



# Урок 4

## План заняття:

1. Введення в стандарти XSL, XSLT, XSL-FO. Короткий огляд видів трансформації XSLT
2. Адресація на мові XPath. Поняття вузла та виразів XPath (синтаксис та приклади написання)
3. Кроки вибірки
  - 3.1. Основні засади
  - 3.2. Вісь навігації
  - 3.3. Тести (перевірки) вузлів
  - 3.4. Предикати
  - 3.5. Скорочений синтаксис
4. Операції та оператори в XPath
5. Функції XPath
6. Приклади побудови логічних виразів та виразів порівняння на мові XPath
7. Домашнє завдання

## 1. Введення в стандарти XSL, XSLT, XSL-FO. Короткий огляд видів трансформації XSLT

В мові HTML/XHTML для оформлення сторінок використовуються таблиці стилів CSS (Cascading Style Sheet). Вони дозволяють задавати загальні правила їх вигляду: колір тексту і фону, розміри та стиль шрифтів в документі, вигляд абзаців (параграфів) тощо. Використовуючи цей набір правил, Ви задаєте єдиний стиль оформлення для Web-сторінки.

Ця ідея набула поширення і не могла бути не реалізована в XML. Це призвело до того, що разом з документами XML також можна було використовувати таблиці стилів CSS (див. урок 1). Для цього в мову XML була введена інструкція по обробці stylesheet. Але не дивлячись на таку можливість, документи XML оформлені за допомогою CSS виглядали не досить привабливо. Крім того, стилі CSS визначають способи показу документа в вікні браузера, його візуалізацію, а мова XML дозволяє визначити структуру документа, нічого не говорячи про його представлення в зручному для читання вигляді. Отже, реалізація стилів для XML повинна бути іншою.

Виходячи з вищесказаного в технології XML для запису стилів була розроблена спеціальна мова **XSL (XML Stylesheet Language)**. Мова XSL представляє документ в вигляді дерева. Процесор мови XSL перетворює це дерево, керуючись таблицею стилів, і форматує його для виводу в вікно браузера, на екран, принтер тощо. Таким чином обробка здійснюється в два етапи:

1. конвертування дерева документа (XML transform);
2. форматування (formatting) дерева, отриманого після конвертування.

Дуже скоро після виходу рекомендацій по XSL стало зрозуміло, що конвертування документів XML – це окрема стихія, самостійна та незалежна задача. Це пов'язано з тим, що етап конвертування може бути доволі складним і кардинально змінити структуру дерева: видалити або додати нові вузли, змінити порядок їх слідування, додати заголовки тощо. Результатом конвертування може стати зовсім новий документ або набір документів, причому які можуть мати відмінний від xml формат (.doc, .pdf, .html тощо). Тому конвертування дерева документів XML було виділено в окрему область дослідження і описано окремою мовою **XSLT (XML Stylesheet Language for Transformation)**, перша версія якої побачила світ в 1999 році.

Після відділення мови XSLT первинна рекомендація мови XSL була підправлена і на даний час основний акцент в ній зроблений на форматування, тому її часто називають **XSL-FO (formatting object – об'єкти форматування)**. Отже, після такого розділення XSLT стала відповідати за конвертування вхідного документа, а XSL – за візуальне відображення результату цього конвертування. На сьогоднішній день ці дві мови розвиваються незалежно одна від одної.

Оскільки вивчення XSL-FO не входить до нашого плану вивчення основ XML, то зачіпати ми її не будемо, а зробимо основний акцент на XSLT.

Отже, XSLT на сьогоднішній день представлена трьома версіями: XSLT 1.0, XSLT 1.1 та XSLT 2.0. Відмінностей між цими версіями багато, але спробуємо перелічити основні:

- Різниця між XSLT 1.0 та XSLT 1.1:
  - В XSLT 1.1 відсутній тип даних result tree fragment (результуючий фрагмент дерева) попередньої версії. Замість нього використовується набори (множини) вузлів, які складаються з єдиного кореневого вузла результуючого фрагменту.
  - Для здійснення трансформації можна використовувати кілька вхідних документів XSLT. В версії XSLT 1.0 був лише один вхідний і один вихідний документ.
  - Доданий елемент xsl:script, який надає додаткові можливості для створення і використання власних функцій.
  - Можна використовувати зовнішні типи даних. Наприклад, в XSLT-процесорах, написаних на Java, можна використовувати як розширення власні класи.



- Додана розширена підтримка просторів імен при конвертуванні;
- виправлено ряд помилок попередньої версії тощо.
- Різниця між XSLT1.1 та XSLT 2.0:
  - Можна змінювати простір імен по замовчуванню для шаблонів;
  - Включили функції для форматування дати і часу;
  - Включена підтримка зовнішніх сутностей, які представлені у вигляді тексту, а також додана підтримка параметризованих сутностей;
  - Підтримка Unicode-рядків тощо.

Як ми вже згадували раніше, XSLT дозволяє конвертувати XML документи в інші типи документів, зокрема в такі як RTF, PDF, SQL, WML, SWF, HTML, в векторні зображення типу SVG, а також в інші XML документи. Отже, для цих цілей XSLT підтримує **три види трансформації XML документів**:

1. xml – xml. Конвертування з XML документа в XML;
2. xml – text. Конвертування з XML документа в текстовий документ;
3. xml – html. Конвертування з XML документа в HTML.

Щодо останніх двох, то тут особливо пояснювати непотрібно. Перший же вид трансформації призначений в основному для бізнес-систем, коли у Вас на вході та на виході повинні існувати дані в XML форматі, але, які мають різну структуру. Для прикладу, повернемось до наших Олі та Васи Пупкіних та розглянутої нами на першій парі їх ситуації з XML документами. Коротко нагадаємо, що вони вирішили написати якусь непогану програмку, яка використовує XML-документи для збереження даних. Наявні дані для заповнення вони поділили між собою, щоб було швидше. Вони створили XML документи і заповнили їх даними, але при зведенні даних виявилось, що їхні документи мають різну структуру та імена елементів і атрибутів відрізнялись. Оля позначала все гарно на англійській мові (product, name, price тощо), а Вася як вже вмів – транслітом (tovar, imja, suva тощо).

Щоб цієї ситуації не сталося, їм потрібно було написати документ з набором правил, яких потрібно дотримуватись, тобто DTD-документ чи XML-схему. Але що ж робити, якщо це вже сталося? В такому випадку допоможе XSLT, який дозволить написати один стандартний набір правил і привести до єдиного формату обидва XML документи. Для цього підійде тип конвертування з XML в XML.

Ще одним прикладом трансформації з XML в XML може слугувати випадок, коли Ви пишете певний COM-компонент або програму, яка пересилає дані з одного прикладного додатку в інших. Як правило, кожна з цих прикладних програм буде мати свою власну структуру XML даних на вході і на виході. Трансформація з XML в XML тут необхідна для того, щоб привести отримані дані в форматі XML до вигляду, який розуміє інша програма.

Але конвертування (трансформація) даних з одного формату в інший не так просто зробити як здається. Перш, ніж зробити конвертування дерева документа в XML, з нього слід вибрати ті вузли, які будуть найбільш перетворені. Їх можна вибрати по імені, вмісту, атрибутам або/та іншим особливостям. Умови відбору вузлів задаються шаблоном (**pattern**), який записується у вигляді одного або кількох виразів на мові **XPath**. Вирази, що містяться в шаблоні, можна об'єднувати символом ( | ), що означає, що обираємий вузол повинен задовольняти хоча б одному виразу шаблону.

Мова XPath була виділена також при відділенні XSLT і призначенням її стало звернення до частин XML-документів з метою здійснення вибірки необхідних його вузлів. Оскільки мову XPath ми ще не вчили, пропонуємо саме на ній і зупинитись, а потім повернутись до XSLT.

## 2. Адресація на мові XPath. Поняття вузла та виразів XPath (синтаксис та приклади написання)

Мова **XPath** не являється реалізацією XML, - це мова, яка основана на наборі синтаксичних правил для виділення різних частин XML документів. XPath була розроблена для застосування в XSLT, XPointer і інших XML-додатках та стала офіційною рекомендацією W3C як мова адресації 16 листопада 1999 року.

На даний час існує дві версії XPath: XPath 1.0 та XPath 2.0.

Основу даної мови складають вирази різних типів, до складу яких входять логічний, числовий та рядковий типи. В цих виразах записуються константи, змінні та функції, які входять до складу XPath. Вони зв'язуються операціями, характерними для даного типу. В результаті обчислення виразу ми отримуємо посилання на одну або кілька ділянок документа.

Вирази XPath мають вигляд шляхів (path) до файлів в файловій системі. Саме від цього і пішла назва даної мови - **“XML Path”**, скорочено **XPath**. Як правило, вирази XPath показують шлях до певної ділянки документа XML, який починається з кореня документа. Вони складаються з кількох кроків пошуку, які виконуються послідовно зліва направо. Самі кроки між собою розділені символами ( / ). Наприклад:

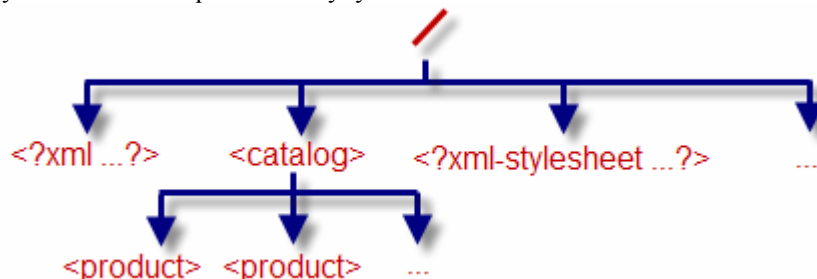
```
root/node1/index.xml
```

Мова XPath розглядає XML документ у вигляді дерева, аналогічно тому, як можуть бути представлені в вигляді дерева папки і файли в файловій системі комп'ютера. Корінь документа позначається символом ( / ), але одразу відмітимо, що



корінь XML документа в теорії XPath не є його кореневим вузлом. І це вірно, оскільки перед кореневим елементом ми можемо розмістити довільну кількість інструкцій по обробці, які також належать документу. Від кореня відходять **гілки**, які складаються з вузлів. **Вузлами (nodes)** можуть бути елементи, їх атрибути та тексти, які являються вмістом даного вузла.

Схематично XML документ можна зобразити наступним чином:



### 3. Кроки вибірки

#### 3.1. Основні засади

Кожен **вираз XPath**, тобто **шлях вибірки** – це послідовність кроків пошуку (location steps), які необхідно зробити, щоб отримати бажаний результат. Кожен крок складається з **трьох частин**:

1. **вісь пошуку** або **навігації** (axis) – вона показує напрямок, в якому буде здійснюватись вибірка на даному кроці. Наприклад, можна обрати дочірні вузли, вузли-атрибути або батьківський вузол поточного вузла;
2. **тест вузла** (node tests) – вказує на те, вузли якого типу або з якими іменами повинні бути обраними на даному кроці;
3. **однин або кілька предикатів** (predicates) (необов'язкова частина) – це логічні вирази, які фільтрують множину вузлів, які були обрані на даному кроці вибірки.

Отже, вісь навігації відповідає за напрямок руху, тест - за те, які вузли обирати, а предикат відбирає вузли, які відповідають не потрібним характеристикам.

Враховуючи все вищесказане, синтаксис використання кроків вибірки буде мати наступний вигляд:

```
axisname::nodetest[predicate]
```

Де **axisname** – ім'я осі,  
**nodetest** – тест вузла,  
**predicate** – предикат.

Наприклад:

```
child::price[price=9.90]
```

Даний крок вибірки складається з осі навігації **child**, яка обирає дочірні елементи **price** поточного вузла. При цьому обираються тільки елементи, значення яких рівне **9.90**.

#### 3.2. Вісь навігації

Як ми вже говорили вище, вісь відповідає за напрямок руху вибірки по дереву документа. Але, на відміну від звичайних шляхів, шлях вибірки може йти в довільному напрямку:

- вниз (прямий порядок) – від поточного вузла документа по гілкам до вкладених вузлів-елементів і листків;
- вверх (зворотньому) – від листків або поточних вузлів-елементів до кореневого вузла документа.

В XPath існує 13 осей навігації. Перших 8 задають напрямок пошуку „зверху вниз”, інші 5 – „знизу вверх”:

Назва осі	Опис
<b>Напрямок „зверху вниз”</b>	
self	Містить <b>поточний</b> контекстний вузол
child	Містить всі <b>дочірні вузли</b> , крім вузлів-атрибутів та вузлів просторів імен. Ця вісь приймається віссю по замовчуванню
descendant	Містить всіх <b>потомків</b> поточного вузла. Ця вісь не містить вузлів атрибутів та просторів імен. По суті це рекурсивне застосування вісі child.
descendant-or-self	Містить поточний вузол плюс всіх його потомків. Не містить вузлів атрибутів та просторів імен. Це об'єднання вісей self та descendant
attribute	Містить всі <b>атрибути</b> поточного вузла
namespace	Містить всі вузли <b>просторів імен</b> поточного вузла



following	Містить всі вузли документа, розміщені після закриваючого тега даного вузла. Тобто вказує на вузли, які йдуть після поточного. Ця вісь не містить потомків поточного вузла, атрибутів та просторів імен
following-sibling	Містить наступний вузол того ж рівня, що і поточний вузол. Якщо поточний вузол є атрибутом або простором імен, то ця вісь буде пуста
<b>Напрямок „знизу вгору”</b>	
parent	Містить <b>батьківський вузол</b> поточного вузла, якщо він існує
ancestor	Містить всіх <b>предків</b> (батьків, прародичів тощо) поточного вузла. Слід пам'ятати, що ця вісь завжди включає в себе кореневий елемент, якщо поточний не є кореневим
ancestor-or-self	Містить поточний вузол плюс всіх його предків. Це об'єднання вісей parent та ancestor
preceding	Містить всі вузли документа, розміщені перед відкриваючим тегом даного вузла. Тобто вказує на вузли, які передують поточному. Ця вісь не містить предків поточного вузла, атрибутів та просторів імен
preceding-sibling	Попередній вузол того ж рівня, що і контекстний вузол. Якщо поточний вузол є атрибутом або простором імен, то ця вісь буде пуста

Наприклад:

Приклад	Результат
//attribute(@name) //attribute:: attribute(@name)	Звернення до атрибутів name всіх потомків
/child::section/child::paragraph	Доступається від поточного вузла до дочірнього вузла section, а від нього до дочірнього вузла paragraph
child::text()	Текстовий дочірній вузол поточного елемента

### 3.3. Тести (перевірки) вузлів

Друга частина вибірки – це власне тести вузла, які дозволяють обрати з конкретної вісі вузли необхідного типу або які мають певні імена. Існують різні випадки використання **тестів вузлів**:

- Тести по імені вузла, які дозволяють обрати елемент з вказаним іменем. Наприклад, тест вузла **book** відповідає лише елементам **<book>** з вказаної осі;
- Тест вузла ( \* ) відповідає всім дочірнім елементам вказаної осі;
- Тест **префікс:\*** виконується для вузлів вказаного простору імен;
- **node()** – всі вузли в вказаній осі;
- **text()** – всі текстові вузли;
- **comment()** – всі вузли-коментарі;
- **processing-instruction( [назва] )** – всі інструкції по обробці або ті, які мають вказану назву;
- **element( [назва][, тип] )** – всі вузли-елементи або ті, які мають вказану назву і вказаний тип (визначається схемою XSD документа);
- **attribute( [@назва][, тип] )** – всі атрибути або ті, які мають вказану назву і вказаний тип (визначається схемою XSD документа);
- **document-node( [вміст] )** – всі кореневі вузли документа або лише ті, які мають вказаний вміст. Наприклад: document-node(element(price)).

Приклади використання тестів з вісями:

Приклад	Результат
child::cd	Всі елементи cd, які є дочірними по відношенню до поточного
attribute::src	Всі атрибути src поточного елемента
child::*	Всі дочірні елементи поточного вузла
child::text()	Текстовий дочірній вузол поточного елемента
child::node()	Всі дочірні вузли поточного елемента
descendant::cd	Всі елементи cd, які є потомками поточного елемента
ancestor-or-self::cd	Всі елементи cd, які є предками поточного елемента, а якщо поточний вузол також є елементом з іменем cd, тоді він також виділяється
child::* / child::price	Всі елементи-онуки price поточного вузла
descendant::processing-instruction()	Всі вузли інструкцій по обробці, які являються потомками поточного контекстного вузла



following::processing-instruction('app')

Всі вузли інструкції по обробці з назвою 'app', які йдуть за контекстним вузлом

### 3.4. Предикати

Після відбору, проведеного віскою і тестом вузла, з отриманої послідовності вузлів можна отримати ті, які задовольняють більш детальним вимогам. Ці вимоги описують в третій частині кроку вибірки – **предикаті**. Іншими словами, предикати дозволяють виділити з множини вузлів новий набір вузлів або конкретний вузол, тобто відфільтрувати вибірку. Предикат являє собою логічний вираз, результатом якого є true або false.

Сам предикат, як вже було показано по синтаксису раніше, поміщається в квадратні дужки ( [ ] ). А фільтрація вузлів здійснюється наступним чином:

1. Фільтруєми набір сортується в напрямку перегляду вісі навігації даного кроку.
2. Вираз предиката обраховується для кожного вузла відсортованого набору.
3. Результат обрахунків предиката конвертується в логічний тип, згідно стандартних правил приведення до типу.
4. З фільтруючого набору виключаються всі вузли, логічне значення предикату для яких було рівне false.
5. В випадку, якщо в кроці вибірки було кілька предикатів, процедура фільтрації повторюється з кожним з них, залишаючи в відфільтрованому наборі лише ті вузли, для яких предикат істинний.

Приклади виразів з використанням предикатів:

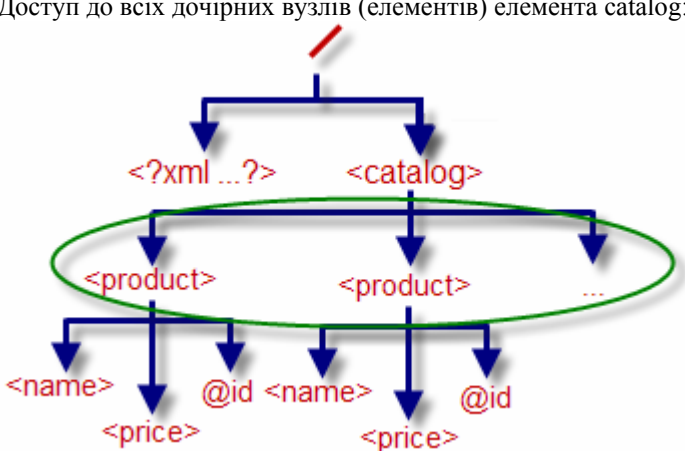
Приклад	Результат
author [degree and award]	Всі елементи <author>, які містять дочірні елементи <degree> та <award>.
//author [@id]	Елементи <author>, які мають атрибут id
//author [@*]	Елементи <author>, які мають хоча б один довільний атрибут
//author [not(@*)]	Елементи <author>, які не мають жодного атрибута
//author [@id='b1']	Елементи <author>, які мають атрибут id з значенням b1

### 3.5. Скорочений синтаксис

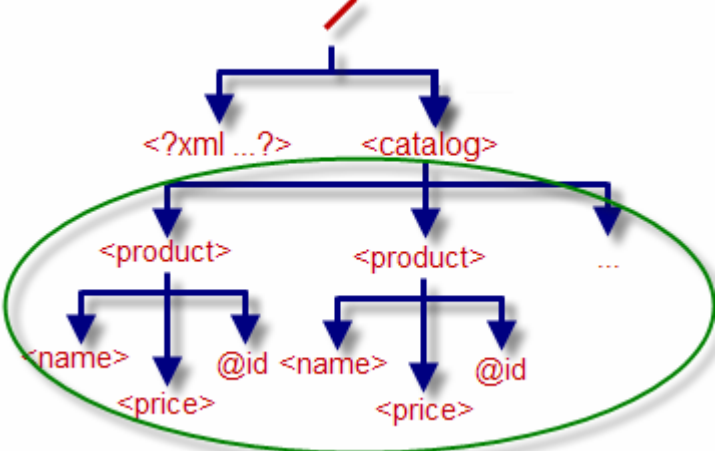
Шляхи або крики вибірки – це найчастіше використовувані вирази XPath і для того, щоб зробити їх менш громіздкими, в XPath їх можна записувати і в скороченій формі. Для цього з кожного кроку вибірки потрібно видалити записи осі (child::, parent:: тощо) по наступним правилам:

Скорочення	Значення
none	child::
@	attribute::
.	self::node()
..	parent::node()
//	/descendant-or-self::node()/

Приведемо кілька прикладів написання скорочених виразів на мові XPath:

Вираз	Опис
/root/x/teg	Доступаємось до елемента teg, що є дочірнім елементом x, який в свою чергу є дочірнім по відношенню до кореневого елемента з іменем root
/root/@name	Доступ до атрибута name кореневого елемента root
/catalog/*	Доступ до всіх дочірніх вузлів (елементів) елемента catalog: 



/catalog//	Доступ до всіх потомків елемента catalog 
/root/processing-instruction("hello")	Доступ до інструкції по обробці з іменем hello, яка міститься всередині кореневого елемента root
.	Отримати текст поточного елемента
./child або child	Доступитись до дочірнього елемента child поточного контекстного вузла
./author або //author	Знайти елемент author, який є потомком (на довільній глибині xml документа) поточного вузла
../	Перейти в дереві на рівень вище
*/*	Знайти всі дочірні елементи контекстного вузла і в середині нього всі дочірні елементи
/catalog/*/price	Виділити всі елементи price, які є онуками елемента catalog
author[country]	Доступитись до елемента author, який має дочірній вузол country
/catalog/cd/title   /catalog/cd/artist	Виділити всі елементи title і artist, які є дочірніми елементами елементів cd (які являються дочірніми елементами елемента catalog)
text()	Виділити всі дочірні текстові вузли поточного елемента

#### 4. Операції та оператори в XPath

Вирази XPath являються обширними конструкціями цієї мови. Шляхи вибірки, які були вже розглянуті, - це лише частковий випадок виразів. Насправді, вирази XPath можуть включати в себе використання різного типу операцій (арифметичні, порівняння, логічні) та виклику функцій для більш ширшого та точнішого доступу до необхідних вузлів.

Отже, розглянемо для початку набір операцій та операторів, які підтримуються в виразах XPath. Перш за все, слід відмітити, що XPath завжди перед виконанням операцій, приводить кожен операнд до числа. А тепер опишемо існуючі операції та оператори:

1. **Арифметичні** – для здійснення арифметичних операцій над числами.

- + (додавання) -  $6 + 4 = 10$
- - (віднімання) -  $6 - 4 = 2$
- \* (множення) -  $6 * 4 = 24$
- div (ділення) -  $8 \text{ div } 4 = 2$
- mod (остача від ділення) -  $5 \text{ mod } 2 = 1$
- унарний - -5

2. **Порівняння** – для порівняння двох числових значень: != (не рівно), = (рівно), >, <, >=, <=. При застосуванні операторів >, >=, <, <= XPath перед здійсненням порівняння кожен операнд переводить в число.

При застосуванні операторів != або = для **перевірка відносного набору вузлів** результати будуть наступні:

- Якщо якесь значення перевіряється на рівність з набором вузлів (node-set), результат при цьому буде мати значення true, якщо набір вузлів містить хоча б один вузол, чиє значення рівне перевіряємому значенню.
- Якщо значення перевіряється на нерівність з набором вузлів, результат буде мати значення true, якщо набір вузлів містить хоча б один вузол, чиє значення не рівне перевіряємому.

Тому набір вузлів може бути рівним і нерівним заданому значенню одночасно.

3. **Логічні** оператори:

- or - логічне АБО;



➤ and - логічне І.

**Примітка!** Оператора логічного заперечення в XPath немає. Замість нього використовується логічна функція `not()`, про яку мова піде в наступному розділі ([5. Функції XPath](#)).

## 5. Функції XPath

Крім операцій, які забезпечують примітивні базові дії, XPath підтримує базову бібліотеку функцій. Варто також відмітити, що більшість процесорів XSLT реалізують механізм розширень, за допомогою якого можна використовувати в XSLT власні функції. Але ця тема доволі обширна і тому ми обмежимося лише вивченням лише функцій базової бібліотеки.

Всі функції базової бібліотеки XPath поділяють на **4 типи**:

1. функції для роботи з наборами (множиною) вузлів;
2. функції для роботи з логічними значеннями;
3. функції для роботи з числами, тобто арифметичні функції;
4. рядкові функції.

Приведемо їх список:

Назва	Опис	Приклад
<b>Функції над набором вузлів</b>		
<code>number count(node-set)</code>	Повертає кількість виділених елементів	
<code>number last()</code>	Повертає кількість вузлів в обраному наборі вузлів	
<code>number position()</code>	Повертає позицію виділеного елемента (поточного вузла)	
<code>string local-name(node-set?)</code>	Повертає локальну частину імені першого в порядку перегляду документа вузла з набору, переданого в якості аргумента. Якщо параметр упущений, то значенням функції являється набір, який містить єдиний контекстний вузол, тобто повертає локальну частину розширеного імені контекстного вузла (якщо вона існує).	
<code>string name(node-set?)</code>	Аналогічна попередній функції, але повертає розширене ім'я першого в порядку перегляду документа вузла з набору, переданого в якості аргумента.	
<code>string namespace-uri(node-set?)</code>	Аналогічна попередній функції, але повертає URI простору імен розширеного імені вузла з вказаного набору вузлів, який йде в документі першим	
<code>node-set id(object)</code>	Повертає елемент по його унікальному ідентифікатору ID	
<b>Рядкові функції</b>		
<code>string string(object?)</code>	Приводить об'єкт до рядка. Якщо аргумент відсутній, функції передається набір вузлів, який складається з єдиного контекстного вузла	<code>string(3.14) = '3,14'</code> <code>string(true) = 'true'</code> <code>string(number('zero')) = NaN</code>
<code>string concat(string, string, string*)</code>	Конкатенація аргументів (двох або більше рядків)	<code>concat('The', ' ', 'XML') = 'The XML'</code>
<code>boolean contains(string, string)</code>	Повертає true, якщо перший рядок містить другий рядок	<code>contains('XML', 'X') = true</code>
<code>string normalize-space(string?)</code>	Видаляє з рядка початкові та кінцеві пробільні символи, а всі послідовності пропусків замінюються одним. Якщо аргумент відсутній, функція виконується над рядковим значенням контекстного вузла	<code>normalize-space(' The XML ') = 'The XML'</code>
<code>boolean starts-with(string, string)</code>	Повертає true, якщо перший рядок починається з другого рядка	<code>starts-with('XML', 'X') = true</code>
<code>number string-length(string?)</code>	Довжина рядка. Повертає кількість символів. Якщо аргумент відсутній, функція повертає довжину рядкового представлення контекстного вузла	<code>string-length('Beatles') = 7</code>
<code>string substring(string, number, number?)</code>	Повертає підрядок, починаючи з вказаної другим параметром позиції і довжиною (третій параметр). Якщо довжина відсутня, повертається підрядок від вказаної позиції до кінця рядка. Позиція першого символу 1.	<code>substring('Beatles', 1, 4) = 'Beat'</code>
<code>string substring-after(string, string)</code>	Повертає підрядок після заданого підрядка	<code>substring-after('12/10', '/') = '10'</code>



string substring-before(string, string)	Повертає підрядок перед заданим підрядком	substring-before('12/10','/') = '12'
string translate(string, string, string)	Робить заміну символів в рядку. Рядок, в якому необхідно здійснити заміну – перший параметр, другий параметр – рядок символів, які потрібно замінити, третій – рядок символів, на які потрібно замінити. Якщо другий аргумент довше третього, символи, для яких немає відповідної заміни, видаляються з рядка. Якщо третій довше другого, тоді залишок рядка ігнорується	translate('12:30',':','!') = '12!30' translate('abcde','ace','XYZ') = 'XbYdZ'
<b>Арифметичні (числові) функції</b>		
number number(object?)	Приводить значення аргументу до числа. Якщо аргумент не вказаний, функції передається набір вузлів, який складається з єдиного контекстного вузла	number(price) number(1.) = 1 number(\$b or not(\$b)) = true Примітка! \$b – змінна.
number ceiling(number)	Округляє до найменшого цілого числа, яке не менше аргумента	ceiling(3.14) = 4
number floor(number)	Округляє до найбільшого цілого числа, яке не більше аргумента	floor(3.14) = 3
number round(number)	Округляє до найближчого цілого	round(3.14) = 3
number sum(node-set)	Рядкове значення кожного вузла в переданому наборі приводиться до числа і повертається їх сума	sum(/cd/price)
<b>Логічні функції</b>		
boolean boolean(object)	Приводить до булевого типу. Якщо значення рівне 0 або являється не числом (NaN), рядок або набір вузлів являються пустими, тоді вони приводяться до значення false, інакше до true	boolean(2-2) = false boolean(-1) = true boolean('false') = true boolean(1 div 0) = true boolean(/) = true boolean(/products) = false or true
boolean false()	Повертає false	number(false()) = 0
boolean not(boolean)	Повертає true, якщо аргумент має значення false, і навпаки	not(false()) = true not(0) = true not(/) = false
boolean true()	Повертає true	number(true()) = 1
boolean lang(string)	Використовується для визначення мови вузла. Для цього використовується значення атрибуту xml:lang елемента.	Для вузлів: <firstname xml:lang="en-us" /> <firstname xml:lang="en" />  Функція lang('en') поверне true

## 6. Приклади побудови логічних виразів та виразів порівняння на мові XPath

Ну і насамкінець приведемо ще ряд прикладів написання виразів XPath:

Шлях	Опис
//author[normalize-space(@name)='Jon']	Обираються елементи author, які мають атрибут name з значенням, яке після нормалізації буде рівне Jon
child::node()	Всі дочірні вузли контекстної ноди, які мають довільний тип
attribute::name	Обрати атрибут name поточного вузла
attribute::*	Обрати всі атрибути поточного вузла
descendant::para	Обрати елемент <para>, який являється потомком контекстного вузла.
ancestor::div	Обрати предка з назвою <div> контекстного вузла.
self::para	Обрати контекстний вузол, якщо елемент носить назву <para>, інакше нічого не обирати
child::chapter/descendant::para	Обрати потомків з назвою <para>, які мають дочірні елементи <chapter> контекстного вузла.





/	Обрати корінь документа
child::x[position()=2] x[position()=2] x[2]	Отримати доступ до дочірнього елемента x, який має позицію 2 (індексація починається з 1)
child::para[position()=last()] para [last()]	Обрати останній дочірній елемент <para> контекстного вузла.
child::para[position()=last()-1]	Обрати передостанній дочірній елемент <para> контекстного вузла
child::para[position()&gt;1]	Обрати всі дочірні елементи <para> поточного контекстного вузла, які йдуть після першого
/child::doc/child::chapter[position()=5]/child::section[ position()=2]	Обрати другий елемент <section>, який йде після п'ятого елемента <chapter>, що міститься в елементі <doc>.
child::para[attribute::type="warning"]	Обрати всі дочірні елементи <para>, значення атрибуту type яких рівне "warning".
child::para[attribute::type="warning"][position()=5]	Обрати п'ятий дочірній елементи <para>, значення атрибуту type серед яких рівне "warning".
child::para[position()=5][attribute::type="warning"]	Обрати п'ятий дочірній елемент <para>, якщо його значення атрибуту type рівне "warning".
child::chapter[child::title]	Обрати дочірній елемент <chapter>, якщо він містить один або більше дочірніх елементів <title>.
child::*[self::chapter or self::appendix]	Обрати дочірні елементи <chapter> і <appendix> контекстного вузла.
author	Всі елементи <author> від поточного контекстного вузла
first.name	Всі елементи <first.name> від поточного контекстного вузла
*[@specialty]	Всі елементи з атрибутом specialty
@*	Всі атрибути поточного контекстного елемента.
author[not(country)]	Доступитись до елемента author, який не містить дочірнього вузла country
author[country = 'US']	Доступитись до елемента author, в якого значення його дочірнього елемента country = 'US'
author[country=US']/name	Доступитись до дочірнього елемента name вузла author, в якого значення дочірнього елемента country = 'US'
author[@country=US']/name	Доступитись до дочірнього елемента name вузла author, в якого значення атрибуту country='US'
book[/bookstore/@specialty = @style]	Отримати доступ до елемента book за умови, якщо значення його атрибуту style рівне значенню атрибуту specialty його дочірнього елемента bookstore
book[../magazine/@style= @style]	Отримати доступ до елемента book за умови, якщо значення його атрибуту style рівне значенню аналогічного атрибуту елемента magazine, який знаходиться з ним на однаковій глибині
my:book	Отримати доступ до елемента <book> з простору імен my.
my:*	Отримати доступ до всіх елементів простору імен my.
@my:*	Всі атрибути для простору імен my

## 7. Домашнє завдання

Обрахувати наступні вирази:

1. book[@style]
2. author[first-name][3]
3. author[1]
4. price/@exchange/total
5. x/y[position() = 1]
6. x[1]/y[2]
7. (x/y)[1]
8. (book/author)[last()]
9. book[excerpt]
10. book[excerpt]/author[degree]
11. author[not(degree or award) and publication]
12. author[last-name [position()=1]='Bob']
13. degree[@from != "Harvard"]
14. author[. = "Matthew Bob"]



- 
15. author[last-name = "Bob" and ../price > 50]
  16. author[\* = "Bob"]
  17. author[last-name = "Bob" and first-name = "Joe"]
  18. price[@intl = "Canada"]
  19. degree[position() < 3]
  20. p/text()[2]
  21. ancestor::author[parent::book][1]
  22. author[degree][award]
  23. ancestor-or-self::div
  24. descendant-or-self::para

Copyright © 2012 Iryn Petrova.