CS3215: Web Technology TY Div C n D AY 2022-23

Study Material for Section-I-Part-I- JSON

(**Resource:** *www.w3schools.com*)

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

- JSON is a syntax for storing and exchanging data.
- JSON is **text**, written with JavaScript object notation.
- **JSON is a lightweight data-interchange format**
-
- JSON is "self-describing" and easy to understand
- JSON is language independent

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.**

JSON uses JavaScript syntax, but the JSON format is text only.

Text can be read and used as a data format by any programming language.

Sending Data

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

<script>

```
var myObj = {name: "John", age: 31, city: "New York"};
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

</script>

Receiving Data

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

<script>

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

</script>
```

Storing Data

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.

```
// 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 Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON Data - A Name and a Value

JSON data is written as name/value pairs.

A name/value pair consists of a field name (**in double quotes**), followed by a colon, followed by a value:

```
"name":"Narendra"
```

JSON names require double quotes. JavaScript names don't.

The JSON format is almost identical to JavaScript objects.

In JSON, ***keys* must be strings**, written with double quotes:

```
{"name":"Narendra"}
```

JSON Values

In **JSON**, *values* must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null

In JSON, *string values* must be written with double quotes:

JSON	JavaScript
`{"name":"Narendra"}`	`{name:'Narendra'}`

JSON Uses JavaScript Syntax

Because JSON syntax is derived from JavaScript object notation, very little extra software is needed to work with JSON within JavaScript.

With JavaScript you can create an object and assign data to it, like this:

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

You can access a JavaScript object like this: `person.name;`

It can also be accessed like this: `person["name"];`

Data can be modified like this: `person.name = "Gilbert";`

It can also be modified like this: `person["name"] = "Gilbert";`

JSON Files

- The file type for JSON files is ".json"
- The MIME type for JSON text is "application/json"

JSON vs XML

XML Example	JSON Example

<employees>
  <employee>
  <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
  <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
  <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
	{"employees":[
	  { "firstName":"John", "lastName":"Doe" },
	  { "firstName":"Anna", "lastName":"Smith" },
	  { "firstName":"Peter", "lastName":"Jones" }
	]}

JSON is Like XML Because

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest

JSON is Unlike XML Because

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

The biggest difference is: XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

Why JSON is Better Than XML

XML is much more difficult to parse than JSON. JSON is parsed into a ready-to-use JavaScript object. For AJAX applications, JSON is faster and easier than XML:

Using XML

- Fetch an XML document

- Use the XML DOM to loop through the document
- Extract values and store in variables

Using JSON

- Fetch a JSON string
- JSON.Parse the JSON string

JSON Data Types

In JSON, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- *null*

JSON values **cannot** be one of the following data types:

- a function
- a date
- *undefined*

JSON Strings

Strings in JSON must be written in double quotes. : `{ "name":"John" }`

JSON Numbers

Numbers in JSON must be an integer or a floating point. : `{ "age":30 }`

JSON Objects

Values in JSON can be objects:

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

Objects as values in JSON must follow the same rules as JSON objects.

JSON Arrays

Values in JSON can be arrays.

Example

```
{
"employees":[ "John", "Anna", "Peter" ]
}
```

JSON Booleans

Values in JSON can be true/false : `{ "sale":true }`

JSON null

Values in JSON can be null : `{ "middlename":null }`

JSON.parse()

When receiving data from a web server, the data is always a string.

Parse the data with `JSON.parse()`, and the data becomes a JavaScript object.

Example - Parsing JSON

Imagine we received this text from a web server:

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

Use the JavaScript function `JSON.parse()` to convert text into a JavaScript object:

```
var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');
```

```
<!DOCTYPE html>
<html>
	<body>
		<h2>Create Object from JSON String</h2>
		<p id="demo"></p>
	<script>
		var txt = '{"name":"John", "age":30, "city":"New York"}'
		var obj = JSON.parse(txt);
		document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;
```

```
    </script>
    </body>
</html>
```

JSON From the Server

You can request JSON from the server by using an AJAX request

As long as the response from the server is written in JSON format, you can parse the string into a JavaScript object.

Use the XMLHttpRequest to get data from the server:

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    var myObj = JSON.parse(this.responseText);
    document.getElementById("demo").innerHTML = myObj.name;
  }
};
xmlhttp.open("GET", "json_demo.txt", true);
xmlhttp.send();
```

Array as JSON

When using the `JSON.parse()` on a JSON derived from an array, the method will return a JavaScript array, instead of a JavaScript object.

The JSON returned from the server is an array:

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    var myArr = JSON.parse(this.responseText);
    document.getElementById("demo").innerHTML = myArr[0];
  }
};
xmlhttp.open("GET", "json_demo_array.txt", true);
xmlhttp.send();
```

Parsing Dates

Date objects are not allowed in JSON.

If you need to include a date, write it as a string.

```
var text = '{ "name":"John", "birth":"1986-12-14", "city":"New York"}';
var obj = JSON.parse(text);
obj.birth = new Date(obj.birth);

document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;
```

Parsing Functions

Functions are not allowed in JSON.

If you need to include a function, write it as a string.

```
var text = '{ "name":"John", "age":"function () {return 30;}", "city":"New York"}';
var obj = JSON.parse(text);
obj.age = eval("(" + obj.age + ")");

document.getElementById("demo").innerHTML = obj.name + ", " + obj.age();
```

JSON.stringify()

When sending data to a web server, the data has to be a string.

Convert a JavaScript object into a string with `JSON.stringify()`.

Stringify a JavaScript Object

Imagine we have this object in JavaScript:

```
var obj = { name: "John", age: 30, city: "New York" };
```

Use the JavaScript function `JSON.stringify()` to convert it into a string.

```
var myJSON = JSON.stringify(obj);
```

The result will be a string following the JSON notation.

`myJSON` is now a string, and ready to be sent to a server:

Example

```
var obj = { name: "John", age: 30, city: "New York" };
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
```

Stringify a JavaScript Array

It is also possible to stringify JavaScript arrays:

Imagine we have this array in JavaScript:

```
var arr = [ "John", "Peter", "Sally", "Jane" ];
```

Use the JavaScript function `JSON.stringify()` to convert it into a string.

```
var myJSON = JSON.stringify(arr);
```

The result will be a string following the JSON notation.

`myJSON` is now a string, and ready to be sent to a server:

Example

```
var arr = [ "John", "Peter", "Sally", "Jane" ];
var myJSON = JSON.stringify(arr);
document.getElementById("demo").innerHTML = myJSON;
```

Exceptions

Stringify Dates

In JSON, date objects are not allowed. The `JSON.stringify()` function will convert any dates into strings.

Example

```
var obj = { name: "John", today: new Date(), city : "New York" };
var myJSON = JSON.stringify(obj);

document.getElementById("demo").innerHTML = myJSON;
```

Stringify Functions

In JSON, functions are not allowed as object values.

The `JSON.stringify()` function will remove any functions from a JavaScript object, both the key and the value:

Example

```
var obj = { name: "John", age: function () {return 30;}, city: "New York"};
var myJSON = JSON.stringify(obj);

document.getElementById("demo").innerHTML = myJSON;
```

This can be omitted if you convert your functions into strings before running the `JSON.stringify()` function.

Example

```
var obj = { name: "John", age: function () {return 30;}, city: "New York" };
obj.age = obj.age.toString();
var myJSON = JSON.stringify(obj);

document.getElementById("demo").innerHTML = myJSON;
```

JSON Objects

Object Syntax : `{ "name":"John", "age":30, "car":null }`

JSON objects are surrounded by curly braces {}.

JSON objects are written in key/value pairs.

Keys must be strings, and values must be a valid JSON data type (string, number, object, array, boolean or null).

Keys and values are separated by a colon.

Each key/value pair is separated by a comma.

Accessing Object Values

You can access the object values by using dot (.) notation:

```
myObj = { "name":"John", "age":30, "car":null };
x = myObj.name;
```

You can also access the object values by using bracket ([]) notation:

```
myObj = { "name":"John", "age":30, "car":null };
x = myObj["name"];
```

Looping an Object

You can loop through object properties by using the for-in loop:

```
myObj = { "name":"John", "age":30, "car":null };
for (x in myObj) {
```

```
  document.getElementById("demo").innerHTML += x;
}
```

In a for-in loop, use the bracket notation to access the property *values*:

```
myObj = { "name":"John", "age":30, "car":null };
for (x in myObj) {
  document.getElementById("demo").innerHTML += myObj[x];
}
```

Nested JSON Objects

Values in a JSON object can be another JSON object.

```
myObj = {
  "name":"John",
  "age":30,
  "cars": {
    "car1":"Ford",
    "car2":"BMW",
    "car3":"Fiat"
  }
 }
```

You can access nested JSON objects by using the dot notation or bracket notation:

```
x = myObj.cars.car2;
// or:
x = myObj.cars["car2"];
```

Modify Values

You can use the dot notation to modify any value in a JSON object:

```
myObj.cars.car2 = "Mercedes";
```

You can also use the bracket notation to modify a value in a JSON object:

```
myObj.cars["car2"] = "Mercedes";
```

Delete Object Properties

Use the `delete` keyword to delete properties from a JSON object:

```
delete myObj.cars.car2;
```

JSON Arrays

Arrays as JSON Objects

```
[ "Ford", "BMW", "Fiat" ]
```

Arrays in JSON are almost the same as arrays in JavaScript.

In JSON, array values must be of type string, number, object, array, boolean or *null*.

In JavaScript, array values can be all of the above, plus any other valid JavaScript expression, including functions, dates, and *undefined.*

Arrays in JSON Objects

Arrays can be values of an object property:

```
{
"name":"John",
"age":30,
"cars":[ "Ford", "BMW", "Fiat" ]
}
```

Accessing Array Values

You access the array values by using the index number:

Example

```
x = myObj.cars[0];
```

Looping Through an Array

You can access array values by using a **for-in** loop:

```
for (i in myObj.cars) {
  x += myObj.cars[i];
}
```

Or you can use a `for` loop:

```
for (i = 0; i < myObj.cars.length; i++) {
  x += myObj.cars[i];
}
```

Nested Arrays in JSON Objects

Values in an array can also be another array, or even another JSON object:

```
myObj = {
  "name":"John",
  "age":30,
  "cars": [
    { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
    { "name":"BMW", "models":[ "320", "X3", "X5" ] },
    { "name":"Fiat", "models":[ "500", "Panda" ] }
  ]
 }
```

To access arrays inside arrays, use a for-in loop for each array:

```
for (i in myObj.cars) {
  x += "<h1>" + myObj.cars[i].name + "</h1>";
  for (j in myObj.cars[i].models) {
    x += myObj.cars[i].models[j];
  }
}
```

Modify Array Values

Use the index number to modify an array:

```
 myObj.cars[1] = "Mercedes";
```

Delete Array Items

Use the `delete` keyword to delete items from an array:

```
delete myObj.cars[1];
```

JSON PHP

A common use of JSON is to read data from a web server, and display the data in a web page.

The PHP File

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 file

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

$myJSON = json_encode($myObj);

echo $myJSON;
?>
```

The Client JavaScript

Here is a JavaScript on the client, using an AJAX call to request the PHP file from the example above:

Example

Use JSON.parse() to convert the result into a JavaScript object:

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    var myObj = JSON.parse(this.responseText);
    document.getElementById("demo").innerHTML = myObj.name;
  }
};
xmlhttp.open("GET", "demo_file.php", true);
xmlhttp.send();
```

PHP Database

PHP is a server side programming language, and can be used to access a database.

Imagine you have a database on your server, and you want to send a request to it from the client where you ask for the 10 first rows in a table called "customers".

On the client, make a JSON object that describes the numbers of rows you want to return.

Before you send the request to the server, convert the JSON object into a string and send it as a parameter to the url of the PHP page

Use JSON.stringify() to convert the JavaScript object into JSON:

```
obj = { "limit":10 };
dbParam = JSON.stringify(obj);
xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML = this.responseText;
  }
};
xmlhttp.open("GET", "json_demo_db.php?x=" + dbParam, true);
xmlhttp.send();
```