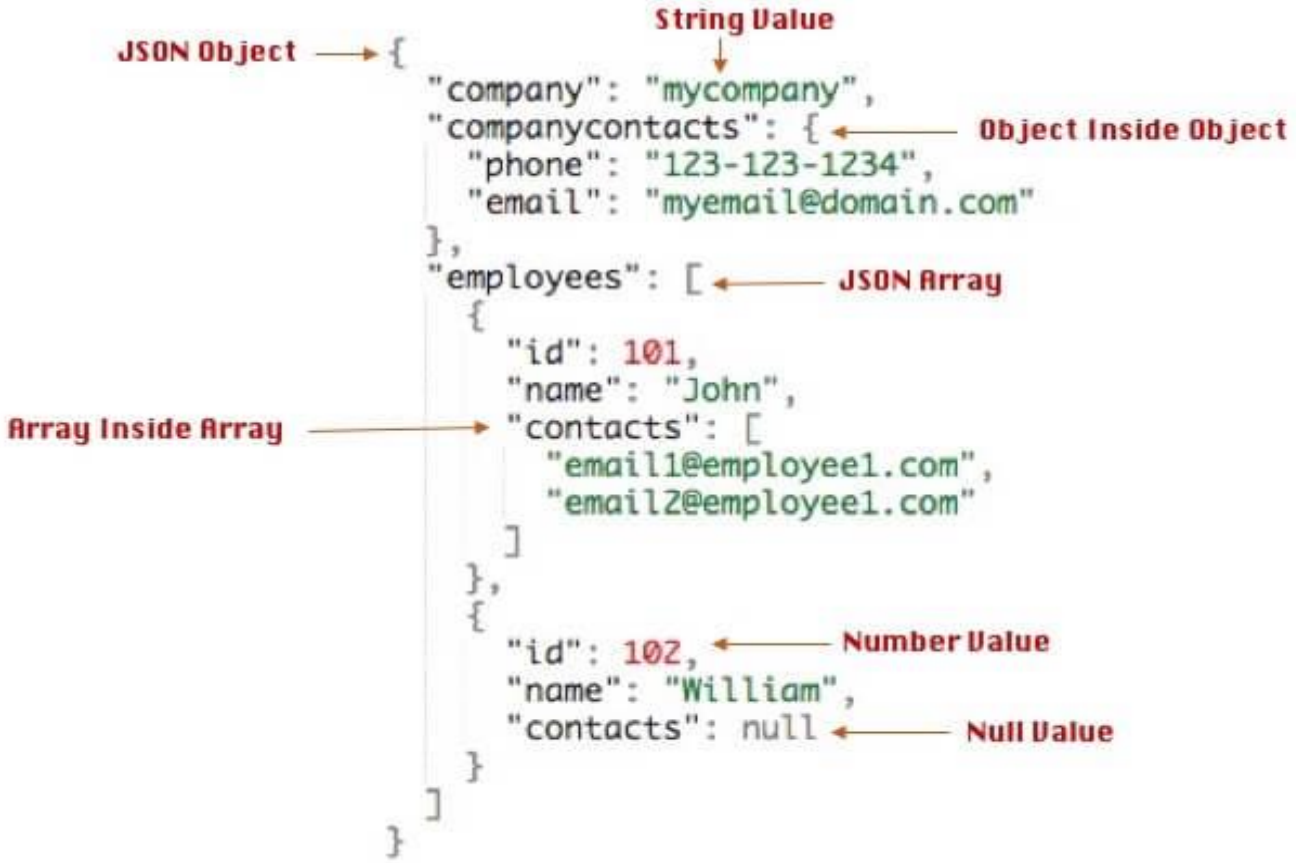
JSON (Java Script Object Notation)

- ✓ JSON stands for **J**ava**S**cript **O**bject **N**otation
- ✓ It is a **text format** for storing and transporting data
- ✓ It is "self-describing" and easy to understand
- ✓ It is a lightweight data-interchange format
- ✓ It is plain text written in JavaScript object notation
- ✓ It is language independent
- ✓ 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.

JSON - Syntax

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



This example is a JSON string:

```
'{"name":"ABC", "age":20}'
```

It defines an JSON string with 2 properties: name and age

This example is a JSON object:

```
{
"name":"ABC",
"age":20
}

OR
{
name:"ABC",
age:20
}
```

It defines an JSON object with 2 properties: name and age

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":"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

```
{
"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}

JSON
USING JS FILE

j1.js

```
var a= {
  "name": "Test",        // String
  "age": 25,             // Number
  "ispass": false,     // Boolean
  "address": {          // Object (Nested JSON)
    "city": "Ahmedabad",
    "zip": "380015"
  },
  "subjects": ["Math", "Science"],  // Array
  "year": null   // Null
}

console.log(a);
console.log(a.age)
console.log(a['address'])
console.log(a['address']['city'])
```

Output:

```
{
  name: 'Test',
  age: 25,
  ispass: false,
  address: { city: 'Ahmedabad', zip: '380015' },
  subjects: [ 'Math', 'Science' ],
  year: null
}
25
{ city: 'Ahmedabad', zip: '380015' }
Ahmedabad
```

To run enter command node j1.js in terminal

Using HTML FILE

J1.html

```
<script>
var a= {
  "name": "Test",
   “age”:20
}
Console.log(a.name)
</script>
```

Using JSON FILE

Create a json file named j1.json

```
{
  "name": "Test",
   “age”:20
}
```

1.js file

```
const data = require("./j1.json");  // Import JSON file
console.log(data.name);

To run enter command node 1.js in terminal
```

JavaScript built-in functions

- ✓ JavaScript has a built in function for converting JSON strings into JavaScript objects:**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.

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()

- ✓ When sending data to a web server, the data has to be a string.
- ✓ Convert a JavaScript object into a string with **JSON.stringify().**

we have this object in JavaScript:

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

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

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.

Below is a simple example of an array of objects–

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

Example:

```
var cars = [{
   color: 'gray',
   model: '1',
   nOfDoors: 4
   },
   {
   color: 'yellow',
   model: '2',
   nOfDoors: 4
}];
var jsonCars = JSON.stringify(cars);
console.log(jsonCars[0].model)
console.log(cars[0].model)
```

Output:

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

JSON vs XML

JSON Example

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

XML Example

```
<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 are hierarchical means values within values
- ✓ Both can be parsed and used by lots of programming languages
- ✓ Both 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

Examples

Write simple JSON object and fetch the values of each properties.

```
<script>
  const a = {
    'Name': 'tree',
    'course': 'Intro',
    'content': ['1', 'B', 'C']
  }
  console.log(a.Name);
  console.log(a.course);
  console.log(a.content);
  console.log(a.content[1]);
</script>
```

Output:

```
tree
Intro
(3) ['1', 'B', 'C']
0:'1'
1:'B'
2:'C'
B
```

Write code to fetch the "Give again" value from user object.

```
<script>
const user = {
        "name": ["ABC", "DEF", "GHI"],
        "age": "28",
        "course": ["FSD-1", "DE", "FSD-2"],
```

```
    "address": ["T1", "T2", { "t3": "Give again" }]
  }
  console.log(user.address[2].t3)
 // console.log(user['address'][2]['t3'])
</script>
```

Output

Give again

From JSON object fetch the values as asked.

```
<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>
```

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>
```

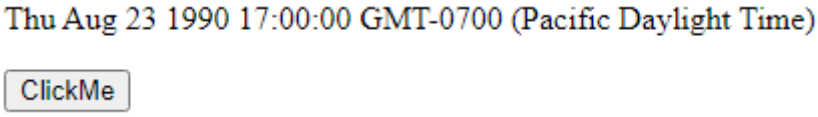
Output

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()
{
 obj=JSON.parse('{"Name":"ABC","Age":18,"City":"Ahmedabad","DOB":"1990-08-24"}');
  obj=new Date(obj.DOB);
 // document.write(obj.DOB);
  document.getElementById("demo").innerHTML=obj
}
</script>
<body>
  <P id="demo">  </P>
  <button onclick="call()"> ClickMe </button>
</body>
</html>
```

Output:



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

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

```
<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
```

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

```
const student =
  [
    {
      name: "NAS",
      age: 25
    },
    {
      name: "ABC",
      age: 30
    },
    {
      name: "XYZ",
      age: 33
    }
  ]
  for (let x = 0; x < student.length; x++){
    for (let y = 0; y < student.length; y++){

      if (student[x].age > student[y].age) {
        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].age);
  }
```

Output:

XYZ 33

ABC 30

NAS 25

OR

```
const student =
   [
      {
         name: "PQR",
         age: 25
      },
      {
         name: "ABC",
         age: 30
      },
      {
         name: "XYZ",
         age: 33
      }
   ]
const a=student.sort((a,b)=>b.age-a.age)

for (x of a){
   console.log(x.name +" "+x.age)
}
```

Output:

XYZ 33
ABC 30
PQR 25

Write a script to define two JSON objects named as “division1” and “division2” having an array to store name of students. These names should be sorted alphabetically and should be displayed in 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 a function named "firstlast" that takes an array and returns an object with below conditions.

1) **The first element of an array must be an object's key**
2) **The last element of an array must be a key's value.**

Input = ['abc','def','ghi','jkl']
Output = { abc: 'jkl' }

```
function firstlast(a) {
   var temp = {};
   temp[a[0]] = a[a.length - 1];
   return temp;
}

var data = ['abc', 'def', 'ghi', 'jkl'];
console.log(firstlast(data));
```

Output:
{ abc: 'jkl' }

Explaination:
a**[0]** → Accesses the **first element** of the array and treats it as a **key**.
a**[a.length - 1]** → Accesses the **last element** of the array and assigns it as a **value**.
t**emp[a[0]] = a[a.length - 1];** → Dynamically assigns the key-value pair in the temp object.

Write a JS to store an array of objects having height and name. Display highest height.

```
const person =
   [
      {
         name: "PQR",
         height: 5.2
      },
      {
         name: "ABC",
         height: 5.5
      },
      {
```

```
        name: "XYZ",
        height: 6.0
    }
  ]
for (let x = 0; x < person.length; x++){
    var t = 0;
    for (let y = 0; y < person.length; y++){

        if (person[x]. height > person[y].height) {
            t++;
        }
    }
    if (t == person.length-1) {
        var t1;
        t1 = person[x];
        console.log(t1.height);
    }
}
```

Output:

6

OR

```
const person = [
    {
        name: "PQR",
        height: 5.2
    },
    {
        name: "ABC",
        height: 5.5
    },
    {
        name: "XYZ",
        height: 6.0
    }]
console.log(Math.max(...person.map(obj => obj.height)));
```

Output:

6

Explanation

- ✓ **person.map(obj => obj.height)**
- ✓ .map() creates a **new array** containing only the height values. **Result:** [5.2, 5.5, 6.0]
- ✓ **Math.max(...array) :** The **spread operator (...)** spreads the array into individual values.
- ✓ Math.max(5.2, 5.5, 6.0) returns **6.0** (the highest value.

OR

```
const person = [
      {
         name: "PQR",
         height: 5.2
      },
      {
         name: "ABC",
         height: 5.5
      },
      {
         name: "XYZ",
         height: 6.0
      }]
const max_height =person.sort((a,b)=>
b.height-a.height)[0]
console.log(max_height.height)
```

Output:

6

Explanation

- ✓ .sort((a, b) => b.height - a.height) sorts the array in descending order based on the height property.
- ✓ **The first element ([0]) is the tallest person. It returns { name: "XYZ", height: 6.0 }**
- ✓ So, **max_height.height** extracts and prints height which is 6.0.

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

```
<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>
```

Output:

Hi! We are students of semester 4 of CSE branch

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

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

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

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

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