JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa). You'll come across it quite often, so in this article we give you all you need to work with JSON using JavaScript, including parsing JSON so you can access data within it, and creating JSON.

JavaScript For Loop

Loops can execute a block of code a number of times.

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

Different Kinds of Loops

JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- for/of - loops through the values of an iterable object
- while - loops through a block of code while a specified condition is true
- do/while - also loops through a block of code while a specified condition is true

The For Loop

The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

Example

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

From the example above, you can read:

Statement 1 sets a variable before the loop starts (let i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

Statement 1

Normally you will use statement 1 to initialize the variable used in the loop (let i = 0).

This is not always the case, JavaScript doesn't care. Statement 1 is optional.

You can initiate many values in statement 1 (separated by comma):

Example

```
for (let i = 0, len = cars.length, text = ""; i < len; i++) {
  text += cars[i] + "<br>";
}
```

And you can omit statement 1 (like when your values are set before the loop starts):

Example

```
let i = 2;
let len = cars.length;
let text = "";
```

```
for (; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Statement 2

Often statement 2 is used to evaluate the condition of the initial variable.

This is not always the case, JavaScript doesn't care. Statement 2 is also optional.

If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.

If you omit statement 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Statement 3

Often statement 3 increments the value of the initial variable.

This is not always the case, JavaScript doesn't care, and statement 3 is optional.

Statement 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Statement 3 can also be omitted (like when you increment your values inside the loop):

Example

```
let i = 0;
let len = cars.length;
let text = "";
for (; i < len; ) {
  text += cars[i] + "<br>";
  i++;
}
```

The For In Loop

The JavaScript for in statement loops through the properties of an Object:

Syntax

```
for (key in object) {
  // code block to be executed
}
```

Example

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";
for (let x in person) {
  text += person[x];
}
```

Example Explained

- The **for in** loop iterates over a **person** object
- Each iteration returns a **key** (x)
- The key is used to access the **value** of the key
- The value of the key is **person[x]**

For In Over Arrays

The JavaScript for in statement can also loop over the properties of an Array:

Syntax

```
for (variable in array) {
  code
}
```

Example

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
for (let x in numbers) {
  txt += numbers[x];
}
```

Do not use **for in** over an Array if the index **order** is important.

The index order is implementation-dependent, and array values may not be accessed in the order you expect.

It is better to use a **for** loop, a **for of** loop, or **Array.forEach()** when the order is important.

Array.forEach()

The forEach() method calls a function (a callback function) once for each array element.

Example

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt += value;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

The example above uses only the value parameter. It can be rewritten to:

Example

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);

function myFunction(value) {
  txt += value;
}
```

The For Of Loop

The JavaScript for of statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

Syntax

```
for (variable of iterable) {
  // code block to be executed
}
```

variable - For every iteration the value of the next property is assigned to the variable. *Variable* can be declared with const, let, or var.

iterable - An object that has iterable properties.

For/of was added to JavaScript in 2015 (ES6)

Safari 7 was the first browser to support for of:

Chrome 38	Edge 12	Firefox 51	Safari 7	Opera 25
Oct 2014	Jul 2015	Oct 2016	Oct 2013	Oct 2014

For/of is not supported in Internet Explorer.

Looping over an Array

Example

```
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x;
}
```

Looping over a String

Example

```
let language = "JavaScript";

let text = "";
for (let x of language) {
text += x;
}
```

MY RESUME JSON FORMAT:-

```
{
          "resume": {
           “profile”: [
        {
           "name": "K.NARESHBABU",
           "email": "bestforevermvi@gmail.com",
           "phone": “91+9159553177”,
           "location": {
             "address": "kallapdi viilage and post kallapadi",
             "pincode": "632601”,
             "city": "gudiyatham",
             "district": "vellore",
           }
        ]
        },
          "work": [
           {
             "company": "lanson toyota",
             "position": "technician",
             "startDate": "2017-06-01",
             "endDate": "2019-12-14",
           }
          ],
          "education": [
           {
             “board”: “anna university”,
             "courses": “mechanical engineering”
             "institution": "s.k.p engineering collage",
             "area": "thiruvannamalai",
             "startDate": "2013-06-01",
             "endDate": "2017-04-30",
             "cgpa": "6.6",

           }
          ],
          "languages": [
           {
             "language": "english and tamil",
             "fluency": "English and tamil"
           }
```

```
    ],
    "interests": [
      {
        "name": "sports",
        "keywords": “basketball, table tennis, volleyball, and wrestling”

      }
    ]
  }
```

Difference between window, screen, and document in JavaScript:

Window:

The JavaScript **window object** sits at the top of the JavaScript Object hierarchy and represents the browser window. The window object is supported by all browsers. All global **JavaScript objects** , functions, and variables automatically become members of the window object. The window is the first thing that gets loaded into the **browser** . This window object has the majority of the properties like length, innerWidth, innerHeight, name, if it has been closed, its parents, and more.

The window object represents the current **browsing context** . It holds things like window.location, window.history, window.screen, window.status, or the **window.document** . Each browser tab has its own top-level window object. Each of these windows gets its own separate global object. window.window always refers to window, but **window.parent** and window.top might refer to enclosing windows, giving access to other execution contexts. Moreover, the window property of a window object points to the window object itself. So the following statements all return the same window object:

```
window.window
window.window.window
window.window.window.window
and so on
```

Window Properties:

Window object has two properties to determine the size of the browser window. They are:

window.innerHeight **:** gives the inner height of the browser window (in pixels)

window.innerWidth : gives the inner width of the browser window (in pixels)

Window methods:

Some window object methods are:

window.open() : open a new window

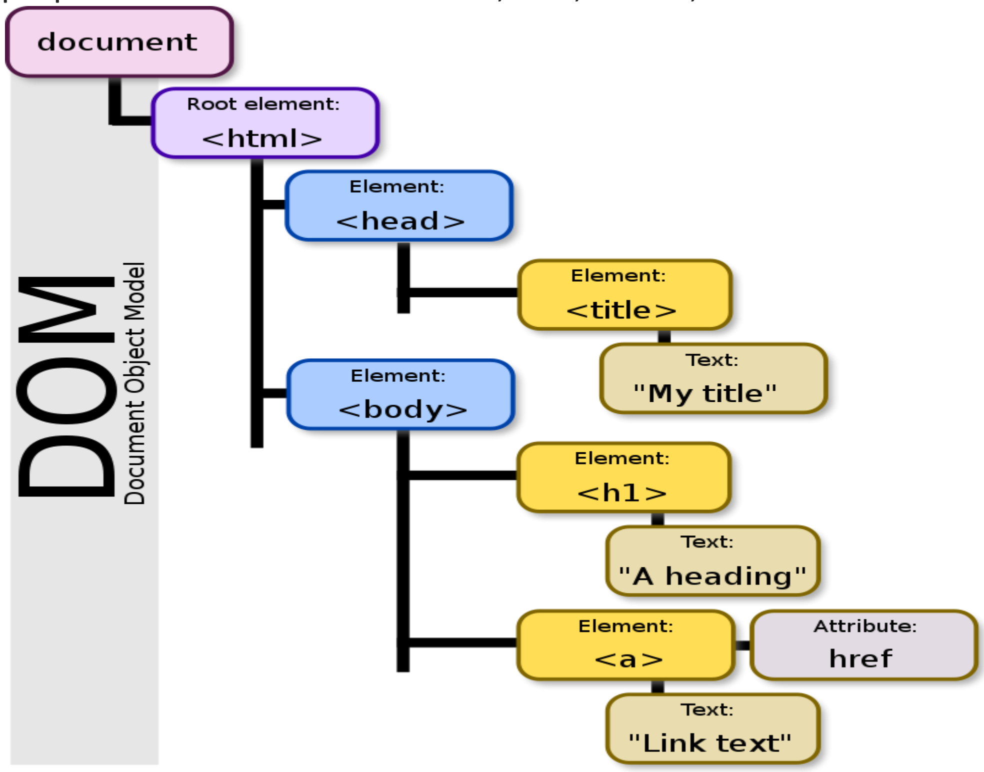
window.close() : close the current window

window.moveTo() : move the current window

window.resizeTo() : resize the current window

Document:

The **Document object** represents any web page loaded in the browser and serves as an entry point into the web page's content, which is the DOM tree. When an HTML document is loaded into a **web browser** , it becomes a document object. It is the root node of the HTML document. The document actually gets loaded inside the window object and has properties available to it like title, URL, cookie, etc.



Finding HTML elements:

We can find the HTML elements by using the below **document object** methods:

document.getElementById(id) : Find and return an element by element id

document.getElementsByTagName(name) : Find and return an element by *tag name*

document.getElementsByClassName(name) : Find return an element by class name

Changing HTML elements:

We can change the HTML element contents like style, text, attribute using the below properties:

element.innerHTML = new html content **:** Change the inner HTML of an element

element.attribute = new value **:** Change the attribute value of an HTML element

element.style.property = new style **:** Change the style of an HTML element

Adding and Deleting HTML elements:

We can create, add, delete and replace HTML elements by using the below methods:

document.createElement(*element*) **:** Create an HTML element

document.removeChild(*element*) **:** Remove an HTML element

document.appendChild(*element*) **:** Add an HTML element

document.replaceChild(*new, old*) **:** Replace an HTML element

document.write(**text*****)* :** Write into the HTML output stream

Adding Event Handlers:

We can also add event handlers when a specific event occurs like onclick, onload, onkeydown etc by using the respective event property:

document.getElementById(id).onclick = function(){code} :*Adding event handler code to an onclick event*

Screen:

Screen is a small information object about physical **screen dimensions** of the user device. It can be used to display screen width, height, colorDepth, pixelDepth etc. It is not mandatory to write **window prefix** with screen object like **window.screen**. It can be written without window prefix.

Properties:

screen.width
screen.height
screen.availWidth
screen.availHeight
screen.colorDepth
screen.pixelDepth