Урок 1

План заняття:

1. Введення в XML. Історія розвитку. Зв'язок з SGML, XHTML, HTML та іншими мовами розмітки
2. Парсери (синтаксичні аналізатори) та редактори XML
3. Синтаксис та структура XML документа
4. Візуальне представлення XML документа за допомогою CSS
5. Інтеграція XML з HTML (острови даних)
6. Формально-вірні XML документи. Введення в DTD та схеми
7. Домашнє завдання

1. Введення в XML. Історія розвитку. Зв'язок з SGML, XHTML, HTML та іншими мовами розмітки

В сучасному світі існує необхідність обміну структурованою інформацією між додатками, які написані на різних мовах та працюють на різних апаратних платформах. Це пов'язано з тим, що кожен додаток використовує свій специфічний бінарний формат даних. Крім того, необхідно, щоб кожному користувачу цей документ також був зрозумілий, а сам бінарний формат звичайній людині неможливо прочитати без спеціальних інструментів.

Розвиток програмування та потреб користувачів призвів до появи Інтернету, який дозволив обмінюватись інформацією в електронному вигляді, об'єми якої з самого початку були вже доволі великими. Та поява Інтернет не подолала стару проблему сумісності, а навпаки її збільшила. Щоб її вирішити, люди почали створювати стандарти та протоколи, які дозволили б різним системам, що обмінюються між собою даними, спілкуватись одна з одною на різних мовах. Разом з тим кількість передаваємих даних зростала і стандартизувати її було вже нагальною проблемою.

На той час вже існувала мова розмітки **HTML (Hypertext Markup Langauge)**, яка дозволяла структуризувати документ. Сама суть розмітки, згідно правил HTML, полягала в тому, що документ поділявся умовно на частини, які відділялись одна від одної текстовими мітками, які називаються **тегами (tags)**. Кожній з цих частин можна було задати певні особливості, правила форматування тощо. Теги мають простий синтаксис і записуються в кутових дужках (< >). Крім того, сам текст, описаний в середині тегів HTML розуміють всі браузери, він зручний та легкий в використанні. Та все ж мова HTML має **два суттєві недоліки**:

1. Набір тегів HTML строго фіксований, його неможливо розширити або змінити. Отже, всі браузери повинні інтерпретувати теги однаково, щоб користувач, який написав текст з розмітками HTML, був впевнений, що цей текст буде однаково виглядати у всіх браузерах.
2. Теги HTML показують лише візуальну розмітку, зовнішній вигляд документа, але нічого не говорять про його структуру.

Та не дивлячись ні на що простота HTML переважила його недоліки. Вона призвела до скаженого росту WEB-сайтів і кожен користувач відчув себе творцем. Але все ж таки обмежені можливості мови почали незадовольняти тих, хто відчув себе професіоналами. Введення таблиць стилів **CSS (Cascading Style Sheet)** та поява серверних сценаріїв лише ненадовго зменшила незадоволення розробників, оскільки професіоналам постійно хочеться додати якусь свою родзинку, а засобів розробки не достатньо.

І тут всі згадали, що в 1986 р. був затверджений стандарт мови створення мов розмітки **SGML (StandardGeneralized Markup Language)**, за допомогою якої власне і побачила світ мова розмітки HTML (1989 р.). Основна особливість даної мови полягала в тому, що **вона дозволяла створити нову мову розміток, визначивши набір тегів мови**. Кожен конкретний набір тегів, розроблений по правилам SGML, супроводжується описом **DTD (Document Type Definition)** – визначенням типу документа, який пояснює зв'язок тегів між собою і правила їх застосування.

З однієї сторони дуже зручно, але з другої вона доволі складна та потребувала доволі громіздкого опису елементів мови, яку вона створює. Лише одна її специфікація містила більше 500 сторінок. Крім того, в мові SGML була присутня велика кількістю зовсім непотрібних в Web можливостей.

В зв'язку з цим, мову SGML також довелось відкинути. Все це було занадто складним і виправдовувало себе тільки при описі крупних проектів, де ця громіздкість була виправдана. Наприклад, при створенні єдиної системи документообігу підприємства.

Золотою серединою між мовами SGML та HTML стала розширювана мова розмітки **XML (eXensible Markup Language)**, що і стала мовою для текстового вираження структурованої інформації в стандартному вигляді. По своїй суті XML являє собою метамову[1], тобто сам по собі він не має операторів, не визначає жодну алгоритмічну послідовність дій і не виконує ніяких розрахунків, його ціль – описувати нові мови документів. Крім того, дана мова являється універсальною та кросплатформенною.

[1] **Метамова** – це мова, засобами якої описуються властивості та відношення окремої предметної області.

Розробка XML почалась в 1996 р. Нею займається громадська організація **W3C (World Wide Web Consorcium)**. XML має безпосерню подібність з HTML. Наприклад, в XML також присутні теги, але вони називаються **елементами**. Та не слід вважати, що XML заміняє HTML. Це невірна думка. За допомогою XML ми лише описуємо структуру майбутнього документа, а за зовнішній вигляд документа (шрифти, кольори, форматування тощо) турбуються інші мови.

Приведемо приклад, який підтвердить дане. Припустимо, що в нас існує наступний текст новини:

```
Увага! Продається 2-х поверховий будинок в гарному стані. Ціна за домовленістю.
Звертатись по тел. 22-22-22.
```

Якщо ми хочемо розмістити дану новину на Web-сайті і звернути на деякі частини даної новини увагу, тоді потрібно їх виділити. Для виділення окремих частин даного документа, нам необхідно скористатись тегами розмітки, які дозволять розмежувати документ.

Спочатку використаємо для цього мову розмітки HTML. Наша новина тепер може набути наступного вигляду:

```
<p><b>Увага!</b> <br/>Продається <u>2-х поверховий будинок</u> в гарному стані.
<u>Ціна за домовленістю</u>.<br/>Звертатись по тел. <b>22-22-22.</b></p>
```

Після такої розмітки, наш документ з новиною носить не лише інформативний характер, але і дані про те, як він повинен виглядати. Це робить документ більш зрозумілим для людини, але не для машини. Словосполучення «2-х поверховий будинок» та «ціна за домовленістю», виділені в тексті однаковими тегами, але описують зовсім різні речі. Перше – властивість будинку, а друге – дані про оплату. Іншими словами, однакові теги можуть нести різний зміст, який важко визначити.

Для вирішення такої проблеми невідповідності, можна використати мову розмітки XML, яка надає засіб, за допомогою якого можна розширити стандартні теги так, щоб вони могли відобразити суть як цілого документа, так і його окремих частин. З використанням XML, наш документ з новиною набуде наступного вигляду:

```
<news><bold>Увага!</bold>
Продається <feature>2-х поверховий будинок</feature> в гарному стані.
<price>Ціна за домовленістю</price>.
Звертатись по тел. <phone>22-22-22.</phone></news>
```

В такому вигляді наш документ містить більш детальну інформацію про свою структуру:

- в тегах <feature> міститься інформація про характеристики будинку;
- в тегах <price> - дані про оплату;
- в тегах <phone> - контактні дані, тобто інформація про телефон.

Тепер даний документ можна обробляти програмно. Наприклад, якщо буде необхідно створити список з інформацією про характеристики будинку, то достатньо буде отримати інформацію з необхідних нам тегів. Ви також не втрачаєте можливості відформатувати зовнішній вигляд документа. Для цього достатньо задати правила форматування для того чи іншого тега.

Отже, розширивши наявну кількість тегів, ми вирішуємо дві проблеми:

- ✓ Явно виділяємо структуру даних, що надає можливість подальшої програмної обробки документа. При цьому користувачу даний документ залишається зрозумілим.
- ✓ Відокремлюємо дані для того, щоб зробити його наглядним, тобто візуально зрозумілим.

2. Парсери (синтаксичні аналізатори) та редактори XML

Один із провідних спеціалістів Дослідного Центра Інформатики в області XML-технологій (Карлсрує), Олексій Валіков колись зауважив, що «Створення XML-документів без програмного забезпечення, яке буде розуміти його семантику – це все-рівно, що писати програми на мові програмування, для якої не існує трансляторів та інтерпретаторів». І це дійсно так, адже такі документи можуть бути ідеально коректними, але зовсім непотрібними.

Нескладний та стандартизований синтаксис XML дозволив багатьом компаніям розробити та випустити ряд синтаксичних аналізаторів (XML-парсерів, XML parser) для аналізу XML-документів, тобто програм, які здійснюють перевірку вхідної послідовності символів на те, чи вона існує в даній мові і вірно використовується (граматика). Отже, кожен XML-документ перед його представленням проходить через синтаксичний аналізатор, який перевіряє його на коректність та відповідність всім синтаксичним правилам XML. Якщо XML-документ не порушує визначені правила та стандарт, то він називається **формально-вірним** або **дійсним**.

В якості прикладів відомих розроблених компаніями **синтаксичних аналізаторів** можна віднести **наступні**:

- ✓ **libxml** – безкоштовний парсер для XML. Остання версія libxml 1.8.17 та libxml2 2.6.17 (з підтримкою Phython 2.4)
- ✓ **libxslt** - безкоштовний парсер для XSLT та є розширенням бібліотеки libxml2. Остання версія libxslt 1.1.12
- ✓ Від фірми Microsoft – **MSXML 5.0** для старіших версій офісу та **MSXML 6.0** – для MS Office 2003. Ставиться як правило разом з MS Windows та вбудований в браузер MS Internet Explorer починаючи з версії IE 5.0. Тобто XML-документи можна перевіряти на коректність, просто запустивши їх в Internet Explorer. Якщо документи пройдуть

синтаксичний аналіз в браузері буде представлена його ієрархічна структура, інакше буде згенерована помилка. Також даний парсер може використовуватись як ActiveX додаток.

- ✓ Фірма Sun Microsystems випустила парсер **Java ProjectX**. Хоча даний парсер міг працювати і як ієрархічний аналізатор, так і як частина XML-редактора, дана розробка в них трішки не вдалась: хоча парсер і ідеально відповідав стандарту, він був доволі слабким та в останні роки майже не обновлювався.
- ✓ Apache Foundation внесла також немалий вклад в розвиток XML парсерів:
  - **Xalan-Java** – XSLT-процесор для трансформації XML-документів в HTML, text та XML. Написаний на Java. Остання версія Xalan-J 2.7.1 (http://xml.apache.org/xalan-j/);
  - **Xerces** – парсер для XML. Існує версія на Java (http://xerces.apache.org/xerces-j/) і на C++ (http://xerces.apache.org/xerces-c/)
  - **FOP (Formatting Object Processor)** – це процессор XSL-FO. Остання версія 0.95. Хоча версія не велика, оскільки знаходиться лише в стадії розробки та потужності доволі великі. Він дозволяє конвертувати XML-документ в формати: .pdf, .rtf, .png, .tiff, .html, ASCII текст тощо. В процесі розробки трансформація в .svg та .mif. Крім того, дозволяє зроротню трансформацію з .svg, .bmp, jpg, .tiff в XML. Для XML-перетворень використовує Xalan та входить до його дистрибутиву (http://apache.infocom.ua/xmlgraphics/fop/);
  - **Batik** – дозволяє конвертацію в *.svg формат (векторний формат даних, який описується за допомогою елементів XML), а також повноцінно працювати з SVG форматом. Містить SVG парсер, генератор, конвертор та браузер (http://xmlgraphics.apache.org/batik/)

Не дивлячись на таке різноманіття синтаксичних аналізаторів, всі вони використовують два основні типи XML-парсерів: SAX та DOM. Вони являються стандартизованими та їх розвиток контролює корпорація W3C. Більш детальніше з їх основними принципами роботи ми познайомимось пізніше.

Для зручності роботи та створення XML-документів вище описані компанії та ряд інших створюють редактори XML-документів, які використовують вбудовані парсери. На сьогоднішній день існує дуже багато редакторів для роботи з XML документами, від комерційних до безкоштовних. Наприклад:

- ❑ **Stylus Studio XML Editor (**http://stylusstudio.com**)** – комерційний та має вбудований синтаксичний аналізатор. Має вбудований редактор для роботи з XML, XSLT, XPath, DTD, XQuery. Крім того, на сьогоднішній день її майже ніхто не підтримує. Підтримує також трансформацію з XML в XML, HTML та PDF, імпорт з HTML, текстових файлів тощо в XML, а також переводити дані таких баз даних як MS SQL Server, Oracle, IBM DB/2, Sybase, MySQL тощо в XML формат.
- ❑ **Altova XML Spy** (http://www.altova.com/products/xmlspy/xml_editor.html) – комерційна. Конкурує з Stylus, а отже підтримка XML та суміжних мов однакова. Має також графічний редактор для побудови схем, відлагоджувач (debuggers), роботу з базами даних та їх імпортом в XML. Також має парсер на C#, дозволяє створювати WSDL контракти, працювати з форматом SOAP, CSS тощо.
- ❑ **Markup Editor 1.1.3 (**http://www.topologi.com/tme1-0/) – комерційний. Дозволяє створювати XML, XSLT, DTD документи тощо. Написаний на Java. Особливих характеристик, які б її виділяли немає. Невимушений дизайн та примітивна кількість операцій.
- ❑ **Xerlin 1.3 (**http://www.xerlin.org/) – безкоштовна. Розробником являється Exari Inc. Підтримка XML, DTD. Написаний на Java та має підтримку парсера xmllib.
- ❑ **EditiX (http://www.editix.com/)** – комерційна. Існує підтримка XML, DTD, схем, XQuery, XPath, XSLT редактор та відлагоджувач тощо для Windows, Linux та Mac OS EditiX.
- ❑ **Vex - A Visual Editor for XML (**http://vex.sourceforge.net/) – безкоштовний та по максимуму простий. Немає підсвітки синтаксису та дуже схожий на розширений блокнот (Notepad++).
- ❑ **MS Visual Studio** – підходить як простий редактор з підсвіткою синтаксису.
- ❑ будь-який текстовий редактор.

Ваш вибір, яким з даних редакторів користуватись.

3. Синтаксис та структура XML документа

А тепер перейдемо до безпосереднього написання першого XML-документа. Слід одразу відмітити, що XML задає лише загальні правила, по яким створюються теги і оформлюється XML документ. Правила оформлення XML-документа описані в специфікації XML і по мірі вивчення предмету, ми їх будемо вивчати, а поки що розпочнемо з створення першого XML-документа. Розширення файлів XML-документів - **.xml**.

Будь-який XML документ починається з обов'язкового **заголовка** або як його ще називають – **пролога**, який має наступний вигляд:

```
<?xml version="1.0"?>
```

Як видно з опису, заголовок розміщується між операторами **<?** та **?>**. В такі блоки поміщають інструкції, які не відносяться до вмісту самого XML-документа, але містять інформацію для прикладної програми, яка буде обробляти даний документ. Ці інструкції носять назву **інструкціями по обробці (processing instruction)**. Перша частина цієї інструкції, як правило, описує мову або програму, яка дозволяє обробити вміст другої частини даної інструкції.

В XML визначена особлива конструкція, яка називається XML-декларацією та має вищенаведений вигляд. Але згідно стандарту ця декларація не являється інструкцією по обробці, хоча вона на неї схожа та носить той же зміст (тобто вказує на те, що інструкції, які будуть описані нижче в даному документі будуть оброблятись за допомогою мови розмітки XML).

Після вказання назви XML-декларації йде опис версії мови, яку буде підтримувати прикладна програма або документ. Версій XML існує небагато, оскільки сама по собі XML є порівняно новою мовою. Існує дві версії мови: 1.0 та 1.1. Ми будемо користуватись версією XML 1.0, оскільки на те є ряд причин.

Стандарт XML 1.0 підтримує Unicode 2.0, тобто версію Unicode, яка на момент виходу XML 1.0 була основною (1998р.). Згідно стандарту Unicode для кожного символа встановлюється новий унікальний код, завдяки чому всі символи повинні коректно відображатись та оброблятись комп'ютерами. Але в стандарті Unicode 2.0 присутні не всі символи, оскільки це дуже кропітка та довготривала робота. І тому розробники XML 1.0 вирішили обмежити використання символів в своїй мові. Тобто дані документа можна було зберігати в Unicode, а от називати елементи чи атрибути ні.

З плином часу Unicode розростався новими стандартами і корпорація W3C вирішила випустити нову версію XML, яка б виправила помилки старої версії, вміщала пітримку нового стандарту Unicode 3.0 та могла бути сумісною з майбутніми версіями Unicode.

Версія XML 1.1 була затверджена 4 лютого 2004 року. Як вже було сказано, відмінність її від попередньої полягала в тому, що вона підтримувала новий стандарт Unicode 3.0 та мала бути сумісною з новими стандартами. Крім того, в ній був виправлений недолік XML 1.0 пов'язаний з відмінністю позначення кінця рядка в XML та Unicode. Ця відмінність в основному викликала проблеми на IBM-майнфреймах, де кінець рядка позначався символом NEL, який не сприймав XML документ. В новому стандарті був введений спеціальний символ кінця рядка - #x85, символ розділення рядка - #x2028 та ще ряд управляючих символів (коди, які знаходяться в діапазонах від #x1 до #x1F).

Але, не дивлячись на значні нововведення в XML 1.1, на сьогоднішній день дана версія XML майже не використовується, тому що:

1. Не дивлячись на те, що документи XML 1.1 сумісні з майбутніми версіями Unicode, вони не до кінця сумісні з документами нижчої версії XML 1.0. В останній версії не підтримуються ряд управляючих символів, які були доступними та використовувались в XML 1.0. Крім того, існують проблеми пов'язані з зовнішніми сутностями, тобто вкладеністю документів XML 1.0 в документи XML 1.1, з перевіркою на валідність XML схемам тощо.
2. Процесори для XML 1.0 та 1.1 відрізняються, що по суті говорить про введення нового формата даних разом з введенням XML 1.1.
3. На сьогоднішній день існує багато інструментів, які працюють з форматом документів версії XML 1.0 і які не будуть працювати з форматом версії XML 1.1 по вищеописаній причині (п.2). Хоча в цьому пункті є і винятки. Де-які компанії почали вбудовувати в свої редактори підтримку і XML 1.1. Прикладом такого інструменту є Xerces (Xerces Java 2.3.0 та Xerces C++ 2.5.0).

Йдемо далі. Крім версії в XML-декларації можна вказувати ще **<u>два псевдоатрибути</u>**:

- ✓ **encoding**, який задає кодову сторінку, в якій буде описаний документ. На сьогоднішній день використовуються багато **<u>кодувань</u>**, але основні з них:
  - **ASCII** – восьмибітна, але яка охоплює лише **128** символів. Включає латинський алфавіт, цифри і основні знаки пунктуації. Важливість її в тому, що всі інші кодові сторінки сумісні з нею.
  - **windows-1251** – кирилиця. Стандартне кодування для Windows. Однобайтне
  - **windows-1252** - західноєвропейські мови. Для англомовних символів. Однобайтне
  - **KOI8-R** – аналог, але для Unix операційних систем. Однобайтне
  - **KOI8-U** – для української мови - для Unix операційних систем. Однобайтне
  - **MacCyrillic** – кирилиця для Apple Macintosh. Однобайтне
  - **UTF-8** (Unicode Transfer Format) – вміщає всі символи ASCII (128 символів), при цьому кожен символ кодується одним байтом (8 біт). Прийняте по замовчуванню.
  - **UTF-16** – по 2 байти на символ. Використовується для символів деяких національних мов та вміщує 65 тис. символів. Аналогом її є кодування **Unicode**
  - **UTF-32** – по 4 байта на символ. Використовується для меньш поширених символів

  Наприклад:

```
<?xml version="1.0" encoding="windows-1251"?>
```

- ✓ **standalone**, що вказує на те, чи використовує даний документ які-небудь зовнішні оголошення. Значення "yes" означає, що документ не використовує елементи, які описані в іншому, зовнішньому документі, інакше "no". Значення "yes" визначене по замовчуванню.

```
<?xml version="1.0" encoding="windows-1251" standalone = "yes"?>
```

З заголовком розібрались. Що далі? А далі дані, тобто вміст XML-документа. XML-документ складається з елементів. Крім того, в XML виділяють поняття **ієрархії елементів**. Це означає, що в документі повинен обов'язково існувати корневий елемент, який містить в собі інші елементи. Корінь, як і в будь-якому дереві, повинен бути один єдиний. Така вимога виникла тому, що один XML-документ можна вкладати в інший. При цьому кореневий елемент вкладеного

документа стане просто одним з елементів документа, в який він вкладений. Така вкладеність не порушить структуру самого XML-документа.

Також варто відмітити, що ім'я корневого елемента вважається іменем всього документа XML.

Елементи в XML створюються по аналогії з HTML. Тобто елемент починається та закінчується кутовими дужками: (<) та (>). Причому елементи також поділяють на початкові (відкриваючі) та кінцеві (закриваючі), які почитаються з символа (/). Та на відміну від HTML, в якому закриваючий елемент був необов'язковий, в XML невказання закриваючого елемента викличе помилку (аналогічно XHTML). Наприклад:

```
<назва_елемента> текст </назва_елемента>
```

Елементи, які не мають вмісту, тобто пусті елементи, інтерпретуються парсером як одинарні. Тобто два наступних приклади з точки зору XML є рівносильними:

```
<назва_елемента></назва_елемента>
<назва_елемента />
```

Тому, якщо елемент не має вмісту, краще робити його одразу одинарним і не вказувати закриваючого елемента. Але в такому випадку відкриваючий тег повинен закінчуватись символами (/>). Пропуск від назви елемента до цих символів не є обов'язковим, але де-які браузери та парсери можуть не прочитати даний елемент, якщо цей пропуск відсутній.

Крім того, ім'я елемента може бути довільне, але все ж існують ряд **обмежень**:

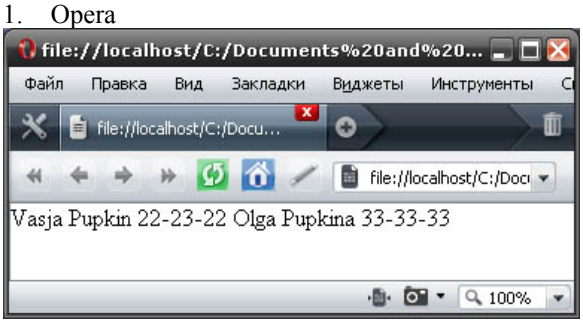
- Імена можуть містити: літери Unicode, арабські цифри, символ дефіса (-), нижнє підкреслення (_), крапку (.), двокрапку (:).
- Імена можуть починатись з літери, символа підкреслення (_) та двокрапки (:).
- Імена не можуть починатись з слова xml в будь-якому регістрі, бо такі імена є захищені правами на інтелектуальну власність консорціума W3C. Кажучи простими словами, - вони зарезервовані.
- В іменах не допускаються пропуски.
- XML регістрозалежний.

Отже, напишемо XML-документ, який буде мітити дані про телефонні номери, тобто записну книжку. Код буде мати наступний вигляд:

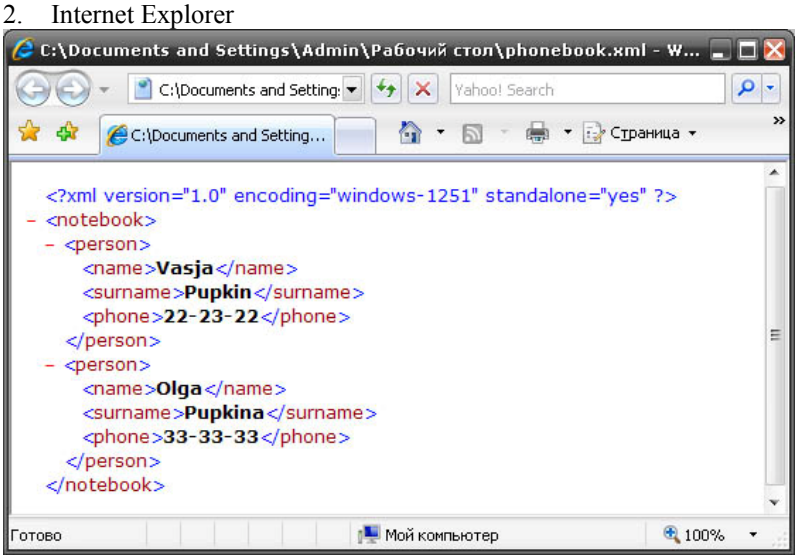
```
<?xml version="1.0" encoding="windows-1251" standalone = "yes"?>
<notebook>
	<person>
		<name>Vasja</name>
		<surname>Pupkin</surname>
		<phone>22-23-22</phone>
	</person>

	<person>
		<name>Olga</name>
		<surname>Pupkina</surname>
		<phone>33-33-33</phone>
	</person>
</notebook>
```

Для того, щоб побачити, що сформований XML-документ являється синтаксично вірним, спробуємо переглянути його в вікні браузера. Але, результат в кожному з них буде трішки відрізнятись, оскільки XML-документи не призначені для такого перегляду (це просто засіб збереження та передачі даних) і кожен браузер містить різний вбудований синтаксичний аналізатор (якщо має взагалі):

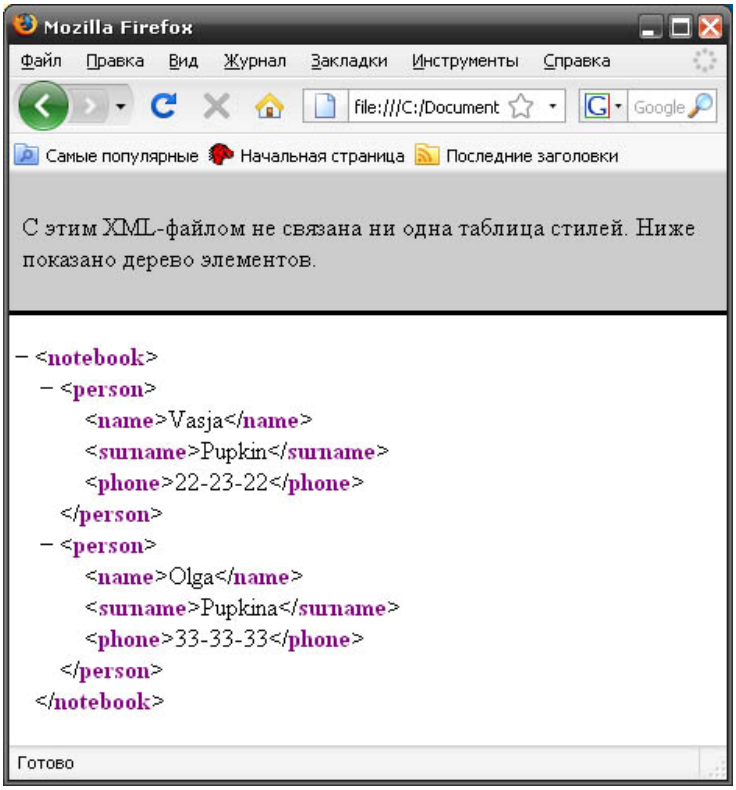
1. Opera



2. Internet Explorer



3. Mozilla Firefox



Ну і на закінчення, варто зауважити, що вкладеність елементів в XML строго контролюється. Тобто, якщо Ви в HTML могли написати щось подібне наступному:

```
<a>La-la-la <b> Tru-la-la </a> He-he-he </b>
```

То в XML, як і в XHTML такий варіант не пройде, оскільки такий XML-документ буде не дійсним, тобто синтаксично невірним. Всі вкладені теги при побудові XML-документів потрібно контролювати, чітко дотримуватись правил побудови. Отже, вищенаписаний код потрібно переписати наступним чином:

```
<a>La-la-la <b> Tru-la-la </b> He-he-he </a>
```

4. Візуальне представлення XML документа за допомогою CSS. Інтеграція XML з HTML (острови даних)

В мові HTML часто використовуються таблиці стилів CSS (Cascading Style Sheet) або CSS2, які задають загальні правила оформлення HTML-документів: колір, фон, шрифт тощо. Виконання цих правил дозволяє документам задати єдиний стиль оформлення. Оскільки така практика оформлення документів є дуже зручною, то вона також повинна була бути реалізована в XML. Це означає, що до XML-документів також можна застосовувати каскадні таблиці стилів CSS. Для цього в мові розмітки XML введена інструкція по обробці **xml:stylesheet**.

Наведемо невеличкий приклад форматування XML-документа на базі розробленої нами записної книжки.

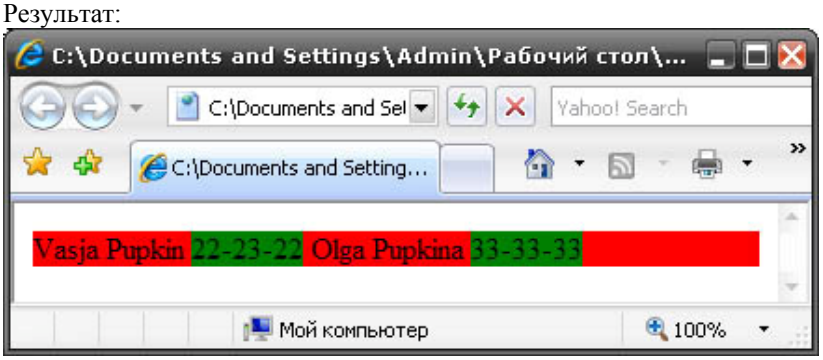
style.css

```
notebook
{
	background-color: red;
	color: #000000;
	display: block;   //блочне форматування
}
phone
{
	background-color: green;
	color: #000000;
	block: inline;  //лінійне форматування
}
```

notebook.xml

```
<?xml version="1.0" encoding="windows-1251" standalone="yes"?>
<?xml-stylesheet type="text/css" href="style.css" ?>
<notebook>
	<person>
		<name>Vasja</name>
		<surname>Pupkin</surname>
		<phone>22-23-22</phone>
	</person>
	<person>
		<name>Olga</name>
		<surname>Pupkina</surname>
		<phone>33-33-33</phone>
	</person>
</notebook>
```

Результат:



Хоча каскадні таблиці стилі CSS і можна використовувати в XML, але погодьтесь, - це не досить зручно та не досить привабливо виглядає. Крім того, стилі CSS визначають засоби показу HTML-документа в вікні браузера, тобто його візуалізацію, а мова XML відображає структуру документа, нічого не кажучи про його зовнішній вигляд. Тому для відображення та форматування XML-документів була розроблена окрема спеціальна мова XSL (XML Stylesheet Language), яка являється реалізацією XML. Вивченню цієї мови ми присвятимо окремі пари.

5. Інтеграція XML з HTML (острови даних)

Крім представлення XML-документів засобами CSS, існує також поняття «**островів даних» (data islands)** – це технологія інтеграції XML даних в HTML-документ, яка розроблена фірмою Microsoft сумісно з корпорацією W3C. З точки зору даної технології, XML-документ виступає в якості джерела даних, яке виводиться в відформатованому вигляді на HTML-сторінку. XML-дані можуть бути безпосередньо описані в HTML-коді, або ж винесені в окремий файл з розширенням .xml.

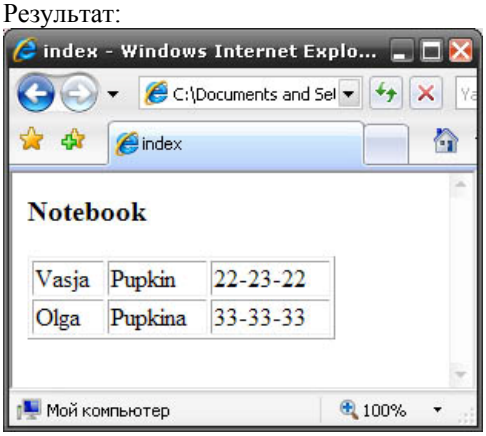
Для прикладу винесемо на сторінку HTML дані ранішестворeного нами XML-документа:

Index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>index</title>
</head>
```

```
<body>
      <xml id="lib" src="notebook.xml"></xml>

      <h3>Notebook</h3>
      <table datasrc="#lib" border="1" width="200">
            <tr>
                  <td><span datafld="name"></span></td>
                  <td><span datafld="surname"></span></td>
                  <td><span datafld="phone"></span></td>
            </tr>
      </table>
</body>
</html>
```

Результат:



А тепер розберемо написаний код:

1. Всю роботу по вбудові XML-документа в HTML виконує елемент **<xml>**, в атрибуті src якого вказується повних шлях до необхідного XML-документа. Даний тег має ідентифікатор, по якому можна звернутись до власне вмісту інтегрованого XML-документа.
2. Вміст XML-документа ми виводимо в таблицю, тому їй необідно вказати джерело даних. Для звернення до нашого джерела даних (XML-документа) використовується атрибут **datasrc**, якому необхідно вказати значення типу # + **id_xml**.
3. Для отримання інформації з полів XML-документа використовується атрибут **datafld**, значенням якого є назва необхідного елемента XML-документа. Атрибут datafld мають не всі теги HTML, тому слід спочатку переглянути необхідну документацію. Наприклад, та мають такий атрибут, а тег <p> - ні.

Як вже згадувалось вище, на HTML-сторінку можна повністю винести весь вміст XML-документа. В такому разі наш код перепишеться.

Index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>index</title>
</head>
<body>
    <xml id="lib">
        <?xml version="1.0" encoding="windows-1251" standalone = "yes"?>
        <notebook>
          <person>
            <name>Vasja</name>
            <surname>Pupkin</surname>
            <phone>22-23-22</phone>
          </person>

          <person>
            <name>Olga</name>
            <surname>Pupkina</surname>
            <phone>33-33-33</phone>
```

```
        </person>
      </notebook>
    </xml>

    <h3>Notebook</h3>
    <table datasrc="#lib" border="1" width="200">
        <tr>
            <td><div datafld="name"></div></td>
            <td><div datafld="surname"></div></td>
            <td><div datafld="phone"></div></td>
        </tr>
    </table>
</body>
</html>
```

Та не крепко радійте, адже таке візуальне представлення XML даних підтримується лише браузером Internet Explorer. Крім того, в створеній таблиці неможливо встановити заголовки полів, - вони постійно будуть повторюватись, оскільки процесор XML при відображенні кожного рядка таблиці буде рекурсивно звертатись до XML-документа та заново зчитувати та виконувати код, який написний всередині елемента <table>.

Отже, таку інтеграцію робити можна, але не варто. І тут ще раз варто повторитись: для відображення та форматування даних XML-документів потрібно використовувати мову XSL, а не HTML.

6. Формально-вірні XML документи. Введення в DTD та схеми

Як було вже сказано в поперніх розділах, якщо XML документ не порушує визначені правила та стандарт, то він називається **формально-вірним** або **дійсним**, і всі синтаксичні аналізатори, які призначені для розбору XML-документів, зможуть працювати з ним коректно.

Однак, крім перевірки на формальну відповідність граматиці мови, в документі можуть бути присутні засоби контролю над вмістом документа. Ці засоби контролю слідкують за дотриманням правил, які визначають необхідні відносини між елементами (вкладеність, порядок слідування, кількість та типи атрибутів тощо), що формують загальну структуру документа. Іншими словами, вони використовуються для стандартизації того, що ми пишемо, та контролюють відповідність XML-документів описаним правилам. Якщо документ пройшов і таку відповідність, тоді його називають ще і **валідним** або **коректним**.

Наведемо приклад того, коли такий контроль дійсно необхідний і чи потрібен він взагалі. Припустимо, що наш славнозвісний Вася Пупкін з Ольгою Пупкіною вирішили написати якусь непогану програмку, яка використовує XML-документи для збереження даних. Наявні дані для заповнення вони поділили між собою, щоб було швидше. Оля дуже постаралась і описала та заповнила доволі багато даних в XML-документі. Наприклад, сторіночок так з 20. Вася також часу не гаяв та описав та заповнив даними також близько 3 сторінок (наскільки вже хватило ☺).

І от вирішили вони переглянути, що ж вони описали та заповнили та звести все в одному місці. Як виявилось, читати думки один одного вони не вміють та використали теги для опису структури документа з різними іменами. Оля поназивала все гарно на англійський мові (product, name, price тощо), а Вася як вже вмів – транслітом (tovar, smja, cyna тощо). Що ж робити? Якось потрібно виправляти ситуацію.

Ситуація неприємна та її не було б, якщо перед тим як заповнювати XML-документи даними, вони домовились та написали стандарт майбутнього XML-документа, тобто набір основних правил його створення. В такому випадку вони б знали, що при описі XML-документів використовуються певні обмеження щодо зандання імен елементів, атрибутів тощо.

Для такої стандартизації до XML-документа слід вкласти формальний опис, зроблений на мові, яку розуміє синтаксичний аналізатор. Такий опис можна зробити різними способами, адже на сьогоднішній день створено кілька мов опису схем XML-документа, які часто називають спільною назвою – **«валідатори»**.

На сьогоднішній день існує два основних типи **валідаторів**:

1. **DTD (Document Type Definition – оголошення типу документа)** – з'явилась ще в першій версії XML для визначення логічної структури XML-документа, використовуючи для цього набір формальних правил.
2. **XML Schema** - які з'явились зовсім недавно (починаючи з 2001 р.) та дозволяють описати структуру XML-документа, визначивши використовувані типи даних, кількість елементів і атрибутів, порядок їх слідування тощо.

7. Домашнє завдання

Завдання 1

Написати XML документ для збереження даних про рецепти, тобто книгу рецептів. Кожен **рецепт (recipe)** буде мати наступну структуру, тобто зберігати дані про:

- **назва (title)** – назва рецепту;
- **фото (photo)** – необов'язковий елемент, який містить фото блюда;
- **інгридієнти (ingredients)**:
  - **інгридієнт (item)** – назва інгридієнту;

- **кількість (quantity)** – необхідна кількість;

- **приготування (preparation)** – власне сам рецепт;
- **порції (serving)** – вихід;
- **калорійність (energy)** – необов'язковий елемент, що містить інформацію про калорійність блюда:
  - **білки (proteins)**;
  - **жири (fats)**;
  - **вуглеводи (carbs)**.

Варто відмітити, що інгридієнтів при приготуванні може бути більше одного, а дочірні елементи в елементі energy (якщо він присутній) можуть розміщуватись в будь-якому порядку. Крім того, ряд елементів можуть мати додаткові властивості, які в XML-документі відображаються за допомогою **атрибутів**:

- **батьківський елемент** кожного окремого рецепту (recipe) обов'язково буде мати **ідентифікатор**;
- елемент **quantity** містить обов'язково інформацію про **одиницю виміру** (грами, літри, ст.ложка тощо);
- елемент **preparation** може містити дані (по бажанню) про:
  - **ступінь складності**, який може містити одне з наступних значень:
    - для початківців;
    - для досвідчених;
    - для майстрів.

    По замовчуванню приймається "для початківців".

  - **час приготування**;
  - **додаткові характеристики** з можливими значеннями:
    - для вегетаріанців;
    - для діабетиків;
    - низькокалорійно;
    - можна заморожувати.

Завдання 2

Оформити створений XML документ за допомогою CSS.