DTD :

Extension : dtd

3 TYPES OF DECLARATIONS :
- **DTD interne :**
```
<!DOCTYPE nom_racine [
  <!-- declarations -->
]>
```
- **DTD externe :**
```
<!DOCTYPE nom_racine SYSTEM "myDTD.dtd">
<!DOCTYPE nom_racine PUBLIC "-//W3C//DTD XHTML 4.0.1 Transitional//EN">
```
- **DTD mixte :**
```
<!DOCTYPE nom_racine SYSTEM "maDTD.dtd" [
  <!-- declarations -->
]>
```

DTD COMPONENTS :
- **ELEMENT DEFINITION :**
  - **SYNTAXE :**
```
<!ELEMENT nom type_de_donnee>
```
  - **nom :** nom de l'element
  - **type de donnée :**
    - **SIMPLE :**
      - **'ANY' :** tout type de donnée
      - **'EMPTY' :** ne contient pas de données spécifiques
      - **'(#PCDATA)' :** chaine de caractères
    - **COMPOUND :** A and B are elements
      - **'(A, B)' :** séquencement, B doit suivre A
      - **'A*' :** A peut apparaitre 0 ou plus
      - **'A+' :** A peut apparaitre au moins une fois
      - **'A?' :** A est optionnel
      - **'A|B' :** A xor B
  - **EXAMPLE :**
```
<!ELEMENT livre (auteur+)> :
    <livre>
      <auteur>
      </auteur>
      <auteur>
      </auteur>
    </livre>
<!ELEMENT br EMPTY> : <br/>
<!ELEMENT Adresse (#PCDATA | ville)*>
```

- **ATTRIBUT DEFINITION :**
  - **SYNTAXE :**
```
<!ATTLIST nom_element nom_attribut type_attribut mode_attribut|valeur_par_defaut>
```
  - **nom_element :** nom de l'element concerné
  - **nom_attribut :** nom de l'attribut
  - **type_attribut :**
    - **CDATA :** données texte
    - **NMTOKEN :** nom XML valide
    - **NMTOKENS :** plusieurs noms XML séparés par des espaces
    - **(val1|val2|...) :** liste des valeurs possibles pour l'attribut
    - **ID :** identificateur unique
    - **IDREF :** identificateur d'un autre élément
    - **IDREFS :** liste d'identificateurs d'autres éléments
    - **ENTITY :** nom d'une entité prédéfinie
    - **ENTITIES :** liste de noms d'entités
    - **NOTATION :** entités externe non XML
  - **mode_attribut :** informe si l'attribut doit être spécifié (#REQUIRED), s'il peut être omis (#IMPLIED) ou s'il peut être fixe (#FIXED). Pour ce dernier mode on lui associe aussi une constante
  - **valeur_par_defaut :** une valeur implicite pour l'attribut si aucune valeur ne lui a été affectée
  - **EXAMPLE :**
```
<!ATTLIST identification isbn CDATA #REQUIRED>
<!ATTLIST livre id ID #REQUIRED>
<!ATTLIST website author CDATA #FIXED "w3school">
```
- **NOTATION DEFINITION :**
  - **DEF :** Permettent de spécifier une application (ou plug-in) qui traitera le type de données spécifié dans la déclaration de la notation
  - **EXEMPLE :**
```
<!NOTATION flash SYSTEM "/usr/bin/flash.exe">
<!ENTITY animation SYSTEM "../anim.fla" NDATA flash>
<objet type="animation"/>
```
- **ENTITY DEFINITION :**
  - **DEF :** Alias peut-être considérée comme un alias permettant de réutiliser des informations internes ou externes au sein du document XML ou de la définition DTD
  - **PREDEFINIES :**
    - **< :** <
    - **> :** >
    - **& :** &
    - **" :** "
    - **' :** '
  - **TYPES :**
    - **Interne :** Définie à l'intérieur du document
    - **Externe :** Définie par URL
    - **Analysables :** contiennent un texte XML bien formé
    - **Non-analysables :** contiennent du texte non-XML ou des données binaires
  - **Combinaisons possibles :**
    - **Entités générales internes :**
      - **SYNTAXE :**
```
<!ENTITY nom_entity "valeur">
```
      - **EXEMPLE :**
```
<!ENTITY nom "valeur">
&nom;
```
    - **Entités générales externes :**
      - **SYNTAXE :**
```
<!ENTITY nom_entity SYSTEM "URI">
```
      - **EXEMPLE :**
```
<!ENTITY chap1 SYSTEM "chapitre1.xml">
&chap1;
```
    - **Entités de paramètres internes :**
      - **SYNTAXE :**
```
<!ENTITY %nom "valeur">
%nom;
```
      - **EXEMPLE :**
```
<!ENTITY %listeMarques "marque (Samsung|Apple) #REQUIRED">
<!ATTLIST telephone %listeMarques;>
```
    - **Entités de paramètres externes :**
      - **SYNTAXE :**
```
<!ENTITY %nom SYSTEM "URI"'>
%nom;
```

EXAM :
```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT gene (personne)+ >
<!ELEMENT personne (nom_p,parents?) >
<!ELEMENT nom_p (#PCDATA) >
<!ELEMENT parents EMPTY >
<!ATTLIST personne
  id ID #REQUIRED
  genre (M|F) #REQUIRED
>
<!ATTLIST parents
  ref IDREFS #REQUIRED
>
```

XSD :

Extension : xsd

Syntaxe :
```
<?xml version="1.0"?>
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- déclarations d'éléments, d'attributs et de types ici -->
</xsd:schema>
```

Reference a schema in an XML document :
- **EXAMPLE :**
  - **WITH NAMESPACE :** USE xsi:schemaLocation AND xs:targetNamespace
```
<?xml version="1.0"?>
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="URI_namespace note.xsd">
  </root>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xs:targetNamespace="URI_namespace">
```
  - **WITHOUT NAMESPACE :** USE xsi:noNamespaceSchemaLocation
```
<?xml version="1.0"?>
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="note.xsd">
  </root>
```

XS:SCHEMA :
- **DEF :** root element of every xml schema
- **SYNTAXE :**
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
</xs:schema>
```

XS:ELEMENT :
- **DEF :** each element of an xml document is described by this
- **2 METHODS TO DEFINE :**
  - **LOCAL :**
```
<xs:element name="x" type="y">
```
  - **BY REFERENCE :**
```
<xs:element ref="x">
```
- **ATTRIBUTS :**
  - **name :** element name
  - **type :** element type, there is 2 main types, default is 'xs:anyType'
  - **fixed :** fixed value or variable
  - **Default :** default value if element is present but empty
  - **minOccurs :** '0' means optional
  - **maxOccurs :** 'unbounded' means infinite
  - **nillable :** can be empty or not, 'false' by default

XS:ATTRIBUT :
- **DEF :** attributs of an element are declared inside its xs:element tag after all sub-elements
- **2 METHODS TO DEFINE :**
  - **LOCAL :**
```
<xs:attribute name="x" type="y" use="z">
```
  - **BY REFERENCE :**
```
<xs:attribute ref="x">
```
- **ATTRIBUTS :**
  - **name :** attribut name
  - **type :** attribut type, it's always a simple type
  - **fixed :** fixed value
  - **default :** default value
  - **use :**
    - #required
    - #optional
    - #prohibited

TYPING :
- **SIMPLE :** attribut declaration or elements with only atomic content without attributes
  - **SYNTAXE :**
```
<xs:simpleType name="x">
</xs:simpleType>
```
  - **BUILT-IN :** already defined in xml schema
    - **PRIMITIF :** simplest types
      - string
      - boolean
      - decimal
      - float
      - double
      - duration
      - dateTime
      - time
      - date
      - gYearMonth
      - gYear
      - gMonthDay
      - gDay
      - gMonth
      - hexBinary
      - base64Binary
      - anyURI
      - Qname
      - NOTATION
    - **DERIVED :** primitf type + some restrictions
      - token, ID, language...
      - integer, long, short...
      - ...
    - **SYNTAXE :**
```
<xs:element name="xxx" type="yyy"/>
```
  - **USER-DEFINED :** defined by user
  - **ATOMIC :** made from a single primitif or derived
    - **SYNTAXE :**
```
<xs:element name="xxx" type="userType_yyy"/>
```
  - **NON-ATOMIC :** made from a bunch of atomics
  - **METHODS TO DEFINE A TYPE :**
    - **BY RESTRICTION :** create new atomic types from primitifs by adding constraints
      - **CONSTRAINTES :**
        - **length, minLength, maxLength :** for string or list
        - **enumeration :** discrete set of values
        - **pattern :** regular expression
        - **whitespace :** preserve, replace or reduce whitespace in strings
        - **minInclusive, maxInclusive :** define interval, for numerical types
        - **minExclusive, maxExclusive :** define interval, for numerical types
      - **SYNTAXE :**
```
<xs:simpleType name="x">
  <xs:retriction base="y">
    <!-- constrainte -->
  </xs:restriction>
</xs:simpleType>
```
      - **EXEMPLE :**
```
<xs:simpleType name="Jours">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Lundi"/>
    <xs:enumeration value="Mardi"/>
    ...
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="Password">
  <xs:restriction base="xs:string">
    <xs:length value="8"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ISBN">
  <xs:restriction>
    <xs:pattern value="\d{2}\s+\d{2}\s+\d{2}\s+\d{2}\s+\d{2}"></xs:pattern>
  </xs:restriction>
</xs:simpleType>
```
    - **BY LISTE :**
      - **SYNTAXE :**
```
<xs:simpleType name="x">
  <xs:list itemType="xs:unsignedByte"/>
</xs:simpleType>
```
      - **EXEMPLE :**
```
<xs:simpleType name="Phone_number">
  <xs:restriction base="numéros">
    <xs:length value="5"/>
  </xs:restriction>
</xs:simpleType>
```
    - **BY UNION :**
      - **SYNTAXE :**
```
<xs:simpleType name="x">
  <xs:union memberTypes="y z"/>
</xs:simpleType>
```
      - **EXEMPLE :**
```
<xs:simpleType name="sizeType">
  <xs:union memberTypes="dressSizeType">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="aaa">
        <xs:enumeration value="bbb">
        <xs:enumeration value="ccc">
      </xs:restriction>
    </xs:simpleType>
  </xs:simpleType>
  <xs:simpleType name="ref_taille">
    <xs:union memberTypes="xs:positiveInteger xs:string"/>
  </xs:simpleType>
```

- **COMPLEX :** other cases
  - **SYNTAXE :**
```
<xs:complexType name="x" mixed="true|false">
  <!-- model -->
</xs:complexType>
```
  - **EMPTY CONTENT WITH ATTRIBUTS :**
    - **SYNTAXE :**
```
<xs:element name="xxx">
  <xs:complexType>
    <xs:attribute name=yyy" type="zzz"/>
  </xs:complexType>
</xs:element>
```
  - **SIMPLE CONTENT WITH ATTRIBUTS :**
    - **SYNTAXE :**
```
<xs:element name="xxx">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="yyy">
        <xs:attribute name="zzz" type="ttt"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```
  - **CONTENT WITH ELEMENTS & ATTRIBUTS :**
    - **TYPE :**
      - **<xs:sequence> :** ordered list of elements
      - **<xs:all> :** unordered list of elements
      - **<xs:choice> :** only one element should be present
    - **SYNTAXE :**
```
<xs:element name="xxx">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="yy1" type="zz1"/>
      <xs:element name="yy2" type="zz2"/>
      <xs:attribute name="yy3" type="zz3" use="required"/>
      <xs:attribute name="yy4" type="zz4" use="optional"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
  - **MIXED CONTENT :** same as previous, but with mixed="true"
    - **SYNTAXE :**
```
<xs:complexType mixed="true">
```

EXAM :
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:simpleType name="datNaiss">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{2}/[0-9]{2}/[0-9]{4}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="pays">
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z]{2,3}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="tel">
    <xs:restriction base="xs:string">
      <xs:pattern value="\+212[0-9]{9}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="nom" type="xs:string"/>
  <xs:element name="personnes">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="personne" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="nom"/>
              <xs:element name="prenom" type="xs:string"/>
              <xs:element name="dat_naiss" type="datNaiss"/>
              <xs:element name="telephone" type="tel"/>
              <xs:element name="adresse">
                <xs:complexType mixed="true">
                  <xs:sequence>
                    <xs:element name="ville" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID" use="required"/>
            <xs:attribute name="pays" type="pays" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XPATH :

XPATH : langage simple de requête qui permet de localiser des parties d'un doc

XPATH EXPRESSION : chemin de localisation constitué d'étapes/steps (séparés par /)
- **SYNTAXE :** STEP_1/STEP_2/.../STEP_N
- **TYPES OF PATH :**
  - **ABSOLUTE :** /STEP_1/...

RELATIVE : STEP_1/...
EXAMPLES :
- **'/' :** sélectionner la racine
- **'/A/B' :** tous les elements B fils de l'element A
- **'//A' :** tous les elements A
- **'|' :** combiner les requêtes
- **'*' :** tous les enfants du noeud contextuel
- **'A/@*' :** tous les attributs de l'élément A

STEP : chaque etape renvoie un ensemble de noeuds vers la suivante
SYNTAXE :
AXIS::NODE_TEST[PREDICATE]
ABBREVIATIONS :
- child::nom **<=>** nom
- attribute:: **<=>** @
- /descendant-or-self::node() **<=>** //
- self::node() **<=>** .
- parent::node() **<=>** ..

AXIS : direction dans laquelle on se dirige à partir du noeud courant
TYPES :
- self
- parent, ancestor, ancestor-or-self
- child, descendant, descendant-or-self
- preceding-sibling, following-sibling, preceding, following

NODE_TEST : designer le noeud concerné selon l'axe
FILTERS :
- **'nom' :** noeuds de l'axe qui portent ce nom
- **'*' :** les noeuds de type element
- **'text()' :** tous les noeuds de type text de l'axe
- **'comment()' :** les noeuds de type commentaire
- **'processing-instruction()' :** tous les noeuds de type instruction de traitement
- **'node()' :** tous les types de noeuds

TYPES :
- racine
- élément
- texte
- attribut
- espace de noms
- instruction de traitement
- commentaire

PREDICATE : Conditions à respecter
OPERATORS :
- **'=' :** equal
- **'!=' :** not equal
- **'<' :** less than
- **'>' :** greater than
- **'<=' :** greater than or equal
- **'AND' :** logical and
- **'OR' :** logical or

MOST USED :
- **'[i]' :** i-ème élément de la sélection
- **'[last()]' :** identifier le dernier élément de la sélection
- **'A[B]' :** localiser tout élément A enfant du noeud contextuel et ayant un élément enfant B
- **'A[@att]' :** Identifier tout élément A enfant du noeud contextuel et possédant un attribut att
- **'A[B='val']' :** Localise tout élément A enfant du noeud contextuel et ayant un élément enfant B possèdant comme valeur 'val'
- **'A[.='val']' :** Localise tout élément A enfant du noudh contextuel et ayant comme valeur 'val'
- **'A[@att='val']' :** condition sur les attributs

USEFUL FUNCTIONS :
- **'last()' :** Identifie le dernier élément de la sélection
- **'position()' :** Retourne l'indice de la position contextuelle d'un élément par rapport à son parent
- **'contains(chaine1, chaine2)' :** test si chaine1 contient chaine2
- **'name()' :** renvoie le nom du noeud contexte
- **'not(expression)' :** négation
- **'string-length(string)' :** retourne la longueur d'un string
- **'starts-with(string1, string2)' :** vrai si la première chaine commence par la deuxième
- **'count(ensemble de noeuds)' :** permet de tester le nombre de noeuds

EXAMPLES :
'/descendant::*' : tout les éléments descendants de la racine
'/xx/yy/descendant::*' : sélectionner tous les descendants de yy (fils de xx)
'//xx/descendant::*' : sélectionner tous les éléments qui ont xx comme ancêtre
'/*/*/*/xx' : sélectionne tous les éléments xx qui ont trois ancêtres
'//livre[last()]' : dernier livre
'//*[contains(name(), 'x')]' : éléments dont le nom contient 'x'

XSLT :

XSLT : Langage de description de transformations à opérer sur un arbre XML avec une syntaxe XML

STRUCTURE :
```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- transformation -->
</xsl:stylesheet>
```

LINK XML FILE TO AN XSLT :
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="my.xsl"?>
```

<xsl:template> :
SYNTAXE :
```
<xsl:stylesheet ...>
  <xsl:template match="xpath">
    <!-- xhtml tags + special xsl tags to populate with data -->
  </xsl:template>
</xsl:stylesheet>
```
ATTRIBUTES :
match : used to associate a template with an XML element or define a template for the entire document

<xsl:value-of> : print value of an element
SYNTAXE :
```
<xsl:trmplate ...>
  <xsl:value-of select="catalog/cd/title"/>
</xsl:template>
```
HINT :
when an xpath expression returns multiple elements, and you call <xsl:value-of> once, only the first element is treated

<xsl:for-each> : iterate through all elements
SYNTAXE :
```
<xsl:trmplate ...>
  <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
    <xsl:value-of select="artist"/>
  </xsl:for-each>
</xsl:trmplate>
```

<xsl:sort> : sort elements based on a criteria
SYNTAXE :
```
<xsl:for-each="catalog/cd">
  <xsl:sort select="artist"/>
  <xsl:value-of select="title"/>
</xsl:for-each>
```

<xsl:if> : if condition
SYNTAXE :
```
<xsl:if test="price &gt; 10">
  <!-- output if expression is true -->
</xsl:if>
```

<xsl:choose> : multiple conditions
SYNTAXE :
```
<xsl:choose>
  <xsl:when test="price &gt; 10">
    <!-- if -->
  </xsl:when>
  <xsl:when test="price &gt; 9">
    <!-- else-if -->
  </xsl:when>
  <xsl:otherwise>
    <!-- else -->
  </xsl:otherwise>
</xsl:choose>
```

<xsl:variable> :
SYNTAXE :
```
<xsl:for-each select="livre">
  <xsl:variable name="type" select="."/>
    <xsl:if test="contains($type, 'XML')">
      <tr>
        <td><xsl:value-of select="titre"></td>
        <td><xsl:value-of select="auteur"></td>
      </tr>
    </xsl:variable>
  </xsl:for-each>
```

<xsl:apply-templates> : apply a template to the current element's child nodes
SYNTAXE :
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
    <body>
    <h2>My CD Collection</h2>
    <xsl:apply-templates/>
    </body>
    </html>
  </xsl:template>

  <xsl:template match="cd">
    <p>
      <xsl:apply-templates select="title"/>
      <xsl:apply-templates select="artist"/>
    </p>
  </xsl:template>

  <xsl:template match="title">
    Title: <span style="color:#ff0000">
    <xsl:value-of select="."/></span>
    <br />
  </xsl:template>

  <xsl:template match="artist">
    Artist: <span style="color:#00ff00">
    <xsl:value-of select="."/></span>
    <br />
  </xsl:template>

</xsl:stylesheet>
```

- The first apply-templates calls the cd template each time an element named "cd" is encountered.
- The cd template, in turn calls the title and artist templates to process the children elements of <cd>.
- title is processed before artist. Note, that the order of artist and title elements in the source XML makes no difference.

APPLY TEMPLATE MULTIPLE TIMES WITH MODE :
```
<xsl:template match="/">
  <xsl:apply-templates select="//livre" mode="auteur"/>
  <xsl:apply-templates select="//livre" mode="lang">
</xsl:template>

<xsl:template match="//livre" mode="auteur">
  <!-- traitement 1 -->
</xsl:template>

<xsl:template match="//livre" mode="lang">
  <!-- traitement 2 -->
</xsl:template>
```

EXAM :
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="xs"
  version="2.0">
  <xsl:output method="html" />
  <xsl:template match="/">
    <h1>liste des personne plus agees</h1>
    <xsl:for-each select="//personne">
      <xsl:if test="dat_naiss &lt; '13/01/2001'">
        <p><xsl:value-of select="nom"/></p>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

XQUERY :

XQuery : XQuery is designed to query XML data

HEADER : HEADER : **xquery version "1.0";**

doc("books.xml") function : used to open an xml file
SYNTAXE : XQuery uses Xpath expressions
doc("books.xml")/bookstore/book/title
doc() opens books.xml file, bookstore selects the root, /book selects all book elements under bookstore
```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

TYPE OF EXPRESSIONS :
XPATH : //author
FLWOR : For, Let, Where, Order By, Return
TESTS : If, Then, Return, Else, Return
FUNCTIONS : Root, predefined, to be defined

FLWOR expressions :
FLWOR ACRONYM :
FOR : selects a sequence of nodes
LET : binds a sequence to a variable
WHERE : filters the nodes
ORDER BY : sorts the nodes
RETURN : what to return (gets evaluated once for every node)
EXAMPLE :
```
doc("books.xml")/bookstore/book[price>30]/title
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
= - - - - - - =
for $d in doc("bib.xml")//book
  let $e := doc("biblio.xml")//book[.=$d]
  where $e/price[.>40]
  order by $e/title[.] ascending
  return $e/title
```
```
<title>Advanced Programming in the Unix environment</title>
<title>TCP/IP Illustrated</title>
```

IF-THEN-ELSE :
EXPRESSIONS :
GENERAL COMPARISONS : =, !=, <, <=, >, >=
VALUE COMPARISONS : eq, ne, lt, le, gt, ge
DIFFERENCE :
'$bookstore//book/@q > 10' : true if any q attributes have a value greater than 10
'$bookstore//book/@q gt 10' : true if there is only one q attribute returned by the expression,
and its value is greater than 10
EXAMPLE :
```
or $x in doc("books.xml")/bookstore/book
return if ($x/@category="children")
  then <child>{data($x/title)}</child>
  else <adult>{data($x/title)}</adult>
```

ITERATION :
EXAMPLE :
```
for $x in (1 to 3)
return <test>{$x}</test>
```
```
<test>1</test>
<test>2</test>
<test>3</test>
```

AT :
EXAMPLE :
```
for $x at $i in doc("books.xml")/bookstore/book/title
return <book>{$i}. {data($x)}</book>
```
```
<book>1. Everyday Italian</book>
<book>2. Harry Potter</book>
```

MULTIPLE-VARS :
EXAMPLE :
```
for $x in (10,20), $y in (100,200)
return <test>x={$x} and y={$y}</test>
```
```
<test>x=10 and y=100</test>
<test>x=10 and y=200</test>
<test>x=20 and y=100</test>
<test>x=20 and y=200</test>
```

ELEMENT CONSTRUCTIONS :
CONTENT ONLY :
EXAMPLE :
```
<auteur>
  {doc("biblio.xml")//book[3]/author/last}
</auteur>
```
```
<auteur>
  <last>Abitebool</last>
  <last>Buneman</last>
  <last>Suciu</last>
</auteur>
```
```
= - - - - - - =
<ul>
{
  for $x in doc("books.xml")/bookstore/book/title
  order by $x
  return <li>{data($x)}</li>
}
</ul>
```

ALL :
EXAMPLE :
```
element {doc("biblio.xml")//book[1]/name(*[1])} {
attribute {doc("biblio.xml")//book[1]/name(@*[1])} {
  doc("biblio.xml")//book[1]/@*[1]
},
doc("biblio.xml")//book[1]/*[1]/text()
}
```
```
<title year="1994">TCP/IP Illustrated</title>
```

JOINS :
EXAMPLE :
```
for $p1 in
doc("exam2017Ex2.xml")/personnes/personne, $p2 in
doc('exam2017Ex1.xml')/gene/personne
return
  if ($p1/@id=$p2/@id and $p2/parents[contains(@ref, 'a1')])
  then <nom>{data($p1/nom)}</nom>
  else()
= - - - - - - =
for $p1 in
doc("exam2017Ex2.xml")/personnes/personne, $p2 in
doc('exam2017Ex1.xml')/gene/personne
return
  if ($p1/@id=$p2/@id and not($p2/parents[contains(@ref, 'a1')]))
  then <nom>{data($p1/@id)}</nom>
  else()
```

FUNCTIONS :
PREDEFINED :
SQL-LIKE : min, max, count ...
NUMERICAL : round, floor, ceiling ...
STRING : string-length, starts-with, upper-case, lower-case ...
OTHER : distinct-values, doc, not, empty, exists ...
EXAMPLE :
```
let $b := doc("bib.xml")//book
let $avg := avg($b//price)
return $b[price > $avg]
= - - - - - - =
for $b in doc("books.xml")//book where
not(some $a in $b/author satisfies $a/
last="Stevens") return $b
= - - - - - - =
for $b in doc("books.xml")//book where
not(empty($b/author)) return $b
```