UNIT – II

JSON

Objective:

- To understand the concepts of JSON

Syllabus:

Introduction,Syntax Rules,JSON vs XML,Data types,Objects,Arrays,Parsing JSON and using stringify function.

Learning Outcomes:

At the end of the unit, students will be able to:

1. Understand the necessity of JSON in web development.
2. Illustrate the objects and arrays.
3. Design the parsing to JSON program
4. Develop JSON programs using stringify functions

Learning Material

JSON

Introduction:

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627.

This tutorial will help you understand JSON and its use within various programming languages such as PHP, PERL, Python, Ruby, Java, etc.

- It was designed for human-readable data interchange.
- It has been extended from the JavaScript scripting language.
- The filename extension is **.json**.
- JSON Internet Media type is **application/json**.
- The Uniform Type Identifier is public.json.

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.

Characteristics of JSON:

- JSON is easy to read and write.
- It is a lightweight text-based interchange format.
- JSON is language independent.

Syntax rules:

The JSON syntax is a subset of the JavaScript syntax.

- Data is in name/value pairs
- Curly braces hold objects
- Square brackets hold arrays
- Begins with { (left brace)
- Ends with } (right brace)
- Each name is followed by : (colon)
- Name/value pairs are separated by , (comma)

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 and followed by a value:

Example

"name":"John"

JSON - Evaluates to JavaScript Objects:

The JSON format is almost identical to JavaScript objects.

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

Example:

```
{ "name":"John" }
```

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 **JavaScript** values can be all of the above, plus any other valid JavaScript expression, including:

- a function
- a date
- undefined

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

```
JSON
{"name":"John"}
```

In JavaScript, you can write string values with double *or* single quotes:

```
<html>
<body>
<p>Access a JavaScript object:</p>
<p id="demo"></p>
```

```
<script>
var myObj, x;
myObj = { name: "John", age: 30, city: "New York" };
x = myObj.name;
document.getElementById("demo").innerHTML = x;
</script>
</body>
    </html>
```

JSON vs XML:

JSON	**XML**
JSON object has a type	XML data is typeless
JSON types: string, number, array, Boolean	All XML data should be string
Data is readily accessible as JSON objects	XML data needs to be parsed.
JSON is supported by most browsers.	Cross-browser XML parsing can be tricky
JSON has no display capabilities.	XML offers the capability to display data because it is a markup language.
JSON supports only text and number data type.	XML support various data types such as number, text, images, charts, graphs, etc. It also provides options for transferring the structure or format of the data with actual data.

Retrieving value is easy	Retrieving value is difficult
Supported by many Ajax toolkit	Not fully supported by Ajax toolkit
A fully automated way of deserializing/serializing JavaScript.	Developers have to write JavaScript code to serialize/de-serialize from XML
Native support for object.	The object has to be express by conventions - mostly missed use of attributes and elements.
It supports only UTF-8 encoding.	It supports various encoding.
It doesn't support comments.	It supports comments.
JSON files are easy to read as compared to XML.	XML documents are relatively more difficult to read and interpret.
It does not provide any support for namespaces.	It supports namespaces.
It is less secured.	It is more secure than 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.

Example

```
{ "name":"John" }
```

JSON Numbers:

Numbers in JSON must be an integer or a floating point.

Example

```
{ "age":30 }
```

JSON Objects:

Values in JSON can be objects.

Example

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

JSON Arrays:

Values in JSON can be arrays.

Example

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

JSON Booleans:

Values in JSON can be true/false.

Example

```
{ "sale":true }
```

JSON null:

Values in JSON can be null.

Example

```
{ "middlename":null }
```

Objects:

Example

```
{ "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:

Example:

```
<html>
<body>
<p>Access a JSON object using dot notation:</p>
<p id="demo"></p>
<script>
var myObj, x;
myObj = {"name":"John", "age":30, "car":null};
x = myObj.name;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Looping an Object:

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

```
<html>
<body>
<p>How to loop through all properties in a JSON object.</p>
<p id="demo"></p>
```

```
<script>
var myObj, x;
myObj = {"name":"John", "age":30, "car":null};
for (x in myObj) {
  document.getElementById("demo").innerHTML += x + "<br>";
}
</script>
</body>
</html>
```

Nested JSON Objects:

Values in a JSON object can be another JSON object.

Example

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

Delete Object Properties:

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

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

Example

```
<html>
<body>
```

```
<p>How to delete properties of a JSON object.</p>

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

<script>
var myObj, i, x = "";
myObj = {
  "name":"John",
  "age":30,
  "cars": {
  "car1":"Ford",
  "car2":"BMW",
  "car3":"Fiat"
  }
}
delete myObj.cars.car2;

for (i in myObj.cars) {
  x += myObj.cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JSON Arrays:

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.*

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

Example:

```
<html>
<body>

<p>Access an array value of a JSON object.</p>

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

<script>
var myObj, x;
myObj = {
  "name":"John",
  "age":30,
  "cars":[ "Ford", "BMW", "Fiat" ]
};
x = myObj.cars[0];
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Parsing JSON:

A common use of JSON is to exchange data to/from a web server.

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"}');

<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>
```

Stringify():

A common use of JSON is to exchange data to/from a web server.

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

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

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);
```

```
<html>
<body>

<h2>Create JSON string from a JavaScript object.</h2>

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

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

</body>
</html>
```

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);
```

Example

```
<html>
<body>

<h2>Create JSON string from a JavaScript array.</h2>

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

<script>

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

</script>

</body>
</html>
```

Stringify Dates:

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

```
<html>
<body>

<h2>JSON.stringify will convert any date objects into strings.</h2>

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

<script>
var obj = { name: "John", today: new Date(), city: "New York" };
```

```
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

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:

```
<html>
<body>

<h2>JSON.stringify will remove any functions from an object.</h2>

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

<script>
var obj = { name: "John", age: function () {return 30;}, city: "New York" };
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```