PRESENTATION TECHNOLOGIES

1) XSL
2) XFORMS
3) XHTML

XSL & XSLT:
XSL stands for EXtensible Stylesheet Language.
What is XSLT?
XSLT stands for XSL Transformations. XSLT is the most important part of XSL. XSLT transforms an XML document into another XML document. XSLT uses XPath to navigate in XML documents. XSLT is a W3C Recommendation. We want to transform the following XML document ("cdcatalog.xml") into XHTML:

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

Then you create an XSL Style Sheet ("cdcatalog.xsl") with a transformation template:

```xml
<?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>
```

```xml
```
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th align="left">Title</th>
<th align="left">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>

The result is:

![My CD Collection table]

XFORMS:
XForms is the next generation of HTML forms. XForms is richer and more flexible than HTML forms. XForms will be the forms standard in XHTML 2.0. XForms is platform and device independent. XForms separates data and logic from presentation. XForms uses XML to define form data. XForms stores and transports data in XML documents. XForms contains features like calculations and validations of forms. XForms reduces or eliminates the need for scripting. XForms is a W3C Recommendation. The XForms Model. The XForms model is used to describe the data. The data model is an instance (a template) of an XML document. The XForms model defines a data model inside a <model> element:

<model>
<instance>
<person>
<fname/>
<lname/>
</person>
</instance>
</model>
The XForms Model

The XForms model is used to describe the data. The data model is an instance (a template) of an XML document. The XForms model defines a data model inside a <model> element:

```xml
<model>
  <instance>
    <person>
      <fname/>
      <lname/>
    </person>
  </instance>
  <submission id="form1" action="submit.asp" method="get"/>
</model>
```

All together it looks as below

```xml
<xforms>
  <model>
    <instance>
      <person>
        <fname/><lname/></person>
    </instance>
    <submission id="form1" action="submit.asp" method="get"/>
  </model>
  <input ref="fname"><label>First Name</label></input><input ref="lname">
    </input>
  <submit submission="form1">
    <label>Submit</label>
  </submit>
</xforms>
```

Output seems like:

```
First Name
Last Name
Submit
```

XHTML:

XHTML stands for EXtensible HyperText Markup Language. XHTML is aimed to replace HTML. XHTML is almost identical to HTML 4.01. XHTML is a stricter and cleaner version of HTML. XHTML is HTML defined as an XML application. XHTML is a W3C Recommendation. XHTML elements must be properly nested. XHTML elements must always be closed. XHTML elements must be in lowercase. XHTML documents must have one root element.
The 3 Document Type Definitions:

1) DTD specifies the syntax of a web page in SGML.
2) DTD is used by SGML applications, such as HTML, to specify rules that apply to the markup of documents of a particular type, including a set of element and entity declarations.
3) XHTML is specified in an SGML document type definition or 'DTD'.

An XHTML DTD describes in precise, computer-readable language, the allowed syntax and grammar of XHTML markup. There are currently 3 XHTML document types:

i. STRICT
ii. TRANSITIONAL
iii. FRAMESET

XHTML 1.0 specifies three XML document types that correspond to three DTDs:

i. Strict
ii. Transitional
iii. Frameset

XHTML 1.0 Strict:
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
We can use this when you want really clean markup, free of presentational clutter. We can use this together with Cascading Style Sheets.

XHTML 1.0 Transitional:
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
We can use this when you need to take advantage of HTML's presentational features and when you want to support browsers that don't understand Cascading Style Sheets.

XHTML 1.0 Frameset:
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd"
We can use this when you want to use HTML Frames to partition the browser window into two or more frames.

Why XHTML Modularization?

By splitting XHTML into modules, the W3C (World Wide web Consortium) has created small and well-defined sets of XHTML elements that can be used separately for small devices, or combined with other XML standards into larger and more complex applications.

Some of the modules are as below:
XSLT • XLINK • XPATH • XQuery

**XLINK:**

**XLink Syntax:**

In HTML, we know (and all the browsers know!) that the `<a>` element defines a hyperlink. However, this is not how it works with XML. In XML documents, you can use whatever element names you want - therefore it is impossible for browsers to predict what hyperlink elements will be called in XML documents. The solution for creating links in XML documents was to put a marker on elements that should act as hyperlinks.

**Example:**

```xml
<?xml version="1.0"?>
<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
  <homepage xlink:type="simple" xlink:href="http://www.w3schools.com">Visit W3Schools</homepage>
  <homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit W3C</homepage>
</homepages>
```

**XLink Attribute Reference**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xlink:actuate</td>
<td>loaded</td>
<td>Defines when the linked resource is read and shown</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>xlink:show</td>
<td>embed</td>
<td>Where to open the link. Replace is default</td>
</tr>
<tr>
<td></td>
<td>new</td>
<td></td>
</tr>
<tr>
<td></td>
<td>replace</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>xlink:type</td>
<td>simple</td>
<td>The type of link</td>
</tr>
<tr>
<td></td>
<td>extended</td>
<td></td>
</tr>
<tr>
<td></td>
<td>locator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>resource</td>
<td></td>
</tr>
<tr>
<td></td>
<td>link</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>
XPath:
XPath is a syntax for defining parts of an XML document. XPath uses path expressions to navigate in XML documents. XPath contains a library of standard functions. XPath is a major element in XSLT. XPath is a W3C Standard.

XPath Terminology

Nodes:
In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes. XML documents are treated as trees of nodes. The root of the tree is called the document node (or root node).

Relationship of Nodes
i. Parent
ii. Children
iii. Siblings
iv. Ancestors
v. Descendants

<xml version="1.0" encoding="ISO-8859-1"/>
<bookstore>
  <book>
    <title lang="eng">Harry Potter</title>
    <price>29.99</price>
  </book>
  <book>
    <title lang="eng">Learning XML</title>
    <price>35.95</price>
  </book>
</bookstore>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ nodename</td>
<td>Selects all child nodes of the named node</td>
</tr>
<tr>
<td>/</td>
<td>Selects from the root node</td>
</tr>
<tr>
<td>//</td>
<td>Selects nodes in the document from the current node that match the selection no matter where they are</td>
</tr>
<tr>
<td>.</td>
<td>Selects the current node</td>
</tr>
<tr>
<td>..</td>
<td>Selects the parent of the current node</td>
</tr>
<tr>
<td>@</td>
<td>Selects attributes</td>
</tr>
</tbody>
</table>

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

Predicates:
### Selecting Unknown Nodes:

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Matches any element node</td>
</tr>
<tr>
<td>@*</td>
<td>Matches any attribute node</td>
</tr>
<tr>
<td>node()</td>
<td>Matches any node of any kind</td>
</tr>
</tbody>
</table>

### Path Expression

<table>
<thead>
<tr>
<th>Path Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>/bookstore/*</td>
<td>Selects all the child nodes of the bookstore element</td>
</tr>
<tr>
<td>//*</td>
<td>Selects all elements in the document</td>
</tr>
<tr>
<td>//title[@*]</td>
<td>Selects all title elements which have any attribute</td>
</tr>
</tbody>
</table>

### Selecting several paths:

<table>
<thead>
<tr>
<th>Path Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>/bookstore/book[last()]</td>
<td>Selects the last book element that is the child of the bookstore element</td>
</tr>
<tr>
<td>/bookstore/book[last()-1]</td>
<td>Selects the last but one book element that is the child of the bookstore element</td>
</tr>
<tr>
<td>/bookstore/book[position()&lt; 3]</td>
<td>Selects the first two book elements that are children of the bookstore element</td>
</tr>
<tr>
<td>//title[@lang]</td>
<td>Selects all the title elements that have an attribute named lang</td>
</tr>
<tr>
<td>//title[@lang='eng']</td>
<td>Selects all the title elements that have an attribute named lang with a value of 'eng'</td>
</tr>
<tr>
<td>/bookstore/book[price&gt;35.00]</td>
<td>Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00</td>
</tr>
</tbody>
</table>
XQuery:

XQuery is the language for querying XML data. XQuery for XML is like SQL for databases. XQuery is built on XPath expressions. XQuery is supported by all the major database engines (IBM, Oracle, Microsoft, etc.). XQuery is a W3C Recommendation.

Functions:

XQuery uses functions to extract data from XML documents. The doc() function is used to open the "books.xml" file:

```
    doc("books.xml")
```

Path Expressions

<table>
<thead>
<tr>
<th>Path Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>//book/title</td>
<td>//book/price</td>
</tr>
<tr>
<td>///title</td>
<td>//price</td>
</tr>
<tr>
<td>/bookstore/book/title</td>
<td>//price</td>
</tr>
</tbody>
</table>

XQuery uses path expressions to navigate through elements in an XML document. The following path expression is used to select all the title elements in the "books.xml" file: doc("books.xml")/bookstore/book/title/bookstore selects the bookstore element, /book selects all
the book elements under the bookstore element, and /title selects all the title elements under each book element), The XQuery above will extract the following:

<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>

**Predicates:**

XQuery uses predicates to limit the extracted data from XML documents. The following predicate is used to select all the book elements under the bookstore element that have a price element with a value that is less than 30: `doc("books.xml")/bookstore/book[price<30]` The XQuery above will extract the following:

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

**With FLWOR:**

FLWOR is an acronym for "For, Let, Where, Order by, Return". The `for` clause selects all book elements under the bookstore element into a variable called $x`. The `where` clause selects only book elements with a price element with a value greater than 30. The `order by` clause defines the sort-order. Will be sort by the title element. The `return` clause specifies what should be returned. Here it returns the title elements.

**Example:** `doc("books.xml")/bookstore/book[price>30]/title`

The following FLWOR expression will select exactly the same as the path expression above:

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

The result will be:

- <title lang="en">XQuery Kick Start</title>
- <title lang="en">Learning XML</title>

With FLWOR you can sort the result:

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