



JAVASCRIPT

© 2017, ACTIBYTI PROJECT SLU, Barcelona
Autor: Ricardo Ahumada



MINISTERIO
DE ENERGÍA, TURISMO
Y AGENDA DIGITAL

red.es



ESTRATEGIA DE
EMPRENDIMIENTO Y
EMPLEO JUVENIL
garantía juvenil



UNIÓN EUROPEA

Fondo Social Europeo
"El FSE invierte en tu futuro"

ÍNDICE DE CONTENIDOS

1. Caso práctico
2. Introducción
3. Sintaxis
4. Funciones
5. Objetos Javascript
6. Interacción con el navegador (DOM)
7. Eventos
8. Validación de formularios

1

CASO PRÁCTICO



Caso Práctico: BananaTube funcional

“BananaTube” es el proyecto estrella de Banana Apps.

BananaTube será el próximo boom! de las redes sociales; permitirá a sus usuarios gestionar videos, exponerlos en su muro, comentar videos propios y de sus amigos, calificarlos y compartirlos en varios canales.

En esta etapa del proyecto se quiere realizar un prototipo navegable funcional, de tal manera que el cliente pueda ver y sentir toda la potencia de la funcionalidad que ofrecerá la aplicación. Por ejemplo, el usuario debe poder eliminar proyectos de la lista y ver que realmente se eliminan,...



Discutamos

- Cómo necesitaremos que se comporte nuestra aplicación?
- Cuál es nuestro entorno?
- Qué tecnología necesitaremos?
- Qué herramientas necesitaremos?

2

INTRODUCCIÓN

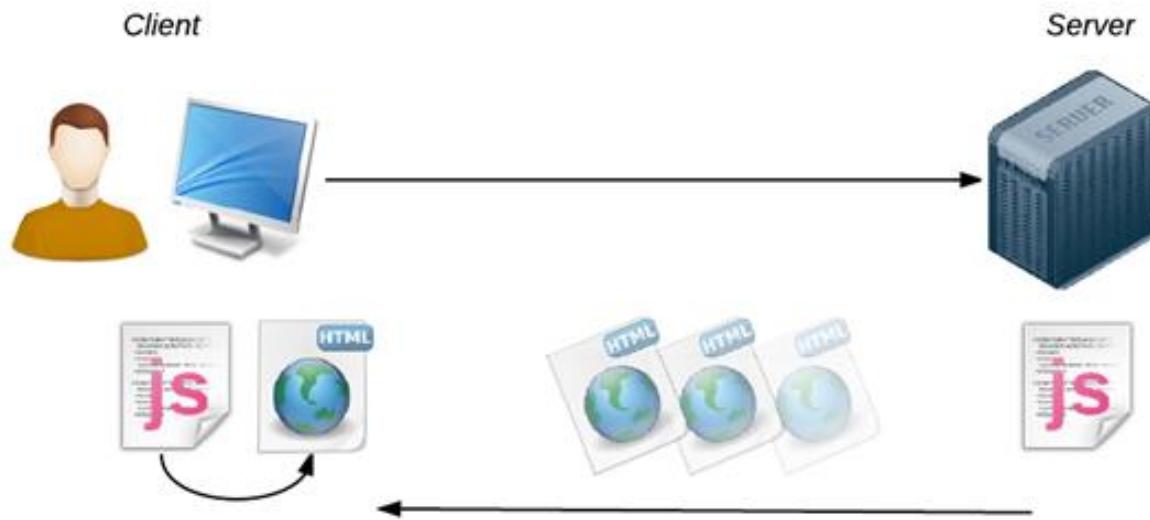
¿Qué es JavaScript?

- Javascript es un lenguaje de programación que permite a los desarrolladores crear acciones en sus páginas web.
- Javascript es un lenguaje con muchas posibilidades
 - Usado para crear pequeños programas que luego son insertados en una página web
 - En programas más grandes, orientados a objetos mucho más complejos.
 - También podemos crear diferentes efectos e interactuar con nuestros usuarios.
- Existen dos tipos de JavaScript
 - El que se ejecuta en el cliente, el Javascript propiamente dicho, técnicamente Navigator JavaScript.
 - El que se ejecuta en el servidor, es más reciente y se denomina LiveWire Javascript.



Esquema de funcionamiento de JavaScript

- JavaScript es enviado por el servidor y se ejecuta en el navegador. Exactamente igual que el código HTML.
- El navegador lee las referencias de javascript en la página HTML, lo interpreta y ejecuta



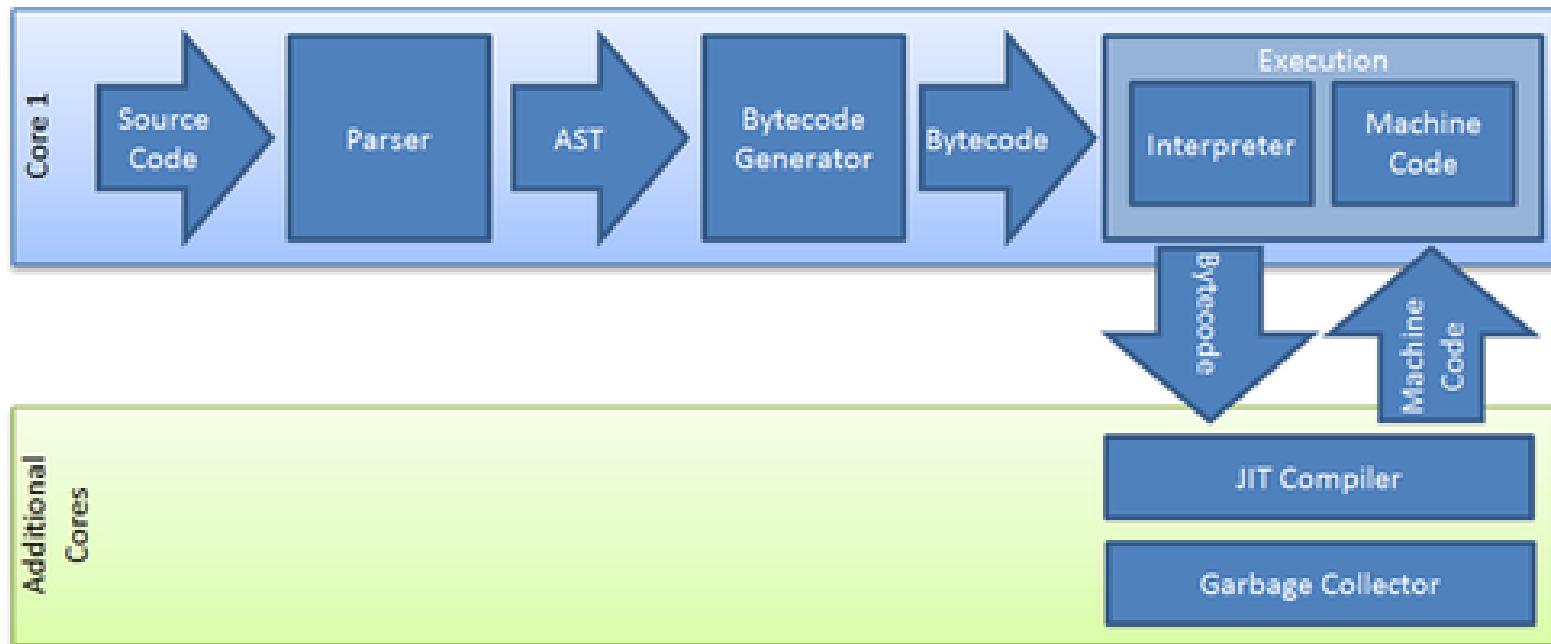
- Actualmente existe de ejecutar javascript en el servidor usando **Node.js**

El navegador desde el punto de vista Javascript

- Los navegadores poseen un **motor de interpretación de Javascript**
- Un motor de JavaScript es un tipo de proceso (máquina virtual) que está diseñado específicamente para interpretar y ejecutar código JavaScript.
- Cada motor de JavaScript implementa una versión de ECMAScript (<https://www.ecma-international.org/publications/standards/Ecma-262.htm>)
- A medida que ECMAScript evoluciona, también lo hacen los motores de JavaScript.
- La razón por la que hay tantos motores diferentes es cada uno está diseñado para trabajar con un navegador web diferente, navegador sin interfaz gráfica (headless browser), o Node.js.
 - Ahora, el nivel de acuerdo es mucho más alto, gracias a los acuerdos sobre los estándares.
 - Existen librerías JavaScript que simulan comportamientos actuales en navegadores antiguos: Modernizr, etc.

Proceso de interpretación

- La interpretación del código depende del motor.
- Los dos principales motores de interés para nosotros, debido a que están apalancados por NativeScript, son JavaScriptCore de WebKit y el motor V8 de Google.
- Estos dos motores gestionan el procesamiento del código de manera diferente.



Donde generar el Javascript

- En cualquier lugar dentro del <html>
 - Preferiblemente después del </body>
- Embebido en las etiquetas <script></script>

Ejemplo:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>Title of the document</title>
6          <script>
7              alert("Hola Mundo Head!")
8      </script>
9      </head>
10
11     <body>
12         Content of the document.....
13     <script>
14         alert("Hola Mundo Body!")
15     </script>
16     </body>
17
18     <script>
19         alert("Hola Mundo Fuera!")
20     </script>
21 </html>
```



Especificar un fichero de Script

- Normalmente se separa el javascript del html en su propio fichero
- El atributo SRC de la etiqueta <SCRIPT> es donde especifica el origen de un fichero de JavaScript.

```
<script src="js/miFichero.js"></script>
```

- Una vez introducido podemos usar las funciones del fichero como si las hubiéramos escrito en la página.
- El fichero debe tener extensión .js
- Estos ficheros solo pueden contener JavaScript.
- Se recomienda poner la referencia al final de tag body en el html.



Ejemplo: reproduce este código

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>

<body>
  <h1>Título</h1>
  <p>Esto es un texto</p>

</body>

<script src="js/script.js"></script>

</html>
```

js/script.js

```
var un_numero = 1;
var un_flotante = 1.5;

var otro_numero = '1';

var un_texto ='esto es un texto';

var esVerdad = true;
```

3

SINTAXIS

Visión general de la sintaxis

```
// dos contrabarras.... Comentario simple

var x; // declara una variable

x = 3 + y; // asigna un valor a la variable `x`

foo(x, y); // llama a la función `foo` con parámetros `x` e `y`
obj.bar(3); // llama al método `bar` del objeto `obj`

// Una declaración condicional
if (x === 0) { // Es `x` igual a cero?
    x = 123;
}

// Define la función `baz` con parámetros `a` y `b`
function baz(a, b) {
    return a + b;
}
```

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

Variables

- Sirven para **almacenar valores** y poder así usarlos posteriormente en el código.
- Se definen con la palabra reservada **var**, seguido del identificador (nombre) de la variable

```
var arg0;  
var _tmp;  
var $elem;  
var π;
```

- El primer carácter de un **identificador** puede ser cualquier letra Unicode, un signo de dólar (\$) o un guión bajo (_).
 - Los caracteres siguientes también pueden ser cualquier dígito Unicode.
 - JavaScript es sensible a mayúsculas y minúsculas: *unVar* no es lo mismo que *UnaVar*.
- Las variables tienen un ámbito:
 - Global (fuera de una función) // por defecto en JS
 - Local (dentro de una función)

Variables: palabras reservadas

- Las siguientes palabras son parte de la sintaxis de JS y **no pueden usarse** como nombre de variables

arguments	break	case	catch
class	const	continue	debugger
default	delete	do	else
enum	export	extends	false
finally	for	function	if
implements	import	in	instanceof
interface	let	new	null
package	private	protected	public
return	static	super	switch
this	throw	true	try
typeof	var	void	while

Valores que pueden recoger las variables

Usaremos una variable siempre que queramos guardar un valor en memoria para después usarlo.

Los valores que pueden adquirir las variables pueden ser:

➤ **Numéricos**

- var x = 2; // tipo numérico
- var y = 1.5; // tipo numérico (real)

➤ **Lógicos**

- var ok = true; // tipo booleano

➤ **Cadenas (String)**

- var a = "Hola"; // tipo String
- var a = "Adiós"; // Asignación Dinámica

➤ **Nulos (null)**

- var a = null; // tipo Nulo

➤ **Sin definir**

- var a; // tipo indefinido (errores undefined y NaN)

Controlando los valores de variables



- Para ver un valor de variable, usa el comando **console.log()**
- Una vez escrito el código abre tu página html en el navegador
- Accede al panel de desarrollador → console
- La salida la podrás obsérvala en la **consola**

```
var un_texto = 'Hola, cómo estás?';  
console.log(un_texto);
```

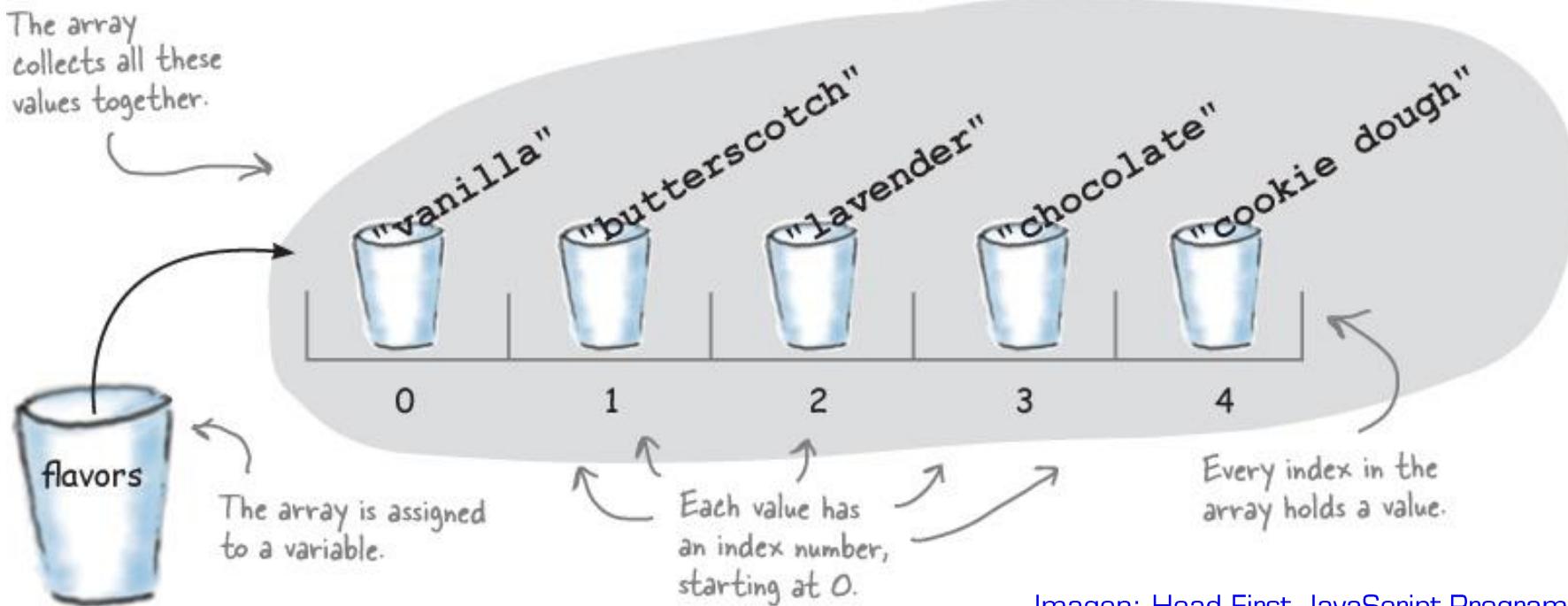
```
un_texto = 'El texto ha cambiado';  
console.log(un_texto);
```

```
var un_numero = 1235;  
console.log(un_numero);
```

```
un_numero = 346;  
console.log(un_numero );
```

Arrays

- Un **array** es un conjunto de datos ordenados por posiciones y todos asociados en una sola variable.
- Los datos pueden ser de cualquier tipo de dato.
- Podremos acceder a estos datos a través de su **posición**: 0...N
 - `unArray[0], unArray[1], ..., unArray[N]`
- Podemos conocer el tamaño de un array usando su propiedad **length**:
 - `unArray.length`



Arrays: Ejemplos

```
var arrayRapido = [12,45,"array inicializado en su declaración"]
```

```
Var miArray=[];  
  
miArray[0] = "Hola";  
miArray[1] = "a" ;  
miArray[2] = "todos";
```

```
var otroArray=[];  
  
otroArray[0] = "un texto";  
otroArray[1] = 1275 ;  
otroArray[1] = 0.78 ;  
otroArray[2] = true;
```

Literales

- Boolean // es el objeto boolean
 - true o false.
- Coma flotante
 - 3.14152...
 - 3E12, .1E-3
- Enteros
 - 052 // 52 en base 8.
 - 0x1A // 1A en base 16.
- Objetos
 - Persona {nombre:"yo", apellidos:"mismo"} // aquí define un objeto Persona y inicializa sus atributos
- Cadena
 - "hola"
 - "una línea \n otra línea"

Expresiones con variables

- › Asignación

$x=7;$

- › Evaluación

$x = 3+4;$

Operadores de asignación

- $x++;$ → $x=x+1$
- $x+=y;$ → $x=x+y;$
- $x *= y;$ → $x=x*y;$
- $x /= y;$ → $x=x/y;$
- $x\% = y;$ → $x=x\%y;$ → módulo o resto de una división

Operadores de Comparación

- Igual (==)
- No igual (!=)
- Igual estricto (===) → verdadero si son iguales y del mismo tipo
- No igual estricto (!==) → verdadero si no son iguales y/o del mismo tipo
- Mayor (>).
- Mayor o igual (>=).
- Menor (<).
- Menor o igual (<=).

Operadores aritméticos

- Módulo (%).
- Incremento (++)
- Decremento (- -)
- Negativo (-)

Más operadores

➤ Lógicos

- and `&&`, or `||`, not `!`

```
var a = true, b=false;  
  
console.log(a && b);  
console.log(a || b);  
console.log(!a);  
console.log(!b);
```

➤ Cadena

- Concatenación de textos: `+`, `+=`

```
var un_texto = 'hola';  
un_texto = un_texto + 'como';  
un_texto = un_texto + 'estás?';
```

➤ Operadores especiales

- Condicional: `?` (expresión ternaria)

- `(evaluación)? <caso verdadero>:<caso false>;`
- `a = isTrue? x : y ;`

```
var valor = (a==b)? 'iguales' : 'diferentes' ;
```

Sentencias condicionales

➤ if (condicion) else

```
var x=2;
if (x==2){
    alert (x);
}else{
    alert (x+1);
}
```

➤ switch

```
var x=2;

switch (x){
    case (1):
        alert ("uno");
        break;
    case (2):
        alert ("dos");
    default:
        alert ("otro");
}
```

Sentencias iterativas

> for

```
for (i=0; i<10; i++) { ... }
```

```
for (variable of iterable) { ... }
```

```
for (variable in object) { ... }
```

> do while

```
do {  
    i++;  
} while(i<10);
```

> while

```
while (i<10) { ... }
```

> break // sale de un bucle

```
for (i=0; i<10; i++) { ...  
    break;  
}
```

> continue // reinicia un bucle



Pongámoslo en práctica

- Crea un array para almacenar el nombre de usuarios
- Usando un bucle for, crea 1000 nombres e insértalos en el array usando su índice.
 - Genera los nombres usando el índice: Nombre_0, Nombre_1, Nombre_2, Nombre_3...
- Recorre el array y muéstralos en consola usando console.log

4

FUNCIONES

Declaración de funciones

- Una función es un conjunto de líneas de código que realizan una tarea específica y puede retornar un valor.
- Las funciones pueden tomar parámetros que modifiquen su funcionamiento.
- En javascript se declaran:

```
function nombreFuncion (parámetroA, parámetroB, ...) {  
    código de la función...  
    return resultado;  
}
```

- Por ejemplo:

```
function suma(a,b) {  
    var c= a+b;  
    return c;  
}
```

- Las funciones devolverán un resultado a través de la orden **return**
- Asimismo las funciones **pueden no devolver resultados** (**void**)

Invocación de funciones

- Para ejecutar las funciones hay que invocarlas (llamarlas).
- Para ello invocaremos su nombre con los parámetros correspondientes:

```
var miNumero = suma (2,3); // miNumero equivaldrá a 5
```



Pongámoslo en práctica

- Define una función que calcule la suma de los primeros números naturales de N.
- Ejemplo
 - N=3
 - Resultado: $3+2+1 = 6$



Pongámoslo en práctica

- Crear dos funciones:
 - Una para generar un array de N nombres de usuarios
 - Otra para mostrar todos los usuarios por consola
- Crear otra función que ponga a null 1 de cada dos nombres de usuario

Array de argumentos: arguments

- Permite acceder a los parámetros de la función, sin declararlos.
- Empiezan a contar a partir del cero.
- El tamaño viene definido por arguments.length
- Los parámetros son flexibles a la hora de ser llamados.

```
function suma() { //No declaramos argumentos
    var i, res= 0;
    for (i = 0; i < arguments.length; i++) {
        res += arguments[i];
    }
    return res;
}
```

Llamamos a la función con 2 argumentos

```
var resultado = suma(5, 23);
```



Pongámoslo en práctica

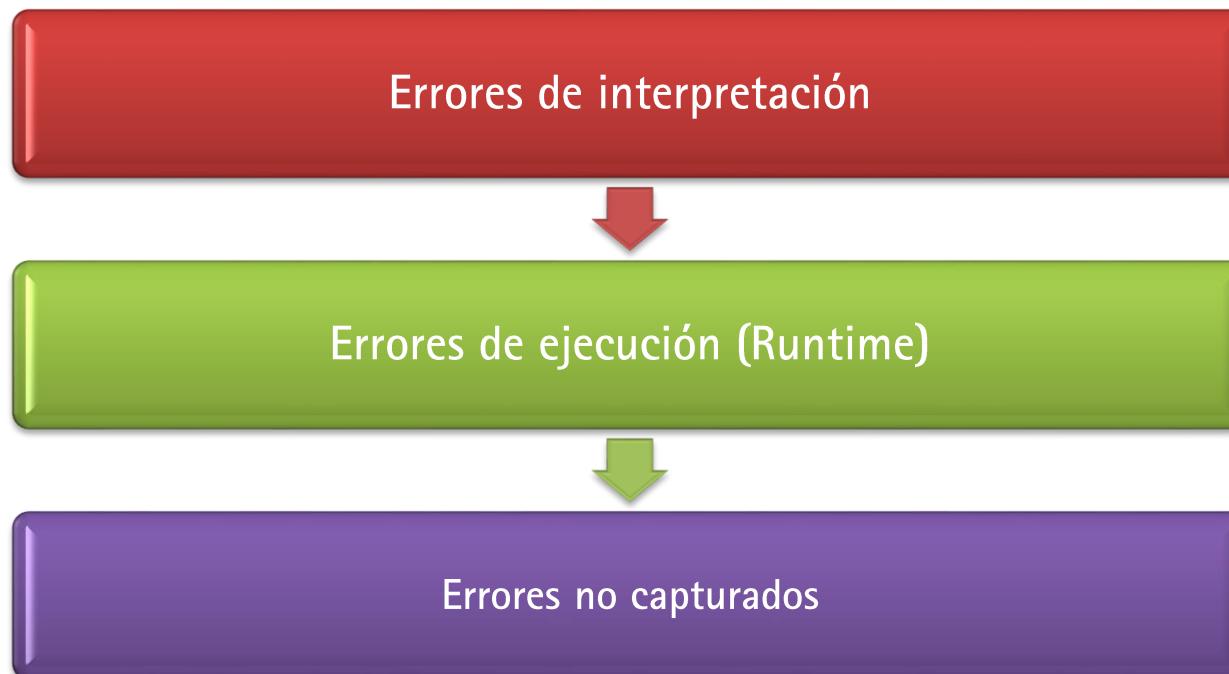
- Crea una función que concatene todos los textos que le pases como argumento.
- Ejemplo: mifuncion("hola", "que", "tal")
- Resultado: "hola que tal"

Funciones predefinidas

- eval // evalúa una expresión
- isFinite // determina si el parámetro es finito
- isNaN // determina si una variable es un número o no
- Number y String // convierte una expresión a número o cadena respectivamente
- escape y unescape // codifican y descodifican una cadena

Errores en el código

- Podríamos dividir los posibles errores que se pueden producir, en 3 categorías:



Gestión de errores

- JavaScript sigue el paradigma de la división de tareas
 - por una parte se concentra el esfuerzo en realizar una determinada tarea sin mezclar con código de gestión de errores.
 - La gestión de errores, cuando se produce, se desplaza fuera de la lógica de negocio.
- Para ello se usan las cláusulas try / catch / finally

```
try {  
    // Lo que quiero que se ejecute  
    ...  
} catch(error: Error) {  
    // Si hay error haré  
    ...  
} finally {  
    // Lo que se ejecutará en cualquier caso  
    ...  
}
```

Errores y depuración

- Los errores de ejecución, son aquellos que se producen al intentar ejecutar un script que es sintácticamente incorrecto.



- Estos errores pueden detectarse, mientras se está desarrollando, si disponemos de un entorno de trabajo integrado, como Eclipse. Netbeans...



Pongámoslo en práctica

- Crea una función “Calculador” a la cual se le pasarán dos parámetros:
 - Un texto con números separados por “,”: 1,2,3,4,5....
 - La operación a realizar (+,-, *, /)
- La función debe devolver como resultado la operación de todos los números y mostrarlo en consola.
- Ten en cuenta los posibles errores y excepciones
- Tip: puedes usar dos estrategias:
 - Reemplazar las “,” por la operación: cadena.replace(./, /g, '+');
 - Generar un array con los números: cadena.split(',');

Depurando Javascript

- El código puede contener errores de sintaxis o errores lógicos.
- Muchos de estos errores son difíciles de diagnosticar.
- Buscar errores en el código (y arreglarlos) se denomina **depuración (debugging)** de código.



Métodos de depuración

➤ console.log()

- Muestra valores JavaScript en la ventana del depurador

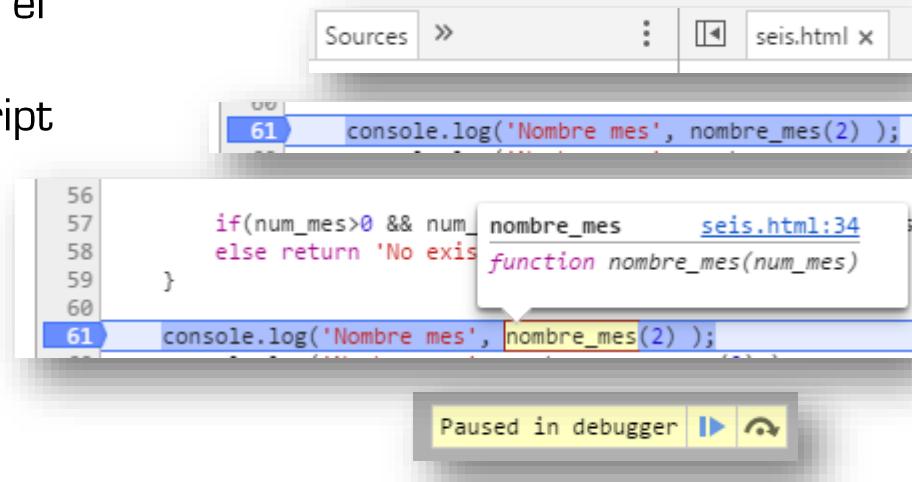
➤ Puntos de interrupción

- En la ventana del depurador (F12->sources -> click en línea), puede establecerse puntos de interrupción en el código.
 - En cada punto de interrupción, JavaScript dejará de ejecutarse y le permitirá examinar valores de JavaScript.
 - Despues de examinar los valores, puedes reanudar la ejecución del código (normalmente con el botón de play).

➤ Orden debugger

- La palabra clave debugger detiene la ejecución de JavaScript y llama (si está disponible) a la función de depuración.
 - Tiene el mismo efecto que establecer un punto de interrupción en el depurador.

```
a = 5;  
b = 6;  
c = a + b;  
console.log(c);
```



```
a = 5;  
b = 6;  
debugger;  
c = a + b;
```



Pongámoslo en práctica

- Depura “Calculadora” en sus puntos críticos.

5

OBJETOS JAVASCRIPT

Qué es un objeto

- Un **objeto** es una entidad autónoma que consiste en datos y procedimientos para manipular los datos.
- Por ejemplo, cualquier elemento que se puede seleccionar y manipular individualmente, como formas e imágenes que aparecen en una IU, así como entidades de software menos tangibles.
- Javascript dispone de una serie de objetos que envuelven los tipos básicos o primitivos (String, Number y Boolean).
- Normalmente se diferencian los tipos primitivos y los objetos, en que los primeros se escriben en minúsculas y los segundos son un constructor cuya primera letra se escribe en mayúscula.
 - var x='hola';
 - var y= String('hola');
- Además tenemos Date, un tipo de objeto muy común y del que no disponemos de un tipo primitivo como tal, ya que este objeto es la representación como fecha y hora de un valor numérico.

Objetos javascript

- Para crear un objeto necesitamos crear una referencia al mismo con el operador “new”:
 - » var datos = new Array();
 - » var fecha = new Date("1 Jan 2010");
- Puedo utilizar los métodos y propiedades de un objeto accediendo al mismo con el operador “.”:
 - » var elementos = datos.length;
 - » datos.sort();
- De hecho existen objetos para los tipos primitivos String, Number y Boolean:
 - » var info = new String("Saludos desde netmind");
 - » var c1 = new Number(123);

Objetos predefinidos

- Objeto Array.
- Objeto Boolean.
- Objeto Date.
- Objeto Function.
- Objeto Math.
- Objeto Number.
- Objeto RegExp.
- Objeto String.

➤ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

Métodos de Array

- concat // concatenación
- join // elementos en un String
- pop // saca el ultimo elemento
- push // añade al final
- reverse // da la vuelta al array
- shift // saca el primer elemento
- slice // saca una sección
- splice // añade o borra
- sort // ordena el array
- unshift // añade al principio

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Métodos de String

➤ **charAt(indice)**

Devuelve el carácter que hay en la posición indicada como índice. Las posiciones de un string empiezan en 0.

➤ **indexOf(carácter,desde)**

Devuelve la posición de la primera vez que aparece el carácter indicado por parámetro en un string. Si no encuentra el carácter en el string devuelve -1. El segundo parámetro es opcional y sirve para indicar a partir de qué posición se desea que empiece la búsqueda.

➤ **lastIndexOf(carácter,desde)**

Busca la posición de un carácter exactamente igual a como lo hace la función indexOf pero desde el final en lugar del principio. El segundo parámetro indica el número de caracteres desde donde se busca, igual que en indexOf.

➤ **replace(substring_a_buscar,nuevoStr)**

Implementado en Javascript 1.2, sirve para reemplazar porciones del texto de un string por otro texto

Métodos de String

➤ **split(separador)**

Este método sólo es compatible con javascript 1.1 en adelante. Sirve para crear un vector a partir de un String en el que cada elemento es la parte del String que está separada por el separador indicado por parámetro.

➤ **substring(inicio,fin)**

Devuelve el substring que empieza en el carácter de inicio y termina en el carácter de fin. Si intercambiamos los parámetros de inicio y fin también funciona. Simplemente nos da el substring que hay entre el carácter menor y el mayor.

➤ **toLowerCase()**

Pone todas los caracteres de un string en minúsculas.

➤ **toUpperCase()**

Pone todas los caracteres de un string en mayúsculas.

➤ **toString()**

Este método lo tienen todos los objetos y se usa para convertirlos en cadenas.

Métodos de Date

- Para cada atributo de la clase Date hay:
 - get
 - set // no hay setDay es setDate
 - to
- Ver la especificación de JavaScript para más detalle.

Métodos de Math

- abs
- sin, cos, tan
- acos, asin, atan
- exp, log
- ceil, floor
- min, max
- pow
- round
- sqrt



Pongámoslo en práctica

- Crea una función que emule los botones de word:
 - Todo mayúsculas
 - Tipo oración
 - Primera letra de cada palabra en mayúsculas

- Esta función recibirá como parámetros:
 - El texto a procesar
 - La transformación a realizar

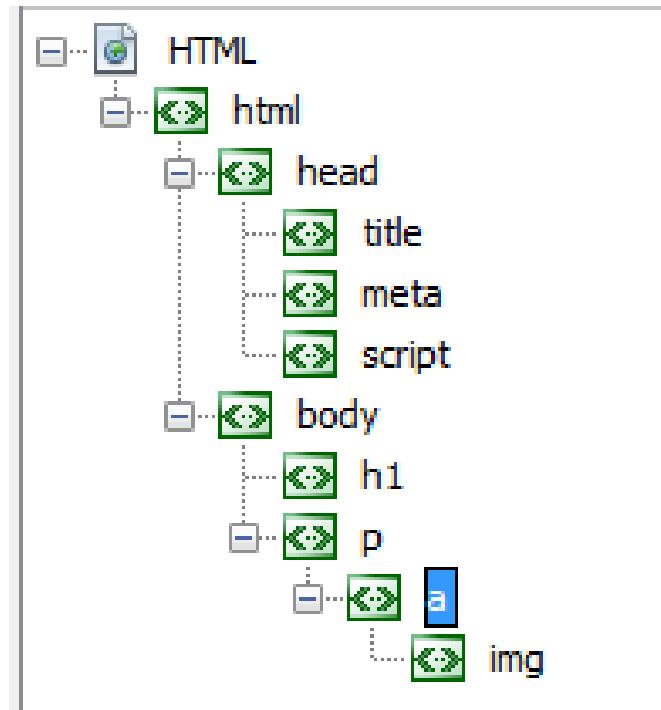
6

INTERACCIÓN CON EL NAVEGADOR (DOM)

El objeto DOM

- El modelo de objetos del documento (DOM: Document Object Model) permite a un desarrollador acceder o representar elementos de una página web, accediendo a sus propiedades y métodos.

- Los elementos HTML de una página web representados por DOM están organizados de una forma jerárquica: estructura de árbol

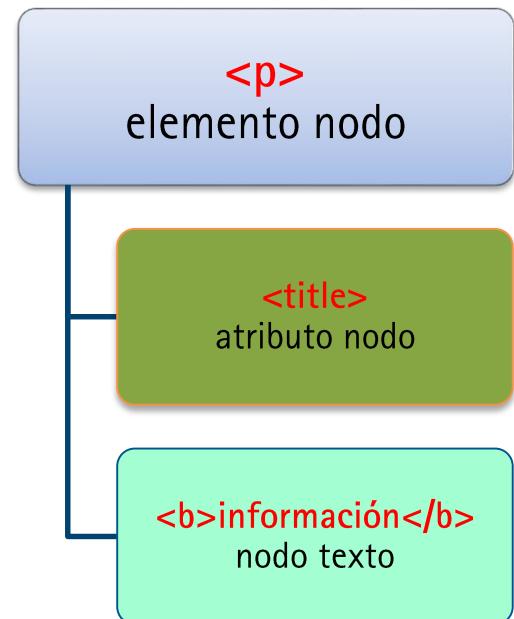


https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

<https://developer.mozilla.org/en-US/docs/Web/API>

Conociendo el objeto DOM

- El árbol DOM está compuesto de un conjunto de “piezas”, como:
 - Nodos
 - Elementos nodo
 - Elemento texto
 - Atributos nodo
- El objeto DOM está compuesto de una colección de nodos, existen diferentes tipo de nodos, algunos nodos contienen a su vez otros nodos.
- Los elementos HTML, son en sí mismos nodos, aunque no contengan información.



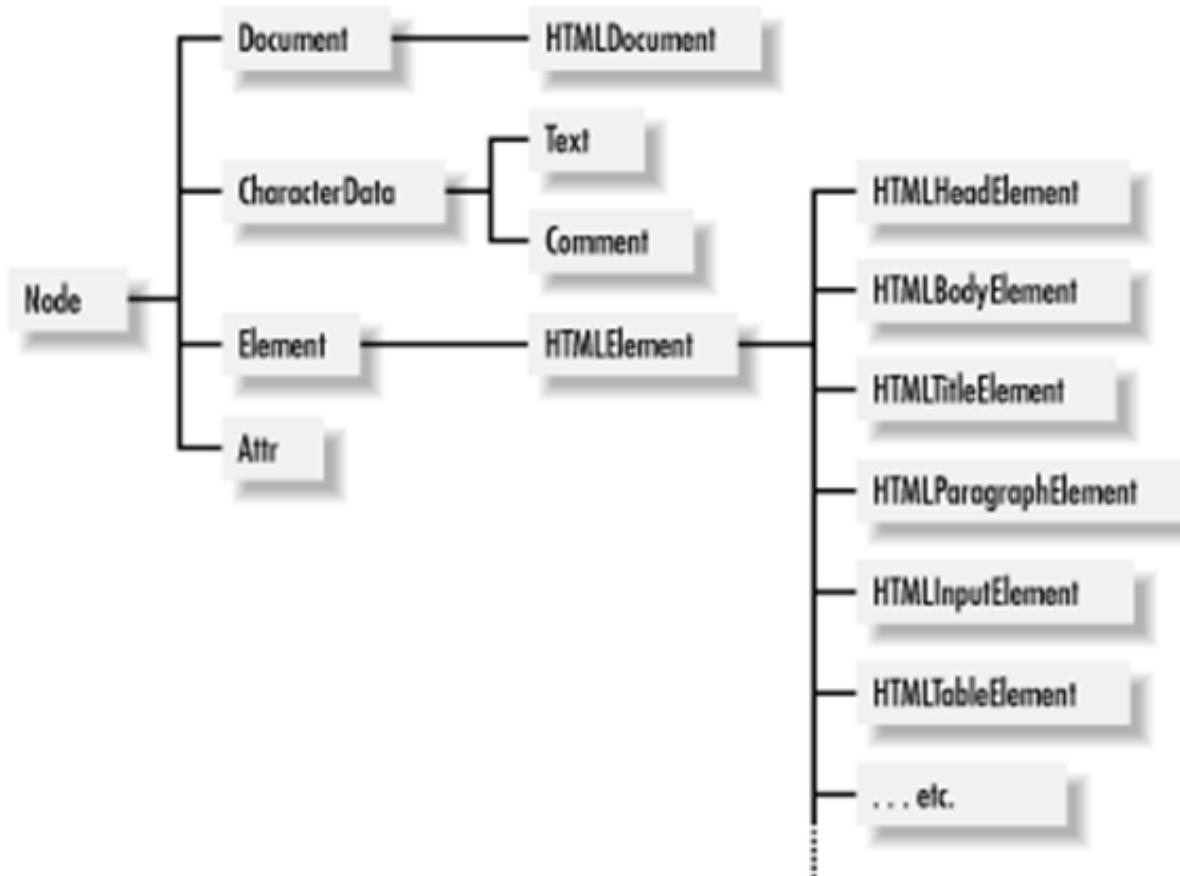
Tipos más habituales de nodos

- Node dispone de subinterfaces que representan diferentes tipos de nodos que pueden formar parte de la estructura DOM, como por ejemplo:

Tipos	Valor de la constante	nodeType value
Element	Node.ELEMENT_NODE	1
Text	Node.TEXT_NODE	3
Document	Node.DOCUMENT_NODE	9
Comment	Node.COMMENT_NODE	8
DocumentFragment	Node.DOCUMENT_FRAGMENT_N ODE	11
Attr	Node.ATTRIBUTE_NODE	2

Conociendo el objeto DOM

➤ Tipos más habituales de nodos





Pongámoslo en práctica

- Genera una página web HTML5 que contenga:
 - Una cabecera de navegación
 - Un artículo con enlaces e imágenes
 - Un formulario de comentarios: nombre, mail, edad, comentario

- Explora la página con la herramienta de desarrollador del navegador

DOM - Propiedades

- El DOM es accesible vía **window.document** o simplemente **document**
- Este objeto a su vez presenta una API de métodos y propiedades

Property/Method	Descripción	W3C
anchors[]	Returns an array of all the anchors in the document	Yes
forms[]	Returns an array of all the forms in the document	Yes
images[]	Returns an array of all the images in the document	Yes
links[]	Returns an array of all the links in the document	Yes
cookie	Returns all name/value pairs of cookies in the document	Yes
domain	Returns the domain name of the server that loaded the document	Yes
lastModified	Returns the date and time the document was last modified	No
readyState	Returns the (loading) status of the document	No
referrer	Returns the URL of the document that loaded the current document	Yes
title	Sets or returns the title of the document	Yes
URL	Returns the full URL of the document	Yes
innerHTML	Return the body of the tag.	

DOM - Métodos

- **document** posee un conjunto de métodos que nos permite acceder a un nodo específico, escuchar eventos o realizar acciones

Métodos	Descripción	W3C
getElementById()	Accesses the first element with the specified id	Yes
getElementsByName()	Accesses all elements with a specified name	Yes
getElementsByTagName()	Accesses all elements with a specified tagname	Yes
getElementsByClassName()	Accesses all elements with a specified class	Yes
open()	Opens an output stream to collect the output from document.write() or document.writeln()	Yes
write()	Writes HTML expressions or JavaScript code to a document	Yes
writeln()	Same as write(), but adds a newline character after each statement	Yes
close()	Closes the output stream previously opened with document.open()	Yes



Pongámoslo en práctica

- Para tu página web:
 - Obtén todos sus enlaces y muestra los ids en un div resultados_a
 - Para todos los input de un formulario, obtén sus valores y muéstralos en resultados_a

Creando elementos DOM

➤ Método create Element

- Usaremos el método createElement para crear un elemento ('div', 'li', 'span', ...)
- Luego lo añadiremos al nodo que nos interese

```
var newNode = document.createElement('span');
node.appendChild(newNode);
```

➤ Con innerHTML

- Cuando el navegador encuentra un string con tags html dentro, los interpreta y añade al DOM

```
node.innerHTML = 'string con tags html';
```

```
node.innerHTML = '<span>Esto es un span:</span>';
```

Creando nuevas ventanas

- Es posible abrir nuevas ventanas o pestañas usando Javascript
 - <https://developer.mozilla.org/en-US/docs/Web/API/Window>

```
var newW = window.open(URL,name,specs,replace);
newW.document.write(<text>);
```

- Para cerrarla usaremos:
 - newW.close() // usando la referencia del objeto creado
 - self.close() // dentro de la propia ventana



Pongámoslo en práctica

- Para una página web:
- Añade dos elementos li (con cuerpo) a un ul
 - Con innerHTML
- Obtén el cuerpo de un section y múestralo en una ventana nueva.

Cambiando estilos

- Podemos modificar los estilos de un elemento usando Jasvascript
- Existen dos aproximaciones:
 - Acceder a las propiedades de estilo mediante style

```
Node.style.[color|height|width|border...]=<valor>;
```

- Modificar la clase del elemento (preferida)
 - En el css definiremos estilos para situaciones normales y para situaciones para las que queremos cambiar

```
Node.className =<nombre_de_clases>;  
var elementClasses = Node.classList.[add(<class>)|remove()|contains()];
```

- http://www.w3schools.com/jsref/dom_obj_style.asp



Pongámoslo en práctica

- Para una página web:
- Crea un css básico para darle estilo a tu página
- Cambia los estilos del título del artículo:
 - Usando la propiedad style
 - Asignándole una clase

7

EVENTOS

Eventos en Javascript

- Las aplicaciones web creadas con JavaScript pueden utilizar el modelo de programación basada en eventos.
- En este tipo de programación, los scripts se dedican a esperar a que el usuario "haga algo" (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador).
- A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.

Definir un evento

- En JS podemos asociar un evento a cualquier elemento del DOM
- Existen varias aproximaciones:
 1. En **línea** en la etiqueta de HTML que se quiera:

```
<input type="button" value="Calcula" onClick="calcular(2,3,4,5)">
```

- Es importante escapar los caracteres. Por ejemplo si es un string usar comillas simples ''.
- **Este método de asociación no se recomienda!** Ya que acopla HTML y JS

2. Mediante una función de **callback**

```
boton.onclick = function(e){...}
```

3. Mediante un **listener**

```
boton.addEventListener('click', function(){...}, false);
```

Definir un evento – atributo de evento

- Podemos programar un evento asociación con funciones anónimas de callback.
- Estas se llamarán cuando suceda el evento

```
<button id="mi_boton">Enviar</button>
```

```
document.getElementById('mi_boton').onclick = function(e){  
    ....  
}
```

- Cuando asociamos un evento a un elemento, podemos acceder al propio elemento dentro de la función de callback con la palabra reservada **this**

```
<div id="mi_div">Texto del div</div>
```

```
document.getElementById('mi_div').onclick = function(e){  
    var elcuerpo= this.innerHTML; //devolverá el cuerpo del propio div  
    ....  
}
```

Modelo básico de eventos

Atributo	Descripción	IE	F	O	W3C
onblur	An element loses focus	3	1	9	Yes
onchange	The content of a field changes	3	1	9	Yes
onclick	Mouse clicks an object	3	1	9	Yes
ondblclick	Mouse double-clicks an object	4	1	9	Yes
onerror	An error occurs when loading a document or an image	4	1	9	Yes
onfocus	An element gets focus	3	1	9	Yes
onkeydown	A keyboard key is pressed	3	1	No	Yes
onkeypress	A keyboard key is pressed or held down	3	1	9	Yes
onkeyup	A keyboard key is released	3	1	9	Yes
onmousedown	A mouse button is pressed	4	1	9	Yes
onmousemove	The mouse is moved	3	1	9	Yes
onmouseout	The mouse is moved off an element	4	1	9	Yes

Modelo básico de eventos

Atributo	Descripción	IE	F	O	W3C
onmouseover	The mouse is moved over an element	3	1	9	Yes
onmouseup	A mouse button is released	4	1	9	Yes
onresize	A window or frame is resized	4	1	9	Yes
onselect	Text is selected	3	1	9	Yes
onunload	The user exits the page	3	1	9	Yes

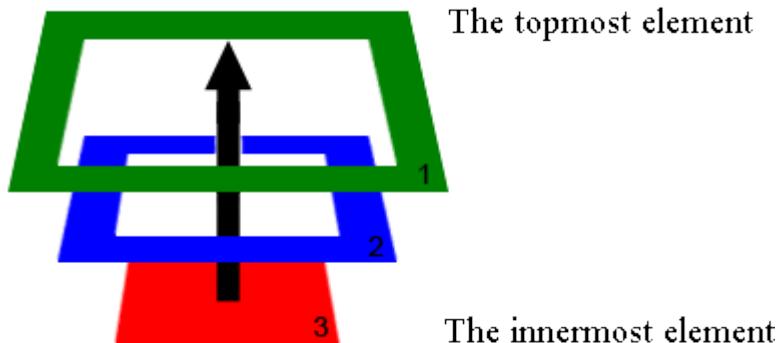


Pongámoslo en práctica

- Asocia un evento click a botones: colorea, blanco-negro, normal
- Estos harán los efectos respectivos cambiando el aspecto de un elemento de tipo section (coloreándolo, poniendo en blanco y negro y en modo normal).

Event Bubbling

- En este modelo de flujo de eventos se produce primero el evento en el elemento más interno de la estructura de árbol y va subiendo jerárquicamente hasta llegar al nodo raíz



- Para evitar la propagación usar el método stopPropagation sobre el evento

```
elem.onclick=function(evnt){  
    ...  
    evnt.stopPropagation();  
}
```

Atributos de evento para ratón y teclado

- Detectando eventos de ratón y teclado, según versiones

Atributo	Descripción	IE	F	O	W3C
altKey	Returns whether or not the "ALT" key was pressed when an event was triggered	6	1	9	Yes
button	Returns which mouse button was clicked when an event was triggered	6	1	9	Yes
clientX	Returns the horizontal coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
clientY	Returns the vertical coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
ctrlKey	Returns whether or not the "CTRL" key was pressed when an event was triggered	6	1	9	Yes
metaKey	Returns whether or not the "meta" key was pressed when an event was triggered	6	1	9	Yes

Atributos de evento para ratón y teclado

- Detectando eventos de ratón y teclado, según versiones

Atributo	Descripción	IE	F	O	W3C
relatedTarget	Returns the element related to the element that triggered the event	No	1	9	Yes
screenX	Returns the horizontal coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
screenY	Returns the vertical coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
shiftKey	Returns whether or not the "SHIFT" key was pressed when an event was triggered	6	1	9	Yes

Otros atributos de eventos

➤ Resto de atributos de eventos

Atributo	Descripción	IE	F	O	W3C
bubbles	Returns a Boolean value that indicates whether or not an event is a bubbling event	No	1	9	Yes
cancelable	Returns a Boolean value that indicates whether or not an event can have its default action prevented	No	1	9	Yes
currentTarget	Returns the element whose event listeners triggered the event	No	1	9	Yes
eventPhase	Returns which phase of the event flow is currently being evaluated				Yes
target	Returns the element that triggered the event	No	1	9	Yes
timeStamp	Returns the time stamp, in milliseconds, from the epoch (system start or event trigger)	No	1	9	Yes
type	Returns the name of the event	6	1	9	Yes

Listeners de eventos

- Los métodos addEventListener y removeEventListener, permiten vincular eventos a objetos dinámicamente

```
function holaMundo() { alert('Hola mundo.');" }  
  
var boton = document.getElementById('boton');  
  
boton.addEventListener('click', holaMundo, false);  
boton.removeEventListener('click', holaMundo, false);  
  
<input type="button" value="Pulsa" id="boton" />
```

8

VALIDACIÓN DE FORMULARIOS

Validar formularios con JavaScript

- JavaScript reduce la carga en el servidor ya que si se envían los datos ya validados, el servidor solo tiene que validarlos una vez y no tiene que estar mandando páginas de error constantemente.
 - Los datos han de validarse SIEMPRE también en el servidor.
- Se pueden validar datos de dos maneras:
 - Dinámicamente, por ejemplo con eventos **onClick**, **onChange**.
 - Antes del envío de datos del formulario mediante un **onSubmit**.
- Para evitar el comportamiento por defecto de botones en un formulario (y en general de cualquier elemento, como enlaces), se puede devolver en la función callback del evento un false o prevenir dicho el comportamiento:

```
elem.onclick=function(){  
    ....  
    return false;  
}
```

```
elem.onclick=function(){  
    ....  
    event.preventDefault();  
}
```

Las API de Validación

- Un nuevo conjunto de mecanismos para validación basados en API
- El objeto fundamental para esa estructura es el **ValidityState**
- Cada elemento de un formulario puede llamar al método **checkValidity**, y analizar su contenido con la idea de mostrar información pertinente de validación al usuario
- Este objeto se expone a través del atributo **validity** que contiene cada formulario

```
document.getElementById('mi_imput').validity;
```

- Puede adoptar una lista de valores dependiendo del tipo de error de validación que se haya producido

Las API de Validación

- Los atributos de **validity** son:
 - valid (de manera global si el formulario campo es válido)
 - valueMissing (campo requerido)
 - typeMismatch (error de tipo)
 - patternMismatch (patrón incorrecto)
 - tooLong (demasiado largo)
 - rangeUnderflow (rango por debajo de lo esperado)
 - rangeOverflow (rango excedido)
 - stepMismatch (paso intermedio incorrecto)
 - customError (error personalizado)
- Estos valores pueden ser consultados con posterioridad utilizando funciones JavaScript

Las API de Validación

```
<label>Tarjeta de Crédito<input name="f" id="f" type="text" /></label>
<script>
document.getElementById('f').oninput=function check() {
    if (this.value == "4B" ||
        this.value == "A.Express" ||
        this.value == "Visa") {
        this.setCustomValidity("'" + this.value +"' no es un tipo de tarjeta válido.');
    } else {
        // Si la entrada es correcta borramos el mensaje
        this.setCustomValidity("");
    }
}
</script>
```



Pongámoslo en práctica

- Crea un formulario con los siguientes campos:
 - Nombre (solo texto y min size 3)
 - Apellido (solo texto y min size 3)
 - Edad (número entre 14 y 100)
 - Email (tipo email o patrón email /S+@\S+\.\S+/)
- Añade al formulario un botón para enviar los datos. Al hacer clic en el botón se debe verificar el tipo de datos que se han introducido.
- Si alguno no es válido, se debe mostrar un mensaje de error en un div. Si todo es correcto, enviar el formulario.



[...]**netmind**

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es



GOBIERNO
DE ESPAÑA

MINISTERIO
DE ENERGÍA, TURISMO
Y AGENDA DIGITAL

red.es



ESTRATEGIA DE
EMPRENDIMIENTO Y
EMPLEO JUVENIL
garantía juvenil



Agenda Digital para España



UNIÓN EUROPEA

Fondo Social Europeo
“El FSE invierte en tu futuro”