Advanced PHP

Prof. Hyuk-Yoon Kwon
https://sites.google.com/view/seoultech-bigdata

Most parts are based on slides used in
(http://cgi.csc.liv.ac.uk/~martin/teaching/comp519/)

What is XML?

- XML is a markup language for encoding documents in a format that is both human-readable and machine-readable
- Is designed to transport and store data
- Emphasizes simplicity, generality, and usability over the Internet
- Has strong support via Unicode for the languages of the world

XML Example

```xml
<?xml version="1.0"?>
<books>
  <book>
    <title>Maktub</title>
    <author>Paulo Coelho</author>
  </book>
  <book>
    <title>Never Crashed!</title>
    <author>Microsoft</author>
  </book>
</books>
```

XML versus HTML

- XML and HTML are both markup languages
- HTML is for displaying data, while XML is for describing data
- XML syntax differences
  - New tags may be defined at will
  - Tags may be nested to arbitrary depth
- XHTML is a version of HTML in XML

XHTML

- XHTML is HTML redesigned as XML
- Document Structure
  - <html>, <head>, <title>, and <body> are mandatory
- XHTML Elements
  - XHTML elements must be properly nested
  - XHTML elements must always be closed
  - XHTML elements must be in lowercase
  - XHTML documents must have one root element
- XHTML Attributes
  - Attribute names must be in lower case
  - Attribute values must be quoted
  - Attribute minimization is forbidden

XML Markup Languages

- Lots of new markup languages have been created with XML, including:
  - XHTML
  - RSS for news feeds
  - RDF for describing resources
  - SVG for scalable vector graphics
  - SMIL for describing multimedia for the web
  - MathML for describing mathematical notation

  …

XML Pros and Cons

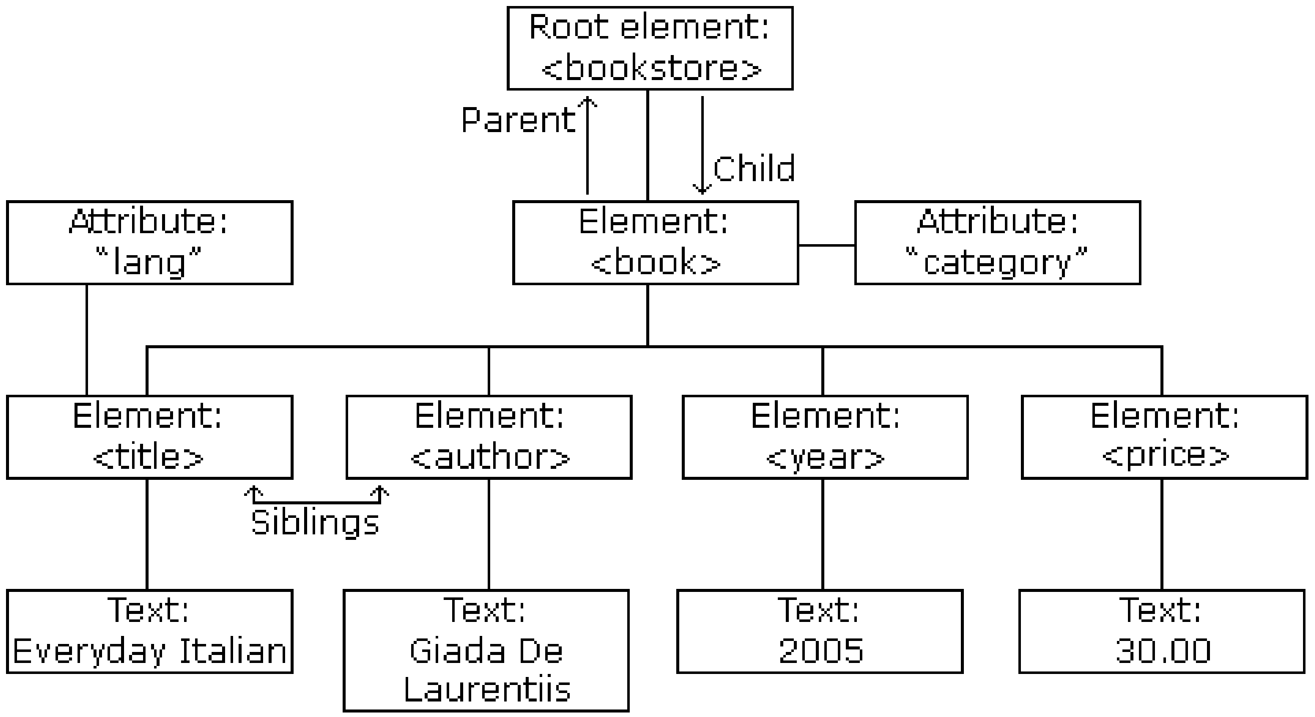
- Pros:
  - software- and hardware-independent
  - simplifying:
    - sharing data between applications
    - transporting data between different platforms
- Cons:
  - verbosity
  - rather complex parsing and mapping to type systems

XML Tree

- Each XML document forms a tree structure that starts at the root and branches to the leaves

```
<bookstore>
   <book category="COOKING">
     <title lang="en">Everyday Italian</title>
     <author>Giada De Laurentiis</author>
     <year>2005</year>
     <price>30.00</price>
    </book>
</bookstore>
```

XML Tree Example



XML Tags

- XML tags are similar to HTML tags but
  - They are case-sensitive
  - All tags must be closed
- Like HTML tags they must be properly nested
- All XML documents must have a single root element that contains all other elements
  - This root element can have any name

XML Attributes

- XML elements can have attributes
- Attribute values must be quoted with either single or double quotes

```
<book title="Let's party!">
```

```
<film name='The "Lost"'/>
```

- Attributes can be represented as children elements
  - Elements are more flexible alternatives

```
<book>
  <title>It's me</title>
  <author>Me who</author>
</book>
```

PHP XML Parsers

- What is an XML Parser?
  - To read and update, create and manipulate an XML document, you will need an XML parser.
- In PHP there are two major types of XML parsers:
  - Tree-Based Parsers
  - Event-Based Parsers
- Tree-Based Parsers
  - Holds the entire document in Memory and transforms the XML document into a Tree structure.
  - Better option for smaller XML documents, but not for large XML document as it causes major performance issues.
- Event-Based Parsers
  - Read in one node at a time and allow you to interact with in real time.
  - Well suited for large XML documents. It parses faster and consumes less memory.

SimpleXML Parser

- A tree-based parser
  - An easy way of getting an element's name, attributes and textual content if you know the XML document's structure or layout.

```
<?php
$myXMLData =
"<?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>";

$xml=simplexml_load_string($myXMLData) or die("Error: Cannot create object");
print_r($xml);
?>
```

Read From File

- The PHP simplexml_load_string() function is used to read XML data from a file

```
<?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>
```

```
<?php
$xml=simplexml_load_file("note.xml") or die("Error:
Cannot create object");
print_r($xml);
?>
```

Get Node Values

```php
<?php
$xml=simplexml_load_file("note.xml") or die("Error: Cannot create object");
echo $xml->to . "<br>";
echo $xml->from . "<br>";
echo $xml->heading . "<br>";
echo $xml->body;
?>
```

books.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en-us">XQuery Kick Start</title>
    <author>James McGovern</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
  <book category="WEB">
    <title lang="en-us">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Get Node Values of Specific Elements

- The following example gets the node value of the <title> element in the first and second <book> elements in the "books.xml" file

```
<?php
$xml=simplexml_load_file("books.xml") or die("Error: Cannot create object");
echo $xml->book[0]->title . "<br>";
echo $xml->book[1]->title;
?>
```

Get Node Values - Loop

- The following example loops through all the <book> elements in the "books.xml" file

```php
<?php
$xml=simplexml_load_file("books.xml") or die("Error: Cannot create object");
foreach($xml->children() as $books) {
    echo $books->title . ", ";
    echo $books->author . ", ";
    echo $books->year . ", ";
    echo $books->price . "<br>";
}
?>
```

Get Attribute Values

- The following example gets the attribute value

```
<?php
$xml=simplexml_load_file("books.xml") or die("Error:
Cannot create object");
echo $xml->book[0]['category'] . "<br>";
echo $xml->book[1]->title['lang'];
?>
```

Get Attribute Values - Loop

- The following example gets the attribute values of the <title> elements in the "books.xml" file

```
<?php
$xml=simplexml_load_file("books.xml") or die("Error:
Cannot create object");
foreach($xml->children() as $books) {
    echo $books->title['lang'];
    echo "<br>";
}
?>
```

PHP XML Expat Parser

- The built-in XML Expat Parser makes it possible to process XML documents in PHP
- The Expat parser is an event-based parser
- Look at the following XML fraction

```
<from>Jani</from>
```

  - An event-based parser reports the XML above as a series of three events:
    - Start element: from
    - Start text, value: Jani
    - Close element: from

How to use XML Expat Parser

```php
<?php
// Initialize the XML parser
$parser=xml_parser_create();
// Function to use at the start of an element
function start($parser,$element_name,$element_attrs) {
  switch($element_name) {
    case "NOTE":
    echo "-- Note --<br>";
    break;
    case "TO":
    echo "To: ";
    break;
    case "FROM":
    echo "From: ";
    break;
    case "HEADING":
    echo "Heading: ";
    break;
    case "BODY":
    echo "Message: ";
  }
}
// Function to use at the end of an element
function stop($parser,$element_name) {
  echo "<br>";
}
```

```
// Function to use when finding character data
function char($parser,$data) {
  echo $data;
}

// Specify element handler
xml_set_element_handler($parser,"start","stop");

// Specify data handler
xml_set_character_data_handler($parser,"char");

// Open XML file
$fp=fopen("note.xml","r");

// Read data
while ($data=fread($fp,4096)) {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
}

// Free the XML parser
xml_parser_free($parser);
?>
```

PHP XML DOM Parser

- The built-in DOM parser makes it possible to process XML documents in PHP
- The DOM parser is a tree-based parser
- Look at the following XML document fraction

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

- The DOM sees the XML above as a tree structure:
  - Level 1: XML Document
  - Level 2: Root element: <from>
  - Level 3: Text element: "Jani"

Load and Output XML

- We want to initialize the XML parser, load the xml, and output it

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
?>
```

- If you select "View source" in the browser window, you will see the following HTML

```
<?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>
```

- The example above creates a DOM Document-Object and loads the XML from "note.xml" into it.
- Then the saveXML() function puts the internal XML document into a string, so we can output it

Looping through XML

- We want to initialize the XML parser, load the XML, and loop through all elements of the <note> element

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item) {
  print $item->nodeName . " = " . $item->nodeValue . "<br>";
}
?>
```

What is JSON?

- JSON stands for JavaScript Object Notation
- It is a lightweight text-data interchange format, commonly used as an alternative to XML
- JSON is smaller, faster and easier to parse
- Although JSON uses JavaScript syntax, it is still language and platform independent.

JSON Examples

```
{
  "message": {
    "from": "Hassan",
    "to": "Hossein",
    "body": "Please give me a call!"
  }
}
```

```
{
  "books": [
    {"title": "Maktub", "author": "Paulo Coelho"},
    {"title": "Crashed!", "author": "Microsoft"}
  ]
}
```

JSON in PHP

- PHP has some built-in functions to handle JSON.
- Objects in PHP can be converted into JSON by using the PHP function json_encode()

```
<?php
$myObj = new Class();
$myObj->name = "John";
$myObj->age = 30;
$myObj->city = "New York";

$myJSON = json_encode($myObj);

echo $myJSON;
?>
```

```php
<?php
$myArr = array("John", "Mary", "Peter", "Sally");

$myJSON = json_encode($myArr);

echo $myJSON;
?>
```

AJAX (Asynchronous JavaScript and XML)

History

- In the 1990s, most web sites were based on complete HTML pages
- Each user action required that a new page (or the same page) be loaded from the server
- This process was inefficient
  - it was slow and required user input
- For web-based email this meant that users had to manually reload their inbox to check and see if they had new mail

History (Cont'd)

- In 1999, Microsoft created the XMLHTTP ActiveX control in IE5, allowing the browser to communicate with the server without requiring a page reload
- The idea was later adopted by Mozilla, Safari, Opera and other browsers as the XMLHttpRequest JavaScript object
- The term AJAX was coined in 2005 by Jesse James Garrett in an article entitled "Ajax: A New Approach to Web Applications"

What is AJAX?

- AJAX stands for Asynchronous JavaScript And XML
- It is a way of using existing standards (such as JavaScript and XML) to make more interactive web applications
- AJAX was popularized in 2005 by Google, specially, with Google suggest

Typical AJAX Transaction

- A typical AJAX transaction looks like this:
  1. An event is triggered by the user/browser (such as mouse events, keyboard events, or time events)
  2. Event handler sends an HTTP request to server
  3. Server replies to the request
  4. The reply handler updates web page using server's reply
- Between steps 2 and 3 the web page is still usable (event is asynchronous)
- At no point during the transaction does the browser open a new web page

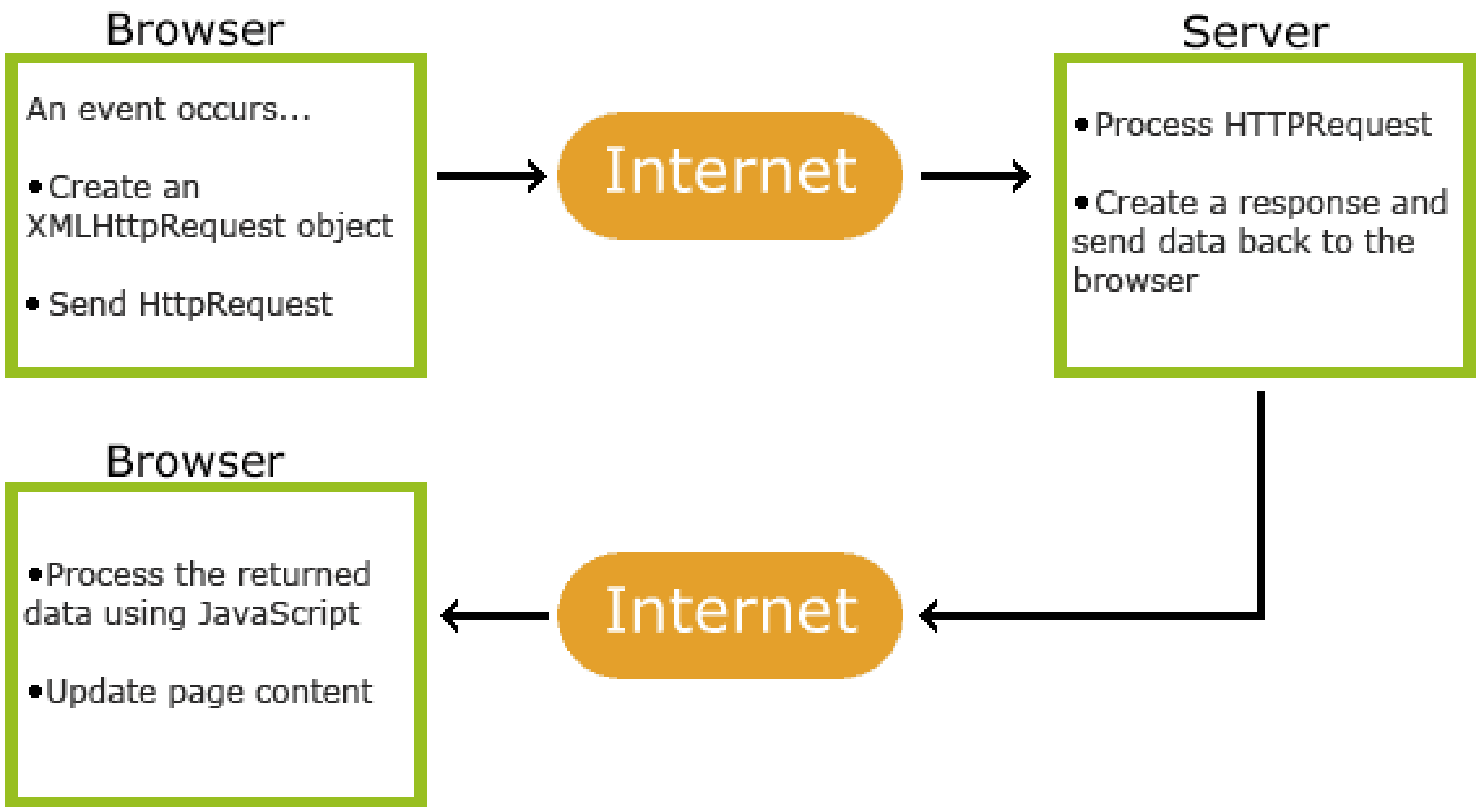
AJAX Pros

- AJAX allows web applications to exchange data with the server without the need for a page reload
- Communication with the server takes place asynchronously, and transparently to the user
- AJAX allows us to avoids clunky GET/POST send/receive interfaces
- Many web applications can only be realized this way (e.g., Google Maps, Google suggest, etc.)

AJAX Cons

- Different browsers implement the AJAX differently
- AJAX can be server intensive
  - e.g., Google Suggest generates a search for every keystroke entered
- Dynamic web page updates are difficult to bookmark, and to be indexed by search engines

How AJAX Works


Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Browser
• Process the returned data using JavaScript
• Update page content
Internet

Sample AJAX

- Here is a simple AJAX transaction:

```
var httpRequest = new XMLHttpRequest();

httpRequest.onreadystatechange = function() {
 if (httpRequest.readyState == 4) {
   alert('Request complete!');
};

httpRequest.open('GET', 'something.py', true);
httpRequest.send(null);
```

Open and Send Methods

- The open method takes three arguments:

```
httpRequest.open(method, url, async)
```

  - **method: 'GET' or 'POST' (or any HTTP request)**
  - **url: the (relative) URL to retrieve**
  - **async: determines whether to send the request asynchronously (true) or synchronously (false)**

- **The send method takes one argument:**

```
httpRequest.send(content)
```

  - content: the content to send (useful when method='POST')

Ready State

- The readyState property Holds 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

Handling Response

- When an XMLHTTPRequest is complete (readyState == 4) the status property contains the response code of the HTTP response

```
if (httpRequest.status === 200) {
    // perfect!
} else {
    // there was a problem with the request,
    // e.g., the response may be 404 (Not Found)
    // or 500 (Internal Server Error)
}
```

Accessing Response

- After checking the HTTP response code, we can access the data through:
  - responseText – the server response as a string
  - responseXML – the response as an XMLDocument object that you can traverse using the JavaScript DOM functions

Example

- This is a simple code for parsing an xml file

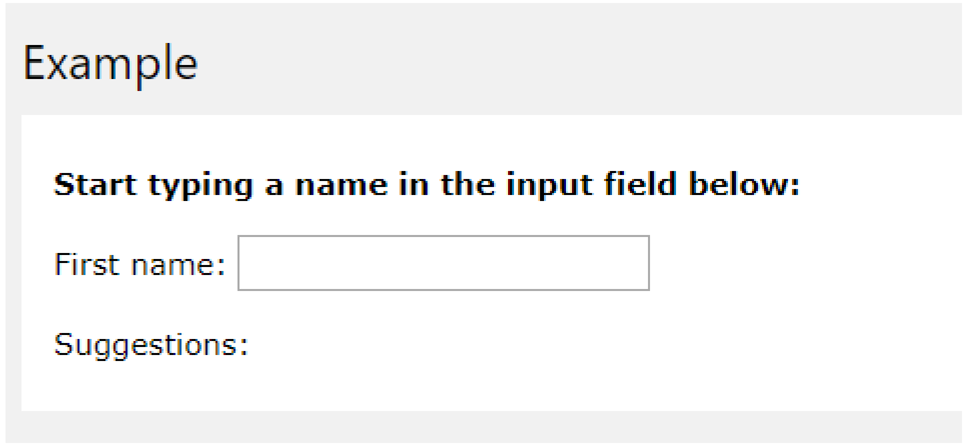
```
xmlDoc = httpRequest.responseXML;
books = xmlDoc.getElementsByTagName('title');

res = '';
for (var i = 0; i < books.length; i++)
   res += books[i].firstChild.nodeValue + '<br>';

document.getElementById('books').innerHTML = res;
```

AJAX PHP Example

- The following example will demonstrate how a web page can communicate with a web server while a user types characters in an input field


Example

Start typing a name in the input field below:

First name:

Suggestions:

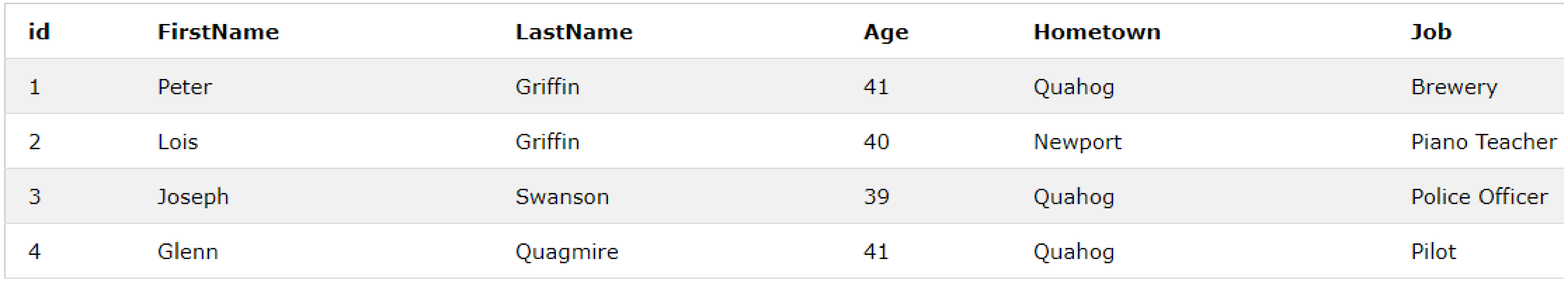
AJAX codes for dynamic suggestions

```
<html>
<head>
<script>
function showHint(str) {
    if (str.length == 0) {
        document.getElementById("txtHint").innerHTML = "";
        return;
    } else {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtHint").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET", "gethint.php?q=" + str, true);
        xmlhttp.send();
    }
}
</script>
</head>
```

```
<body>
<p><b>Start typing a name in the input field
below:</b></p>
<form>
First
name: <input type="text" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

Practice - AJAX and MySQL

Databases



id	FirstName	LastName	Age	Hometown	Job
1	Peter	Griffin	41	Quahog	Brewery
2	Lois	Griffin	40	Newport	Piano Teacher
3	Joseph	Swanson	39	Quahog	Police Officer
4	Glenn	Quagmire	41	Quahog	Pilot

- **1st file for handling AJAX**

```
<html>
<head>
<script>
function showUser(str) {
    if (str == "") {
        document.getElementById("txtHint").innerHTML = "";
        return;
    } else {
        if (window.XMLHttpRequest) {
            // code for IE7+, Firefox, Chrome, Opera, Safari
            xmlhttp = new XMLHttpRequest();
        } else {
            // code for IE6, IE5
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtHint").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET","getuser.php?q="+str,true);
        xmlhttp.send();
    }
}
</script>
</head>
```

```
<body>
<form>
<select name="users" onchange="showUser(this.value)">
  <option value="">Select a person:</option>
  <option value="1">Peter Griffin</option>
  <option value="2">Lois Griffin</option>
  <option value="3">Joseph Swanson</option>
  <option value="4">Glenn Quagmire</option>
  </select>
</form>
<br>
<div id="txtHint"><b>Person info will be listed here...</b></div>
</body>
</html>
```

You need to modify some codes in this page to get correct results !

2nd file for accessing the database

```
<?php
$q = $_GET['q'];

$conn = new mysqli(“localhost”, “root”, “”, 'ajax_demo');
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql="SELECT * FROM user WHERE Firstname = '".$q."'";
$result = $conn->query($sql);

echo "<table>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";
while($row = $result->fetch_assoc()) {
    echo "<tr>";
    echo "<td>" . $row['FirstName'] . "</td>";
    echo "<td>" . $row['LastName'] . "</td>";
    echo "<td>" . $row['Age'] . "</td>";
    echo "<td>" . $row['Hometown'] . "</td>";
    echo "<td>" . $row['Job'] . "</td>";
    echo "</tr>";
}
echo "</table>";
$conn->close();
?>
```