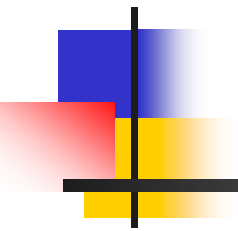
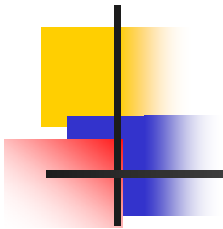


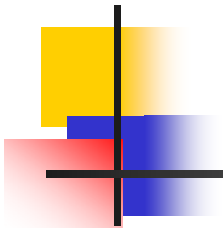
XML and Internet Databases

Chapter 26



Lecture Outline

- Introduction
- The anatomy of XML document
- Components of XML document
- XML validation
- Rules for well-formed XML document
- XML DTD
- More XML components
- References
- Reading list

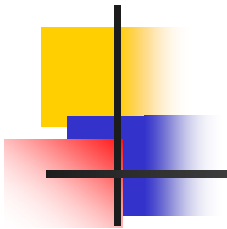


- Introduction

- What is XML
- How can XML be used
- What does XML look like
- XML and HTML
- XML is free and extensible

-- What is XML

- XML stands for Extensible Markup Language.
- XML developed by the World Wide Web Consortium (www.W3C.org)
- Created in 1996. The first specification was published in 1998 by the W3C
- It is specifically designed for delivering information over the internet.
- XML like HTML is a markup language, but unlike HTML it doesn't have predefined elements.
- You create your own elements and you assign them any name you like, hence the term extensible.
- HTML describes the presentation of the content, XML describes the content.
- You can use XML to describe virtually any type of document: Koran, works of Shakespeare, and others.
  - Go to http://www.ibiblio.org/boask to download



-- How can XML be Used?

- XML is used to Exchange Data
- With XML, data can be exchanged between incompatible systems
- With XML, financial information can be exchanged over the Internet
- XML can be used to Share Data
- XML can be used to Store Data
- XML can make your Data more Useful
- XML can be used to Create new Languages

-- What does XML look like

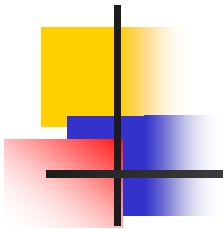
Books

Title	Author	year
Java	Mustafa	1995
Pascal	Ahmed	1980
Basic	Ali	1975
Oracle	Emad	1973
….	….	

Relation

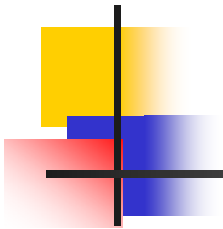
```
<Books>

    <Book>
        <Title>    Java      </Title>
        <Author>   Mustafa    </Author>
        <Year>     1995       </year>
    </Book>
    …
    …
    …
    <Book>
        <Title>    Oracle    </Title>
        <Author>   Emad      </Author>
        <Year>     1973       </Year>
    </Book>
    ….
     ….
</ Books>
```

XML document



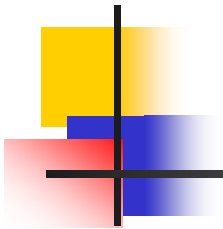
-- XML and HTML ...

- XML is not a replacement for HTML
- XML was designed to carry data
- XML and HTML were designed with different goals
  - XML was designed to describe data and to focus on what data is
  - HTML was designed to display data and to focus on how data looks.
- HTML is about displaying information, while XML is about describing information



... -- XML and HTML

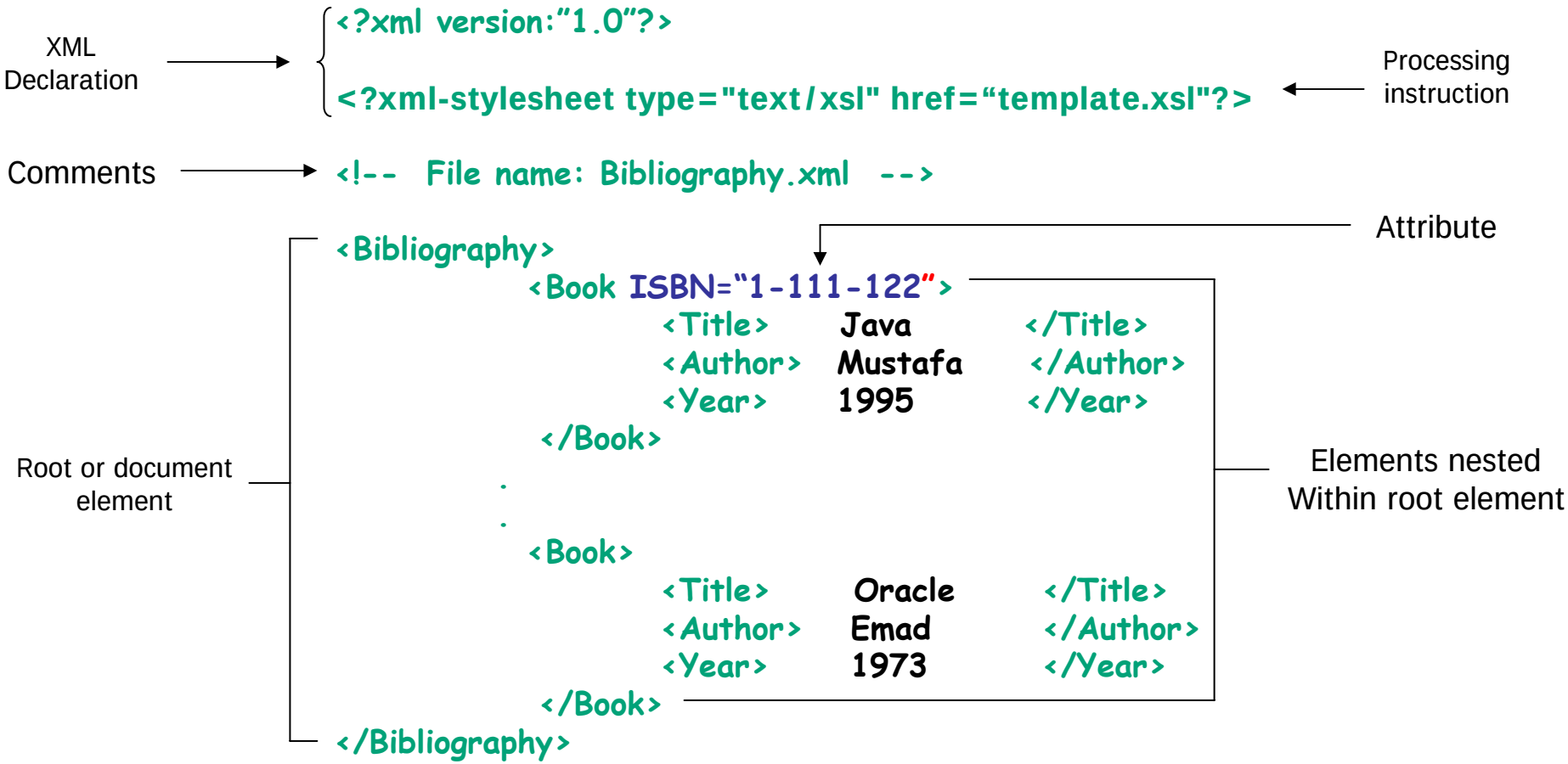
- HTML is for humans
  - HTML describes web pages
  - You don’t want to see error messages about the web pages you visit
  - Browsers ignore and/or correct as many HTML errors as they can, so HTML is often sloppy
- XML is for computers
  - XML describes data
  - The rules are strict and errors are not allowed
    - In this way, XML is like a programming language
  - Current versions of most browsers can display XML

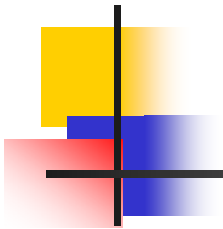


-- XML is free and extensible

- XML tags are not predefined
  - You must "invent" your own tags
  - The tags used to mark up HTML documents and the structure of HTML documents are predefined
  - The author of HTML documents can only use tags that are defined in the HTML standard
- XML allows the author to define his own tags and his own document structure, hence the term extensible.

-The Anatomy of XML Document





- Components of an XML Document

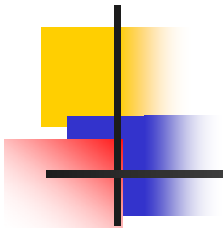
- Elements
  - Each element has a beginning and ending tag
  - <TAG_NAME>…</TAG_NAME>
  - Elements can be empty (<TAG_NAME />)
- Attributes
  - Describes an element; e.g. data type, data range, etc.
  - Can only appear on beginning tag
    - Example: <Book **ISBN = “1-111-123**”>
- Processing instructions
  - Encoding specification (Unicode by default)
  - Namespace declaration
  - Schema declaration

-- XML declaration

- The XML declaration looks like this:

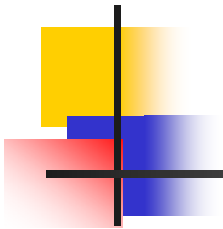
  `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`

  - The XML declaration is not required by browsers, but is required by most XML processors (so include it!)
  - If present, the XML declaration must be first--not even white space should precede it
  - Note that the brackets are <? and ?>
  - version="1.0" is required (I am not sure it is the only version so far)
  - encoding can be "UTF-8" (ASCII) or "UTF-16" (Unicode), or something else, or it can be omitted
  - standalone tells whether there is a separate DTD



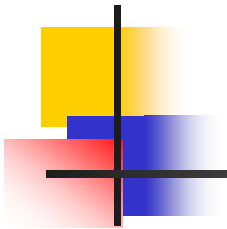
-- Processing Instructions

- PIs (Processing Instructions) may occur anywhere in the XML document (but usually in the beginning)
- A PI is a command to the program processing the XML document to handle it in a certain way
- XML documents are typically processed by more than one program
- Programs that do not recognize a given PI should just ignore it
- General format of a PI: `<?target instructions?>`
- Example: `<?xml-stylesheet type="text/css" href="mySheet.css"?>`



-- XML Elements

- An XML element is everything from the element's start tag to the element's end tag
- XML Elements are extensible and they have relationships
- XML Elements have simple naming rules
  - Names can contain letters, numbers, and other characters
  - Names must not start with a number or punctuation character
  - Names must not start with the letters xml (or XML or Xml ..)
  - Names cannot contain spaces

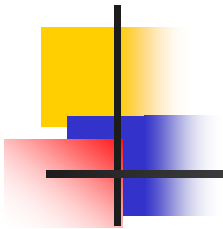


-- XML Attributes

- XML elements can have attributes
- Data can be stored in child elements or in attributes
- Should you avoid using attributes?
  - Here are some of the problems using attributes:
    - attributes cannot contain multiple values (child elements can)
    - attributes are not easily expandable (for future changes)
    - attributes cannot describe structures (child elements can)
    - attributes are more difficult to manipulate by program code
    - attribute values are not easy to test against a Document Type Definition (DTD) - which is used to define the legal elements of an XML document

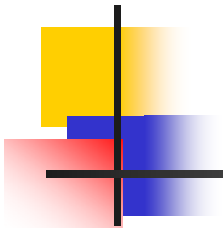
-- Distinction between subelement and attribute

- In the context of documents, attributes are part of markup, while subelement contents are part of the basic document contents
- In the context of data representation, the difference is unclear and may be confusing
  - Same information can be represented in two ways
    - <Book … Publisher = "McGraw Hill"> … <??Book>
    - <Book>
      …
      <Publisher> McGraw Hill </Publisher>
      …
      </Book>
- Suggestion: use attributes for identifiers of elements, and use subelements for contents



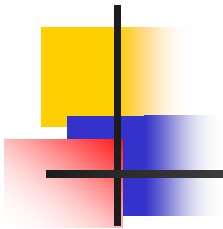
- XML Validation

- Well-Formed XML document:
  - Is an XML document with the correct basic syntax
- Valid XML document:
  - Must be well formed plus
  - Conforms to a predefined DTD or XML Schema.



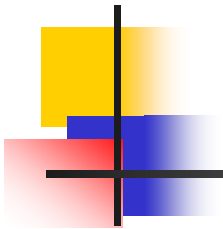
- Rules For Well-Formed XML

- Must begin with the XML declaration
- Must have one unique root element
- All start tags must match end-tags
- XML tags are case sensitive
- All elements must be closed
- All elements must be properly nested
- All attribute values must be quoted
- XML entities must be used for special characters



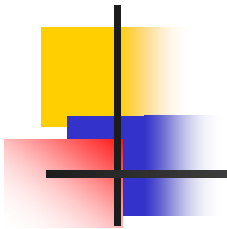
- XML DTD

- A DTD defines the legal elements of an XML document
  - defines the document structure with a list of legal elements and attributes
- XML Schema
  - XML Schema is an XML based alternative to DTD
- Errors in XML documents will stop the XML program
- XML Validators



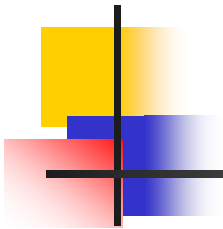
-- CDATA

- By default, all text inside an XML document is parsed
- You can force text to be treated as unparsed character data by enclosing it in <![CDATA[...]]>
- Any characters, even & and <, can occur inside a CDATA
- White space inside a CDATA is (usually) preserved
- The only real restriction is that the character sequence]]> cannot occur inside a CDATA
- CDATA is useful when your text has a lot of illegal characters (for example, if your XML document contains some HTML text)



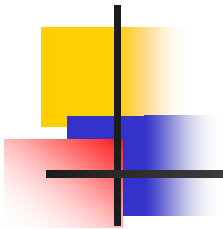
-- XML and DTDs

- A DTD (**D**ocument **T**ype **D**efinition) describes the structure of one or more XML documents.
- Specifically, a DTD describes:
  - Elements
  - Attributes, and
  - Entities
- An XML document is *well-structured* if it follows certain simple syntactic rules
- An XML document is *valid* if it also specifies and conforms to a DTD



-- Why DTDs?

- With DTD, each of your XML files can carry a description of its own format with it.
- With a DTD, independent groups of people can agree to use a common DTD for interchanging data.
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.



-- Parsers

- An *XML parser* is an API that reads the content of an XML document
  - Currently popular APIs are DOM (Document Object Model) and SAX (**S**imple **A**PI for **X**ML)
- A *validating parser* is an XML parser that compares the XML document to a DTD and reports any errors

-- An XML example

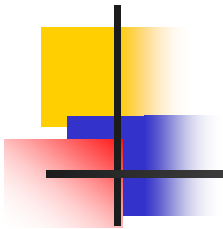
```
<novel>
        <foreword>
                <paragraph> This is a great novel </paragraph>
        </foreword>
        <chapter number="1">
                <paragraph>It was a dark and stormy night.</paragraph>
                <paragraph>Suddenly, a shot rang out!</paragraph>
        </chapter>
</novel>
```

- An XML document contains (and the DTD describes):
  - Elements, such as novel and paragraph, consisting of tags and content
  - Attributes, such as number="1", consisting of a name and a value
  - Entities (not used in this example)

-- A DTD example

```
<!DOCTYPE novel [
        <!ELEMENT novel (foreword, chapter+)>
        <!ELEMENT foreword (paragraph+)>
        <!ELEMENT chapter (paragraph+)>
        <!ELEMENT paragraph (#PCDATA)>
        <!ATTRIBUTE chapter number CDATA #REQUIRED>
]>
```

- A novel consists of a foreword and one or more chapters, in that order
  - Each chapter must have a number attribute
- A foreword consists of one or more paragraphs
- A chapter also consists of one or more paragraphs
- A paragraph consists of parsed character data (text that cannot contain any other elements)



- ELEMENT descriptions

- Suffixes:

?	optional	foreword?
+	one or more	chapter+
*****	zero or more	appendix*

- Separators:

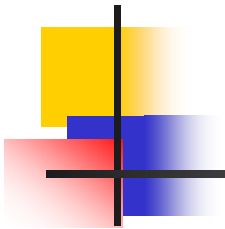
,	both, in order	foreword?, chapter+
|	or	section|chapter

- Grouping:

( )		grouping (section|chapter)+

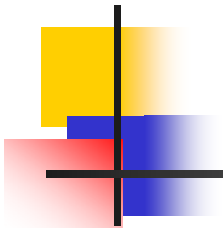
-- Another example: XML

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

<!DOCTYPE myXmlDoc SYSTEM "http://www.mysite.com/mydoc.dtd">
<weatherReport>
    <date>05/29/2002</date>
    <location>
        <city>Philadelphia</city>
        <state>PA</state>
        <country>USA</country>
    </location>
    <temperature-range>
        <high scale="F">84</high>
        <low scale="F">51</low>
    </temperature-range>
</weatherReport>
```



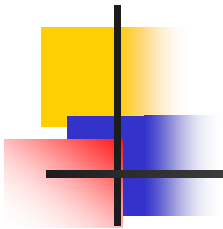
-- The DTD for this example

```
<!ELEMENT weatherReport (date, location, temperature-range)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT location (city, state, country)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT temperature-range ((low, high)|(high, low))>
<!ELEMENT low (#PCDATA)>
<!ELEMENT high (#PCDATA)>
<!ATTLIST low scale (C|F) #REQUIRED>
<!ATTLIST high scale (C|F) #REQUIRED>
```



-- XML Schema ...

- The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.
- An XML Schema:
  - defines elements that can appear in a document
  - defines attributes that can appear in a document
  - defines which elements are child elements
  - defines the order of child elements
  - defines the number of child elements
  - defines whether an element is empty or can include text
  - defines data types for elements and attributes
  - defines default and fixed values for elements and attributes



... -- XML Schema ...

- Many think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:
  - XML Schemas are extensible to future additions
  - XML Schemas are richer and more useful than DTDs
  - XML Schemas are written in XML
  - XML Schemas support data types
  - XML Schemas support namespaces

... -- XML Schema ...

- Look at this simple XML document called "note.xml":
  - <?xml version="1.0"?>
    <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body> Don't forget me this weekend!</body>
    </note>
- This is a simple DTD file called "note.dtd" that defines the elements of the XML document above ("note.xml"):
  - <!ELEMENT note (to, from, heading, body)>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>

-- Simple XML schema

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.w3schools.com"
      xmlns="http://www.w3schools.com" elementFormDefault="qualified">
      <xs:element name="note">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="to" type="xs:string"/>
                        <xs:element name="from" type="xs:string"/>
                        <xs:element name="heading" type="xs:string"/>
                        <xs:element name="body" type="xs:string"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

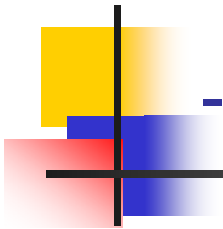
... -- XML schema

- The <schema> is the root element of every XML schema

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

<xs:schema>
    ...
    ...
</xs:schema>
```

- The <schema> element may contain some attributes. A schema declaration often looks something like this:

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

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



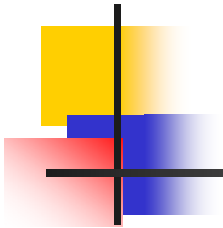
-- Xpath

- XPath is a syntax used for selecting parts of an XML document
- The way XPath describes paths to elements is similar to the way an operating system describes paths to files
- XPath is almost a small programming language; it has functions, tests, and expressions
  - XPath is a W3C standard

--- Terminology

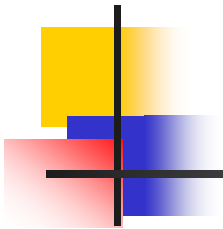
```
<library>
    <book>
        <chapter>
        </chapter>
        <chapter>
            <section>
                <paragraph/>
                <paragraph/>
            </section>
        </chapter>
    </book>
</library>
```

- library is the parent of book; book is the parent of the two chapters
- The two chapters are the children of book, and the section is the child of the second chapter
- The two chapters of the book are siblings (they have the same parent)
- library, book, and the second chapter are the ancestors of the section
- The two chapters, the section, and the two paragraphs are the descendents of the book



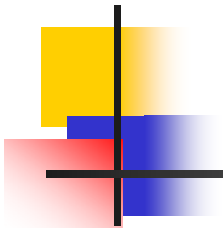
--- Paths

- **Operating System**
  - / = the root directory
  - /users/dave/foo = the file named foo in dave in users
  - foo = the file named foo in the current directory
  - . = the current directory
  - .. = the parent directory
  - /users/dave/* = all the files in /users/dave
- **Xpath**
  - /library = the root element (if named library)
  - /library/book/chapter/section = *every* section element in a chapter in every book in the library
  - section = *every* section element that is a child of the current element
  - . = the current element
  - .. = parent of the current element
  - /library/book/chapter/* = all the elements in /library/book/chapter



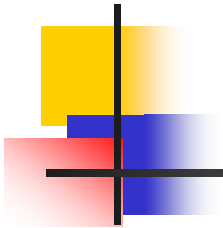
--- Slashes

- A path that begins with a / represents an *absolute path*, starting from the top of the document
  - Example: /email/message/header/from
  - Note that even an absolute path can select *more than one* element
  - A slash by itself means “the whole document”
- A path that does *not* begin with a / represents a path starting from the current element
  - Example: header/from
- A path that begins with // can start from *anywhere* in the document
  - Example: //header/from selects every element from that is a child of an element header
  - This can be expensive, since it involves searching the entire document



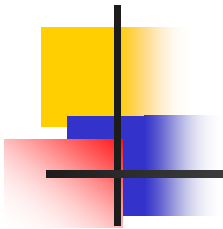
--- Brackets and last()

- A number in brackets selects a particular matching child
  - Example: /library/book[1] selects the first book of the library
  - Example: //chapter/section[2] selects the second section of every chapter in the XML document
  - Example: //book/chapter[1]/section[2]
  - Only *matching* elements are counted; for example, if a book has both sections and exercises, the latter are ignored when counting sections
- The function last() in brackets selects the last matching child
  - Example: /library/book/chapter[last()]
- You can even do simple arithmetic
  - Example: /library/book/chapter[last()-1]



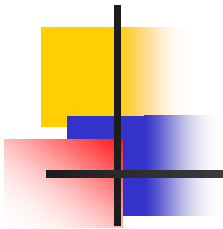
--- Stars

- A star, or asterisk, is a “wild card”--it means “all the elements at this level”
  - Example: /library/book/chapter/* selects every child of every chapter of every book in the library
  - Example: //book/* selects every child of every book (chapters, tableOfContents, index, etc.)
  - Example: /*/*/*/paragraph selects every paragraph that has exactly three ancestors
  - Example: //* selects every element in the entire document



-- 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 defined by the W3C
- XQuery is supported by all the major database engines (IBM, Oracle, Microsoft, etc.)
- XQuery will become a W3C standard - and developers can be sure that the code will work among different products
- XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators.



--- XQuery Basic Syntax Rules

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

--- XQuery Example

- Example:
  - 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]
    - Output

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

--- XQuery FLWOR Expressions

- The syntax of Flower expression looks like the combination of SQL and path expression
- The following path expression will select all the title elements under the book elements that is under the bookstore element that have a price element with a value that is higher than 30.

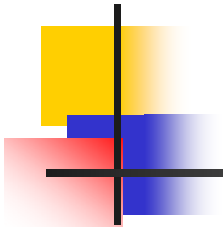
```
doc("books.xml")/bookstore/book[price>30]/title
```

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

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

  - Output

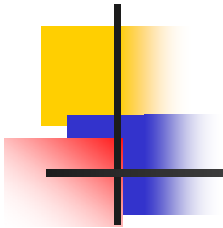
```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```



--- FLWOR briefly explained

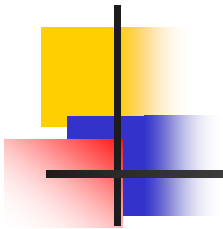
```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

- 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** sorts the results according to the specified element
  - The **return** clause specifies what should be returned. Here it returns the title elements



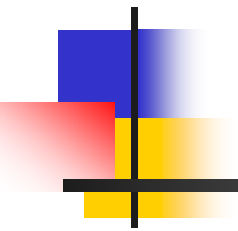
- References

- W3 Schools XML Tutorial
  - http://www.w3schools.com/xml/default.asp
- W3C XML page
  - http://www.w3.org/XML/
- XML Tutorials
  - http://www.programmingtutorials.com/xml.aspx
- Online resource for markup language technologies
  - http://xml.coverpages.org/
- Several Online Presentations



- Reading List

- W3 Schools XML Tutorial
  - http://www.w3schools.com/xml/default.asp



END