

**TEMA 5:**  
**ECMAScript, DOM, NAVEGADOR**  
**EI1042 - TECNOLOGÍAS Y APLICACIONES WEB**  
**EI1036- TECNOLOGÍAS WEB PARA LOS SISTEMAS DE**  
**INFORMACIÓN**  
**(2018/2019)**

*Professora: Dra. Dolores M<sup>a</sup> Llidó Escrivá*

**Universitat Jaume I.**

## TABLA DE CONTENIDOS

1. Introducción EMACScript:
  - ScriptEngine
2. JavaScript.
  - Objetos
  - Variables
  - Tipos de datos
  - Funciones
3. DOM .
4. Componentes de los Navegadores.
5. CSSOM, RenderTree.
6. JavaScript Navegador. BOM.
  - Window
  - DocumentElement
  - This
7. Gestión Eventos: Event Target.
8. Template.
9. Iframe.
10. JSON.
11. Generadores.

# 1. INTRODUCCIÓN EMACSCRIPT

- El lenguaje **JavaScript** se creó inicialmente para los navegadores web.
- En 1997 JavaScript adoptado como estándar de la European Computer Manufacturers Association ECMA.
- Evolucionó en un lenguaje con muchos usos y plataformas: **EMACScript**.
- Una **plataforma** puede ser un cliente web (Chrome) o un servidor web (Node.js) u otro anfitrión.
- El **Script Engine** es el programa que ejecuta código en un dialecto del lenguaje de script (JavaScript) proporciona además del núcleo del lenguaje, los objetos y funciones específicos de la plataforma. Cada navegador tiene su propia implementación (webkit->safari, v8->Chrome, Node.js).

## EVOLUCIÓN ECMAScript

Year	Name	Description
1997	ECMAScript_1	First Edition.
1998	ECMAScript_2	Editorial changes only.
1999	ECMAScript_3	Added Regular Expressions, Try/catch.
	<b>ECMAScript_4</b>	<b>Was never released.</b>
2009	ECMAScript_5	Added "strict mode",JSON support.
2011	ECMAScript_5.1	Editorial changes.
2015	ECMAScript_6	Added classes and modules. Promises
2016	ECMAScript_7	Async/Aqu

## DIALECTOS/API

- JavaScript, JScript y ActionScript son un dialectos de ECMAScript.
- Los **dialectos** incluyen extensiones al lenguaje y **APIs**.
- Una aplicación escrita en un dialecto puede ser incompatible con otra, si no utilizan un subconjunto común de características **APIs compatibles**

# JAVASCRIPT

- Lenguaje interpretado, dialecto de ECMAScript.
- La sintaxis es semejante al C pero las variables no están tipificadas y no distingue entre mayúsculas y minúsculas.
- No existe un cuerpo principal del programa (main), todo lo que no esté dentro de una función es ejecutado.
- Una función debe ser declarada antes de usarse.
- En JavaScript todo son objetos.

## EDITORES EN LÍNEA:

<http://jsfiddle.net/>

<https://codepen.io/idesi/pen/rLgaJO>

## OBJETOS

- Una lista es en realidad una clase de objeto.
- Un objeto de JavaScript es un array asociativo formado por las propiedades y los métodos.
- Objetos/Interfaces predefinidos en el núcleo de JavaScript
  - Date: Un objeto genérico conteniendo una fecha.
  - Math: Una biblioteca incorporada de funciones matemáticas y constantes. `var pi = Math.pi`
  - Console: Es una interface para comunicarnos por código con la consola.



# CREACIÓN DE OBJETOS

- Instancia directa formato breve.

```
personObj = {nombre: "Carlos Sempere", apodo: "Doe", eda
```

- Instancia directa con creando objeto "vacío".

```
var Autor = new Object();  
Autor.nombre = "Carlos Sempere";  
Autor.apodo = "Doe" ;  
Autor.edad = 50;  
//objeto 'Autor' con atributos
```

- Con un constructor.

```
function Persona(nombre, apodo, edad) {  
    this.nombre=nombre;  
    this.apodo=apodo;  
    this.age=edad;  
}  
var Autor= new Persona("Anas Sempere", "Doe", 50);
```

## DECLARACIÓN VARIABLES /AMBITOS

- Var: permite definir una variable **local** en una función o **global** si está fuera de una función.
- Let: Declara una variable local en un bloque de ámbito. Si está dentro de {}, este es su ámbito
- Const: Declara una constante de solo lectura en un ámbito.

## EJEMPLO JS

```
var z; // sin inicializarla
var x = 42
y = 42 // sin declarar variable
let y = 13
const PI = 3.14;
let arr = [ 3 , 5 , 7 ];
arr.foo = "hola";

for (let i in arr) {
    console.log(i); // logs "0", "1", "2", "foo"
}

for (let i of arr) {
    console.log(i); // logs "3", "5", "7"
}
```

# TIPOS DE DATOS

- Number: `var x = 5;`
- String: `var cad = '32';`
- Boolean: `var b1 = x>0;`
- Object: `var dog = { name : 'Spot', breed : 'Dalmatian' };`
- Function: `function apellido (separador) {}`
- Array: `var lista = new Array(); var frutas = ['Manzana', 'Banana'];`
- Date: `var fin = Date.now();`
- RegExp: `var re = /\w+/;`
- null: `foo=null // definida no inicializada`
- Undefined: `typeof(y) //no definida no inicializada`

## ARRAY

- Uso de un array:

```
var colors = ["red", "green", "blue"];
```

- Crear un array vacio y luego poner los elementos:

```
var colors = new Array();  
colors[0] = "red"; colors[2] = "blue";  
colors[1]="green";
```

- Crear un array con sus elementos:

```
var colors = new Array(3,2,1,0);
```

## FUNCIONES

```
function NombreFuncion (parametro1, ..., parámetro N )  
{...  
    return valor;  
}
```

- Algunas Funciones Predefinidas:
  - Tipo de una variable: *typeof(variable)*;
  - Evalúa un código JavaScript : *eval("Primera(p1, p2)")*

# FUNCIONES

Las funciones son un objeto por ello podemos:

- asignar funciones a variables, y referenciarla utilizando la variable
- pasar funciones como parámetros a otras funciones
- obtener funciones como resultados de la ejecución de la función.

```
function grado() {  
  function titulo(name) {  
    return "Dr. " + name; }  
  return titulo; //una funcion!  
}  
var phd = grado();  
phd("Turing"); //Dr Turing
```

## EJEMPLOS JAVASCRIPT(1) TRY-THROW-CATH

```
function getMonthName(mo) {  
    mo = mo - 1 ; // Adjust month number for array index (1  
    var months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul"  
    "Aug", "Sep", "Oct", "Nov", "Dec"];  
    if (months[mo]) { return months[mo]; }  
    else { throw "InvalidMonth"; // Lanzamos una excepción.  
}  
myMonth=5;  
try { // statements to try  
    monthName = getMonthName(myMonth); // function could th  
}catch (e) {  
    monthName = "unknown";  
    console.log(e); // pass exception object to error han  
}
```



## EJEMPLOS JAVASCRIPT(2)

```
function Primera(p1, p2) {  
    p1(p2);}  
// Función Literal  
var Segunda = function (m1) {  
    console.log(m1 / 10 );  
};  
// Pasando la Función Segunda como parámetro de Primera  
Primera(Segunda, 20 );
```

¿Que aparecerá devuelve la llamada: Primera(Segunda, 20 );

# FUNCIÓN ANÓNIMA

```
function (parametro1, ..., parametro N )  
{ ...  
return valor;  
}
```

¿Cómo la ejecutamos si no la podemos llamar?

- Poniendo la definición de la función anónima entre paréntesis (), y pasándole parámetros

```
/* funcion anónima sin parámetros*/  
(function() { alert("hola mundo") })()  
/* funcion anonima como un parametro*/  
(function(quien) {alert("hola "+quien)})( "mundo" )
```

- Con return

```
return function(quien) {alert("hola "+quien)} ( 'mundo' )
```

## EJEMPLOS JAVASCRIPT(3)

```
var f1=function(x,y)
{
    var s=x+y;
    return s;
}
console.log(f1( 4 , 6 ));
var f2=f1;
console.log(f2( 3 , 3 ));
```

¿Que aparecerá en la pantalla?

# JAVASCRIPT EN EL NAVEGADOR

- *window* : OBJETO GLOBAL que contiene todas las variables.
- *window.document* o *document* : objeto que tiene el documento html cargado en el navegador, o sea el DOM del documento HTML.

El código JS, se puede poner:

- Como valor de un atributo que es un evento. (No usar)

```
<button onclick="return handleClick(event, this);">
```

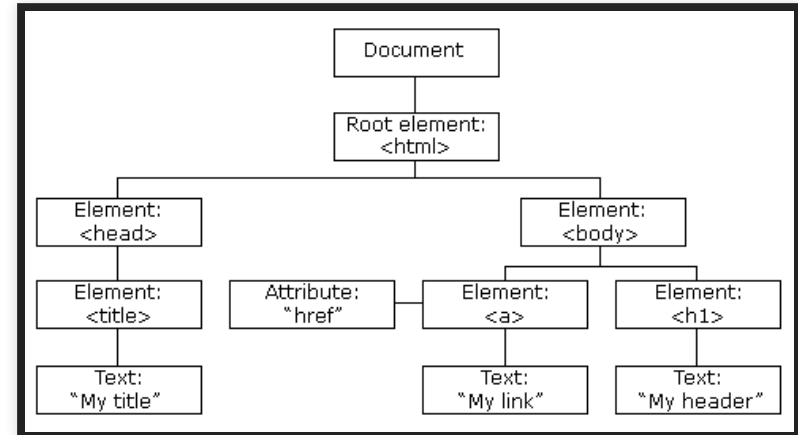
- Dentro de la etiqueta *script* tanto en la cabecera como en el body.

```
<script > console.log( 'hola' );</script>
```

- En un fichero externo

```
<script src="url"></script>
```

# DOM (DOCUMENT OBJECT MODEL)



- El Modelo de Objetos del Documento
- Interfaz de lenguaje neutral independiente de plataforma que permite a los programas y scripts el acceso dinámico y la actualización del contenido, la estructura o estilo de documentos.
- Proporciona una representación estructurada del documento como un árbol de nodos(document, head, frame, body, p, div) los cuales contienen propiedades, métodos y eventos.
- Tipos de DOM : ( 1. Core DOM, 2. XML DOM, 3. HTML DOM)

## ORIGEN

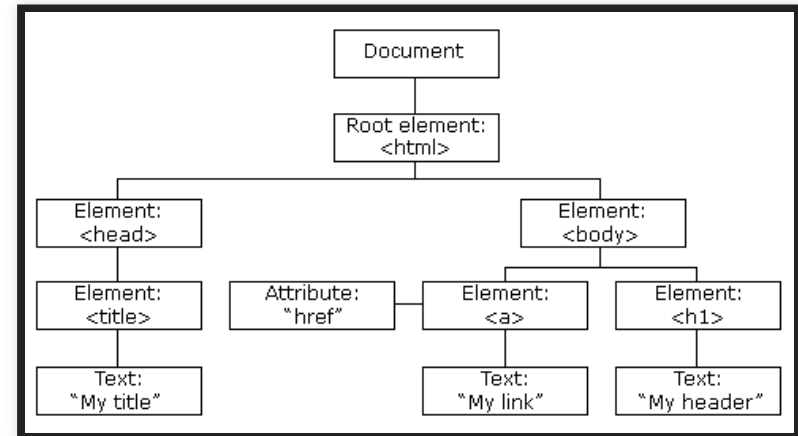
- Con el auge de DHTML se incorpora en el Javascript 1.0 una forma de acceder y manipular los elementos de la página.
- Actualmente estamos en la versión DOM4  
<https://www.w3.org/DOM/DOMTR> o DOM Living Standard  
<https://dom.spec.whatwg.org/>
- **WHATWG** (The Web Hypertext Application Technology Working Group) "A community of people interested in evolving the web through standards and tests".

## ÁRBOL DEL DOM: CORE DOM

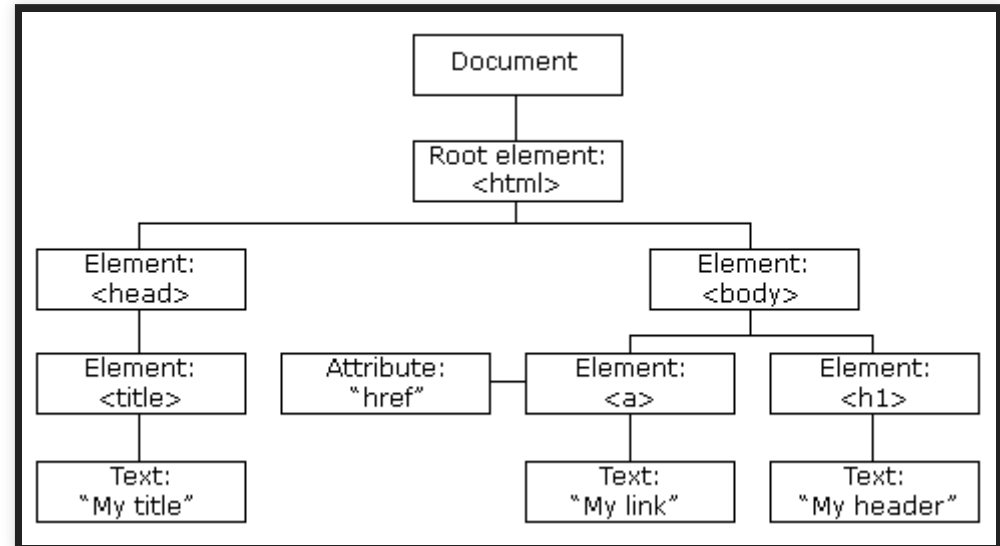
- **root node:** Nodo raíz. No tiene padre.
- **father node:** Todos los nodos(-raíz) tienen padre.
- **child node:** Un nodo puede tener varios hijos.
- **leaf node:** El nodo hoja es un nodo sin hijos.
- **sibling nodes:** Nodos hermanos tienen el mismo padre.

### Métodos de los nodos

- `x.appendChild(y):`
- `x.removeChild(nodoy):`
- `x.replaceChild(nodo1,nodo2):`



# TIPOS DE NODOS



- DOCUMENT\_NODE: Representa el documento entero.
- ELEMENT\_NODE: Todo elemento (etiquetas) es un nodo elemento.
- TEXT\_NODE: Contiene el texto entre las etiquetas de un elemento.
- COMMENT\_NODE: Nodo que representa un comentario.
- DOCUMENT\_TYPE\_NODE: Nodo que contiene el docType del documento.
- DOCUMENT\_FRAGMENT\_NODE: Nodo que representa un fragmento de documento que debe estar bien formado ("template").



## ATRIBUTOS DE LOS NODOS

- nodeName – Nombre del nodo x(no el atributo name)
- nodeValue – Valor del nodo x
- parentNode – Nodo padre de x
- firstchild
- lastchild
- previousSibling
- nextSibling
- parentNode
- childNodes – Lista nodos hijos.
- textContent – Devuelve/cambia contenido del nodo como texto.

## DOCUMENT NODE

- `x.createAttribute()`
- `x.createElement()`
- `x.createTextNode()`
- `x.getElementsByTagName()`
- `x.getElementsByClassName()`
- `x.getElementById(String id)`

## ELEMENT NODE

- Es un nodo del documento que representa una etiqueta en html o XML y su contenido.
- Métodos:
  - `x.getElementsByTagName(nombre)`: lista de “element node” con la etiqueta ‘nombre’.
  - `x.getElementById( nombre)`: elemento cuyo atributo “id” es el que se especifica como parámetro.
  - `x.getAttribute(y)`: Toma el valor del atributo que se especifica como parámetro.
  - `x.setAttribute(name, value)`: modifica el atributo 'name' asignándole el valor ‘value’.

# PHP DOM

Libreria libxml (Parser DOM):

- Core DOM
- No incluye la etiqueta Doctype.
- Compatible con el DOM level 3 core.

# EJ:DOM PHP

```
<?php
$doc = new DOMDocument();
$root = $doc->createElement( 'html' );
$doc->appendChild($root);
$head = $doc->createElement( 'head' );
$root->appendChild($head);

$title = $doc->createElement( 'title' );
$title->appendChild ( $doc->createTextNode( 'Este es el título' );
$head->appendChild($title);
$body = $doc->createElement( 'body' );
$root->appendChild($body);
$h1 = $doc->createElement( 'h1' );
$root->appendChild($h1);
$h1->appendChild( $doc->createTextNode( 'Esto es el cuerpo' );
$doctype="<!DOCTYPE html >";
echo $doctype.$doc->saveHTML( );
?>
```

<https://piruletas.000webhostapp.com/teoria/T5/HTMLDOM.php>

## EJ: DOM PHP LOAD FROM STRING

```
<?php
$html='<html><head> <meta
charset="utf-8"><title>PHP_WEB</title></head><body>
<div><h1>Web page parsing</h1>
<p>This is an example Webpage.</p></div></body></html>';
$doc = new DOMDocument();
$doc->loadHTML ($html);
$h2 = $doc->createElement( 'h2' );
$h1= $doc->getElementsByTagName( "h1" )[ 0 ];
$h1->parentNode->appendChild($h2);
$h2->appendChild($doc->createTextNode( 'Esto es el H2' ));

$doctype="<!DOCTYPE html >";
echo $doctype.$doc->saveHTML( );
?>
```

[https://piruletas.000webhostapp.com/teoria/T5/HTMLDOM\\_Load.php](https://piruletas.000webhostapp.com/teoria/T5/HTMLDOM_Load.php)

# DOM JS NAVEGADOR

Ejercicio: ¿Que hace este código?

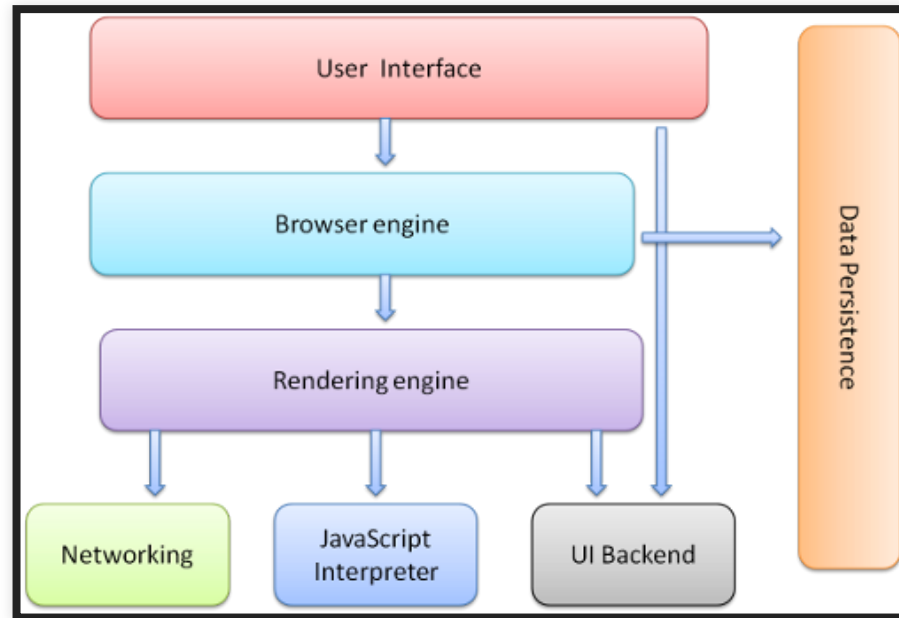
```
var doc = document.implementation.createHTMLDocument
var p = doc.createElement("p");
p.innerHTML = "This is a new paragraph.";

doc.body.appendChild(p);
p = doc.createElement("p");
p.textContent = "This is a Other new paragraf";
doc.body.appendChild(p);

// Replace the new HTML document
Nuevo = doc.documentElement
Ori = document.documentElement;
document.replaceChild(Nuevo, Ori);
```

Ejercicio: Crear una tabla con el DOM

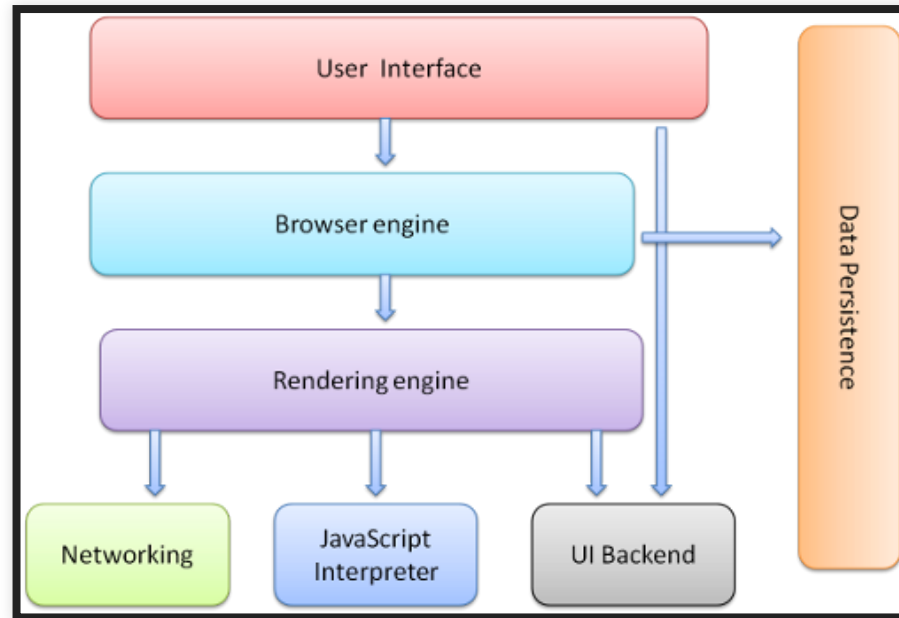
# COMPONENTES NAVEGADOR WEB



- *Interfaz de usuario* : incluye la barra de direcciones, el botón de avance/retroceso, el menú de marcadores, etc. (todo excepto la ventana principal donde se muestra la página web).
- *Motor del Navegador* : coordina las acciones entre la interfaz y el motor de renderización. Carga URL, mensajes error, botón atrás,
- *Motor de renderización* : responsable de mostrar el contenido.
- *Red* : responsable de las llamadas de red, como las solicitudes HTTP (Caché de documentos).








- *Intérprete de JavaScript* : permite analizar y ejecutar el código JavaScript. El resultado se pasa al motor de renderización.
- *Almacenamiento de datos* : capa de persistencia. Gestiona los datos de usuario, tales como marcadores, cookies y preferencias.
- *Backend de interfaz de usuario*: proporciona primitivas de dibujo, widgets de la interfaz de usuario, fuentes, etc. (utiliza métodos sistema operativo)

<https://www.html5rocks.com/es/tutorials/internals/howbrowserswork/>

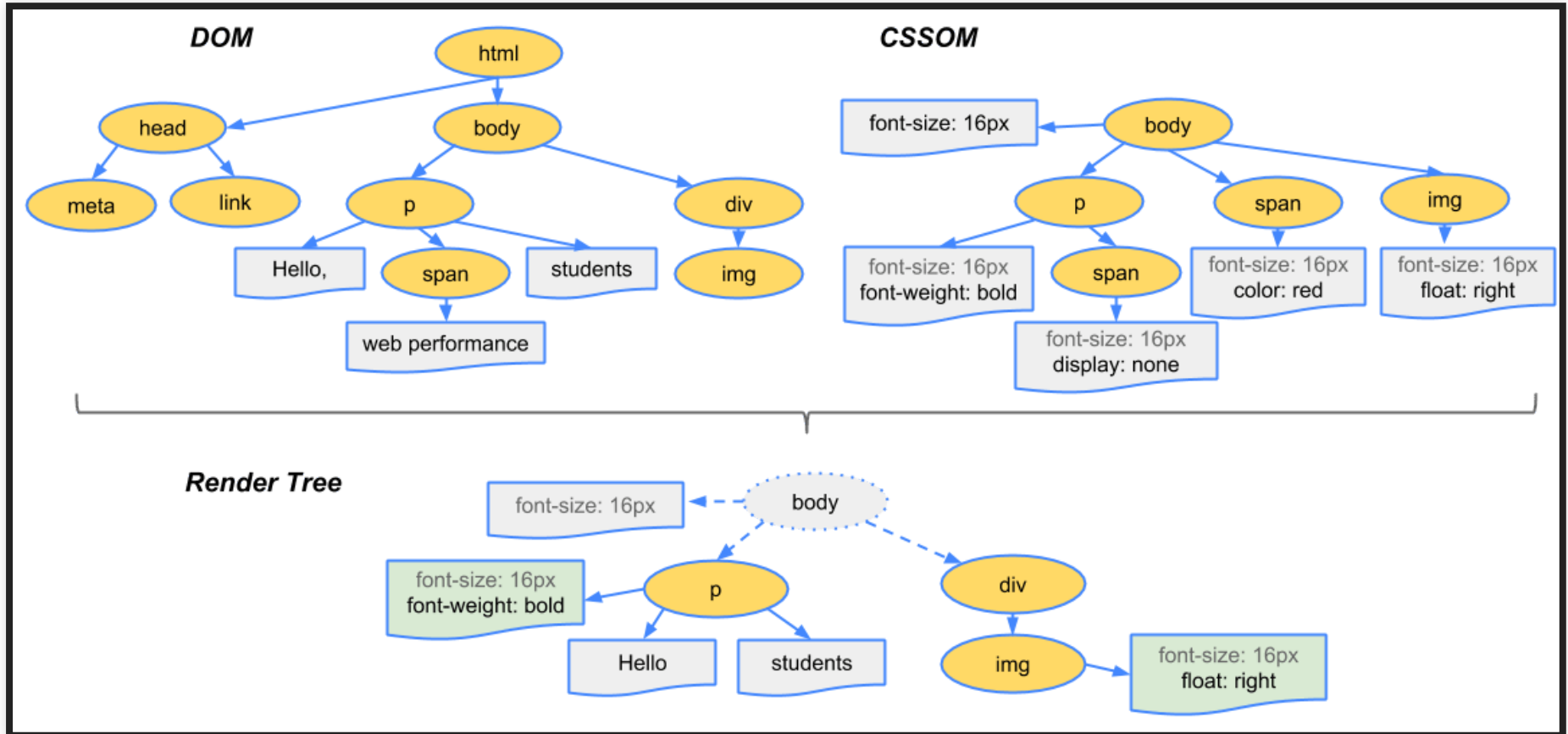
## MOTOR DE RENDERIZADO

- Un motor de renderizado (web browser engine) es el software que toma contenido marcado (como HTML, XML, archivos de imágenes, etc.) e información de formateo (como CSS, XSL, etc.) y luego muestra el contenido ya formateado en la pantalla.
  - Los motores de renderizado lo usan los navegadores web, clientes de correo electrónico, u otras aplicaciones que deban mostrar contenidos web.
  - Cada motor de renderizado del navegador suele tener su propio intérprete javascript (**script engine**)

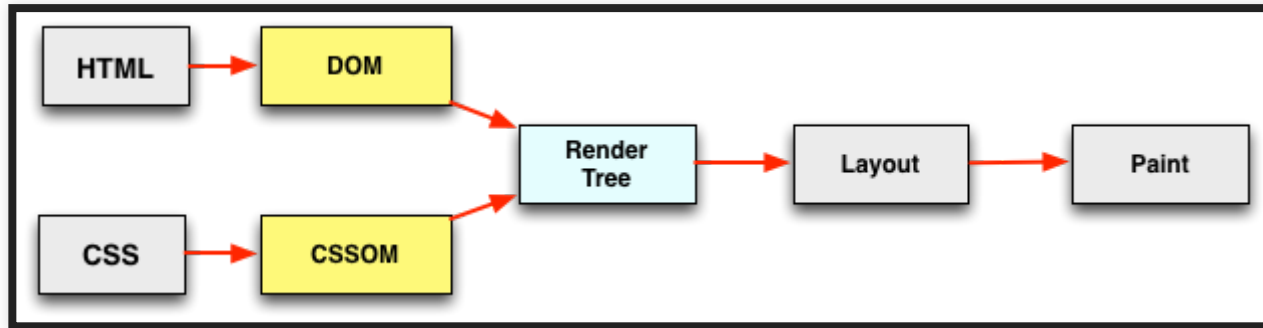
## COMPONENTES NAVEGADORES

<input type="checkbox"/>	 <b>A</b> Browser ▾	<b>A</b> Rendering / Layout Eng... ▾	<b>A</b> Scripting Engine ▾
1	Chrome	Blink (C++)	V8 (C++)
2	Mozilla Firefox	Gecko (C++)	SpiderMonkey (C/C++)
3	IE Edge	EdgeHTML (C++)	Chakra JavaScript engine (...)
4	Opera	Blink (C++)	V8 (C++)
5	Internet Explo...	Trident (C++)	Chakra JScript engine (C++)
6	Apple Safari	WebKit (C++)	JavaScript Core (Nitro)

# RENDER TREE



# FLUJO BÁSICO DEL MOTOR DE RENDERIZACIÓN



1. Los árboles DOM y CSSOM se combinan para formar el árbol de representación.
2. Render tree: El árbol de representación sólo contiene los nodos necesarios para representar la página.
3. Layout: El diseño calcula la posición y el tamaño exactos de cada objeto.
4. Paint: El último paso es la pintura, que recibe el árbol de representación final y representa los píxeles en la pantalla.

<https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction?hl=es-419>

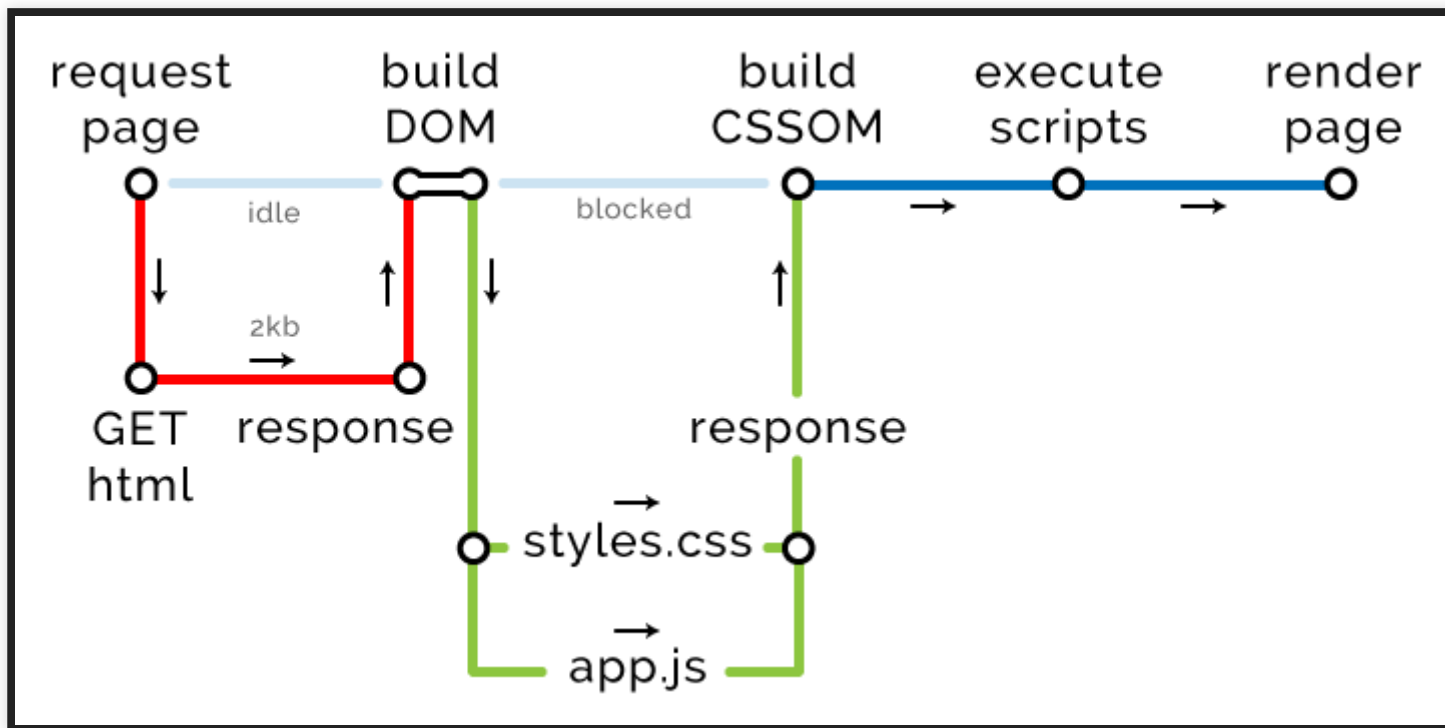
## CSS RENDER

- En <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference> tenemos una guía de CSS estándar.

```
div.error-label{  
  color: #fff;  
  background-color: #DA362A;  
  -webkit-border-radius: 5px;  
  -moz-border-radius: 5px;  
  border-radius: 5px;  
}
```

- Pero los motores de renderizado tienen ciertas extensiones de CSS que no son todavía estándares:
  - [Mozilla CSS extensions](#) (prefixed `-moz-`)
  - [WebKit CSS extensions](#) (prefixed `-webkit-`)
  - [Microsoft CSS extensions](#) (prefixed `-ms-`)

# CRP: CRITICAL-RENDERING-PATH



<https://hackernoon.com/optimising-the-front-end-for-the-browser-f2f51a29c572>

Podemos ver el CRT (ruta acceso de representación crítica ) con la línea de tiempo de las herramientas de desarrolladores.



[https://piruletas.000webhostapp.com/teoria/T5/HTML\\_DOM.html](https://piruletas.000webhostapp.com/teoria/T5/HTML_DOM.html)

```
<script type="text/javascript">
var Autor="";    var node0;
function borrar(val) { node0=val;
    console.info("nodo a borrar:"+node0.nodeName);
    console.info("val:"+val.nodeName);
    var node=val.parentNode;
    node.removeChild(node0);
    alert("¿Algo borrado ?");
}
function recuperar(Id) {
    var node = document.getElementById(Id);
    node.appendChild(node0);
    alert("nodo recuperado:");
}
</script>
<p id="borrar">Mueve ratón <span
onMouseOver="borrar(this) ;">AQUÍ</span>:</p>
<p><span onClick="recuperar('borrar') ;">PULSA
AQUI</span> para recuperar el original</p>
</div>
```



Ejercicio: Que vale :

- si el script se pone en el Head?
- si el script se pone al principio del Body
- si el script se pone al final del Body?

## CUESTIONES

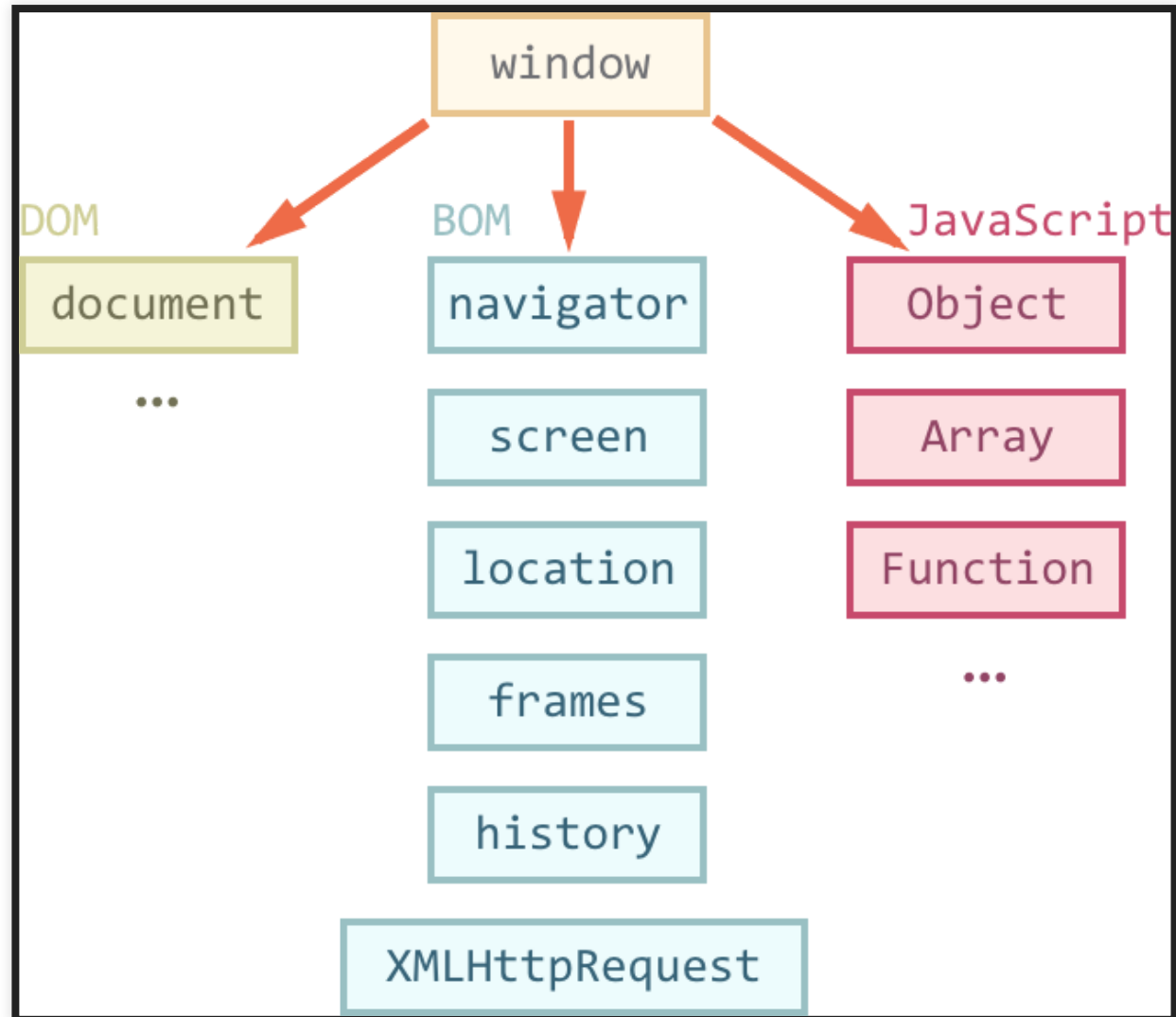
- ¿Cómo modifico un documento cargado en el navegador?
- ¿Por qué es diferente el manejo del DOM en php y HTML?
- Diferencia entre innerHTML y contentText.
- ¿Se carga primero el body o se ejecuta primero el script?

## CUESTIONES DOM

<https://piruletas.000webhostapp.com/teoria/T5/FormEventHandlerError.html>

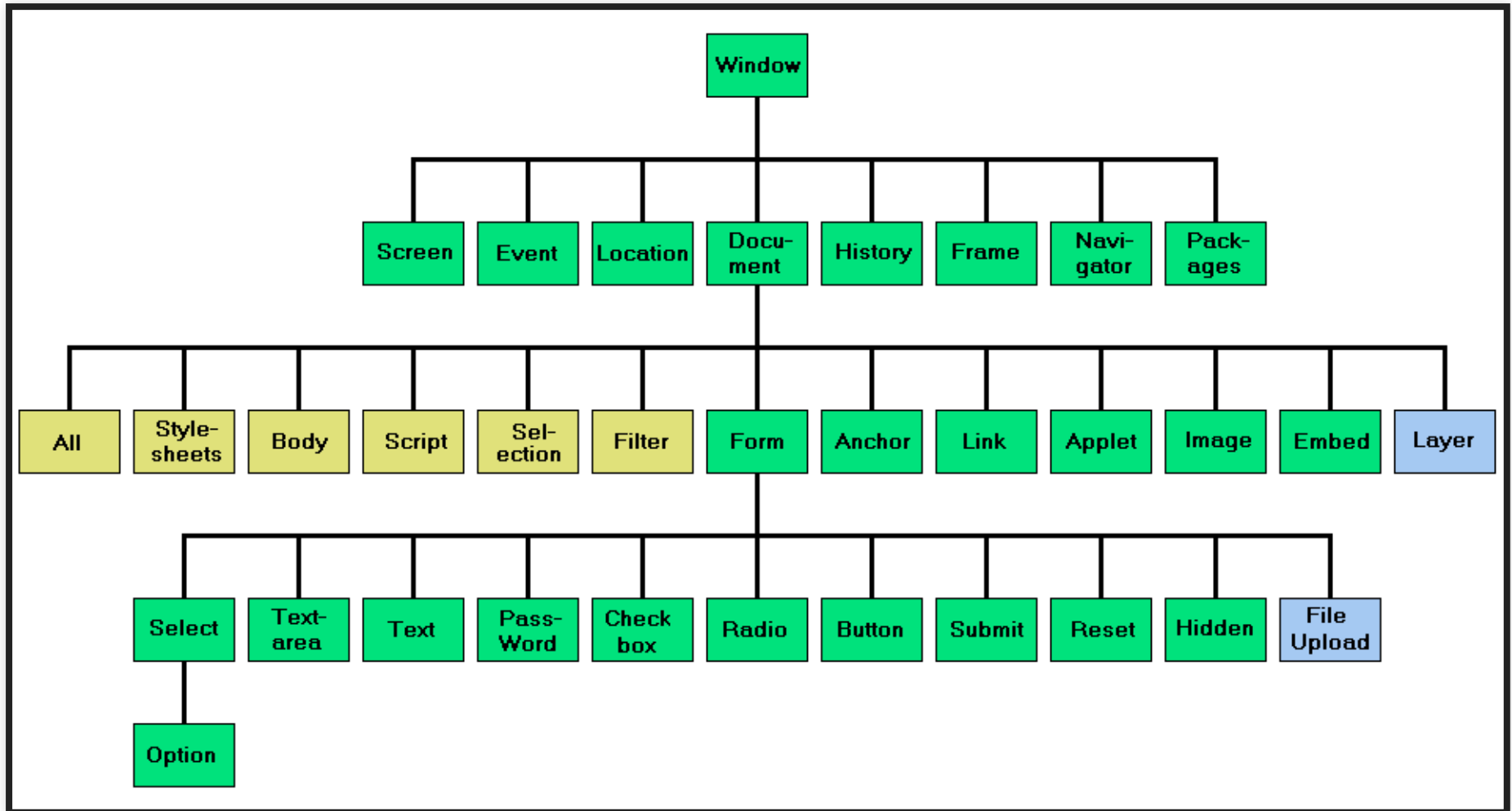
```
<script type="text/javascript">
    var ventana=window.open();
    var father, child;
    father=ventana.document.querySelector('head');
    child = document.createElement('title');
    child.textContent='HTMLDOM';
    father.appendChild(child);
    father=ventana.document.querySelector('body');
    child = document.createElement('h1');
    father.appendChild(child);
    child.textContent='Bienvenido';
    child = document.createElement('h2');
    child.innerHTML='Esto es el H2';
    father.appendChild(child);
</script>
```

# BOM: BROWSE OBJECT MODEL (BROWSER WEB API)



<https://javascript.info/browser-environment>

# BOM



[http://www.cs.ucc.ie/~gavin/javascript/05\\_img01.gif](http://www.cs.ucc.ie/~gavin/javascript/05_img01.gif)

## WINDOW OBJECT

- Es el Nodo raíz del BOM o sea de las **API Web** de los navegadores.
- El objeto window es el de mayor nivel en la jerarquía de objetos de JavaScript en el navegador.
- Es un objeto global en JavaScript. Contiene todas las variables y funciones globales
- Representa la "ventana del navegador" y proporciona métodos para controlarlo.



# DOCUMENT OBJECT

- Es el Nodo raíz del DOM
- Todos los nodos heredan los atributos del objeto `eventTarget`
- Atributos:
  - `document.documentURI`
  - `document.contentType`
  - `document.styleSheets`
  - `document.images`: lista de imágenes
  - `document.anchors`: lista de hiperenlaces.
  - `document.body`: nodo elemento body.

## EJEMPLO CREAR NUEVA VENTANA EN NAVEGADOR CON UN DOCUMENTO.

```
var ventana=window.open();
var father, child;
father=ventana.document.querySelector('head');
child = document.createElement('title');
child.textContent='HTMLDOM';
father.appendChild(child);
father=ventana.document.querySelector('body');
child = document.createElement('h1');
father.appendChild(child);
child.textContent='Bienvenido </br>';
child = document.createElement('h2');
child.innerHTML='Esto es el H2<br/>';
father.appendChild(child);
```

# ELEMENT Y DOCUMENT

Los nodos Element y Document tienen:

- Propiedades:
  - `X.events` // lista de Eventos de x
  - `x.methods` // lista de métodos de x
  - `x.elements` // colección de nodos de tipo elementos de x
  - `x.name` // lista de hijos de X con el nombre 'name'
  - `x.innerHTML` // asigna o devuelve el contenido html del nodo elemento(Crítico).
  - `x.style` // permite dar estilo a las etiquetas.
- Eventos:
  - `blur()` :Quita el foco en de un elemento.
  - `click()`: realiza un click en un elemento.
  - `focus()`: Pone el foco en un elemento.
  - `toString()`:convierte el contenido en una cadena.

## THIS OBJECT

- Cuando estamos dentro del atributo de una etiqueta **html** , **this** se refiere a la **etiqueta** que contiene el atributo.
- Cuando estamos en un Objeto , **this.xxx** se llama dentro de un método y **this** refiere al **objeto** que lo contiene.
- En otro caso, **this** se refiere a la variable global, **window** del documento activo cargado en la ventana.

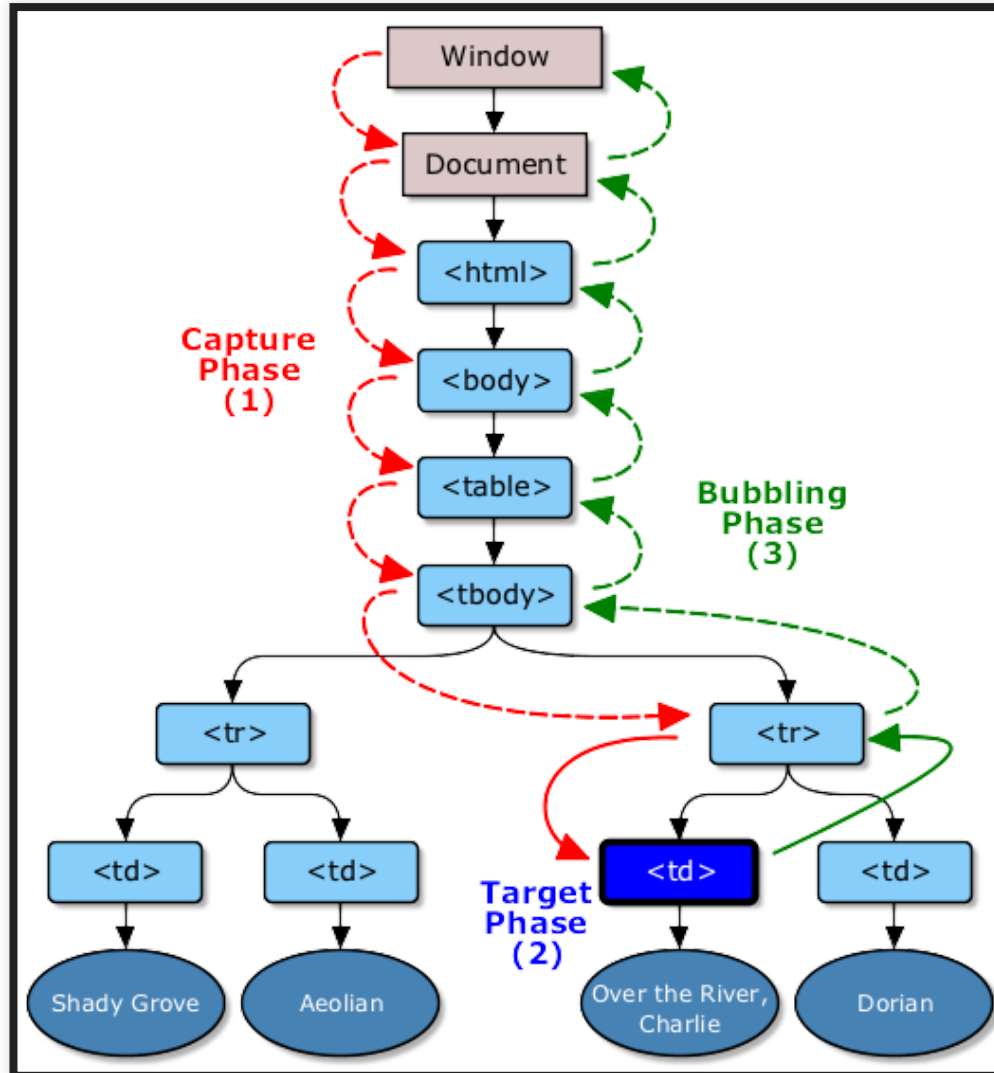
<https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores>

```
window.alert( 'error' );  
alert( "sin window!" );
```

## GESTIÓN DE EVENTOS

- **Capture** : Al realizar el cliente un evento, el navegador sabe que se ha producido un evento, y busca desde la raíz del DOM para ver donde se ha producido. *event object*
- **Target** : Una vez sabe en qué elemento se ha producido (*event target object*) el explorador comprueba si tiene algún controlador en ese elemento, si existe lo ejecuta.
- **Boobling** : Después de disparar el lanzador, busca hacia la raíz, si en los niveles padre del elemento hay algún otro controlador que se active con ese evento, en su caso lo lanzará. Esta etapa de su movimiento hacia arriba se llama la propagación de eventos

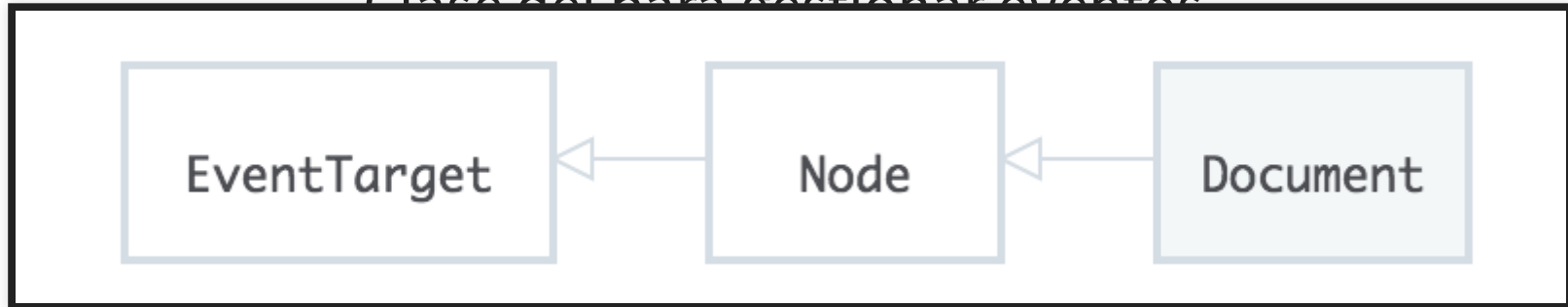
# CAPTURE/BLUBBING



<http://www.thatjsdude.com/images/eventBubble.png>

## EVENT TARGET INTERFACE

*Clase del navegador para gestionar eventos*



- EventTarget permite detectar/recibir los eventos que se producen en el DOM o BOM
- Windows, Document y Element soportan definir detectores de eventos, “event hanlers”.

## REGISTRO EVENTOS

`x.addEventListener ("Evento", funcionEjecutar, Boolean*)`

- El **evento** es el nombre del evento “click”, “load”,
- La **funcionEjecutar** se ejecuta cuando se produce el evento
- El **Boolean es opcional** especifica si el evento debe ser capturado (true) o no debe serlo (false, que es el valor que se aplica por defecto si no se especifica este parámetro).



## EJEMPLOS THIS

```
var o = {  
  prop: 37 ,  
  f: function() {  
    return this.prop;  
  }};  
console.log(o.f()); // logs 37  
function bluify(e){  
  console.log(this === e.currentTarget); // Siempre true  
  this.style.backgroundColor = '#A5D9F3';  
}  
x.addEventListener('click', bluify, false);
```

<https://piruletas.000webhostapp.com/teoria/T5/FormEventHandlerType.html>

# TEMPLATE: DOCUMENT FRAGMENT

```
<table id="producTable">
<thead>
<tr>
<td>UPC_Code</td>
<td>Product_Name</td>
</tr>
</thead>
<tbody><!-- Si existen datos se incluye aqui -->
</tbody>
</table>
<template id="productRow">
<tr>
<td class="record"></td>
<td></td>
</tr>
</template>
```

El contenido del nodo **template** es un **document fragment**. Este se representa en un árbol del DOM distinto al documento HTML. Un document Fragment no tiene nodo padre.



## TEMPLATE: DOCUMENT FRAGMENT

```
if ('content' in document.createElement('template')) {  
  
    // Instantiate the table with the existing HTML tbody  
    // and the row with the template  
    var t = document.querySelector('#productrow');  
  
    // Clone the new row and insert it into the table  
    var tb = document.querySelector("tbody");  
    var clone = document.importNode(t.content, true);  
    td = clone.querySelectorAll("td");  
    td[0].textContent = "1235646565";  
    td[1].textContent = "Stuff";  
  
    tb.appendChild(clone);  
  
    // Clone the new row and insert it into the table  
    var clone2 = document.importNode(t.content, true);  
    td = clone2.querySelectorAll("td");  
    td[0].textContent = "0384928528";  
    td[1].textContent = "Acme Kidney Beans 2";  
}
```

```
tb.appendChild(clone2);
```

```
}
```

[https://mdn.mozillademos.org/en-US/docs/Web/HTML/Element/template\\$samples/Example?revision=93413158](https://mdn.mozillademos.org/en-US/docs/Web/HTML/Element/template$samples/Example?revision=93413158)

```
<body>
<table id="producttable">
<thead>

<tr>

<td>UPC_Code</td>

<td>Product_Name</td>

</tr>
</thead>
<tbody>

<!-- existing data could optionally be included here -->
</tbody>
```

```
</table>

<template id="productrow">
<tr>

<td class="record"></td>

<td></td>
</tr>
</template>
</html></body>
```

# JSON:DATO ESTRUCTURADO

JSON, acrónimo de **JavaScript Object Notation** , es un formato ligero para el intercambio de datos.

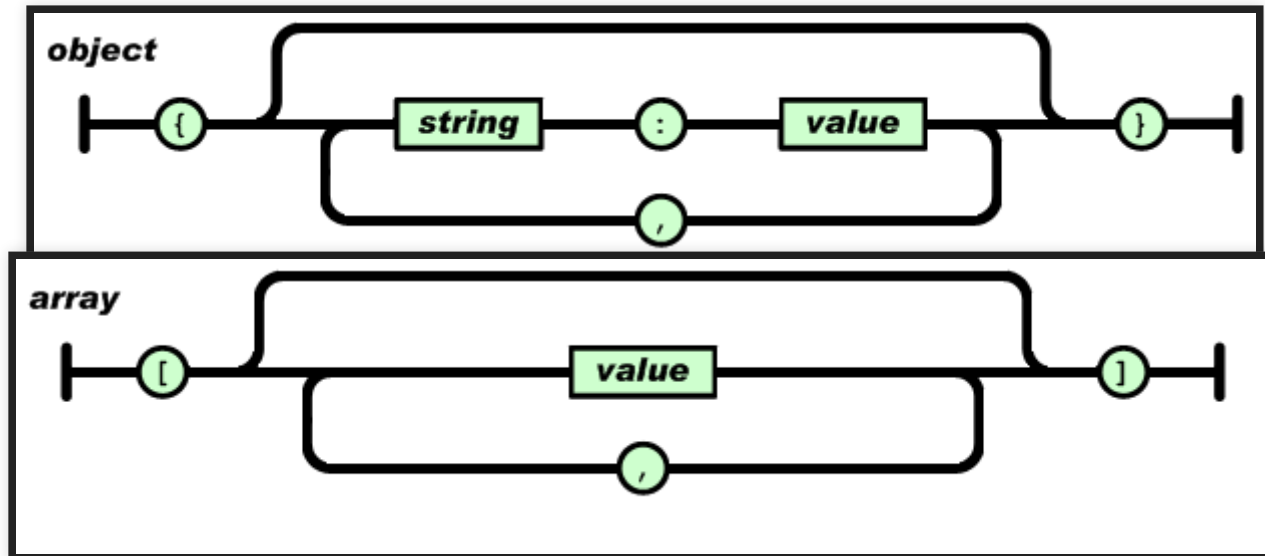
## Propiedades:

- Ligero: formato ligero para el intercambio de datos
- Simple: Formato sencillo de leer por humanos y de parsear/analizar por máquinas
- Funcional: Los objetos JSON están tipificados.
- Abierto: Formato de texto independiente del lenguaje.
- Extensible: formado por unas estructuras que pueden anidarse.



## ESTRUCTURA DATOS JSON

- JSON es una sintaxis para serializar objetos, arreglos, números, cadenas, booleanos y nulos.
- La estructura de datos de JSON está basada en pares nombre:valor
- Las 2 estructuras principales son el objeto Json y la Lista JSON.



## ACCESO ELEMENTOS OBJETO JSON

```
JSONdata={ "menu": {  
  "id": "file",  
  "value": "File",  
  "popup": {  
    "menuitem": [  
      {"value": "New", "onclick": "CreateNewDoc()"},  
      {"value": "Open", "onclick": "OpenDoc()"},  
      {"value": "Close", "onclick": "CloseDoc()"}  
    ]  
  }  
}}
```

```
var menu = JSONdata.menu;  
fichero = JSONdata.menu.value; (  
var submenu = JSONdata.menu.menuitem;  
var submenuVal = JSONdata.menu.popup.menuitem[ 0 ]["value"]
```

## Ejercicios:

- Construye un el Json `//{"bar":"new property","baz":3}`
- Construye un el Json `//["pepe","Jose"]`

## EJERCICIOS

```
vfoo.bar = ar foo = {}; "new property" (^) ;  
foo.baz = 3 ;  
var console.log (jsonString); jsonString = JSON.stringify(  
//{"bar":"new property","baz":3}
```

# JSON HTTP

Se transmiten cadenas en HTTP por ello el proceso es:

- Envío
  - Definir un objeto JSON
  - Convertir JSON a una cadena
- Recepción:
  - Recibir cadena convertirla

```
- a objeto JSON en **JavaScript**.  
- a un diccionario o array en **PHP**.
```

## MÉTODOS OBJETO JSON JAVASCRIPT

- `JSON.parse()`: Analiza una cadena de texto con formato JSON retornando un objeto JSON. Se puede añadir una función para la transformación como parámetro.
- `JSON.stringify()`: Devuelve un string a partir de un JSON. Se puede añadir una función para la transformación como parámetro.

## EJEMPLO ENVÍO JSON CON JAVASCRIPT

```
var foo = {};  
foo.bar = "new property";  
foo.baz = 3 ;  
var jsonString = JSON.stringify(foo);  
console.log (jsonString);  
//{"bar":"new property","baz":3}  
var url = "JSONExample3.php createXMLHttpRequest()";  
xmlHttp.open("POST", url, true);  
xmlHttp.setRequestHeader("Content-Type","application/json");  
xmlHttp.onreadystatechange = handleStateChange;  
  
xmlHttp.send(jsonString);
```

# JSON PHP

- **json\_encode** (cadena): Función que devuelve un objeto o arreglo a partir de una cadena con sintaxis JSON.
- **json\_decode** (dato); función que devuelve un objeto o arreglo en formato cadena con sintaxis JSON.

```
$json =  
'[{"0":"1","id_fruta":"1","1":"Manzana","nombre_fruta":"M"  
,"2":"100","cantidad":"100"}, {"0":"2","id_fruta":"2","1"  
"nombre_fruta":"Platano","2":"167","cantidad":"167"}, {"0"  
_fruta":"3","1":"Pera","nombre_fruta":"Pera","2":"820","c  
":"820"}]';  
$array = json_decode($json);  
print_r($_array)  
//OBTENER UN DATO DIRECTAMENTE DEL ARRAY.  
echo $array[ 0 ]->nombre_fruta;  
//RECORRER Y RECUPERAR VALORES JSON CON  
FOREACH.  
foreach($array as $obj){  
$id_fruta = $obj->id_fruta;  
$nombre_fruta = $obj->nombre_fruta;
```



```
$cantidad = $obj->cantidad;  
echo $id_fruta." ".$nombre_fruta." ".$cantidad;  
}
```

1 Manzana 1002 Plátano 167

3 Pera 820

```
Array ( [0] => stdClass Object ( [0] => 1 [id_fruta] => 1  
=> 100 ) [1] => stdClass Object ( [0] => 2 [id_fruta] =>  
=> 167 ) [2] => stdClass Object ( [0] => 3 [id_fruta] =>  
820 ) )
```

## EJEMPLO ENVÍO UNA TABLA CON JSON

```
[{"id": "1", "nombre": "pepe", "apellido": null, "email": null, "clave": null}, {"id": "2", "nombre": "pepe", "apellido": null, "email": null, "clave": null}]
```

```
header('Content-type: application/json');
$result= $pdo->prepare("SELECT * FROM usuarios ");
$result->execute();
$datos= $result->fetchAll(PDO::FETCH_ASSOC);
echo json_encode($datos);
```

<https://dllido.al.nisu.org/Lab2017/T4/jsonTemplate0.php>

var datos=

```
[{"id": "1", "nombre": "pepe", "apellido": null, "email": null, "clave": null}, {"id": "2", "nombre": "pepe", "apellido": null, "email": null, "clave": null}]
```

```
var t = document.querySelector( '#productrow' );
var tb = document.getElementsByTagName( "tbody" );
var clone;
```

```
td = t.content.querySelectorAll("td"); for (var i = 0; i < datos.length; i++){
```

```
td[0].textContent =datos[i].id;
td[1].textContent = datos[i].nombre;
clone = document.importNode(t.content, true);
tb[0].appendChild(clone);
}
73
```

<https://dllido.al.nisu.org/Lab2017/T4/listarTemplate0.htm>

```
<table
id="producttable">
<thead>
<tr>
<td>UserID</td>
<td>UserName</td>
>
</tr>
</thead>
<tbody>
</tbody>
</table>
<template
id="productrow">
<tr>
<td>
```

```
class="record"></td>  
<td></td>  
</tr>  
</template>
```

## IFRAME NODE

- Permite anidar un documento html dentro de otra página HTML.
- Cada elemento *iframe* tiene su propio historial de sesión y su propio objeto Document.
- El contexto de navegación de nivel superior del iframe es la ventana del navegador: *window*
- **window.frames** devuelve la lista de iframes del documento.

## SAMPLE IFRAME

```
<!DOCTYPE html>
<html><head>
<meta charset="utf-8">
<title>Web page parsing</title>
</head>
<body>
<div>
<h1>Web page parsing</h1>
<p>This is an example Web page.</p>
</div>
<iframe width="400" height="215" frameborder="0"
scrolling="no" marginheight="0" marginwidth="0"
src="https://maps.google.com/maps?f=q&source=s_q&
buenos+aires&sll=37.0625,-95.677068&sspn=38.63881
&hq=&hnear=Buenos+Aires,+Argentina&z=11&l
tput=embed"></iframe>**
</html>
```

<https://piruletas.000webhostapp.com/teoria/T5/webDOMiframe.html>

# CORS

- Supongamos que tu aplicación reside en example.com y quieres extraer datos de <http://www.example2.com>. Normalmente, si intentas ejecutar este tipo de llamada desde Javascript, la solicitud fallará y el navegador lanzará un error por la falta de correspondencia de los orígenes.
- Uso compartido de recursos de origen cruzado (CORS) permite a las aplicaciones web de un dominio realizar solicitudes de dominio cruzado.
  - HTML5 introduce Cross-Origin Resource Sharing (CORS)

<http://www.w3.org/TR/cors/> <http://enable-cors.org> -Hay que habilitar estas llamadas tanto en el servidor como en el cliente.

## CORS EN EL SERVIDOR

<http://www.example2.com> puede que habilitar CORS de dos formas:

- añadir un encabezado (header) para permitir solicitudes de example.com:

```
<?php header( 'Access-Control-Allow-Origin: *' ); ?>
```

- o en el fichero .htaccess

```
Header set Access-Control-Allow-Origin "*"
```



## CORS EN EL CLIENTE

En la etiqueta script añadir atributo **crossorigin**:. Indica si la petición hecha a un servidor externo debe presentar credenciales CORS o no.

- **crossorigin=anonymous** : las peticiones CORS para el elemento tendrán la etiqueta "omit credentials" establecida.
- **crossorigin=use-credentials** : las peticiones CORS para el elemento no tendrán la etiqueta "omit credentials" establecida.

EJEMPLO Google maps con cors y sin cors

# GENERADORES

Soluciones para utilizar código JS no compatible:

1- Transpiler(compilador): Transforma tu código reemplazando las secciones de código para que se puedan ejecutar con el JS nativo.

- Babel : Permite transformar el código con las nuevas características de ES6, en un JS ES5 o inferior.
- Traceur: Transpiles ES6, ES7, and beyond into ES5

2- Código polyfill (Plugin): permite emular algunas APIs aunque no estén implementadas en el JS nativo. Se utiliza para que los navegadores soporten las últimas APIs que quieras utilizar. Implementa el código que los navegadores no soportan.

## BABEL

```
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
<!-- Your custom script here -->
<script type="text/babel">
const getMessage = () => "Hello World";
document.getElementById('output').innerHTML = getMessage('');
</script>
```

--

##

## BIBLIOGRAFÍA

- <https://dom.spec.whatwg.org/> DOM LIVE -  
<https://www.html5rocks.com/es/tutorials/internals/howbrowserswork/>