Advance Web Technologies & Programming (CSC350)

What is JSON

- JSON stands for JavaScript Object Notation.
- JSON is a lightweight data-interchange format.
- JSON is plain text written in JavaScript object notation.
- JSON is used to send data between computers.
- JSON is language independent .
- JSON is a data interchange format.
- Interactive Web 2.0 applications, no more use page replacement. Data transfer without refreshing a page.
- The most important aspects of data transfer are simplicity, extensibility, interoperability, openness and human readability
- Key idea in AJAX – Asynchronous Java Script and XML.

Why Use JSON?

- The JSON format is syntactically similar to the code for creating JavaScript objects.
- Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.
- Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.
- JavaScript has a built in function for converting JSON strings into JavaScript objects:

  JSON.parse()

- JavaScript also has a built in function for converting an object into a JSON string:

  JSON.stringify()

Example of XML-formatted data

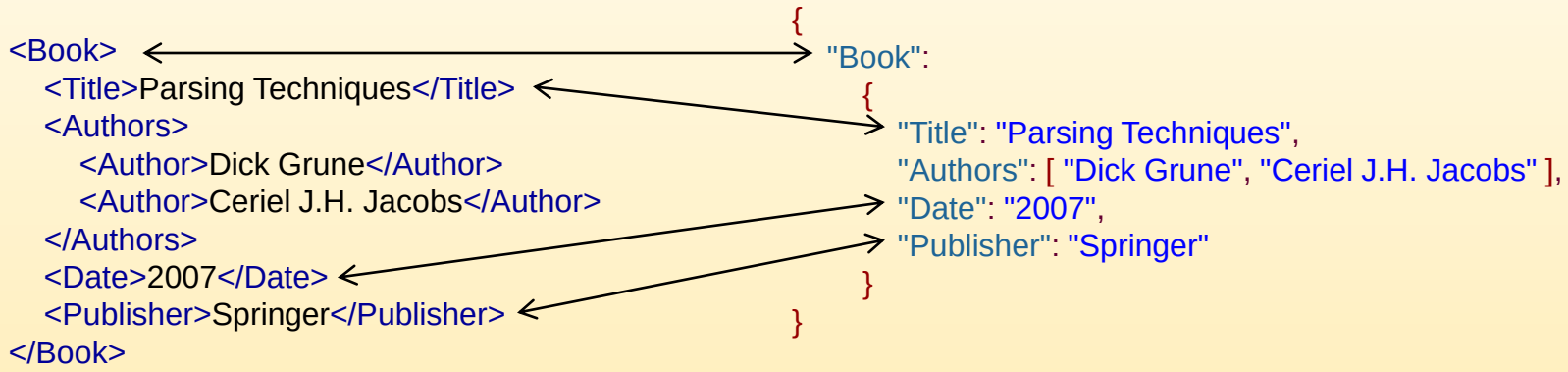
• The below XML document contains data about a book: its title, authors, date of publication, and publisher.

```
<Book>
   <Title>Parsing Techniques</Title>
   <Authors>
      <Author>Dick Grune</Author>
      <Author>Ceriel J.H. Jacobs</Author>
   </Authors>
   <Date>2007</Date>
   <Publisher>Springer</Publisher>
</Book>
```

Same XML data, in JSON format

```
{
   "Book":
      {
         "Title": "Parsing Techniques",
         "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ],
         "Date": "2007",
         "Publisher": "Springer"
      }
}
```

XML and JSON, side-by-side



How does it work?

- JSON is a subset of Java Script. JSON can be parsed by a Java Script parser.
- It can represent either complex or simple data as it has data types
- They are Strings, Number, Boolean, Objects and Arrays
- E.g. of Object:
- { "name": "Jack (\"Bee\") Nimble", "format": { "type": "rect", "width": 120, "interlace": false}}

Example

```
{"firstName": "John",
 "lastName" : "Smith",
 "age"        : 25,
 "address"   :
   {"streetAdr” : "21 2nd Street",
    "city"       : "New York",
    "state"      : "NY",
    ”zip"        : "10021"},
 "phoneNumber":
   [{"type"  : "home",
    "number": "212 555-1234"},
   {"type"  : "fax",
    "number” : "646 555-4567"}]
 }
```

- This is a JSON object with five key-value pairs
- Objects are wrapped by curly braces
- There are no object IDs
- Keys are strings
- Values are numbers, strings, objects or arrays
- Arrays are wrapped by square brackets

Using JSON you can define arbitrarily complex structures

```
{
   "Book":
      {
         "Title": "Parsing Techniques",
         "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ]
      }
}
```

```
{
   "Book":
      {
         "Title": "Parsing Techniques",
         "Authors": [
            {"name":"Dick Grune", "university": "Vrije Universiteit"},
            {"name":"Ceriel J.H. Jacobs", "university": "Vrije Universiteit"}
         ]
      }
}
```

Extend, and infinitum

```
{
   "Book":
      {
         "Title": "Parsing Techniques",
         "Authors": [
            {"name": {"first":"Dick", "last":"Grune"},
             "university": "Vrije Universiteit"},
            {"name": {"first":"Ceriel", "last":"Jacobs"},
             "university": "Vrije Universiteit"}
         ]
      }
}
```

Exchanging Data

- The JSON format is almost identical to JavaScript objects.
- In JSON, *keys* must be strings, written with double quotes:
- In JSON, *string values* must be written with double quotes:
- The file type for JSON files is ".json"
-

Exchanging Data in JS

- When exchanging data between a browser and a server, the data can only be text.
- JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects.
- This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

Parse and Stringify

- JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects: JSON.parse()
- So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.
- When sending data to a web server, the data has to be a string.
- Convert a JavaScript object into a string with JSON.stringify().
- In JSON, date objects and functions are not allowed. The JSON.stringify() function will convert any dates or function into strings.

Sending Data in JS

- If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:
- Example

```
var myObj = { "name":"John", "age":31, "city":"New York" };

var myJSON = JSON.stringify(myObj);

window.location = "demo_json.php?x=" + myJSON;
```

Receiving Data in JS

- If you receive data in JSON format, you can convert it into a JavaScript object:
- Example

```
var myJSON = '{ "name":"John", "age":31, "city":"New York" }';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML =
myObj.name;
```

Storing Data in local storage

- When storing data, the data has to be a certain format, and regardless of where you choose to store it, *text* is always one of the legal formats.
- JSON makes it possible to store JavaScript objects as text.
- Example: Storing data in local storage

- //Storing data:
  myObj = { "name":"John", "age":31, "city":"New York" };
  myJSON = JSON.stringify(myObj);
  localStorage.setItem("testJSON", myJSON);

- //Retrieving data:
  text = localStorage.getItem("testJSON");
  obj = JSON.parse(text);
  document.getElementById("demo").innerHTML = obj.name;

JSON and 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->name = "John";
$myObj->age = 30;
$myObj->city = "New York";
$myJSON = json_encode($myObj);
echo $myJSON;
?>
```

Output:

```
{"name":"John","age":30,"city":"New York"}
```

Receiving JSON in PHP

```
ini_set("allow_url_fopen", 1); //if not open
$json = file_get_contents('php://input');
$obj = json_decode($json);
echo $obj->variablekey;
OR
Echo $obj['variablekey']; OR
foreach ($obj as $key=>$value) {
    echo $key . ' = ' . $value;}
```

Ajax

Transaction Steps in JS

- Create an XMLHTTPRequestobject
- Set up the response handler
- Open the request
- Send the request

Example

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);
```

The XMLHttpRequest Object

- var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
  // Typical action to be performed when the document is ready:
  document.getElementById("demo").innerHTML = xhttp.responseText;
  }
  };
  xhttp.open("GET", "*filename*", true);
  xhttp.send();

XML

eXtensible Markup Language

XML, Xpath and XSLT

- XML is an acronym for eXtensible Markup Language.
  - Its purpose is to describe structured data
- XPath is a language for navigating through an XML document.
  - It's used to select specific pieces of information from the document
- XSLT is a language for transforming XML into something else.
  - Often used to generate HTML or another XML document.

XML Basics

XML Basics

- Basic Text

```
<?xml version = “1.0”?>
<!-- This is Student Data Xml File Student.xml -->
<student>
   <Name>
      <FirstName> A </FirstName>
      <LastName> S </LastName>
   </Name>
   <Department> Computer Science </Department>
   <Age> 18 </Age>
</student>
```

- Processing XML Document (parsers, processor)
- Validating XML Document
  - Document Type Definition, DTD
  - W3C XML Schema

❑ XML Basics(Tags and Elements)

- ❑ (Freely definable) **tags**: **`student`**, **`Name`**, **`FirstName`**,**`Age`**, **`....`**
  - with start tag: **`< student >`** etc.
  - and end tag: **`</ student >`** etc.
- ❑ **Elements**: **`< student > ... </ student >`**
- ❑ Elements have a **name** (**`student`**) and a **content** (**`...`**)
- ❑ Elements may be nested.
- ❑ Elements may be empty: **`<this_is_empty/>`**
- ❑ Element content is typically **"Parsed character data"** (PCDATA), i.e., strings with special characters, and/or nested elements (*mixed content* if both).
- ❑ Each XML document has exactly one root element and forms a tree.

Element Body Rules

- Element bodies may contain text or markup or both.
  - By text, we mean character strings with no markup.
  - Markup is text with embedded markup characters:
    - & < > ‘ and “
  - Elements may also contain CDATA sections, designed to support text including large sections of markup but not interpreted as markup:
    - <! [CDATA[…]]>
    - These cannot be used to carry binary data.

CDATA

- By default, all text inside an XML document is parsed
- You can force text to be treated as unparsed *character data* by enclosing it in <![CDATA[…]]>
- Any characters, even & and <, can occur inside a CDATA
- Whitespace inside a CDATA is (usually) preserved
- The only real restriction is that the character sequence]]> cannot occur inside a CDATA
- CDATA is useful when your text has a lot of illegal characters (for example, if your XML document contains some HTML text)

Illegal Characters

- Certain characters are reserved for markup and are illegal in names and payload text:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	‘	apostrophe
&quot;	“	quotation mark

- We represent them in XML with the escape sequence shown on the left, e.g.: < if we want a less than character in payload text.

XML Example(Elements)

```
<CATALOG>
   <CD>
      <TITLE>Nayyara Sings Faiz</TITLE>
      <ARTIST>Nayyara Noor</ARTIST>
      <COUNTRY>Pakistan</COUNTRY>
      <COMPANY>EMI</COMPANY>
      <PRICE>250.00</PRICE>
      <YEAR>1976</YEAR>
   </CD>
   <CD>
      <TITLE>A Tribute To Faiz Ahmed Faiz</TITLE>
      <ARTIST>Iqbal Bano</ARTIST>
      <COUNTRY>Pakistan</COUNTRY>
      <COMPANY>EMI</COMPANY>
      <PRICE>300.00</PRICE>
      <YEAR>1990</YEAR>
   </CD>
</CATALOG>
```

XML Attributes

Elements may have **attributes** (in the start tag) that have a **name** and a **value**, e.g. **`<section number="1">`**.

```
<person gender="female">
  <firstname>Natasha</firstname>
  <lastname>Ahmed</lastname>
</person>
```

XML Attributes

- What is the difference between elements and attributes?
  - Only one attribute with a given name per element (but an arbitrary number of subelements)
  - Attributes have no structure, simply strings (while elements can have subelements)
- As a *rule of thumb*:
  - Content into elements
  - Metadata into attributes

Example:

```
<person born="1912-06-23" died="1954-06-07">
Alan Turing</person> proved that…
```

Example XML document

```
<?xml version="1.0"?>
<weatherReport>
   <date>7/14/97</date>
   <city>North Place</city>
   <state>NX</state>
   <country>USA</country>
   High Temp: <high scale="F">103</high>
   Low Temp: <low scale="F">70</low>
   Morning: <morning>Partly cloudy, Hazy</morning>
   Afternoon: <afternoon>Sunny &amp;amp; hot</afternoon>
   Evening: <evening>Clear and Cooler</evening>
</weatherReport>
```

From: **XML: A Primer**, by Simon St. Laurent