JSON

Introduction :

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. Conventions used by JSON are known to programmers, which include C, C++, Java, Python, Perl, etc.

- JSON stands for JavaScript Object Notation.
- The format was specified by Douglas Crockford.
- 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.

JSON Grammar / 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 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.
- The implementations of the two structures are represented in the forms of the object and array.
- Crockford outlines the two structural representations of JSON through a series of syntax diagrams.

JSON Values :

- JSON is a subset of JavaScript and does not add anything that the JavaScript language does not possess.
- The values that can be utilized within our JSON structures are represented by types.
- JSON makes use of four primitive types and two structured types.
- A JSON value can only be a representative of string, number, object, array, true, false, and null.
- JSON string must always begin and end with the use of double quotes.
- The numbers grammar does not begin or end with any particular symbolic representation.

Examples of JSON Text Containing a Variety of Valid JSON Values:

- // JSON text of an array with primitives.

[null, true, 8]

- // JSON text of an object with two members.

{ "first": "Ben", "last": "Smith", }

- // JSON text of an array with nested composites.

[{ "abc": "123" }, ["0", 1, 2, 3, 4, 100]]

- //JSON text of an object with nested composites.

{ "object": { "array": [true] } }

JSON Tokens :

In JSON, tokens are the basic building blocks of the syntax. Here's a breakdown of the key JSON tokens:

1. Whitespace :

- Spaces, tabs, and newline characters are ignored and used to format the JSON.

2. Structural Tokens :

- Curly Braces ({ and }): Represent the beginning and end of an object.
- Square Brackets ([and]): Represent the beginning and end of an array.

3. Key/Value Pair Tokens :

- Colon (:): Separates keys from values in an object.

4. Comma (,) :

- Separates key/value pairs in objects and elements in arrays.

5. Data Type Tokens

- Strings: Enclosed in double quotes ("...").
- Numbers: Can be integers or floating-point (e.g., 42, 3.14).
- Boolean: The values true and false.
- Null: The token null.

JSON vs XML :

JSON	XML
1. It is JavaScript Object Notation.	1. It is Extensible markup language.
2. It is based on JavaScript language.	2. It is based on JavaScript language.
3. It is a way of representing objects.	3. It is a markup language and uses tag structure to represent data items.
4. It does not provides any support for namespaces.	4. It supports namespaces.
5. It supports array.	5. It doesn't supports array.
6. Its files are very easy to read as compared to XML.	6. Its documents are comparatively difficult to read and interpret.
7. It doesn't use end tag.	7. It has start and end tags.
8. It is less secured.	8. It is more secured than JSON.
9. It doesn't supports comments.	9. It supports comments.
10. It supports only UTF-8 encoding.	10. It supports various encoding.

Similarity between JSON and XML :

- 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

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*

- string, number, boolean, null are simple datatype or primitive datatypes whereas object and array are referred as complex or structure datatypes.
- 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.
- It is set of name or value pairs inserted between { } (curly braces).
- Example **:** {

- "employee":{ "name":"John", "age":30, "city":"New York" }
- }
  - Objects as values in JSON must follow the same rules as JSON objects.

JSON Arrays :

- It is an ordered collection of values and begins with [(left bracket) and ends with] (right bracket).
- The values of array are separated by ,(comma).
- Example **:** {

"employees":["John", "Anna", "Peter"]

}

JSON Booleans :

- This datatype can be either true/false.
- Example **:** { "sale":true }

JSON null :

- It is just a define nullable value.
- Example **:** { "middlename":null }

Objects :

JSON objects can be created with JavaScript. Let us see the various ways of creating JSON objects using JavaScript:

- Creation of an empty Object:

```
var JSONObj = {};
```

- Creation of a new Object:

```
var JSONObj = new Object();
```

- Creation of an object with attribute bookname with value in string, attribute price with numeric value. Attribute is accessed by using '.' Operator:

```
var JSONObj = { "bookname ":"VB BLACK BOOK", "price":500 };
```

This is an example that shows creation of an object in javascript using JSON, save the below code as **json_object.html**:

```
<html>
<head>
<title>Creating Object JSON with JavaScript</title>
<script language="javascript" >
        var JSONObj = { "name" : "tutorialspoint.com", "year"  : 2005 };
        document.write("<h1>JSON with JavaScript example</h1>");
        document.write("<br>");
        document.write("<h3>Website Name="+JSONObj.name+"</h3>");
        document.write("<h3>Year="+JSONObj.year+"</h3>");
</script>
</head>
<body>
</body>
</html>
```

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*.

Key Features of JSON Arrays

- Ordered: The values in a JSON array maintain their order, which means the sequence in which you define them matters.
- Heterogeneous: A JSON array can contain multiple data types, including strings, numbers, objects, arrays, booleans, and null.
- Comma-Separated: Elements within the array are separated by commas, but there should be no trailing comma after the last element.

Array Objects :

In JSON, an array of objects is a powerful way to represent a collection of items, where each item has multiple attributes. Each item in the array is a JSON object, allowing for a structured representation of complex data.

Syntax of Array of Objects

An array of objects is defined using square brackets [], and each object within the array is enclosed in curly braces {}. Objects are composed of key/value pairs.

Example :

```
var books = { "Pascal" : [
      { "Name"  : "Pascal Made Simple", "price" : 700 },
      { "Name"  : "Guide to Pascal", "price" : 400 }
   ],
   "Scala"  : [
      { "Name"  : "Scala for the Impatient", "price" : 1000 },
      { "Name"  : "Scala in Depth", "price" : 1300 }
   ]
}
```

JSON Schema :

JSON Schema is a specification for JSON based format for defining the structure of JSON data. It was written under IETF draft which expired in 2011. 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 Validation Libraries :

There are several validators currently available for different programming languages. Currently the most complete and compliant JSON Schema validator available is JSV.

JSON Schema Example :

```
{
"$schema": "http://json-schema.org/draft-04/schema#",
"title": "Product",
"description": "A product from Acme's catalog", "type": "object",
"properties": { "id": {
"description": "The unique identifier for a product", "type": "integer"
},
"name": {
"description": "Name of the product", "type": "string"
},
"price": {
"type": "number", "minimum": 0, "exclusiveMinimum": true
}
},
"required": ["id", "name", "price"]
}
```

Various important keywords :

- $schema : The $schema keyword states that this schema is written according to the draft v4 specification.
- title : You will use this to give a title to your schema.
- description : A little description of the schema.
- type : The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
- properties : Defines various keys and their value types, minimum and maximum values to be used in JSON file.

Parsing JSON :

- Parsing is the process of analyzing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar.
- As the grammar of JSON is a subset of JavaScript, the analysis of its tokens by the parser occurs indifferently from how the Engine parses source code. Because of this, the data produced from the analysis of the JSON grammar will be that of objects, arrays, strings, and numbers.
- Additionally, the three literals true, false, and null are produced as well.

JSON.parse

- In a nutshell, JSON.parse converts serialized JSON into usable JavaScript values.
- Syntax of the JSON.parse() Method :

```
JSON.parse(text [, reviver]);
```

- JSON.parse can accept two parameters, text and reviver.
- The name of the parameter text is indicative of the value it expects to receive.
- The parameter reviver is used similarly to the replacer parameter of stringify, in that it offers the ability for custom logic to be supplied for necessary parsing that would otherwise not be possible by default.
- The parameter text implies the JavaScript value, which should be supplied.
- This is a rather important aspect, because any invalid argument will automatically result in a parse error.
- Eg : Invalid JSON Grammar Throws a Syntax Error :

```
var str = JSON.parse( "abc123" ); //SyntaxError: JSON.parse: unexpectedcharacter
```

- Throws an error because it was provided a string literal and not serialized JSON.
- Therefore, "abc123" must be escaped and wrapped with an additional set of quotation marks.
- Valid JSON Grammer Is Successfully Parsed

```
var str = JSON.parse( "\"abc123\"" ); //valid JSON string value      console.log(str) //abc123;

console.log(typeof str) //string;
```

JSON 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.
- We can 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.
- Syntax of the JSON stringify Method JSON.stringify(value[, replacer [, space]]);
- The value parameter of the stringify method is the only required parameter of the three outlined by the signature. The argument supplied to the method represents the JavaScript value intended to be serialized. This can be that of any object, primitive, or even a composite of the two.

- The value parameter of the stringify method is the only required parameter of the three outlined by the signature. The argument supplied to the method represents the JavaScript value intended to be serialized. This can be that of any object, primitive, or even a composite of the two.
- The optional replacer parameter is either a function that alters the way objects and arrays are stringified or an array of strings and numbers that acts as a white list for selecting the object properties that will be stringified.
- The third parameter, space, is also optional and allows you to specify the amount of padding that separates each value from one another within the produced JSON text. This padding provides an added layer of readability to the produced string.

Code:

```
<html>
<head>
<title>JSON programs </title>
</head>
<body>
<script type="text/javascript"> var data={
"Bname":"JSON",
"Publisher": "TataMcgraw", "author": "Smith", "price":250,
"ISBN":"1256897912345"
};
document.writeln(JSON.stringify(data));
document.writeln(JSON.stringify(data,["Bname","author","price"]));
document.write(JSON.stringify(data,["Bname","author","price"],5));
</script>
</body>
</html>
```

Persistenting JSON :

- Over the years, many a developer has needed to be able to string together the isolated requests of a common server, in order to facilitate things such as shopping carts for e-commerce.
- One of the technologies that was forged from this requirement brought forth a technique that we will leverage in order to achieve the persistence of JSON. That technology is the HTTP cookie.
- The HTTP cookie, or cookie for short, was created as a means to string together the actions taken by the user per “isolated” request and provide a convenient way to persist the state of one page into that of another.
- The cookie is simply a chunk of data that the browser has been notified to retain. Furthermore, the browser will have to supply, per subsequent request, the retained cookie to the server for the domain that set it, thereby providing state to a stateless protocol.
- The cookie can be utilized on the client side of an application with JavaScript.

- Additionally, it is available to the server, supplied within the header of each request made by the browser.
- The header can be parsed for any cookies and made available to server-side code. Cookies provide both front-end and back-end technologies the ability to collaborate and reflect the captured state, in order to properly handle each page view or request accordingly.
- The ability to continue to progress the state from one page to another allows each action to no
- longer be isolated and, instead, occur within the entirety of the user's interaction with a web site.
- Syntax:
- The cookie is simply a string of ASCII encoded characters composed of one or more attribute-value pairs, separated by a semicolon (;) token.
- Key/Value Pairs Intended to Be Persisted As a Cookie Must Both Be Valid ASCII Characters.

Data Interchange :

Hypertext Transfer Protocol :

- The Hypertext Transfer Protocol, or simply HTTP, is the underlying mechanism responsible for our daily interactions with the Internet.
- It is used in conjunction with many underlying networks of protocols, in order to facilitate the appropriate request/response between a client and a server.

HTTP-Request :

- In the preceding analogy, the HTTP protocol is the waiter, the order is the HTTP request, and the food provided represents the HTTP response.
- The HTTP request consists of three general components, each with a particular use for detailing what resource is required from a server.

Request Headers :

- The second category of headers is that of the request headers.
- These headers can be supplied with the request to provide the server with preferential information that will assist in the request.
- They outline the configurations of the client making the request.
- Such headers may reveal information about the user-agent making the request or the preferred data type that the response should provide.
- The request headers are the most commonly configured headers.
- One very useful header is the Accept header.
- It can be used to inform the server as to what MIME type or data type the client can properly handle.
- This can often be set to a particular MIME type, such as application/json, or text/plain.
- It can even be set to */*, which informs the server that the client can accept all MIME types.
- The response provided by the server is expected to reflect one of the MIME types the client can handle.

The following are request headers:

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Expect
- From
- Host
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- Proxy-Authorization
- Range
- Referer
- TE
- User-Agent

Response Headers :

- The second category of headers is the response headers.
- These headers provide the client of the request with information pertaining to the configurations of the server, as well as the requested URI.
- For example, the server can provide response headers to inform the request of what HTTP methods are accepted, as well as whether authorization is required in order to access the specified URI.
- These headers can even inform the request whether it should occur at a later point in time.

The following are response headers:

- Accept-Ranges
- Age
- ETag
- Location
- Proxy-Authentication
- Retry-After
- Server
- Vary
- WWW-Authenticate

XMLHttpRequest Interface :

- The XMLHttpRequest object, consists of a variety of methods, event handlers, properties, and states, all of which provide our JavaScript application the ability to successfully facilitate an HTTP request, in addition to obtaining the response from a server.
- Creating an Instance of the XMLHttpRequest Object

```
var xhr = new XMLHttpRequest();
```

- The XMLHttpRequest Level 2 standard outlines the event handlers possessed by each xhr instance, so that we may remain aware of the status of the request.

These event handlers can be viewed as :

The xhr Event Handlers for Monitoring the Progress of the HTTP Request :

Event Handlers	Event Handler Event Type
onloadstart *	loadstart *
onprogress	progress
onload	load
onloadend *	loadended *
onerror	error
ontimeout	timeout
onabort *	abort *
onreadystatechange	readystatechange

JSON PHP :

- A common use of JSON is to read data from a web server, and display the data in a web page. We can exchange JSON data between the client and a PHP server. PHP has some built-in functions to handle JSON.
- Objects in PHP can be converted into JSON by using the PHP function **json_encode()**
- For Example:

```
<?php

$myObj->name = "John";

$myObj->age = 30;

$myObj->city = "New York";

$myJSON = json_encode($myObj);

echo $myJSON;

?>
```

- PHP's **json_decode()** function takes a JSON string and converts it into a PHP variable.
- Typically, theJSON data will represent a JavaScript array or object literal which json_decode will convert into a PHP array or object.
- The following two examples demonstrate, first with an array, then with an object:
- Example 1:

```
$json = '["apple","orange","banana","strawberry"]';

$ar = json_decode($json);

// access first element of $ar array

echo $ar[0]; // apple
```

- Example 2:

```
$json = '{
"title": "JavaScript: The Definitive Guide",
"author": "David Flanagan",
"edition": 6
}';
$book = json_decode($json);
// access title of $book object
echo $book->title; // JavaScript: The Definitive Guide
```

JSON Python :

Encoding JSON in Python (encode):

- Python encode() function encodes the Python object into a JSON string representation.
- **Syntax**

demjson.encode(self, obj, nest_level=0)

- **Example**

```
import demjson

data = [ { 'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4, 'e' : 5 } ]

json = demjson.encode(data)

print json
```

Decoding JSON in Python (decode):

- Python can use demjson.decode() function for decoding JSON. This function returns the value decoded from json to an appropriate Python type.
- **Syntax**

demjson.decode(self, txt)

- **Example**

```
import demjson

json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

text = demjson.decode(json)

print text
```

JSONP :

- JSONP is a method for sending JSON data without worrying about cross-domain issues.
- JSONP does not use the XMLHttpRequest object.
- JSONP uses the <script> tag instead.
- JSONP stands for JSON with Padding.
- Requesting a file from another domain can cause problems, due to cross-domain policy.
- Requesting an external script from another domain does not have this problem.
- JSONP uses this advantage, and request files using the script tag instead of the XMLHttpRequest object.

<script src="demo_jsonp.php">

- The file on the server wraps the result inside a function call:

```
<?php
$myJSON = '{ "name":"John", "age":30, "city":"New York" }';

echo "myFunc(".$myJSON.");";
?>
```

- The function named "myFunc" is located on the client, and ready to handle JSON data:

```
function myFunc(myObj) {
  document.getElementById("demo").innerHTML = myObj.name;
}
```