UNIT-1: JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is "self-describing" and easy to understand

• A JSON object is an unordered set of name/value pairs

• A JSON 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)

Why Use JSON?

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

Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.

JavaScript has a built in function for converting JSON strings into JavaScript objects:

JSON.parse()

JavaScript also has a built in function for converting an object into a JSON string:

JSON.stringify()

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.

JSON Files

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

Characteristics of JSON

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

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name(key)/value pairs
- Data is separated by commas
- Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma). **[Key:Value]**
- Square brackets hold arrays and values are separated by ,(comma).

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.

Example:1

var Obj= {“marks”:97}

console.log(Obj) ----------> {marks: 97}

console.log(Obj.marks) ----------> 97

Example:2

var Obj= {“name”:"PKP",”marks”:97,”dist”:true}

console.log(Obj) ----------> {name: 'PKP', marks: 97, dist: true}

console.log(Obj.name) ----------> PKP

console.log(Obj.marks) ----------> 97

console.log(Obj.dist) ----------> true

Example of a JSON string: '{"name":"ABC", "age":20}' - It defines an JSON string with 2 properties: Name and age

Example of a JSON object: { "name":"ABC", "age":20 } OR { name:"ABC", age:20 } - It defines an JSON object with 2 properties: Name and age

JSON - DATATYPES

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":"LJU"}

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
var obj= {
"employee":{"name":"LJU", "age":30, "city":"Ahmedabad"}
}

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}

Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.

The following example shows how to use JSON to store information related to books based on their topic and edition.

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

Parse & Stringify

JSON.parse()

JavaScript has a built in function for converting JSON strings into JavaScript objects:

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.

We received this text from a web server: '{"var1":"LJ","var2":"University"}'

Use the JavaScript function JSON.parse() to convert it into an object.

Example:

```
<script>
let obj = JSON.parse('{"var1":"LJ","var2":"University"}');
console.log(obj);
console.log(obj.var1 + " " + obj.var2);
</script>
```

Output:

```
{var1: 'LJ', var2: 'University'}

LJ University
```

JSON.stringify()

JavaScript has a built in function for converting objects into JavaScript strings:

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

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

Example:

```
<script>
let obj = JSON.stringify({var1: 'LJ', var2: 'University'});
console.log(obj);
</script>
```

Output:

```
'{"var1":"LJ","var2":"University"}'
```

One of the most common uses for JSON is when using an API, both in requests and responses.

JSON.parse(string) takes a string of valid JSON and returns a JavaScript object.

The inverse of this function is JSON.stringify(object) which takes a JavaScript object and returns a string of JSON, which can then be transmitted in an API request or response.

Example of an array of objects

```
var cars = [{
   color: 'gray',
   model: '1',
   nOfDoors: 4
   },
   {
   color: 'yellow',
   model: '2',
   nOfDoors: 4
}];
```

var jsonCars = JSON.stringify(cars);

```
"[{"color":"gray","model":"1","nOfDoors":4},{"color":"yellow","model":"2","nOfDoors":4}]"
```

Program:

```
<html>
<script>
const user='{"name":["ABC","PKP","XYZ"],"age":"28"}'; //value is in String formation
console.log(user)  //Print string formation -----> {"name":["ABC","PKP","XYZ"],"age":"28"}
console.log(user.name) // Undefined

obj=JSON.parse(user) //convert string data into objects
console.log(obj) // obje is object using parse --->{name: Array(3), age: '28'}
console.log(obj.name) // ---> Detail of name --->(3) ['ABC', 'PKP', 'XYZ']
obj4=JSON.stringify(obj) // convert into string again
console.log(obj4) // ---------->{"name":["ABC","PKP","XYZ"],"age":"28"}
document.write(obj.name) // To wite in Chrome
</script>
</html>
```

JSON vs XML

```
{"students":[
  { "firstName":"A", "lastName":"B" },
  { "firstName":"C", "lastName":"D" },
  { "firstName":"E", "lastName":"F" }
]}
```

```
<students>
  <student>
    <firstName>A</firstName> <lastName>B</lastName>
  </ student >
  < student >
    <firstName>C</firstName> <lastName>D</lastName>
  </ student >
  < student >
    <firstName>E</firstName> <lastName>F</lastName>
  </ student >
</ student >
```

JSON = XML Because

- Both JSON and XML are "self describing" means human readable
- Both JSON and XML are hierarchical means 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 != XML Because

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays
- XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

XML is much more difficult to parse than JSON. JSON is parsed into a ready-to-use JavaScript object. So JSON is Better Than XML.

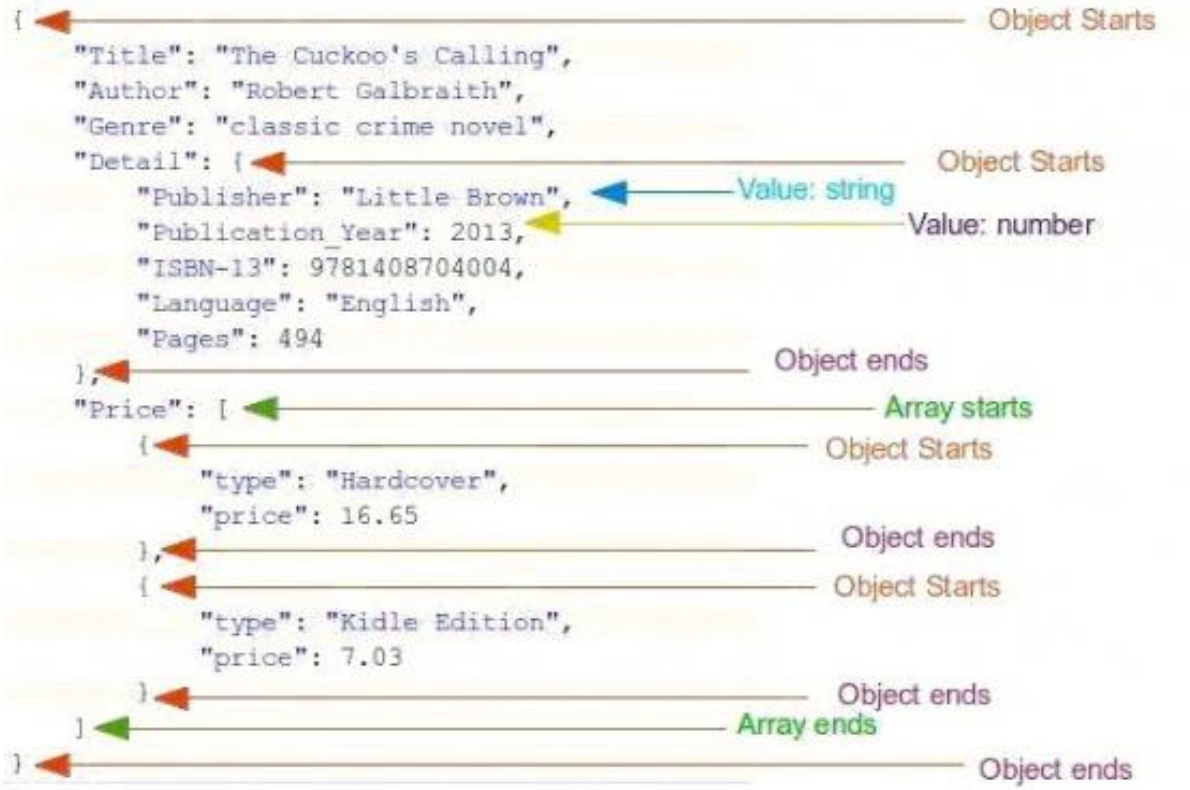
Using XML

- 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 ---→Parse the JSON string and extract values

JSON Data Example



Examples

- ## Simple JSON object

```
const user={
   "name":"abc",
   "age":"28",
   "course":["FSD-1","DE","FSD-2"],
   "adress":["T1","t2",{"t3":"Give again"}]
}
console.log(user.name[0])
console.log(user.age)
console.log(user.course)
console.log(user.adress[1])
console.log(user.adress[2])
console.log(user.adress[2].t3)
```

Output

a
28
(3) ['FSD-1', 'DE', 'FSD-2']
t2
{t3: 'Give again'}
Give again

- ## Example on Nested JSON

```
<html>
<script>
const sub={
 "FSD":[
        {   "Topic":"HTML",
        "course":"Beginer",
         "content":{"tags":"table","form":"xyz"}
        },
        { "Topic":"CSS",
         "course":"Beginer",
         "content":["tags","table","form"]}
         ]
         };
console.log(sub.FSD[0].content.tags);
console.log(sub.FSD[1].content[0]);
</script>
</html>
```

Output:
table
tags

- **From below JSON object fetch the values as asked.**

```
<html>
<body>
   <script >
    const a= {
      "Datastructures":
      [
         {
            "Name": "tree",
           "course":"Intro",
           "content":["1","B","C"]
         },
         {
           "Name": "tree1",
           "course":"Intro1",
           "content":["1","B","C","d"]
         }
      ],
      "xyz":
      {
         "Name":"Graphics",
         "Topic":["BFS","CDF","Sorting"],
      }
    }
     console.log(a.Datastructures[1].Name); //tree1
     console.log(a.Datastructures[0].Name); //tree
     console.log(a.xyz.Name);//Graphics
     console.log(a.xyz.Topic); //['BFS', 'CDF', 'Sorting']
     console.log(a.xyz.Topic[0]); //BFS
     console.log(a.Datastructures[1]); //   {  "Name": "tree1",  "course":"Intro1",
"content":["1","B","C","d"]  }
     console.log(a.Name); //undefined
     console.log(a.xyz); //{Name: 'Graphics', Topic: Array(3)}
     </script>
</body>
</html>
```

Consider below JSON object and fetch values using for loop (for…in/for…of).

```
const sub = {
"FSD": [ { "Topic": "HTML", "course": "Beginer", "content": ["tags", "table", "form"], },
          { "Topic": "CSS", "course": "Beginer", "content": ["tags", "table", "form"] }
        ] };
```

Solution

```
<script>
const sub =
{ "FSD": [ {
"Topic": "HTML",
"course": "Beginer", "content": ["tags", "table", "form"], },
{
"Topic": "CSS",
"course": "Beginer",
"content": ["tags", "table", "form"]
} ] };
document.write("// Using for…in loop <br>")
// Using for…in loop
for (x in sub.FSD) {
for (y in sub.FSD[x]) {
document.write(sub.FSD[x][y] + "<br>")
} }
document.write("// Using for…of loop <br>")
// Using for…of loop
for(x1 of sub.FSD) {
document.write(x1.Topic + "<br>")
document.write(x1.course + "<br>")
document.write(x1.content + "<br>")
console.log(x)
}
</script>
```

Output:

```
// Using for…in loop
HTML
Beginer
tags,table,form
CSS
Beginer
tags,table,form
// Using for…of loop
HTML
Beginer
tags,table,form
CSS
Beginer
tags,table,form
```

New date object Task

Write one JSON string with date property (yyyy-mm-dd) and print date in India standard time.

```
<script>
const obj=JSON.parse(`{"name":"xyz","age":"17","dob":"1997-03-14"}`);
console.log(obj.dob);
a = new Date(obj.dob);
console.log(a);
</script>
```

Output

Fri Mar 14 1997 05:30:00 GMT+0530 (India Standard Time)

Write one JSON string with date property (yyyy-mm-dd) and print date in India standard time While clicking on button..

```
<html>
<script>
function call()
{
   text='{"Name":"ABC","Age":18,"City":"Ahmedabad","DOB":"1990-08-24"}';
   obj=JSON.parse(text);
   console.log(obj.DOB);
   obj.DOB=new Date(obj.DOB)
   document.getElementById("demo").innerHTML= "Birth day of "+ obj.Name +" is "+ obj.DOB;
}
</script>
<body>
   <P id="demo">  </P>
   <button onclick="call()"> Click Me </button>
</body>
</html>
```

Output:

Birth day of ABC is Fri Aug 24 1990 05:30:00 GMT+0530 (India Standard Time)

Click Me

Date Objects are directly not allowed in JSON. If you need to include date, write it as string.

- **Write a function 'fromListToObject' which takes in an array of arrays, and returns an object with each pair of elements in the array as a key-value pair.**

Example input:
[['make', 'Ford'], ['model', 'Mustang'], ['year', 1964]]

Function's return value (output):
{
make : 'Ford'
model : 'Mustang',
year : 1964
}

Do not change the input string. Assume that all elements in the array will be of type 'string'.

Note that the input may have a different number of elements than the given sample.
For instance, if the input had 6 values instead of 4, your code should flexibly accommodate that.

Starter Code:
function fromListToObject(array) {
// your code here
}

Solution:

```
var array = [['make', 'Ford'], ['model', 'Mustang'], ['year', 1964]]
function fromListToObject(array) {
emptyObject = {}
for(var i = 0; i < array.length; i++){
var newArray = array[i];
emptyObject[newArray[0]] = newArray[1];
}
return emptyObject;
}
var object = fromListToObject(array);
console.log(object);
```

Output

{make: 'Ford', model: 'Mustang', year: 1964}

- **Write a function 'transformFirstAndLast' that takes in an array, and returns an object with:**
  1) the first element of the array as the object's key, and
  2) the last element of the array as that key's value.

Example input:
['Queen', 'Elizabeth', 'Of Hearts', 'Beyonce']
Function's return value (output):
{Queen : 'Beyonce'}
Do not change the input array. Assume all elements in the input array will be of type 'string'.
Note that the input array may have a varying number of elements. Your code should flexibly accommodate that.

E.g. it should handle input like:
['Kevin', 'Bacon', 'Love', 'Hart', 'Costner', 'Spacey']
Output:

```
function transformFirstAndLast(array) {
var object = {};
object[array[0]] = array[array.length-1];
return object;
}
var arrayList = ['Queen', 'Elizabeth', 'Of Hearts', 'Beyonce'];
//var arrayList2 = ['Kevin', 'Bacon', 'Love', 'Hart', 'Costner', 'Spacey']

console.log(transformFirstAndLast(arrayList));
//console.log(transformFirstAndLast(arrayList2));
```

Solution :

{Queen: ‘Beyonce’}
//{Kevin:’Spacey’}

- **Write a JS to store an array of objects having name and age. display name and age of person with highest age using for loop.**

```
const person =
  [
    {
      name:
      "PQR", age:
      38
    },
    {
      name:
      "ABC", age:
      35
    },
    {
      name:
      "XYZ", age:
      47
    }
  ]
```

Solution:

```
<script>
 let person = [
 { name: "PQR", age: 10
 8 },
 { name: "ABC", age: 55 },
 { name: "XYZ", age: 47 }
 ];
 let oldestPerson = person[0];
 for (let i = 1; i < person.length; i++) {
   if (person[i].age > oldestPerson.age) {
     oldestPerson = person[i];
   }
 }
 console.log(oldestPerson)

 </script>
```

Output:

{ name: 'XYZ', age: 47 }

- **Write a script to define two JSON objects named as “division1” and “division2” having an array to store names of students. These names should be sorted alphabetically in the object and should be written on console. At last, both division objects should be visible with names sorted alphabetically on console.**

```
var test = {
   "division1":
   {
      "name":["Z","B","H"]
   },
   "division2": {
      "name" :["Y","A","G"]
   }
}
const div1_data = test.division1.name;
const div2_data = test.division2.name;

var combine_data = div1_data.concat(div2_data).sort();

console.log(combine_data);
```

Output:

['A', 'B', 'G', 'H', 'Y', 'Z']

- **Write an array of 3 objects. Each object contains 2 properties name and height. Now, sort an array values by height in descending order. Print name in terminal as per the sorted height.**

```
const student =
  [
    {
      name:
      "ABC",
      height: 5.7
    },
    {
      name:
      "PQR",
      height: 6.0
    },
    {
      name:
      "XYZ",
      height: 6.2
    }
  ]

  for (let x = 0; x < student.length; x++){
    for (let y = 0; y < student.length; y++){

      if (student[x].height > student[y].height) {
        var temp;
        temp = student[y];
        student[y] = student[x];
        student[x] = temp;
      }}}
  for (let i = 0; i < student.length; i++){
    console.log(student[i].name, student[i].height);
  }
```

```
// Sort the array by height in descending order – Alternative Approch
const sortedStudents = student.sort((a, b) => b.height - a.height);

// Print name and height of each student
for (const person of sortedStudents) {
  console.log(`${person.name}: ${person.height}`);
}
```

Output

XYZ 6.2

PQR 6

ABC 5.7

- **Get JSON object values using for loop and make HTML table to display these values.**

```
const sub =
{

  "FSD": [
  {
   "Topic": "HTML",
   "course": "Beginer",
   "content": ["tags", "table", "form"],
  },
  {
   "Topic": "CSS",
   "course": "Beginer",
   "content": ["tags", "table", "form"]
  }
  ]
 };
var temp = "<table border='2px'>";
temp += "<tr>";
for (x in sub.FSD[0]) {
 temp += "<th>" + x + "</th>";
}
temp += "</tr>"; for
(x in sub.FSD) {
console.log(x); temp
+= "<tr>";
 for (y in sub.FSD[x]) {

  temp += "<td>" + sub.FSD[x][y] + "</td>";
 }
 temp += "</tr>";
}
temp += "</table>";
//console.log(temp);
document.write(temp);
```

Topic	course	content
HTML	Beginer	tags,table,form
CSS	Beginer	tags,table,form

Write a code to fetch the values of keys and print a sentence as "Hi! We are students of semester 4 of CSE branch" in h1 tag on browser.

```
<script>
const a = {
"A" : "LJU",
"B" : ["CSE", "IT" , "CE"],
"C" : [
{
"D" : "Hi",
"E" : ['are', 4, {'F' : ['semester', 'We']} ]
}
],
"G" : {
"H" : "students",
"I" : ["of","!"]
},
"J" : [{"K" :"Python", "L":"branch"},"FSD-2"]
}
document.write(a.C[0].D, a.G.I[1] + " " + a.C[0].E[2].F[1] +" " + a.C[0].E[0] + " " +a.G.H + "
" +a.G.I[0] + " "+ a.C[0].E[2].F[0] + " "+ a.C[0].E[1] + " " +a.G.I[0] + " "+ a.B[0] + " "+
a.J[0].L)
</script>
```

Task

- **Print following statement in console from given JSON objects.**

```
 const a = {"Name" : "Ramesh",
"Subects" : ["Maths", "Scence" , "chemistry"],
"Grade" : {"Type" : "marks","Total" : [88,90,99,87]},
"Range" : {"opt" : "100", "type" : ["secure","subject","class"]},
"achive" : [{"Rank" :"rank", "place":[1,2,3]}, {"Ordinalindicator":"st"},"12"],
"joints" : ['outof','got','and']
}
```

Output: Ramesh got 99 outof 100 marks
and secure 1st rank.

- **Make a statement : "John has BMW- X3 at the age of 30" in web browser**

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