

# BOM

There are no official standards for the Browser Object Model (BOM).

Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

- window.innerWidth
- window.innerHeight
- window.open()
- window.close()
- window.moveTo()
- window.resizeTo()
- 
- screen.width
- screen.height
- screen.availWidth
- screen.availHeight
- screen.colorDepth
- screen.pixelDepth
- 
- window.location.href
- window.location.hostname
- window.location.pathname
- window.location.protocol
- window.location.assign()
- 
- history.back()
- history.forward()

- 
- navigator.appName
- navigator.appCodeName
- navigator.platform
- navigator.onLine
- navigator.javaEnabled()
- navigator.language
- navigator.userAgent
- 
- window.confirm()

## BOM -> setTimeout() and setInterval()

setTimeout(function, milliseconds)

Executes a function, after waiting a specified number of milliseconds.

setInterval(function, milliseconds)

Same as setTimeout(), but repeats the execution of the function continuously.

The setTimeout() and setInterval() are both methods of the HTML DOM Window object.

Example :

```
<button onclick="setTimeout(myFunction, 3000)">Try it</button>
```

```
<button onclick="clearTimeout(myVar)">Stop it</button>
```

```
*****
```

```
setInterval(myTimer, 1000);   clearInterval()
```

**practice** : clock , cornometer , webpack-sess47/index33.clock2.html , slider => webpack-sess50/index5-sliderman.html

# Cookies

Cookies let you store user information in web pages.

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

- `document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";`

## practice

- webpack-sess43/index2-accor2.html
- webpack-sess43/index4-todo2.html
- webpack-sess46/index16-loading.html
- webpack-sess47/index27-keyboardsample.html
- webpack-sess47/index28-increment.html
- webpack-sess48/index1-writetxt.html
- webpack-sess48/index6-captcha3.html
- webpack-sess50/index8-tab.html
- webpack-sess52/index1-card.html
- webpack-sess53/index1-window-resize.html

## plugins

- owl
- carosel
- slick
- hijri
- webpack-sess49

extra file => webpack-sess54\webpack-sess54

## Web Storage API

The Web Storage API is a simple syntax for storing and retrieving data in the browser. It is very easy to use:

```
localStorage.setItem("name", "John Doe");
```

```
document.getElementById("demo").innerHTML = localStorage.getItem("name");
```

```
localStorage.removeItem("mytime");
```

```
localStorage.clear()
```

```
sessionStorage.clear();
```

```
localStorage.length;
```

```
sessionStorage.setItem("lastname", "Smith");  
  
document.getElementById("result").innerHTML = sessionStorage.getItem("lastname");
```

# Web Workers API

A web worker is a JavaScript running in the background, without affecting the performance of the page.

When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

```
<p>Count numbers: <output id="result"></output></p>  
<button onclick="startWorker()">Start Worker</button>  
<button onclick="stopWorker()">Stop Worker</button>  
  
<script>  
let w;  
  
function startWorker() {  
  if(typeof(w) == "undefined") {  
    w = new Worker("demo_workers.js");  
  }  
  
  w.onmessage = function(event) {  
    document.getElementById("result").innerHTML = event.data;  
  }  
}
```

```
};  
}  
function stopWorker() {  
    w.terminate();  
    w = undefined;  
}  
</script>
```

# Regular Expressions

A regular expression is a sequence of characters that forms a search pattern.

The search pattern can be used for text search and text replace operations.

```
/pattern/modifiers;
```

```
let text = "Visit para!";
```

```
let n = text.search("para");
```

```
.....
```

```
let text = "Visit para";
```

```
let n = text.search(/Para/i);
```

```
.....
```

```
[abc]
```

[^abc]

[0-9]

[^0-9]

(x|y)

.....

```
let text = "Visit para!";
```

```
let result = text.replace("para", "training");
```

.....

```
let text = "Visit para!";
```

```
let result = text.replace(/para /i, " training ");
```

.....

```
const pattern = /e/;
```

```
pattern.test("The best things in life are free!");
```

.....

```
/e/.exec("The best things in life are free!");
```

.....

```
let text = "The rain in SPAIN stays mainly in the plain";
```

```
let result = text.match(/ain/g);
```

.....

```
let text = "Give 100%!";
```

```
let result = text.match(/\\d/g);
```

.....

```
let text = "Is this all there is?";
```

```
let result = text.match(/\s/g);
```

.....

```
let text = "HELLO, LOOK AT YOU!";
```

```
let result = text.search(/\bLO/);
```

.....

```
let text = "Hellooo World! Hello W3Schools!";
```

```
let result = text.match(/o+/g);           =>          000,0,0,00
```

.....

```
let text = "Hellooo World! Hello W3Schools!";
```

```
let result = text.match(/lo*/g);          =>          l,looo,l,l,lo,l
```

.....

```
let text = "1, 100 or 1000?";
```

```
let result = text.match(/10?/g);          =>          1,10,10
```



# Throw and Try to Catch

```
try {  
    adddler("Welcome guest!");  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.message;  
} finally {  
    ****  
}
```

## Use Strict

"use strict"; Defines that JavaScript code should be executed in "strict mode".

You can use strict mode in all your programs. It helps you to write cleaner code, like preventing you from using undeclared variables.

Strict mode is declared by adding "use strict"; to the beginning of a script or a function.

```
"use strict";  
x = 3.14;
```

Deleting a variable (or object) is not allowed.

```
"use strict";  
let x = 3.14;  
delete x;
```

Deleting a function is not allowed.

```
"use strict";  
function x(p1, p2) {};  
delete x;
```

Duplicating a parameter name is not allowed:

```
"use strict";  
function x(p1, p1) {};
```

Octal numeric literals and Octal escape characters are not allowed:

```
"use strict";  
let x = 010;
```

Writing to a read-only property is not allowed:

```
"use strict";  
const obj = {};  
Object.defineProperty(obj, "x", {value:0, writable:false});  
obj.x = 3.14;
```

Writing to a get-only property is not allowed:

```
"use strict";  
const obj = {get x() {return 0} };  
obj.x = 3.14;
```

Deleting an undeletable property is not allowed:

```
"use strict";  
delete Object.prototype;
```

The word eval and arguments cannot be used as a variable:

```
"use strict";  
let eval = 3.14;
```

For security reasons, eval() is not allowed to create variables in the scope from which it was called:

```
"use strict";  
eval ("let x = 2");  
alert (x);
```

# JSON

JSON is a format for storing and transporting data.

JSON is often used when data is sent from a server to a web page.

JSON stands for **JavaScript Object Notation**

JSON is a lightweight data interchange format

JSON is language independent \*

```
{  
  "employees": [
```

```
{ "firstName": "John", "lastName": "Doe" },  
{ "firstName": "Anna", "lastName": "Smith" },  
{ "firstName": "Peter", "lastName": "Jones" }  
]  
}
```

### JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON objects => { "firstName": "John", "lastName": "Doe" }

JSON arrays =>

```
"employees": [  
  { "firstName": "John", "lastName": "Doe" },  
  { "firstName": "Anna", "lastName": "Smith" },  
  { "firstName": "Peter", "lastName": "Jones" }  
]
```

Then, use the JavaScript built-in function `JSON.parse()` to convert the string into a JavaScript object:

```
let text = '{"employees":[' +
```

```
'{"firstName":"John","lastName":"Doe" },' +  
'{"firstName":"Anna","lastName":"Smith" },' +  
'{"firstName":"Peter","lastName":"Jones" }'}';
```

```
const obj = JSON.parse(text);  
document.getElementById("demo").innerHTML =  
obj.employees[1].firstName + " " + obj.employees[1].lastName;      =>      Anna Smith
```

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

```
const myJSON = JSON.stringify(obj);
```

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

```
const obj = {name: "John", age: 30, city: "New York"};  
const myJSON = JSON.stringify(obj);  
document.getElementById("demo").innerHTML = myJSON;
```

# AJAX

The keystone of AJAX is the XMLHttpRequest object.

1. Create an XMLHttpRequest object
2. Define a callback function
3. Open the XMLHttpRequest object
4. Send a Request to a server

```
<div id="demo">
```

```
<button type="button" onclick="loadDoc()">Change Content</button>
```

```
</div>
```

```
<script>
```

```
trainingsitedesign.ir
```

@parsa\_ghorbanian\_web

09194723906

```
function loadDoc() {  
    const xhttp = new XMLHttpRequest();  
    xhttp.onload = function() {  
        document.getElementById("demo").innerHTML = this.responseText;  
    }  
    xhttp.open("GET", "ajax_info.txt");  
    xhttp.send();  
}  
</script>
```

**you can...**  
**continue to work!**

\55