Motivation

- Applications consume and transfer data
  - Software libraries require files
  - Communication between online services
  - How to represent such data usefully?
- Option 1: every app defines its own syntax
  - For example, many of the UNIX programs
  - Requires specialized language design and parsers
- Option 2: use a common "extensible" syntax
  - But which one? The relational model?
  - Allows for software reuse, but often introduces significant challenges: proper decomposition, nulls due to fixed attributes, etc.
  - Often, translation into relations is unnatural and problematic

What is XML?

- Depending on who you’re asking
- Answer 1: Rich documents that enrich text with markup
  - Markup captures mainly formatting, meta data (e.g., title) and links
- Answer 2: A hierarchical data model
  - Elegantly generalizes the relational model, object model
  - Most prominent model of semistructured data

Objects: Relations vs. XML

- Faculty: Computer Science
  - Building: Taub
  - Member: Ora Grumberg
    - Office: Taub 630
    - Phone: 4327
  - Member: Irad Yavneh
    - Office: Taub 618, Taub 537
    - Phone: 4361, 4362

Document: Relations vs. XML

- Haifa
  - Technion generally refers to the 1.2-square-kilometer site located on the pine-covered north-eastern slopes of Mount Carmel. The campus comprises 100 buildings, occupied by thousands of people everyday. The Technion has two additional campuses. Its original building in mountain, in use by the Technion until the mid-1960s, now houses the Israel National Museum of Science, Technology and Space. The Rapaport Faculty of Medicine is located in the neighborhood of St. Galin, adjacent to Ramat Hanadiv, the largest medical center in Northern Israel.

Standardization Organizations

- ISO
  - International Organization for Standardization
  - Founded in 1947 to promote global commerce
  - In fact, UN backed reform of the 1926 "ISA"
  - Representatives from 162 countries
- W3C
  - World Wide Web Consortium
  - International standardization for the Web
  - Founded in 1994, by Tim Berners-Lee, supported by MIT, European Commission, and DARPA
  - Berners Lee is still heading W3C
  - Sponsored by industrial companies
  - Offices all around the world
**XML History**

- **1986**: SGML ISO standard for sharing documentation readable by machines
  - Stands for Standard Generalized Markup Language
  - Considered highly complicated, expensive to support
  - Extensible data model
  - Can be extended to many special cases using schemas
- **1991**: Tim Berners-Lee proposes the first version of HTML as an instantiation of SGML
  - Much simpler than SGML; restricted to Web pages
- **1998**: XML 1.0 released by W3C
  - Extensible and clean like SGML, but things stripped off to get the simplicity of HTML

---

**XML-Based Technologies**

- RDF (format for the Semantic Web)
- WSDL (Web-service protocol)
- SOAP (object communication)
- RSS (Web-feed format)
- SVG (graphics)
- MathML (format for math notation)

---

**Related Standards**

- XML Schemas strengthen the typing and schema capabilities (compared to built-in DTDs)
- XPath is a language for querying and accessing XML elements
- XSLT is a language for transforming XML documents into other XML documents
  - Including XHTML for displaying XML files
- XQuery is a query language for XML
- XLink and Xpointer provide a rich support cross-references among XML documents/elements

---

**Outline**

- Introduction
- **XML Syntax**
  - DTD
    - Element Declaration
    - Attribute Declaration
    - Entities
    - Validity
  - XPath
    - Axes
    - Predicates
    - Examples of XPath Uses
  - Namespaces
XML Components:

- **XML Declaration**
  - `<?xml version="1.0" standalone="yes" encoding="UTF-8"?>`
    - With standalone="yes" we mean that we allow an external DTD
      - Default is "no"
    - Default encoding is UTF-8
      - Good for Arabic, Armenian, Cyrillic, Greek, Hebrew, Latin, ...
    - The entire declaration is optional
      - But it is pretty conventional to include it

- **Document Type Definition (DTD)**
  - Defines a schema
    - What sequences of elements can each element have as children?
    - For a given element name, which attributes are required/allowed?
    - We will study DTD in depth later
  - Can be:
    - Internal (inside the XML document) or
    - External (in an external URL)

- **Internal DTD Example (w3schools.com)**
  ```xml
  <!DOCTYPE note [ 
    <!ELEMENT note (to,from,heading,body)> 
    <!ELEMENT from (#PCDATA)> 
    <!ELEMENT heading (#PCDATA)> 
    <!ELEMENT body (#PCDATA)> 
  ]>
  <note>
    <to>Tove</to>
    <from>Jonas</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend</body>
  </note>
  ```

- **External DTD Example**
  ```xml
  <!DOCTYPE countries SYSTEM "world.dtd">
  <country>
    <country name="Asia"> Israel: Note: 2001 population 6199008</country>
    <city name="Jerusalem"/> <city name="Ashdod"/>
    <country name="Europe"> France: 2001 population 60424213
  </country>
  ```

XML Elements:

- Structure:
  - Opening tag: `<name attribute="v1", ... attribute="vn">`
  - Closing tag: `</name>`
- Proper nesting is required
  - proper-nesting := `<tag …>` proper-nesting `</tag>`
  - Example of illegal XML: `<i>Hello</i>`
  - (Web browsers will accept it as legal HTML)
- Useful abbreviation for empty elements:
  - `<e>…</e>`
  - Examples in XHTML: `<br/>` `<hr/>`
- The entire document must be nested within a single element, denoted as the root
Attributes

- **Restriction**: An element cannot have two occurrences of the same attribute.
  - For example, this is not allowed:
    `<person name="bill" name="william">`

- **Design**: not always clear whether an information item should be an element or an attribute.
  - `<country population="7M"/>`
  - `<country><population>7M</population></country>`

- If an attribute should have its own attributes (e.g., year), then it should be an element.

Text

- **XML has a single primitive type**: `text`
- **Always nested within an element**
  - Some special characters (e.g., `<`, `>`, `;`, ...) are disallowed – encoded as `entity references`
  - `<eq>\&gt;</eq>`
  - `<eq>\&apos;</eq>`

- **Textual elements are termed PCDATA**
  - Name originated in SGML

(N)CDATA

We want to see the text as is, even though it includes tags.

```xml
<message>
  <head>
    Entering a Kennel Club Member
  </head>
  <description>
    Enter the member by the name on his or her papers. Use the NAME tag.
    The NAME tag has two attributes. Common (all in lowercase, please!) is the dog's call name.
    Breed (also in all lowercase) is the dog's breed. Please see the breed reference guide for acceptable breeds.
    Your entry should look something like this:
    <example>
      <![CDATA[
        <NAME common="freddy" breed="springer-spaniel">Sir Fredrick of Ledyard's End</NAME>
      ]]>
    </example>
  </description>
</message>
```

XML Must be Well Formed

- An XML document is **well-formed** if
  - Tags are syntactically correct
  - Every start tag has an end tag
  - Tags are properly nested
  - There is a root tag
  - A start tag does not have two occurrences of the same attribute

- When a DTD is specified, a document must be both **well formed** and **valid**

Outline

- **Introduction**
- **XML Syntax**
  - **DTD**
    - Element Declaration
    - Attribute Declaration
    - Entities
    - Validity
  - **XPath**
    - Axes
    - Predicates
    - Examples of XPath Uses
  - **Namespaces**
Motivation

- A DTD adds syntactical requirements in addition to the well-formed requirement
- Why is it useful?
  - The usual “why schema” arguments
  - Helps avoiding errors when creating/editing XML
  - Facilitates communication via XML
  - Allows processing programs to make assumptions
  - Macros for constants/includes (entities)

Example: An Address Book

Example Revisited

The Address Book DTD

Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses
- Namespaces

Forms of Element Definitions:

- A regular expression
  - (name, greet?, address*, (fax | tel)*, email*)
- EMPTY
  - The element has no content
  - Example: <ELEMENT br EMPTY> (in XML: <br/>)
- ANY
  - Mixture of PCDATA and elements defined in the DTD
- Mixed content
  - (#PCDATA)
  - (#PCDATA | address | name)*
  - (#PCDATA | italic | bold)*
**DTD Regular Expressions**

<table>
<thead>
<tr>
<th>Format</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>name, tel, ...</td>
<td>Element name</td>
</tr>
<tr>
<td>e₁, e₂</td>
<td>e₁ followed by e₂</td>
</tr>
<tr>
<td>e*</td>
<td>Zero or more occurrences of e</td>
</tr>
<tr>
<td>e?</td>
<td>Zero or one occurrences of e</td>
</tr>
<tr>
<td>e+</td>
<td>One or more occurrences of e</td>
</tr>
<tr>
<td>e₁, e₂</td>
<td>e₁ or e₂</td>
</tr>
<tr>
<td>(e)</td>
<td>Grouping</td>
</tr>
</tbody>
</table>

(Note: an inductive definition)

**Restriction on Regular Expressions**

- DTD standard does not allow every regular expression; only ones that can be "efficiently verified" in the following sense:
  - We can determine whether a string s matches the regex by scanning s left to right; on every symbol we will know which regex symbol it matches without looking ahead in the string
- Such a string is called 1-unambiguous
- Example:
  - (a|b)*, a is not 1-unambiguous
  - b*, a(b*, a)* is 1-unambiguous
    - Note: the two express the same language (string set)

**Left-to-Right Scanning**

\[(a | b)^*, a \]

The requirement states (or can be formalized as):

Every regular expression has a deterministic Glushkov automaton

**Slightly More Precisely: Glushkov Automata**

- Glushkov automaton of a regex [1961]:
  - Preprocessing: replace each \( a^+ \) with \( aa^* \)
  - State = symbol occurrence + init state
  - Transition \( a \rightarrow b \) whenever b is a possible follower of a in the left-to-right parse
  - Accepting states = possible last symbols

**DTD Unambiguity Requirement**

The requirement states (or can be formalized as):

Every regular expression has a deterministic Glushkov automaton

This is a violation of the DTD recommendation:

```xml
<!ELEMENT filming ((movie | director)*, (movie | director))>
```
Mixed Content

- Described by a repeatable OR group
  \((\text{#PCDATA} \mid \text{element}_1 \mid \cdots \mid \text{element}_k)^*\)

  Rules:
  - This is the only regular expression allowed
  - #PCDATA must be first

Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses
- Namespaces

Attributes

\[
\text{<!ELEMENT height (#PCDATA)>}
\]

\[
\text{<!ATTLIST height}
\]

\[
\text{unit CDATA "cm" default value "cm" #REQUIRED #IMPLIED>}
\]

\[
\text{<ATTLIST element-name attribute-name attribute-type attribute-behavior>}
\]

Attribute Types

- CDATA: General text
- ID: Unique identifier
  - At most one ID attribute per element
  - No two elements can have the same identifying attribute values
- IDREF: ID value of an element in the document
  - Can be any element (not typed)
- IDREFS: A list of IDREFs (separated by space)
- ENTITY: A declared entity (later)
- ENTITIES: A list of ENTITYs (separated by space)
  \((\text{value}_1 \mid \cdots \mid \text{value}_k)\): One of \text{value}_1, …, \text{value}_k

Attribute Behavior

- #REQUIRED: Attribute must occur
  - name CDATA #REQUIRED : <person name="Alma"/>
- #IMPLIED: Optional
  - spouse CDATA #IMPLIED : <person>
- #FIXED: Has a predefined value (in the DTD)
  - genus CDATA #FIXED "Panthera" : <lion genus="Panthera"/>
- Default value: implied unless the attribute is given (with a different value)
  - unit CDATA "cm" : <length>

Example of Recursive XML

\[
\text{<!ELEMENT people (person*)>}
\]

\[
\text{<!ELEMENT person (name, dateOfBirth, person, person)>}
\]

Problem: not satisfiable by any finite XML document

\[
\text{<!ELEMENT people (person*)>}
\]

\[
\text{<!ELEMENT person (name, dateOfBirth, person?, person?)>}
\]

Problem: illegal (not 1-unambiguous)

Problem: if there is one parent, is it the mother of the father?
Problem: we need to replicate parents for siblings
Using References

```xml
<people>
  <person id="lisa" mother="marge" father="homer">
    <name>Lisa Simpson</name>
  </person>
  <person id="bart" mother="marge" father="homer">
    <name>Bart Simpson</name>
  </person>
  <person id="marge" children="bart lisa">
    <name>Marge Simpson</name>
  </person>
  <person id="homer" children="bart lisa">
    <name>Homer Simpson</name>
  </person>
</people>
```

Outline

- Introduction
- XML Syntax
  - DTD
    - Element Declaration
    - Attribute Declaration
    - Entities
    - Validity
  - XPath
    - Axes
    - Predicates
    - Examples of XPath Uses
  - Namespaces

XML Entities (Macros)

- Used for:
  1. Referring to special characters
     - Seen in action, e.g., &lt;
  2. Defining shared global constants (user defined) – these are named entities
     - Example follows
  3. Including (embedding) external XML
     - Example follows
  4. Defining DTD macros
     - Not discussed

Named Entities

- Syntax: `<ENTITY name "value">`
- Reference an entity by `&name;`
- Examples:
  - `<ENTITY d "Donald">`
  - `<ENTITY dd "&d; Duck">`
    - In XML: `<name>Mr. &dd;</name>`
  - `<ENTITY eu "Europe">`
    - In XML: `<country continent="&eu;">`

Including External Files

```xml
<!DOCTYPE jokes [
  <!ELEMENT jokes (joke)>]
<!ELEMENT joke (#PCDATA)>
<!ENTITY joke.1 SYSTEM "http://j.com/joke1.txt">
<!ENTITY joke.2 SYSTEM "http://j.com/joke2.txt">
<!ENTITY joke.3 SYSTEM "http://j.com/joke3.txt">
]
<jokes>
  <joke>&joke.1;</joke>
  <joke>&joke.2;</joke>
  <joke>&joke.3;</joke>
</jokes>
```

Even Better

```xml
<!DOCTYPE jokes [
  <!ELEMENT jokes (joke)>]
<!ELEMENT joke (#PCDATA)>
<!ENTITY joke.1 SYSTEM "http://j.com/joke1.txt">
<!ENTITY joke.2 SYSTEM "http://j.com/joke2.txt">
<!ENTITY joke.3 SYSTEM "http://j.com/joke3.txt">
]
<jokes>
  <joke><![CDATA[&joke.1;]]></joke>
  <joke><![CDATA[&joke.2;]]></joke>
  <joke><![CDATA[&joke.3;]]></joke>
</jokes>
```

XML Entities (Macros)

- Used for:
  1. Referring to special characters
     - Seen in action, e.g., &lt;
  2. Defining shared global constants (user defined) – these are named entities
     - Example follows
  3. Including (embedding) external XML
     - Example follows
  4. Defining DTD macros
     - Not discussed

Named Entities

- Syntax: `<ENTITY name "value">`
- Reference an entity by `&name;`
- Examples:
  - `<ENTITY d "Donald">`
  - `<ENTITY dd "&d; Duck">`
    - In XML: `<name>Mr. &dd;</name>`
  - `<ENTITY eu "Europe">`
    - In XML: `<country continent="&eu;">`

Including External Files

```xml
<!DOCTYPE jokes [
  <!ELEMENT jokes (joke)>]
<!ELEMENT joke (#PCDATA)>
<!ENTITY joke.1 SYSTEM "http://j.com/joke1.txt">
<!ENTITY joke.2 SYSTEM "http://j.com/joke2.txt">
<!ENTITY joke.3 SYSTEM "http://j.com/joke3.txt">
]
<jokes>
  <joke>&joke.1;</joke>
  <joke>&joke.2;</joke>
  <joke>&joke.3;</joke>
</jokes>
```

Even Better

```xml
<!DOCTYPE jokes [
  <!ELEMENT jokes (joke)>]
<!ELEMENT joke (#PCDATA)>
<!ENTITY joke.1 SYSTEM "http://j.com/joke1.txt">
<!ENTITY joke.2 SYSTEM "http://j.com/joke2.txt">
<!ENTITY joke.3 SYSTEM "http://j.com/joke3.txt">
]
<jokes>
  <joke><![CDATA[&joke.1;]]></joke>
  <joke><![CDATA[&joke.2;]]></joke>
  <joke><![CDATA[&joke.3;]]></joke>
</jokes>
```
**Valid Documents**

A well-formed XML document is *valid* if it conforms to its DTD:
- The sequence of names of the children of each element $e$ matches the regex of $\text{name}(e)$
- The types and values of attributes are correct
- IDs are unique
- IDREF attributes point to identifier values

**Outline**

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses
- Namespaces

**DTDs vs. Schemas**

- DTDs are rather weak specifications by DB & PL standards
  - Only one base type – PCDATA
  - No numbers, Booleans, dates, etc.
  - IDREFs are untyped
  - That is, the type of the object referenced is not known
  - No constraints beyond parent/child
  - No inheritance
  - Context-free element definitions
    - For example, `<role>` in a `<movie>` or a `<play>`?
- A much richer notion of a schema is *XML Schema*, which we do not study here

```xml
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city capital="yes"><name>Jerusalem</name></city>
    <city><name>Ashdod</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60424213</population>
  </country>
</countries>
```

**Outline**

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses
- Namespaces

```xml
<!ELEMENT countries (country*)>
<!ELEMENT country (name, population?, city*)>
<!ATTLIST country continent CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT city (name)>
<!ATTLIST city capital (yes|no) "no">
<!ELEMENT population (#PCDATA)>
<!ATTLIST population year CDATA #IMPLIED>
<!ENTITY &as; "Asia">
<!ENTITY &eu; "Europe">
<!ENTITY &af; "Africa">
<!ENTITY &am; "America">
<!ENTITY &au; "Australia">
```

```xml
world.dtd
```
The XML DOM Tree

DOM = Document Object Model

The XPath Language

- XPath expressions are used for referencing elements (nodes) of an XML document
- Used in XSLT (next subject today) and in XQuery (a query language for XML)
  - We will see examples in the end
- The syntax resembles that of the Unix file system

```
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
countries:
  <country continent="&as;">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city capital="yes">
      <name>Jerusalem</name>
    </city>
    <city>
      <name>Ashdod</name>
    </city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60424213</population>
  </country>
</countries>
```

XPath Expressions

- An XPath expression (or just XPath for short) matches paths in the XML tree
- An absolute path begins at the root of the document
  - Starts with / or //
  - For example, /countries/country/city, //city
- A relative path begins with a context node that is defined by the application that uses the XPath
  - For example, city/name, or ./name

Applying XPath to XML

- Applying an XPath expression \( e \) to a context node \( v \) results in the list of all nodes \( u \), such that \( e \) matches the path from \( v \) to \( u \)
- Applying an XPath expression \( e \) to a document \( d \) means applying \( e \) to root(\( d \))
- The order in the list is the one induced by the preorder of the nodes in the DOM tree
**Outline**

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses
- Namespaces

**XPath Steps and Axis**

- An XPath describes a sequence of steps that together characterize a path
- A step is defined by an axis that specifies a binary relationship between nodes
  - The axis describes how to get from the current node to the next one
  - For example, parent-child, child-parent, ancestor-descendant, etc.
- Consecutive steps are separated by /

**Child Axis**

- A child axis has the simple form `tagName`
  - Go to an element child with the tag `tagName`
- For example,
  - `/tagName` matches the `tagName` child of root
  - `city/name`
  - `/countries/country/city`
- The child axis `*` matches every tag
  - For example: `/*/*/city`, `*/name`

**Child-Axis Examples**

```
/countries
```

```
<countries>
  <country>
    <continent name="Asia" year="2001"/>
    <country name="Israel" population="6190000"/>
  </country>
  <city name="Jerusalem" capital="yes"/>
  <city name="Ashdod" capital="no"/>
</countries>
```

```
/countries/country/city
```

```
<countries>
  <country>
    <continent name="Asia" year="2001"/>
    <country name="Israel" population="6190000"/>
  </country>
  <city name="Jerusalem" capital="yes"/>
  <city name="Ashdod" capital="no"/>
</countries>
```

```
/countries/country/city
```

```
<countries>
  <country>
    <continent name="Asia" year="2001"/>
    <country name="Israel" population="6190000"/>
  </country>
  <city name="Jerusalem" capital="yes"/>
  <city name="Ashdod" capital="no"/>
</countries>
```
**Descendant Examples**

/* /country/*

document root

An attribute is not an element child!

**Self and Descendant-or-Self**

- The *self* axis "." denotes the identity relationship
  - That is, the step "remain in the current node"
  - /countries//country: 2 /countries//country
  - country//city // country//city
- The *descendant-or-self* axis means: either stay in the current node or go to some descendant of the current node
  - descendant-or-self.node()
- // is a shorthand for /descendant-or-self:node()/
  - For example, country//name

**Descendant Examples**

/countries//name

document root

countries
  country
    continent name population city name capital name capital
    Asia Israel 6190000 name capital name capital
    2001 Jerusalem yes Ashdod no

**Descendant Examples**

//country/*

document root

countries
  country
    continent name population city name capital name capital
    Asia Israel 6190000 name capital name capital
    2001 Jerusalem yes Ashdod no

**Descendant Examples**

//country/.

document root

countries
  country
    continent name population city name capital name capital
    Asia Israel 6190000 name capital name capital
    2001 Jerusalem yes Ashdod no

**Descendant Examples**

//*/

document root

countries
  country
    continent name population city name capital name capital
    Asia Israel 6190000 name capital name capital
    2001 Jerusalem yes Ashdod no

Context
Other Axis Types

- The parent axis "." denotes the parent relationship
  - “Go to the parent of the current node”
- XPath has more axis types (denoted by a different syntax from the ones shown earlier); examples:
  - descendant
  - ancestor
  - ancestor-or-self
  - following-sibling
  - preceding-sibling

Referring Attributes

- The attribute axis is written as @attName
  - That is, “go to the attribute attName of the current node”
- The operator @* matches every attribute

Attribute Examples

//country/@continent

Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses
- Namespaces
XPath Predicates

- Predicates are used for filtering steps out
  - For example, //city[@capital="yes"] will match only capital cities
- Formally, given a predicate [P]:
  - P evaluated over target node → true/false
  - The step is taken iff the value is true
  - The node reached in the last step is the context node
- XPath has a rich logic for predicates; we demonstrate only the common ones

---

//population[@city/@name="Jerusalem"]

```xml
<DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&eu;">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city>Jerusalem</city>
  </country>
</countries>
```

//country[@city]

```xml
<DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&eu;">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city>Jerusalem</city>
  </country>
</countries>
```

//population[@city/@name="Jerusalem"]

```xml
<DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60424213</population>
  </country>
</countries>
```

//country[@city]

```xml
<DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60424213</population>
  </country>
</countries>
```

An XPath evaluates to true if and only if its result is not empty.

//country[@city/@name="Jerusalem"]

```xml
<DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&eu;">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city>Jerusalem</city>
  </country>
</countries>
```

//country[@city]

```xml
<DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&eu;">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city>Jerusalem</city>
  </country>
</countries>
```

//country[@city/@name="Jerusalem"]

```xml
<DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&eu;">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city>Jerusalem</city>
  </country>
</countries>
```
Functions

- Inside XPath predicates, you can use predefined functions
- Examples:
  - `last()` – returns the number of nodes obtained from the last axis step
  - `position()` – returns the position of the node in the list of nodes from the last axis step
  - `name()` – returns the tag of the current node
  - `count(XPath)` – returns the number of nodes satisfying XPath

A number acts as an index

That is, the number $n$ evaluates to true iff $n$ is the position of the node among all those reached in the last step (city)
Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses
- Namespaces

Final Remarks on XPath

- We presented the abbreviated (sugared) syntax syntax of Xpath
- For example, city//@name is an abbr. of child::city[parent::node]/attribute::name
- More details on XPath:
  - XPath tutorial in W3Schools
  - XPath W3C Recommendation

XPath in XQuery

```
<catalog>
  <cd title="1984">...
    <artist>David Bowie</artist>
    <id country="UK">561</id>
    <price>
      <above>$5.99</above>
      <below>$3.99</below>
    </price>
</cd>
{ for $t1 in catalog/cd } return {
  $t1/@country
  &lt;id country="US"&gt;
  &lt;price&gt;
    &lt;above>$11.90&lt;/above&gt;
    &lt;below>$9.90&lt;/below&gt;
  &lt;/price&gt;
  &lt;artist&gt;Aretha: Lady Soul&lt;/artist&gt;
  &lt;year&gt;1981&lt;/year&gt;
  &lt;id country="US"&gt;
  &lt;/cd>
}</catalog>
```

FLWOR expressions:
For Let Where Order by Return
Web Pages – The Whole Picture

Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses
- Namespaces

Add XML Namespaces

- A mechanism for creating intuitive unique names (for elements and attributes)
  - Those can be used all over the Web, cf. RDF
- Semantically, a namespace is a collection of names that were created for a specific domain of applications
- We will see namespaces in action when we learn RDF

Adding Namespaces

```xml
<root
  xmlns:h="http://www.w3.org/TR/html4/
  xmlns:f="http://www.w3schools.com/furniture">

  <table>
    <tr>
      <td h:Apples>100</td>
      <td h:Bananas>200</td>
    </tr>
  </table>

  <table f:name="African Coffee Table" f:length="120" f:width="120">
    <tr>
      <td f:name="African Coffee Table">100</td>
      <td f:width="120">200</td>
    </tr>
  </table>

</root>
```
### Terminology

<table>
<thead>
<tr>
<th>prefix</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>local name</td>
<td>table</td>
</tr>
<tr>
<td>qualified name</td>
<td>table</td>
</tr>
<tr>
<td>namespace URI</td>
<td><a href="http://www.w3.org/TR/html4/">http://www.w3.org/TR/html4/</a></td>
</tr>
<tr>
<td>expanded name</td>
<td><a href="http://www.w3.org/TR/html4/table">http://www.w3.org/TR/html4/table</a></td>
</tr>
</tbody>
</table>

### Alternative Semantics (Default NS)

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

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

### Scope of Namespaces

- The scope of a namespace declaration is the element containing the declaration and all descendant elements.
- More than one namespace can be declared in the same scope:
  - At most one can be the default namespace.
  - All others must have unique prefixes.