Lecture 8
XML Tutorial [2]

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hknu.ac.kr/

HTML 문서로부터 정보 가져오기

XML 응용 사례 (1/8)

웹 애플리케이션에 대한 요청

- ✓ 사용자 (클라이언트)로 부터
- ✓ 다른 웹 애플리케이션 → "웹 애플리케이션 모델"로부터
  - 웹 애플리케이션이 다른 웹 애플리케이션과 연결

응용 사례

기상청 홈페이지로부터 날씨(온도) 정보를 가져와서 온도를 토대로 특정 조건이 만족되면 서비스를 받는 고객들에게 이 조건을 경고(Warning) 해주는 PowerWarning 애플리케이션을 작성하시오.

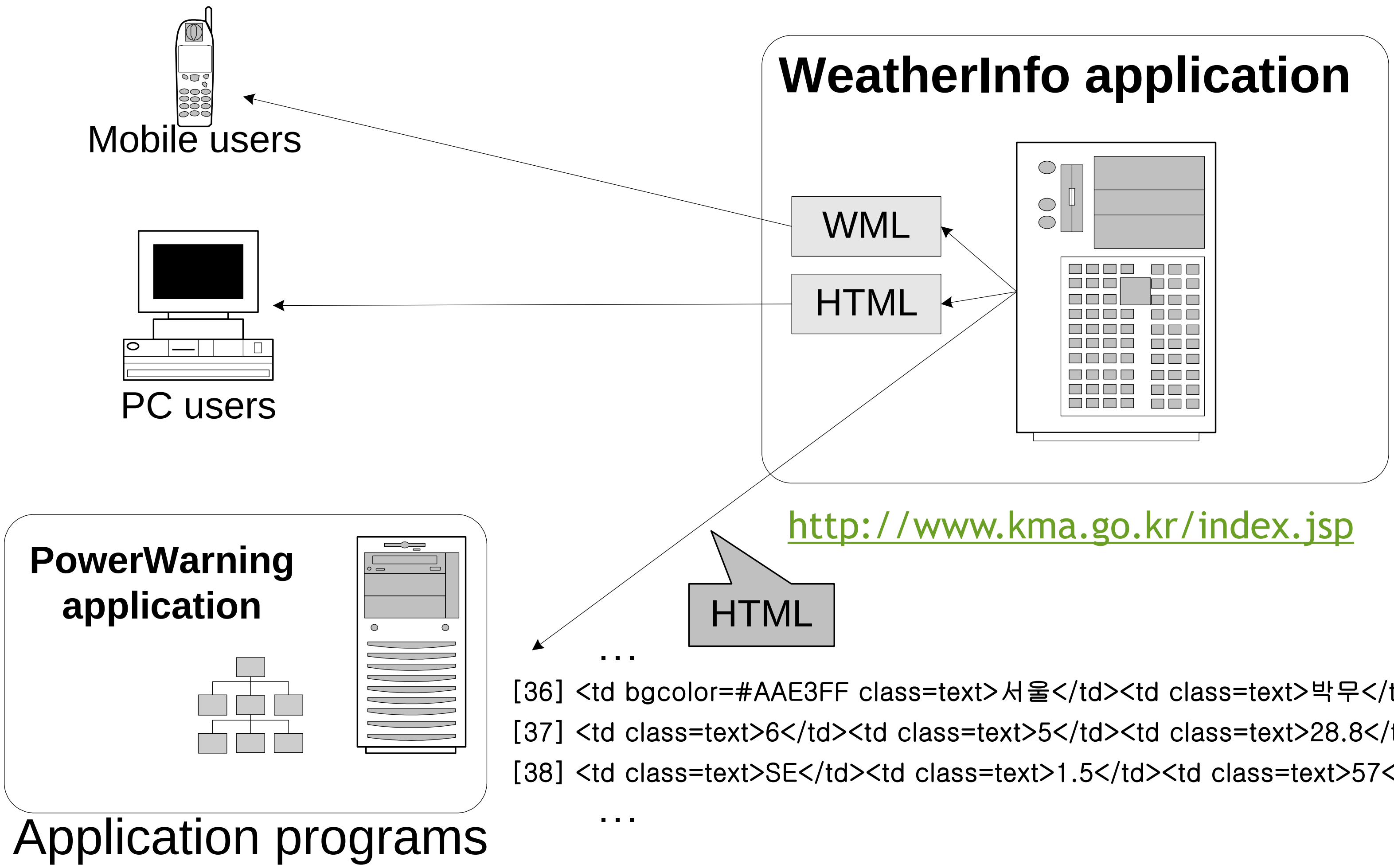
XML 응용 사례 (2/8)

웹 애플리케이션에서 필요한 작업

- ✓ 날씨(온도) 정보를 제공하는 웹 페이지를 매시간 얻어 온다
- ✓ 가져온 페이지로부터 관련된 온도 정보를 추출한다
- ✓ 온도가 3시간 이상 연속하여 섭씨 30도를 넘는지 점검한다

XML 응용 사례 (3/8)

HTML 구문 분석을 통한 서비스



http://www.kma.go.kr/index.jsp

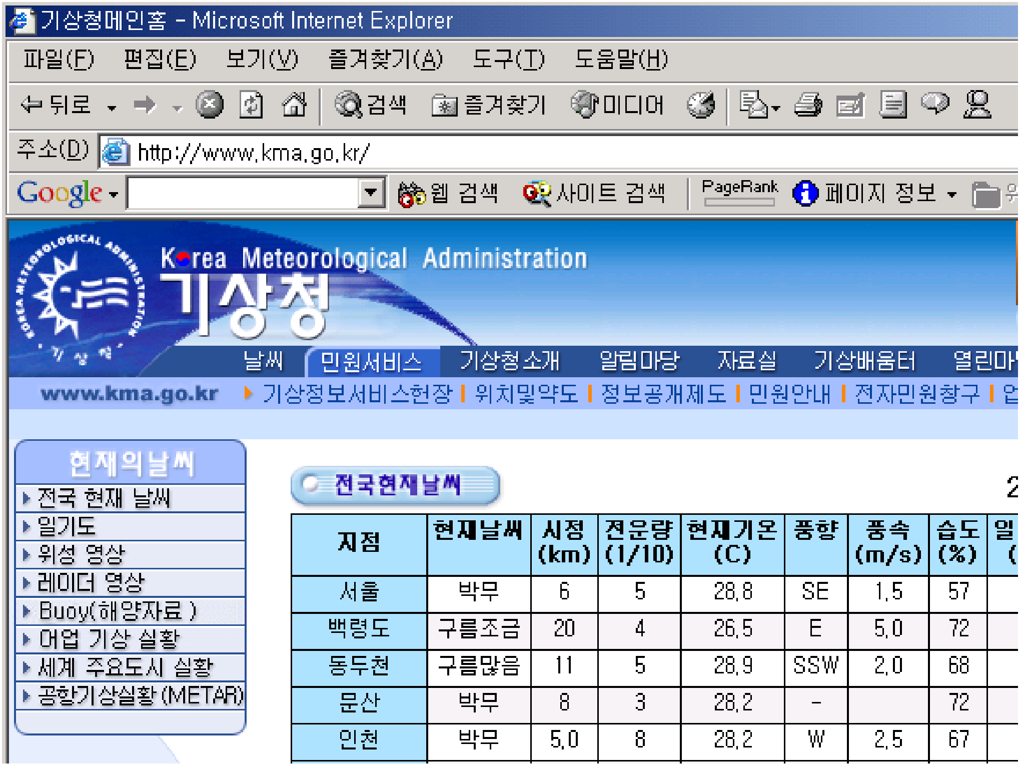
```
   …
[36] <td bgcolor=#AAE3FF class=text>서울</td><td class=text>박무</td>
[37] <td class=text>6</td><td class=text>5</td><td class=text>28.8</td>
[38] <td class=text>SE</td><td class=text>1.5</td><td class=text>57</td>
   …
```

XML 응용 사례 (4/8)

기상청 홈페이지로부터 가져온 HTML 페이지 샘플:

http://www.kma.go.kr/weather/observation/currentweather.jsp

```
. . .
[35] <tr align=center bgcolor=#ffffff>
[36] <td bgcolor=#AAE3FF class=text>서울</td><td class=text>박무</td>
[37] <td class=text>6</td><td class=text>5</td><td class=text>28.8</td>
[38] <td class=text>SE</td><td class=text>1.5</td><td
class=text>57</td>
[39] <td class=text>&nbsp;</td><td class
[40] <td class=text>1007.9</td></tr>
[41] <tr align=center bgcolor=#f2f2f2>
[42] <td bgcolor=#AAE3FF class=text>백령
[43] <td class=text nowrap>구름조금</td>
[44] <td class=text>20</td><td class=te
class=text>26.5</td>
[45] <td class=text>E</td><td class=text
[46] <td class=text>&nbsp;</td><td class
[47] <td class=text>1007.8</td></tr>
. . .
```


기상청메인홈 - Microsoft Internet Explorer
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
뒤로 검색 즐겨찾기 미디어
주소(D) http://www.kma.go.kr/
Google 웹 검색 사이트 검색 PageRank 페이지 정보
Korea Meteorological Administration
기상청
날씨 민원서비스 기상청소개 알림마당 자료실 기상배움터
www.kma.go.kr 기상정보서비스현장 위치및약도 정보공개제도 민원안내 전자민원창구
현재의날씨
전국 현재 날씨
일기도
위성 영상
레이더 영상
Buoy(해양자료)
어업 기상 실황
세계 주요도시 실황
공항기상실황(METAR)
전국현재날씨

지점	현재날씨	시정 (km)	전운량 (1/10)	현재기온 (C)	풍향	풍속 (m/s)	습도 (%)
서울	박무	6	5	28.8	SE	1.5	57
백령도	구름조금	20	4	26.5	E	5.0	72
동두천	구름많음	11	5	28.9	SSW	2.0	68
문산	박무	8	3	28.2	-		72
인천	박무	5.0	8	28.2	W	2.5	67

XML 응용 사례 (5/8)

<u>방법 1:</u>

- ✓ 서울의 현재 온도를 얻기 위해서는 웹 페이지의 37행 57열로부터 시작하여 다음의 < 를 만날 때까지 읽어 들인다

<u>방법 2:</u>

- ✓ 서울의 현재 온도를 얻기 위해서는 첫 번째 <table> 태그로 간 다음,
  표 안의 네 번째 <tr>로 간다.
  그리고 행 안의 다섯 번째 <td>로 간다

XML 응용 사례 (6/8)

HTML - 데이터 교환 포맷으로서의 문제점

- 문서의 표현 구조를 나타내도록 설계
  - ✓ 하나의 문서를 헤더, 타이틀, 문단, 표제 등과 같이 문서에 대한 구조적 부분으로 나누어 표현
  - ✓ 현재 온도와 같은 논리적 데이터(logical data)를 표현하는 태그가 없음

- HTML 페이지에서 데이터를 다루는 방식
  - ✓ 웹 브라우저 상에서 페이지만 적당하게 보이면 만족하는 경향때문에 HTML 구문에 덜 민감
  - ✓ HTML 페이지의 구문 분석을 통한 **데이터 추출이 어려움**

XML 응용 사례 (7/8)

방법 3: 날씨 정보를 XML로 저장

- 서울의 현재 온도를 얻기 위해서는 <CurrTemp> 태그로 간다
  - ✓ XML: 데이터를 지정된 위치에 직접적으로 표현할 수 있는 구조 정의
  - ✓ 날씨 데이터에 대한 논리적 표현
  - ✓ 페이지의 표현 구조에 독립적, 화면에 표시되는 방식과도 독립적

```
<?xml version="1.0"?>
<!DOCTYPE WeatherReport System http://www.kma.go.kr/WeatherReport.dtd>
<WeatherReport>
                 <City>Seoul</City>
                 <Date>Sat Sept 10 2002</Date>
                 <Time>11 AM ROK</Time>
                 <CurrTemp unit="Celsius">27</CurrTemp>
                 <High unit="Celsius">33</High>
                 <Low unit="Celsius">17</Low>
</WeatherReport>
```

XML 응용 사례 (8/8)

<u>방법 3이 잘 동작하려면</u>

- ✓ 기상청 날씨 정보 페이지를 참조하는 모든 애플리케이션에서도 이 태그를 이해할 수 있어야 한다.
- ✓ 기상청 날씨 정보 페이지에서 사용한 태그들은 이를 참조하는 모든 애플리케이션에서 이 태그 사용에 동의할 수 있도록 Document Type Definition(DTD)이라는 문법에 맞게 정의되어야 한다.
- ✓ DTD는 공표될 수 있다.
  - 이 사이트를 참조하는 모든 애플리케이션에서 동일한 방식으로 사용
- ✓ 공표된 DTD는 마치 http://www.kma.go.kr/에 있는 기상청 사이트의 웹 애플리케이션에 대한 스펙(Specification)처럼 동작한다.

XML Validation

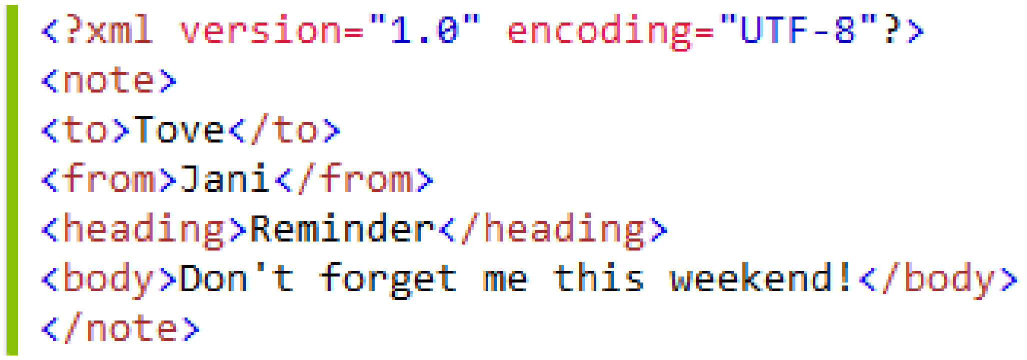
http://www.w3schools.com/xml/

XML Document Types

XML with correct syntax is "**Well Formed**" XML.
“**Valid**” XML document must also confirm to a **specified document type.**

Well Formed XML Documents

- ✓ XML documents must have a root element
- ✓ XML elements must have a closing tag
- ✓ XML tags are case sensitive
- ✓ XML elements must be properly nested
- ✓ XML attribute values must be quoted



```xml
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

XML Validator

- ✓ Checks the syntax of your XML files
- ✓ Uses the XML validator to **syntax-check** your XML.

XML Errors Will Stop You

- ✓ Errors in XML documents will **stop** your XML applications
- ✓ W3C XML specification states that a program should stop processing an XML document if it finds an error
- ✓ HTML browsers will display HTML documents with errors (like missing end tags)
- ✓ **With XML, errors are not allowed**

Valid XML Documents

Valid XML document is not the same as a well formed XML document:

- ✓ First rule, **it must be well formed**
- ✓ Second rule, **it must conform to a document type**

Rules that defines legal elements and attributes for XML documents are often called **document definitions**, or **document schemas.**

Document Definitions

There are different types of document definitions that can be used with XML:

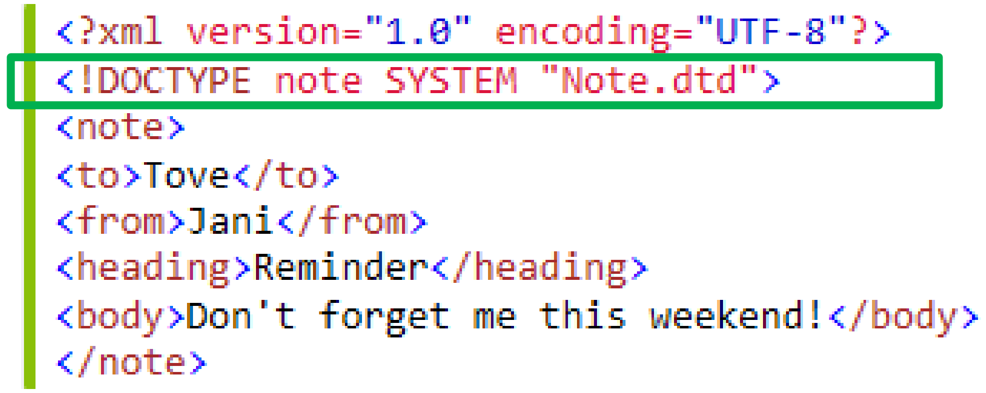
- ✓ Original Document Type Definition (**DTD**)
- ✓ Newer, and XML based, **XML Schema**

XML DTD

XML document with correct syntax is called "Well Formed".
XML document validated against a DTD is "**Well Formed**" and "**Valid**".

Valid XML Documents

"Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a DTD:



```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

DOCTYPE declaration is a **reference** to an external DTD file.

XML DTD

DTD의 목적 - XML 문서의 구조 정의:

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

#PCDATA means parse-able text data.

DTD above is interpreted like this:

- ✓ !DOCTYPE **note** defines that the root element of the document is note
- ✓ !ELEMENT **note** defines that the note element contains four elements: "to, from, heading, body"
- ✓ !ELEMENT **to** defines the to element to be of type "#PCDATA"
- ✓ !ELEMENT **from** defines the from element to be of type "#PCDATA"
- ✓ !ELEMENT **heading** defines the heading element to be of type "#PCDATA"
- ✓ !ELEMENT **body** defines the body element to be of type "#PCDATA"

Using DTD for Entity Declaration

Doctype declaration can also be **used to define special characters and character strings,** used in the document:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE note [
<!ENTITY nbsp "&#xA0;">
<!ENTITY writer "Writer: Donald Duck.">
<!ENTITY copyright "Copyright: W3Schools.">
]>

<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
<footer>&writer;&nbsp;&copyright;</footer>
</note>
```

1. Try it yourself >>

An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

Why Use a DTD?

With a DTD, your XML files can **carry a description of its own format**

With a DTD, independent groups of people can **agree on a standard for interchanging data**

With a DTD, you can **verify that the data you receive from the outside world is valid**

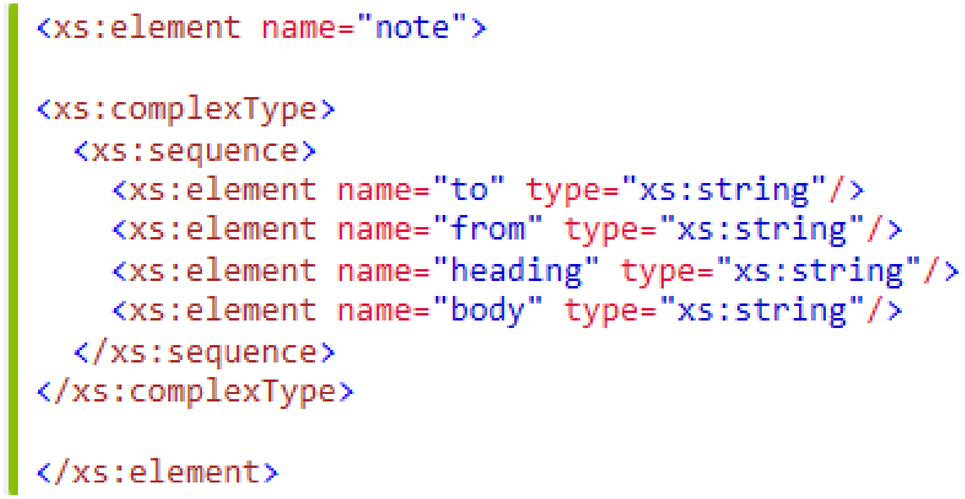
XML Schema

XML Schema describes the **structure of an XML document,** just like a DTD

XML document with correct syntax is called "Well Formed"

XML document validated against an XML Schema is both "Well Formed" and "Valid"

XML Schema is an XML-based alternative to DTD:



```
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

Schema above is interpreted like this:

- ✓ <xs:element name="**note**"> defines the element called "note"
- ✓ <xs:complexType> the "**note**" element is a complex type
- ✓ <xs:sequence> the **complex type** is a sequence of elements
- ✓ <xs:element name="**to**" type="xs:string"> the element "to" is of type string (text)
- ✓ <xs:element name="**from**" type="xs:string"> the element "from" is of type string
- ✓ <xs:element name="**heading**" type="xs:string"> the element "heading" is of type string
- ✓ <xs:element name="**body**" type="xs:string"> the element "body" is of type string

XML Schemas are More Powerful than DTD

- ✓ XML Schemas are written in XML
- ✓ XML Schemas are extensible to additions
- ✓ XML Schemas support data types
- ✓ XML Schemas support namespaces

Why Use an XML Schema? ***DTD와 동일!!***

- ✓ With XML Schema, your XML files can carry a description of its own format
- ✓ With XML Schema, independent groups of people can agree on a standard for interchanging data
- ✓ With XML Schema, you can verify data

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the **support for data types:**

- ✓ It is easier to **describe** document content
- ✓ It is easier to **define** restrictions on data
- ✓ It is easier to **validate** the correctness of data
- ✓ It is easier to **convert** data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are **written in XML:**

- ✓ You don't have to learn a new language
- ✓ You can use your XML editor to edit your Schema files
- ✓ You can use your XML parser to parse your Schema files
- ✓ You can manipulate your Schemas with the XML DOM
- ✓ You can transform your Schemas with XSLT

XML Applications

XML Document Used: "cd_catalog.xml" file.

Display XML Data in an HTML Table

This example loops through each <CD> element, and display the values of the <ARTIST> and the <TITLE> elements in an HTML table:

2. Try it yourself »

Display the First CD in an HTML div Element

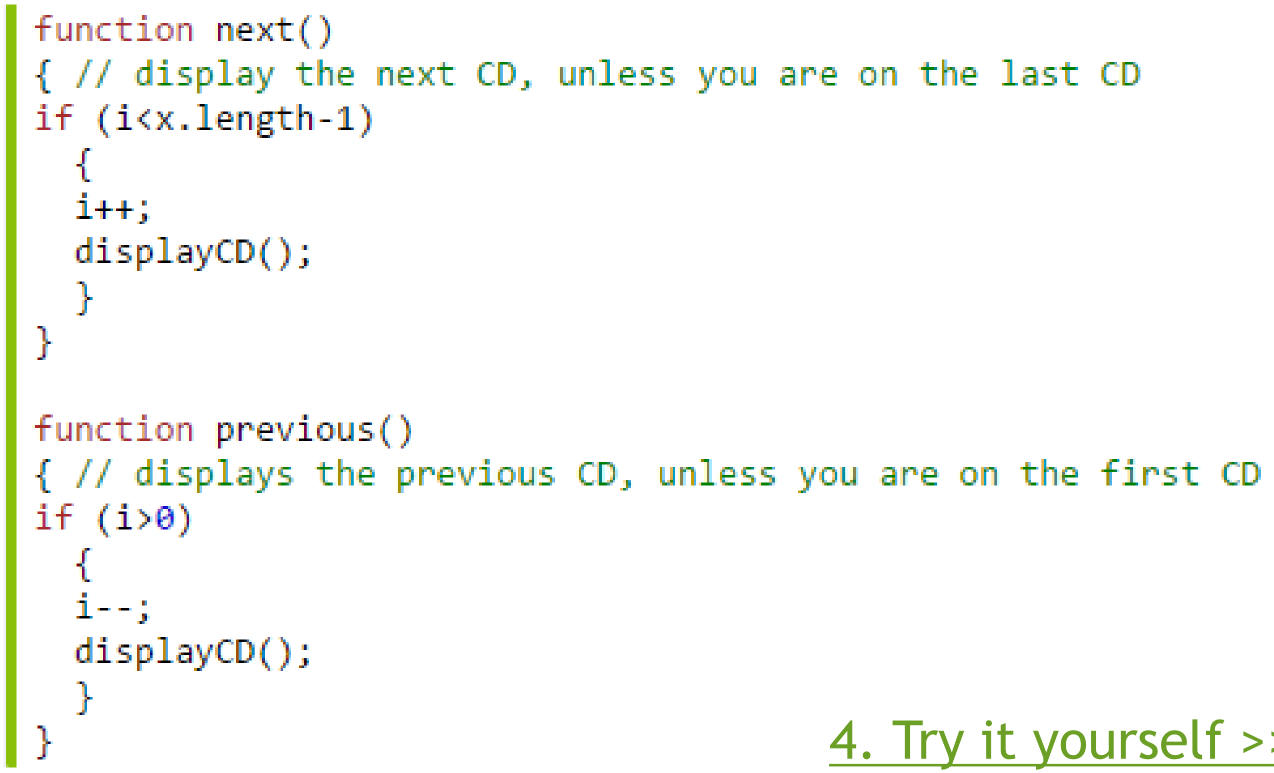
Uses a function to display the first CD element in an HTML element with id="showCD“.

```
displayCD(0);

function displayCD(i) {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
            myFunction(xmlhttp, i);
        }
    }
    xmlhttp.open("GET", "cd_catalog.xml", true);
    xmlhttp.send();
}

function myFunction(xml, i) {
    var xmlDoc = xml.responseXML;
    x = xmlDoc.getElementsByTagName("CD");
    document.getElementById("showCD").innerHTML =
    "Artist: " +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "<br>Title: " +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "<br>Year: " +
    x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```

3. Try it yourself »

Navigate Between the CDs

To navigate between the CDs, add a next() and previous() function:

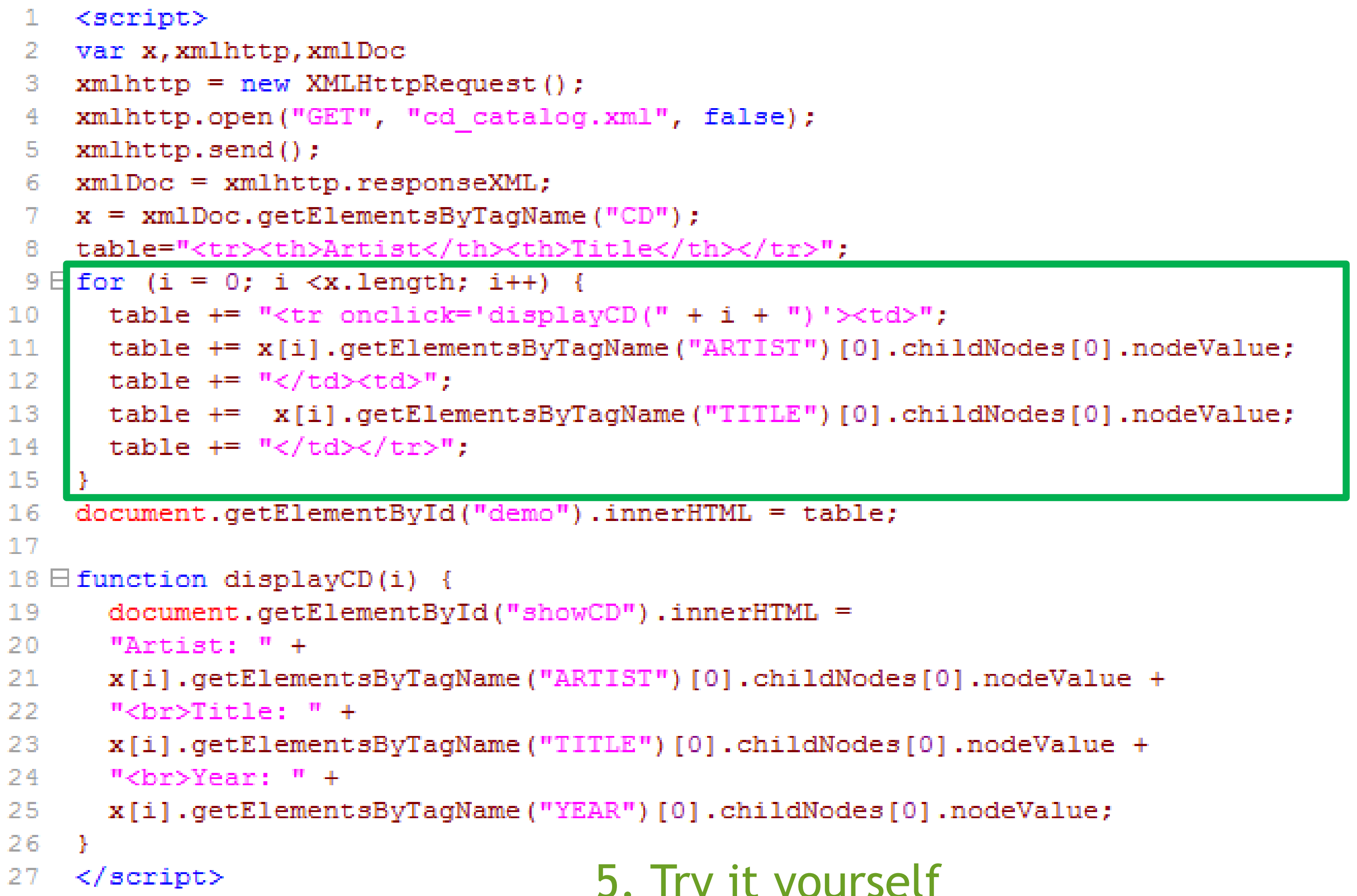


```
function next()
{ // display the next CD, unless you are on the last CD
if (i<x.length-1)
  {
  i++;
  displayCD();
  }
}

function previous()
{ // displays the previous CD, unless you are on the first CD
if (i>0)
  {
  i--;
  displayCD();
  }
}
```

4. Try it yourself >>

Show Album Information When Clicking On a CD

The last example shows how you can show album information when the user clicks on a CD:



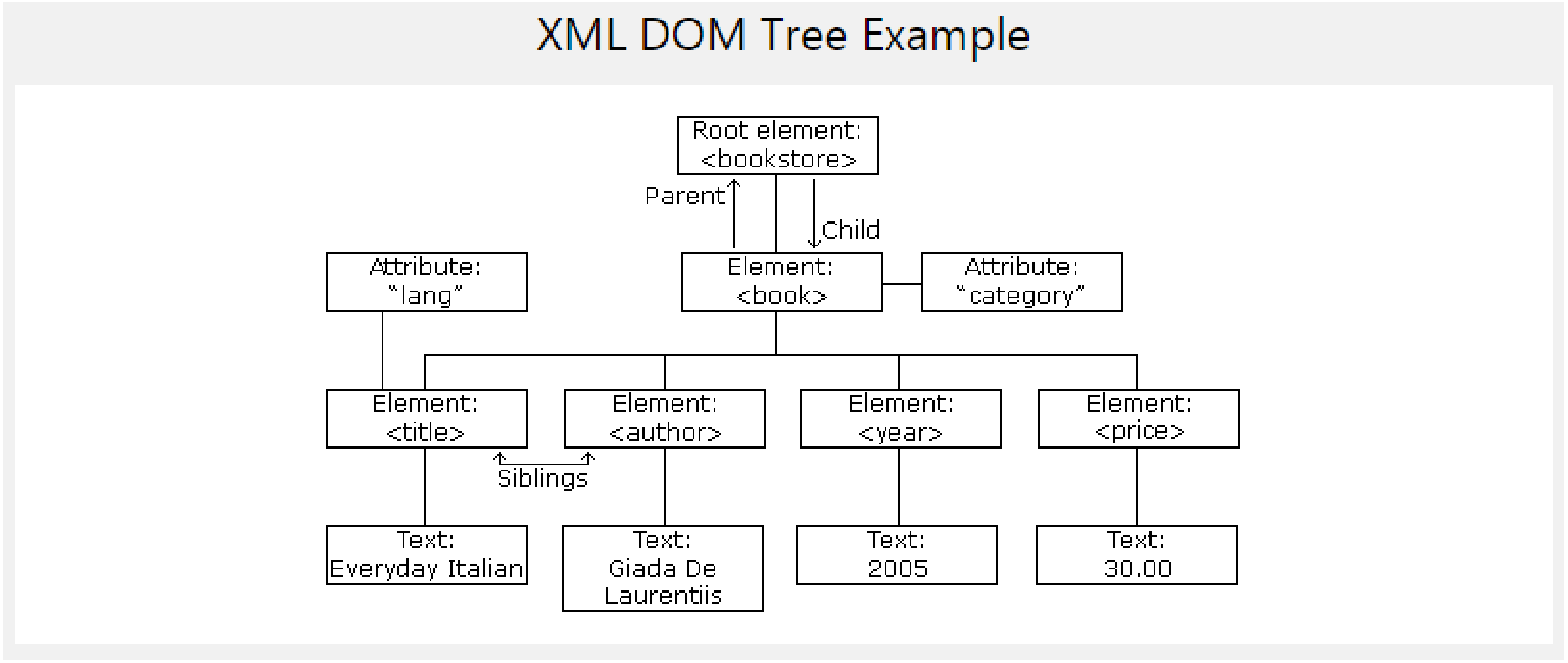
```
<script>
var x,xmlhttp,xmlDoc
xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "cd_catalog.xml", false);
xmlhttp.send();
xmlDoc = xmlhttp.responseXML;
x = xmlDoc.getElementsByTagName("CD");
table="<tr><th>Artist</th><th>Title</th></tr>";
for (i = 0; i <x.length; i++) {
  table += "<tr onclick='displayCD(" + i + ")'><td>";
  table += x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue;
  table += "</td><td>";
  table +=  x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue;
  table += "</td></tr>";
}
document.getElementById("demo").innerHTML = table;

function displayCD(i) {
  document.getElementById("showCD").innerHTML =
  "Artist: " +
  x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "<br>Title: " +
  x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "<br>Year: " +
  x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
</script>
```

5. Try it yourself

XML DOM

XML DOM Tutorial

DOM defines a standard for accessing and manipulating documents
XML DOM presents an XML document as a **tree-structure**
HTML DOM presents an HTML document as a tree-structure
Understanding the DOM is a must for anyone working with HTML or XML



What is the DOM?

DOM defines a standard for accessing documents like XML and HTML:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

DOM is separated into 3 different parts/levels:

- ✓ Core DOM - standard model for any structured document
- ✓ XML DOM - standard model for XML documents
- ✓ HTML DOM - standard model for HTML documents

DOM defines the objects and properties of all document elements, and the methods (interface) to access them.

HTML DOM

- ✓ HTML DOM defines a standard way for accessing and manipulating HTML documents.
- ✓ All HTML elements can be accessed through the HTML DOM.
- ✓ The HTML DOM defines the **objects, properties** and **methods** of all HTML elements.

Change the Value of an HTML Element

This example changes the value of an HTML element with id="demo":

```
<h1 id="demo">This is a Heading</h1>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
```

6. Try it yourself »

Loading an XML File

XML file used in the examples below is books.xml.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
    myFunction(xhttp);
    }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();

function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    document.getElementById("demo").innerHTML =
    xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
}
</script>

</body>
</html>
```

7. Try it yourself »

Programming Interface

DOM models XML as a set of node objects.
The nodes can be accessed with JavaScript or other programming languages.
In this tutorial we use **JavaScript**.

Programming interface to the DOM is defined by a set standard properties and methods.

Properties are often referred to as something that is (i.e. nodename is "book").

Methods are often referred to as something that is done (i.e. delete "book").

XML DOM Properties

- ✓ These are some typical DOM properties:
- ✓ **x.nodeName** - the name of x
- ✓ **x.nodeValue** - the value of x
- ✓ **x.parentNode** - the parent node of x
- ✓ **x.childNodes** - the child nodes of x
- ✓ **x.attributes** - the attributes nodes of x

XML DOM Methods

- ✓ **x.getElementsByTagName**(*name*) - get all elements with a specified tag name
- ✓ **x.appendChild**(*node*) - insert a child node to x
- ✓ **x.removeChild**(*node*) - remove a child node from x

Note: In the list above, x is a node object.

XMLHttpRequest Object

All modern browsers have a built-in **XMLHttpRequest** object to request data from a server.

All major browsers have a built-in **XML parser** to access and manipulate XML.

XMLHttpRequest Object

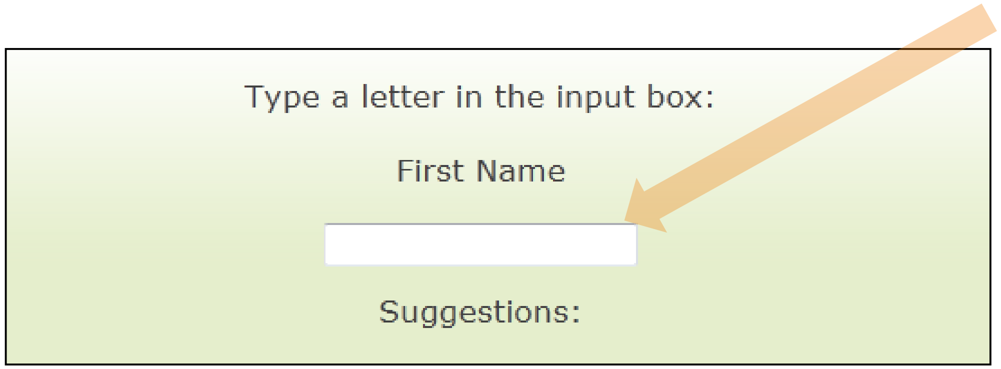
XMLHttpRequest object can be used to request data from a web server.

XMLHttpRequest object is a **developers' dream**, because you can:

- ✓ Update a web page without reloading the page
- ✓ Request data from a server - after the page has loaded
- ✓ Receive data from a server - after the page has loaded
- ✓ Send data to a server - in the background

XMLHttpRequest Example

- ✓ 아래 input 필드에 문자를 타이핑하면 XMLHttpRequest가 서버로 보내진다
- ✓ Name suggestions 이 서버의 파일로부터 리턴된다



Type a letter in the input box:

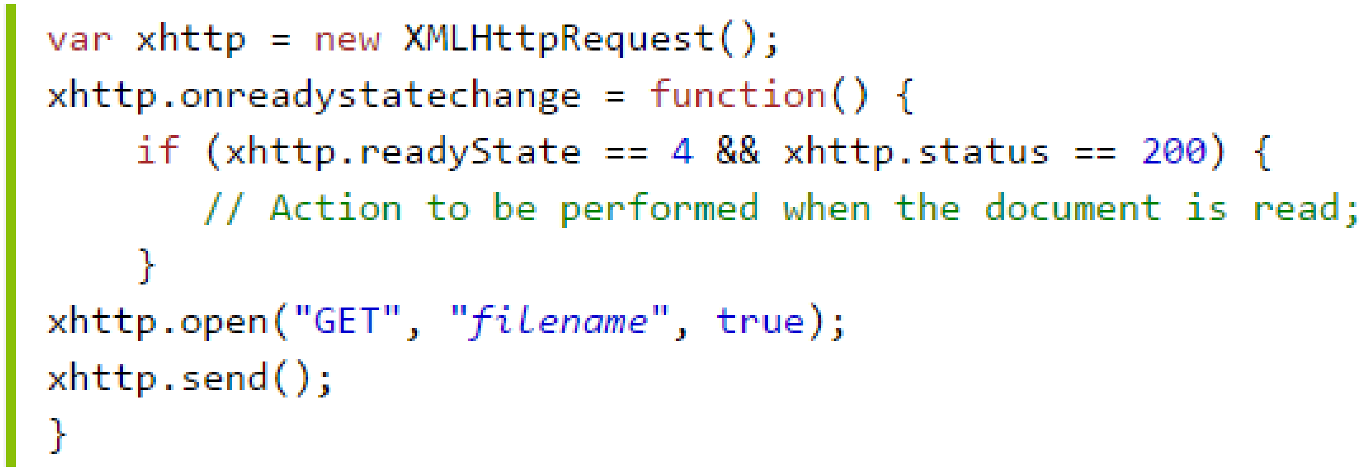
First Name

Suggestions:

Sending an XMLHttpRequest

All modern browsers have a built-in XMLHttpRequest object

A common JavaScript syntax for using it looks much like this:

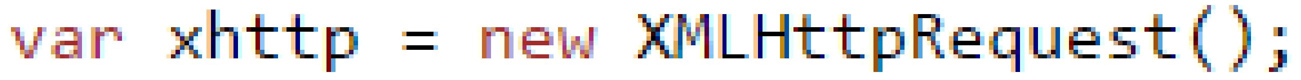


```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
       // Action to be performed when the document is read;
    }
xhttp.open("GET", "filename", true);
xhttp.send();
}
```

8. Try it yourself »

Creating an XMLHttpRequest Object

The first line in the example above creates an XMLHttpRequest **object**:



```
var xhttp = new XMLHttpRequest();
```

onreadystatechange Event

- ✓ **readyState** property holds the status of the XMLHttpRequest
- ✓ **onreadystatechange** event is triggered every time the readyState changes
- ✓ During a server request, the readyState changes from 0 to 4:

  0: request not initialized
  1: server connection established
  2: request received
  3: processing request
  4: request finished and response is ready

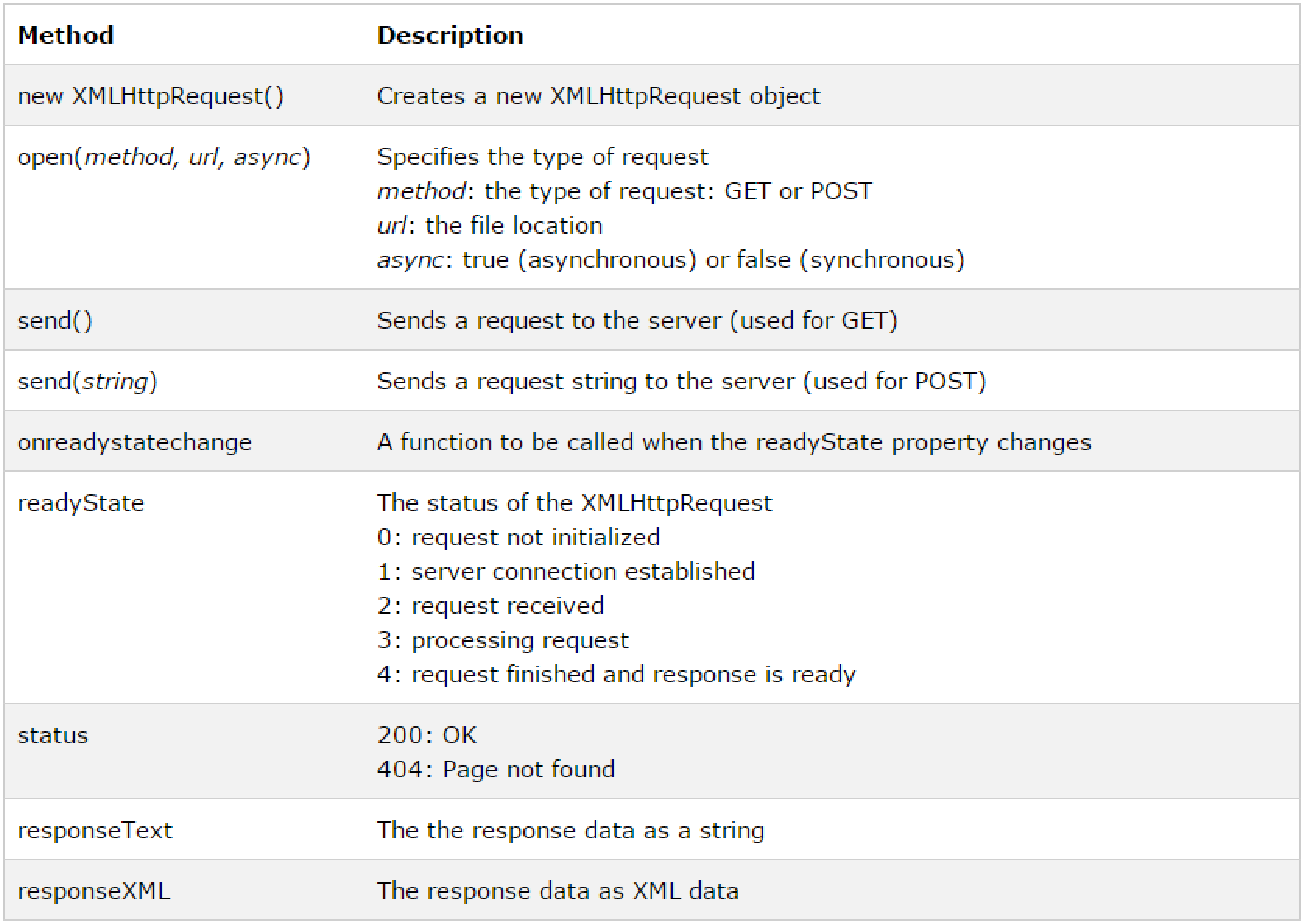
In the onreadystatechange property, specify a function to be executed when the readyState changes:

```
xhttp.onreadystatechange = function()
```

예) When readyState is 4 and status is 200, the response is ready:

```
if (xhttp.readyState == 4 && xhttp.status == 200)
```

XMLHttpRequest Properties and Methods



Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
open(*method, url, async*)	Specifies the type of request *method*: the type of request: GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous)
send()	Sends a request to the server (used for GET)
send(*string*)	Sends a request string to the server (used for POST)
onreadystatechange	A function to be called when the readyState property changes
readyState	The status of the XMLHttpRequest 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: OK 404: Page not found
responseText	The the response data as a string
responseXML	The response data as XML data

Access Across Domains

For security reasons, **modern browsers do not allow access across domains**

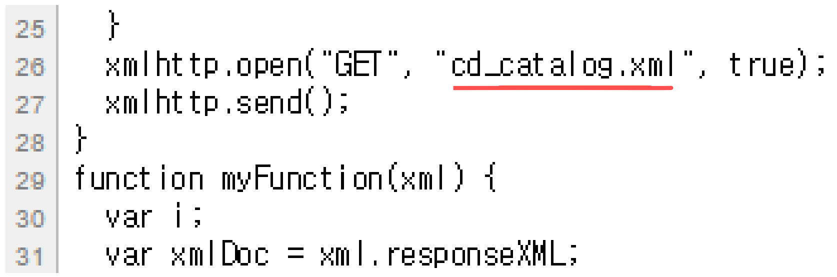
This means that both the web page and the XML file it tries to load, **must be located on the same server**

The examples on W3Schools all open XML files located on the W3Schools domain

If you want to use the example above on one of your own web pages, the XML files you load must be located on your own server

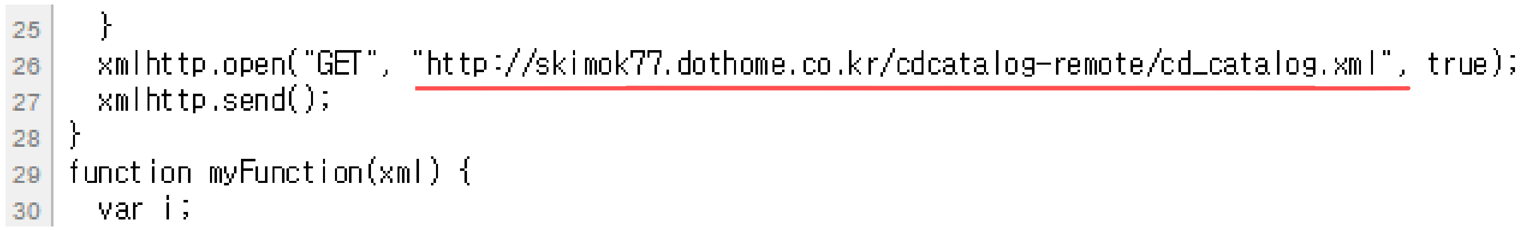
XML in HTML

http://localhost:8080/cdcatalog-local/cd-catalog.html



```
    }
    xmlhttp.open("GET", "cd_catalog.xml", true);
    xmlhttp.send();
}
function myFunction(xml) {
    var i;
    var xmlDoc = xml.responseXML;
```

http://localhost:8080/cdcatalog-local/cd-catalog.html



```
    }
    xmlhttp.open("GET", "http://skimok77.dothome.co.kr/cdcatalog-remote/cd_catalog.xml", true);
    xmlhttp.send();
}
function myFunction(xml) {
    var i;
```

Cross Domain Policy Test:

✓ cd-catalog.html -> localhost:8080/cdcatalog-local/cd-catalog.html
cd_catalog.xml -> remote: my-id.dothome.co.kr/cdcatalog-remote/cd_catalog.xml

responseText Property

responseText property returns the response as a string.
If you want to use the response as a text string, use the responseText property:

```
document.getElementById("demo").innerHTML = xmlhttp.responseText;
```

9. Try it yourself »

responseXML Property

responseXML property returns the response as an XML DOM object.

If you want to use the response as an XML DOM object, use the responseXML property:

```
xmlDoc = xmlhttp.responseXML;
txt = "";
x = xmlDoc.getElementsByTagName("ARTIST");
for (i = 0; i < x.length; i++) {
    txt += x[i].childNodes[0].nodeValue + "<br>";
}
document.getElementById("demo").innerHTML = txt;
```

Request the file cd_catalog.xml and use the response as an XML DOM object:

10. Try it yourself »

GET or POST?

GET is simpler and faster than **POST**, and can be used in most cases.

However, **always use POST requests** when:

- ✓ A cached file is not an option (update a file or database on the server)
- ✓ Sending a large amount of data to the server (POST has no size limitations)
- ✓ Sending user input (which can contain unknown characters), POST is more robust and secure than GET

url - A File On a Server

- ✓ url parameter of the open() method, is an address to a file on a server:

```
xmlhttp.open("GET", "xmlhttp_info.txt", true);
```

Asynchronous - True or False?

To send the request asynchronously, the **async** parameter of the **open()** method has to be set to true:

```
xmlhttp.open("GET", "xmlhttp_info.txt", true);
```

Sending asynchronously requests is a huge improvement for web developers.

Many of the tasks performed on the server are very **time consuming.**

By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:

- ✓ execute other scripts while waiting for server response
- ✓ deal with the response when the response is ready

Async = true

When using async = true, specify a function to execute when the response is ready in the onreadystatechange event:

```
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
       document.getElementById("demo").innerHTML = xmlhttp.responseText;
    }
}
xmlhttp.open("GET", "xmlhttp_info.txt", true);
xmlhttp.send();
```

11. Try it yourself »

Async = false

To use async = false, change the third parameter in the open() method to false:

```
xmlhttp.open("GET", "xmlhttp_info.txt", false);
```

Using async = false is **not recommended**, but for a few small requests this can be ok

NOTE When you use async = false, do NOT write an onreadystatechange function - just put the code after the send() statement:

```
xmlhttp.open("GET", "xmlhttp_info.txt", false);
xmlhttp.send();
document.getElementById("demo").innerHTML = xmlhttp.responseText;
```

12. Try it yourself »

XML Parser

- ✓ All modern browsers have a **built-in XML parser**
- ✓ XML Document Object Model (the XML DOM) contains a lot of methods to access and edit XML
- ✓ However, before an XML document can be accessed, it must be **loaded into an XML DOM object**

XML parser can read plain text and convert it into an XML DOM object

Parsing a Text String

This example **parses a text string into an XML DOM object**, and extracts the info from it with JavaScript:

```
<html>
<body>

<p id="demo"></p>

<script>
var text, parser, xmlDoc;

text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```

13. Try it yourself »

XML DOM – Accessing Nodes

getElementsByTagName() Method

getElementsByTagName() returns all elements with a specified tag name.
Syntax:

```
node.getElementsByTagName("tagname");
```

Example:
The following example returns all <title> elements under the x element:

```
x.getElementsByTagName("title");
```

Note that the example above only returns <title> elements under the x node.
To return all <title> elements in the XML document use:

```
xmlDoc.getElementsByTagName("title");
```

where xmlDoc is the document itself (document node).

DOM Node List

getElementsByTagName() method returns a **node list.**
A node list is an array of nodes:

```
x = xmlDoc.getElementsByTagName("title");
```

<title> elements in x can be accessed by index number.
To access the third <title> you can write:

```
y = x[2];
```

Note: The index starts at 0.

DOM Node List Length

length property defines the length of a node list (the number of nodes).

You can loop through a node list by using the length property:

```
var x = xmlDoc.getElementsByTagName("title");

for (i = 0; i <x.length; i++) {
  // do something for each node
  }
```

14. Try it yourself »

Node Types

- ✓ **documentElement** property of the XML document is the root node
- ✓ **nodeName** property of a node is the name of the node
- ✓ **nodeType** property of a node is the type of the node

Traversing Nodes

The following code loops through the child nodes, that are also element nodes, of the root node:

```
txt = "";
x = xmlDoc.documentElement.childNodes;

for (i = 0; i <x.length; i++) {
  // Process only element nodes (type 1)
  if (x[i].nodeType == 1) {
    txt += x[i].nodeName + "<br>";
  }
}
```

15. Try it yourself »

Navigating Node Relationships

The following code **navigates the node tree** using the **node relationships:**

```
x = xmlDoc.getElementsByTagName("book")[0];
xlen = x.childNodes.length;
y = x.firstChild;

txt = "";
for (i = 0; i <xlen; i++) {
  // Process only element nodes (type 1)
  if (y.nodeType == 1) {
    txt += y.nodeName + "<br>";
  }
  y = y.nextSibling;
}
```

16. Try it yourself »

Example explained:

1. Suppose you have loaded "books.xml" into xmlDoc
2. Get the child nodes of the first book element
3. Set the "y" variable to be the first child node of the first book element
4. For each child node (starting with the first child node "y"):
5. Check the node type. If the node type is "1" it is an element node
6. Output the name of the node if it is an element node
7. Set the "y" variable to be the next sibling node, and run through the loop again

XML DOM Node Information

Get the Value of an Element

The following code **retrieves the text node value** of the first <title> element:

```
var x = xmlDoc.getElementsByTagName("title")[0].childNodes[0];
var txt = x.nodeValue;
```

17. Try it yourself »

Result: txt = "Everyday Italian"

Example explained:

1. Suppose you have loaded "books.xml" into xmlDoc
2. Get text node of the first <title> element node
3. Set the txt variable to be the value of the text node

Change the Value of an Element

The following code changes the text node value of the first <title> element:

```
var x = xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue = "Easy Cooking";
```

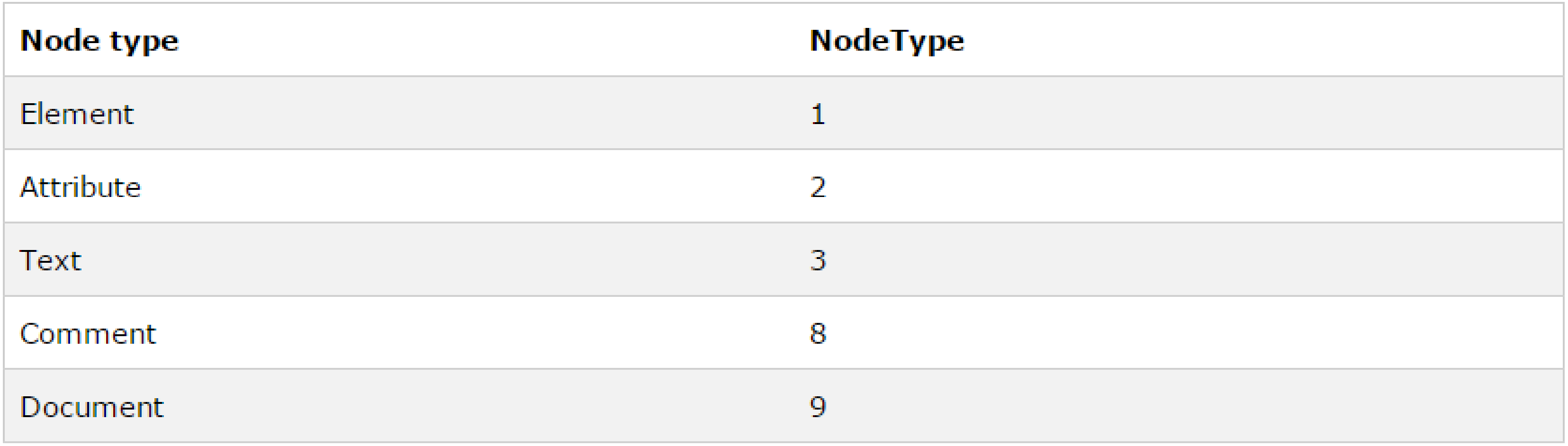
18. Try it yourself »

Example explained:

1. Suppose you have loaded "books.xml" into xmlDoc
2. Get text node of the first <title> element node
3. Change the value of the text node to "Easy Cooking"

nodeType Property

- ✓ **nodeType** property specifies the type of node
- ✓ nodeType is read only
- ✓ The most important node types are:



Node type	**NodeType**
Element	1
Attribute	2
Text	3
Comment	8
Document	9

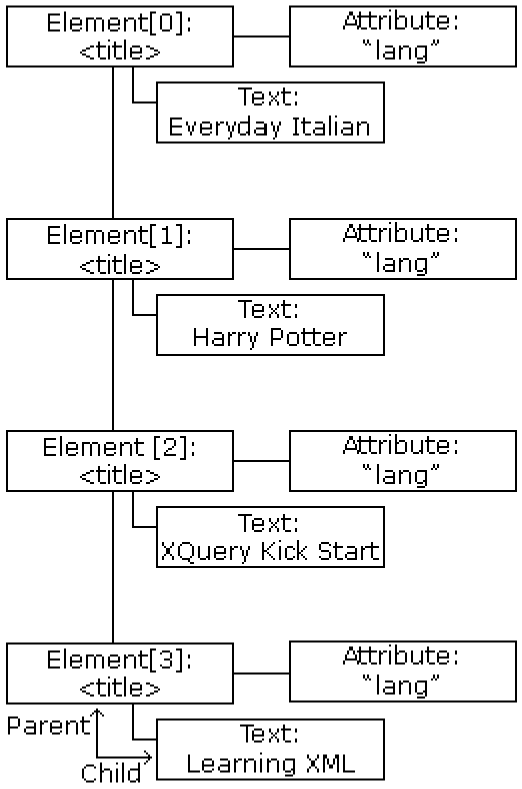
19. Try it yourself >>

XML DOM Node List

When using properties or methods like childNodes or getElementsByTagName(), a node list object is returned.

- ✓ A node list object represents a list of nodes, in the same order as in the XML
- ✓ Nodes in the node list are accessed with index numbers starting from 0
- ✓ The following image represents a node list of the <title> elements in "books.xml":

```
x = xmlDoc.getElementsByTagName("title");
```



- ✓ The following code fragment returns the text from the first <title> element in the node list (x):

```
var txt = x[0].childNodes[0].nodeValue;
```

20. Try it yourself »

Node List Length

A node list object keeps itself up-to-date. If an element is deleted or added, the list is automatically updated

The length property of a node list is the number of nodes in the list

This code fragment returns the number of <title> elements in "**books.xml**":

```
x = xmlDoc.getElementsByTagName('title').length;
```

After the execution of the statement above, the value of x will be 4.

The length of the node list can be used to loop through all the elements in the list.

This code fragment uses the length property to loop through the list of <title> elements:

```
x = xmlDoc.getElementsByTagName('title');
xLen = x.length;

for (i = 0; i <xLen; i++) {
    txt += x[i].childNodes[0].nodeValue) + " ";
}
```

21. Try it yourself »

Output:

```
Everyday Italian
Harry Potter
XQuery Kick Start
Learning XML
```

DOM Attribute List (Named Node Map)

- ✓ The attributes property of an element node **returns a list of attribute nodes**
- ✓ This is called a named node map, and is similar to a node list, except for some differences in methods and properties
- ✓ A attribute list keeps itself up-to-date. If an attribute is deleted or added, the list is automatically updated
- ✓ This code fragment returns a list of attribute nodes from the first <book> element in "books.xml":

```
x = xmlDoc.getElementsByTagName('book')[0].attributes;
```

- ✓ After the execution of the code above, **x.length** is the number of attributes and **x.getNamedItem()** can be used to return an attribute node

This code fragment gets the value of the "category" attribute, and the number of attributes, of a book:

```
x = xmlDoc.getElementsByTagName("book")[0].attributes;

txt = x.getNamedItem("category").nodeValue + " " + x.length);
```

22. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Set the x variable to hold a list of all attributes of the first <book> element
3. Get the value of the "category" attribute and the length of the attribute list

XML DOM Traverse Node Tree

Traversing the Node Tree

- ✓ Often you want to loop an XML document, for example: when you want to extract the value of each element
- ✓ This is called "Traversing the node tree"
- ✓ The example below **loops through all child nodes of <book>,** and displays their names and values:

23. Try it yourself »

Example explained:

1. Load the XML string into xmlDoc
2. Get the child nodes of the root element
3. For each child node, output the node name and the node value of the text node

PCDATA - Parsed Character Data

- ✓ **XML parsers normally parse all the text in an XML document**
- ✓ When an XML element is parsed, the text between the XML tags is also parsed:

```
<message>This text is also parsed</message>
```

- ✓ XML elements can contain other elements:

```
<name>
  <first>Bill</first>
  <last>Gates</last>
</name>
```

- ✓ and the parser will break it up into sub-elements like this:

```
<name><first>Bill</first><last>Gates</last></name>
```

- ✓ Parsed Character Data (PCDATA) is a term used about text data that will be parsed by the XML parser

CDATA - (Unparsed) Character Data

The term CDATA is used about text data that **should not be parsed by the XML parser.**

Characters like "<" and "&" are illegal in XML elements.

"<" will generate an error because the parser interprets it as the start of a new element.

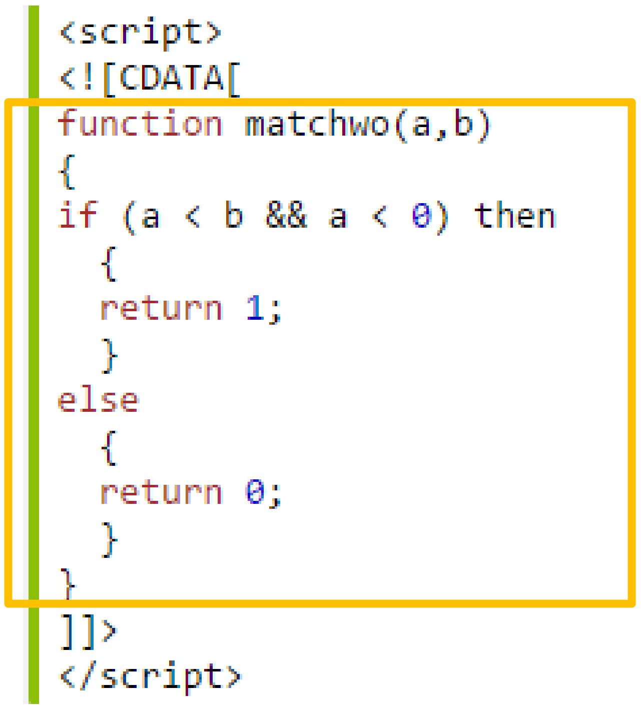
"&" will generate an error because the parser interprets it as the start of an character entity.

Some text, like JavaScript code, contains a lot of "<" or "&" characters

- ✓ To avoid errors script code can be defined as CDATA

Everything inside a CDATA section is ignored by the parser

A CDATA section starts with "<![CDATA[" and ends with "]]>":



```
<script>
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
  {
  return 1;
  }
else
  {
  return 0;
  }
}
]]>
</script>
```

Everything inside the CDATA section is ignored by the parser

XML DOM Get Node Values

nodeValue property is used to get the text value of a node.

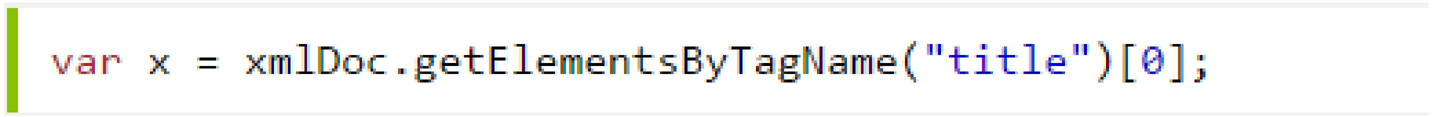
getAttribute() method returns the value of an attribute.

Get the Value of an Element

- ✓ In the DOM, everything is a node. Element nodes do not have a text value.
- ✓ The text value of an element node is stored in a child node. This node is called a text node.

getElementsByTagName Method

- ✓ getElementsByTagName() method returns a **node list of all elements**, with the specified tag name, in the same order as they appear in the source document
- ✓ Suppose "books.xml" has been loaded into xmlDoc
- ✓ This code retrieves the first <title> element:



```
var x = xmlDoc.getElementsByTagName("title")[0];
```

ChildNodes Property

- ✓ childNodes property returns a list of an element's child nodes.
- ✓ The following code retrieves the text node of the first <title> element:

```
x = xmlDoc.getElementsByTagName("title")[0];
y = x.childNodes[0];
```

- ✓ **nodeValue Property**
- ✓ nodeValue property returns the text value of a text node.
- ✓ The following code retrieves the text value of the text node of the first <title> element:

```
x = xmlDoc.getElementsByTagName("title")[0];
y = x.childNodes[0];
z = y.nodeValue;
```

24. Try it yourself »

Change the Value of an Element

- ✓ In the DOM, everything is a node. Element nodes do not have a text value
- ✓ The text value of an element node is stored in a child node. This node is called a text node

Change the Value of a Text Node

nodeValue property can be used to change **the value of a text node**

Suppose "books.xml" has been loaded into xmlDoc

This code changes the text node value of the first <title> element:

```
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue = "new content"
```

25. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Get the first child node of the <title> element
3. Change the node value to "*new content*"

Change the Value of an Attribute

- ✓ In the DOM, attributes are nodes. Unlike element nodes, attribute nodes have text values
- ✓ The way to change the value of an attribute, is to change its text value
- ✓ This can be done using the setAttribute() method or setting the nodeValue property of the attribute node

Change an Attribute Using setAttribute()

- ✓ setAttribute() method **changes the value of an attribute**.
- ✓ If the attribute does not exist, a new attribute is created.
- ✓ This code changes the category attribute of the <book> element:

```
xmlDoc.getElementsByTagName("book")[0].setAttribute("category","food");
```

26. Try it yourself »

Change an Attribute Using nodeValue

- ✓ nodeValue property is **the value of a attribute node**
- ✓ Changing the value property changes the value of the attribute

```
xmlDoc.getElementsByTagName("book")[0].getAttributeNode("category").nodeValue = "food";
```

27. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Get the "category" attribute of the first <book> element
3. Change the attribute node value to "food"

XML DOM Remove Nodes

Remove an Element Node

- ✓ removeChild() method removes a specified node
- ✓ When a node is removed, all its child nodes are also removed
- ✓ This code will remove the first <book> element from the loaded xml:

```
y = xmlDoc.getElementsByTagName("book")[0];

xmlDoc.documentElement.removeChild(y);
```

28. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded xmlDoc
2. Set the variable y to be the element node to remove
3. Remove the element node by using the removeChild() method from the parent node

Remove Myself - Remove the Current Node

- ✓ removeChild() method is the only way to remove a specified node.
- ✓ When you have navigated to the node you want to remove, it is possible to remove that node using the parentNode property and the removeChild() method:

```
x = xmlDoc.getElementsByTagName("book")[0];

x.parentNode.removeChild(x);
```

29. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Set the variable y to be the element node to remove
3. Remove the element node by using the parentNode property and the removeChild() method

Remove a Text Node

removeChild() method can also be used to remove a text node:

```
x = xmlDoc.getElementsByTagName("title")[0];
y = x.childNodes[0];
x.removeChild(y);
```

30. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Set the variable x to be the first title element node
3. Set the variable y to be the text node to remove
4. Remove the element node by using the removeChild() method from the parent node

It is not very common to use removeChild() just to remove the text from a node. The nodeValue property can be used instead. See next paragraph.

Clear a Text Node

nodeValue property can be used to change the value of a text node:

```
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue = "";
```

31. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Get the first title element's first child node.
3. Use the nodeValue property to clear the text from the text node

Loop through and change the text node of all <title> elements:

Remove an Attribute Node by Name

- ✓ removeAttribute() method removes an attribute node by its name.
- ✓ Example: removeAttribute('category')
- ✓ This code removes the "category" attribute in the first <book> element:

```
x = xmlDoc.getElementsByTagName("book");
x[0].removeAttribute("category");
```

32. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Use getElementsByTagName() to get book nodes
3. Remove the "category" attribute form the first book element node
4. Loop through and remove the "category" attribute of all <book> elements:

Remove Attribute Nodes by Object

- ✓ removeAttributeNode() method removes an attribute node, using the node object as parameter
- ✓ Example: removeAttributeNode(x)
- ✓ This code removes all the attributes of all <book> elements:

```
x = xmlDoc.getElementsByTagName("book");

for (i = 0; i < x.length; i++) {
    while (x[i].attributes.length > 0) {
        attnode = x[i].attributes[0];
        old_att = x[i].removeAttributeNode(attnode);
    }
}
```

33. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Use getElementsByTagName() to get all book nodes
3. For each book element check if there are any attributes
4. While there are attributes in a book element, remove the attribute

XML DOM Replace Nodes

Replace an Element Node

- ✓ replaceChild() method is used to replace a node
- ✓ The following code fragment replaces the first <book> element:

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.documentElement;

//create a book element, title element and a text node
newNode=xmlDoc.createElement("book");
newTitle=xmlDoc.createElement("title");
newText=xmlDoc.createTextNode("A Notebook");

//add the text node to the title node,
newTitle.appendChild(newText);
//add the title node to the book node
newNode.appendChild(newTitle);

y=xmlDoc.getElementsByTagName("book")[0]
//replace the first book node with the new node
x.replaceChild(newNode,y);
```

34. Try it yourself »

Replace Data In a Text Node

- ✓ replaceData() method is used to replace data in a text node.
- ✓ replaceData() method has three parameters:
- ✓ offset - Where to begin replacing characters. Offset value starts at zero
- ✓ length - How many characters to replace
- ✓ string - The string to insert

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];

x.replaceData(0,8,"Easy");
```

35. Try it yourself »

Example explained:

1. Load "books.xml" into xmlDoc
2. Get the text node of the first <title> element node
3. Use the replaceData method to replace the eight first characters from the text node with "Easy"

Use the nodeValue Property Instead

- ✓ It is easier to replace the data in a text node using the nodeValue property
- ✓ The following code fragment will replace the text node value in the first <title> element with "Easy Italian":

```
xmlDoc=loadXMLDoc("books.xml");

x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];

x.nodeValue="Easy Italian";
```

36. Try it yourself »

Example explained:

1. Load "books.xml" into xmlDoc
2. Get the text node of the first <title> element node
3. Use the nodeValue property to change the text of the text node

XML DOM Create Nodes

Create a New Element Node

createElement() method creates a new element node:

```
newElement = xmlDoc.createElement("edition");

xmlDoc.getElementsByTagName("book")[0].appendChild(newElement);
```

37. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Create a new element node <edition>
3. Append the element node to the first <book> element

Create a New Attribute Node

createAttribute() is used to create a new attribute node:

```
newAtt = xmlDoc.createAttribute("edition");
newAtt.nodeValue = "first";

xmlDoc.getElementsByTagName("title")[0].setAttributeNode(newAtt);
```

38. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Create a new attribute node "edition“
3. Set the value of the attribute node to "first“
4. Add the new attribute node to the first <title> element

Create an Attribute Using setAttribute()

Since the setAttribute() method creates a new attribute if the attribute does not exist, it can be used to create a new attribute.

```
xmlDoc.getElementsByTagName('book')[0].setAttribute("edition","first");
```

39. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Set the attribute "edition" value to "first" for the first <book> element

Create a Text Node

createTextNode() method creates a new text node:

```
newEle = xmlDoc.createElement("edition");
newText = xmlDoc.createTextNode("first");
newEle.appendChild(newText);

xmlDoc.getElementsByTagName("book")[0].appendChild(newEle);
```

40. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Create a new element node <edition>
3. Create a new text node with the text "first"
4. Append the new text node to the element node
5. Append the new element node to the first <book> element

Create a CDATA Section Node

createCDATASection() method creates a new CDATA section node.

```
newCDATA = xmlDoc.createCDATASection("Special Offer & Book Sale");

xmlDoc.getElementsByTagName("book")[0].appendChild(newCDATA);
```

41. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Create a new CDATA section node
3. Append the new CDATA node to the first <book> element

Create a Comment Node

createComment() method creates a new comment node.

```
newComment = xmlDoc.createComment("Revised March 2015");

xmlDoc.getElementsByTagName("book")[0].appendChild(newComment);
```

42. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc using
2. Create a new comment node
3. Append the new comment node to the first <book> element

XML DOM Add Nodes

Add a Node - appendChild()

- ✓ appendChild() method adds a child node to an existing node
- ✓ The new node is added (appended) after any existing child nodes

Note: Use insertBefore() if the position of the node is important.

- ✓ This code fragment creates an element (<edition>), and adds it after the last child of the first <book> element:

```
newEle = xmlDoc.createElement("edition");

xmlDoc.getElementsByTagName("book")[0].appendChild(newEle);
```

43. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Create a new node <edition>
3. Append the node to the first <book> element

Add a Node - appendChild() (계속)

This code fragment does the same as above, but the new element is added with a value:

```
newEle = xmlDoc.createElement("edition");
newText=xmlDoc.createTextNode("first");
newEle.appendChild(newText);

xmlDoc.getElementsByTagName("book")[0].appendChild(newEle);
```

44. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Create a new node <edition>
3. Create a new text node "first"
4. Append the text node to the <edition> node
5. Append the <addition> node to the <book> element

Insert a Node - insertBefore()

- ✓ The **insertBefore()** method inserts a node before a specified child node
- ✓ This method is useful when the position of the added node is important:

```
newNode = xmlDoc.createElement("book");

x = xmlDoc.documentElement;
y = xmlDoc.getElementsByTagName("book")[3];

x.insertBefore(newNode,y);
```

45. Try it yourself »

Example explained:

1. Suppose "books.xml" is loaded into xmlDoc
2. Create a new element node <book>
3. Insert the new node in front of the last <book> element node

If the second parameter of insertBefore() is null, the new node will be added after the last existing child node.

x.insertBefore(newNode, null) and **x.appendChild(newNode)** will both append a new child node to x.

Add a New Attribute

setAttribute() method sets the value of an attribute

```
xmlDoc.getElementsByTagName('book')[0].setAttribute("edition","first");
```

46. Try it yourself »

Example explained:

1. Suppose "books.xml" has been loaded into xmlDoc
2. Set the value of the attribute "edition" to "first" for the first <book> element

Add Text to a Text Node - insertData()

- ✓ insertData() method inserts data into an existing text node
- ✓ insertData() method has two parameters:
- ✓ offset - Where to begin inserting characters (starts at zero)
- ✓ string - The string to insert
- ✓ The following code fragment will add "Easy" to the text node of the first <title> element of the loaded XML:

```
xmlDoc.getElementsByTagName("title")[0].childNodes[0].insertData(0,"Easy ");
```

47. Try it yourself »

XML DOM Clone Nodes

Copy a Node

- ✓ cloneNode() method creates a copy of a specified node
- ✓ cloneNode() method has a parameter (true or false). This parameter indicates if the cloned node should include all attributes and child nodes of the original node
- ✓ The following code fragment copies the first <book> node and appends it to the root node of the document:

```
oldNode = xmlDoc.getElementsByTagName('book')[0];
newNode = oldNode.cloneNode(true);
xmlDoc.documentElement.appendChild(newNode);
```

48. Try it yourself »

Result:

```
Everyday Italian
Harry Potter
XQuery Kick Start
Learning XML
Everyday Italian
```

실습과제 08-1

본문에 나오는 모든 [Try it yourself >>]를 구현하여 자신의 웹서버에 업로드하고 URL 부분이 잘 보이도록 캡처하여 제출하시오(결과화면에 반드시 자신의 학번 이름이 명시되어야 함).

NOTE

예제에서 books.xml이 사용되는 경우 모두 지난 시간에 작성한 bookstore.xml의 내용으로 변경하여 적용한다.

Q & A