

# JSON & XML



## Web Technology

**Asst. Prof. Manop Phankokkruad, Ph.D.**

Faculty of Information Technology

King Mongkut's Institute of Technology Ladkrabang



# Outline

- ❑ What is the JSON?
- ❑ JSON Syntax
- ❑ JSON data types
- ❑ JSON Schema
  - What is XML?
  - XML Structure & Syntax
- ❑ XML and JSON



# Introduction

- ❑ XML and JSON are the two most common formats for data interchange in the Web today.
- ❑ Although their purposes are not identical, they are frequently used to accomplish the same task, which is data interchange.
- ❑ Both have well-documented open standards on the Web (JSON : RFC 7159, XML : RFC 4825), and both are human and machine-readable. Neither one is absolutely superior to the other, as each is better suited for different use cases.

# What is the JSON?

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, ECMA-262 3rd Edition.

- ❑ JSON is a text format that is completely language independent
- ❑ JSON is built on two structures:
  - ❑ A collection of name/value pairs
  - ❑ An ordered list of values

# Uses of JSON

- ❑ It is used while writing JavaScript based applications that includes browser extensions and websites.
- ❑ JSON format is used for serializing and transmitting structured data over network connection.
- ❑ It is primarily used to transmit data between a server and web applications.
- ❑ Web services and APIs use JSON format to provide public data.
- ❑ It can be used with modern programming languages.

# JSON - Syntax

JSON syntax is basically considered as a subset of JavaScript syntax; it includes the following:

- Data is represented in name/value pairs.
- Curly braces hold objects and each name is followed by **:** (colon), the name/value pairs are separated by **,** (comma).
- Square brackets hold arrays and values are separated by **,** (comma).

```
{  
  "book": [  
    {  
      "id": "01",  
      "language": "Java",  
      "edition": "third",  
      "author": "Herbert Schildt"  
    },  
    {  
      "id": "07",  
      "language": "C++",  
      "edition": "second",  
      "author": "E.Balagurusamy"  
    }  
  ]  
}
```

# JSON - Structures

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.

These are universal data structures, and interchangeable with programming languages. JSON take on these forms: object, array, value, string, and number.

# JSON : Data Types

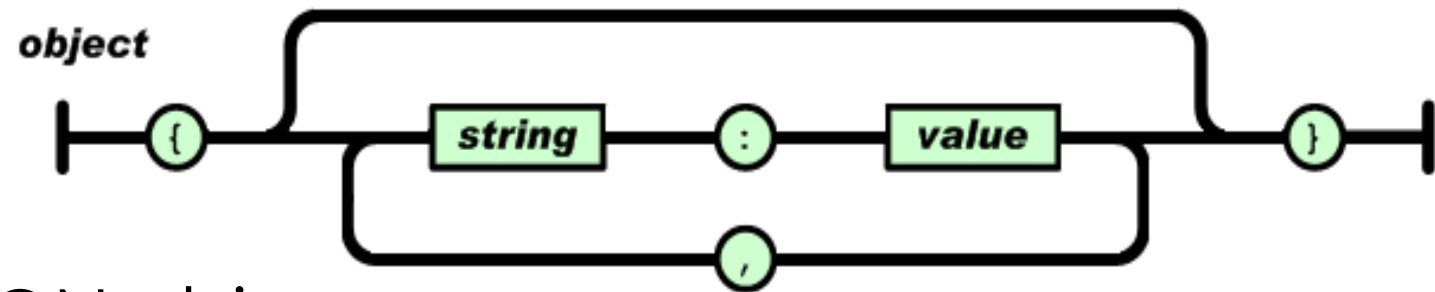
❑ JSON format supports the following data types:

Type	Description
Number	double- precision floating-point format in JavaScript
String	double-quoted Unicode with backslash escaping
Boolean	true or false
Array	an ordered sequence of values
Value	it can be a string, a number, true or false, null etc
Object	an unordered collection of key:value pairs
Whitespace	can be used between any pair of tokens
null	empty



# JSON : object

An **object** is an unordered set of name/value pairs. An object begins with **{** and ends with **}**. Each name is followed by **:** and the name/value pairs are separated by **,**. In a JSON object the name(key) must be unique.



Example of a JSON object:

```
{ "name": "John Doe",  
  "age": 30,  
  "married": true  
}
```

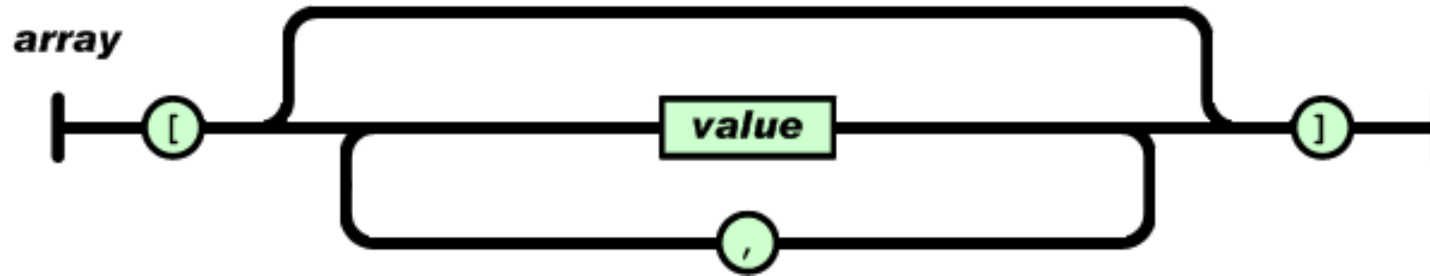
# JSON : object

Example of a simple JSON object:

```
{  
  "address" : {  
    "line1" : "555 Any Street",  
    "city" : "Denver",  
    "stateOrProvince" : "CO",  
    "zipOrPostalCode" : "80202",  
    "country" : "USA"  
  }  
}
```

# JSON : array

An **array** is an ordered collection of values. An array begins with **[** and ends with **]**. Values are separated by **,**.



Example of Object with a nested Array

```
{ "name": "John Doe",  
  "age": 30,  
  "married": true,  
  "siblings": ["John", "Mary", "Pat"]  
}
```

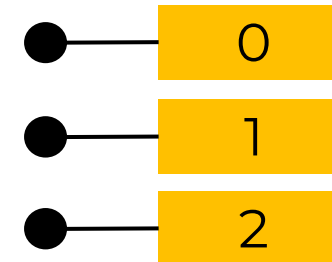
# JSON : array

## Syntax

```
[ value, .....]
```

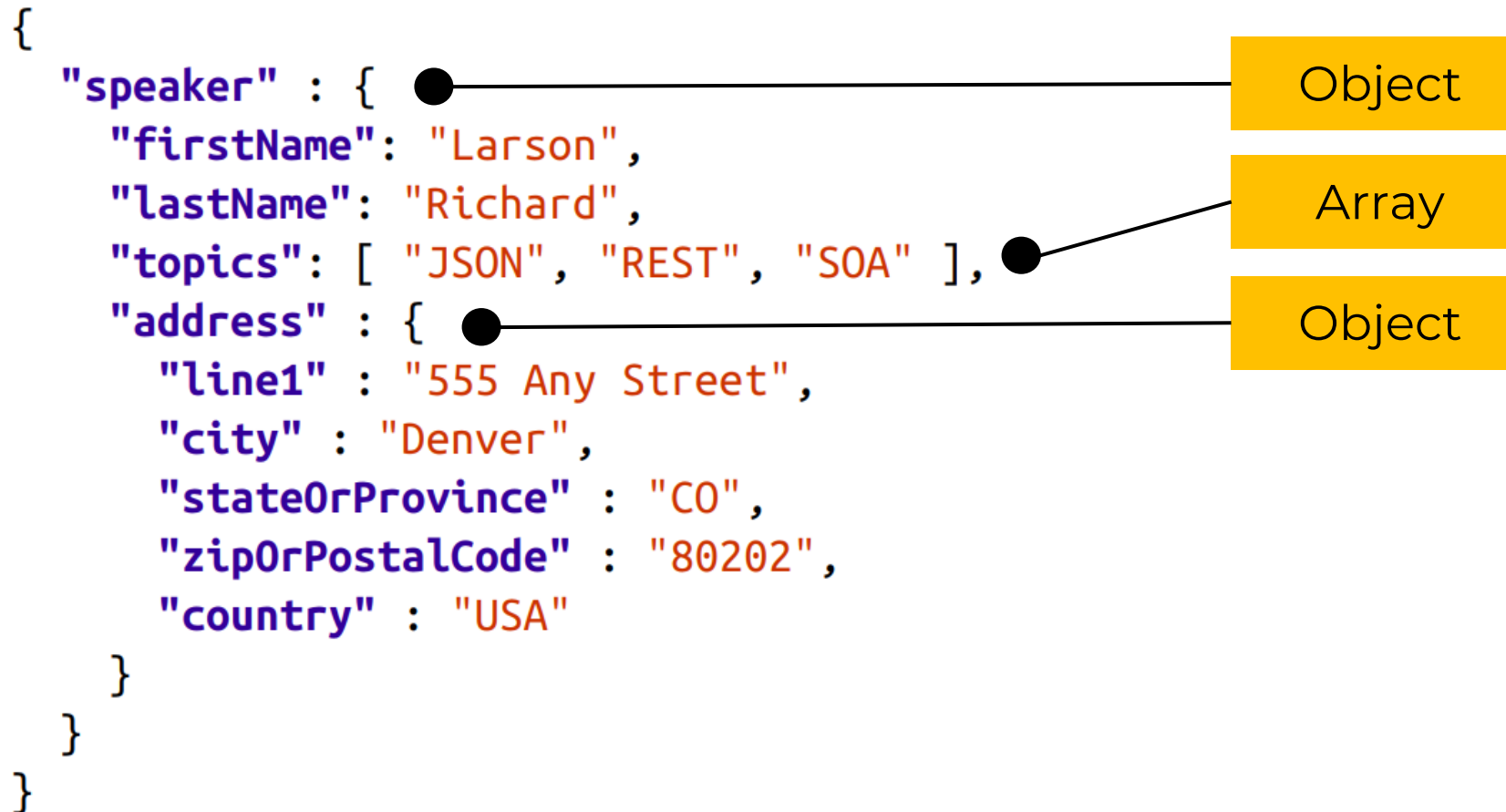
## Example

```
{  
  "books": [  
    { "language": "Java", "edition": "second" },  
    { "language": "C++", "lastName": "fifth" },  
    { "language": "C", "lastName": "third" }  
  ]  
}
```



# JSON : array

Example of Object that contains another Object



# JSON : array of object

Each item in an array may be any of the seven JSON values(as described in the next slide).

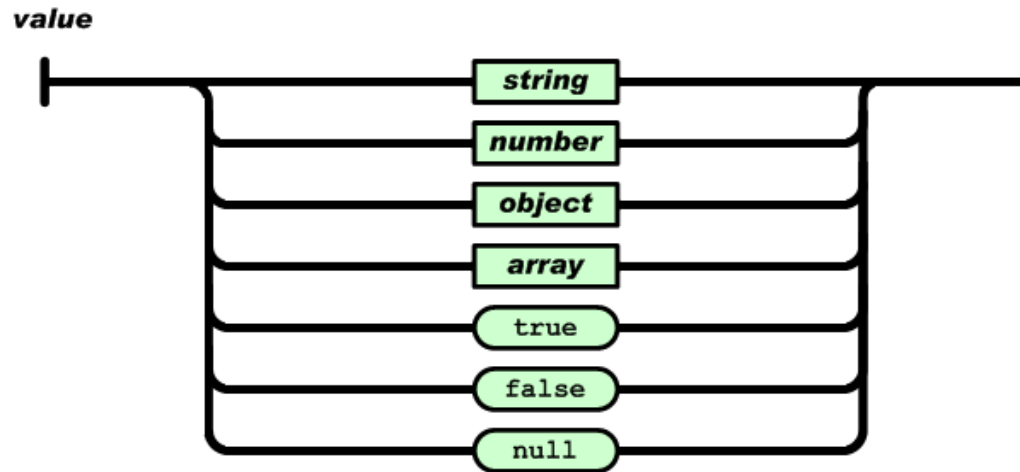
Example of a JSON array of object

```
{  
  "name": "John Doe",  
  "age": 30,  
  "married": true,  
  "siblings": [  
    {"name": "John", "age": 25},  
    true,  
    "Hello World"  
  ]  
}
```

*The array contains 3 items. The first item is an object, the second item is a boolean, and the third item is a string.*

# JSON : value

A **value** can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.



## Syntax

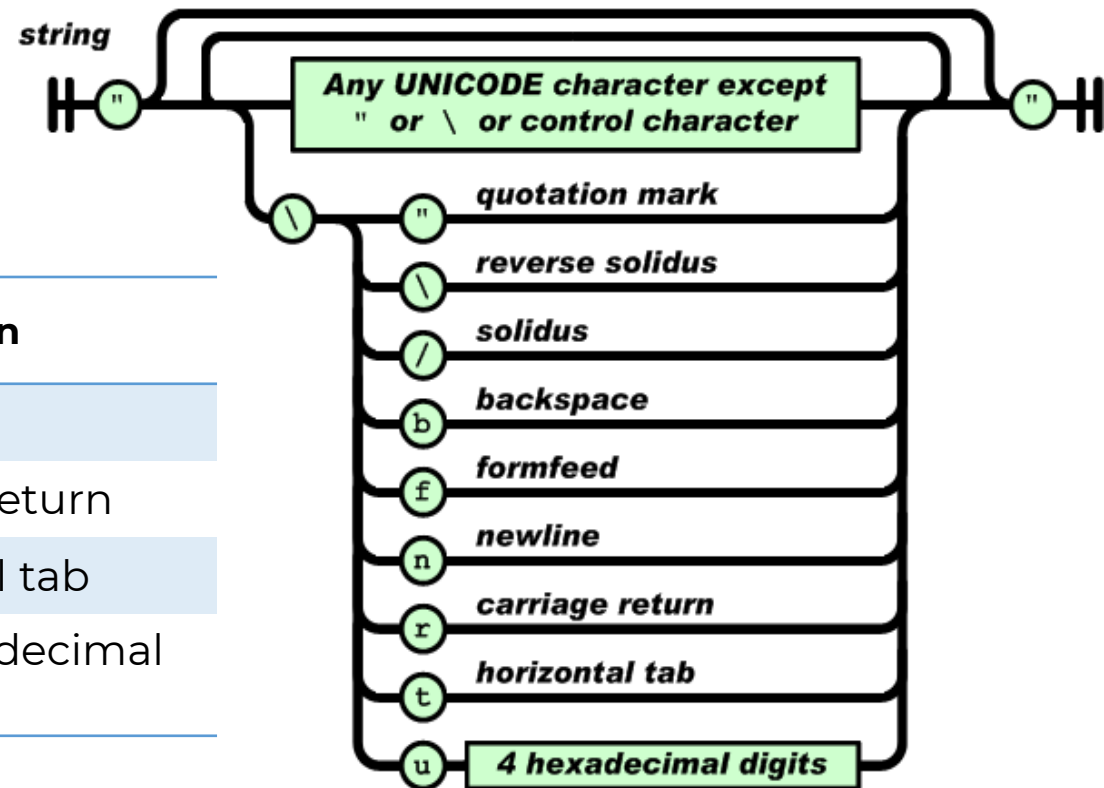
```
let object-name = { name : value, ..... }
```

## Example

```
let obj = { "price": 500, "product": "Shampoo" }
```

# JSON : string

A **string** is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string.



Type	Description	Type	Description
\"	double quotation	\n	New line
\\	backslash	\r	carriage return
\/	forward slash	\t	horizontal tab
\b	backspace	\u	four hexadecimal digits
\f	form feed		



# JSON : string

Example of a JSON string

```
{  
  "name": "John Doe",  
  "age": 30,  
  "married": true,  
  "siblings": ["John", "Mary", "Pat"]  
}
```

Syntax

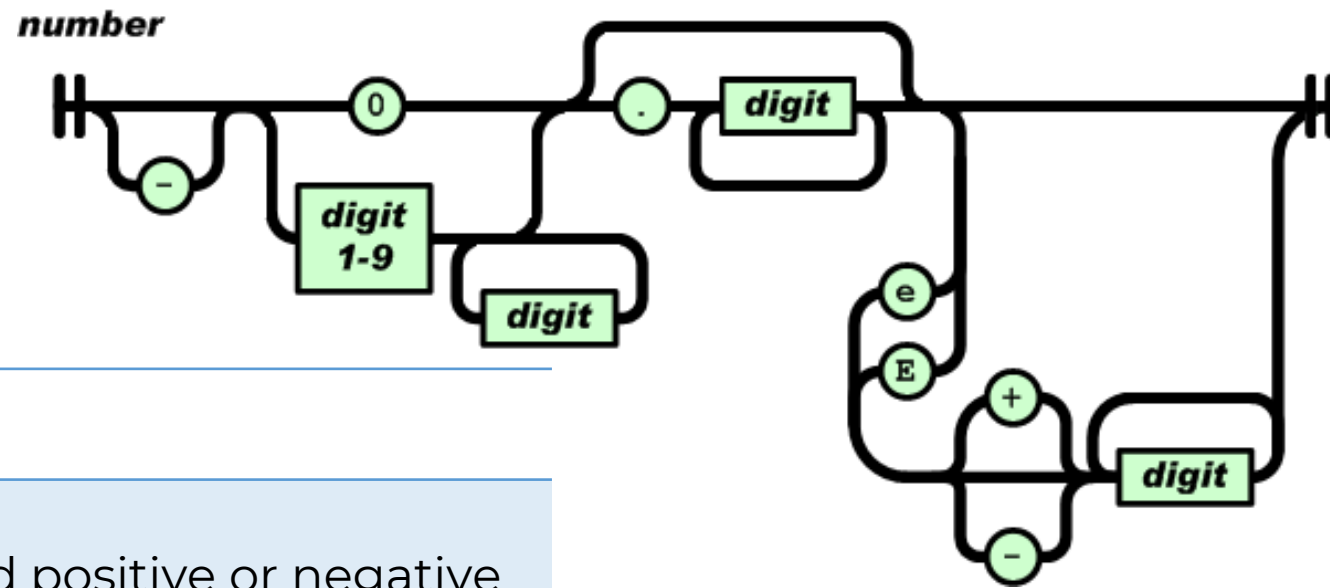
```
let json-object-name = { string : "string value", .....}
```

Example

```
let obj = {"name": "Amit"}
```

# JSON : number

A number is very much like a C or Java number, except that the octal and hexadecimal formats are not used.



Type	Description
Integer	Digits 1-9, 0 and positive or negative
Fraction	Fractions like .3, .7
Exponent	Exponent like e, e+, e-, E, E+, E-

# JSON : number

## Syntax

```
let json-object-name = {"string" : number_value, .....}
```

## Example

```
{  
  "age": 29,  
  "cost": 299.99,  
  "temperature": -10.5,  
  "unitCost": 0.2,  
  "speedOfLight": 1.23e11,  
  "speedOfLight2": 1.23e+11,  
  "avogadro": 6.023E23,  
  "avogadro2": 6.023E+23,  
  "oneHundredth": 10e-3,  
  "oneTenth": 10E-2  
}
```

# JSON : boolean

Booleans have the following properties:

- Booleans can have a value of only true or false.
- The true or false value on the righthand side of the colon(:) is not surrounded by quotes.

Example of a JSON boolean

```
{  
  "isRegistered": true,  
  "emailValidated": false  
}
```

# JSON : null

**null** values have the following characteristics:

- Are not surrounded by quotes

Example of a JSON null

```
{  
  "address": {  
    "line1": "555 Any Street",  
    "line2": null,  
    "city": "Denver",  
    "stateOrProvince": "CO",  
    "zipOrPostalCode": "80202",  
    "country": "USA"  
  }  
}
```

# JSON Schema

- ❑ JSON Schema is a specification for JSON based format for defining the structure of JSON data. It was written under IETF draft.
- ❑ JSON Schema :
- ❑ Describes your existing data format.
- ❑ Clear, human- and machine-readable documentation.
- ❑ Complete structural validation, useful for automated testing.
- ❑ Complete structural validation, validating client-submitted data.



# JSON Schema

## ❑ JSON Schema Example

```
{  
  "firstName": "John",  
  "lastName": "Doe",  
  "age": 21  
}
```

JSON

```
{  
  "$id": "https://example.com/person.schema.json",  
  "$schema": "http://json-schema.org/draft-07/schema#",  
  "title": "Person",  
  "type": "object",  
  "properties": {  
    "firstName": {  
      "type": "string",  
      "description": "The person's first name."  
    },  
  },  
}
```

JSON Schema

# JSON Schema

## ❑ JSON Schema Example (cont.)

```
"lastName": {  
  "type": "string",  
  "description": "The person's last name."  
},  
"age": {  
  "description": "Age in years which must be equal to or greater than  
zero.",  
  "type": "integer",  
  "minimum": 0  
}  
}
```

JSON Schema



# JSON and JavaScript

The JSON format is syntactically identical to the code for creating JavaScript objects. Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

JSON data is normally accessed in JavaScript through dot notation.

```
let person = { "name": "Brad", "age": 31, "address": { "city": "Boston" } };  
document.write( person.name );           // Brad  
document.write( person.address.city );    // Boston  
document.write( person["name"] );         // Brad
```

# Functions for Working with JSON

The **JSON.stringify()** function converts an object to a JSON string.

```
var obj = {"first_name" : "Sammy", "last_name" : "Shark",  
"location" : "Ocean"}  
let s = JSON.stringify(obj);
```

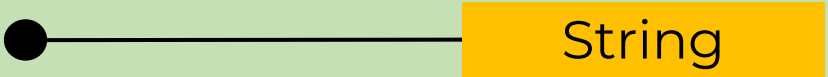
We'll have the JSON available to us as a string rather than an object.

```
'{"first_name" : "Sammy", "last_name" : "Shark", "location" :  
"Ocean"}'
```

# Functions for Working with JSON

The **JSON.parse()** method parses a JSON string, constructing the JavaScript value or object.

```
<script>  
let s = '{"first_name": "Sammy", "last_name": "Shark",  
"location": "Ocean"}';  
let obj = JSON.parse(s);  
  
document.write("Name: " + obj.first_name + "  
+obj.last_name + "<br>");  
Document.write("Location: " + obj.location );  
</script>
```



A diagram illustrating the variable `s` in the code above. A black dot representing the variable `s` is connected by a horizontal line to a yellow rectangular box containing the word `String`, indicating that `s` holds a JSON string.

# What is XML ?

“eXtensible Markup Language(XML) is set of rules for defining and representing data as structured documents for applications on the Internet.”

- ❑ XML is a restricted form of SGML (Standard Generalized Markup Language).
- ❑ XML is designed to describe data.
- ❑ XML is compatible with major Internet transmission protocols and is also highly compressible for faster transmission.
- ❑ Almost all major software vendors fully support the general XML standard.

# What is XML ?

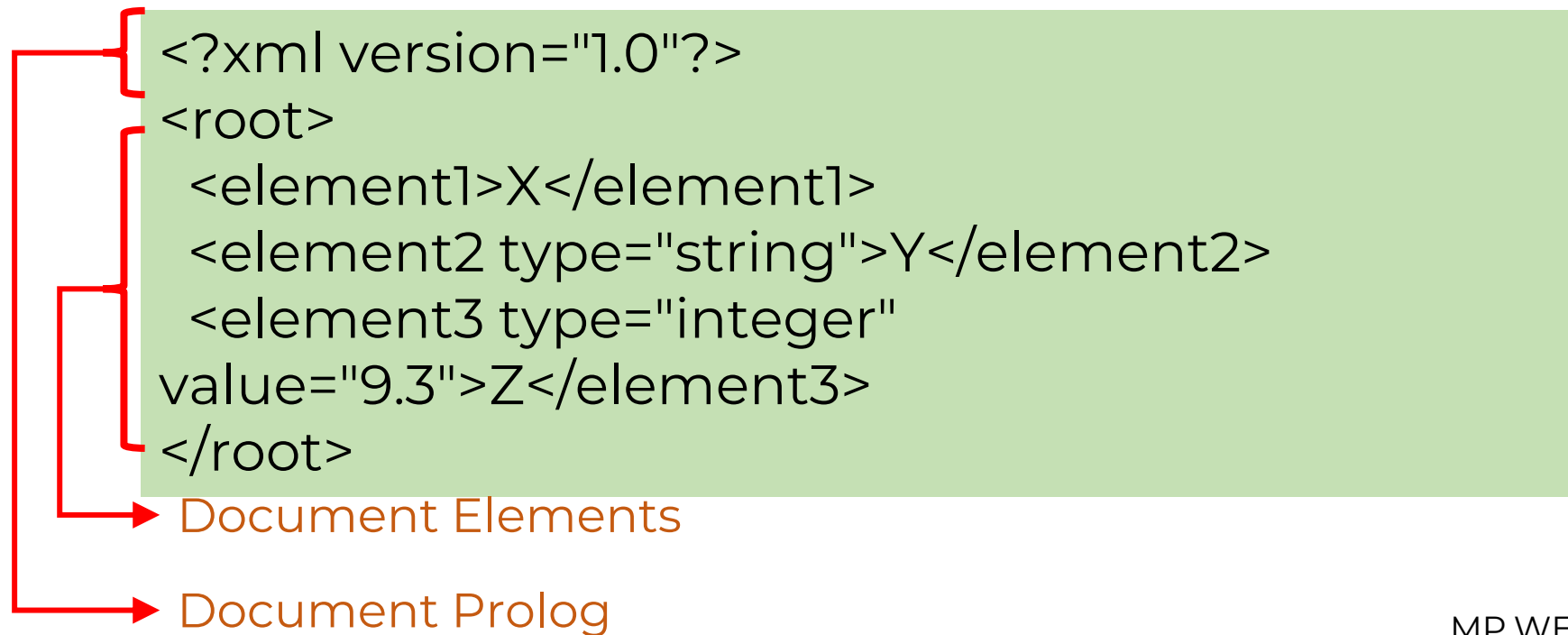
- ❑ The XML standard is designed to be independent of vendor, operating system, source application, destination application, database, and transport protocol.
- ❑ XML Tags are added to the document to provide the extra information.
- ❑ XML tags are not predefined unlike HTML.
- ❑ XML tags give a reader some idea what some of the data means.
- ❑ There are 3 main components for XML content: XML, DTD and XML Schema define rules to describe data.

# Advantages of Using XML

- ❑ XML is text (Unicode) based.
  - ❑ Takes up less space.
  - ❑ Can be transmitted efficiently.
  - ❑ Easily readable by human users
- ❑ One XML document can be displayed differently in different media.
- ❑ XML documents can be modularized. Parts can be reused.
- ❑ Very flexible and customizable (no finite tag set)
- ❑ Easy to convert into other representations (XML transformation languages)

# XML Structure

- **Document Prolog** comes at the top of the document. This section contains XML declaration and Document type declaration.
- **Document Elements** are the building blocks of XML.



# Building Blocks of XML

- ❑ Elements (Tags) are the primary components of XML documents.

*Element author with Attribute id* → `<?xml version="1.0"/>`  
`<author id = "0123">`

*Element fname nested inside element author.* → `<fname>Manop</fname>`  
`<lname>Phankokkruad</lname>`  
`<phone>212-346-1234</phone>`  
`</author>`  
`<!-- I am comment -->`

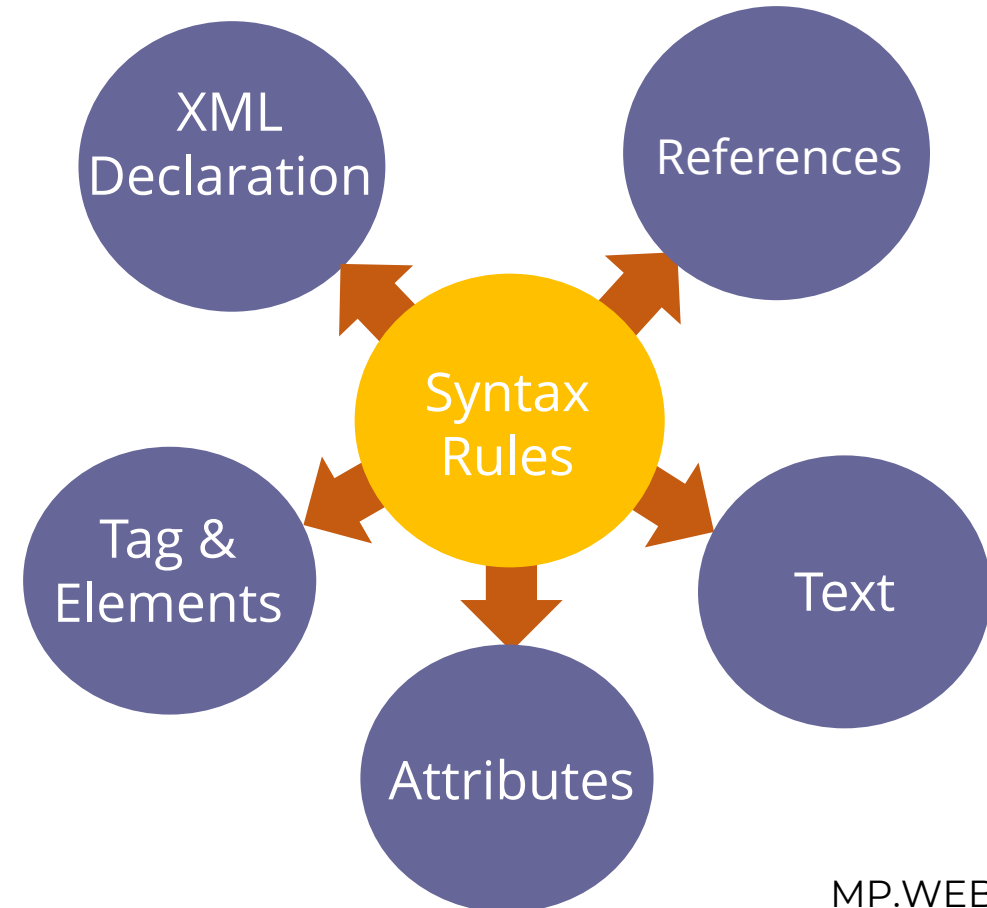
- ❑ Attributes provide additional information about Elements. Values of the Attributes are set inside the Elements .
- ❑ Comments start with `<!--` and end with `-->`.



# XML – Syntax

A diagram depicts the syntax rules to write different types of markup and text in an XML document.

1. XML Declaration
2. Tags and Elements
3. XML Attributes
4. XML References
5. XML Text



# XML Declaration

The XML document can optionally have an XML declaration. It is written as follows:

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

## Syntax Rules for XML Declaration

- The XML declaration is case sensitive and must begin with **<?xml>** where **xml** is written in lower-case.
- It strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.

# Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. The names of XML-elements are enclosed in triangular brackets “< >” .

## Syntax Rules for Tags and Elements

- **Element Syntax:** Each XML-element needs to be closed either with start or with end elements.

```
<element> ... </element>
```

- **Nesting of Elements:** An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap.

# Tags and Elements

**Root Element:** An XML document can have only one root element.

```
<root>  
  <x>...</x>  
  <y>...</y>  
</root>
```

**Case Sensitivity:** The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example, **<contact-info>** is different from **<Contact-Info>**.

# XML Attributes

An attribute specifies a single property for the element, using a name/value pair. An XML element can have one or more attributes.

```
<person gender="female"> ... </person>
```

## Syntax Rules for XML Attributes

- Attribute names in XML are case sensitive.
- Same attribute cannot have two values in a syntax.
- Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks.

# XML References

References usually allow you to include additional text or markup in an XML document. References always begin with the symbol "&" which is a reserved character and end with the symbol ";".

- **Entity References:** An entity reference contains a name between the start and the end delimiters.
- **Character References:** These contain references, such as `&#65;`. In this case, 65 refers to alphabet "A".

# XML References

## □ An example of XML References

Character	Predeclared Entity
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

```
<gangster name='George "Shotgun" Ziegler'>
```

```
<gangster name="George  
&quot;Shotgun&quot; Ziegler">
```



# XML Text

- ❑ The names of XML-elements and XML-attributes are case-sensitive. To avoid character encoding problems, all XML files should be saved as **Unicode UTF-8 or UTF-16** files.
- ❑ Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored.
- ❑ Some characters are reserved by the XML syntax itself. To use them, some replacement-entities are used.



# Validity

- ❑ A well-formed document has a tree structure and obeys all the XML rules.
- ❑ A particular application may add more rules in either a DTD (document type definition) or in a schema.
- ❑ Many specialized DTDs and schemas have been created to describe particular areas.

# Validity

- ❑ An XML document with correct syntax is called **well-formed** document.
  - ❑ 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 and JSON

- ❑ JSON and XML are human readable formats and are language independent. They both have support for creation, reading and decoding in real world situations.
- ❑ **Verbose** : XML is more verbose than JSON, so it is faster to write JSON for programmers.
- ❑ **Arrays Usage** : XML is used to describe the structured data, which doesn't include arrays whereas JSON include arrays.
- ❑ **Parsing** : JavaScript's eval method parses JSON. When applied to JSON, eval returns the described object.

# XML and JSON

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

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

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

# XML and JSON

Trees are well-studied

- ❑ The tree data structure has been well-studied by computer scientists and mathematicians.
- ❑ There are many well-known algorithms for processing and traversing trees.
- ❑ Both XML and JSON can leverage this.

# XML and JSON

- ❑ Both JSON and XML can be used to receive data from a web server.
- ❑ XML is much more difficult to parse than JSON.
- ❑ JSON is parsed into a ready-to-use JavaScript object.

## Like

- self describing (human readable)
- hierarchical (values within values)
- be parsed and used by lots of programming languages
- be fetched with an XMLHttpRequest

## Unlike

- doesn't use end tag
- shorter
- quicker to read and write
- can use arrays

# More Information

- ❑ Introducing JSON  
<https://https://www.json.org>
- ❑ What is JSON?  
[https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)
- ❑ JSON - Introduction  
[https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)
- ❑ JSON Tutorial  
<https://www.tutorialspoint.com/json/index.htm>
- ❑ JavaScript Tutorial  
<https://www.w3schools.com/js/default.asp>
- ❑ XML DOM Tutorial  
[https://www.w3schools.com/xml/dom\\_intro.asp](https://www.w3schools.com/xml/dom_intro.asp)
- ❑ XML DOM Tutorial  
<https://www.tutorialspoint.com/dom/>