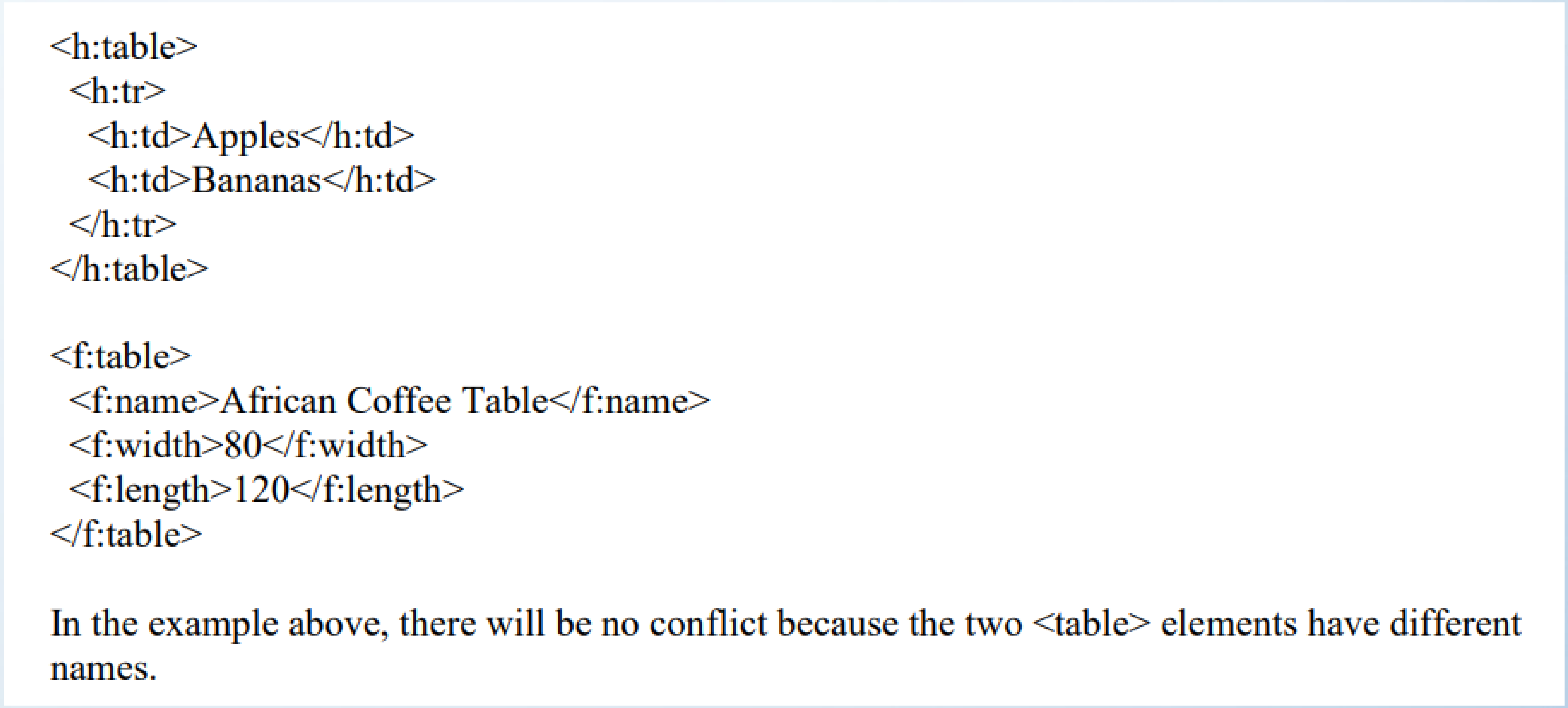
```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```



This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Namespaces: Solving the Name Conflict Using a Prefix Name

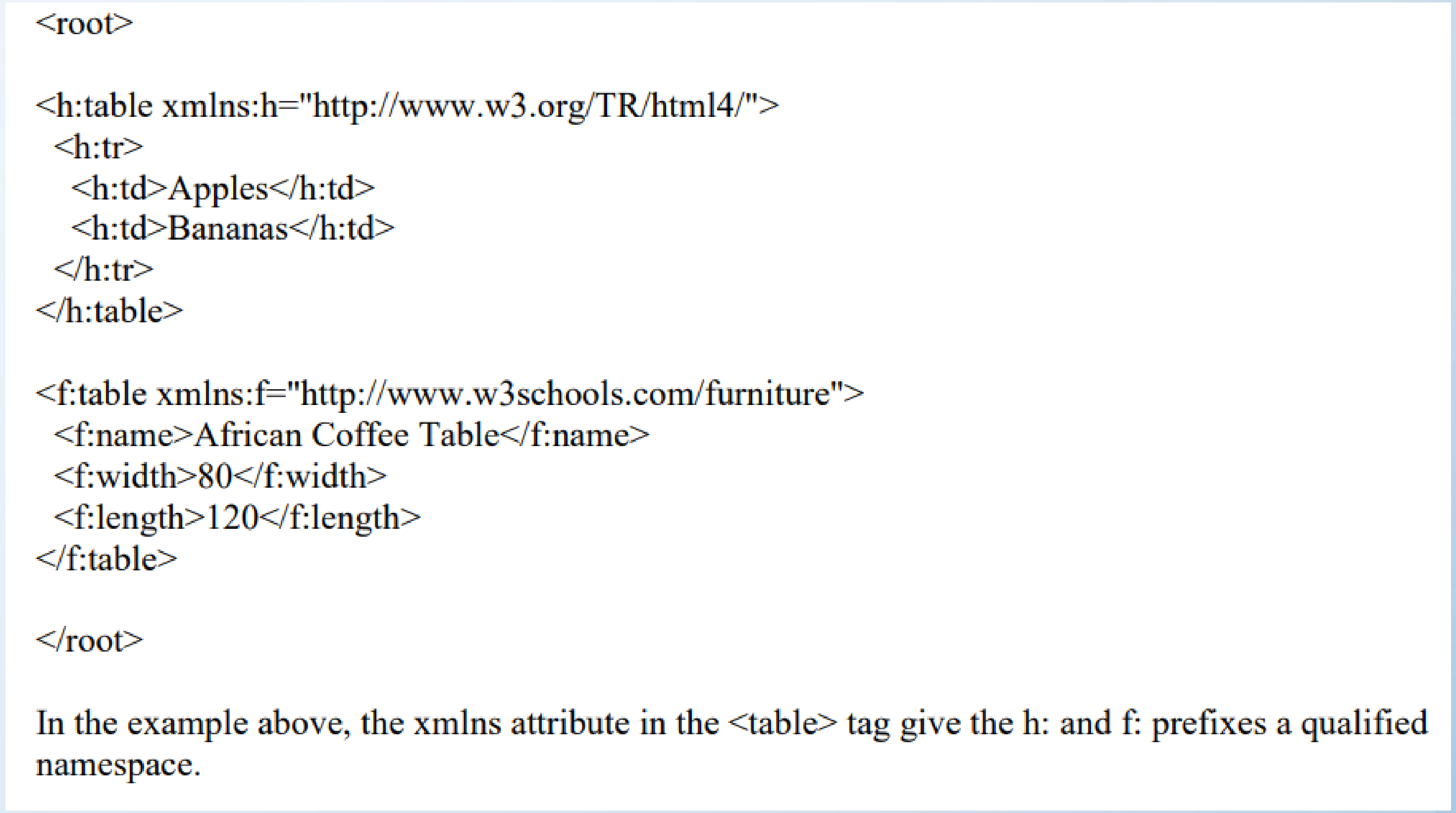


```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.

Namespaces: Solving the Name Conflict Using a Prefix Name : xmlns

- The xmlns Attribute When using prefixes in XML, a so-called namespace for the prefix must be defined.
- The namespace is defined by the **xmlns** attribute in the start tag of an element.
- The namespace declaration has the following syntax. **xmlns:prefix="URI".**



```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
 <h:tr>
  <h:td>Apples</h:td>
  <h:td>Bananas</h:td>
 </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
 <f:name>African Coffee Table</f:name>
 <f:width>80</f:width>
 <f:length>120</f:length>
</f:table>

</root>
```

In the example above, the xmlns attribute in the <table> tag give the h: and f: prefixes a qualified namespace.

XML-DTD (Document Type Declaration)

XML-DTD (Document Type Declaration)

- Describe XML language precisely.
- Used to define structure of a XML document.
- Contains list of legal elements
- Used to perform validation
- Check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.
- Can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Basic Syntax of DTD

```
<!DOCTYPE element DTD identifier [
          declaration1
          declaration2
          .....
]>
```

In the above syntax

- The **DTD** starts with <!DOCTYPE delimiter.
- An element tells the parser to parse the document from the specified root element.
- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset**.
- **The square brackets [ ]** enclose an optional list of entity declarations called Internal Subset.

Types of DTD

Internal

Elements are declared within the XML file.

Syntax:

<!DOCTYPE root_element [element-declaration]>

External

Elements are declared outside XML file. (Private/System & Public)

Syntax:

Private: <!DOCTYPE root_element SYSTEM "file-name">

Public: <!DOCTYPE root_element PUBLIC "DTD_name" "DTD_location">

Internal DTD

- Elements are declared within the XML files.
- standalone attribute in XML declaration must be set to **yes**.
  Syntax:

  <!DOCTYPE root-element [element-declarations]>

  here root-element is the name of root element and element-declarations is where you declare the elements.

Example of Internal DTD

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
        <!ELEMENT address (name, company, phone)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT company (#PCDATA)>
        <!ELEMENT phone (#PCDATA)>
]>
<address>
        <name>Your Name</name>
        <company>XYZ Company</company>
        <phone>(011) 123-4567</Phone>
</address>
```

Rules for Internal DTD

- The document type declaration must appear at the start of the document (preceded only by the XML header) - it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

- Elements are declared outside the XML file.
- Accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL.
- standalone attribute in the XML declaration must be set as no.

Syntax:

<!DOCTYPE root-element SYSTEM "file-name">

here *file-name* is the file with **.dtd** extension.

The content of the DTD file **address.xml** are as shown:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
        <name>Your Name</name>
        <company>XYZ Company</company>
        <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name, company, phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Refer to an external DTD by using either **system identifiers** or **public identifiers**.

SYSTEM IDENTIFIERS

A system identifier enables you to specify the location of an external file containing DTD declarations.

Syntax:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

DTD Element Specification

Elements in a document are defined by rules of the form:

<!ELEMENT nameOfElement contentOfElement>

Possible Content for element

- Sequence of elements (use a comma)
  - <!ELEMENT name (first, middle, last)>
- Choice of elements (use |)
  - <!ELEMENT student (undergrad | grad | nondegree)>
- Textual data (parsed character data)
  - <!ELEMENT first (#PCDATA)>

DTD Element Specification

- Optional elements (use ?)
  - <!ELEMENT name (first, middle?, last)>
- Repeated elements (use * and +)

<!ELEMENT entries (entry*)> <!-- zero or more elements -->

<!ELEMENT entries (entry+)> <!-- one or more elements -->

- Parentheses can be used for grouping

<!ELEMENT article (title, (abstract | body))>

<!ELEMENT article ((title, abstract) | body)>

DTD Element Specification

- Empty content

<!ELEMENT signal EMPTY>

- Any content (might be used during development)

<!ELEMENT tag ANY>

- Mixed content

<!ELEMENT narrative (#PCDATA | italics | bold)*>

Mixed Note: #PCDATA must come first; no sequencing (,) and no occurrence modifiers (*, +, ?) are allowed on sub elements.

DTD Example

```
<phoneNumbers>
 <title>Phone Numbers</title>
 <entries>
 <entry>
 <name>
 <first>Rusty</first>
 <last>Nail</last>
 </name>
<phone>335-0055</phone>
 <city>Iowa City</city>
 </entry>
 <entry>
 <name>
 <first>Helen</first>
 <last>Back</last>
 </name>
 <phone>337-5967</phone>
 <city>Iowa City</city>
 </entry>
 </entries>
</phoneNumbers>
```

```
<!DOCTYPE phoneNumbers [
<!ELEMENT phoneNumbers (title, entries)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT entries (entry*)>
<!ELEMENT entry (name, phone, city?)>
<!ELEMENT name (first, middle?, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT middle (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT city (#PCDATA)>
]>
```

DTD Attributes

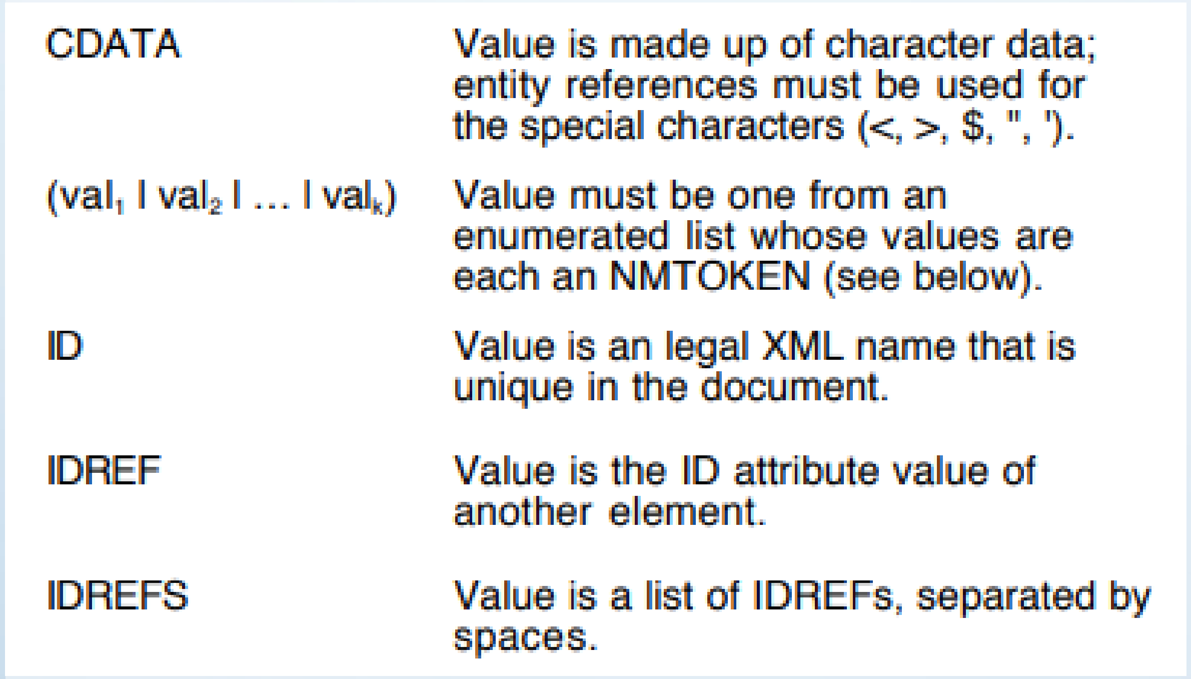
- The attributes for any element can be specified using a rule of the form (order makes no difference):

```
<!ATTLIST elemName attName1 attType restriction/default
attName2 attType restriction/default
attName3 attType restriction/default>
```

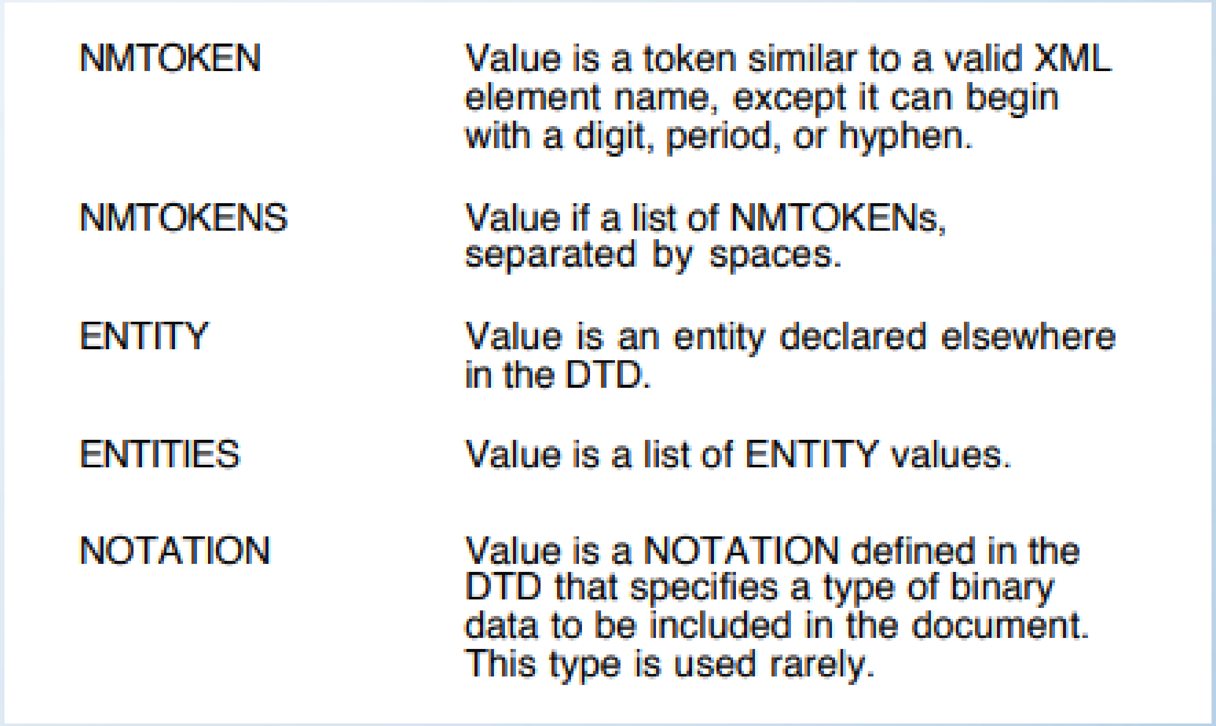
- Some designers prefer individual rules.

```
<!ATTLIST elemName attName1 attType restriction/default>
<!ATTLIST elemName attName2 attType restriction/default>
<!ATTLIST elemName attName3 attType restriction/default>
```

DTD Attributes: The possible formats



CDATA	Value is made up of character data; entity references must be used for the special characters (<, >, $, ", ').
(val_1 | val_2 | … | val_k)	Value must be one from an enumerated list whose values are each an NMTOKEN (see below).
ID	Value is an legal XML name that is unique in the document.
IDREF	Value is the ID attribute value of another element.
IDREFS	Value is a list of IDREFs, separated by spaces.



NMTOKEN	Value is a token similar to a valid XML element name, except it can begin with a digit, period, or hyphen.
NMTOKENS	Value if a list of NMTOKENs, separated by spaces.
ENTITY	Value is an entity declared elsewhere in the DTD.
ENTITIES	Value is a list of ENTITY values.
NOTATION	Value is a NOTATION defined in the DTD that specifies a type of binary data to be included in the document. This type is used rarely.

https://www.liquid-technologies.com/DTD/Datatypes/ID.aspx

Using ID and IDREFS

DTD

```
<!ELEMENT artist EMPTY>

<!ATTLIST artist name CDATA #REQUIRED>

<!ATTLIST artist artistID ID #REQUIRED>

<!ELEMENT album EMPTY>

<!ATTLIST album name CDATA #REQUIRED>

<!ATTLIST album albumArtistID IDREF #IMPLIED>

<!ATTLIST album contributingArtistIDs IDREFS #IMPLIED>
```

XML:

```
<artist name="Nick Cave" artistID="NC"/>

<artist name="Kylie Minogue" artistID="KM"/>

<artist name="PJ Harvey" artistID="PJH"/>

<artist name="Shane MacGowan" artistID="SM"/>

<album name="Murder Ballads" albumArtistID="NC" contributingArtistIDs="KM PJH SM"/>
```

Using ID and IDREFS

Similar to anchors in HTML

```
<!ELEMENT book (title,author,publisher)>

<!ATTLIST book destination ID #REQUIRED>

<!ELEMENT bestbooks (bookref)* >

<!ELEMENT bookref (#PCDATA)>

<!ATTLIST bookref target IDREF #IMPLIED>
```

So a book might be:

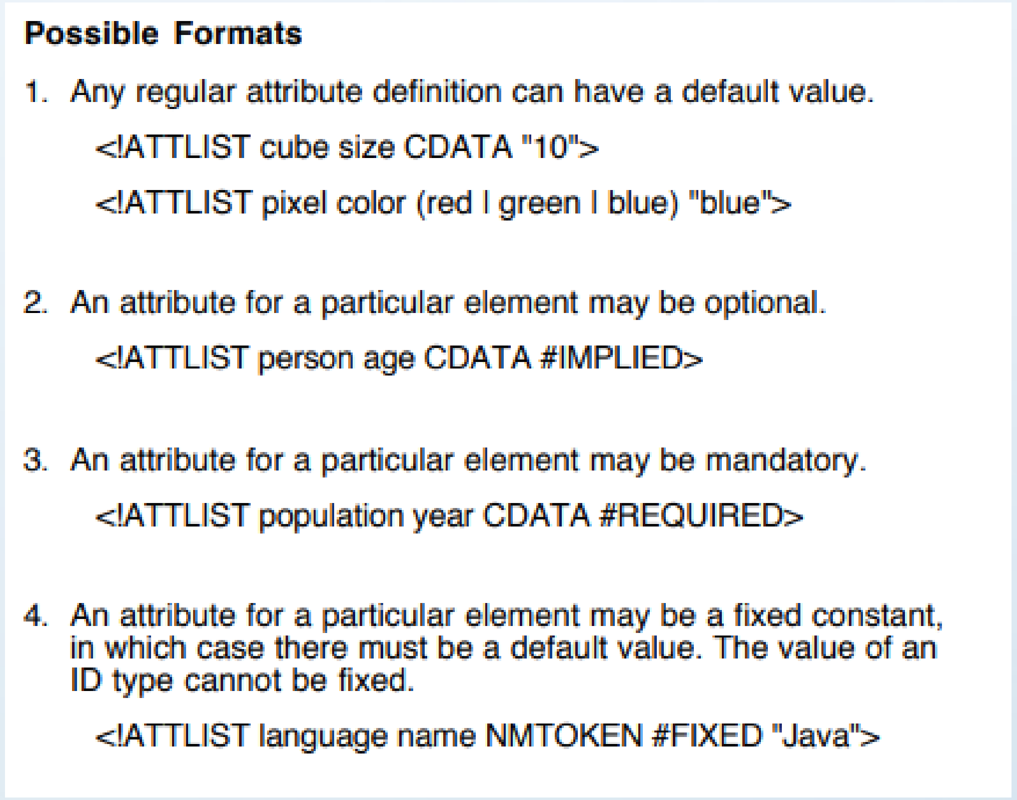
```
  <book destination="BookId28752">

    <title>Introduction to GKS</title>

    <author>Bob Hopgood</author>

<publisher>Academic</publisher>

  </book>
```

And a reference to it might be:

```
  <bestbooks>

    <bookref target="BookId28752">Introduction to GKS</bookref>

    <bookref>GKS for Dummies</bookref>

  </bestbooks>
```

DTD Attributes: Restrictions and Default Values

- Restrictions and Default Values Some keywords put restrictions on the occurrences of an attribute and its value. Further, in some situations, a default attribute value can be specified in the DTD, but note that some parsers may not recognize default values.



Possible Formats

1. Any regular attribute definition can have a default value.

```
<!ATTLIST cube size CDATA "10">
<!ATTLIST pixel color (red I green I blue) "blue">
```

2. An attribute for a particular element may be optional.

```
<!ATTLIST person age CDATA #IMPLIED>
```

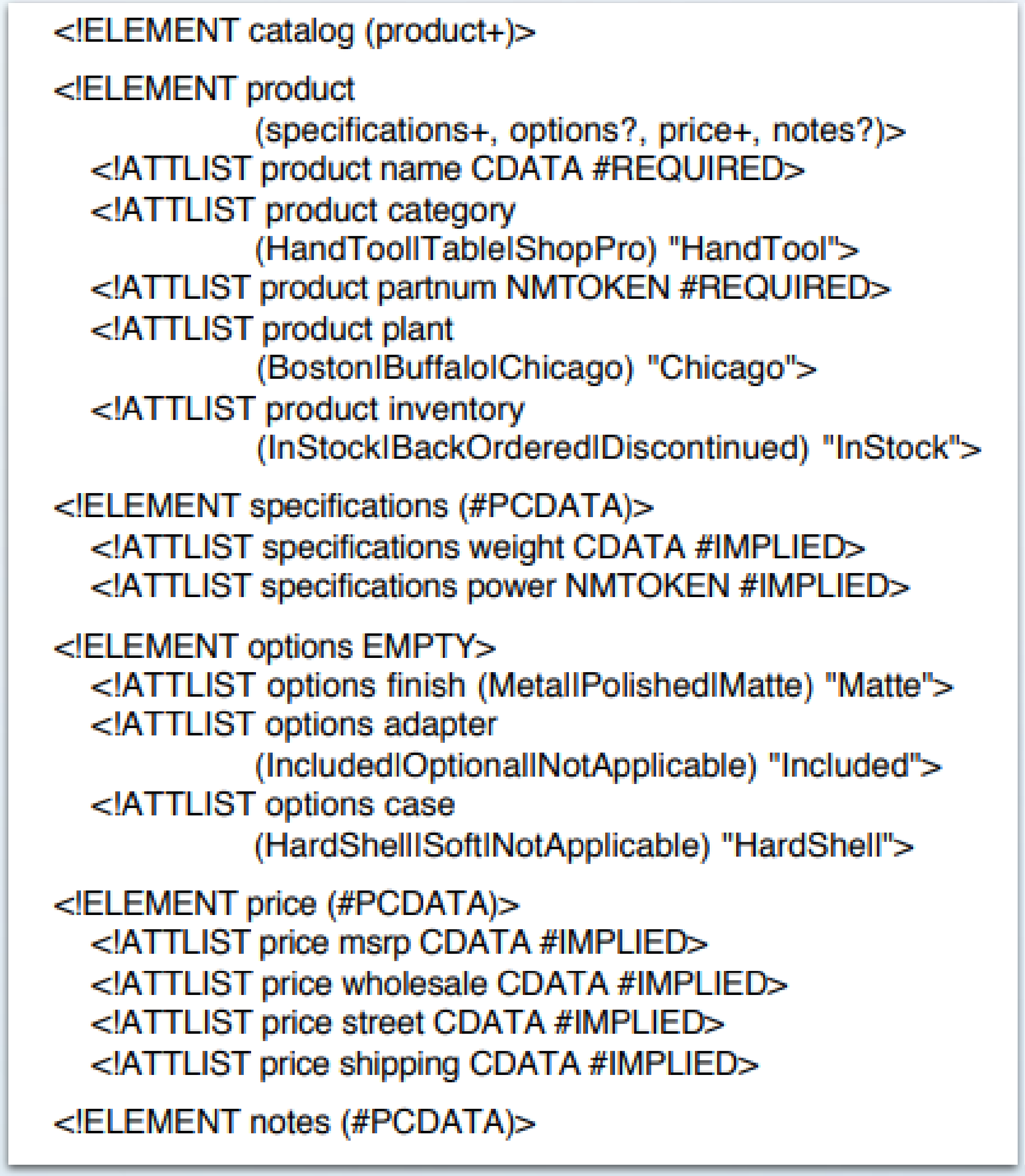
3. An attribute for a particular element may be mandatory.

```
<!ATTLIST population year CDATA #REQUIRED>
```

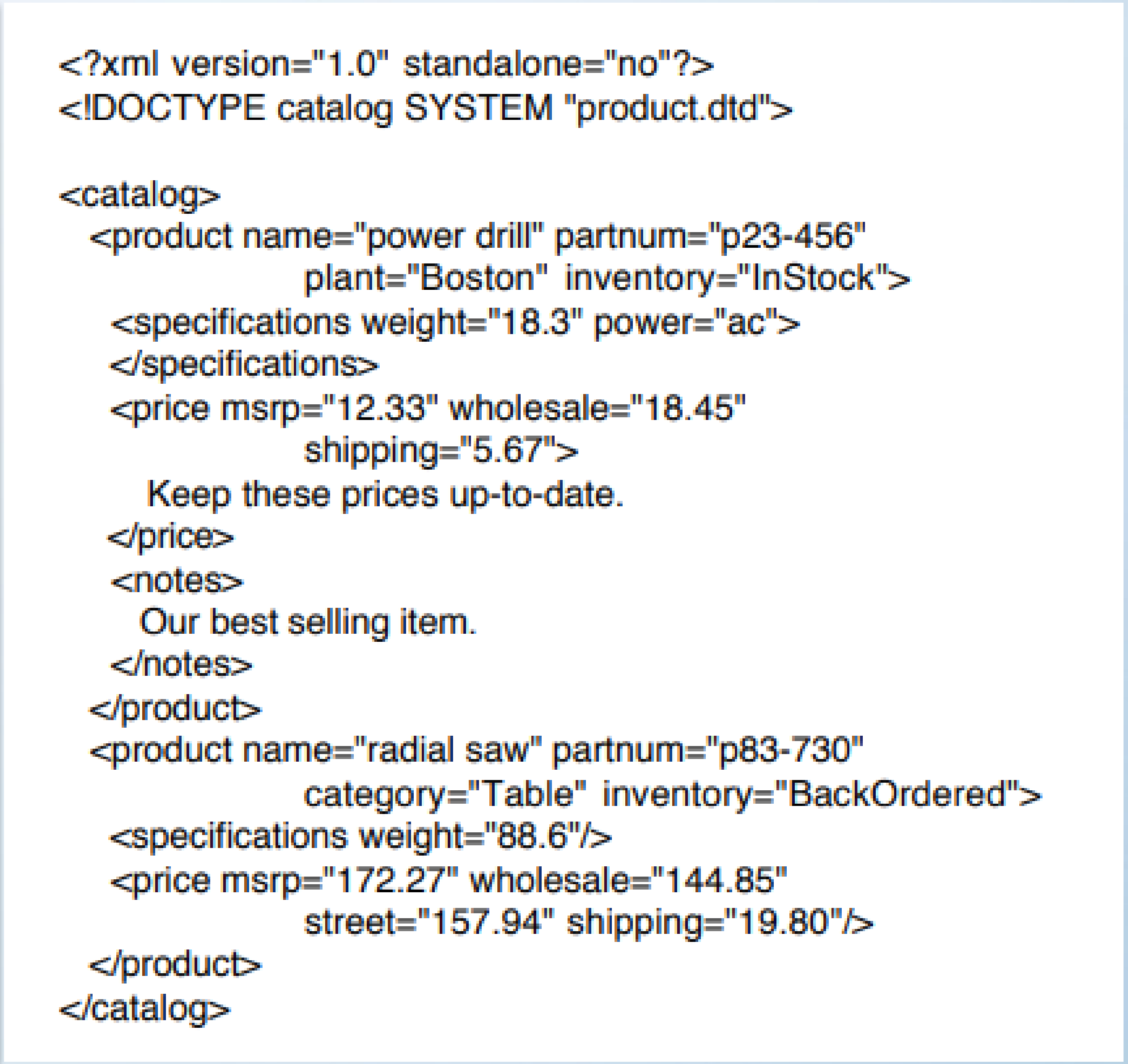
4. An attribute for a particular element may be a fixed constant, in which case there must be a default value. The value of an ID type cannot be fixed.

```
<!ATTLIST language name NMTOKEN #FIXED "Java">
```

Example :Attribute



```
<!ELEMENT catalog (product+)>
<!ELEMENT product
                (specifications+, options?, price+, notes?)>
   <!ATTLIST product name CDATA #REQUIRED>
   <!ATTLIST product category
                (HandTool|Table|ShopPro) "HandTool">
   <!ATTLIST product partnum NMTOKEN #REQUIRED>
   <!ATTLIST product plant
                (Boston|Buffalo|Chicago) "Chicago">
   <!ATTLIST product inventory
                (InStock|BackOrdered|Discontinued) "InStock">
<!ELEMENT specifications (#PCDATA)>
   <!ATTLIST specifications weight CDATA #IMPLIED>
   <!ATTLIST specifications power NMTOKEN #IMPLIED>
<!ELEMENT options EMPTY>
   <!ATTLIST options finish (Metal|Polished|Matte) "Matte">
   <!ATTLIST options adapter
                (Included|Optional|NotApplicable) "Included">
   <!ATTLIST options case
                (HardShell|Soft|NotApplicable) "HardShell">
<!ELEMENT price (#PCDATA)>
   <!ATTLIST price msrp CDATA #IMPLIED>
   <!ATTLIST price wholesale CDATA #IMPLIED>
   <!ATTLIST price street CDATA #IMPLIED>
   <!ATTLIST price shipping CDATA #IMPLIED>
<!ELEMENT notes (#PCDATA)>
```



```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE catalog SYSTEM "product.dtd">

<catalog>
  <product name="power drill" partnum="p23-456"
                plant="Boston" inventory="InStock">
   <specifications weight="18.3" power="ac">
   </specifications>
   <price msrp="12.33" wholesale="18.45"
                shipping="5.67">
     Keep these prices up-to-date.
   </price>
   <notes>
     Our best selling item.
   </notes>
  </product>
  <product name="radial saw" partnum="p83-730"
                category="Table" inventory="BackOrdered">
   <specifications weight="88.6"/>
   <price msrp="172.27" wholesale="144.85"
                street="157.94" shipping="19.80"/>
  </product>
</catalog>
```

Public DTDs

- A DTD intended for widespread usage is denoted with the keyword PUBLIC and a Formal Public Identifier (FPI) followed by a URI specifying its location if needed.
- **Formal Public Identifiers**
  - Official names of resources such as DTDs.
  - Format: **standardsIndicator //organizationName //DTDname //language**
- **Standards Indicator**
  - ❖ **+** Means document has been approved by a standards body such as ISO.
  - ❖ **-** Means document has no backing from a standards group; W3C is not a standards organization; it only makes recommendations.

Public DTDs

- **Organization Name**
  - This indicates the name of the organization or company that is responsible for the DTD.
- **DTD Name**
  - A descriptive name for the DTD, including a version number.
- **Language**
  - A language code, using the ISO 639 list, that specifies which language was used to write the DTD.

```
<!DOCTYPE score-partwise PUBLIC
 "-//Recordare//DTD MusicXML 0.7a Partwise//EN"
 "http://www.musicxml.org/dtds/partwise.dtd">
```

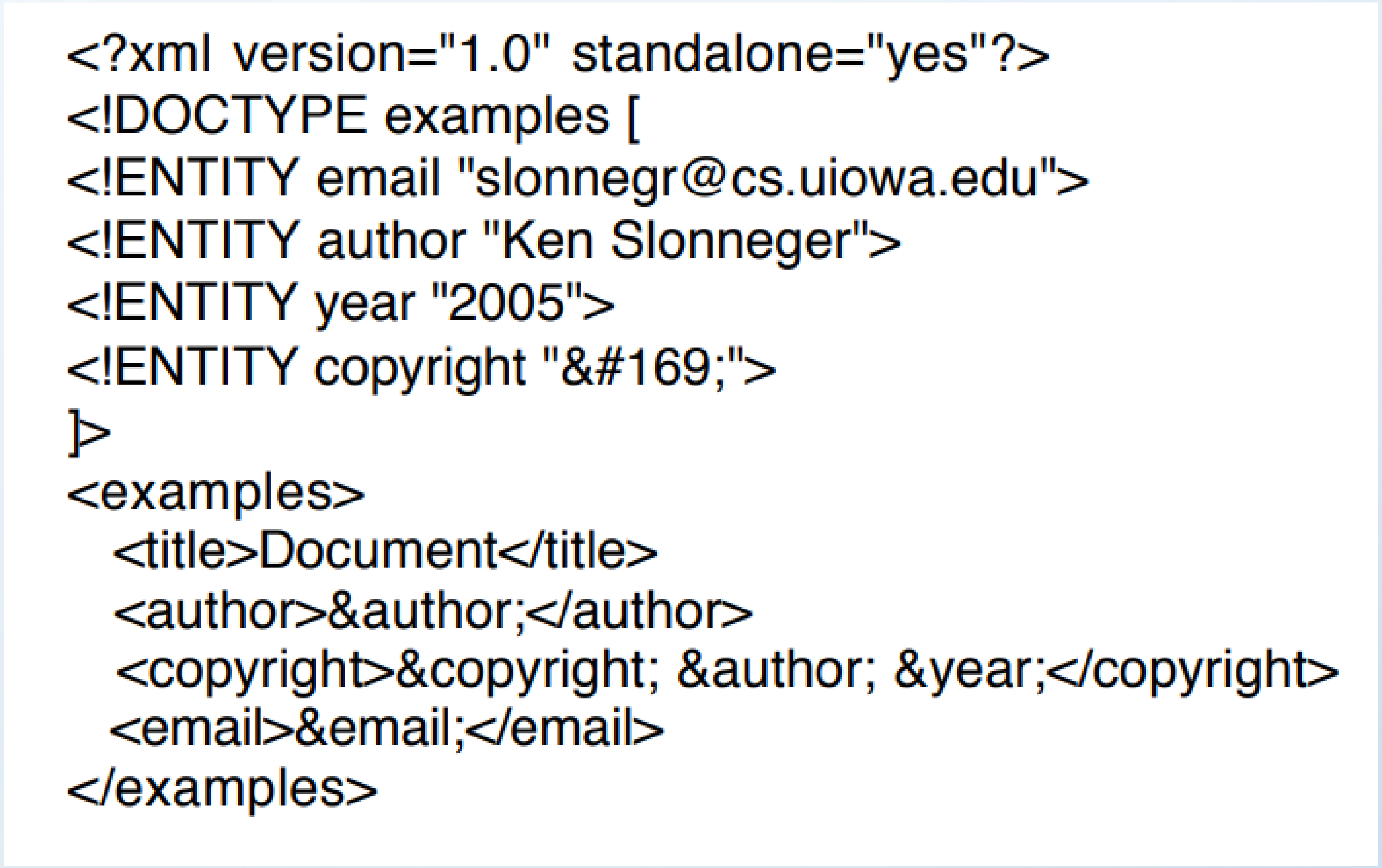
Entities in DTDs

- An entity definition plays a role similar to a macro definition (#define in C) or a constant specification in a programming language. It is one of the four kinds of rules in a DTD.

  <!ENTITY entName "substitution text">

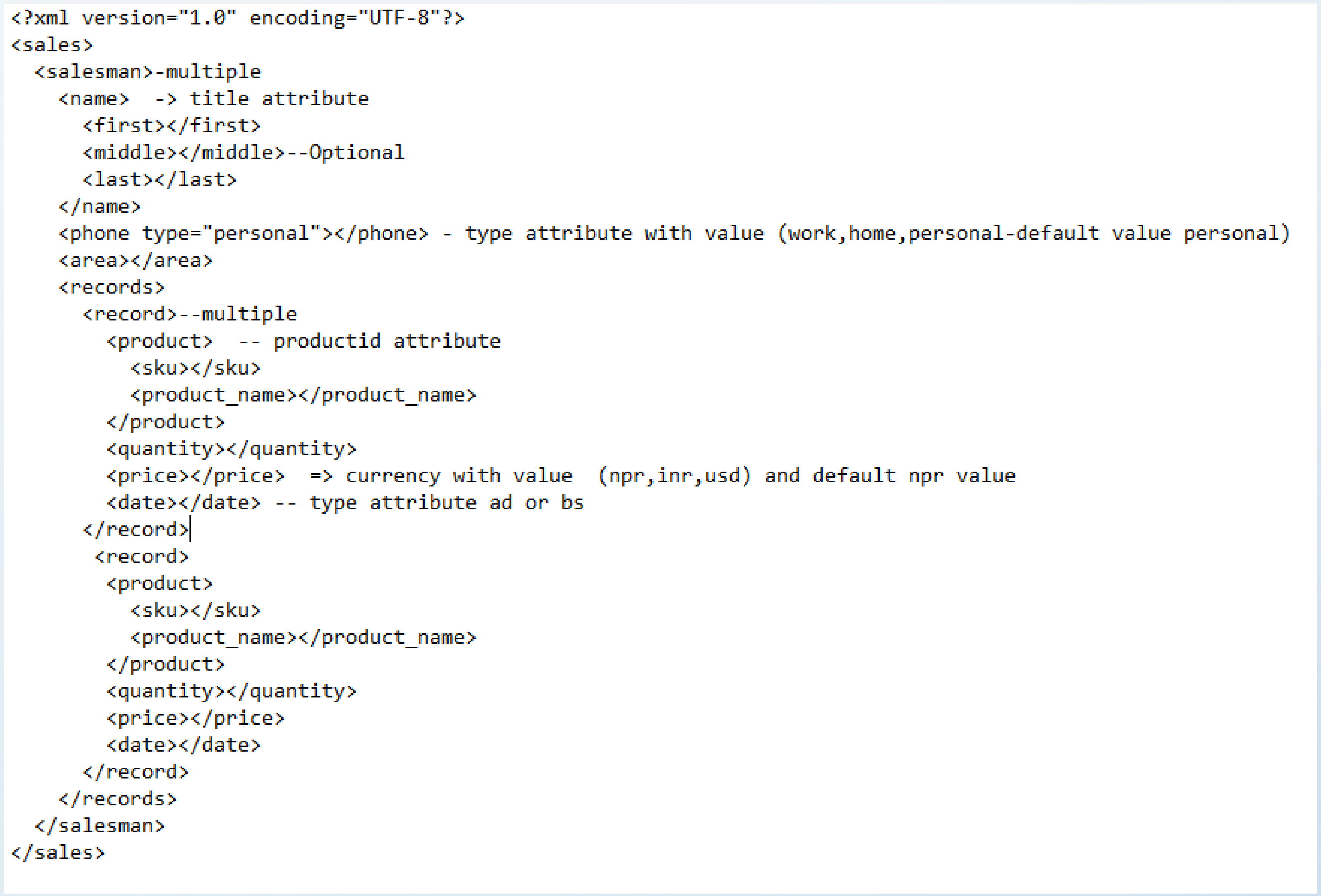
- **Advantages**
  - An entity acts as an alias for commonly used text.
  - Repetitive typing can be reduced.
  - Items that might change many places in an XML document can be collected into entity definitions in the DTD.
- **Entity Values**
  - Parsed entities
    - well-formed XML
  - Unparsed entities
    - other forms of text
    - binary data

Entities in DTDs :Example



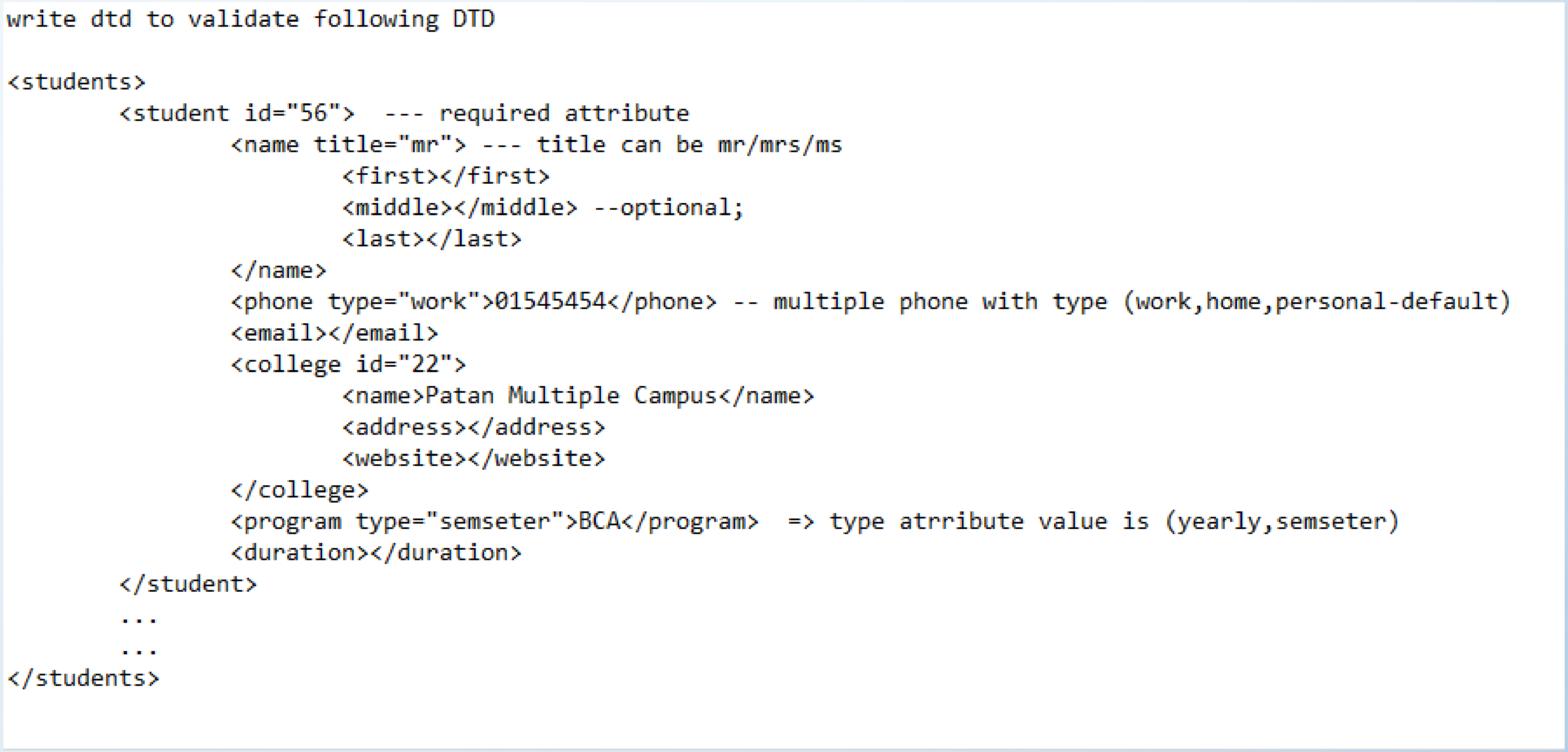
```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE examples [
<!ENTITY email "slonnegr@cs.uiowa.edu">
<!ENTITY author "Ken Slonneger">
<!ENTITY year "2005">
<!ENTITY copyright "&#169;">
]>
<examples>
   <title>Document</title>
   <author>&author;</author>
   <copyright>&copyright; &author; &year;</copyright>
   <email>&email;</email>
</examples>
```

Tasks: Create DTD to validate following structure of XML Document



```
<?xml version="1.0" encoding="UTF-8"?>
<sales>
  <salesman>-multiple
    <name>  -> title attribute
      <first></first>
      <middle></middle>--Optional
      <last></last>
    </name>
    <phone type="personal"></phone> - type attribute with value (work,home,personal-default value personal)
    <area></area>
    <records>
      <record>--multiple
        <product>  -- productid attribute
          <sku></sku>
          <product_name></product_name>
        </product>
        <quantity></quantity>
        <price></price>  => currency with value  (npr,inr,usd) and default npr value
        <date></date> -- type attribute ad or bs
      </record>
       <record>
        <product>
          <sku></sku>
          <product_name></product_name>
        </product>
        <quantity></quantity>
        <price></price>
        <date></date>
      </record>
    </records>
  </salesman>
</sales>
```

Tasks: Create DTD to validate following structure of XML Document



```
write dtd to validate following DTD

<students>
        <student id="56">  --- required attribute
                <name title="mr"> --- title can be mr/mrs/ms
                        <first></first>
                        <middle></middle> --optional;
                        <last></last>
                </name>
                <phone type="work">01545454</phone> -- multiple phone with type (work,home,personal-default)
                <email></email>
                <college id="22">
                        <name>Patan Multiple Campus</name>
                        <address></address>
                        <website></website>
                </college>
                <program type="semseter">BCA</program>  => type atrribute value is (yearly,semseter)
                <duration></duration>
        </student>
        ...
        ...
</students>
```

Problem of DTD

- Written in a language other than XML; so need a separate parser.
- All definitions in a DTD are global, applying to the entire document. Cannot have two elements with the same name but with different content in separate contexts.
- The text content of an element can be PCDATA only; no finer typing for numbers or special string formats.
- Limited typing for attribute values.
- DTDs are not truly aware of namespaces; they recognize prefixes but not the underlying URI.

XML Schemas

XML Schemas – Commonly known as **XML Schema Definition** (XSD). It is used to describe & validate the structure and content of XML Data.

- Also known as XML Schema Definition (XSD).
- Used to describe and validate the structure and the content of XML data.
- Defines the elements, attributes and data types.
- Schema element supports Namespaces.
- Similar to a database schema that describes the data in a database.
- Describe the legitimate format that an XML document can take.
- It is a method of expressing constraints about XML documents
- It is like DTD but provides more control on XML structure.

XML Schemas Syntax

- An XML Schema is an XML document.
- First item: xml declaration

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

- XML comments and processing instructions are allowed.
- Root element: **schema** with a namespace declaration.

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- schema rules go here -->
</xs:schema>
```

- Possible namespace prefixes: xs, xsd, or none

schema element

- The <schema> element is the root element of every XML Schema.
- The <schema> element may contain some attributes.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">
...
...
</xs:schema>
```

schema element

xmlns:xs=http://www.w3.org/2001/XMLSchema

- Above information indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace.
- It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with **xs:**

schema element

targetNamespace=https://www.websitename.com

- **targetNamespace** indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "https://www.websitename.com" namespace.

- **xmlns=https://www.websitename.com**

Xmlns indicates that the default namespace is "https://www.websitename.com".

- **elementFormDefault="qualified“**

elementFormDefault indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

xs:element tag

- All elements defined at the top level of a schema (i.e. with an xs:element immediately below the document xs:schema element) are considered to be defined globally.

xs:element tag

```
<element
name = NCName
 type = Qname
 maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
 ref = QName
  abstract = Boolean : false
  block = (#all | List of (extension | restriction | substitution))
  default = string
  final = (#all | List of (extension | restriction))
  fixed = string
  form = (qualified | unqualified)
  id = ID
nillable = Boolean : false
substitutionGroup = QName
  {any attributes with non-schema Namespace}...>
Content: (annotation?, ((simpleType | complexType)?, (unique | key |
keyref)*))
</element>
```

xs:element tag

Name: Optional.

- The name of the element. The name must be a no-colon-name (NCName) as defined in the XML Namespaces specification.
- **Name** and **ref** attributes cannot both be present.
- Required if the containing element is the **schema** element.

Ref : Optional.

- The name of an element declared in this schema (or another schema indicated by the specified namespace).
- The **ref** value must be a QName. The **ref** can include a namespace prefix.
- Prohibited if the containing element is the **schema** element.
- If the **ref** attribute is present, **complexType**, **simpleType**, **key**, **keyref**, and **unique** elements and **nillable**, **default**, **fixed**, **form**, **block**, and **type** attributes cannot be present.

xs:element tag

Type:optional

- Either the name of a built-in data type, or the name of a **simpleType** or **complexType** element defined in this schema (or another schema indicated by the specified namespace).
- The supplied value must correspond to the **name** attribute on the **simpleType** or **complexType** element that is referenced.
- The **type** and **ref** attributes are mutually exclusive.
- To declare an element using an existing simple type or complex type definition, use the **type** attribute to specify the existing type.

xs:element tag

maxOccurs : optional

- The maximum number of times the element can occur within the containing element. The value can be an integer greater than or equal to zero.
- To set no limit on the maximum number, use the string "unbounded".
- Prohibited if the containing element is the **schema** element.

minOccurs: optional

- The minimum number of times the element can occur within the containing element.
- The value can be an integer greater than or equal to zero.
- To specify that this element is optional, set this attribute to zero.
- Prohibited if the containing element is the **schema** element.

xs:attribute tag

- Attribute represents the attribute of an XML element. XSD defines it as a simple type.
- <xs:attribute name = "attribute-name" type = "attribute-type"/>

attribute-name	Name of the Attribute. For example,<xs:attribute name = "rollno" type = "xs:integer"/> defines following rollno attribute which can be used in an XML element. For example <student rollno = "393" />
attribute-type	Type of the Attribute. For example,<xs:attribute name = "rollno" type = "xs:integer"/> defines type of attribute as integer, rollno should have value of type int. <student rollno = "393" />

xs:attribute tag

Default Value

Attribute can have a default value assigned to it. Default value is used in case the attribute has no value.

```
<xs:attribute name = "grade" type = "xs:string" default = "NA" />
```

Fixed Value

Attribute can have a fix value assigned. In case a fixed value is assigned, then the element can not have any value.

```
<xs:attribute name = "class" type = "xs:string" fixed = "1" />
```

Restriction

Attributes are by default optional. But to make an attribute mandatory, "use" attribute can be used.

```
<xs:attribute name = "rollno" type = "xs:integer" use = "required"/>
```

Example (contact.xsd) : Schema File

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="contact">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="name" type="xs:string" />
                                <xs:element name="company" type="xs:string" />
                                <xs:element name="phone" type="xs:int" />
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Root Declaration: XML Inside XML File (contact.xml)

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

<contact xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

   xsi:noNamespaceSchemaLocation="contact.xsd">

   <name>Rai</name>

   <company>Tom</company>

   <phone>985454545</phone>

  </ contact >
```

XML document with Complex Type Schema Definitions implementation

#File “Add.xml”

```
<?xml version=“1.0” encoding=“UTF-8”?>
<Address xsi:schemaLocation=“./path/Add.xsd”>
    <Name>Aman</Name>
    <Phone>9840051645</Phone>
</Address>
```

#File “Add.xsd”

```
<?xml version=“1.0” encoding=“UTF-8”?>
<xs:schema xmlns:xs=“Schema1…”>
  <xs:element name=“Address”>
            <xs:complexType>
              <xs:sequence> //Child elements should appear in sequence
                        <xs:element name=“Name” type=“xs:String”/>
                        <xs:element name=“Phone” type=“xs:int”/>
             </xs:sequence>
            </xs:complexType>
   </xs:element>
</xs:schema>
```

Element Specification

- Elements are declared using an element named xs:element with an attribute that gives the name of the element being defined.
- The type of the content of the new element can be specified by another attribute or by the content of the xs:element definition.
- Element declarations can be one of two sorts.
  - Simple Type
  - Complex Type

Element Specification: Simple Type

- Content of these elements can be text only.

<xs:element name=“elementname" type=“datatype"/>

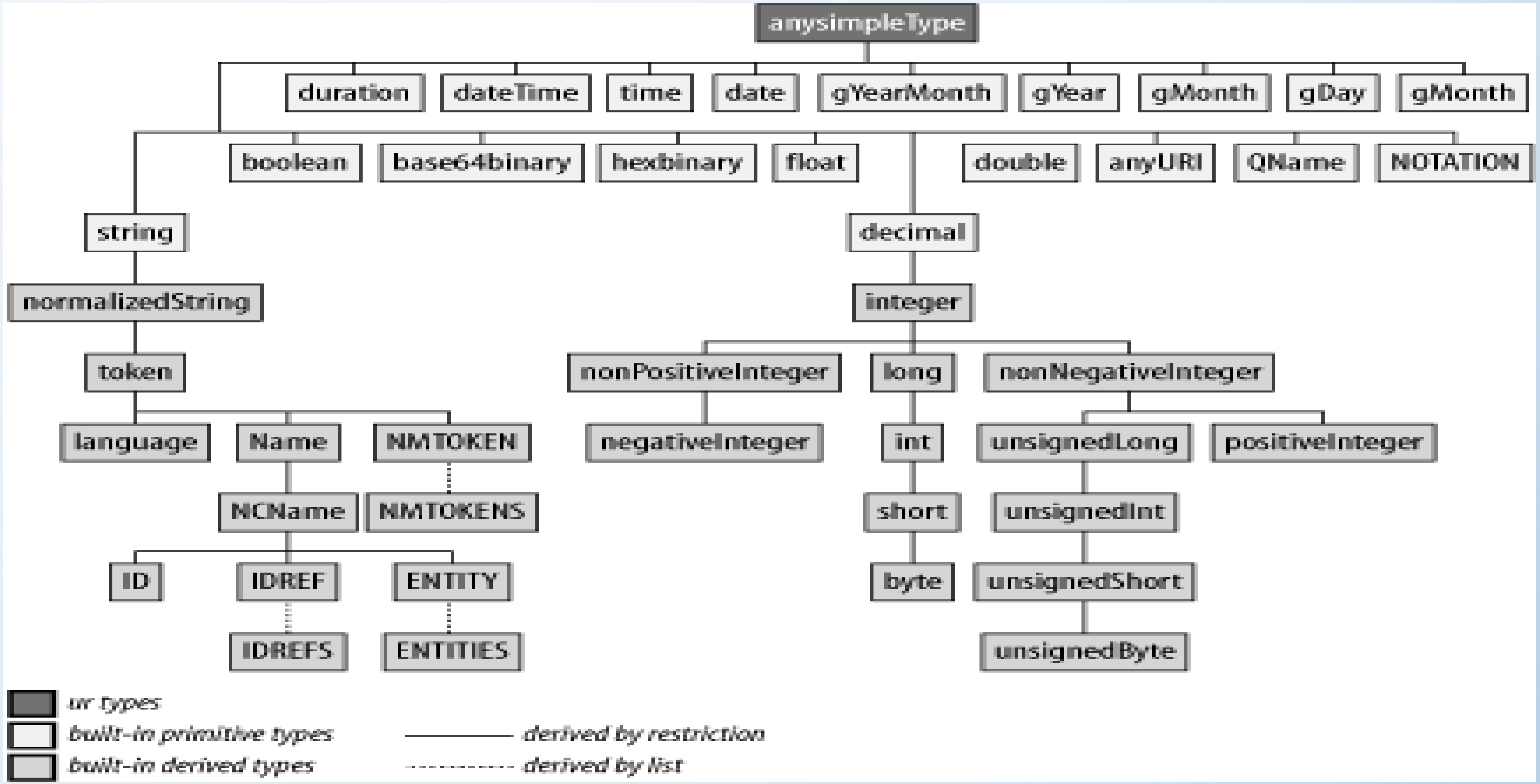
Examples

<xs:element name="item" type="xs:string"/>

<xs:element name="price" type="xs:decimal"/>

- The values xs:string and xs:decimal are two of the **44** simple
- Types predefined in the XML Schema language.

XML Schema Data Types



Example: Simple Type

With XML

simple.xml

```
<firstname xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
    i:schemaLocation="http://localhost:8000/simple.xsd">
    Hari
</firstname>
```

simple.sxd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="firstname" type="xsd:string" />
</xsd:schema>
```

Element Specification: Complex Type

- An XML element with at least one attribute of at least one child element is called a Complex Element.
- A Complex Element can be used to declare a simple element.
- Complex kinds can include characteristics, contain additional components, and mix elements and text, among other things.
- Child components or attributes are defined in a complex form of data.
- An element can have its own data types and can be declared as its own data datatype.
- The content of a complex element can be empty elements, other elements, text, or a combination of elements and text.

Element Specification: Complex Type

- **Sample DTD:**

```
<!ELEMENT location (city, state)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
```

- **Sample Schema**

```
<xs:element name="location">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="city" type="xs:string"/>
                        <xs:element name="state" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

Complex Type: Example

```
<xs:element name="Address">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="name" type="xs:string" />
                        <xs:element name="company" type="xs:string" />
                        <xs:element name="phone" type="xs:integer" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

Here, Address element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Example : DTD

```
<!ELEMENT phoneNumbers (title, entries)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT entries (entry*)>
<!ELEMENT entry (name, phone, city?)>
<!ELEMENT name (first, middle?, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT middle (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT city (#PCDATA)>
```

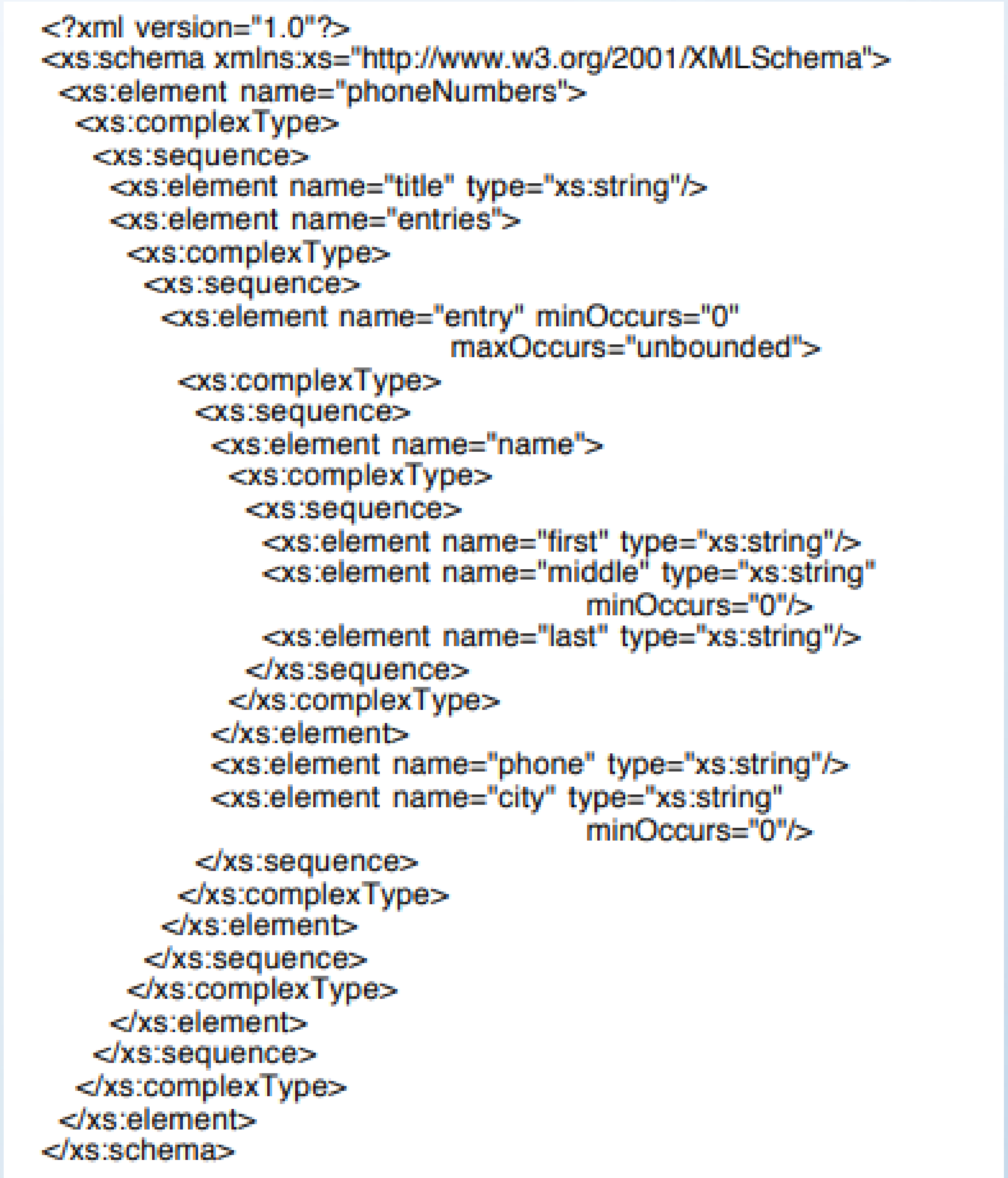
We use *xs:string* in place of PCDATA.

Element sequencing is handled with *xs:sequence*.

The attributes *minOccurs* and *maxOccurs* take care of the * and ? in the DTD.

```
<phoneNumbers>
 <title>Phone Numbers</title>
 <entries>
 <entry>
 <name>
 <first>Rusty</first>
 <last>Nail</last>
 </name>
<phone>335-0055</phone>
 <city>Iowa City</city>
 </entry>
 <entry>
 <name>
 <first>Helen</first>
 <last>Back</last>
 </name>
 <phone>337-5967</phone>
 <city>Iowa City</city>
 </entry>
 </entries>
</phoneNumbers>
```

Example :



```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="phoneNumbers">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="entries">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="entry" minOccurs="0"
                                 maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="name">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="first" type="xs:string"/>
             <xs:element name="middle" type="xs:string"
                                          minOccurs="0"/>
             <xs:element name="last" type="xs:string"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="phone" type="xs:string"/>
          <xs:element name="city" type="xs:string"
                                          minOccurs="0"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Alternative Naming: Global Type: Named Type

Defining a single type in document, which can be used by all other references.

Example: To generalize the person and company for different addresses of the company. In such case, we define a general type as below:

```
<xs:element name="AddressType">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="name" type="xs:string" />
                        <xs:element name="company" type="xs:string" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

```
<xs:element name="Address1">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="address" type="AddressType" />
                        <xs:element name="phone1" type="xs:integer" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:element name="Address2">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="address" type="AddressType" />
                        <xs:element name="phone2" type="xs:integer" />
                </xs:sequence>
```

Example: Complex Type with Attribute

complex.xml

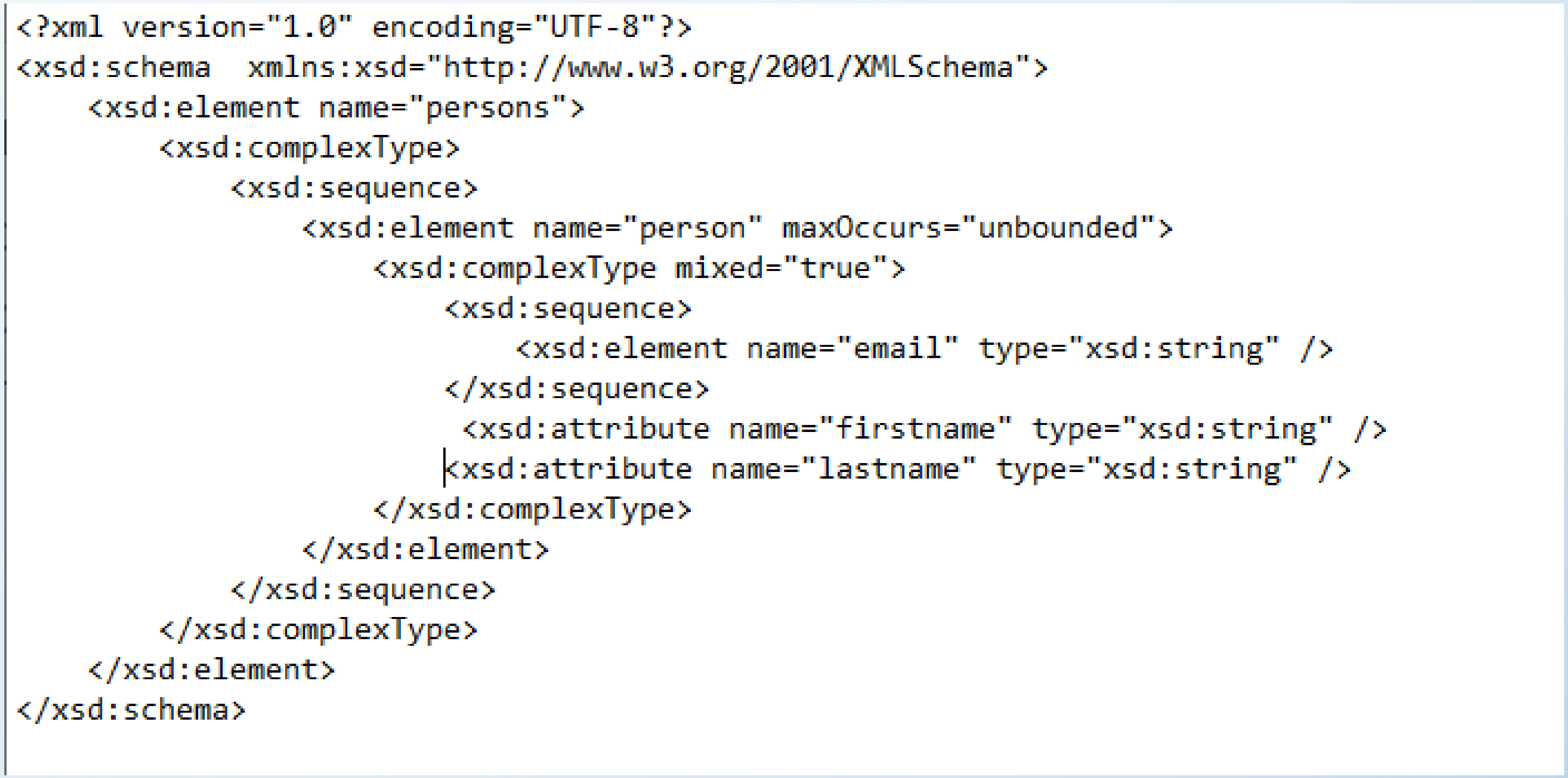
```
<?xml version="1.0" encoding="UTF-8"?>

<persons xmlns:i="http://www.w3.org/2001/XMLSchema-instance"

   i:schemaLocation="http://localhost:8000/complex.xsd">

   <person firstname="yourname" lastname="yourlastname">

      <email>info@yourname.name.np</email>

   </person>

   <person>

      <email>john@email.com</email>

   </person>

</persons>
```

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <xsd:element name="persons">

      <xsd:complexType>

         <xsd:sequence>

            <xsd:element name="person" maxOccurs="unbounded">

               <xsd:complexType mixed="true">

                  <xsd:sequence>

                     <xsd:element name="email" type="xsd:string" />

                  </xsd:sequence>


                  <!-- Attributes doesn't use order like element sequence -->

                  <xsd:attribute name="firstname" type="xsd:string" />

                  <xsd:attribute name="lastname" type="xsd:string" />

               </xsd:complexType>

            </xsd:element>
```

Example: Complex Type with Attribute :complex.xml

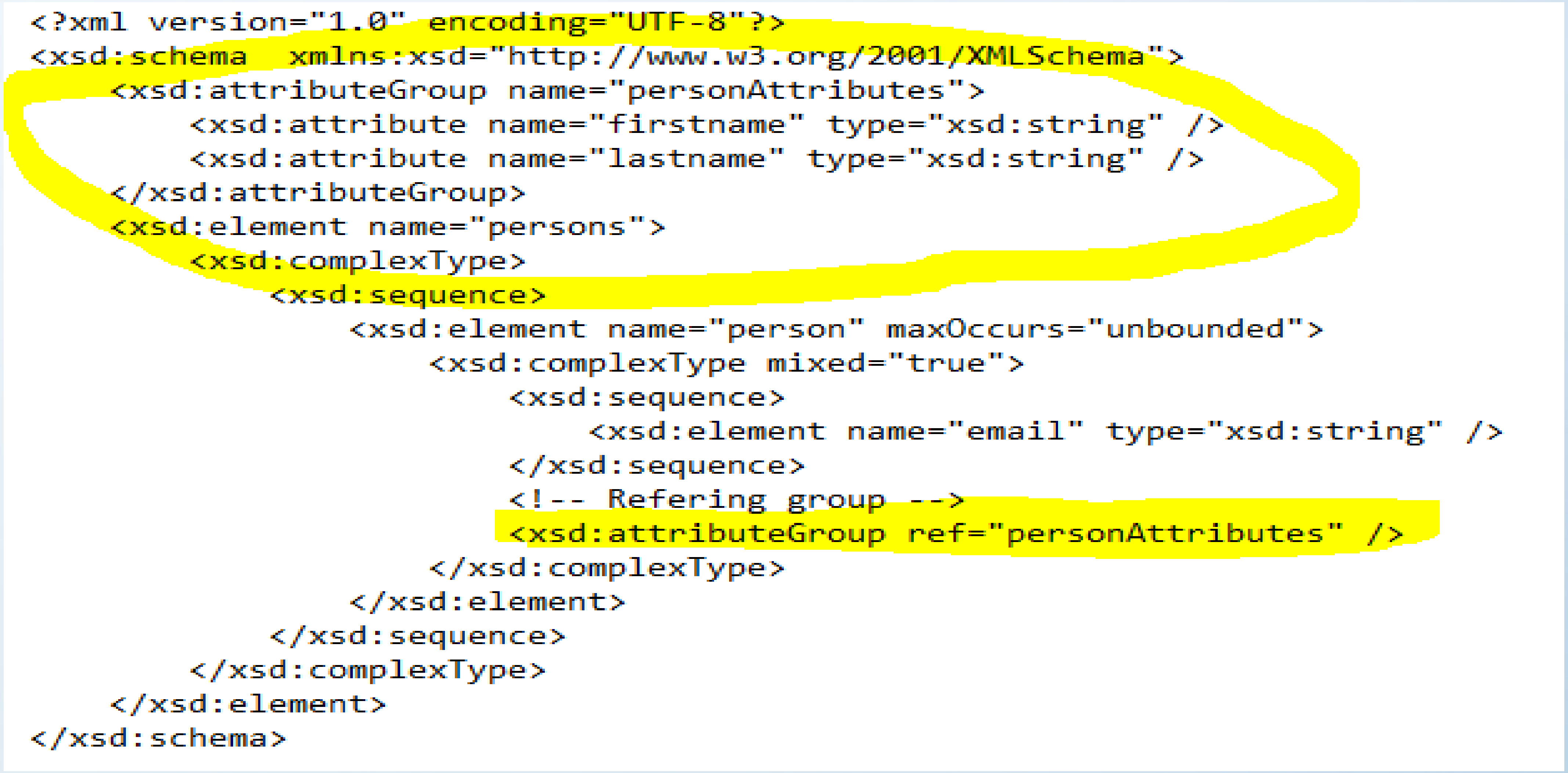
```
<?xml version="1.0" encoding="UTF-8"?>
<persons xmlns:i="http://www.w3.org/2001/XMLSchema-
instance"
  i:schemaLocation="http://localhost:8000/complex.xsd">
  <person firstname="“yourname" lastname="“lastname">
    <email>info@yourdomain.com</email>
  </person>
  <person>
    <email>john@email.com</email>
  </person>
</persons>
```

Schema for attribute without grouping



```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="persons">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="person" maxOccurs="unbounded">
                    <xsd:complexType mixed="true">
                        <xsd:sequence>
                            <xsd:element name="email" type="xsd:string" />
                        </xsd:sequence>
                         <xsd:attribute name="firstname" type="xsd:string" />
                        <xsd:attribute name="lastname" type="xsd:string" />
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Schema for attribute with grouping



```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:attributeGroup name="personAttributes">
        <xsd:attribute name="firstname" type="xsd:string" />
        <xsd:attribute name="lastname" type="xsd:string" />
    </xsd:attributeGroup>
    <xsd:element name="persons">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="person" maxOccurs="unbounded">
                    <xsd:complexType mixed="true">
                        <xsd:sequence>
                            <xsd:element name="email" type="xsd:string" />
                        </xsd:sequence>
                        <!-- Refering group -->
                        <xsd:attributeGroup ref="personAttributes" />
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

XSD Indicators

XSD indicators are used to control how elements are to be used in documents with indicators. There are seven indicators:

1. Order indicators: *They contain;*

All, Choice, Sequence

2. Occurrence indicators: They include;

maxOccurs, minOccurs

3. Group indicators: *They contain;*

Group name, attributeGroup name

Order Indicators

All Indicator

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

```
<xs:element name="person">
        <xs:complexType>
                <xs:all>
                        <xs:element name="firstname" type="xs:string"/>
                        <xs:element name="lastname" type="xs:string"/>
                </xs:all>
        </xs:complexType>
</xs:element>
```

Note: When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1

Choice Indicator

The <choice> indicator specifies that either one child element or another can occur:

```
<xs:element name="person">
        <xs:complexType>
                <xs:choice>
                        <xs:element name="employee" type="employee"/>
                        <xs:element name="member" type="member"/>
                </xs:choice>
        </xs:complexType>
</xs:element>
```

Sequence Indicator

The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<xs:element name="person">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="firstname" type="xs:string"/>
                        <xs:element name="lastname" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

Occurrence Indicators

Used to define how often an element can occur. Note: For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) the default value for maxOccurs and minOccurs is 1.

maxOccurs Indicator:

The **maxOccurs** indicator specifies the maximum number of times an element can occur:

```
<xs:element name="person">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="full_name" type="xs:string"/>
                        <xs:element name="child_name" type="xs:string"
maxOccurs="10"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

minOccurs Indicator:

The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<xs:element name="person">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="full_name" type="xs:string"/>
                        <xs:element name="child_name" type="xs:string" maxOccurs="10" minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of zero times and a maximum of ten times in the "person" element.

XML and XSD with Indicators

To allow an element to appear an **unlimited** number of times, use the maxOccurs="**unbounded**" statement:

Consider an example; An XML file called "Myfamily.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation="family.xsd">

                <person>

                                <full_name>Anjolina</full_name>

                                <child_name>Janet</child_name>

                </person>

                <person>

                                <full_name>Dhritrasta</full_name>

                                <child_name>Duryodhan</child_name>

                                <child_name>Dushasan</child_name>

                                <child_name>Kushashan</child_name>

                                <child_name>Sushasan</child_name>

                </person>

                <person>

                                <full_name>Bhisma pitamaha</full_name>

                </person>

</persons>
```

The XML file above contains a root element named "persons". Inside this root element we have defined three "person" elements. Each "person" element must contain a "full_name" element and it can contain up to five "child_name" elements.

Here is the schema file "family.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" element FormDefault="qualified">

        <xs:element name="persons">

                <xs:complexType>

                        <xs:sequence>

                                <xs:element name="person" maxOccurs="unbounded">

                                        <xs:complexType>

                                                <xs:sequence>

                                                        <xs:element name="full_name" type="xs:string"/>

                                                        <xs:element name="child_name" type="xs:string"
                                                        minOccurs="0" maxOccurs="5"/>

                                                </xs:sequence>

                                        </xs:complexType>

                                </xs:element>

                        </xs:sequence>

                </xs:complexType>

        </xs:element>

</xs:schema>
```

Group Indicators

- Group indicators are used to define related sets of elements.
- **Element Groups:** Element groups are defined with the group declaration, like this:

```
<xs:group name="groupname"> ... </xs:group>
```

Must define an all, choice, or sequence element inside the group declaration.

- The following example defines a group named "persongroup", that defines a group of elements that must occur in an exact sequence:

```
<xs:group name="persongroup">
    <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
        <xs:element name="birthday" type="xs:date"/>
    </xs:sequence>
</xs:group>
```

Attribute Groups: Attribute groups are defined with the attributeGroup declaration, like this:

```
<xs:attributeGroup name="groupname"> ... </xs:attributeGroup>
```

Example:

```
<xs:attributeGroup name="personattrgroup">
        <xs:attribute name="firstname" type="xs:string"/>
        <xs:attribute name="lastname" type="xs:string"/>
        <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

Facets

- Restrictions on XML elements.
- Restrictions are used to define acceptable values for XML elements or attributes.
- 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.

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 **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>
```

Pattern/Restrictions on **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>
```

Pattern/Restrictions on **Series of Values**

This example also defines 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>
```

Pattern/Restrictions on **Series of Values**

- The only acceptable value is THREE of the UPPERCASE letters from a to z:

```
<xs:pattern value="[A-Z][A-Z][A-Z]"/>
```

- The only acceptable value is ONE of the following letters: x, y, OR z:

```
<xs:pattern value="[xyz]"/>
```

- The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

```
<xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
```

Restrictions on **Whitespace Characters**

- To specify how whitespace characters should be handled, we would use the whiteSpace constraint.
- This example defines an element called "address" with a restriction. The whiteSpace constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters:

```
<xs:element name="address">

        <xs:simpleType>

                <xs:restriction base="xs:string">

                        <xs:whiteSpace value="preserve"/>

                </xs:restriction>

        </xs:simpleType>

</xs:element>
```

Restrictions on **Whitespace Characters**

- The whiteSpace constraint is set to "replace", which means that the XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces:

```
<xs:whiteSpace value="replace"/>
```

- The whiteSpace constraint is set to "collapse", which means that the XML processor WILL REMOVE all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space:

```
<xs:whiteSpace value="collapse"/>
```

Restrictions on **Length**

- To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints.
- This example defines an element called "password" with a restriction. The value must be exactly eight characters:

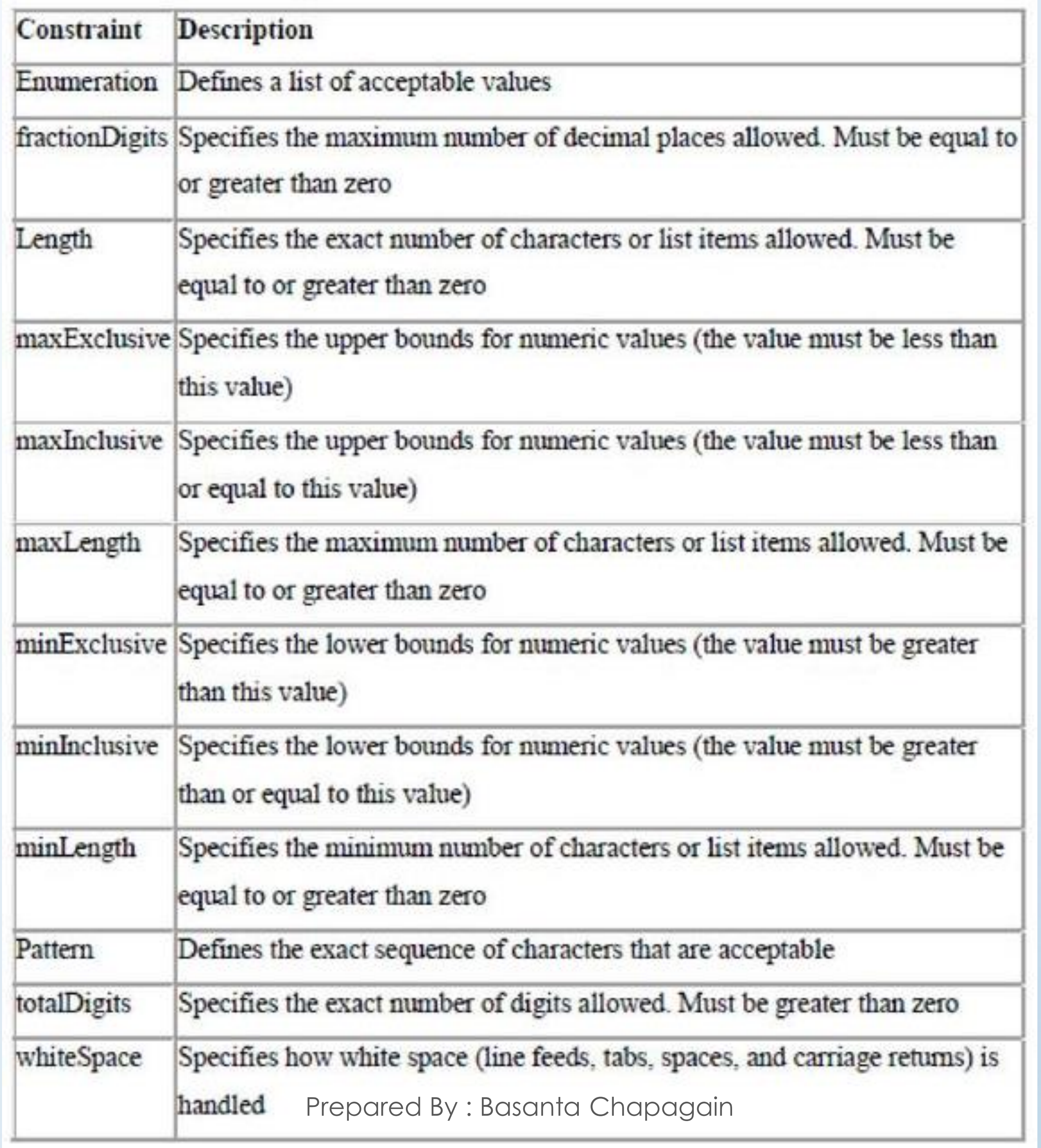
```
<xs:element name="password">
          <xs:simpleType>
                    <xs:restriction base="xs:string">
                              <xs:length value="8"/>
                    </xs:restriction>
          </xs:simpleType>
</xs:element>
```

Restrictions on **Length**

This example defines another element called "password" with a restriction. The value must be minimum five characters and maximum eight characters:

```
<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>
```

Restrictions on **Data Types**



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

Restrictions for **Data Types** (Constraints/Facets)

- enumeration
- fractionDigits
- length
- maxExclusive
- maxInclusive
- maxLength
- minExclusive
- minInclusive
- minLength
- pattern
- totalDigits
- whiteSpace

DTD vs. XML Schemas

XML Schemas are the successors of DTDs. In near future, XML Schemas will be used in most Web applications as a replacement for DTDs because of the following reasons;

- XML Schemas are extensible to future additions.
- XML Schemas are richer and more powerful than DTDs.
- XML Schemas are written in XML.
- XML Schemas support data types.
- XML Schemas support namespaces.

DTDs are better for text-intensive applications, while schemas have several advantages for data-intensive workflows.

Schemas are written in XML and thusly follow the same rules, while DTDs are written in a completely different language.

HTML	XML	DTD	XSD
Display data (Look and Feel)	Transport and Store the data	Document Type Definition	XML Schema Definition
Markup language itself	Provide framework to define markup languages	Doesn't support data-types	Supports data-types
Not case-sensitive	Case-sensitive	Doesn't support namespace	Supports namespace
Predefined tags	Can create own tags	Doesn't define order for child elements	Order can be defined for child elements
Static	Dynamic	Not extensible	Extensible
Example: `<!DOCTYPE html> <html lang="en"> <head> <title>Predefined tags</title> </head> <body> <p>Display data</p> </body> </html>`	**Example:** `<?xml version="1.0" encoding="UTF-8"?> <College> <Class> <Name>Transportable data</Name> </Class> </College>`	**Example:** Root `<!DOCTYPE College [ <!ELEMENT College (Name)> <!ELEMENT Name (#PCDATA)> ]>`	**Example: 'xs'** is namespace `<?xml version="1.0" encoding="UTF-8"?> <xs:element name="College"> <xs:complexType> <xs:sequence> <xs:element name="Name" type="xs:string"/> </xs:sequence> </xs:complexType> </xs:element>`

XSL Language

- XSL stands for E**X**tensible **S**tylesheet **L**anguage.
- The World Wide Web Consortium (W3C) started to develop XSL.
- XML does not use predefined tags (we can use any tag- names we like), and therefore the meaning of each tag is **not well understood**.
- XSL describes how the XML document should be displayed, however its more than a Style Sheet Language.

XSL consists of three parts:

1. XSLT - a language for transforming XML documents
2. XPath - a language for navigating in XML documents
3. XSL-FO - a language for formatting XML documents

XSLT (Extensible Stylesheet Language Transformations)

- Transforms an XML document into another XML document.
- Uses XPath to navigate in XML documents.
- Transforms each XML element into an (X)HTML element.
- Adding/removing elements and attributes to or from the output file can be done.
- Rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more can be done.

XSLT transformation process

- XSLT uses XPath to define parts of the source document that should match one or more predefined templates. (XPath is used to navigate through elements and attributes in XML documents.)
- When a match is found, XSLT will transform the matching part of the source document into the result document.

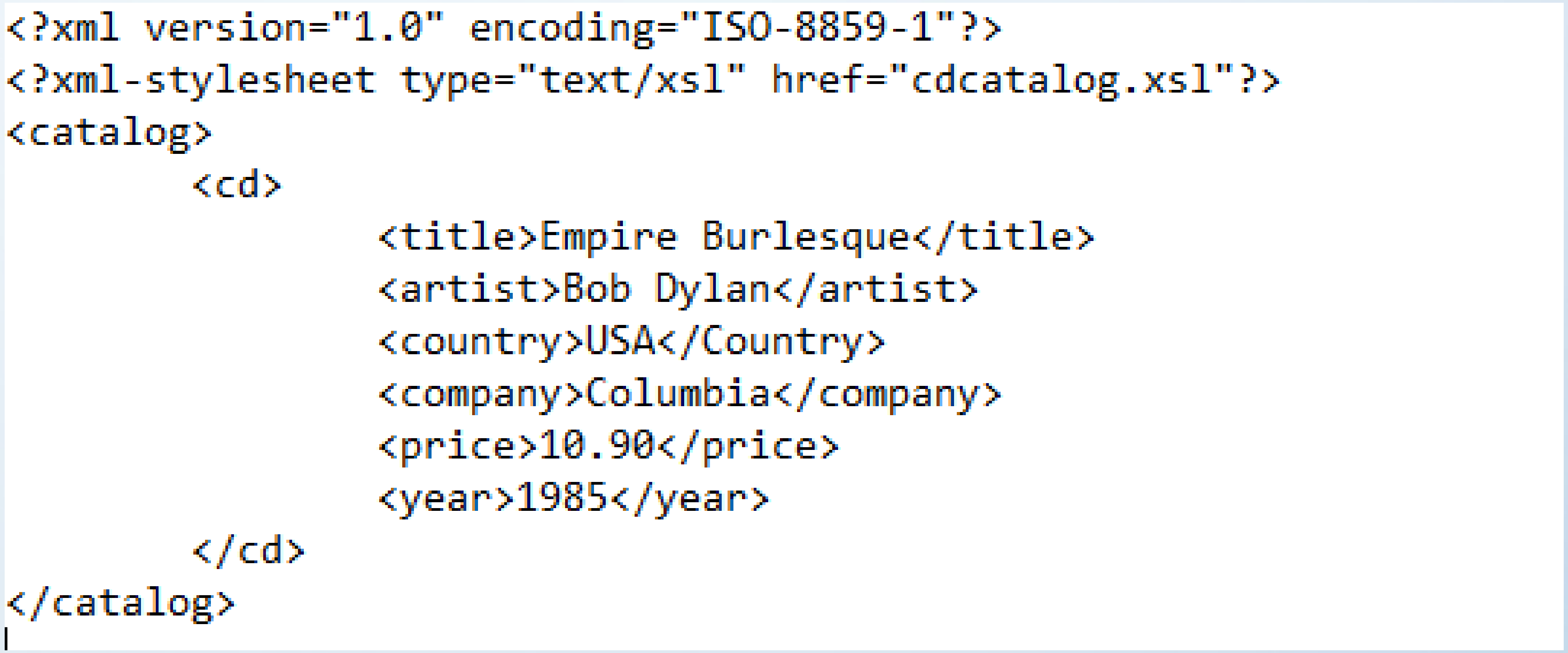
  XSLT transforms an XML source-tree into an XML result-tree.

- Root element that declares the document to be an XSL stylesheet is <xsl:stylesheet> or <xsl:transform>
- The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:
  <xsl:stylesheet version="1.0“ xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  or:
  <xsl:transform version="1.0“ xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.

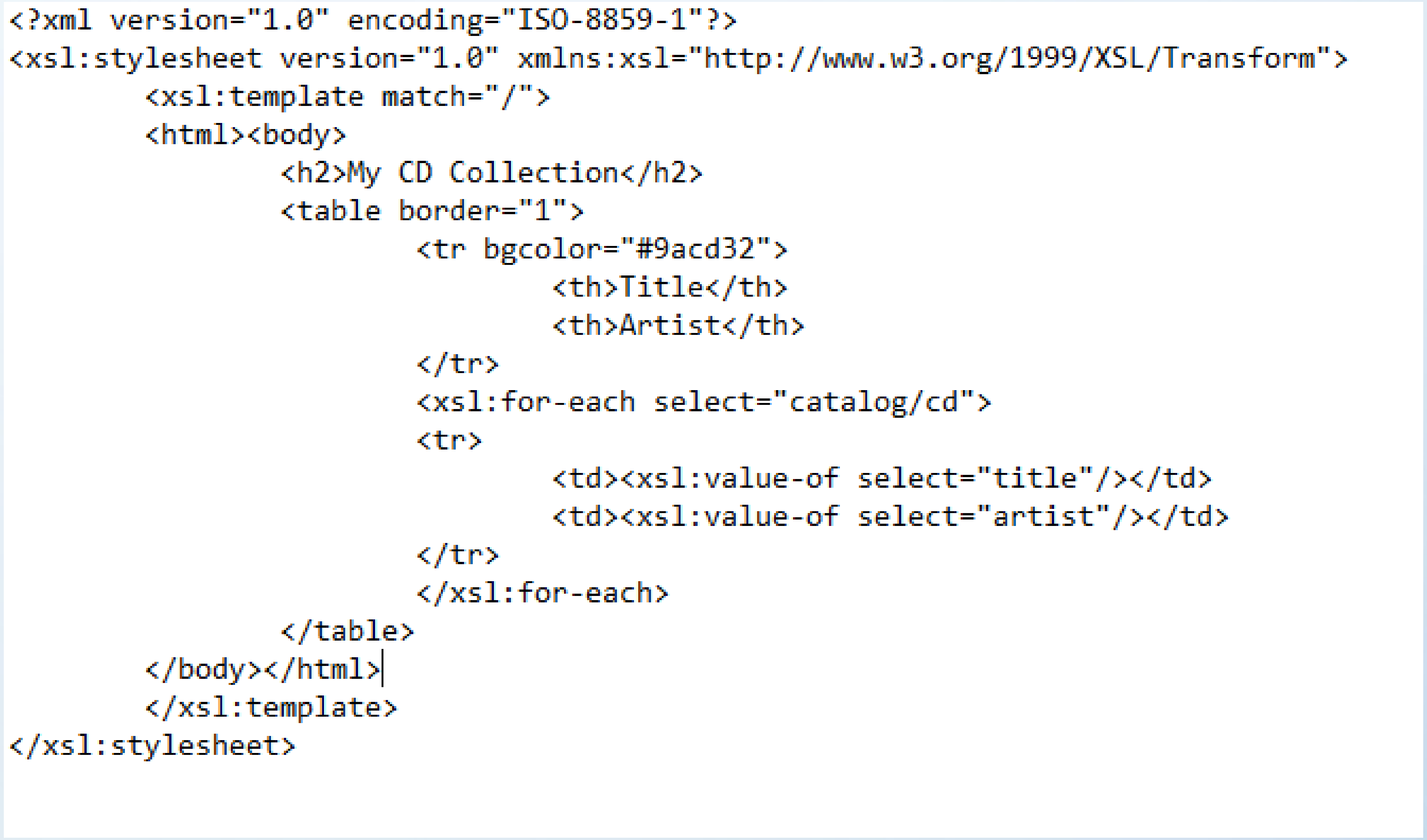
XSLT transformation process

Consider an example below, which we want to transform the following XML document ("**cdcatalog.xml**") into XHTML:



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.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>
</catalog>
```

We can create an XSL Style Sheet ("**cdcatalog.xsl**") with a transformation template:



```
<?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>
                        <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>
```

In example of **cdcatalog.xsl**, it can be explained as;

- Since an XSL stylesheet is an XML document, it always begins with the XML declaration:
  <?xml version="1.0" encoding="ISO-8859-1"?>
  or,
  <?xml version="1.0" encoding=“UTF-8"?>
- The next element, <xsl:stylesheet>, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).
- The <xsl:template> element defines a template. The match="/" attribute associates the template with the root of the XML source document.
- The content inside the <xsl:template> element defines some HTML to write to the output. The last two lines define the end of the template and the end of the style sheet.

In example of **cdcatalog.xsl**, it can be explained as;

- The **<xsl:value-of>** Element

  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.

  In the example cdcatalog.xsl, we have used it as;

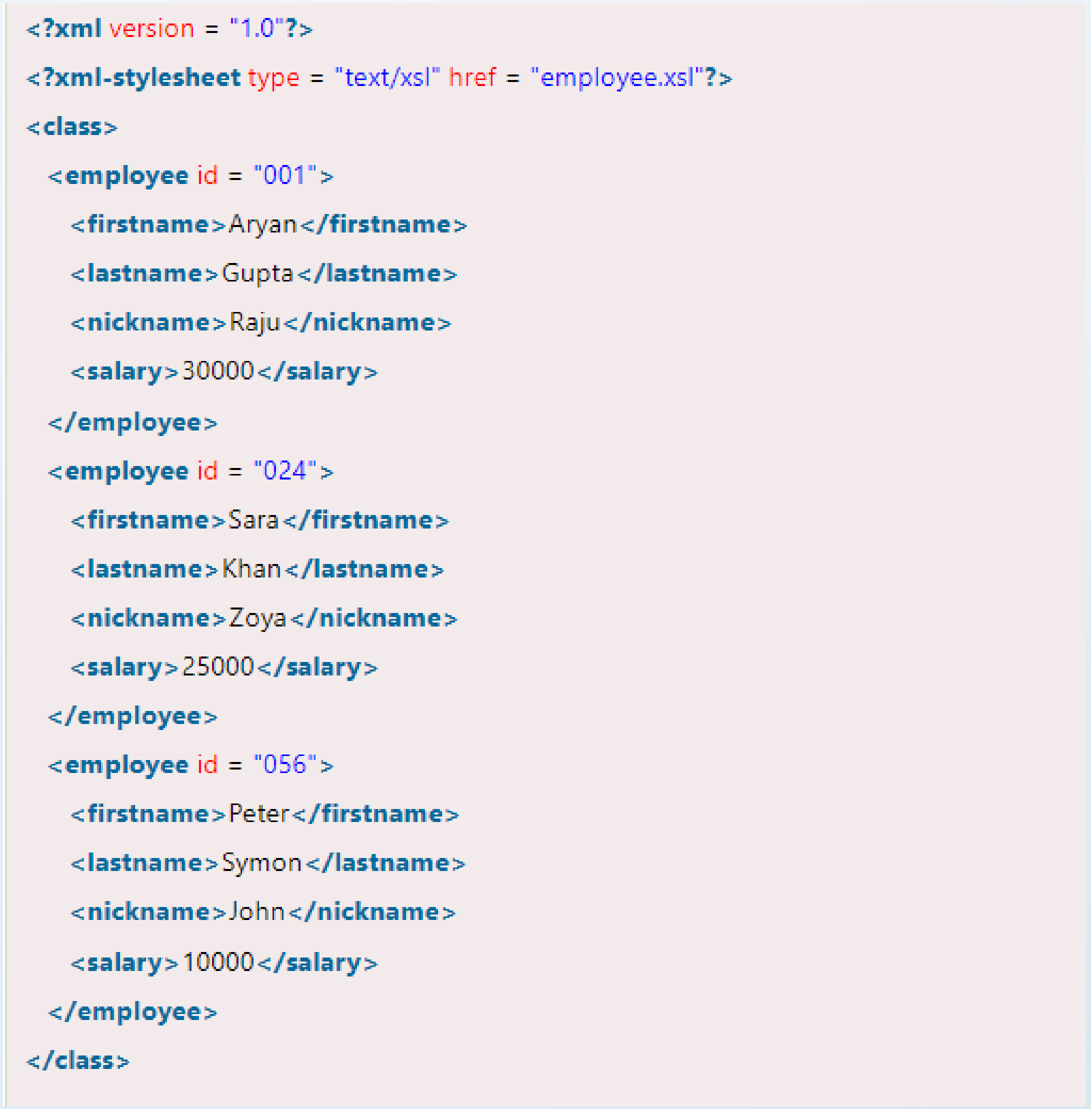
  ```
  <xsl:value-of select="catalog/cd/title"/>
  <xsl:value-of select="catalog/cd/artist"/>
  ```

Syntax:

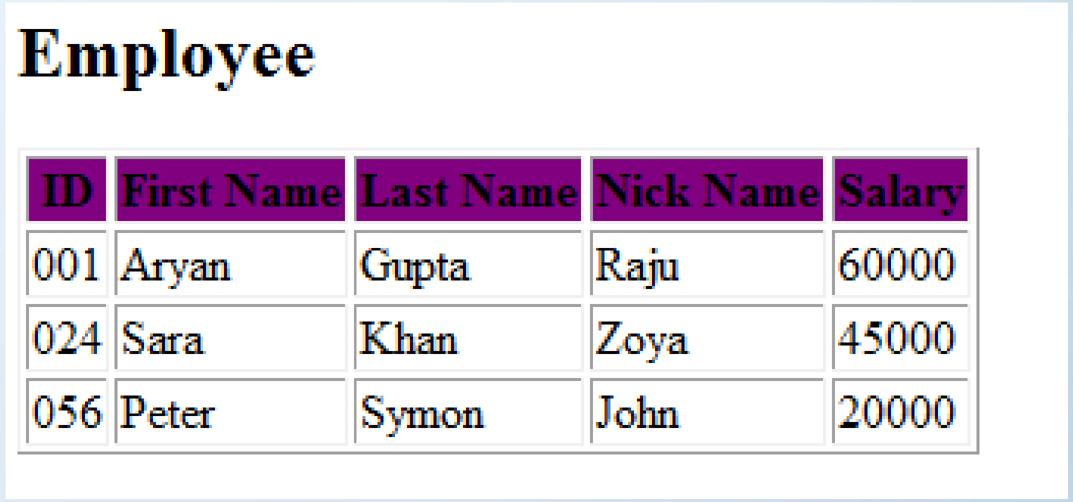
```
<xsl:value-of
   select = Expression
   disable-output-escaping = "yes" | "no">
</xsl:value-of>
```

Index	Name	Description
1)	**select**	**It specifies the XPpath expression to be evaluated in current context.**
2)	**disable-outputescaping**	**Default-"no". If "yes", output text will not escape XML characters from text.**

XSLT <xsl:value-of> Element Example



```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "employee.xsl"?>
<class>
  <employee id = "001">
    <firstname>Aryan</firstname>
    <lastname>Gupta</lastname>
    <nickname>Raju</nickname>
    <salary>30000</salary>
  </employee>
  <employee id = "024">
    <firstname>Sara</firstname>
    <lastname>Khan</lastname>
    <nickname>Zoya</nickname>
    <salary>25000</salary>
  </employee>
  <employee id = "056">
    <firstname>Peter</firstname>
    <lastname>Symon</lastname>
    <nickname>John</nickname>
    <salary>10000</salary>
  </employee>
</class>
```



Employee

ID	First Name	Last Name	Nick Name	Salary
001	Aryan	Gupta	Raju	60000
024	Sara	Khan	Zoya	45000
056	Peter	Symon	John	20000

XSLT <xsl:value-of> Element Example

```
<?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>Employee</h2>
        <table border = "1">
          <tr bgcolor = "purple">
            <th>ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Nick Name</th>
            <th>Salary</th>
          </tr>
          <!-- for-each processing instruction
          Looks for each element matching the XPath expression
          -->
          <xsl:for-each select="class/employee">
            <tr>
              <td>
                <!-- value-of processing instruction
                process the value of the element matching the XPath expression
                -->
                <xsl:value-of select = "@id"/>
              </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 = "salary"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

XSLT <xsl:template> element:

- An XSL stylesheet consists of one or more set of rules that are called templates.
- A template contains rules to apply when a specified node is matched.
- <xsl:template> element is used to build templates.
- The match attribute is used to associate a template with an XML element.
- match attribute is an XPath expression (i.e. match="/" defines the whole document).

XSLT <xsl:for-each> element:

- Allows to do looping in XSLT.
- It can be used to select every XML element of a specified node-set.

```
<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>
```

- We can filter the output from the XML file by adding a criteria on to the select attribute
  in the <xsl:for-each> element.

<xsl:for-each select="catalog/cd[artist = 'Bob Dylan']">

Legal filter operators are:

=	(equal)
!=	(not equal)
&lt;	less than
&gt;	greater than

- In the example of **cdcatalog.xsl**; we can **use restriction** as;

```
<xsl:for-each select="catalog/cd[artist='Bob Dylan']">
<tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
```

XSL <xsl:sort> element:

- The <xsl:sort> element is used to sort the output.
- To sort the output, simply add an <xsl:sort> element inside the <xsl:for-each> element in the XSL file.

```
<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>
```

XSL <xsl:if> element:

- It is used to put a conditional test against the content of the XML file.
- To add a conditional test, add the <xsl:if> element inside the <xsl:for-each> element in the XSL file.
- **Example:**

```
        <xsl:for-each select="catalog/cd">
                <xsl:if test="price &gt; 10">
                <tr>
                        <td><xsl:value-of select="title"/></td>
                        <td><xsl:value-of select="artist"/></td>
                </tr>
                </xsl:if>
        </xsl:for-each>
```

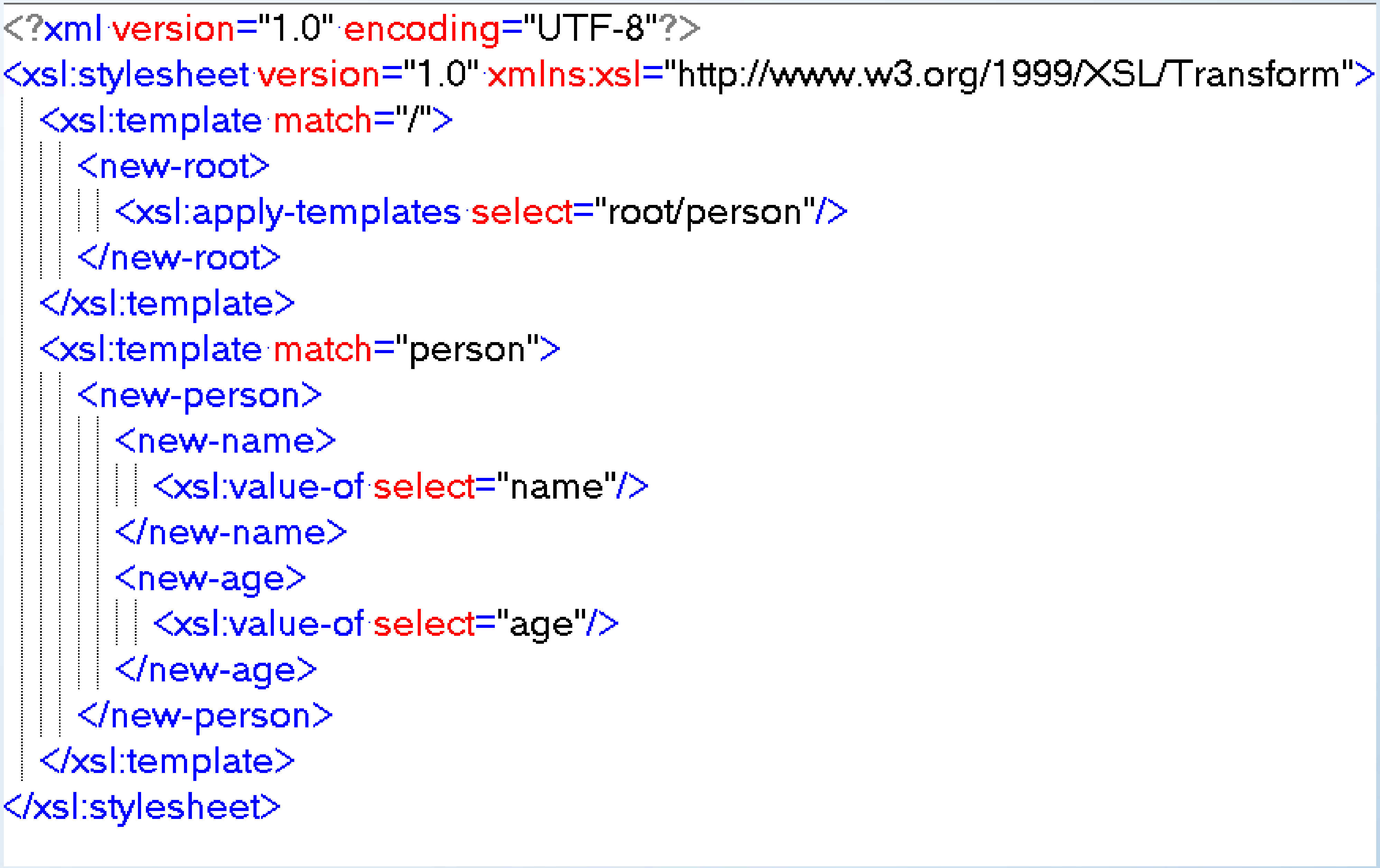
XSL <xsl:choose> element:

- The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

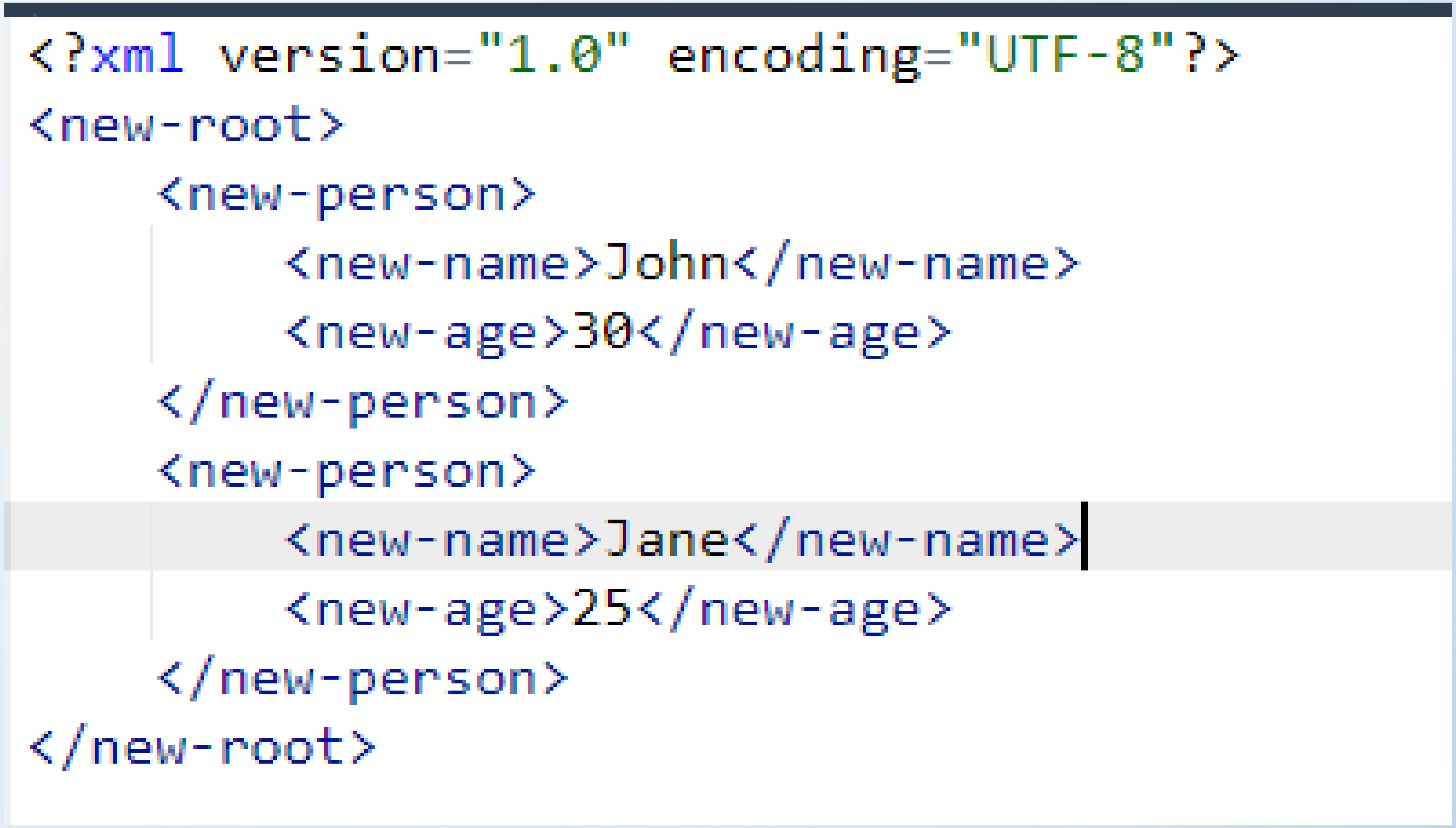
```
<xsl:for-each select="catalog/cd">
        <tr>
                <td><xsl:value-of select="title"/></td>
                <xsl:choose>
                        <xsl:when test="price &gt; 10">
                                <td bgcolor="#ff00ff"> <xsl:value-of select="artist"/></td>
                        </xsl:when>
                        <xsl:when test="price &gt; 9">
                                <td bgcolor="#cccccc"> <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>
```

//program to transform xml document to xml document

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="newtestxml.xsl"?>
<root>
  <person>
    <name>John</name>
    <age>30</age>
  </person>
  <person>
    <name>Jane</name>
    <age>25</age>
  </person>
</root>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <new-root>
      <xsl:apply-templates select="root/person"/>
    </new-root>
  </xsl:template>
  <xsl:template match="person">
    <new-person>
      <new-name>
        <xsl:value-of select="name"/>
      </new-name>
      <new-age>
        <xsl:value-of select="age"/>
      </new-age>
    </new-person>
  </xsl:template>
</xsl:stylesheet>
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<new-root>
    <new-person>
        <new-name>John</new-name>
        <new-age>30</new-age>
    </new-person>
    <new-person>
        <new-name>Jane</new-name>
        <new-age>25</new-age>
    </new-person>
</new-root>
```

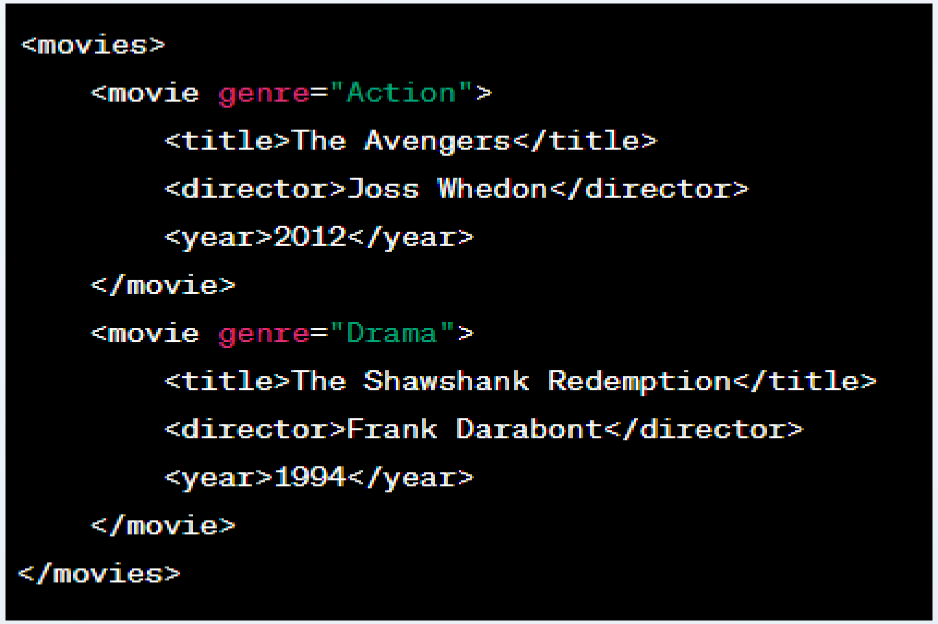
Another Example : XSLT

```xml
<library>
    <book>
        <title>Harry Potter and the Sorcerer's Stone</title>
        <author>J.K. Rowling</author>
        <genre>Fantasy</genre>
    </book>
    <book>
        <title>To Kill a Mockingbird</title>
        <author>Harper Lee</author>
        <genre>Drama</genre>
    </book>
</library>
```

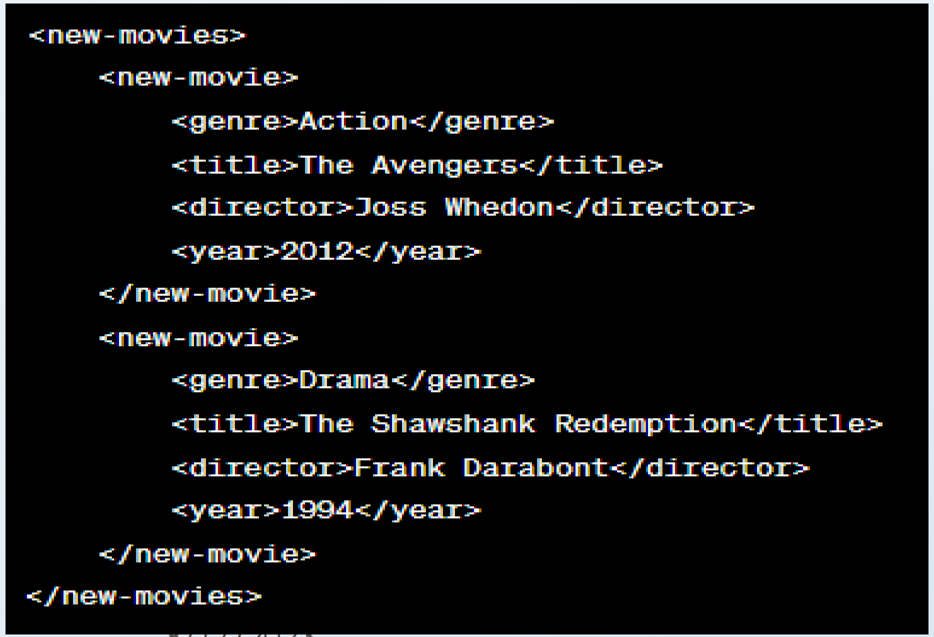
```xml
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Trans
    <xsl:template match="/">
        <new-library>
            <xsl:apply-templates select="library/book"/>
        </new-library>
    </xsl:template>

    <xsl:template match="book">
        <new-book>
            <title>
                <xsl:value-of select="title"/>
            </title>
            <author>
                <xsl:value-of select="author"/>
            </author>
        </new-book>
    </xsl:template>
</xsl:stylesheet>
```

```xml
<new-library>
    <new-book>
        <title>Harry Potter and the Sorcerer's Stone</title>
        <author>J.K. Rowling</author>
    </new-book>
    <new-book>
        <title>To Kill a Mockingbird</title>
        <author>Harper Lee</author>
    </new-book>
</new-library>
```

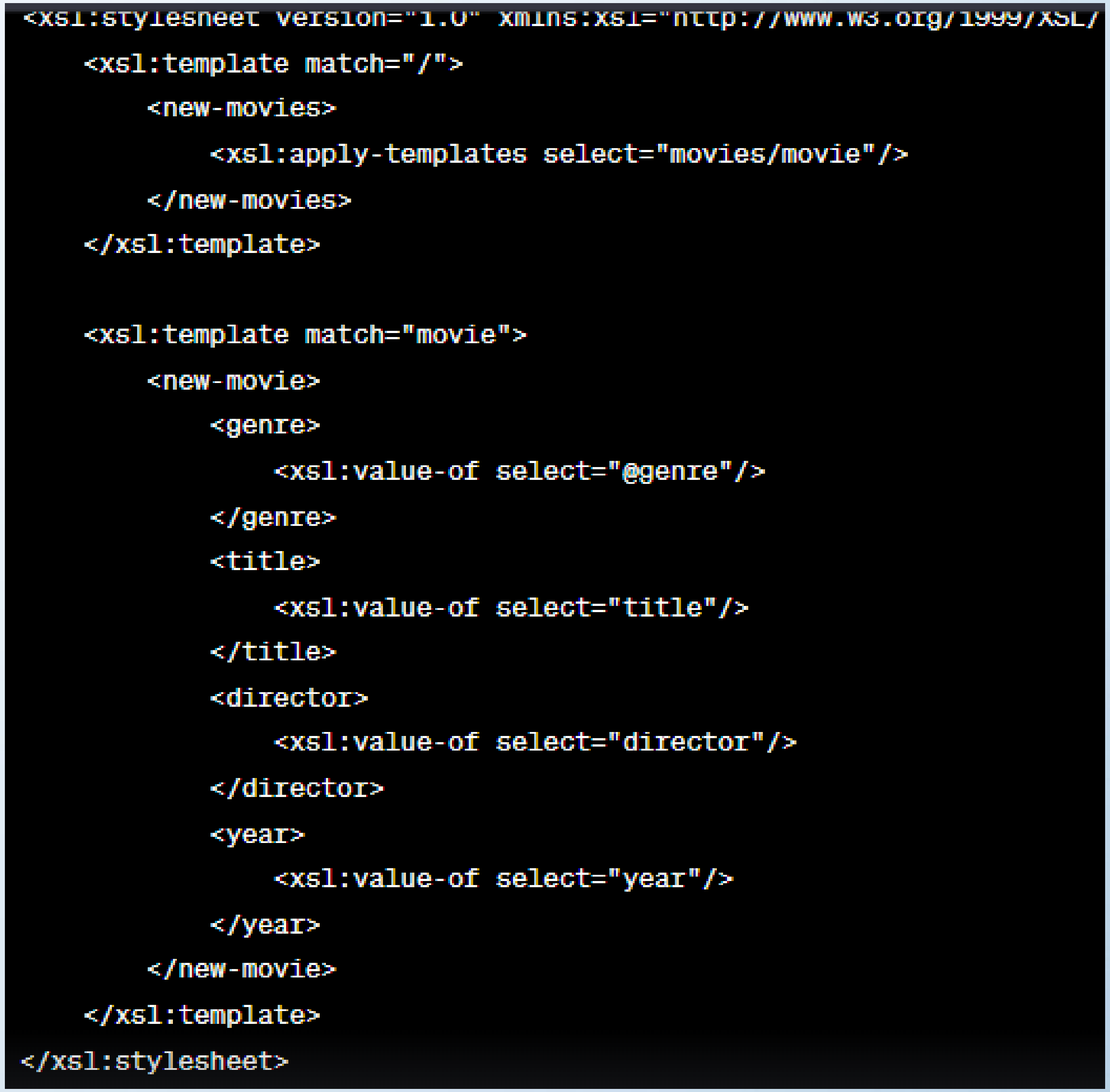
Another Example : XSLT



```
<movies>
    <movie genre="Action">
        <title>The Avengers</title>
        <director>Joss Whedon</director>
        <year>2012</year>
    </movie>
    <movie genre="Drama">
        <title>The Shawshank Redemption</title>
        <director>Frank Darabont</director>
        <year>1994</year>
    </movie>
</movies>
```

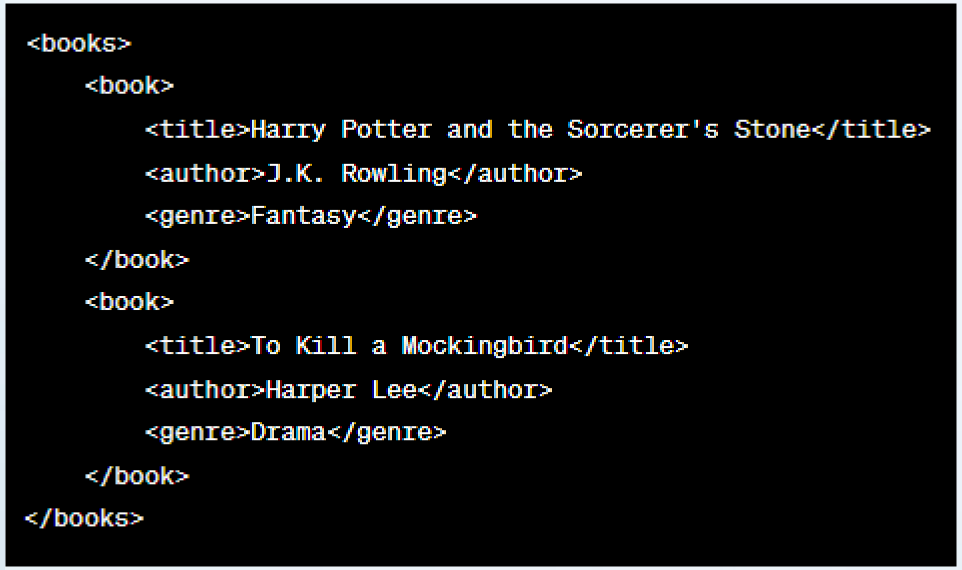


```
<new-movies>
    <new-movie>
        <genre>Action</genre>
        <title>The Avengers</title>
        <director>Joss Whedon</director>
        <year>2012</year>
    </new-movie>
    <new-movie>
        <genre>Drama</genre>
        <title>The Shawshank Redemption</title>
        <director>Frank Darabont</director>
        <year>1994</year>
    </new-movie>
</new-movies>
```

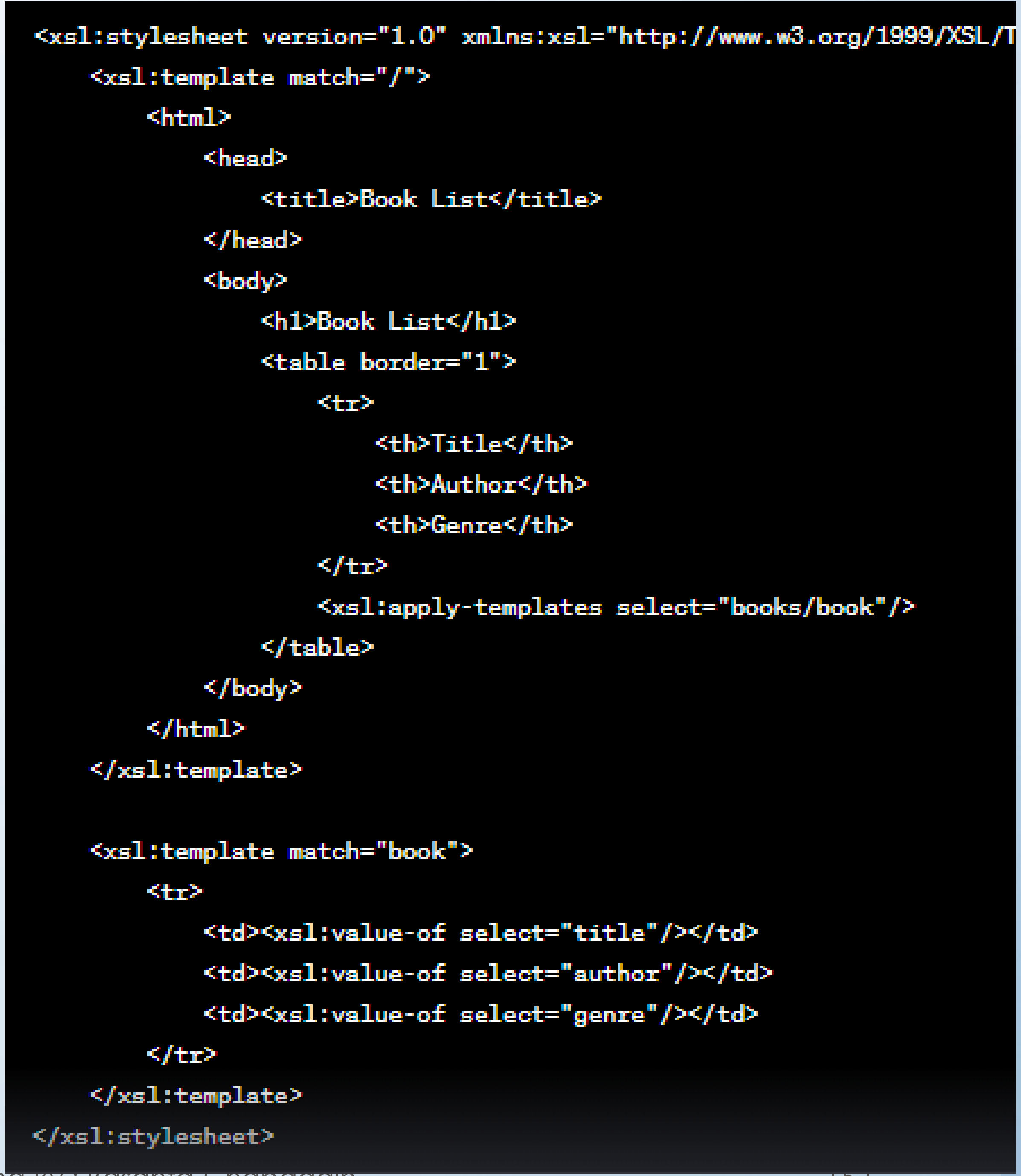


```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    <xsl:template match="/">
        <new-movies>
            <xsl:apply-templates select="movies/movie"/>
        </new-movies>
    </xsl:template>

    <xsl:template match="movie">
        <new-movie>
            <genre>
                <xsl:value-of select="@genre"/>
            </genre>
            <title>
                <xsl:value-of select="title"/>
            </title>
            <director>
                <xsl:value-of select="director"/>
            </director>
            <year>
                <xsl:value-of select="year"/>
            </year>
        </new-movie>
    </xsl:template>
</xsl:stylesheet>
```

Another Example : XSLT

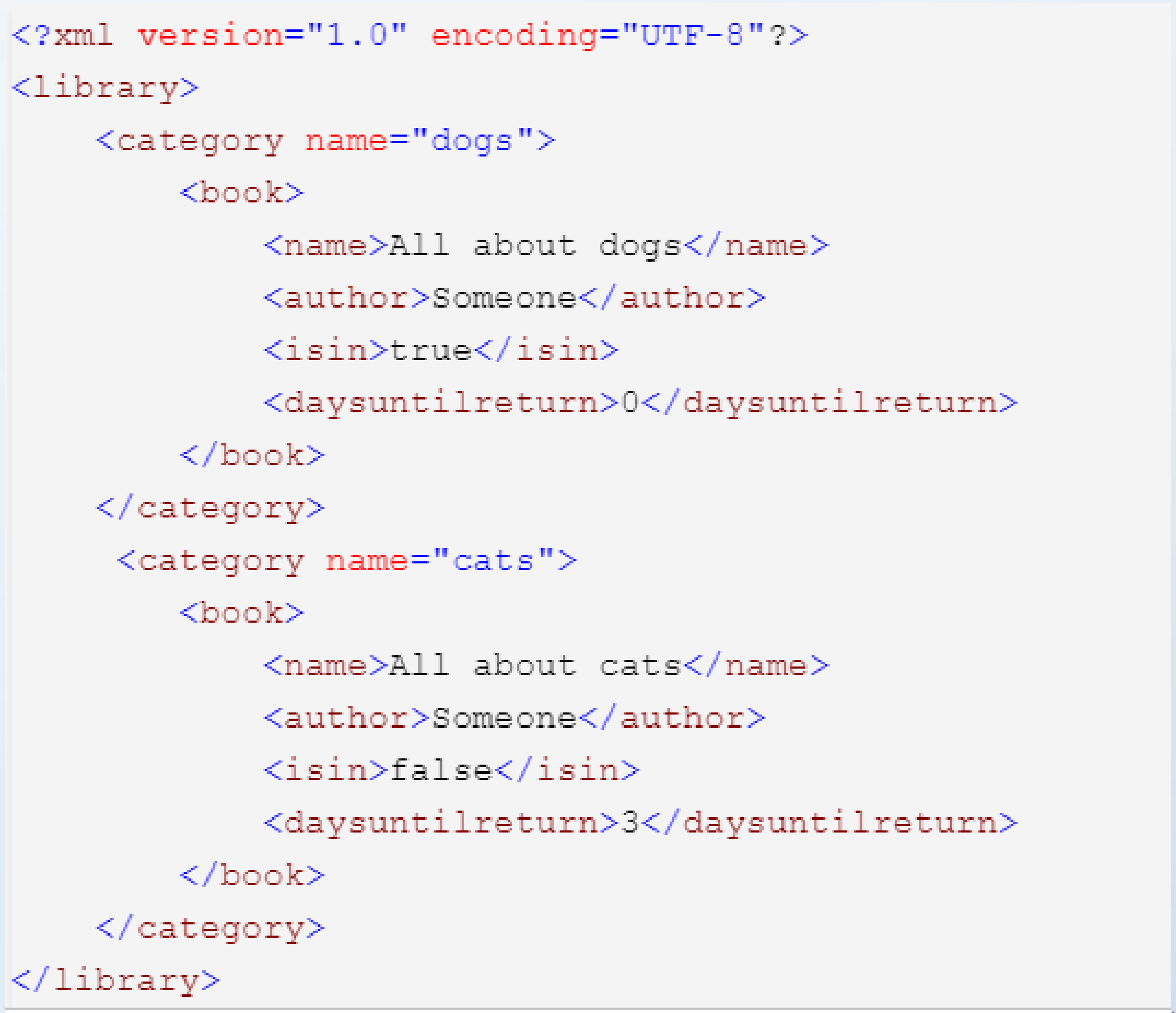


```
<books>
    <book>
        <title>Harry Potter and the Sorcerer's Stone</title>
        <author>J.K. Rowling</author>
        <genre>Fantasy</genre>
    </book>
    <book>
        <title>To Kill a Mockingbird</title>
        <author>Harper Lee</author>
        <genre>Drama</genre>
    </book>
</books>
```



```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/T
    <xsl:template match="/">
        <html>
            <head>
                <title>Book List</title>
            </head>
            <body>
                <h1>Book List</h1>
                <table border="1">
                    <tr>
                        <th>Title</th>
                        <th>Author</th>
                        <th>Genre</th>
                    </tr>
                    <xsl:apply-templates select="books/book"/>
                </table>
            </body>
        </html>
    </xsl:template>

    <xsl:template match="book">
        <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="author"/></td>
            <td><xsl:value-of select="genre"/></td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

Examples : Fictional library as an input XML.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <category name="dogs">
        <book>
            <name>All about dogs</name>
            <author>Someone</author>
            <isin>true</isin>
            <daysuntilreturn>0</daysuntilreturn>
        </book>
    </category>
     <category name="cats">
        <book>
            <name>All about cats</name>
            <author>Someone</author>
            <isin>false</isin>
            <daysuntilreturn>3</daysuntilreturn>
        </book>
    </category>
</library>
```

XPath

- The **XML Path Language**, is a query language for selecting nodes from an XML document.
- Language for addressing parts of an XML document, designed to be used by both XSLT and XPointer.
- XPath uses path expressions to navigate in XML documents.
- It also contains a library of standard functions for string values, numeric values, date and time comparison etc.
- XML documents are treated as trees of nodes. The topmost element of the tree is called the **root** element.
- XPath uses path expressions to select nodes or node-sets in an XML document.

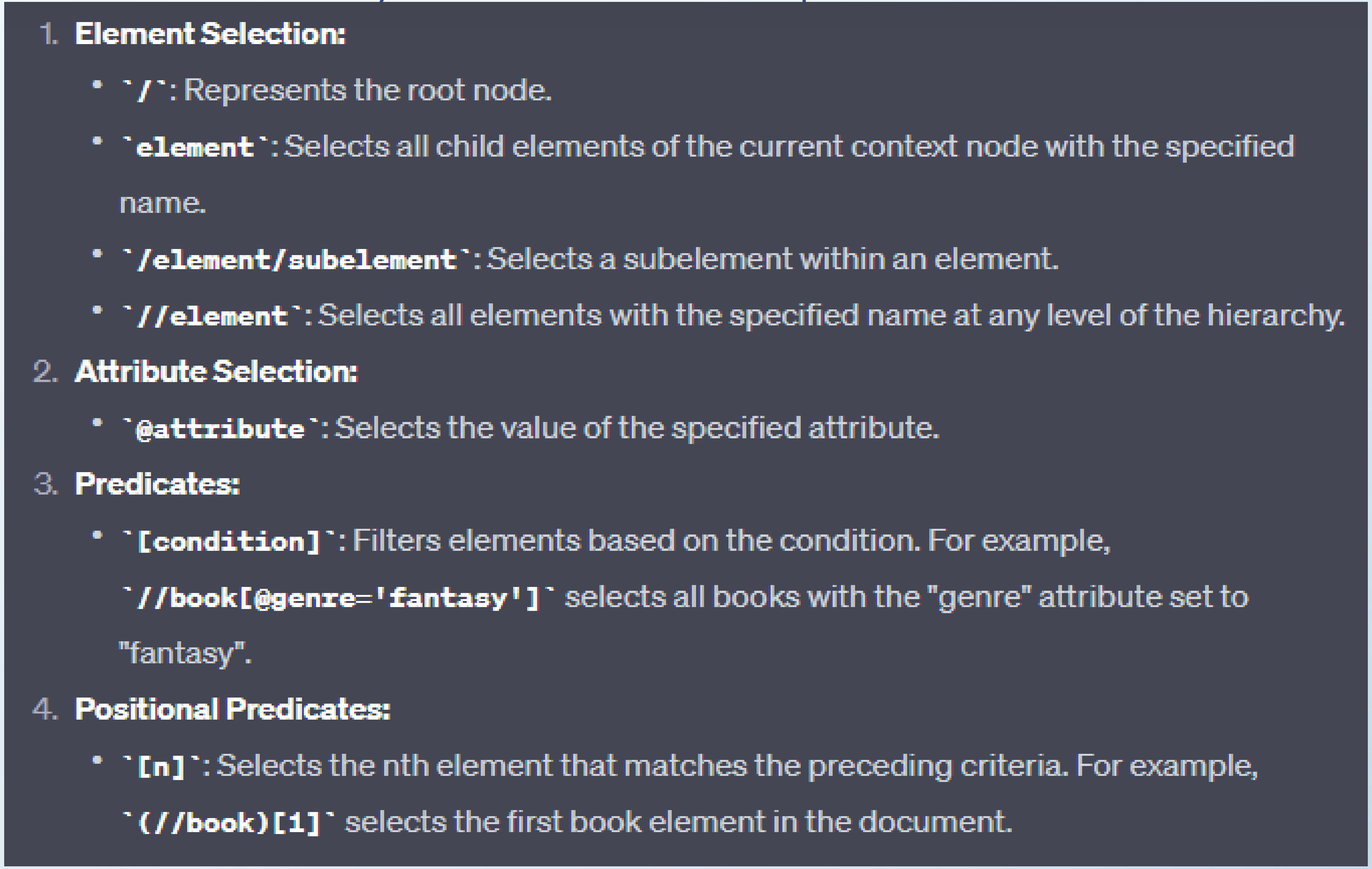
XPath

- The most useful path expressions are listed below:

Expression	**Description**
nodename	Selects all child nodes of the named node
/	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

XPath

- Some commonly used XPath components:

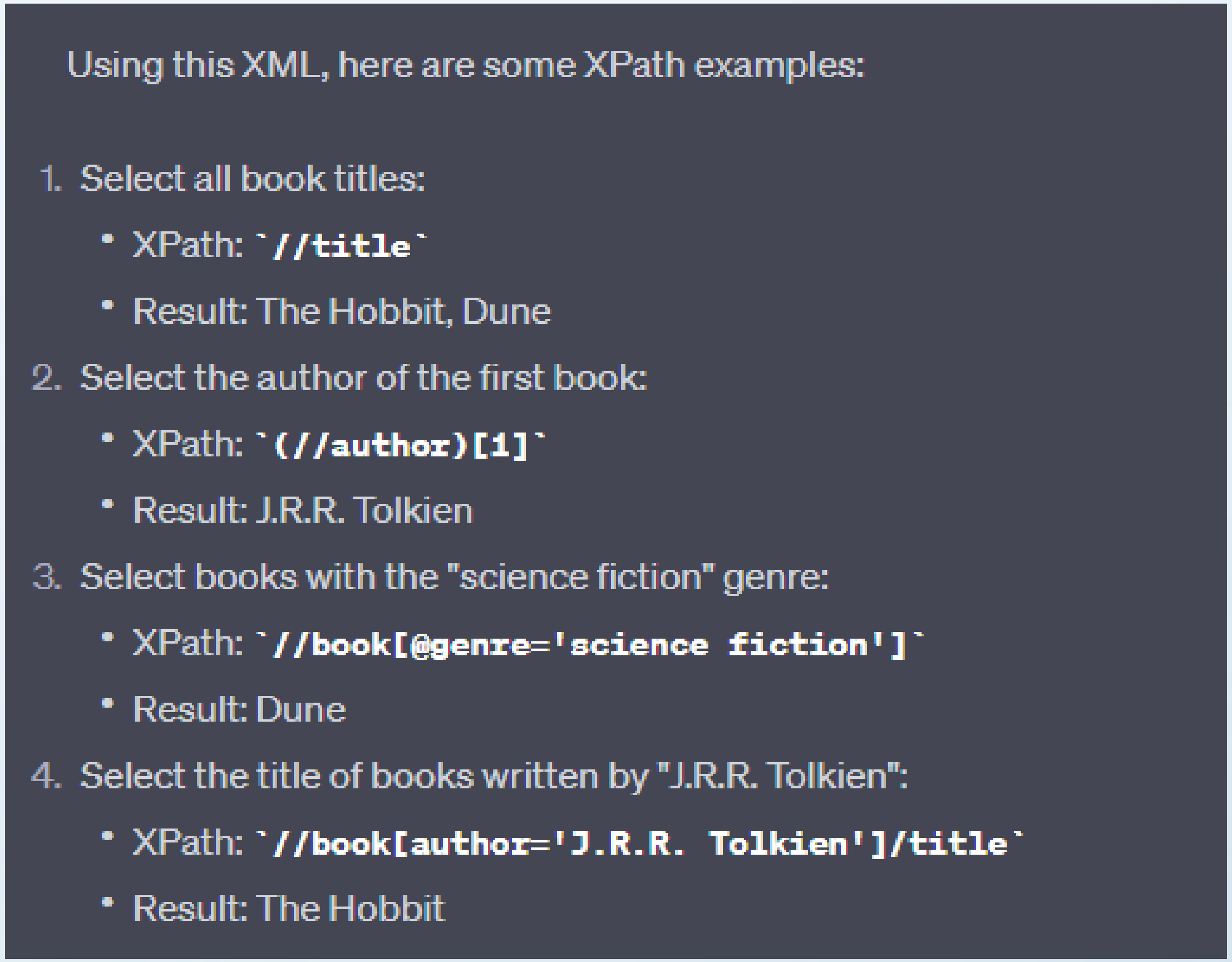


1. **Element Selection:**
   - `/`: Represents the root node.
   - `element`: Selects all child elements of the current context node with the specified name.
   - `/element/subelement`: Selects a subelement within an element.
   - `//element`: Selects all elements with the specified name at any level of the hierarchy.
2. **Attribute Selection:**
   - `@attribute`: Selects the value of the specified attribute.
3. **Predicates:**
   - `[condition]`: Filters elements based on the condition. For example, `//book[@genre='fantasy']` selects all books with the "genre" attribute set to "fantasy".
4. **Positional Predicates:**
   - `[n]`: Selects the nth element that matches the preceding criteria. For example, `(//book)[1]` selects the first book element in the document.

Xpath Example

```
<library>
  <book genre="fantasy">
    <title>The Hobbit</title>
    <author>J.R.R. Tolkien</author>
  </book>
  <book genre="science fiction">
    <title>Dune</title>
    <author>Frank Herbert</author>
  </book>
</library>
```

XPath



Using this XML, here are some XPath examples:

1. Select all book titles:
   - XPath: `//title`
   - Result: The Hobbit, Dune
2. Select the author of the first book:
   - XPath: `(//author)[1]`
   - Result: J.R.R. Tolkien
3. Select books with the "science fiction" genre:
   - XPath: `//book[@genre='science fiction']`
   - Result: Dune
4. Select the title of books written by "J.R.R. Tolkien":
   - XPath: `//book[author='J.R.R. Tolkien']/title`
   - Result: The Hobbit

Xpath Example

```
<movies>
 <movie id="1">
  <title>The Shawshank Redemption</title>
  <director>Frank Darabont</director>
  <genre>Drama</genre>
 </movie>
 <movie id="2">
  <title>Inception</title>
  <director>Christopher Nolan</director>
  <genre>Science Fiction</genre>
 </movie>
 <movie id="3">
  <title>Avatar</title>
  <director>James Cameron</director>
  <genre>Adventure</genre>
 </movie>
</movies>
```

Xpath Example

1. Select the titles of all movies:
   - XPath: //title
   - Result: The Shawshank Redemption, Inception, Avatar
2. Select the director of the movie with id="2":
   - XPath: //movie[@id='2']/director
   - Result: Christopher Nolan
3. Select the titles of all movies with genre "Science Fiction":
   - XPath: //movie[genre='Science Fiction']/title
   - Result: Inception
4. Select the titles of movies directed by "James Cameron":
   - XPath: //movie[director='James Cameron']/title
   - Result: Avatar
5. Select the title of the first movie:
   - XPath: (//title)[1]
   - Result: The Shawshank Redemption
6. Select the titles of the first two movies:
   - XPath: (//title)[position() <= 2]
   - Result: The Shawshank Redemption, Inception
7. Select the title of the last movie:
   - XPath: (//title)[last()]
   - Result: Avatar
8. Select the title of the second-to-last movie:
   - XPath: (//title)[last()-1]
   - Result: Inception

XQuery

- XQuery is to XML what SQL is to databases.
- XQuery is designed to query XML data.
- XQuery is built on XPath expressions
- XQuery is supported by all major databases XQuery is a W3C Recommendation

Example:

```
for $x in doc("books.xml")/bookstore/book
          where $x/price>30
                    order by $x/title
          return $x/title
```

XQuery

- XQuery is a query and functional programming language designed for querying and transforming XML data.
- It allows you to retrieve, manipulate, and combine XML documents and data from different sources.
- XQuery is similar to XPath in terms of selecting XML nodes, but it goes beyond simple selection and enables more complex operations on the data.
- XQuery is a powerful language for working with XML data, and it's especially useful when you need to perform more advanced querying and transformation tasks compared to what XPath offers.
- It's used in scenarios like database querying, data integration, and content transformations.

XQuery

- XQuery is a language for finding and extracting elements and attributes from XML documents.

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

XQuery

```
<library>
 <book>
  <title>The Hobbit</title>
  <author>J.R.R. Tolkien</author>
  <genre>Fantasy</genre>
 </book>
 <book>
  <title>Dune</title>
  <author>Frank Herbert</author>
  <genre>Science Fiction</genre>
 </book>
</library>
```

XQuery

Basic Selection:

- You can use XQuery to select nodes similar to XPath. For instance, selecting all book titles:

```
xquery

for $book in //book

return $book/title
```

- This XQuery expression retrieves the titles of all books in the library.

XQuery

Filtering:

- XQuery allows you to filter nodes based on conditions. For example, selecting titles of fantasy books:

xquery

```
for $book in //book

where $book/genre = "Fantasy"

return $book/title
```

- This XQuery expression retrieves the titles of books that have the genre "Fantasy".

XQuery

Transformation:

XQuery enables you to transform data. For instance, creating a list of book details:

```
<bookList>
{
  for $book in //book
  return
  <book>
    {$book/title}
    <author>{$book/author}</author>
  </book>
}
</bookList>
```

This XQuery expression creates a new XML structure containing book titles and authors.

XQuery

Aggregation:

- You can aggregate data using XQuery. Let's find the total number of books:

```
xquery

let $totalBooks := count(//book)

return <result>Total books: {$totalBooks}</result>
```

- This XQuery expression counts the number of book elements and displays the result.

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)

SAX (Simple API for XML)

- Allows a programmer to interpret a Web file that uses the XML
- SAX is an alternative to using the Document Object Model (DOM) to interpret the XML file.
- appropriate where many or very large files are to be processed, but it contains fewer capabilities for manipulating the data content.
- SAX is an event-driven interface. The programmer specifies an event that may happen and, if it does, SAX gets control and handles the situation. SAX works directly with an XML parser.
- SAX was developed collaboratively by members of the XML- DEV mailing list (currently hosted by OASIS).
- Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

SAX (Simple API for XML)

XML processing with SAX

- A Parser that implements SAX (i.e., a SAX Parser) functions as a stream parser, with an event-driven API.
- The user defines a number of callback
- methods that will be called when events occur during parsing.
- The SAX events include (among others):

  - XML Text nodes

  - XML Element Starts and Ends

  - XML processing instructions - XML Comments
- SAX interface does not deal in elements, but in events that largely correspond to tags.
- SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again.

- **Example:**

```
<?xml version="1.0" encoding="UTF-8"?>
<DocumentElement param="value">
        <FirstElement>
                &#xb6; Some Text
        </FirstElement>
        <?some_pi some_attr="some_value"?>
        <SecondElement param2="something">
                Pre-Text <Inline>Inline text</Inline> Post-text.
        </SecondElement>
</DocumentElement>
```

This **XML document**, when **passed through a SAX parser**,

will **generate a sequence of events** like the following:

- *XML Element start, named DocumentElement, with an* attribute param equal to "value"
- *XML Element start, named FirstElement*
- XML Text node, with data equal to "&#xb6; Some Text" (note: certain white spaces can be changed)
- *XML Element end, named FirstElement*
- Processing Instruction event, with the target some_pi and *data some_attr="some_value"* *(the content after the target is just text; however, it is very common to imitate the syntax of XML attributes, as in this example)*

This **XML document**, when **passed through a SAX parser**,

will **generate a sequence of events** like the following:

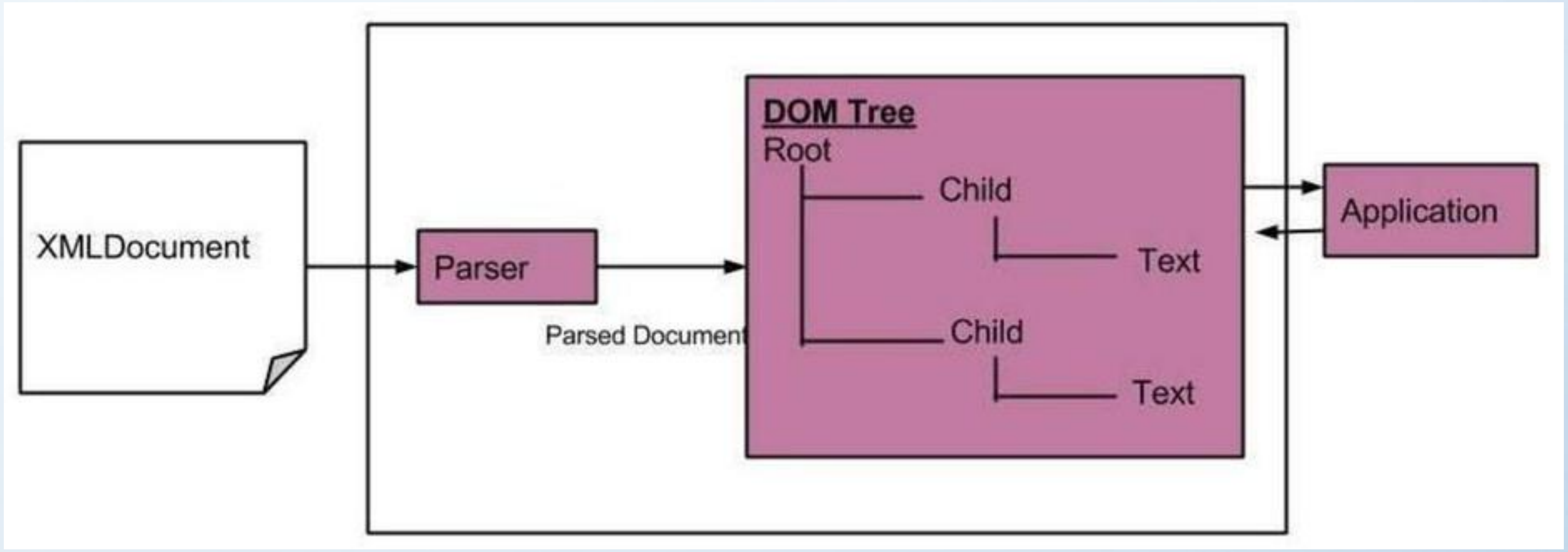
- *XML Element start, named SecondElement, with an* attribute param2 equal to "something"
- XML Text node, with data equal to "Pre-Text"
- XML Element start, named Inline
- XML Text node, with data equal to "Inlined text"
- XML Element end, named Inline
- XML Text node, with data equal to "Post-text.'
- *XML Element end, named SecondElement*
- *XML Element end, named DocumentElement*

DOM (Document Object Model)

- It is a W3C standard.
- Defines a standard for accessing documents like HTML and XML.
- Definition of DOM as put by the W3C is:

  "The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.“
- Defines the objects and properties and methods (interface) to access all XML elements.
- XML documents have a hierarchy of informational units called nodes; DOM is a standard programming interface of describing those nodes and the relationships between them.
- Provides an API that allows a developer to add, edit, move or remove nodes at any point on the tree in order to create an application.

Below is the diagram for the DOM structure which depicts that parser evaluates an XML document as a DOM structure by traversing through each nodes.



Sample DOM Parse

```
<html>

<body>

  <p id="demo"></p>

  <script>

  var text, parser, xmlDoc;

  text = "<bookstore><book>"+

            "<title>Everyday Nepal</title>"+

            "<author>yourname</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>
```

XML HTTP Requests (Considering DOM)

cd_catalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
 <TITLE>Greatest Hits</TITLE>
 <ARTIST>Dolly Parton</ARTIST>
 <COUNTRY>USA</COUNTRY>
 <COMPANY>RCA</COMPANY>
 <PRICE>9.90</PRICE>
 <YEAR>1982</YEAR>
</CD>
<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>
```

XML HTTP Requests (Considering DOM)

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>XML HTTP Request</title>
 </head>
 <body>
  <!-- The XMLHttpRequest Object
   _________________________
   - The XMLHttpRequest Object has a built in XML Parser.
   - The responseText property returns the response as a string.
   - The responseXML property returns the response as an XML DOM object.
   - If you want to use response as XML DOM object, you can use responseXML property -->
  <h2>My CD Collection:</h2>
  <button type="button" onclick="loadXMLDoc()">Get CD collection</button>
  <p id="demo"></p>
```

```
<script>
 function loadXMLDoc() {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function () {
   if (this.readyState == 4 && this.status == 200) {
     myFunction(this);
   }
  };
  xmlhttp.open("GET", "cd_catalog.xml", true);
  xmlhttp.send();
 }
 function myFunction(xml) {
  var x, i, xmlDoc, txt;
  xmlDoc = xml.responseXML;
  txt = "";
  x = xmlDoc.getElementsByTagName("ARTIST");
  for (i = 0; i < x.length; i++) {
   txt += x[i].childNodes[0].nodeValue + "<br>";
  }
  document.getElementById("demo").innerHTML = txt;
 }
</script>
```

Advantages

- XML DOM is language and platform independent.
- XML DOM is traversable - Information in XML DOM is organized in a hierarchy which allows developer to navigate around the hierarchy looking for specific information.
- XML DOM is modifiable - It is dynamic in nature providing developer a scope to add, edit, move or remove nodes at any point on the tree.

Disadvantages

- It consumes more memory (if the XML structure is large) as program written once remains in memory all the time until and unless removed explicitly.
- Due to the larger usage of memory its operational speed, compared to SAX is slower.

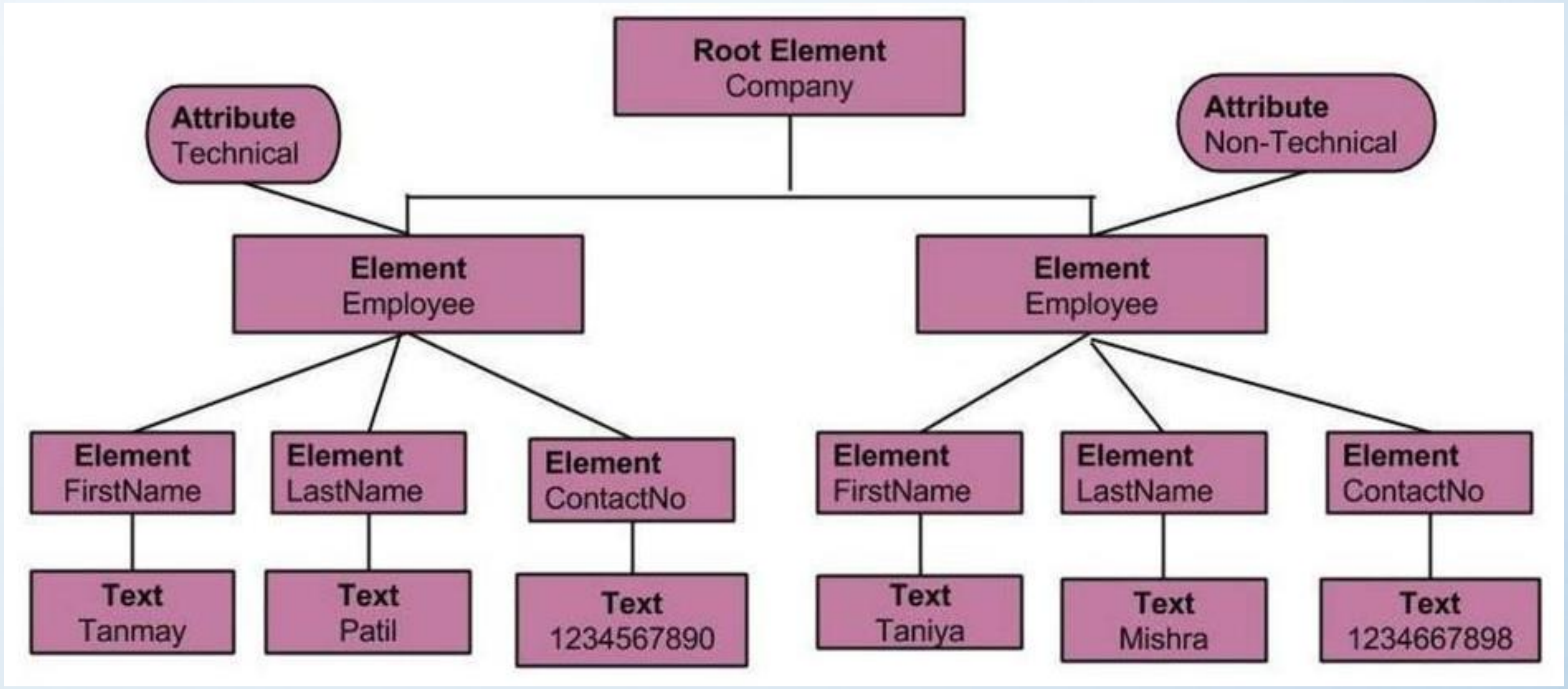
List of the node types and which node types they may have as children

- ***Document*** - Element (maximum of one), ProcessingInstruction, Comment, DocumentType (maximum of one)
- **DocumentFragment -** Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- **EntityReference -** Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- **Element -** Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
- **Attr -** Text, EntityReference
- ***ProcessingInstruction -*** *no children*
- **Comment -** no children
- ***Text* -** no children
- ***CDATASection -*** *no children*
- **Entity -** Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- ***Notation -*** *no children*

Example:node.xml

```
<?xml version="1.0"?>
<Company>
        <Employee category="technical">
                <FirstName>Tanmay</FirstName>
                <LastName>Patil</LastName>
                <ContactNo>1234567890</ContactNo>
        </Employee>
        <Employee category="non-technical">
                <FirstName>Taniya</FirstName>
                <LastName>Mishra</LastName>
                <ContactNo>1234667898</ContactNo>
        </Employee>
</Company>
```

Document Object Model of the XML Document



- The topmost node of a tree is called the root. The root node is <Company> which in turn contains the two nodes of <Employee>. These nodes are referred to as child nodes.
- The child node <Employee> of root node
- <Company>, in turn consists of its own child node (<FirstName>, <LastName>, <ContactNo>).
- The two child nodes, <Employee> have attribute values Technical and Non-Technical, are referred as attribute nodes.
- The text within the every node is called as text node.
- *Node object can have only one parent node object. This occupies the position above all the nodes. Here it is Company.*
- The parent node can have multiple nodes called as child nodes. These child nodes can have additional node called as attribute node. In the above example we have two attribute nodes Technical and Non-Technical. The attribute node is not actually a child of the element node, but is still associated with it.
- These child nodes in turn can have multiple child nodes. The text within the nodes is called as text node.
- The node objects at the same level are called as siblings.
- The DOM Identifies:

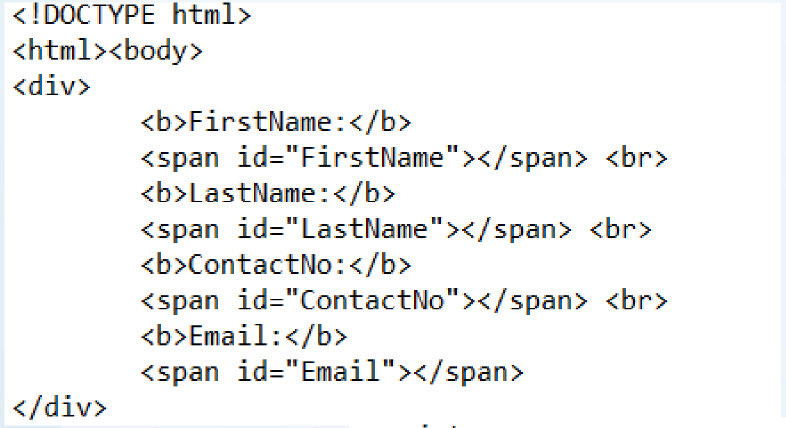
  - the objects to represent the interface and manipulate the document.

  - the relationship among the objects and interfaces.

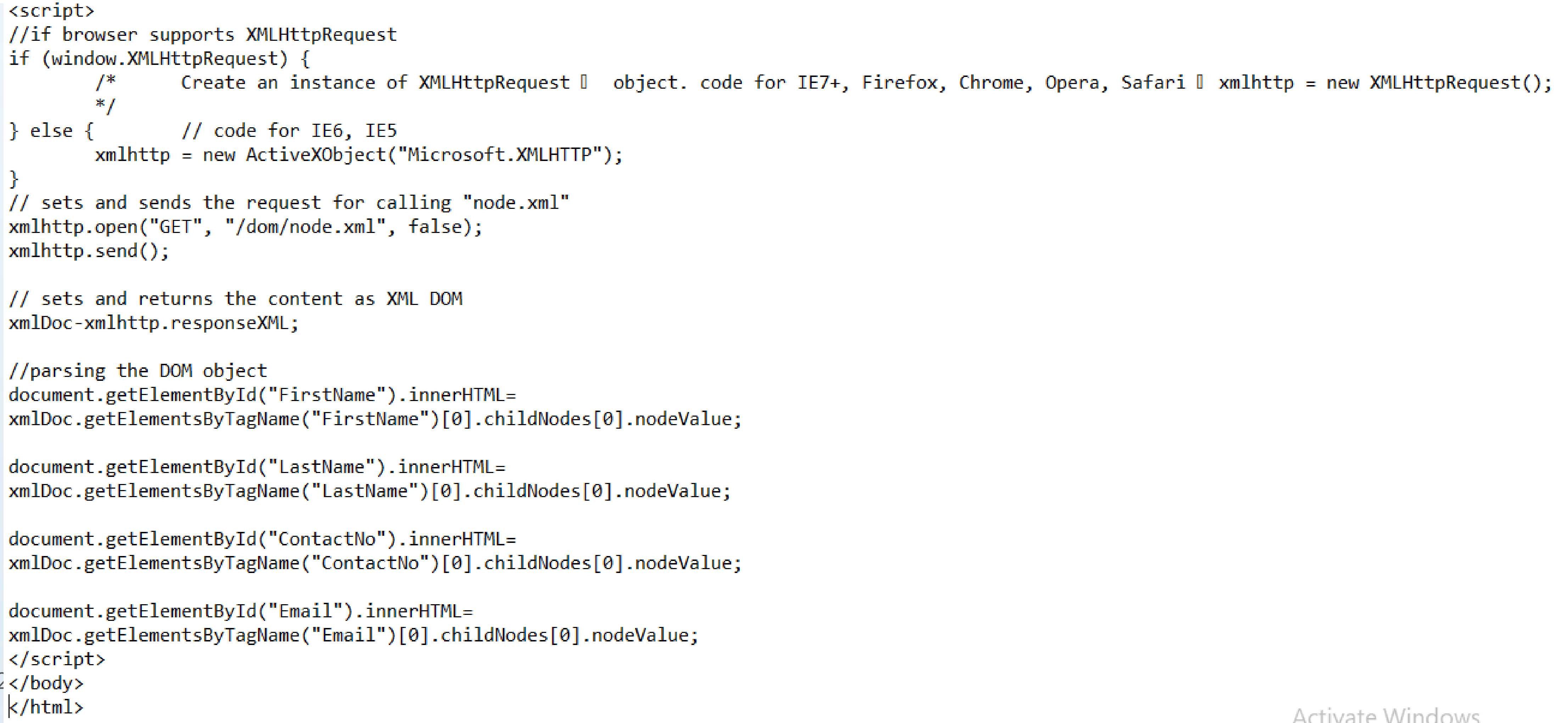
Accessing Nodes

Access a node in three ways:

- By using the getElementsByTagName() method.
- By looping through (traversing) the nodes tree.
- By navigating the node tree, using the node relationships.



```
<!DOCTYPE html>
<html><body>
<div>
        <b>FirstName:</b>
        <span id="FirstName"></span> <br>
        <b>LastName:</b>
        <span id="LastName"></span> <br>
        <b>ContactNo:</b>
        <span id="ContactNo"></span> <br>
        <b>Email:</b>
        <span id="Email"></span>
</div>
```



```
<script>
//if browser supports XMLHttpRequest
if (window.XMLHttpRequest) {
        /*      Create an instance of XMLHttpRequest   object. code for IE7+, Firefox, Chrome, Opera, Safari  xmlhttp = new XMLHttpRequest();
        */
} else {        // code for IE6, IE5
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
// sets and sends the request for calling "node.xml"
xmlhttp.open("GET", "/dom/node.xml", false);
xmlhttp.send();

// sets and returns the content as XML DOM
xmlDoc-xmlhttp.responseXML;

//parsing the DOM object
document.getElementById("FirstName").innerHTML=
xmlDoc.getElementsByTagName("FirstName")[0].childNodes[0].nodeValue;

document.getElementById("LastName").innerHTML=
xmlDoc.getElementsByTagName("LastName")[0].childNodes[0].nodeValue;

document.getElementById("ContactNo").innerHTML=
xmlDoc.getElementsByTagName("ContactNo")[0].childNodes[0].nodeValue;

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

SAX vs DOM

SAX = Simple API for XML

- Java-specific
- Interprets XML as a stream of events
- you supply event-handling callbacks
- SAX parser invokes your event- handlers as it parses
- Doesn't build data model in memory
- Serial access
- Very fast, lightweight
- Good choice when

  - no data model is needed, or

  - natural structure for data model is list, matrix, etc.

DOM = Document Object Model

- W3C standard for representing structured documents
- Platform and language neutral (not Java-specific!)
- Interprets XML as a tree of nodes
- Builds data model in memory
- Enables random access to data
- Therefore good for interactive apps
- More CPU and memory-intensive
- Good choice when data model has natural tree structure

Thank you.