# Advanced Web Technologies


**Iosif Polenakis**

PhD Candidate,
Department of Computer Science and Engineering,
University of Ioannina.

email: ipolenak@cs.uoi.gr, tel.: 2651008831

# XML Basics

- **XML**: E**X**tensible **M**arkup **L**anguage

# XML Basics

- **XML**: EXtensible **M**arkup **L**anguage
  - ✓ Main Purpose → Data Definition

# XML Basics

- **XML**: EXtensible Markup Language
  - ✓ Main Purpose → Data Definition
  - ✓ User Defined Tags → Effectively Parsing Data

# XML Basics

- **XML**: EXtensible Markup Language
  - ✓ Main Purpose → Data Definition
  - ✓ User Defined Tags → Effectively Parsing Data
  - ✓ Application:
    - XML aims on Transferring, Storing and Describing Data
    - Data Interconnection between web entities (i.e., services)
    - Unique Type Interpetation

# XML Basics

- **XML**: E**X**tensible **M**arkup **L**anguage
  - ✓ Example

```
<?xml version="1.0"?
   encoding="UTF-8"]
   standalone="yes"] >
<package packagerversion="1.4.0a4">
      <name> Services_Trackback</name>
      <channel>pear.php.net</channel>
</package>
```

# XML Basics

- **XML**: Shapes of XML

Key Insights :

- ✓ **Tag**

  A tag is a markup construct that begins with < and ends with /> as follows:

  - start-tag, such as <section>;
  - end-tag, such as </section>;
  - empty-element tag, such as <line-break />

# XML Basics

- **XML**: Shapes of XML

  Key Insights :

  - ✓ **Element**

    An element is a logical document component that either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag. The characters between the start-tag and end-tag, if any, are the element's content, and may contain markup, including other elements, which are called child elements.

    <greeting>*Hello, world!*</greeting>. *Another is* <line-break />.

# XML Basics

- **XML**: Shapes of XML

Key Insights :

- ✓ **Attribute**

  An attribute is a markup construct consisting of a name–value pair that exists within a start-tag or empty-element tag. The names of the attributes and their values are defined inside the start-tags.

  An XML attribute can only have a single value and each attribute can appear at most once on each element. In the common situation where a list of multiple values is desired, this must be done by encoding the list into a well-formed XML attribute with some format beyond what XML defines itself.

# XML Basics

- **XML**: Shapes of XML

  Key Insights :

  - ✓ **Namespaces**
    An XML namespace is declared using the reserved XML attribute xmlns or xmlns:prefix, the value of which must be a valid namespace name.

    For example, the following declaration maps the "xhtml:" prefix to the XHTML namespace:

    xmlns:xhtml="http://www.w3.org/1999/xhtml"

# XML Basics

- **XML**: Shapes of XML

Key Insights :

- ✓ **Namespaces**
  Any element or attribute whose name starts with the prefix "xhtml:" is considered to be in the XHTML namespace, if it or an ancestor has the above namespace declaration. It is also possible to declare a default namespace → xmlns="http://www.w3.org/1999/xhtml"
  - In this case, any element without a namespace prefix is considered to be in the XHTML namespace, if it or an ancestor has the above default namespace declaration.

  Attributes are never subject to the default namespace.
  An attribute without an explicit namespace prefix is considered not to be in any namespace.

# XML Basics

- **XML**: Shapes of XML

  The XML Syntax

  ✓ EXACTLY ONE Root Element

# XML Basics

- **XML**: Shapes of XML

  The XML Syntax

  - ✓ EXACTLY ONE Root Element
  - ✓ Child Elements
    - Other elements on the same level (siblings)
    - Other elements on nested levels (parent-child relations)
    - Text/Empty Elements etc…
    - Definitions start ONLY with character (except "xml…")
    - Further info about element can be stored as:
      - Attribute (metadata)
      - child element

# XML Basics

- **XML**: Shapes of XML

  XML Basic Rules …

# XML Basics

- **XML**: Shapes of XML

  XML Basic Rules …

  - ✓ Root element (encapsulates all tags)
    <package></package>

# XML Basics

- **XML**: Shapes of XML

  XML Basic Rules …

  - ✓ Root element (encapsulates all tags)
    <package></package>
  - ✓ Case sensitive (<Name></name> → WRONG!!! )

# XML Basics

- **XML**: Shapes of XML

  Right syntax on XML Document at a glance:

  - ✓ Only one Root Element is defined inside the document,
  - ✓ Elements are Children of Root Element,
  - ✓ Tags are properly defined (no unclosed tags),
  - ✓ Tags are properly closed (XML is case sensitive),
  - ✓ Elements follow a proper hierarchy,
  - ✓ Right syntax on elements' definitions.

# XML Basics


- **XML**: Shapes of XML

Right syntax on XML Document:

- ✓ The document contains only properly encoded legal Unicode characters.
- ✓ None of the special syntax characters such as < and & appear except when performing their markup-delineation roles.
- ✓ The start-tag, end-tag, and empty-element tag that delimit elements are correctly nested, with none missing and none overlapping.

# XML Basics


- **XML**: Shapes of XML

Right syntax on XML Document:

- ✓ Tag names are case-sensitive; the start-tag and end-tag must match exactly.
- ✓ Tag names cannot contain any of the characters !"#$%&'()*+,/;<=>?@[\]^`{|}~, nor a space character, and cannot begin with "-", ".", or a numeric digit.
- ✓ A single root element contains all the other elements.

# XML Basics

- **XML**: Shapes of XML

  The XML Document Structure

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

<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
        <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:string"/>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

# XML Basics

- **XML**: Shapes of XML

## The XML Document Structure

```
<?xml version="1.0"?>
<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

# XML Basics

- **XML**: Shapes of XML

## The XML Document Structure

```
<?xml version="1.0"?>
<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

# XML Basics

- **XML**: Shapes of XML

## The XML Document Structure

```
<?xml version="1.0"?>
<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

Parent Element

Child Elements

# XML Basics

- **XML**: Shapes of XML

## The XML Document Structure

```
<?xml version="1.0"?>
<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

Root Element

Siblings

Parent Element

Child Elements

# XML Basics

- **XML**: Shapes of XML

  XML Schemas and Validation

  - ✓ An XML document contains a reference to a Document Type Definition (DTD), ensuring that its elements and attributes are declared in that DTD and follow the grammatical rules for them that the DTD specifies.

  - ✓ XML processors are classified as validating or non-validating depending on whether or not they check XML documents for validity. A processor that discovers a validity error must be able to report it, but may continue normal processing.

# XML Basics

- **XML**: Shapes of XML

  XML Schemas and Validation

  - ✓ A DTD is an example of a schema or grammar. A schema languages typically constrain the set of elements that may be used in a document, which attributes may be applied to them, the order in which they may appear, and the allowable parent/child relationships.

# XML Basics

- **XML**: Shapes of XML

  Document Type Definition (DTD)

  ✓ The oldest schema language for XML.

  DTDs have the following benefits:

  - Support is ubiquitous due to its inclusion in the XML 1.0 standard.
  - Present more information in a single screen.
  - Define a document type rather than the types used by a namespace, thus grouping all constraints for a document in a single collection.

# XML Basics

- **XML**: Shapes of XML

  Document Type Definition (DTD)

  ✓ The oldest schema language for XML.

  DTDs have the following limitations:

  - They have no explicit support for newer features of XML, most importantly namespaces.

  - They lack expressiveness and readability.
    There are certain structures that cannot be expressed with regular grammars as DTDs only support basic data-types, using syntax based on regular expression syntax, to describe the schema.

# XML Basics

- **XML**: Shapes of XML

Document Type Definition (DTD)


```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>

<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

# XML Basics

- **XML**: Shapes of XML

Document Type Definition (DTD)

External

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

# XML Basics

- **XML**: Shapes of XML

  XML Schema Definition (XSD)

  - ✓ A newer schema language, described by the W3C as the successor of DTDs, is XML Schema, often referred as (XML Schema Definition).

  - ✓ XSDs are far more powerful than DTDs in describing XML languages.

# XML Basics

- **XML**: Shapes of XML

  XML Schema Definition (XSD)

  - ✓ They use a rich data typing system and allow for more detailed constraints on an XML document's logical structure.
  - ✓ XSDs also use an XML-based format, which makes it possible to use ordinary XML tools to help process them.\

# XML Basics

- **XML**: Shapes of XML

  XML Schema Definition (XSD)

  xs:schema element that defines a schema:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
</xs:schema>
```

# XML Basics

- **XML**: Shapes of XML

  XML Schema Definition (XSD)

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
   <xs:element name = "contact">
      <xs:complexType>
         <xs:sequence>
            <xs:element name = "name" type = "xs:string" />
            <xs:element name = "company" type = "xs:string" />
            <xs:element name = "phone" type = "xs:int" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

# XML Basics

- **XML**: Shapes of XML

  XML Schema Definition (XSD)

  - ✓ The main components of a schema are:
    - Element declarations, that define properties of elements.
    - Attribute declarations, that define properties of attributes.
    - Simple and complex types.
    - Model group and attribute group definitions.

# XML Basics

- **XML**: Shapes of XML

  XML Schema Definition (XSD)

  ✓ The main components of a schema are:

  - An attribute use represents the relationship of a complex type and an attribute declaration, and indicates whether the attribute is mandatory or optional when it is used in that type.
  - An element particle similarly represents the relationship of a complex type and an element declaration, and indicates the minimum and maximum number of times the element may appear in the content.

# XML Basics

- **XML**: Shapes of XML

  XML DOM (Data Object Model)

  ✓ The DOM defines a standard for accessing and manipulating documents:

  ✓ 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

# XML Basics

- **XML**: Shapes of XML

  XML DOM (Data Object Model)

  ✓ The DOM defines a standard for accessing and manipulating documents:

  - The HTML DOM defines a standard way for accessing and manipulating HTML documents.
    - It presents an HTML document as a tree-structure.
  - The XML DOM defines a standard way for accessing and manipulating XML documents.
    - It presents an XML document as a tree-structure.

# XML Basics

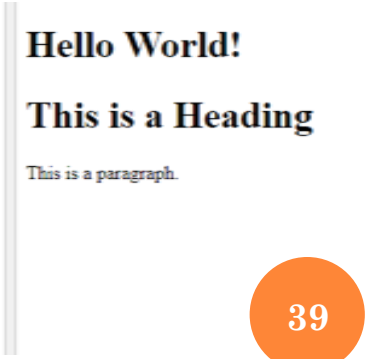
- **XML**: Shapes of XML

  XML DOM (Data Object Model)

  ✓ The DOM defines a standard for accessing and manipulating documents:

  - The HTML DOM defines a standard way for accessing and manipulating HTML documents.

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

<h1>This is a Heading</h1>

<h1>This is a Heading</h1>

<p>This is a paragraph.</p>

<script>
document.getElementsByTagName("h1")[0].innerHTML = "Hello World!";
</script>

</body>
</html>
```



# XML Basics

- **XML**: Shapes of XML

  XML DOM (Data Object Model)

  ✓ The DOM defines a standard for accessing and manipulating documents:

  - The HTML DOM defines a standard way for accessing and manipulating HTML documents.

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

<h1>This is a Heading</h1>

<h1>This is a Heading</h1>

<p>This is a paragraph.</p>

<script>
document.getElementsByTagName("h1")[1].innerHTML = "Hello World!";
</script>

</body>
</html>
```

# XML Basics

- **XML**: Shapes of XML

  XML DOM (Data Object Model)

  ✓ The DOM defines a standard for accessing and manipulating documents:

  - The XML DOM defines a standard way for accessing and manipulating XML documents.

    All XML elements can be accessed through the XML DOM.

    The XML DOM is standard for how to get, change, add, or delete XML elements.

# XML Basics

- **XML**: Shapes of XML

  XML DOM (Data Object Model)

  ✓ The DOM defines a standard for accessing and manipulating documents:

  - The XML DOM defines a standard way for accessing and manipulating XML documents.

    All XML elements can be accessed through the XML DOM.

    The XML DOM is:
    - A standard object model for XML
    - A standard programming interface for XML
    - Platform- and language-independent
    - A W3C standard

# XML Basics

- **XML**: Shapes of XML

  XML DOM (Data Object Model)

  ✓ The DOM defines a standard for accessing and manipulating documents:

  The 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.
  - The 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 Basics

- **XML**: Shapes of XML

  XML DOM (Data Object Model)

  ✓ The DOM defines a standard for accessing and manipulating documents:

  The DOM models XML as a set of node objects.

  - XML DOM Properties (x is a node object)
    - 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 Basics

- **XML**: Shapes of XML

  XML DOM (Data Object Model)

  ✓ The DOM defines a standard for accessing and manipulating documents:

  The DOM models XML as a set of node objects.

  - XML DOM Methods (x is a node object)
    - 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

# XML Basics

- **XML**: Shapes of XML

## XML DOM (Data Object Model)

- ✓ The DOM defines a standard for accessing and manipulating documents:

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

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

<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
};
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>
```

# XML Basics

- **XML**: Shapes of XML

## XML DOM (Data Object Model)

- ✓ The DOM defines a standard for accessing and manipulating documents:

```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>

</bookstore>
```

"books.xml"

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

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

<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
};
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>
```

# XML Basics

- **XML**: Shapes of XML

  XML DOM Parser

  - ✓ All major browsers have a built-in XML parser to access and manipulate XML.
  - ✓ The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.
  - ✓ However, before an XML document can be accessed, it must be loaded into an XML DOM object.
  - ✓ All browsers have a built-in XML parser that can convert text into an XML DOM object.

# XML Basics

- **XML**: Shapes of XML

  XML DOM Parser (example)

```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>
```

"books.xml"

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

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

</body>
</html>
```

# XML Basics

- **XML**: Shapes of XML

  XML DOM Parser (example)

```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>
```

"books.xml"

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

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

</body>
</html>
```

# XML Basics

- **XML**: Shapes of XML

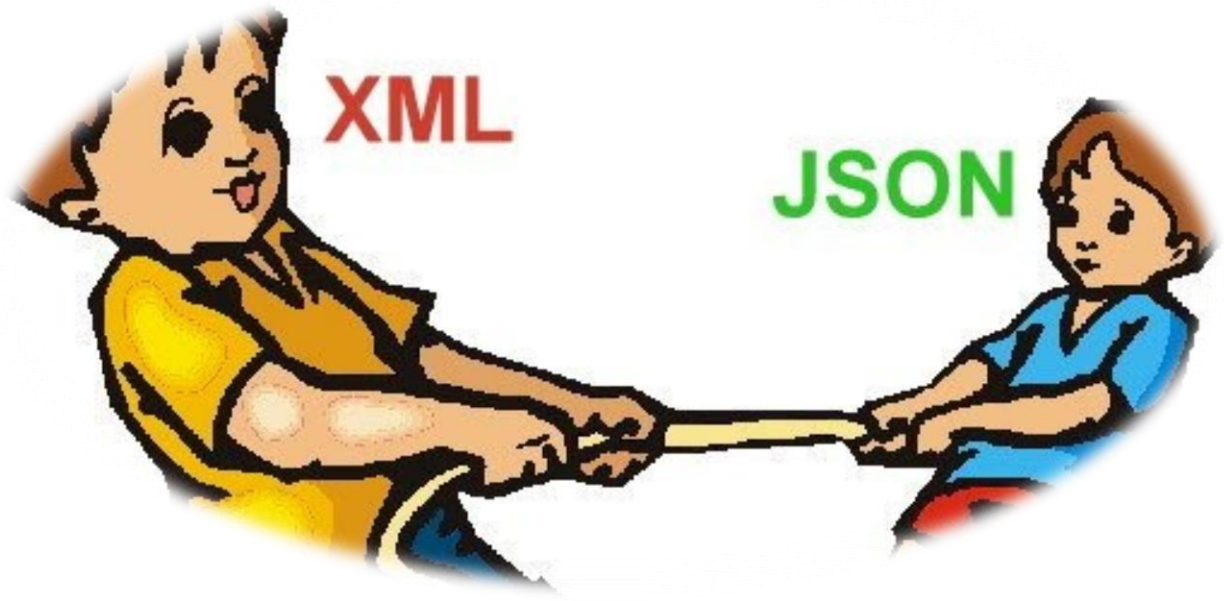
  XML DOM Parser (example)

```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>
```

"books.xml"

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

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

</body>
</html>
```

Everyday Italian
Harry Potter


XML
JSON

# JSON INTRO

- **JSON**: **J**ava **S**cript **O**bject **N**otation

  An open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).

  - ✓ lightweight data-interchange format,
  - ✓ easy for humans to read and write,
  - ✓ easy for machines to parse and generate,
  - ✓ based on a subset of the JS Programming Language.

Standard ECMA-262 3rd Edition - December 1999

# JSON Intro

- **JSON**: **J**ava **S**cript **O**bject **N**otation

## Basic Data Types:

- **Number**: a signed decimal number that may contain a fractional part and may use exponential E notation, but cannot include non-numbers such as NaN. The format makes no distinction between integer and floating-point. JavaScript uses a double-precision floating-point format for all its numeric values, but other languages implementing JSON may encode numbers differently.

- **String**: a sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.

- **Boolean**: either of the values true or false

# JSON INTRO

- **JSON**: **J**ava **S**cript **O**bject **N**otation

  **Basic Data Types:**

  - **Object:** an unordered collection of name–value pairs where the names (also called keys) are strings. Since objects are intended to represent associative arrays, it is recommended, though not required, that each key is unique within an object. Objects are delimited with curly brackets and use commas to separate each pair, while within each pair the colon ':' character separates the key or name from its value.

  - **Array:** an ordered list of zero or more values, each of which may be of any type. Arrays use square bracket notation and elements are comma-separated.

  - **null:** An empty value, using the word null

# JSON Basics

- **JSON**: **J**ava **S**cript **O**bject **N**otation

  **JSON is built on two structures:**

  - A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

  - An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

# JSON Basics

- **JSON**: **J**ava **S**cript **O**bject **N**otation

## Example (person):

```
{
  "firstName": "Sujon",
  "lastName": "Mamun",
  "isAlive": true,
  "age": 30,
  "address": {
    "streetAddress": "Monthana",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

# JSON Basics

- **JSON**: **J**ava **S**cript **O**bject **N**otation

  **JSON Schema & Validation:**

  - ✓ JSON Schema specifies a JSON-based format to define the structure of JSON data for validation, documentation, and interaction control.

  - ✓ It provides a contract for the JSON data required by a given application, and how that data can be modified.

# JSON Basics

- **JSON**: **J**ava **S**cript **O**bject **N**otation

  **JSON Schema & Validation:**

  - ✓ JSON Schema is based on the concepts from XML Schema (XSD), but is JSON-based.
  - ✓ As in XSD, the same serialization/deserialization tools can be used both for the schema and data; and is self-describing.
  - ✓ There are several validators available for different programming languages, each with varying levels of conformance.

# JSON Basics

- **JSON**: **J**ava **S**cript **O**bject **N**otation

## Example – Validation Scheme:

```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}
```

```
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
    "Bar",
    "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

# JSON Basics

- **JSON**: **J**ava **S**cript **O**bject **N**otation

## Example – Validation Scheme:

```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}
```

```
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
    "Bar",
    "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

# JSON BASICS

- **JSON**: **J**ava **S**cript **O**bject **N**otation

## Example – Validation Scheme:

```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}
```

```
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
    "Bar",
    "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

# JSON Basics

- **JSON**: **J**ava **S**cript **O**bject **N**otation

## Example – Validation Scheme:

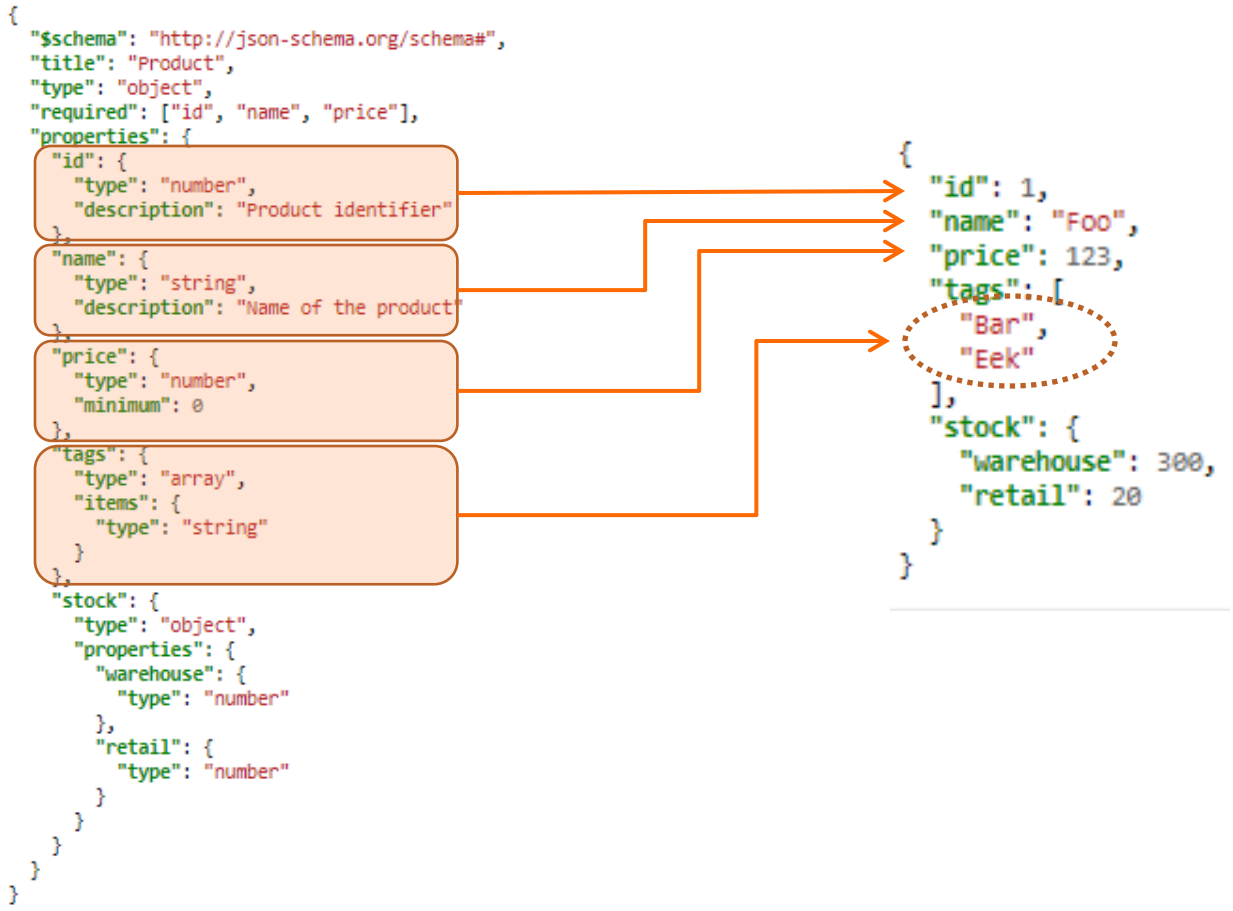
```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}
```

```
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
    "Bar",
    "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

# JSON BASICS

- **JSON**: **J**ava **S**cript **O**bject **N**otation

## Example – Validation Scheme:



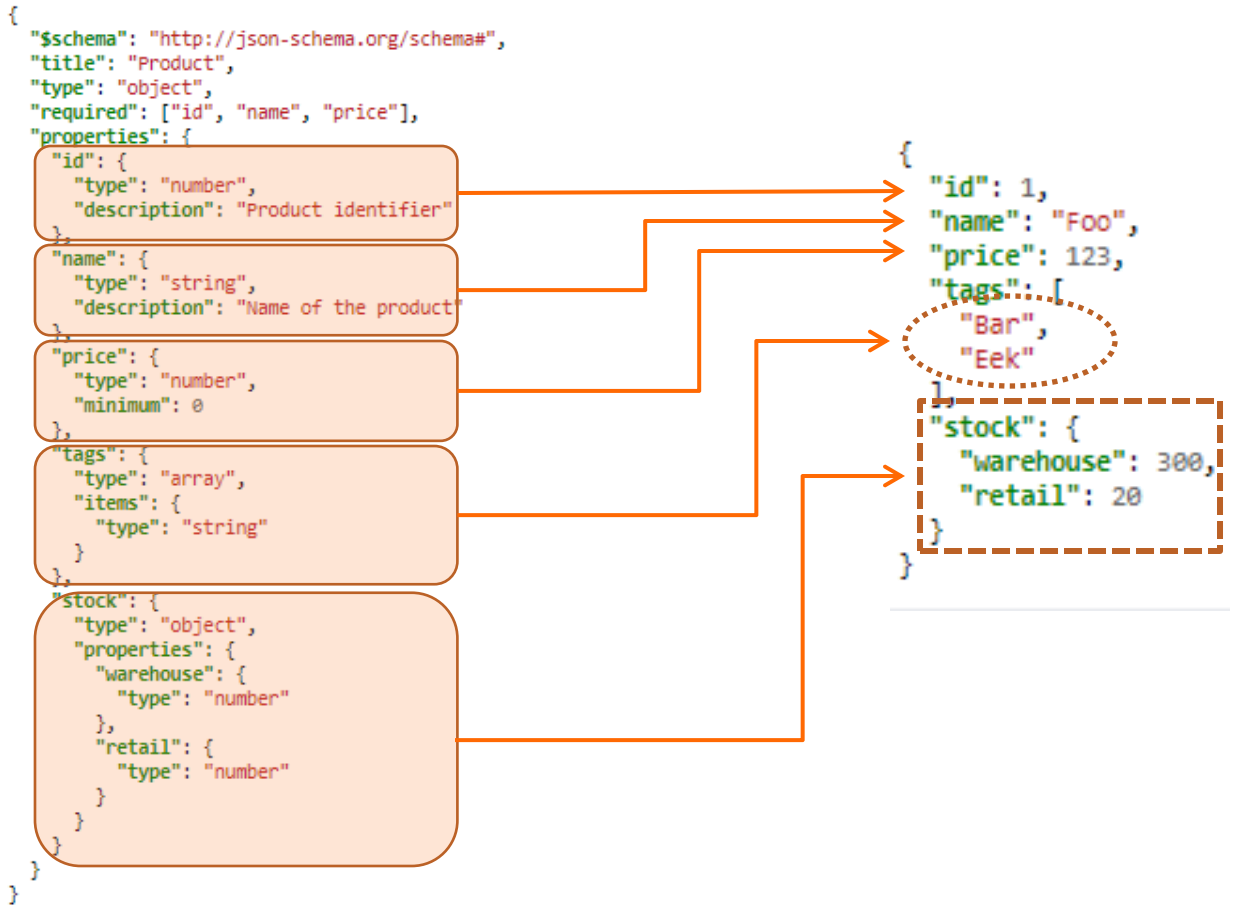
```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}
```

```
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
    "Bar",
    "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

# JSON BASICS

- **JSON**: **J**ava **S**cript **O**bject **N**otation

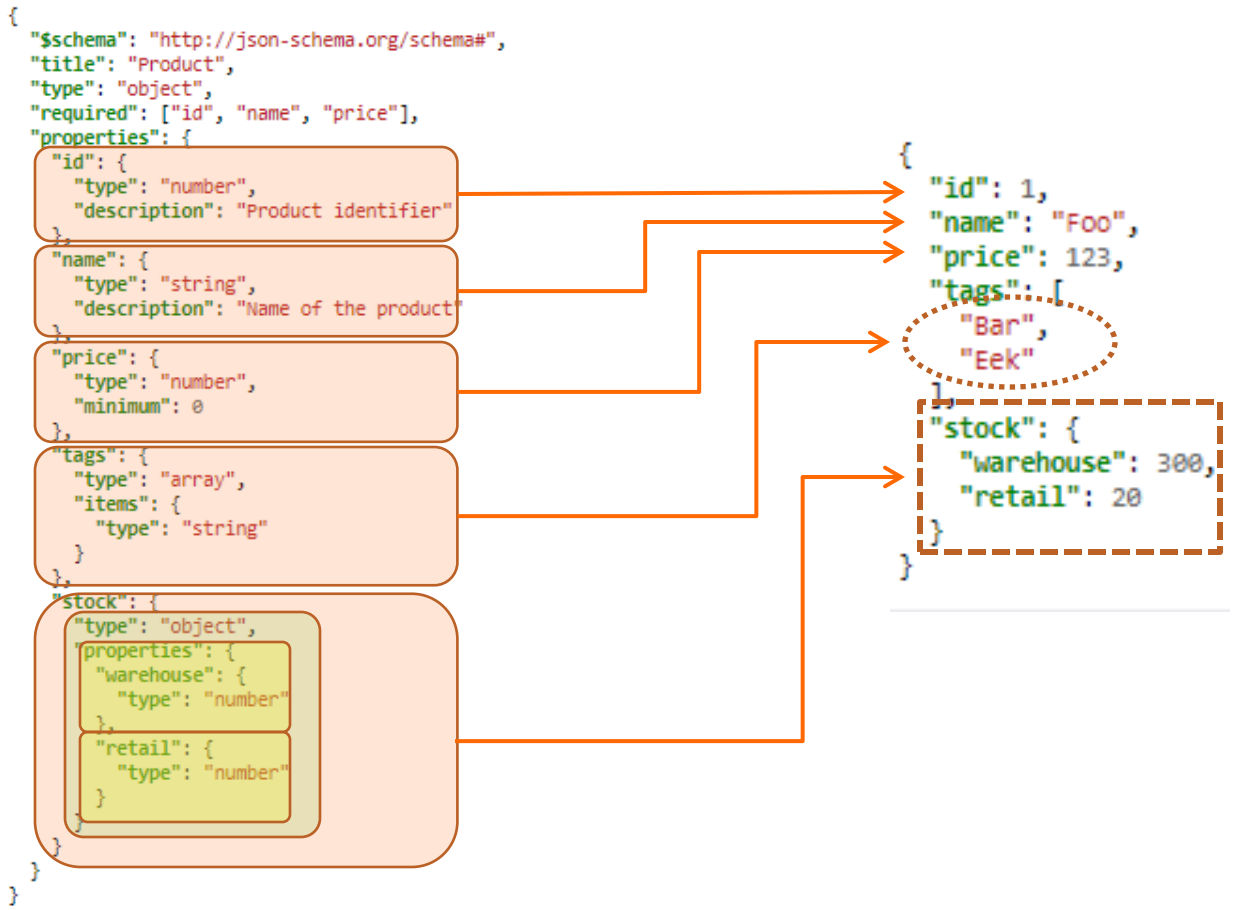
## Example – Validation Scheme:



# JSON BASICS

- **JSON**: **J**ava **S**cript **O**bject **N**otation

## Example – Validation Scheme:



# JSON BASICS

- **JSON**: **J**ava **S**cript **O**bject **N**otation

## Example – Validation Scheme: