



University of Dublin  
Trinity College



---

# Navigating an XML Document – Part 2

Owen.Conlan@scss.tcd.ie  
Athanasios.Staikopoulos@scss.tcd.ie

---

# XPath versions

- W3C Recommendation
  - XPATH 1.0 - (1999)
  - XPATH 2.0 – (2010) – backwards compatible
- XPath 1.0
  - considers a single XML document as a tree of nodes
  - Nodes have identity
  - Set of nodes – unordered collection of nodes
- XPath 2.0
  - More complex, is a superset of XPath 1.0
  - More elaborate data model
  - More functions
  - It does not considers on a single document tree, but on arbitrary data sets
  - These can be arranged in sequences of items – ordered sets



# Basic Concepts

- Node Types
  - XML documents are treated as trees of nodes
  - The topmost element of the tree is called the root (or document) node
  - XPath defines seven node types
- Context Node
  - Provides the starting point (current node) that is basis of path navigation and evaluation
  - Default is the root (document)
- Location Steps
  - Provides the directions
  - Sequences the nodes
  - The evaluation of each node provides the current context
  - Example: [/node1/node2/node3](#)



# Node Types

- XPath defines 7 node types

Root/Document Node	The root of the tree representing the entire document contents, represented by the "/"
Element	Element nodes are defined by pairs of start <code>&lt;title&gt;</code> and end tags <code>&lt;/title&gt;</code>
Text	A character sequence in an element, comment, processing instruction, or namespace
Attribute	The name and value of an attribute in an element
Comment	Comments in an XML source document, such as <code>&lt;!-- model diagram --&gt;</code>
Processing Instruction	An instruction in the source document, such as the <code>&lt;?xml-stylesheet href="book.xsl" type="text/xsl"?&gt;</code>
Namespace	A namespace declaration



# Useful Properties of a Node

- Name (Except root, text and comment nodes)
  - Qualified by the namespace, such as <xm:term> -"xm" is the namespace, "term" is the local part. They can be accessed by using the functions name(), namespace-uri(), local-name().
- String-value
  - E.g. text if text node, comment text if comment node, attribute value if attribute node.
  - It can be accessed by the string() function.
- Child
  - List of child nodes
- Parent
  - Every node except root
- Has-attribute
  - List of attribute nodes associated with element node
- Has-namespace
  - List of namespace nodes associated with element node



# Data Types

- XPath 1.0
  - A **number** : stored as a floating point
  - A **string** : a sequence of characters
  - A **boolean** : a true or false value
  - A **node set** : an unordered collection of unique nodes
- XPath 2.0
  - Data types in XPATH 1.0 are pretty primitive
  - Supports data types taken from XML Schema
  - XPath 2.0 defines five additional datatypes
    - anyAtomicType, untyped, untypedAtomic, dayTimeDuration, and yearMonthDuration.



# XPath Evaluation Results

- The result of an XPath 1.0 expression is a node set
- Node sets
  - duplicates are not allowed (unique)
  - No order is implied
- The result of an XPath 2.0 expression is a sequence
- Sequences
  - Are ordered collections (list)
  - Zero, one or more items are allowed (or just nodes)
  - duplicates are allowed
  - The empty sequence is a valid sequence
- Items
  - an item is a reference to a node or an atomic value
  - Each item has a value (42) as well as a type (xs:integer)
- Atomic values (integers, string, booleans, etc.)



# Absolute & Relative Paths

- A location path can be absolute or relative.
- If the location path starts with the root node (/) then you are using an absolute location path

For example

- /root/node1/node2
- /html/body/h3

- If the location path begins with the name of a descendant, you're using a relative location path.

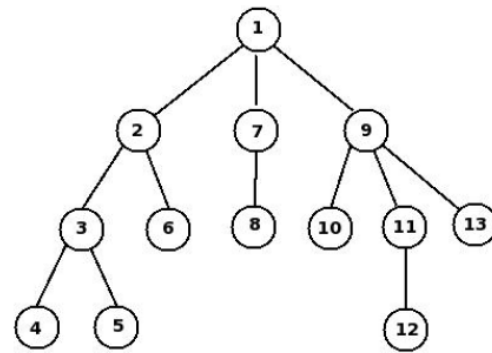
For example,

- node1/node2
- //node1/node2 (anywhere in document)

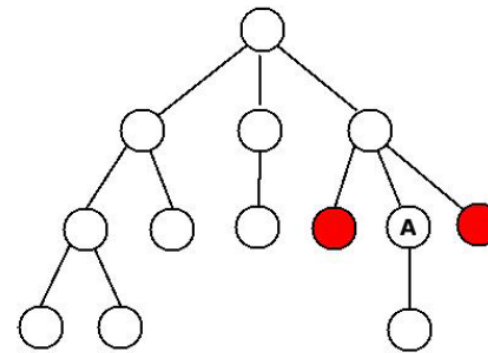


# XML Document Navigation

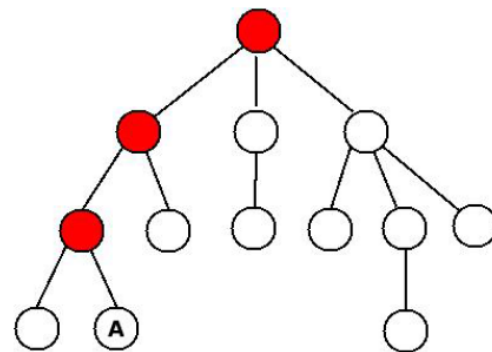
- The XPath data model treats an XML document as a tree of nodes, based on DOM
- Formally, a tree is a connected, acyclic, undirected graph



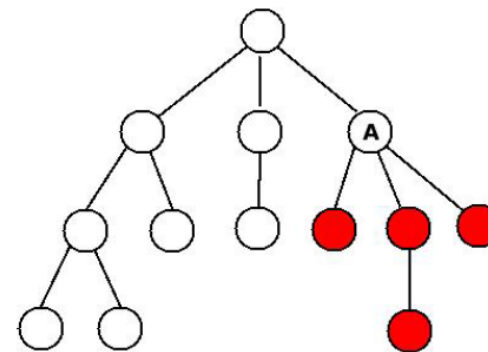
Document order



Siblings of A



Ancestors of A



Descendants of A



# Namespaces

- Any path expression can use a QName (prefix:local-name),
  - For example, //foo:book
    - Selects all 'book' elements in the document that belong to the foo namespace
- Matching is based on the local name and the namespace name (and not the prefix)
- The prefix binding to a namespace (e.g., foo to http://foo.example.com) is not part of the path expression.
  - it is defined externally (application specific)
- A path expression without a prefix will only match elements without an associated namespace



# Select Parent and Ancestors

- From the context node you can access your parent and ancestors
- `..` matches the parent of the current context node
  - `../section`
- Navigate just like directories
- You can go back many levels
  - `../../../body`



# Select Unknown Elements (\*)

- XPath wildcard (\*) put in place in a tag represents any one tag
- Example `/*/*/MARK` will return any MARK object appearing at the third level of nesting in the document



# Select Attribute @

- Attributes are referred to by putting at (@) before the name
- Appear in the path as if nested within the tag
- For example
  - /book/@lang
    - Select the 'lang' attribute of books



# Select Several Paths (|)

- By using the **union** (|) operator in an XPath expression you can select several paths
- For example
  - `//book/title | //book/price`
    - Selects all the 'title' elements AND the 'price' elements within the 'book' elements



# Predicates - Conditional Matching

- A tag in a path that is followed by a condition `[..]` will ensure that only nodes that satisfy the condition are included in the resultant set
- Example
  - `/bookstore/book[price>35.00]`
    - Selects the 'books' elements of a 'bookstore' where the 'price' element has a value greater than 35.00



# Anatomy of a Location Step

- A step in an XPath expression consists of three parts: an *axis*, a *node* test, and zero or more *predicate* tests

Specifies direction to go in document tree

Tests whether nodes encountered should be selected for next step

Filters nodes selected by the node test

`Child::Student[name="paul"]`



# Axis

- An **axis** defines the nodes selected relative to the current node. In XPath there are 13 axes defined:

- ancestor
- ancestor-or-self
- attribute
- child
- descendant
- descendant-or-self
- following
- following-sibling
- Namespace
- parent
- preceding
- preceding-sibling
- self



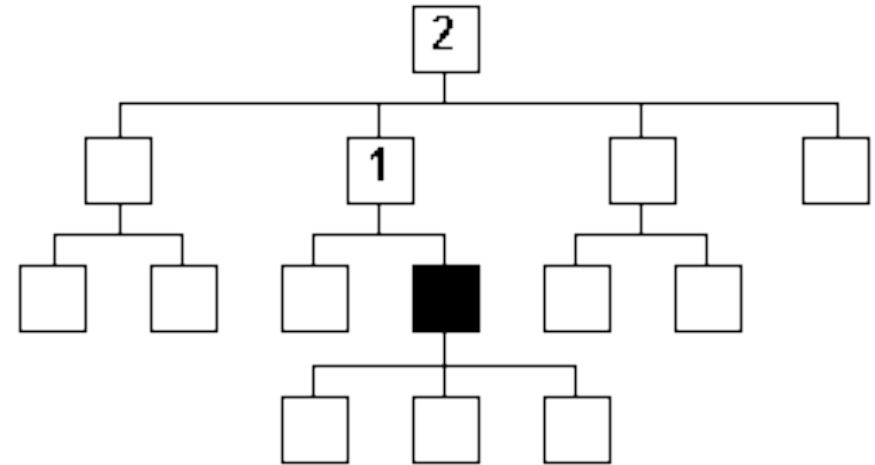
# Axis ancestor::

- **ancestor**

Selects all the nodes that are ancestors of the origin

- **Syntax**

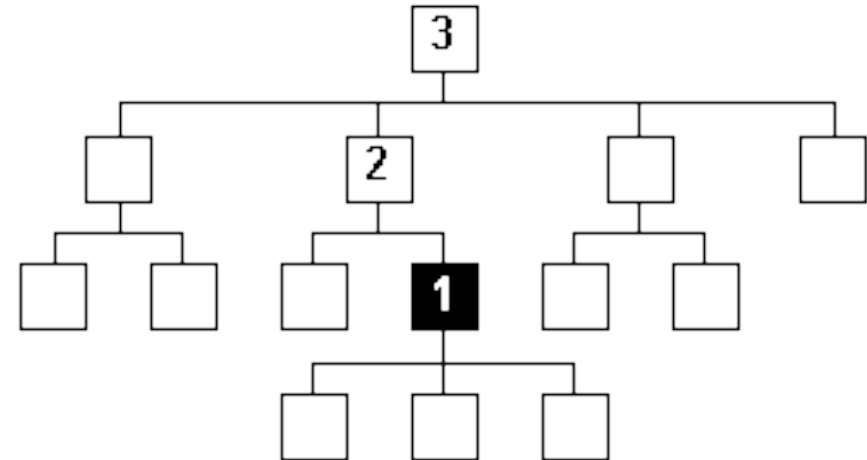
ancestor::node



# Axis ancestor-or-self::

- **ancestor-or-self**

Selects the same nodes as the ancestor axis, but starting with the origin node



- Syntax

ancestor-or-self::node



# Axis attribute::

- **attribute**

If the origin node is an element, this axis selects all its attribute nodes.

Otherwise, it selects nothing (an empty sequence). The order for attributes is arbitrary.

- **Syntax**

1. `attribute::lang`
2. `@lang`



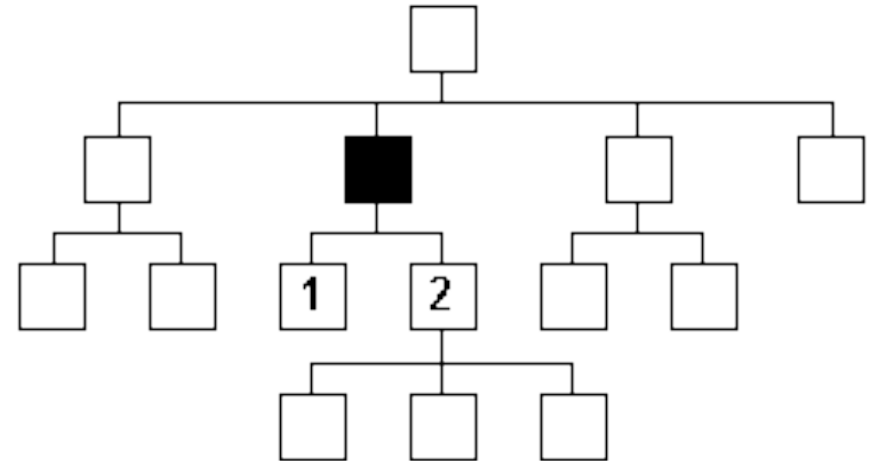
# Axis child::

- **child**

Selects all the children of the origin node, in document order.

- **Syntax**

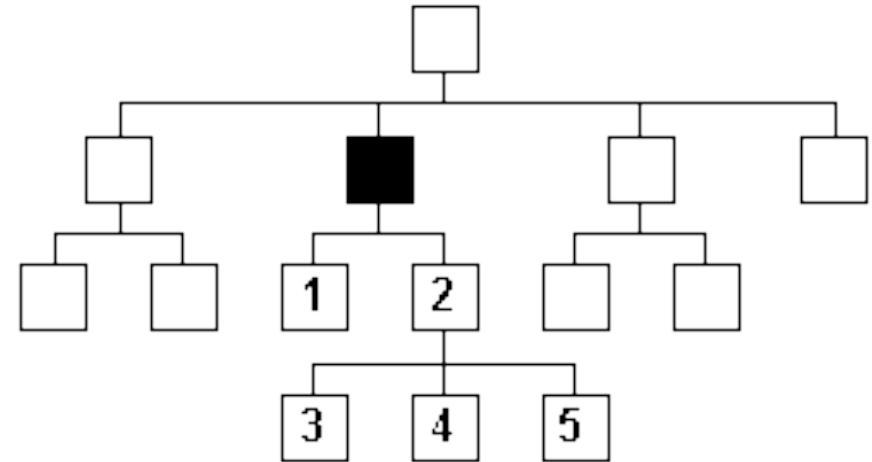
1. `child::node`
2. `/node`



# Axis descendant::

- **descendant**

Selects all the children of the origin node, and their children, and so on recursively. The resulting nodes are in document order.



- **Syntax**

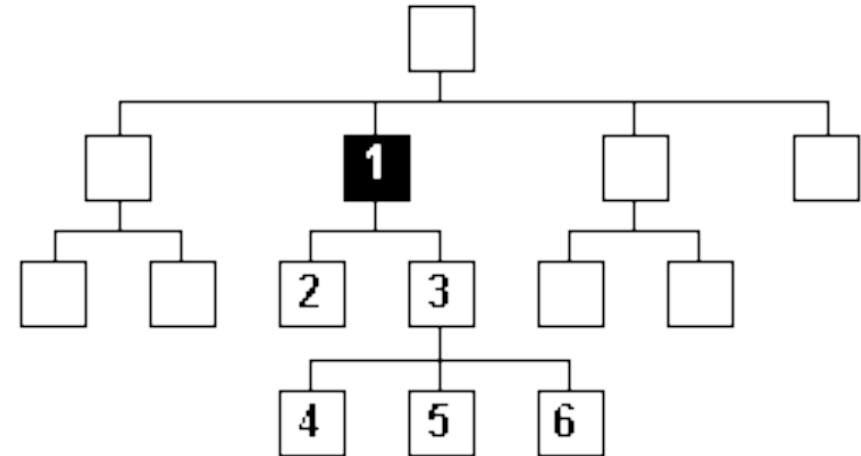
descendant::node



# Axis descendant-or-self::

- **descendant-or-self**

This is the same as the descendant axis, except that the first node selected is the origin node itself.



- Syntax

1. Descendant-or-self::node

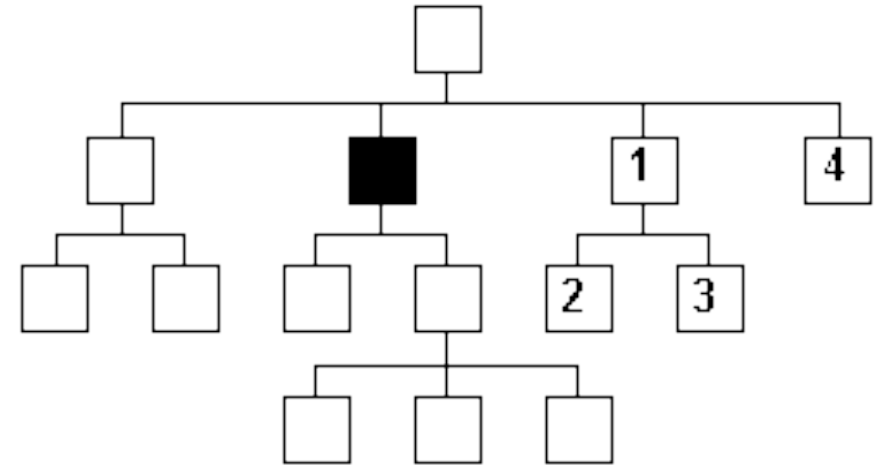
2. //



# Axis following::

- **following**

This selects all the nodes that appear after the origin node in document order, excluding the descendants of the origin node



- Syntax

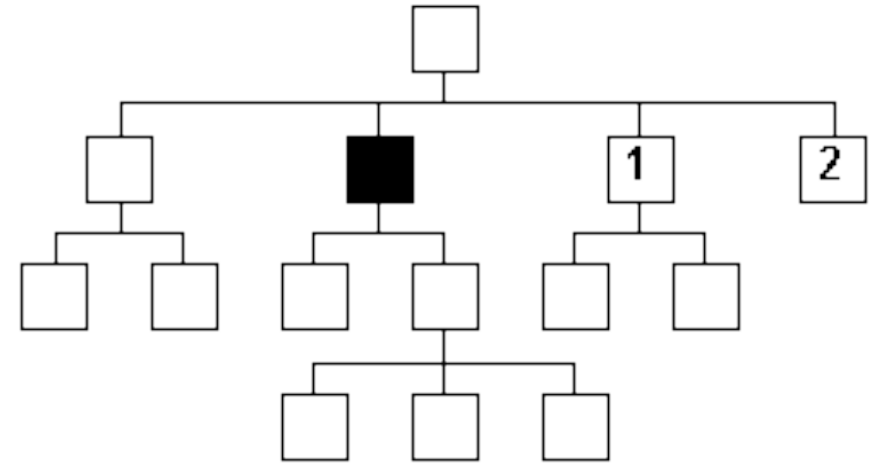
following::node



# Axis following-sibling::

- **following-sibling**

This selects all the nodes that follow the origin node in document order, and that are children of the same parent node.



- Syntax

Following-sibling::node



# Axis namespace::

- **namespace**

If the origin node is an element, this axis selects all the namespace nodes that are in scope for that element; otherwise, it is empty. The order of the namespace nodes is undefined

- **Syntax**

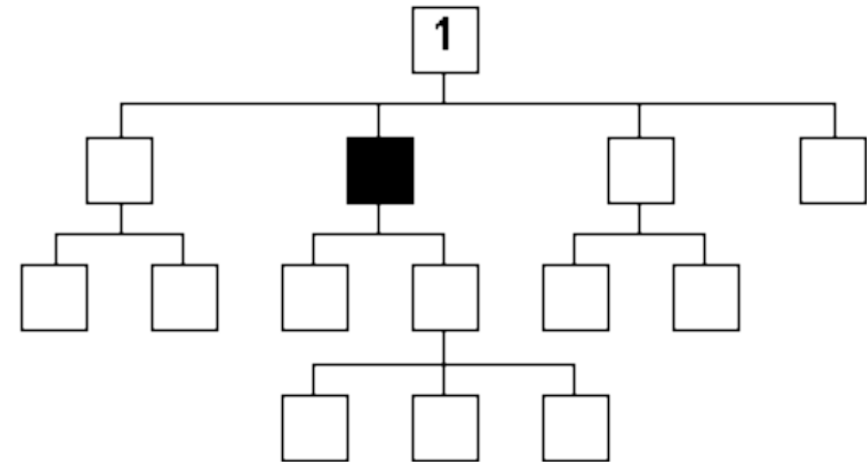
namespace::node



# Axis parent:::

- **parent**

This axis selects a single node, the parent of the origin



## Syntax

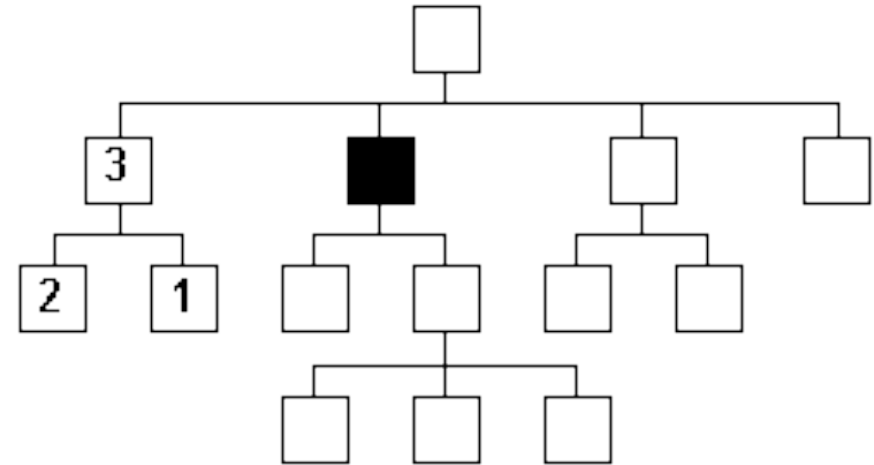
1. parent:::node
2. ..



# Axis preceding::

- **preceding**

This selects all the nodes that appear before the origin node, excluding the ancestors of the origin node.



- **Syntax**

preceding::node



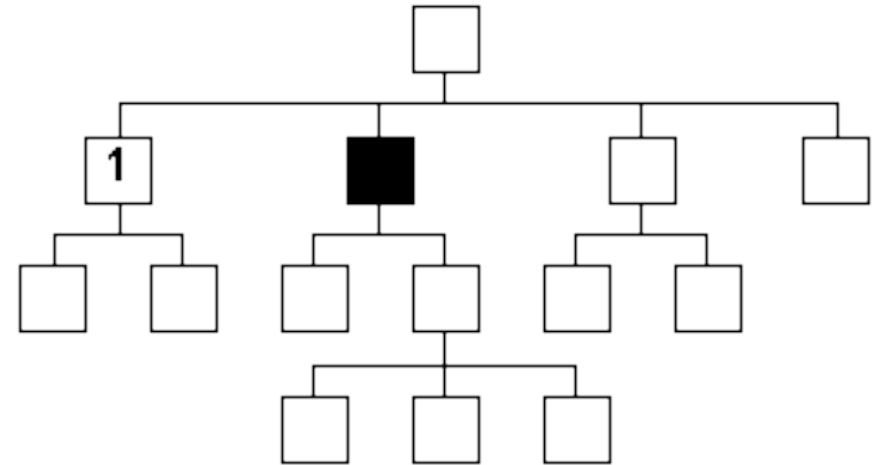
# Axis preceding-siblings::

- **preceding-siblings**

This selects all the nodes that precede the origin node, and that are children of the same parent node

- Syntax

`preceding-siblings::node`



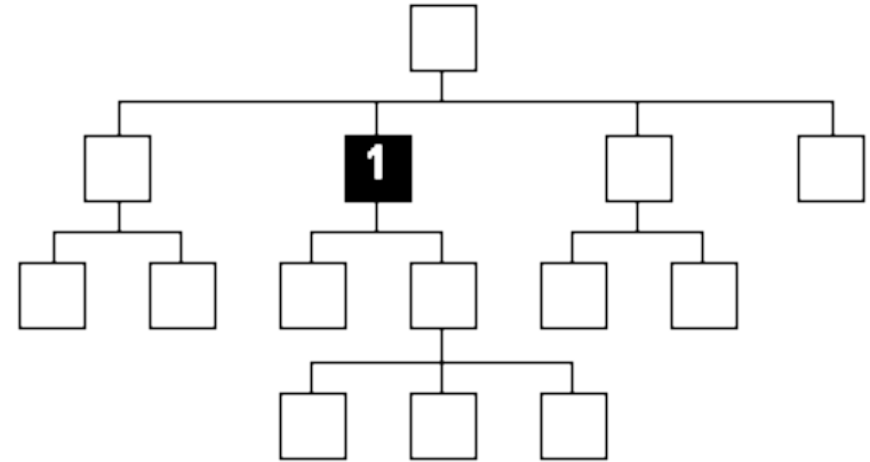
# Axis self:::

- **self**

This selects a single node, the origin node itself. This axis will never be empty.

## Syntax

1. self::node
2. .



# Node Tests

- A node test defines the nodes to select.
- Test the node in the tree document
  - **By name of node:** test the node to see if it has an element name the same as that specified. E.g. `child::Student` would test if the child node has an element named "Student"
  - **By kind/type of node:** test the node if is a text, comment, or processing instruction node. E.g. `text()`
  - **By the schema defined type**



# Node Tests: By Name

- selects nodes based on the node name

*	Match all elements
@*	Select all the attributes
xm:*()	Matches all element nodes in the namespace with the "xm" prefix
*:term	Any name matching the local name "term", regardless of namespace



# Node Tests: by Type

- selects nodes based strictly upon their node type
- In XPath 1.0

<code>node()</code>	True for a node of any type.
<code>text()</code>	True for a text node.
<code>comment()</code>	True for a comment node.
<code>processing-instruction()</code>	True for a processing instruction node.

- In XPath 2.0

<code>element()</code>	Matches any element node
<code>attribute("src")</code>	Matches any attribute named "src"
<code>item()</code>	Retrieves any item (node or atomic value)
<code>element("type")</code>	Matches any element node named "type"



# Node Tests: by Schema type

element(*, xs:date)	Any element of (simple) type xs:date
element (*, caption)	Matches any element node whose (schema) type is "caption" (or a type derived from "caption") User defined type.



# Predicates

- A predicate refers to the expressions (conditions) written in square brackets []. They restrict/filter the selected nodes in a node set.
  - Attribute Tests: @ indicates attribute
  - Boolean Tests (Functions): boolean, true, false, not, ...
  - Node Set Tests (Functions): count, id, position, last, ...
  - Number Tests (Functions): ceiling, floor, round, sum, ...
  - String Tests (Functions): concat, contains, string-length, substring, translate, ...
- There is no limit to the number of predicates in a step
  - Keywords (and, or), consecutive predicates [][]



# Path Operators and Special Characters

- XPath expressions are constructed using the following operators and special characters

/	Child operator, selects immediate children
//	Recursive descent, searches for the specified element at any depth
.	Indicates the current context (node)
..	The parent of the current context node
*	Wildcard, selects all elements regardless of the element name
@	Attribute, prefix for an attribute name
@*	Attribute wildcard, selects all attributes regardless of name
:	Namespace separator
()	Groups operations to explicitly establish precedence
[]	Applies a filter pattern



# Operators

- An XPath 1.0 expression returns either a node-set, a string, a Boolean, or a number.

" "	union operator, forms the union of two node-sets
"+", "-", "*", "div" (divide), "mod"	Arithmetic operators
"and", "or", "not() "	Boolean operators
"=", "!=", "<", ">", "<=", ">="	Comparison operators



# XPath Functions

- Functions to manipulate strings:
  - concat(), substring(), contains(), substring-before(), substring-after(), translate(), normalize-space(), string-length()
- Functions to manipulate numbers:
  - sum(), round(), floor(), ceiling()
- Functions to get properties of nodes:
  - name(), local-name(), namespace-uri()
- Functions to get information about the processing context:
  - position(), last()
- Type conversion functions:
  - string(), number(), boolean()



# Summary

- Selects (a set of) ELEMENTs within an XML document based on
  - Conditions
  - Hierarchy
- Usage
  - Retrieving info from a single XML document
  - Applying XSL style sheet rules
  - Making XQuerys



# Tutorial & Exercise

- <https://www.cs.tcd.ie/Owen.Conlan/php/xpath/xpathlab.html>
- Other XPath tools
  - XPath Checker – as Firefox addon
  - PathEnq – as Chrome plugin
- Form contents -
  - books.xml, books2.xml
  - booksTable.xsl OR booksList.xsl

