XPATH 

- Press `Space` to navigate through the slides
- Use `Shift+Space` to go back

INTRODUCTION TO 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 used in **XSL**

WHAT IS XPATH?

- Language for addressing parts of an XML document
    - Used in **XSLT**, **XQuery**
- A dedicated and powerful expression language for forming queries based on the tree structure of an XML document.
- It is used to:
    - Locate nodes in a tree
    - Extract information
    - Provide basic operations over data: e.g., manipulation of strings, numbers and booleans
- Compact, non-XML syntax for use within URIs and XML attribute values
- Operates on the abstract, logical structure of the XML document

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

XPath defines 7 node types

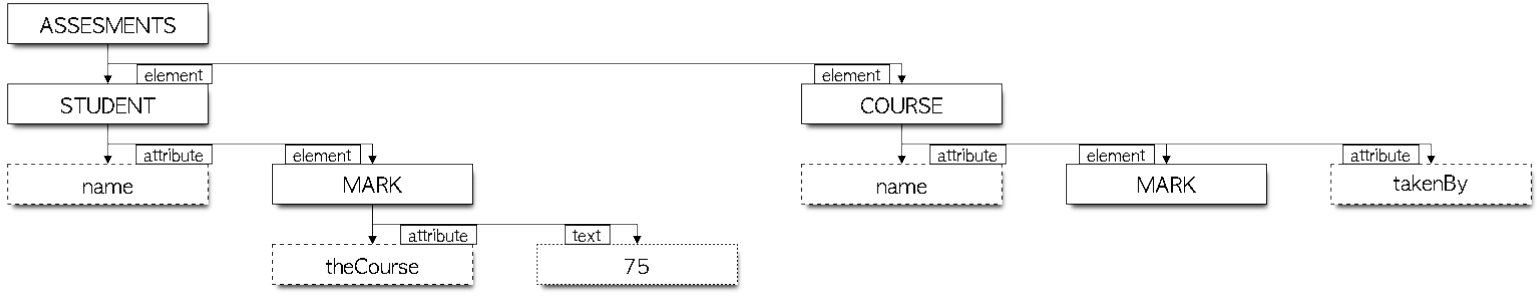
Root/Document Node	**The root of the tree representing the entire document contents, represented by the "/"**
Element	Element nodes are defined by pairs of start
Text	A character sequence in an element, comment, processing instruction, or namespace
Attribute	The name and value of an attribute in an element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

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`

EXAMPLE DOCUMENT

```
<ASSESSMENTS>
    <STUDENT name="Smith">
        <MARK theCourse="4BA1">75</MARK>
        <MARK theCourse="4BA5">99</MARK>
    </STUDENT>
    ...
    <COURSE name="4BA1" takenBy="Smith, Jones, ... ">
        <MARK>60</MARK>
    </COURSE>
...
</ASSESSMENTS>
```



USEFUL PROPERTIES OF A NODE

- **Name** (Except root, text and comment nodes)
    - Qualified by the namespace, such as xm:term (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**
    - number : stored as a floating point
    - string : a sequence of characters
    - boolean : a true or false value
    - 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.

LOCATION PATHS

- A location path (or path expression) identifies a set of nodes within an XML document
- A location path consists of a series of steps

- Simple path descriptors are sequences of location steps separated by slashes `/`
    - `bookstore/book`

- By default trying to match any child nodes from current location

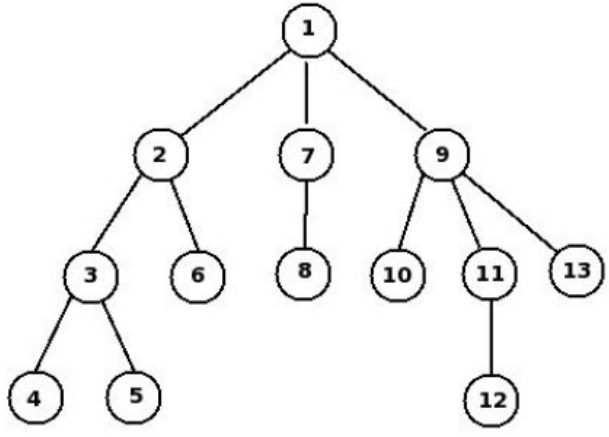
- Document Root:
    - A forward slash `/` at the start of a location path indicates that the starting position for the context node is the document `root` node

ABSOLUTE & RELATIVE PATHS

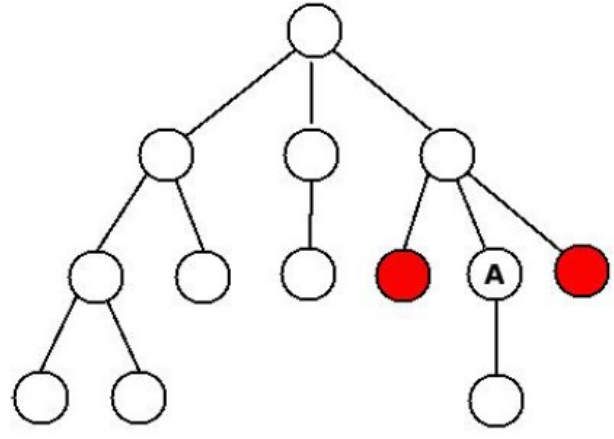
- A location path can be absolute or relative.
- If the location path starts with the root node (/) then you are using an absolute location path For example
    - `/root/node1/node2`
    - `/html/body/h3`
- If the location path begins with the name of a descendant, you're using a relative location path. For example,
    - `node1/node2`
    - `//node1/node2` (anywhere in document)

XML DOCUMENT NAVIGATION

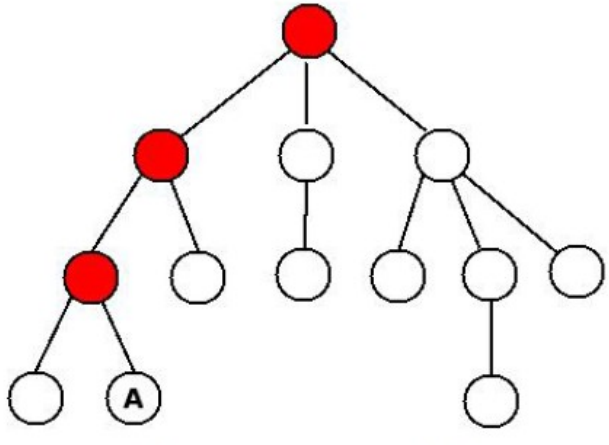
- The XPath data model treats an XML document as a tree of nodes, based on DOM
- Formally, a tree is a connected, acyclic, undirected graph



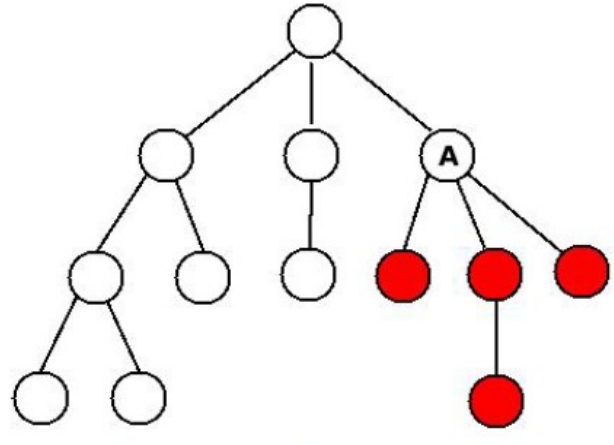
Document order



Siblings of A



Ancestors of A



Descendants of A

EXAMPLE: /ASSESSMENTS/STUDENT/MARK

```
<ASSESSMENTS>
    <STUDENT name="Smith">
        <MARK theCourse="4BA1">75</MARK>
        <MARK theCourse="4BA5">99</MARK>
    </STUDENT>
    ...
    <COURSE name="4BA1" takenBy="Smith, Jones, ... ">
        <MARK>60</MARK>
    </COURSE>
...
</ASSESSMENTS>
```

- Describes the set with two **MARK** element nodes in **STUDENT** as well as any other **MARK** elements nodes for any other **STUDENT**

In [397]:
```
r = tree.xpath('/ASSESSMENTS/STUDENT/MARK')
root = tree.getroot()
print(r)
```

```
[<Element MARK at 0x112e72b08>, <Element MARK at 0x112e72b48>]
```

NAMESPACES

- Any path expression can use a QNAME (prefix:localname),
    - 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)
- A path expression without a prefix will only match elements without an associated namespace

SKIPPING LEVELS

- A double forward slash `//` matches any descendent nodes below the current location
- For example:
    - `/section//cite`
        - will match all ’cite’ elements that are descendants of ’section’
    - `//author`
        - will match all ’author’ elements in the document

EXAMPLE: //MARK

```
<ASSESSMENTS>
    <STUDENT name="Smith">
        <MARK theCourse="4BA1">75</MARK>
        <MARK theCourse="4BA5">99</MARK>
    </STUDENT>
    ...
    <COURSE name="4BA1" takenBy="Smith, Jones, ... ">
        <MARK>60</MARK>
    </COURSE>
...
</ASSESSMENTS>
```

- Still returns nodes from the document with a node named **MARK** but this time not just those noted in student assessment statements e.g. a mark allocated to a course by an external examiner

In [399]:
```
r = tree.xpath('//MARK')
print(r)
```

```
[<Element MARK at 0x112e72b08>, <Element MARK at 0x112e72b48>, <Element MARK at
0x112e21588>]
```

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

EXAMPLE: /ASSESSMENTS/*

```
<ASSESSMENTS>
    <STUDENT name="Smith">
        <MARK theCourse="4BA1">75</MARK>
        <MARK theCourse="4BA5">99</MARK>
    </STUDENT>
    ...
    <COURSE name="4BA1" takenBy="Smith, Jones, ... ">
        <MARK>60</MARK>
    </COURSE>
...
</ASSESSMENTS>
```

- Return all nodes at first level of nesting in the document

In [420]:
```
r = tree.xpath('/ASSESSMENTS/*')
print(r)
```

```
[<Element STUDENT at 0x112e7fa88>, <Element COURSE at 0x112eab948>]
```

SELECT ATTRIBUTE @

- Attributes are referred to by putting ampersand `@` before the name
- Appear in the path as if nested within the tag
- For example `/book/@lang`
    - Select the ‘lang’ attribute of books

EXAMPLE: /ASSESSMENTS/*/@NAME

```
<ASSESSMENTS>
    <STUDENT name="Smith">
        <MARK theCourse="4BA1">75</MARK>
        <MARK theCourse="4BA5">99</MARK>
    </STUDENT>
    ...
    <COURSE name="4BA1" takenBy="Smith, Jones, ... ">
        <MARK>60</MARK>
    </COURSE>
...
</ASSESSMENTS>
```

- Select all `name` attributes appearing at first level of nesting

In [402]:
```
r = tree.xpath('/ASSESSMENTS/*/@name')
print(r)
```

```
['Smith', '4BA1']
```

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 then 35.00

EXAMPLE: /ASSESSMENTS/STUDENT[MARK > 80]

```
<ASSESSMENTS>
    <STUDENT name="Smith">
        <MARK theCourse="4BA1">75</MARK>
        <MARK theCourse="4BA5">99</MARK>
    </STUDENT>
    ...
    <COURSE name="4BA1" takenBy="Smith, Jones, ... ">
        <MARK>60</MARK>
    </COURSE>
...
</ASSESSMENTS>
```

- The parent element of **MARK** valued 99 is returned as it satisfies the condition

In [403]:
```
r = tree.xpath('/ASSESSMENTS/STUDENT[MARK > 80]')
print(r)
```

```
[<Element STUDENT at 0x112e7fa88>]
```

EXAMPLE:

/ASSESSMENTS/STUDENT/MARK[@THECOURSE = "4BA1"]

```
<ASSESSMENTS>
    <STUDENT name="Smith">
        <MARK theCourse="4BA1">75</MARK>
        <MARK theCourse="4BA5">99</MARK>
    </STUDENT>
    ...
    <COURSE name="4BA1" takenBy="Smith, Jones, ... ">
        <MARK>60</MARK>
    </COURSE>
...
</ASSESSMENTS>
```

- Any student mark objects for **4BA1**

In [408]:
```
r = tree.xpath('/ASSESSMENTS/STUDENT/MARK[@theCourse="4BA1"]')
print(r)
# print(r[0].text)
```

```
[<Element MARK at 0x112e72b08>]
```

ANATOMY OF A LOCATION STEP

- A step in an XPath expression consists of three parts:
    - an axis
        - Specifies direction to go in document tree
    - a node test
        - Tests whether nodes encountered should be selected
    - zero or more predicate tests
        - Filters nodes selected by the node test

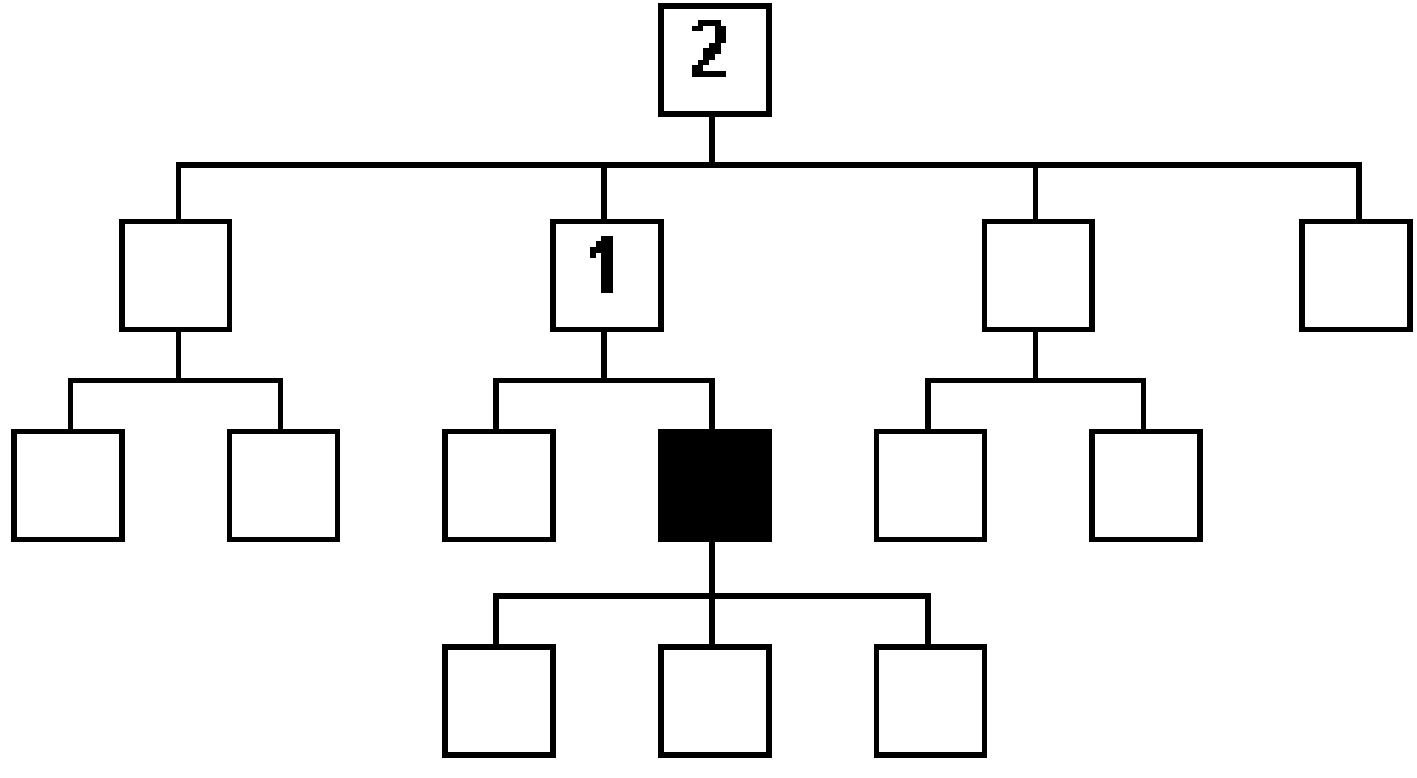
```
Child::Student[name="Smith"]
```

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****

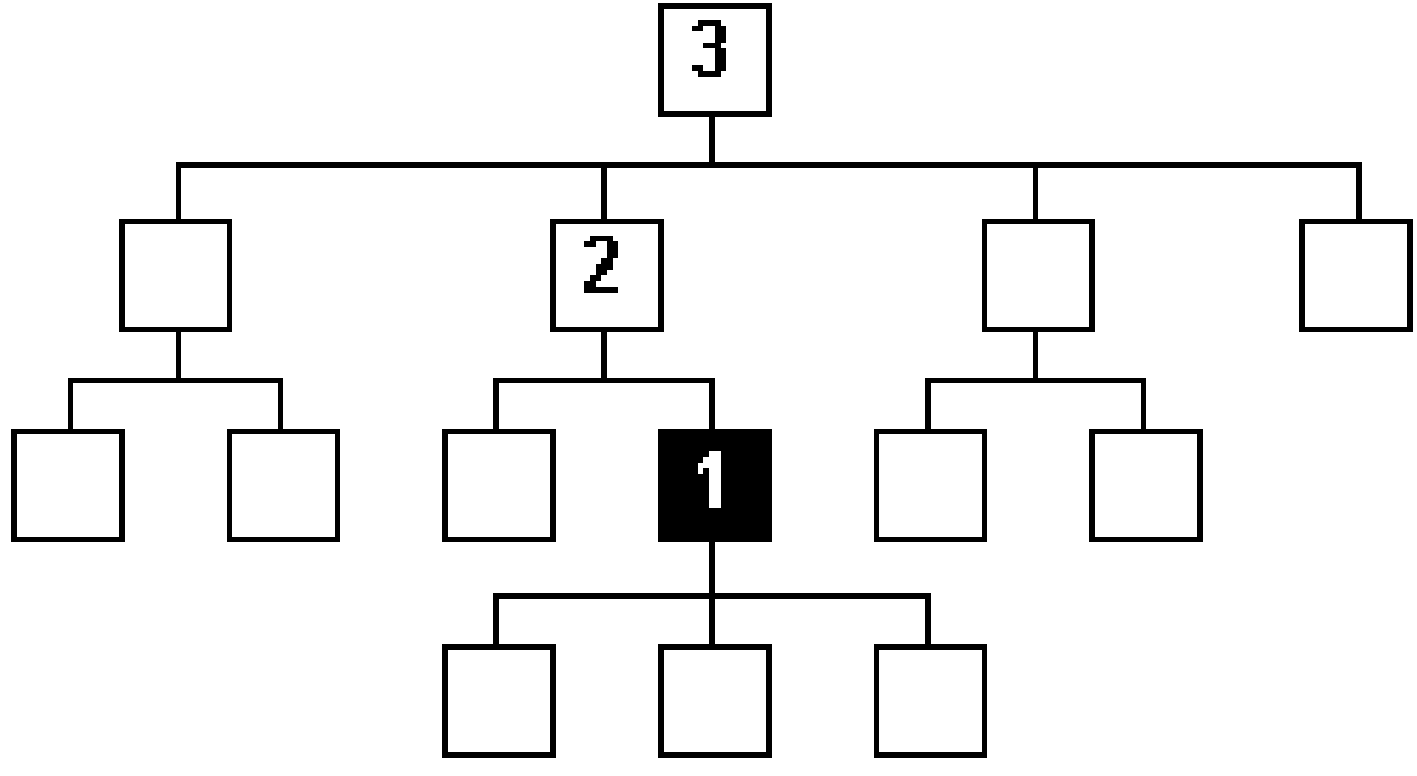
AXIS ANCESTOR::

- **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`

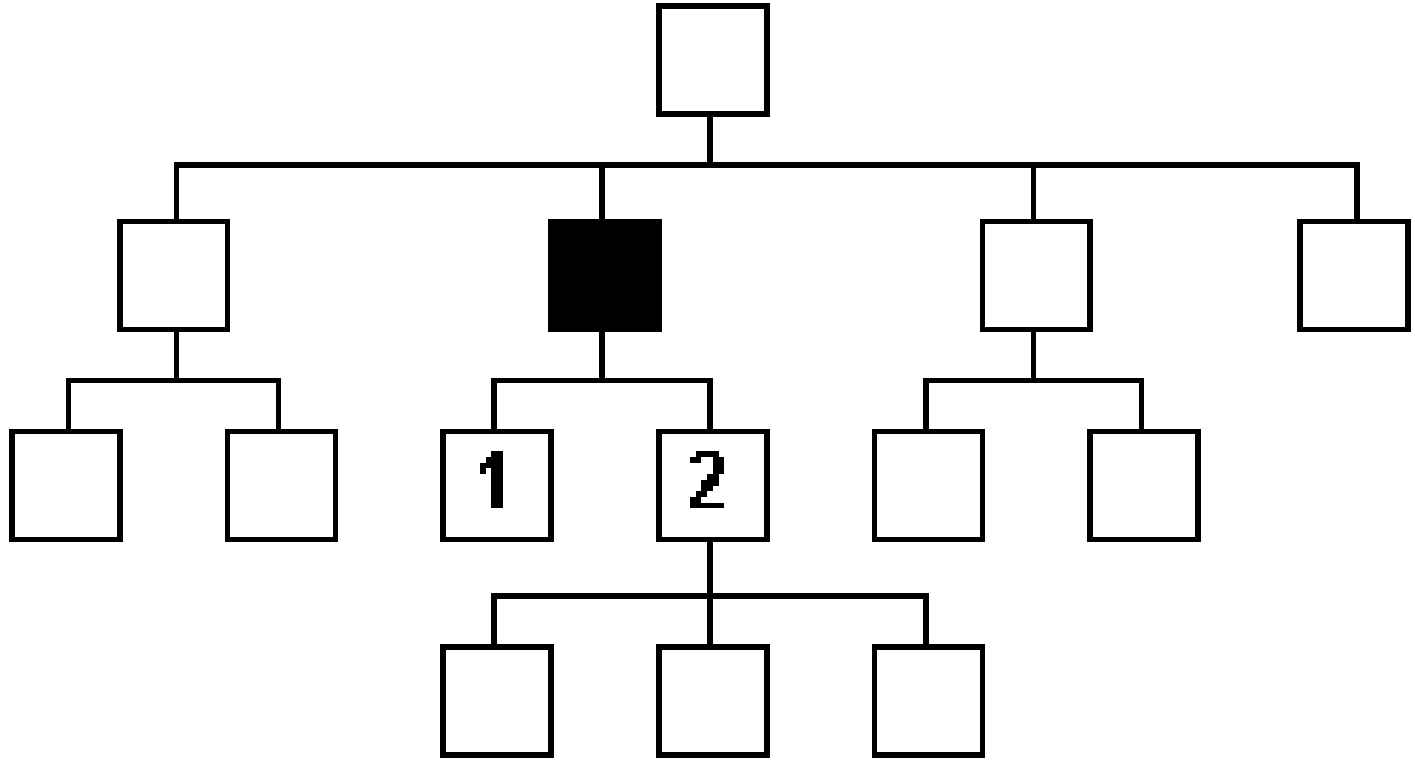


AXIS ATTRIBUTE::

- **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
    - `attribute::lang`
    - `@lang`

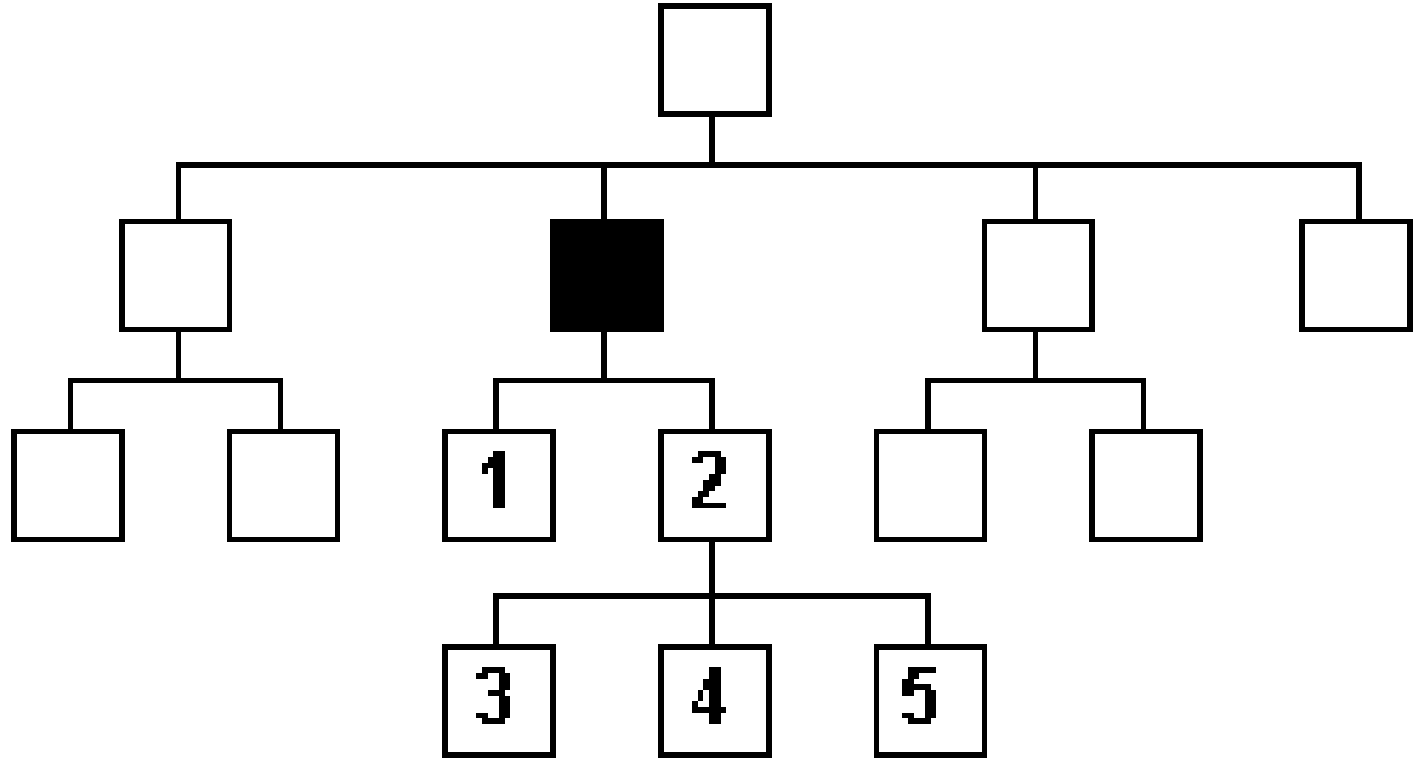
AXIS CHILD::

- **child**
    - Selects all the children of the origin node, in document order.
- Syntax
    - `child::node`
    - `/node`



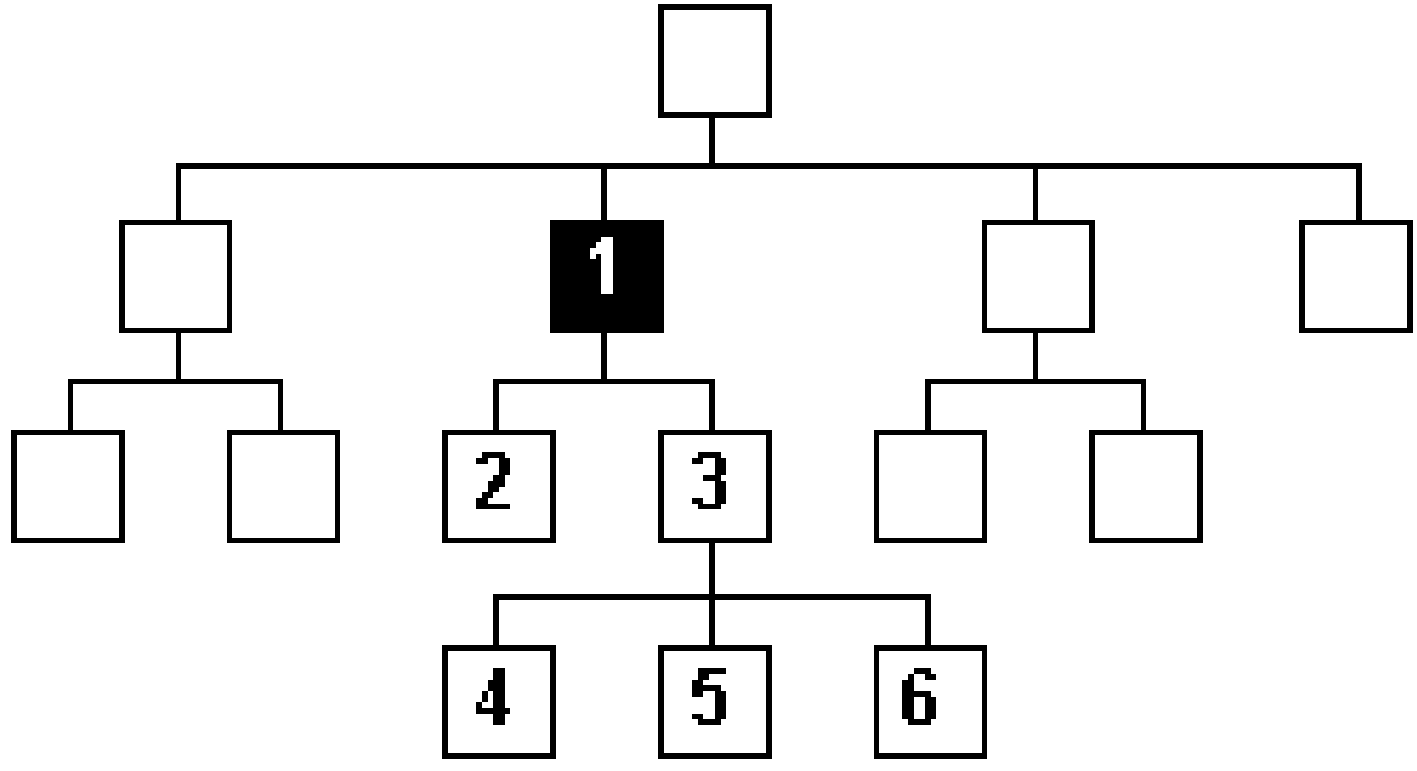
AXIS DESCENDANT::

- **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`



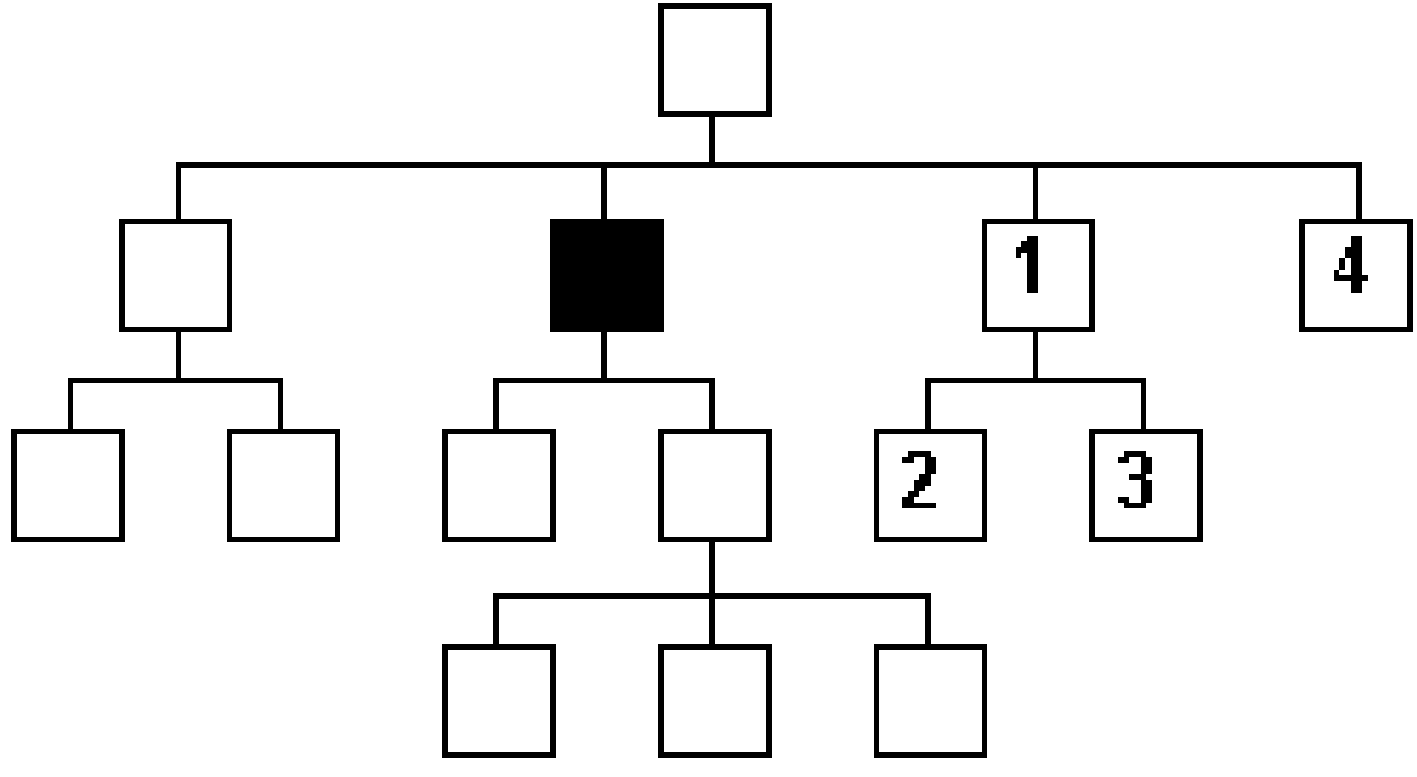
AXIS DESCENDANT-OR-SELF::

- **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-orself::node`
    2. `//`


1
2
3
4
5
6

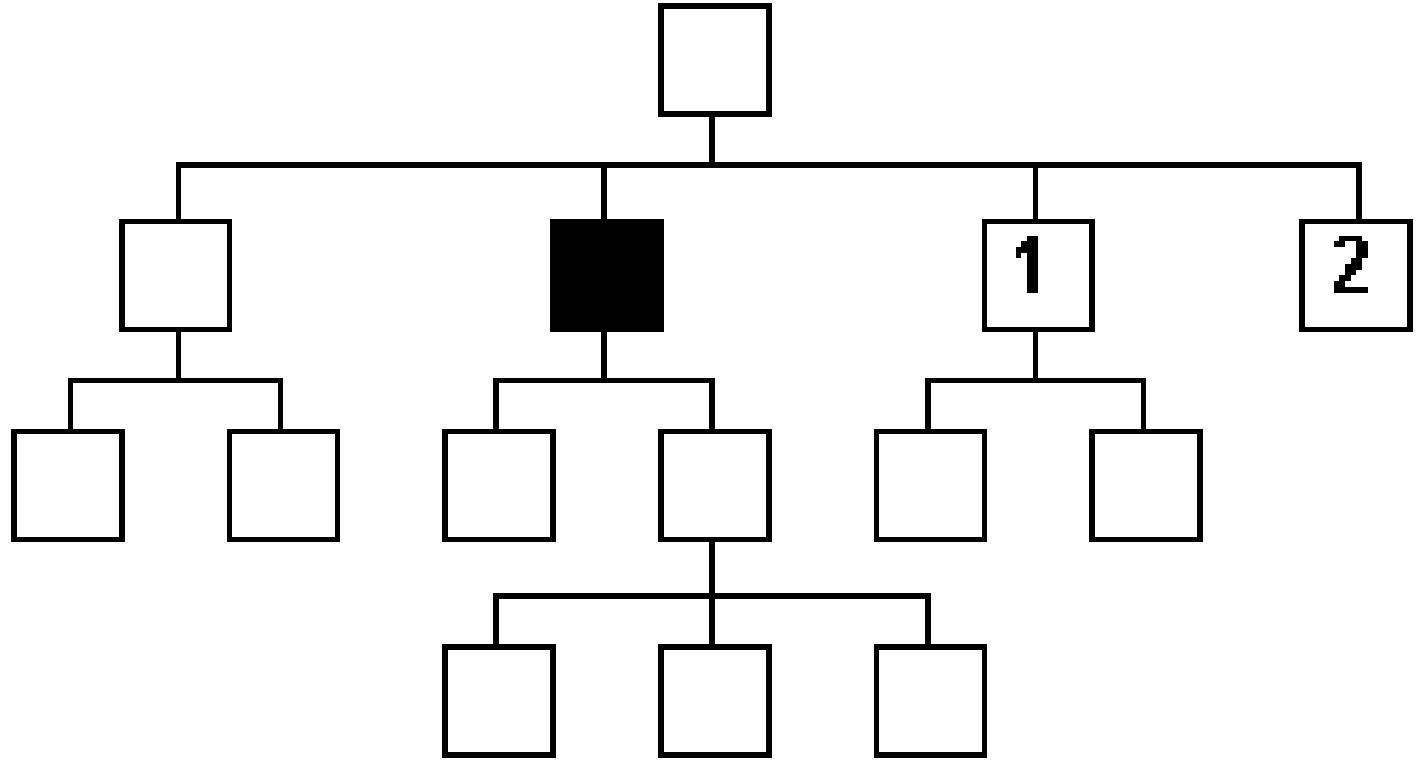
AXIS FOLLOWING::

- **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`

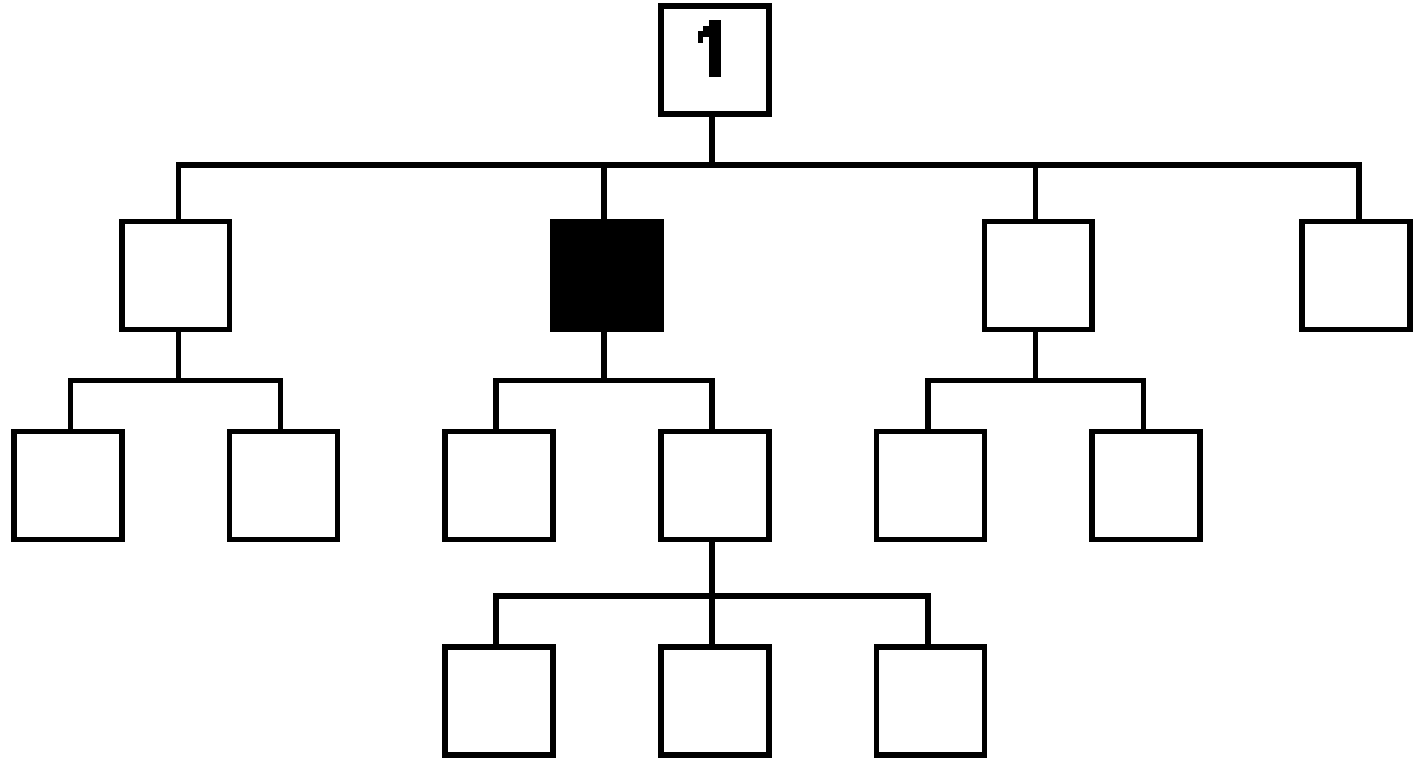


AXIS NAMESPACE::

- **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`

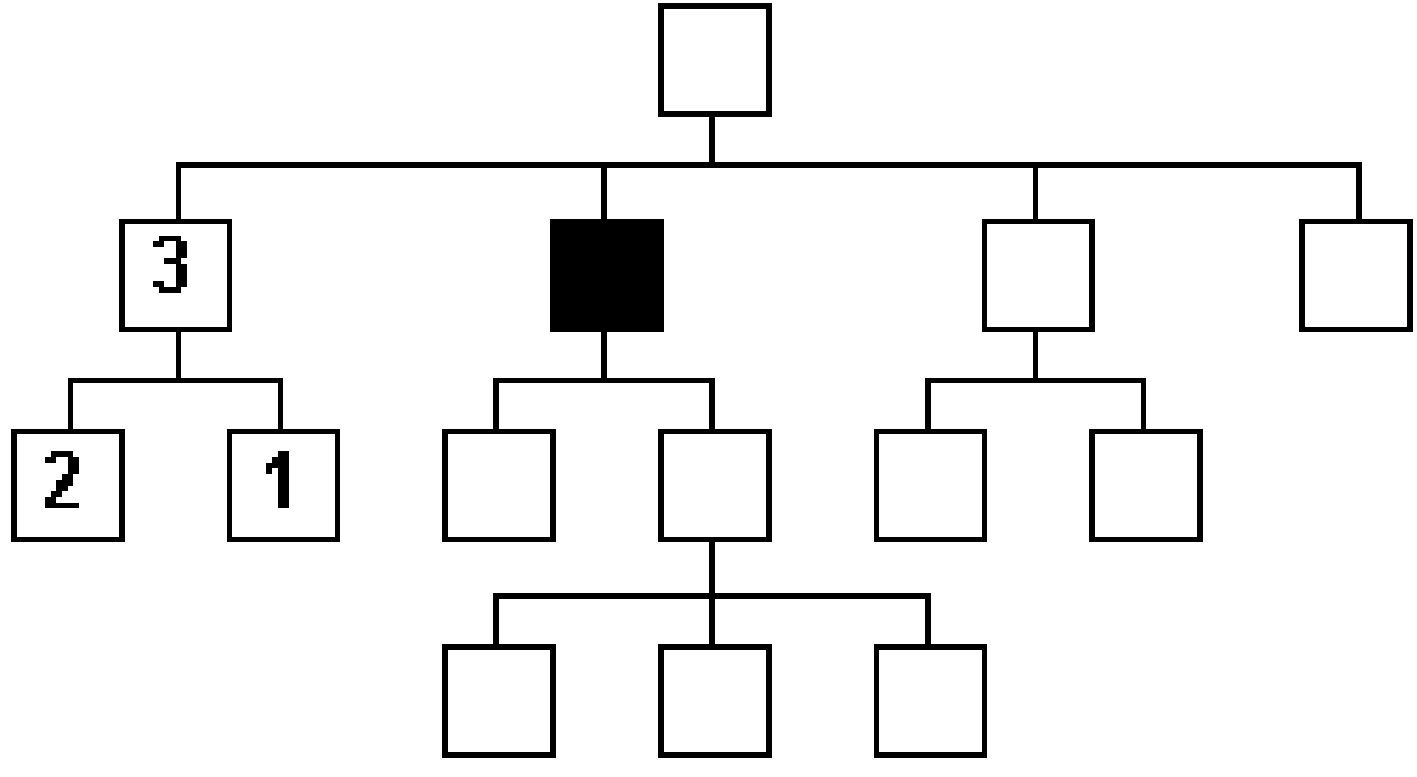
AXIS PARENT::

- **parent**
    - This axis selects a single node, the parent of the origin
- Syntax
    1. `parent::node`
    2. `..`



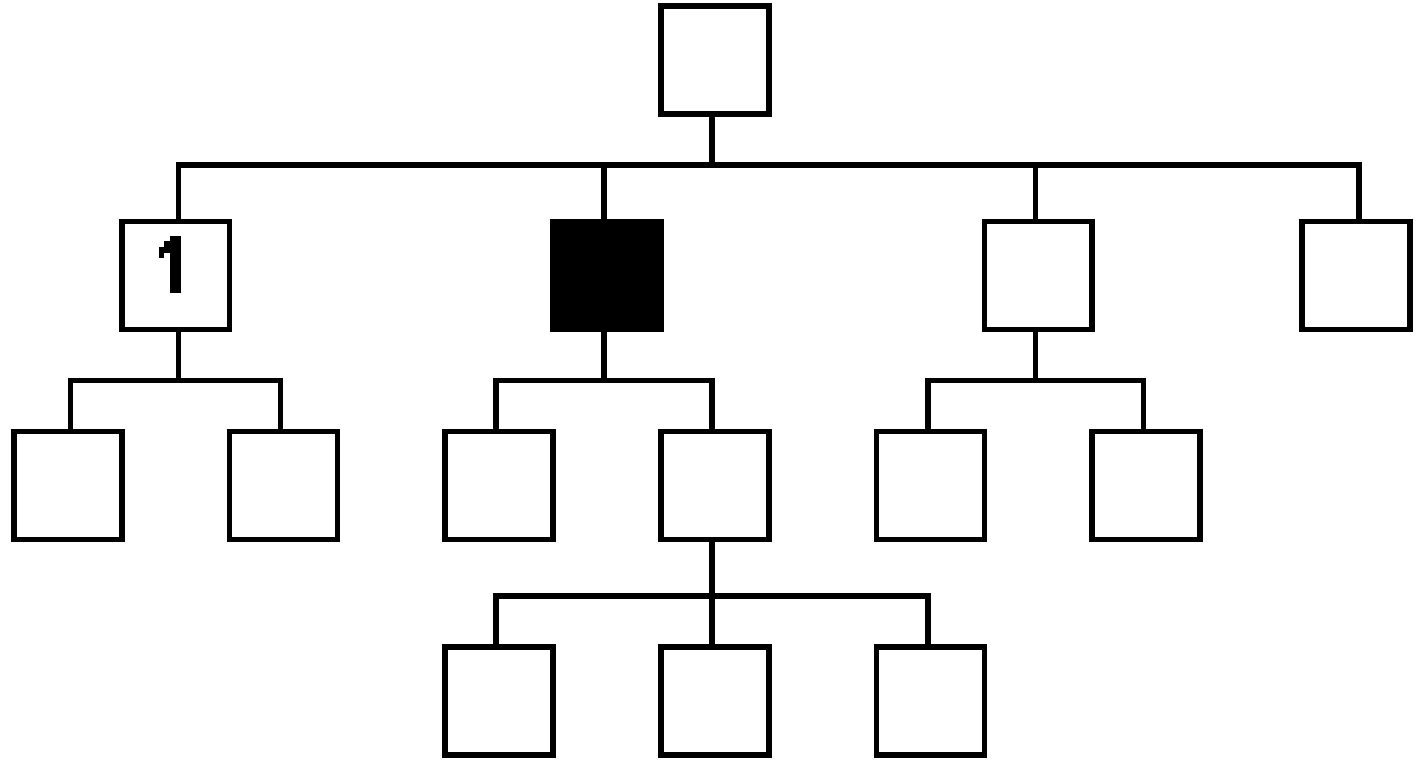
AXIS PROCEDING::

- **proceding**
    - This selects all the nodes that appear before the origin node, excluding the ancestors of the origin node.
- Syntax
    - `proceding::node`



AXIS PROCEDING-SIBLINGS::

- proceding-siblings
    - This selects all the nodes that precede the origin node, and that are children of the same parent node
- Syntax
    - proceding-siblings::node

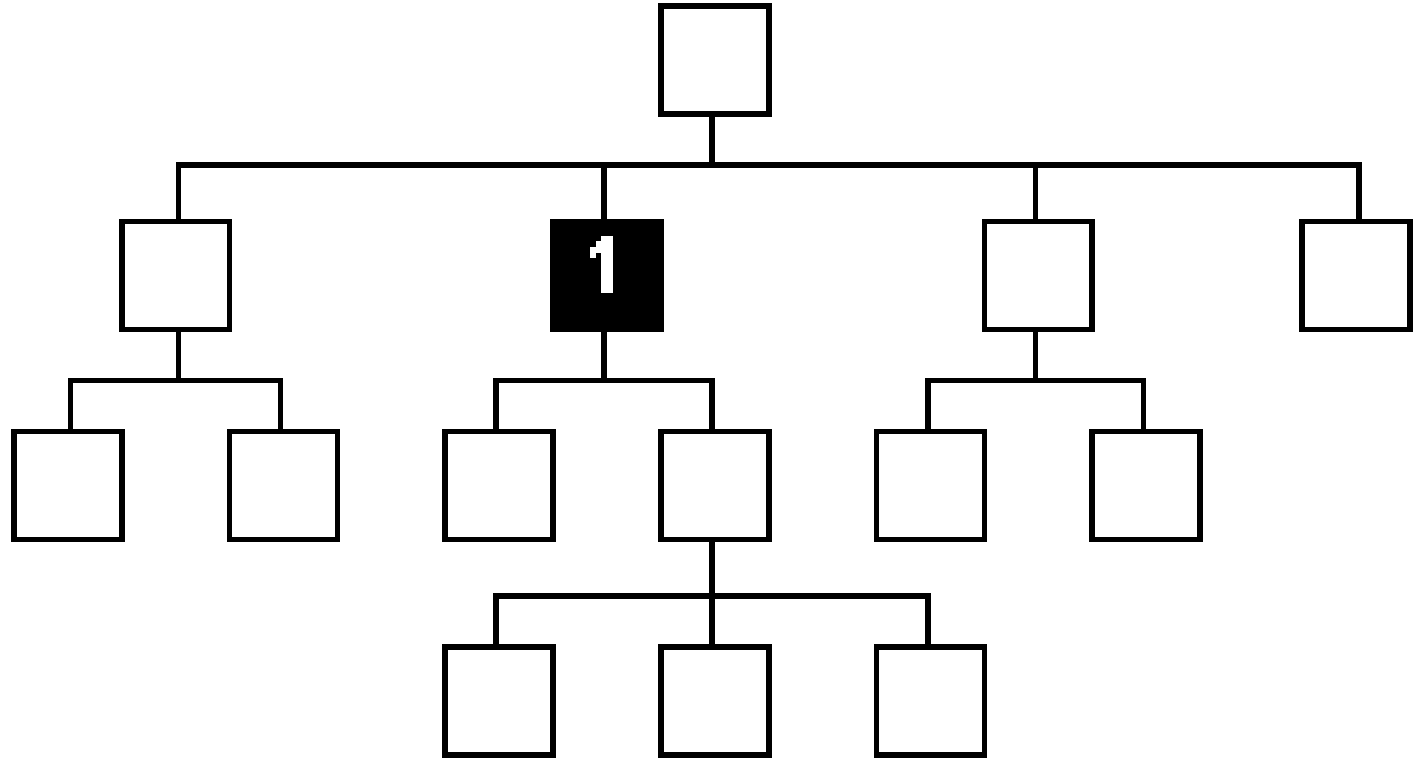


AXIS SELF::

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

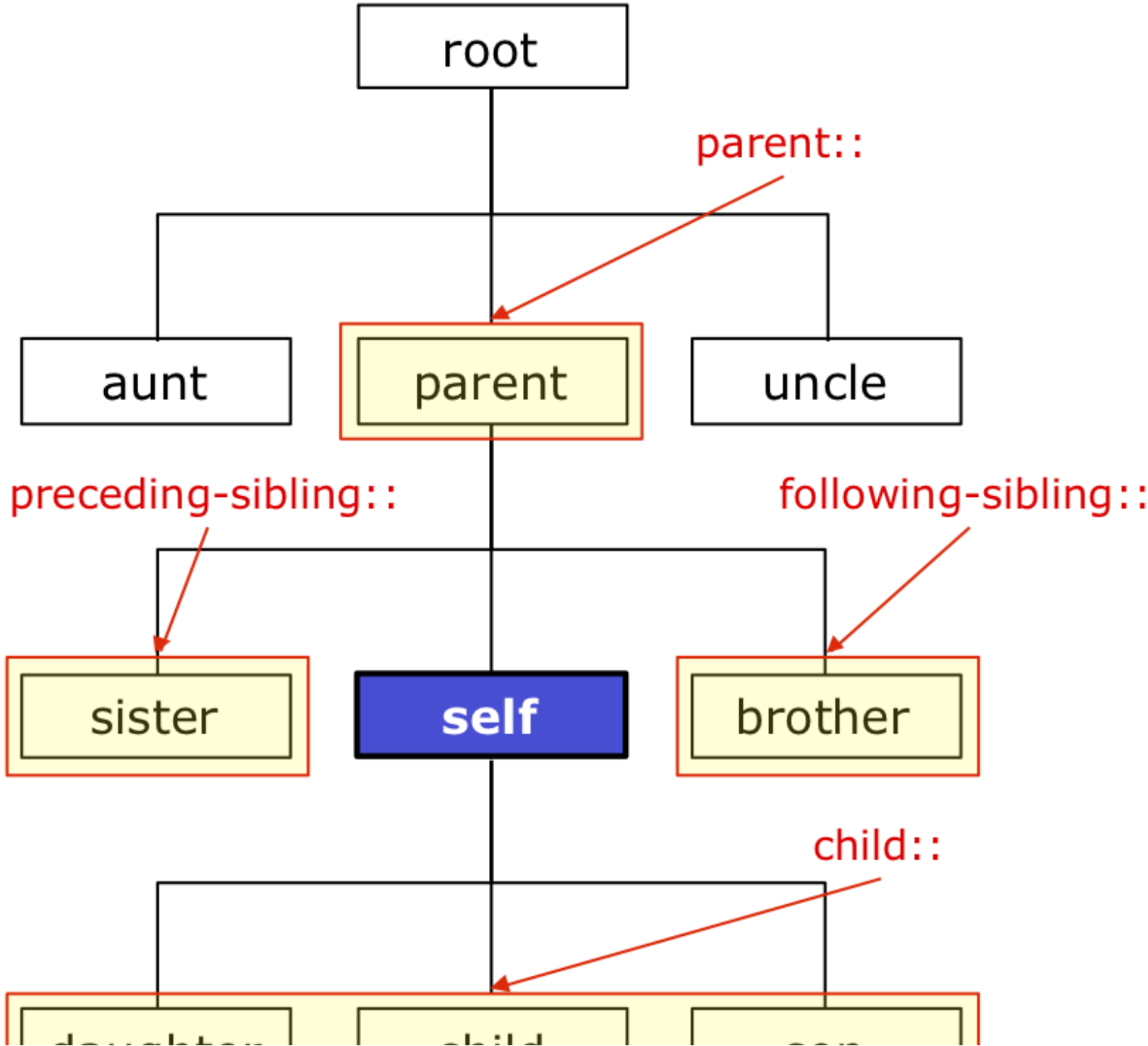
Syntax

1. `self::node`
2. `.`



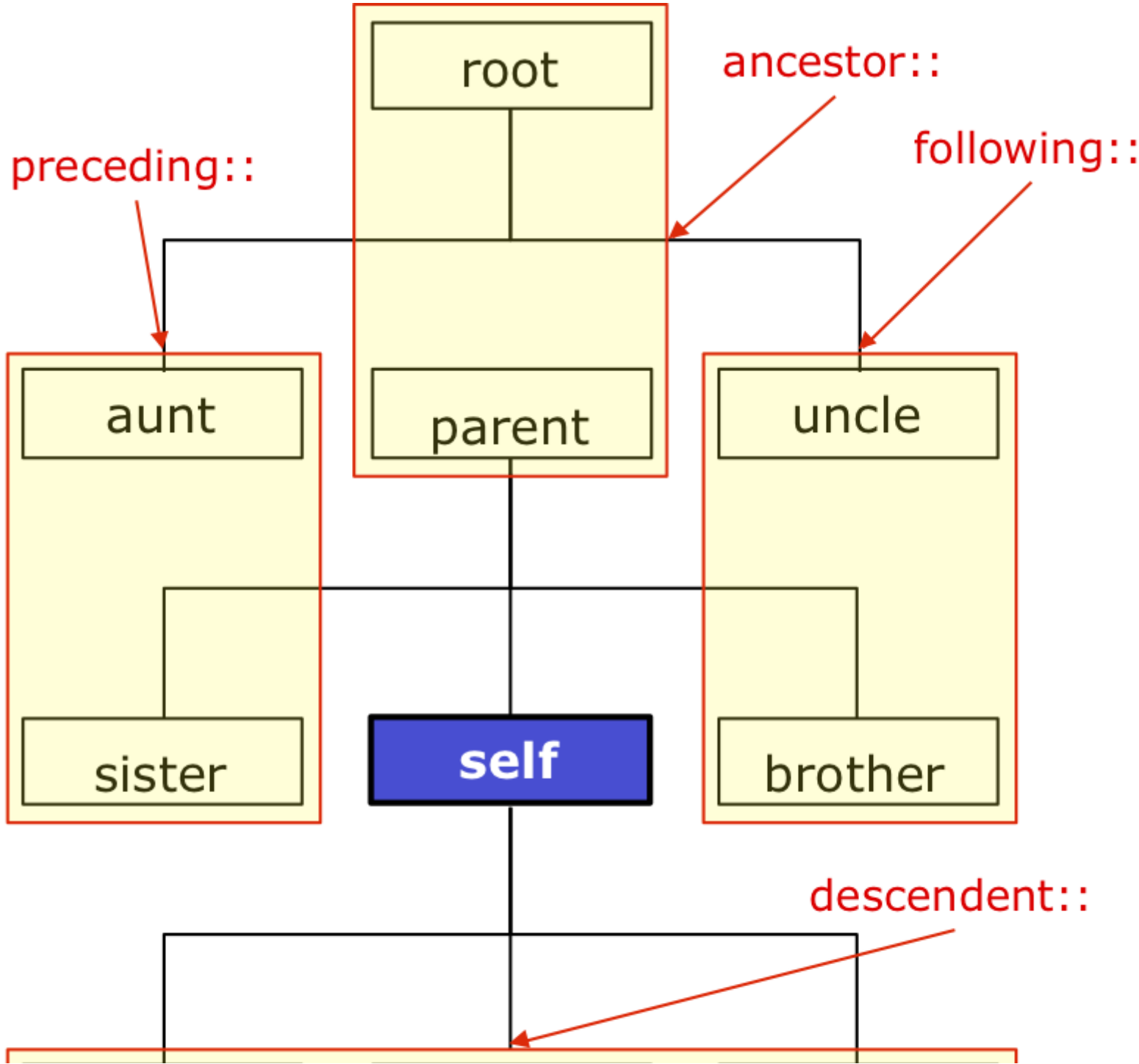
AXES EXAMPLE (1)

```
<root>
    <aunt />
    <parent>
        <sister />
        <self>
            <son>
                <grandchild />
            </son>
            <child />
            <daughter>
                <grandchild />
            </daughter>
        </self>
        <brother />
    </parent>
    <uncle />
</root>
```


root
parent::
aunt
parent
uncle
preceding-sibling::
following-sibling::
sister
self
brother
child::

AXES EXAMPLE (2)

```
<root>
    <aunt />
    <parent>
        <sister />
        <self>
            <son>
                <grandchild />
            </son>
            <child />
            <daughter>
                <grandchild />
            </daughter>
        </self>
        <brother />
    </parent>
    <uncle />
</root>
```


root
ancestor::
following::
preceding::
aunt
parent
uncle
sister
self
brother
descendent::

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

node()	True for a node of any type.
text()	True for a text node.
comment()	True for a comment node.
processing-instruction()	True for a processing instruction node.

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(), stringlength()
- 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()

XPATH EXAMPLES

```
<doc type="book" isbn="1-56592-796-9">
    <title>A Guide to XML</title>
    <author>Norman Walsh</author>
    <chapter>[...]</chapter>
    <chapter>
        <title>What Do XML Documents Look Like?</title>
        <paragraph>If you are [...]</paragraph>
        <paragraph>A few things [...]</paragraph>
        <ol>
            <item>
                <paragraph>The document begins [...]</paragraph>
            </item>
            <item>
                <paragraph type="warning">There's no document [...]</paragraph>
            </item>
            <item>
                <paragraph>Empty elements have [...]</paragraph>
                <paragraph>In a very[...]</paragraph>
            </item>
        </ol>
        <paragraph>XML documents are [...]</paragraph>
        <section>[...]</section>
        [...]
    </chapter>
</doc>
```

//ol//paragraph[@type=‘‘warning’’]

```
<paragraph type="warning">There's no document [...]</paragraph>
```

//paragraph

```
<paragraph>If you are [...]</paragraph>
<paragraph>A few things [...]</paragraph>
<paragraph>The document begins [...]</paragraph>
<paragraph type="warning">There's no document [...]</paragraph>
<paragraph>Empty elements have [...]</paragraph>
<paragraph>In a very [...]</paragraph>
<paragraph>XML documents are [...]</paragraph>
```

/doc/chapter[2]/ol/ittem[position()=last()]

```
<item>
  <paragraph>Empty elements have [...]</paragraph>
  <paragraph>In a very [...]</paragraph>
</item>
```

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 

HTTP://LEARN.ONION.NET/LANGUAGE=EN/35426/W3C-XPATH (HTTP://LEARN.ONION.NET/LANGUAGE=EN/35426/W3C-XPATH)