Java Script Object Notation
JSON

- Introduction to the JavaScript Object Notation (JSON)
- Need of JSON
- JSON Syntax Rules
- JSON Data - a Name and a Value
- JSON Objects
- JSON Arrays
- JSON Uses JavaScript Syntax
- JSON Files
- JSON & Security Concerns, Cross Site Request Forgery (CSRF), Injection Attacks,
- JS XMLHttpRequest functions
- JavaScript XMLHttpRequest & Web APIs
- JSON & Client Side Frameworks
- JSON & Server Side Frameworks
- Replacing XML with JSON
- JSON parsing
- AJAX using JSON and jQuery

Introduction to the JavaScript Object Notation (JSON)

1) JSON (JavaScript Object Notation) is a lightweight data-interchange format.

2) JSON is a syntax for storing and exchanging data.

3) JSON is an easier-to-use alternative to XML.

4) It is based on a subset of the JavaScript Programming Language

Introduction to the JavaScript Object Notation (JSON)

5) JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.

6) A JSON filetype is .json.

7) The JSON format was originally specified by Douglas Crockford,

Need of JSON

Standard Structure: As we have seen so far that JSON objects are having a standard structure that makes developers job easy to read and write code, because they know what to expect from JSON.

Light weight: When working with AJAX, it is important to load the data quickly and asynchronously without requesting the page re-load. Since JSON is light weighted, it becomes easier to get and load the requested data quickly.

Scalable: JSON is language independent, which means it can work well with most of the modern programming language. Let's say if we need to change the server side language, in that case it would be easier for us to go ahead with that change as JSON structure is same for all the languages.

JSON VS XML:

In the formatting of transferring data, JSON is similar to XML but lighter in weight. Moreover each value in XML requires the opening and closing tag, where as in JSON just requires the value.

JSON data is sent to server in a palin text format, there is no need to set content-type header in JSON data.

Java Script Object Notation

JSON Style:

```
{
  "Students": [
              {
                "name": "John",
                "age":"23",
"city":"Agra"
},
{
"name": "Steve",
"age":"28",
"city":"Delhi"
},
{
"name": "Peter",
"age":"32",
"city":"Chennai"
},
{
"name": "Chaitanya",
"age":"28",
"city":"Bangalore"
              }
         ]
}
```

Extensible Markup Language (*XML*)

XML Style:

```
<students>
 <student>
  <name>John</name><age>23</age>
<city>Agra</city>
 </student>
 <student>
  <name>Steve</name><age>28</age>
<city>Delhi</city>
 </student>
 <student>
  <name>Peter</name><age>32</age>
<city>Chennai</city>
 </student>
 <student>
  <name>Chaitanya</name><age>28</age>
<city>Bangalore</city>
 </student>
</students>
```

JSON vs XML

- JSON is lightweight thus simple to read and write.
- JSON supports array data structure.
- JSON files are more human readable.
- JSON has no display capabilities .
- Provides scalar data types and the ability to express structured data through arrays and objects.
- Native object support.

- XML is less simple than JSON.
- XML doesn't support array data structure.
- XML files are less human readable.
- XML provides the capability to display data because it is a markup language.
- Does not provide any notion of data types. One must rely on XML Schema for adding type information.
- Objects have to be expressed by conventions, often through a mixed use of attributes and elements.

Similarities between JSON and XML

Both are simple and open.

Both supports unicode. So internationalization is supported by JSON and XML both.

Both represents self describing data.

Both are interoperable or language-independent.

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).

Below is a simple example −

```
{
  "book": [

    {
      "id":"01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt",
    },

    {
      "id":"07",
      "language": "C++",
      "edition": "second",
      "author": "E.Balagurusamy",
    }

  ]
}
```

JSON supports the following two data structures −

Collection of name/value pairs − This Data Structure is supported by different programming languages.

Ordered list of values − It includes array, list, vector or sequence etc.

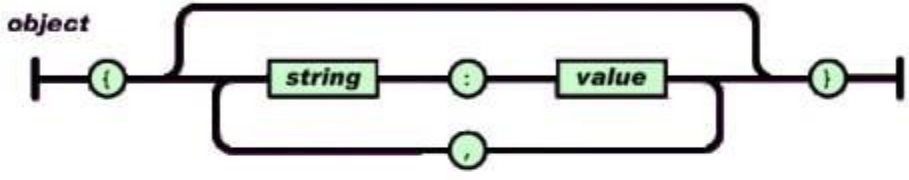
Data Types

- Number - It can be real number, interger or a floating number
- String – It can be any text or Unicode double-quoted with blackslash escapement
- Boolean – The Boolean data type represents the true or false value
- Null – The Null valuue represents that the associated variable dosent have any value
- Object – it is collection of key-value pairs separated by comma and encosed in curly brackets
- Array – it is an ordered serquence of values seperated

JSON Object Syntax Rule

- 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 object begins with {(left brace) and ends with }(right brace). Each name is followed by :(colon) and the name/value pairs are separated by ,(comma).
- The string value must be enclosed within double quote " ".
- The number value will be simply written.
- The boolean values will be simple written as true or false.
- JSON supports numbers in double precision floating-point format digit, fraction and exponent

JSON Object



- **JSON Object Example - 1:**

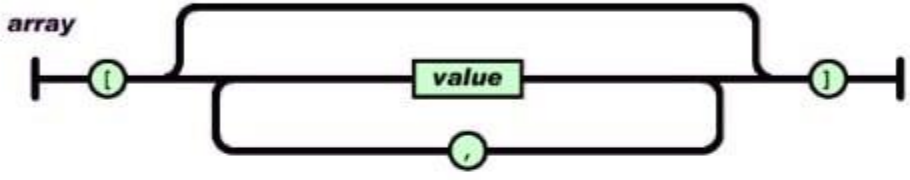
The "{" curly brace represents the JSON object.

```
{"student":{
  "firstName" : "json",
  "hometown" : "Hyderabad",
  "age" : "28"
}}
```

To access the information stored in json, we can **simply refer to the name of the property we need.**
document.write('Jason is ' jason.age); // Output: Jason is 28

document.write('Jason is a ' jason.hometown); // Output: Jason is Hyderabad

JSON Array



An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence. An array is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).

- JSON array represents ordered list of values.
- JSON array can store similar type of multiple values.
- It can store string, number, boolean or object in JSON array.
- The "[" (square bracket) represents the JSON array.
- In JSON array, values must be separated by comma **(,)**.
- A JSON array can have values and objects.

JSON Array Example

["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

[24,21,23,56,21,65,105]

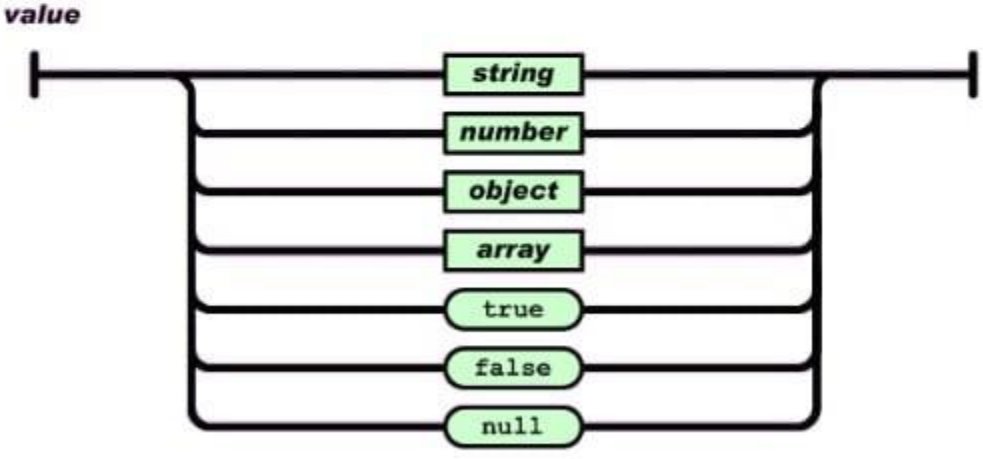
enclosing multiple objects in square brackets,which signifies an array

```
var student = [ {
   "firstName" : "json",
   "hometown" : "Hyderabad",
   "age" : "23"
},
{
   "firstName" : "xml",
   "hometown" : "Secunderabad",
   "age" : "24"
}
];
```

To access this information, we need to access the array index of the person we wish to access. For example, we would use the following snippet to access info stored in student:

```
document.write(student[1].firstname); // Output: xml
document.write(student[0].hometown); // Output: Hyderabad
```

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.



A string value must always be surrounded in double quotes.

{ "animal" : "cat" }

This is not valid JSON

{'title': 'This is my title.',
'body': 'This is the body.'
}

This code won't work

{"promo": "Say "Bob's the best!" at checkout for free 8oz bag of kibble."
}

Using a backslash character to escape quotes inside of strings fixes the problem

{
"promo": "Say \"Bob's the best!\" at checkout for free 8oz bag of kibble."
}

The JSON Number Data Type

A number in JSON can be an integer, decimal, negative number, or an exponent.

```
{
"widgetInventory": 289,
"sadSavingsAccount": 22.59,
"seattleLatitude": 47.606209,
"seattleLongitude": -122.332071,
"earthsMass": 5.97219e+24
}
```

The JSON Boolean Data Type:

```
{
"toastWithBreakfast": false,
"breadWithLunch": true
}
```

The JSON Null Data Type:

```
{
"freckleCount": 1,
"hairy": false,
"watchColor": null
}
```

JSON Objects

{"firstName":"John", "lastName":"Doe"}examples

JSON Arrays

```
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName":"Jones"}
]
```

In the example above, the object "employees" is an array containing three objects. Each object is a record of a person (with a first name and a last name).

JSON Uses JavaScript Syntax

With JavaScript you can create an array of objects and assign data to it, like this:

```
var employees = [
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName": "Jones"}
];
```

The first entry in the JavaScript object array can be accessed like this:

Example

```
// returns John Doe
employees[0].firstName + " " + employees[0].lastName;
```

JSON files and javascript syntax

We Will be using collection of JSON data using array & object in JSON format, and save this file with **".js"** extension. We can keep any name here it is **myJSONdata.js**

We will create a JavaScript Function to Display JSON Data using the above created ".js" file and the JSON data inside it.

Now we will create an HTML file (**index.html**) to load JSON data using javascript Function and javascript file we made above.

```
<html><head><script>
// JavaScript Display Function Code start from here
function myFunction(entry) {
var out = "<ul>";
var i;
for(i = 0; i < entry.length; i++) {
out += '<li><a href="' + entry[i].url + '">' + entry[i].display + '</a></li>';
}
out += "</ul>"
document.getElementById("dispayData").innerHTML = out;
}
// JavaScript Display Function Code start from here </script></head>
<body>
```

JSON & Security Concerns

It's only a data interchange format but the real security concerns with JSON arise in the way that it is used. The most common security concerns for JSON on the Web:
1) cross-site request forgery and 2) cross-site scripting.

Cross-Site Request Forgery (CSRF):

- Cross-site request forgery, or CSRF (pronounced sea-surf), is an exploit that takes advantage of a site's trust in a user's Internet browser.
- CSRF vulnerabilities have been around a long time, way before JSON came into existence.
- You sign into a banking website. This website has a JSON URL that includes some sensitive information about you like:

Sensitive information in URL	Descriptioon
[{"user": "bobbarker"}, {"phone": "555-555-5555"}] **This is a valid JSON and a valid javascript as it is missing curly brackets so is dangerous as well.	• So a bad guy can put this dynamic url in script tag and keep in his website like this. <script src="https://www.yourspecialbank.com/user.json"></script> Now user is logged in bank site , and bad guy sends mail similar to familiar bank. You may not observe and click the link . Meanwhile you are in trust with the bank site. Now as soon as you click link in mail you observe you landed to an odd website. By the time your information is with bad guy. •

Cross-Site Request Forgery (CSRF) Protection:

Thhe bank could have turned the array into a value in a JSON object. This
- would make it invalid JavaScript like this.

```
{
“info”:[
          {"user": "bobbarker"},
          {"phone": "555-555-5555"}
        ]
}
```

The bank had only allowed POST requests instead of GET requests to retrieve
- the sensitive JSON, then the bad guy couldn’t have used the link in his URL. If a server allows a GET request for a link, it can be linked to directly in a browser or a script tag. POST however, cannot be linked to directly.

Cross-Site Scripting (XSS)

Cross-site scripting (XSS) attacks are a type of injection attack. A hole in security that often takes place with JSON is at the point where the JavaScript fetches a string of

- JSON and turns it into a JavaScript Object.

- If we want to do anything useful with that textual representation of an object, it needs to be loaded into memory as object. It can then be manipulated, inspected, and used in programming logic.

In JavaScript, one way to do this is by using a function called eval(). The eval() function takes a string, compiles it, and then executes it.

-
- The issue with the eval() function is it takes a string, compiles and executes it without
- discrimination. If my JSON is coming from a third-party server and is replaced with a malicious script, then innocent website will be compiling and executing this malicious code in the Internet browsers of those who visit my site.

As JSON has grown up over the years this vulnerability has been recognized. The JSON.parse() function deals with this vulnerability by being discriminate. This function

- will only parse JSON, and will not execute scripts.

** Example file xss.html

Web API

- A web API is a set of instructions and standards for interacting with a service over HTTP. That interaction can include create, read, update, and delete (CRUD) operations and the web API will have a reference outlining these instructions and standards.

JSON weather data:

{
"dt": 1433383200,
"temp": {"day": 293.5, "min": 293.5, "max": 293.5, "night": 293.5, "eve": 293.5, "morn": 293.5},
"pressure": 1015.06,
"humidity": 98,
"weather": [{"id": 802, "main": "Clouds", "description": "scattered clouds","icon": "03n"}],
"speed": 2.86,
"deg": 134,
"clouds": 44
}

The JSON weather data example can be “read” by any code capable of parsing JSON. This JSON resource can be requested with a URL.

For example, a news web page might include a sidebar with real-time weather data. While you’re reading your news article, code in the back-ground could asynchronously update the weather display every 60 seconds. This operation would not require the page to reload and interrupting any other events.

JSON and Client-Side Frameworks

- In computing, a framework is not an underlying structure. It is a structure that sits on a layer above software or a programming language to provide support to the developers.

- The jQuery JavaScript framework supports JSON in a way that cuts down on production time for requesting and parsing JSON. The jQuery framework is an abstraction tool that caters to manipulation of the Document Object Model (DOM).

- The AngularJS framework is an abstraction tool that caters to single-page applications. Single-page web applications are web pages that break away from the traditional multipage web application experience into a single page.

It provides a framework based on the model-view-controller (MVC) architectural concept.

AngularJS implements the MVC concept as follows:

Model : JavaScript objects are the data model
View : HTML (uses syntax for data binding with the model)
Controller : The JavaScript files that use the AngularJS syntax defining and handling theinteractons with the Model and the view

JSON on the Server Side

- On the server side, there is PHP, ASP.NET, Node.js, Ruby on Rails, Java, Go, and many, many more

- As a web client, we send requests for resources to a server using HTTP. The server responds with a document, such as HTML or JSON. When that document is a JSON document, the server side code must handle the creation of it.

- Here will be seeing example in PHP:

PHP also includes built-in support for serializing and deserializing JSON. PHP refers to

this as encoding and decoding JSON. From the perspective of PHP, JSON is in a coded format. Therefore, to serialize JSON, the “json_encode” function is called, and to deserialize JSON, the “json_decode” function is called.

Serialization is the act of converting the object into text.
Deserialization is the act of the text back into an object.

**** Demo file jsonphp.php**

AJAX using JSON and jQuery :

JSON using jquery Example:

** Demo files are **example.js, example.json and example.html**

JSON Using Ajax:

** Demo file is **jsonajax.html**