

JavaScript

HTML
CSS

Javascript



01

Introducción



MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro

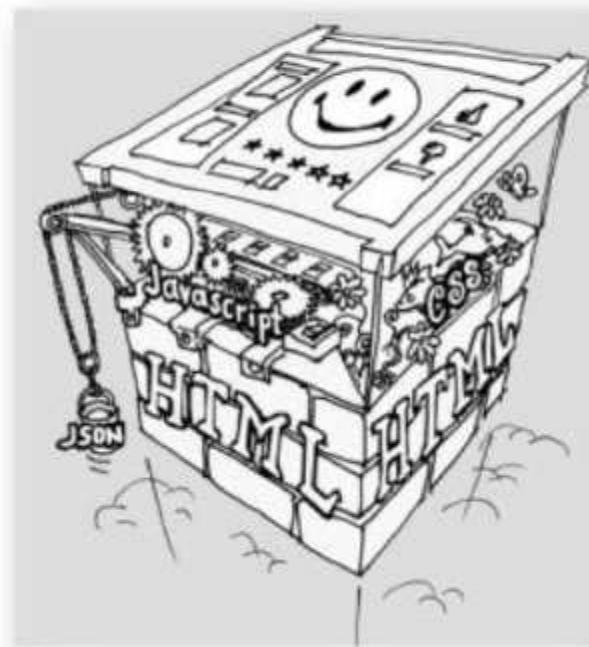
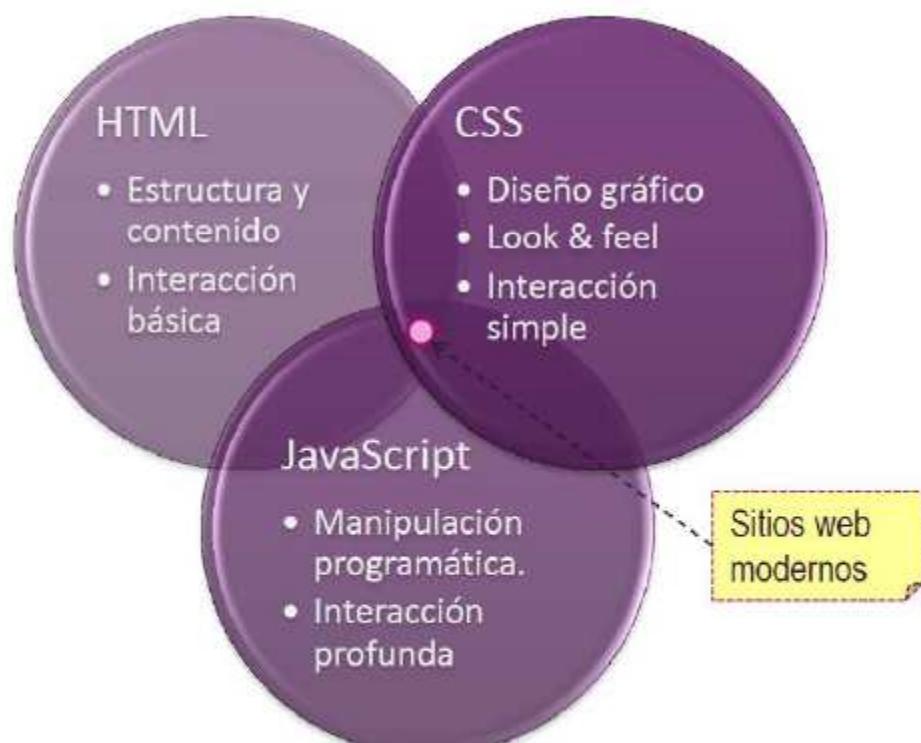




- Creado por Netscape y llamado originalmente Livescript, pero cambió a JavaScript ya que tenía también soporte para Java (marketing).
- Utiliza una especificación del lenguaje llamada ECMAScript.
- Muy utilizado para las operaciones contra el DOM.
- Hoy en día se utiliza ya tanto en lado cliente como en lado servidor.
- Interpretado.
- Imperativo y estructurado.
- Orientado a objetos* (prácticamente todo es un objeto).
- Basado en prototipos.
- Dinámico y débilmente tipado (todas las variables son 'var').



Historia



JavaScript

Junto con HTML

¿Cómo utilizarlo?

Uso en el Navegador

- Etiqueta `<script>`
 - Dentro del documento HTML, incluyendo el código entre la apertura y el cierre de la etiqueta

```
<script>
  // Instrucciones JavaScript
</script>
```

- Referenciando a un fichero externo → atributos `async`, `defer`

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

- Manejador de un evento → `onclick`, `onmouseover`

```
<button onclick="nombreFuncionJavaScript()" />
```

- Protocolo URL → javascript:
`Validar`

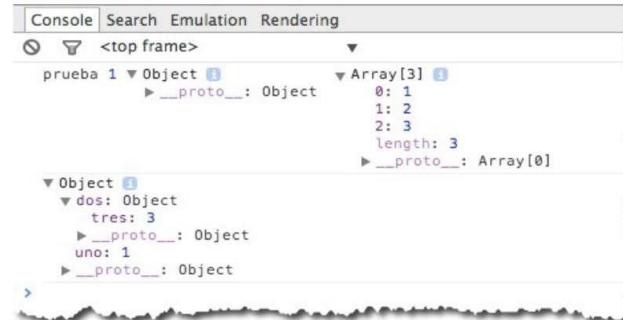
JavaScript

Junto con HTML

Mensajes por consola

- Para escribir en la consola, usaremos el objeto `console` mediante diferentes métodos que reciben como parámetros el elemento que queremos mostrar:
 - `log()`
`info()`, `warn()`, `error()`: diferentes niveles de log
 - `dir()`: enumera las propiedades
 - `assert()`: aserciones
 - `time()` y `timeEnd()`: rangos de tiempo

```
console.log("prueba", 1, {}, [1,2,3]);
console.dir({uno: 1, dos: {tres: 3}});
```



Mensajes por consola

Ejercicio1

Crear una plantilla Html5 llamada **ejercicio1.html** que use un Script Javascript **js/ejercicio1.js** y realizar las siguientes tareas :

- a) Mostrar un mensaje de información por consola : MENSAJE DE INFO
- b) Mostrar un mensaje de Warning por consola : MENSAJE DE WARNING
- c) Mostrar un mensaje de error por consola : MENSAJE DE ERROR
- d) Mostrar un mensaje de Log por consola : MENSAJE DE LOG
- e) Mostrar un mensaje de tiempo por consola

Sintaxis

- Las variables son definidas mediante identificadores que deben cumplir las siguientes reglas:
 - Secuencia alfanumérica de caracteres cuyo primer carácter no sea numérico.
 - Se puede utilizar símbolos como '\$' y '_', excepto los que tienen significado, como '%' ó '?'.
 - Mejor evitar acentos debido a problemas de portabilidad.
- Existe un conjunto de palabras reservadas, que son utilizadas por el lenguaje:

break case catch class debugger default delete do
else export extends finally import instanceof long
new null return switch this throw true try typeof
var while

... entre otras.



Tipos de datos

- Una **variable** es una referencia a un **valor** o a un **objeto** de un tipo dado.

```
var miVariable;  
var v1, v2, v3;
```

(Se aconseja la segunda forma, declarar de forma conjunta varias variables)

- Podemos definir variables y asignándole valor:

```
var numero = 69,  
mensaje = "Hola Sema!",  
enabled = true,  
hoy = new Date();
```



Tipos de datos

- JavaScript cuenta con tipos de datos primitivos agrupados por categorías:
 - Números
 - Booleanos
 - Cadenas de texto
 - Funciones
 - Objetos
 - `undefined`
- Además, JavaScript cuenta con el operador `typeof(x)` que permite averiguar el tipo de dato de una variable.



Numéricos

- Los números podrán ser escritos con y sin punto decimal. También podremos utilizar la forma exponencial.

```
var x1 = 69;          // Sin decimales
var x2 = 56.00;       // Con decimales
var x3 = 283e5;        // 28300000
var x4 = 283e-5;       // 0.00283
```



Numéricos

- Clase Number:

```
// Uso básico
var variable1 = new Number(16); // Entero
var variable2 = new Number(3.141592); // Decimal

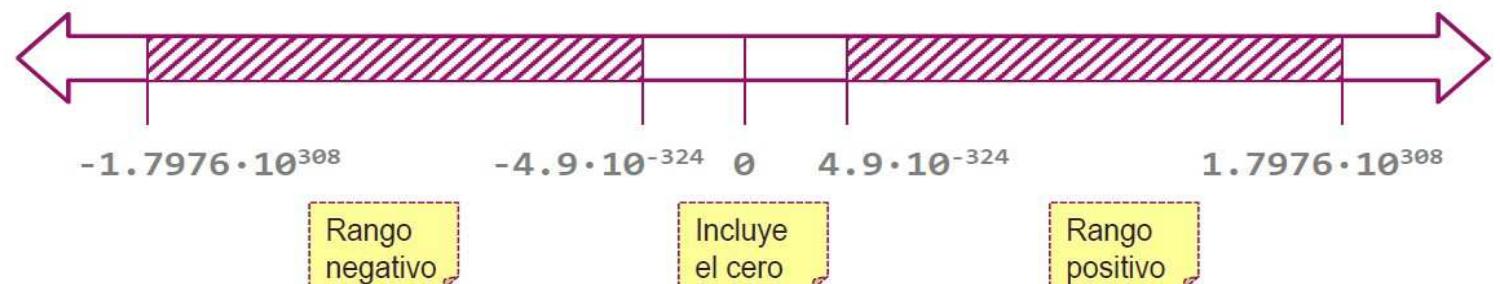
// Para obtener el valor numérico almacenado, se puede utilizar el método valueOf():
var variable1 = new Number(16);
var variable2 = variable1.valueOf(); // variable2 = 16

// Uno de los métodos más útiles para los números es toFixed(), que trunca el número de
// decimales de un número al valor indicado como parámetro:
var variable1 = new Number(3.141592);
var variable2 = variable1.toFixed(); // variable2 = 3
var variable3 = variable1.toFixed(2); // variable3 = 3.14
var variable4 = variable1.toFixed(10); // variable4 = 3.1415920000
```



Numéricos

- JavaScript maneja variables de tipo numérico (number). Internamente se representan como un número de punto flotante de 64 bits, lo que permite doble precisión y el rango siguiente:



Ejercicio2

Crear una plantilla Html5 llamada **ejercicio2.html** que use un Script Javascript **js/ejercicio2.js** y realizar las siguientes tareas :

- a) Declarar una variable numérica llamada *alto* con valor 12 y otra variable numérica llamada *ancho* con valor 12.203004
- b) Declarar una variable numérica llamada *altoPorAncho* con valor resultado de multiplicar la variable *alto* por la variable *ancho*
- c) Mostrar por consola el valor de la variable *altoPorAncho*
- d) Mostrar por consola el valor de la variable *altoPorAncho* redondeando a 2 decimales

JavaScript

Variables

Cadenas de texto

- Se define mediante comillas, bien sean simples o dobles.
- Ambas pueden ser intercambiables, siempre y cuando se respete el orden de cierre de las mismas.
- Internamente, son una cadena de caracteres sin un máximo establecido.
- Este objeto tiene una serie de métodos asociados, como por ejemplo, uno de los más conocidos: `string.length`.



```
var cad = "", cad2 = '';
var cadena2 = "Esto es una 'cadena'";
var cadena3 = 'Y "esto" también';
console.log(cad2.length + ", " + cadena2.length);
```

JavaScript

Variables

Cadenas de texto

- Se define mediante comillas, bien sean simples o dobles.
- Ambas pueden ser intercambiables, siempre y cuando se respete el orden de cierre de las mismas.
- Internamente, son una cadena de caracteres sin un máximo establecido.
- Este objeto tiene una serie de métodos asociados, como por ejemplo, uno de los más conocidos: `string.length`.



Cadenas de texto

- Para buscar la primera posición de un string dentro de otro, se utiliza `indexOf`, y la última, con `lastIndexOf`:

```
str = "ser o no ser, esa es la cuestión";
```

↑↑↑↑↑↑↑↑↑↑
0123456789 18 26

```
str.indexOf("ser");
```

→ 0

```
str.lastIndexOf("es");
```

→ 26

```
str.indexOf("ser", 3);
```

→ 9

```
str.lastIndexOf("es", 25);
```

→ 18

```
str.indexOf("hamlet");
```

→ -1

-1 cuando no lo encuentra



JavaScript

Variables

Cadenas de texto

- indexOf se utiliza también para saber si un string contiene una cadena:

```
str = "Cinema Paradiso"; str.indexOf("a P") != -1 ➔ true
```

- Para extraer un carácter en una posición, se utiliza charAt:

```
str = "Cinema Paradiso"; str.charAt(2) ➔ 'n'
```

0 a n-1

- Para extraer un trozo de una cadena, se utiliza substring:



```
Desde 2, largo 4
str = "casablanca";
str.substring(2, 6) ➔ "sabl"
Restar: 6 - 2 = 4
```

Ubicar posición 2

0123456789

The diagram shows the string "casablanca" with arrows pointing from each character to its index (0 to 9). A yellow box highlights "Ubicar posición 2" (Locate position 2) with an arrow to index 2. Another yellow box highlights "Desde 2, largo 4" (From 2, length 4) with an arrow to index 6. A third yellow box highlights "Restar: 6 - 2 = 4" (Subtract: 6 - 2 = 4) with an arrow to index 4. A circled minus sign is placed between index 2 and index 6. The result "sabl" is shown in a yellow box.

Cadenas de texto

- Para pasar el contenido todo a minúsculas, se utiliza `toLowerCase()`, y a mayúsculas es con `toUpperCase()`:

```
str = "Artificial Intelligence";  
  
str.toLowerCase() → "artificial intelligence"  
str.toUpperCase() → "ARTIFICIAL INTELLIGENCE"
```

- Para comparar dos textos, se utiliza `"=="`:

```
var str1 = "euro";  
var str2 = "\u0065\u0075\u0072\u006f";  
  
str1 == str2 → true
```

"euro" en unicode



Cadenas de texto

- Para comparar dos textos, ignorando mayúsculas y minúsculas (case-insensitive), se deben pasar a mayúsculas o minúsculas y utilizar "==":

```
str1 = "lower or UPPER";
str2 = "Lower or Upper";
str1.toLowerCase() == str2.toLowerCase() → true
```

- Si se compara un texto y un número con el mismo valor, sucede lo siguiente:



```
str1 = "12";
str2 = 12;
```

| | | |
|-----------------------|---------|-------------------------|
| str1 == str2 | → true | Compara valores |
| str1 === str2 | → false | Compara valores y tipos |
| str1 === String(str2) | → true | |
| Number(str1) === str2 | → true | |

Cadenas de texto

- Para reemplazar en un String las ocurrencias de una cadena de caracteres por otra, se utiliza el método replace:

```
name = "les aventures de tintin et milou";
```

Por default, sólo reemplaza la primera ocurrencia.

```
name = name.replace("tin", "tan"); → "les aventures de tantin et milou"
```

- El método replace admite expresiones regulares (se ven más adelante), lo que flexibiliza su uso:

```
name = "Déjà vu";
```

Encuentra las minúsculas de la "a" a la "z"

```
name.replace(/[a-z]/, "_");
```

Reemplaza la primera por un "

```
"Dé_à vu"
```

```
name.replace(/[a-z]/g, "_");
```

Reemplaza todas las minúsculas por un "

```
"Dé_à __"
```

```
name.replace(/[a-z]/gi, "_");
```

Reemplaza todas, ignorando mayúsculas y minúsculas

```
"_é_à vu"
```



Cadenas de texto

```
var str = "Esto es " + "un string";
```

```
s1 = 1 + 3 + "5" + "7"; // 457
```

Regla de izquierda a derecha.
Suma 1 + 3, y luego concatena "5" y "7".

```
s2 = 1 + (3 + "5") + 7; // 1357
```

Calcula el paréntesis primero.
Concatena 1, el resultado, y 7.

```
s3 = "1" + (3 + 5) + 7; // 187
```

Como el primer operando es un string,
comienza concatenando.

```
s4 = 1 + "3" + 5 + 7; // 1357
```

Como la primera operación incluye a un
string, comienza concatenando.

```
s5 = "" + 1 + 3 + 5 + 7; // 1357
```

Si se quieren concatenar sólo números,
se puede agregar un string vacío.

```
s6 = "" + (1 + 3 + 5 + 7); // 16
```

Si se quiere convertir una suma a string.

```
s6 = String(1 + 3 + 5 + 7); // 16
```



Ejercicio3

Crear una plantilla Html5 llamada **ejercicio3.html** que use un Script Javascript **js/ejercicio3.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Crear una variable de tipo cadena llamada cadena1 con valor CADENA1
- b) La posición del texto "ENA" en la variable cadena1
- c) El carácter de la última posición de la cadena
- d) El valor de la variable cadena1 convertido a minúsculas
- e) Sustituir en cadena1 "DE" por "ME"
- f) Crear una variable de tipo cadena llamada cadena2 con valor CADENA2 y concatenar cadena2 con cadena1

JavaScript

Carácteres especiales

Saltos de línea, tabulación, ...

- En JavaScript existe una representación específica para algunos caracteres especiales, que permite definirlos y utilizarlos:

| Carácter | Significado | Utilización |
|----------|----------------------------------|--|
| \n | newline (Nueva línea) | Para saltos de línea en manejo de cadenas |
| \r | carriage return (Salto de carro) | Idem anterior, en Windows. |
| \t | tab (tabulación) | Detectar o insertar tabulaciones |
| \" | comillas (también existe \'') | Colocar comillas en un String. Ej: <code>str = "Esto va \"entre comillas\"";</code> |
| \\ | escapa un \ | Para escribir un "\" en una cadena. Ej: <code>str = "Para newline se utiliza \\n";</code> |
| \b | backspace | Poco usado |
| \f | form feed | Poco usado |



Conversiones de tipo

- Si tenemos un dato de tipo texto, por ejemplo "1", y se quiere manejar como número, JavaScript provee funciones para convertir entre tipos de variables. Esto es útil porque al ser todas 'var', no es posible diferenciarlas por su tipo.

```
var strOne = "1";
var numOne = Number(strOne);

var strEnabled = "true";
var enabled = Boolean(strEnabled);
```

- También es posible obtener el tipo de la variable, como vimos antes, con el operador *typeof(x)*:

```
if (typeof numOne == "number") {
    console.log("numOne es un number");
}
```



Booleanos

- Sólo contendrán dos valores: **true** o **false**.
- Existen lo que se denominan valores falsos y verdaderos:
 - **Falsy values:**
 - false
 - null
 - undefined
 - "" // (cadena vacía)
 - 0 // (número cero)
 - NaN // (Not A Number)
 - **Truly values:** todos los demás y cualquier objeto se considerará *true*.



JavaScript

Variables

Objetos

- JavaScript dispone de ciertos objetos ya predefinidos, aunque nosotros podremos crear los nuestros personalizados.



```
// Array
var values = new Array();

// Fecha
var today = new Date();

// Objeto propio
var miObjeto = new Object();
```

Objetos

Los Objetos propios también se pueden definir de la siguiente manera



```
var persona = {  
    edad: '21',  
    nombre: 'Antonio',  
    apellido1: 'Martínez',  
    apellido2: 'García'  
};
```

Nombre del Objeto

Objeto definido entre llaves

Propiedades del Objeto

Las propiedades del objeto tienen el siguiente formato :

<nombre_propiedad> : <valor>

JavaScript

Variables

Objetos

Para acceder a la propiedad de un Objeto, usamos **persona.edad** o **persona['edad']**

Si queremos añadir una nueva propiedad a un objeto
Persona['nueva_propiedad']='NUEVO VALOR';



Las propiedades del objeto tienen el siguiente formato :

<nombre_propiedad> : <valor>

JavaScript

Variables

Ámbito

- Una variable tiene visibilidad y existe en el ámbito en el que se creó, es decir, el bloque que la contiene.



```
var count;  
  
function inc() {  
    count++;  
  
    var message = "La cuenta es " + count;  
  
    alert(message);  
}
```

Inicio bloque
Variable local
Fin bloque

Las variables que están fuera de un bloque son globales dentro de la página.

Si una variable se declara sin 'var', se considera global. Esto es **no recomendable**.

Undefined

- Una variable a la que no se le ha asignado un valor o que no ha sido inicializada, tendrá un **valor** de ***undefined***. El **tipo** también es “*undefined*” (cadena).
- *Undefined* significa “¡No lo sé (todavía)!“.
- También aplica para una variable que no exista.



```
var variable1;
typeof variable1; // devuelve "undefined"

typeof variable2; // devuelve "undefined", la variable2 no ha sido declarada
```

JavaScript

Variables

Null

- Tipo de dato muy similar a `undefined`, que se suele utilizar para representar objetos que todavía no existen.
- Se suele utilizar para “declarar” a una variable con un valor nulo (`null` es un literal).
- A diferencia de otros lenguajes, `null` se considera un valor.
- Diferencias:
 - `undefined`: la variable no es todavía de ningún tipo, indefinida.
 - `null`: la variable está “casi” tipificada, será de un tipo determinado, pero no tiene valor aún.



JavaScript

Variables

Null

- ¿Qué devuelven las siguientes sentencias?

```
typeof null          // Caso a
typeof undefined    // Caso b

null === undefined // Caso c
null == undefined  // Caso d
```



JavaScript

Variables

Null



```
var sema = null;
sema === null          // Ejemplo 1
sema == null           // Ejemplo 2
sema == undefined     // Ejemplo 3

var sema = undefined
sema == undefined      // Ejemplo 4
sema === undefined     // Ejemplo 5
typeof(sema) == "undefined" // Ejemplo 6
typeof(sema) == undefined // Ejemplo 7
typeof(sema) === "undefined" // Ejemplo 8
typeof(sema) === undefined // Ejemplo 9
```

JavaScript

Variables

Arrays

- ¿Qué devuelven las siguientes sentencias?

```
var nums = new Array();
nums[0] = 10;
nums[1] = 20;
nums[2] = 30;

// Otra forma
var nums = [10, 20, 30];

// Cacheamos el valor para evitar recálculos innecesarios
var length = nums.length;
for(var i=0; i<length; i++) {
    console.log("Valor " + i + ": " + nums[i]);
}
```



JavaScript

Variables

Arrays

- Disponemos de dos maneras de estructurar valores:
 - **Arrays:** listas ordenadas de elementos:
 - Representación mediante '[' y ']' => [1, 2, 3, 4]
 - Los valores de un array pueden ser de cualquier tipo... ¡Incluso una función!
 - Funciones típicas: length, push, pop, shift, unshift...
 - **Objetos:** colección/diccionario de propiedades (agregador):
 - Se representa mediante '{' y '}'.
 - Propiedades y métodos.
 - Una propiedad es un “espacio” con nombre propio que puede contener cualquier valor.
 - Se puede acceder como un array asociativo o a través del operador punto.



Ejercicio4

Crear una plantilla Html5 llamada **ejercicio4.html** que use un Script Javascript **js/ejercicio4.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Crear una variable de tipo cadena llamada cadena1 con valor CADENA1
- b) Crear una variable con el nombre cadena2 sin valor
- c) Mostrar el tipo de cadena1
- d) Mostrar el tipo de cadena2
- e) Con la sentencia if , si el tipo de cadena2 es undefined mostrar por consola el texto "SIN DEFINIR"
- f) Definir un Objeto llamado *cliente* añadir las siguientes propiedades con sus valores *nombre* = "PEPITO", *tel*="656666666"
- g) Añadir al Objeto *cliente* la propiedad *direccion*="C/ Salud,21"
- h) Recorrer todas las propiedades del Objeto *cliente* y mostrarlas por pantalla

02

Sentencias Operadores



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



JavaScript

Sentencias

If

```
if (<condición>) {  
    <instrucciones>  
} else {  
    <instrucciones>  
}
```

Ejemplo:

```
if (a > b) {  
    alert("A gana");  
} else {  
    alert("B gana");  
}
```



```
if (<condición1>) {  
    <instrucciones>  
} else if (<condición2>) {  
    <instrucciones>  
}  
  
//varias condiciones else if...  
  
} else if (<condiciónN>) {  
    <instrucciones>  
} else {  
    <instrucciones>  
}
```

JavaScript

Sentencias

Switch - case

```
switch (<expresión>){  
    case <literal1>:  
        <instrucciones>  
        break;  
    case <literal2>:  
        <instrucciones>  
        break;  
    //N valores...  
    case <literalN>:  
        <instrucciones>  
        break;  
    default:  
        <instrucciones>  
}
```

Cuando entra a un "case", continúa hasta el siguiente "break".

Ejemplo:

```
switch (estado){  
    case 0:  
        alert("Opción 0");  
        break;  
    case 1:  
        alert("Opción 1");  
    case 2:  
        alert("Opción 2");  
        break;  
    default:  
        alert("Ninguna");  
}
```

si estado es 0:
"Opción 0"
si estado es 1:
"Opción 1" ↪
"Opción 2"

Permite cualquier tipo de variable. Normalmente se utiliza Number y String.

Ya que no hay break en case 1.



Ejercicio5

Crear una plantilla Html5 llamada **ejercicio5.html** que use un Script Javascript **js/ejercicio5.js** y realizar las siguientes tareas :

Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Crear una variabe de tipo array llamada array1 con valor [1,2,3,4]
- b) Crear un pequeño programa que recorra el array y cuando encuentre el 1 muestre UNO , el 2 nuestre DOS, el 3 muestre TRES, el 4 muestre CUATRO Utilizar la instrucción Switch
- c) Introducir un 3 al final del array y visualizar el array
- d) Quitar los 3 últimos elementos del array y visualizar el array

JavaScript

Sentencias

While, do-While

```
while (<condición>) {  
    <instrucciones>  
}
```

```
do {  
    <instrucciones>  
} while (<condición>);
```

Ejemplo:

```
var a = 3;  
  
while (a > 0) {  
    alert(a);  
    a--;  
}
```



Ejemplo:

```
var a = 0;  
  
do {  
    alert(a);  
    a++;  
} while (a > 0);
```

Pasa al menos
una vez.

JavaScript

Sentencias

for

```
for (<declaración>; <condición>;  
<instrucción>) {  
    <instrucciones>  
}
```

Ejemplo:

```
for (var i = 0; i < 3; i++) {  
    alert(i);  
}
```



```
for (var <índice> in <iterable>) {  
    <instrucciones>  
}
```

Esto es un array

Ejemplo:

```
var v = [4,7,9,1];  
  
for (var a in v){  
    alert(v[a]);  
}
```

Ejercicio6

Crear una plantilla Html5 llamada **ejercicio6.html** que use un Script Javascript **js/ejercicio6.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- Cambiar todas las sentencias for del *ejercicio5* por el for iterable

JavaScript

Sentencias

Break y continue

- Existen dos elementos de control para los ciclos:
 - **break**: detiene el ciclo, continuando con las instrucciones posteriores
 - **continue**: detiene la iteración, continuando con la siguiente.

```
for (var i = 0; i < 5; i++) {  
    if (i == 3) {  
        break;  
    }  
    console.log(i);  
}
```

¿Salida? => 0, 1, 2

```
for (var i = 0; i < 5; i++) {  
    if (i == 3) {  
        continue;  
    }  
    console.log(i);  
}
```

¿Salida? => 0, 1, 2, 4



JavaScript

Sentencias

Comentarios

```
//Comentario  
//de  
//línea
```

```
/*  
Comentario de bloque  
*/
```

Ejemplo:

```
var a = 56; //asign. básica
```



Ejemplo:

```
/*  
Con el siguiente comando se obtienen  
las propiedades del navegador  
*/  
var nav = window.navigator;
```

Operadores

- Un operador es un símbolo formado por uno o más caracteres, que permite realizar una determinada operación entre uno o más datos y produce un resultado.
- Operadores aritméticos:
 - Unarios: negativo (-) y positivo (+)
 - Binarios: suma (+), resta (-), producto (*), división (/) y módulo (%)
- Operadores relacionales:
 - igualdad (==). No confundir con =, que es para asignación.
 - desigualdad (!=)
 - mayor que (>)
 - menor que (<)
 - mayor o igual que (>=)
 - menor o igual que (<=)



Operadores

- Operadores incrementales:
 - Preincremento: `++variable`
 - Postincremento: `variable++`
- Operadores con cadenas de caracteres:
 - Concatenar: `+`
 - Equivalencia: `==`, `!=`
 - Comparación: `<`, `>`, `<=`, `>=`
- Operaciones con booleanos:
 - Combinación: `&&`, `||` (perezosos); `&`, `|` (no perezosos)
 - Negación (lo opuesto): `!`
 - mayor o igual que (`>=`)
 - menor o igual que (`<=`)
- Operador de asignación:
 - Asignaciones con aritmética: `+=`, `-=`, `*=`, `/=`, `%=`



Operadores

■ ¿Cuál es el valor de...?

- `true || false`
- `false || true`
- `true || "Cadena"`
- `false || "Cadena"`
- `"Cadena" || "Recursividad"`
- `"Cadena" || "" || "Recursividad"`
- `"Cadena" || "Recursividad" || ""`



Operadores

■ ¿Cuál es el valor de...?

- true **&&** false
- false **&&** true
- true **&&** “Cadena”
- false **&&** “Cadena”
- “Cadena” **&&** “Recursividad”
- “Cadena” **&&** “” **&&** “Recursividad”
- “Cadena” **&&** “Recursividad” **&&** “”



Operadores

- Operador **identidad**
- No realiza conversión de tipos
 -  → **sí** que realiza conversión de tipos

```
var verdadero = ("1" == true);
var falso = ("1" === true);
```

- Mejor usar el operador identidad para evitar la conversión de tipos
- Recuerda: **3 mejor que 2**



Operadores

- ¿Cuál es el valor de **a** y de **c** después de ejecutarse el siguiente código?

```
var a = "Er";
a += "ic";
var b = a > "John Lennon" || a + " Clapton";
a = b;
var c = !(a > b || a != b) && a != "Eric";
```



Operadores

- Único operador de JavaScript con tres operandos:

```
<condición> ? <expresión1> : <expresión2>
```

- Se evalúa “condición” y:
 - Si es **verdadera**, devuelve el resultado de evaluar expresión1.
 - Si es **falsa**, devuelve el resultado de evaluar expresión2.

```
max = (a > b) ? a : b;
```



JavaScript

Sentencias

Math

- Objeto con métodos estáticos para realizar operaciones matemáticas.
 - Potencia: `Math.pow(base, exp)`
 - Raíz cuadrada: `Math.sqrt(num)`
 - Redondear: `Math.round(num)`, `Math.ceil(num)`, `Math.floor(num)`
 - Mayor: `Math.max(num1, num2, ...)`
 - Menor: `Math.min(num1, num2, ...)`
 - Número real aleatorio entre 0 y 1: `Math.random()`
 - Número pi: `Math.PI`



Ejercicio7

Crear una plantilla Html5 llamada **ejercicio7.html** que use un Script Javascript **js/ejercicio7.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Crear dos Objetos : *pieza1* , *pieza2* con las propiedades *peso=20* para *pieza1* y *peso='20'* para *pieza2*
- b) Comprobar con el operador == si las propiedades *peso* de los objetos *pieza1* y *pieza2* son iguales
- c) Comprobar con el operador === si las propiedades *peso* de los objetos *pieza1* y *pieza2* son iguales
- d) Con el operador ternario , sumar el valor de las propiedades *peso* de los objetos *pieza1* y *pieza2* si la propiedad *peso* de *pieza1='20'*, multiplicar en caso contrario

03

Programación
Estructurada



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



JavaScript

Programación estructurada

Funciones

- Una función está compuesta por los siguientes elementos:
 - La palabra clave *function* que define a la función
 - Una lista de parámetros
 - El valor de retorno



```
function promptNumero (param1, param2, param3) {  
    // Cuerpo de la función  
    // ...  
    return undefined;  
}
```

JavaScript

Programación estructurada

Funciones

- ¿Qué devuelven estas funciones?

```
// Caso 1
function nada() {};
```

Nada

```
// Caso 2
function numeros() {
    return 6;
    return 7;
};
```

6



JavaScript

Programación estructurada

Funciones anónimas

- Es una función que no tiene nombre para ser invocada, por eso se denominan también auto-ejecutables.



```
// Función anónima (para asignar)
function() {
    console.log("Función anónima sin nombre");
}

// Función anónima => (function(){...})()
(function(){
    console.log("Me ejecuto!");
}());

// Función anónima con parámetros
(function(uno, dos, tres){
    console.log(uno);
    console.log(dos);
    console.log(tres);
})(1, 2, 3));
```

Ejercicio8

Crear una plantilla Html5 llamada **ejercicio8.html** que use un Script Javascript **js/ejercicio8.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Crear una función llamada suma que reciba dos parametros param1,param2 y devuelva el resultado de la suma de los dos parámetros. Invocar la función con los parámetros, (12,12)
- b) Crear una función anónima auto-ejecutable que calcule lo mismo que en el apartado anterior

Funciones anónimas

- ¿Qué hace?

```
// Rizando el rizo: funciones anidadas y función anónima
(function(num){
    var counter = 1; // Por si la necesitáramos

    // Funciones
    function viaAlert() { alert("Valor: " + (++num)); };
    function viaConsole() { console.log("Valor: " + (num--)); };

    // Ejecutamos
    viaAlert();
    viaConsole();
})(3); // Invocamos pasando un parámetro
```



Funciones anónimas

- ¿Qué hace?

```
// Rizando el rizo: funciones anidadas y función anónima
(function(num){
    var counter = 1; // Por si la necesitáramos

    // Funciones
    function viaAlert() { alert("Valor: " + (++num)); };
    function viaConsole() { console.log("Valor: " + (num--)); };

    // Ejecutamos
    viaAlert();
    viaConsole();
})(3); // Invocamos pasando un parámetro
```



Funciones : variables locales

- ¿Qué hace?

```
var color = "Azul";  
  
function experimento () {  
    console.log(color);  
}  
  
experimento();  
  
var color = "Verde";  
  
experimento();
```



```
var color = "Azul";  
  
function experimento () {  
    var color = "Amarillo";  
    console.log(color);  
}  
  
experimento();  
  
var color = "Verde";  
  
experimento();
```

```
var color = "Azul";  
  
function experimento () {  
    var color = "Amarillo";  
    console.log(color);  
}  
  
experimento();  
  
var color = "Verde";  
experimento();  
  
console.log(color);
```

Ejercicio9

Crear una plantilla Html5 llamada **ejercicio9.html** que use un Script Javascript **js/ejercicio9.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- Implementar el ejemplo anterior y razonar el por qué de los resultados

JavaScript

Programación estructurada

Funciones : invocación

- ¿Qué devuelve?

```
// Definición del objeto
var obj = {
  counter: 0,
  inc: function(value) {
    this.counter += typeof value === 'number' ? value : 1;
  },
};

// Caso 1 - Invocación sin parámetro
obj.inc();
console.log("Valor del contador: " + obj.counter);

// Caso 2 - Invocación con parámetro
obj.inc(2);
console.log("Valor del contador: " + obj.counter);
```



Ejercicio10

Crear una plantilla Html5 llamada **ejercicio10.html** que use un Script Javascript **js/ejercicio10.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Crear un objeto llamado **funciones** e incluir una propiedad que sea una función llamada **colorVerde** que cuando se invoque muestre por consola “VERDE”. Invocar la función **colorVerde**

- b) Añadir al Objeto **funciones** otra propiedad función llamada **colorRojo** que cuando se invoque muestre por consola “ROJO”

04

Objeto
this



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



JavaScript

Objeto this

Objeto this

- El receptor del mensaje es **this** (metáfora mensaje/receptor).

```
var nombre = "Sonia";
var obj = {
  nombre: "Sandra",
  saludo: function() {
    console.log("Hola " + this.nombre);
  }
}
obj.saludo();
```



- ¿Qué pinta la llamada a la última función?

JavaScript

Objeto this

Objeto this

- ¡Valor dinámico!



- ¿Y ahora qué?

```
var nombre = "Sonia";
var obj = {
  nombre: "Sandra",
  saludo: function() {
    console.log("Hola " + this.nombre);
  }
}
var maria = {
  nombre: "Maria"
};
maria.saludo = obj.saludo;
maria.saludo();
```

Ejercicio11

Crear una plantilla Html5 llamada **ejercicio11.html** que use un Script Javascript **js/ejercicio11.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- A partir del código fuente del ejercicio10 realizar lo siguiente , invocar desde la función colorRojo a la función colorVerde

JavaScript

Objeto this

Objeto this

- ¡Valor dinámico!. Su significado es dinámico.
- Al ser dinámico, se decide en el momento de ejecutar la función.
- Se suele llamar como “el contexto de la función”, y cada función tiene su propio puntero *this*.
- Si no hay receptor, apunta al objeto global.
- Además, semánticamente, es como un parámetro oculto que el receptor se encargará de proveer:



```
var obj = {  
    nombre: "Sandra",  
    saludo: function() {  
        console.log("Hola " + this.nombre);  
    }  
};
```

obj.saludo(); => saludo([obj]);

JavaScript

Objeto this

Objeto this

```
var nombre = "Sonia";
var obj = {
  nombre: "Pepito",
  saludo: function() {
    var saludo_fn = function() {
      console.log("hola " + this.nombre);
    };
    saludo_fn();
  }
};
obj.saludo();
```

```
var nombre = "Sonia";
var obj = {
  nombre: "Pepito",
  saludo: function([this]) {
    var saludo_fn = function([this]) {
      console.log("hola " + this.nombre);
    };
    saludo_fn([objeto global]);
  }
};
obj.saludo([obj]);
```



Ejercicio12

Crear una plantilla Html5 llamada **ejercicio12.html** que use un Script Javascript **js/ejercicio12.js** y realizar las siguientes tareas :

Hay que mostrar por consola todos el resultado de todos los apartados:

- A partir del código fuente del ejemplo anterior , usando el objeto **this** realizar las modificaciones necesarias para conseguir que al invocar a la función **saludo_fn** se visualice correctamente la propiedad **nombre:'Pepito'**

JavaScript

Ordenar el código

Objeto this

- Una función anidada en otra NO comparte el receptor.
- El valor de *this* depende de la invocación, NO de la definición.



```
var obj = {
  clicks: 0,
  init: function() {
    console.log("init: ", this);
    $("#elemento").click(function() {
      console.log("Clicked: ", this);
      this.clicks += 1;
      console.log("Clicks: " + this.clicks);
    });
  }
};
obj.init();
```

init: ▼ Object {clicks: 0, init: function} ↴
 clicks: 0
 ► init: function () {
 ► __proto__: Object

Clicked: <input type="button" id="yo" value="pincha">

Clicks: NaN

JavaScript

Objeto this

Objeto this

- Una función anidada en otra NO comparte el receptor.
- El valor de *this* depende de la invocación, NO de la definición.



```
var obj = {
  clicks: 0,
  init: function() {
    console.log("init: ", this);
    var that = this;
    $("#elemento").click(function() {
      console.log("This: ", this);
      console.log("That: ", that);
      that.clicks += 1;
      console.log("Clicks: " + that.clicks);
    });
  };
obj.init();
```



```
<top frame> Preset log
init: > Object {clicks: 0, init: function}
This: <input type="button" id="yo" value="pincha">
That: > Object {clicks: 0, init: function}
Clicks: 1
This: <input type="button" id="yo" value="pincha">
That: > Object {clicks: 1, init: function}
Clicks: 2
This: <input type="button" id="yo" value="pincha">
That: > Object {clicks: 2, init: function}
Clicks: 3
> |
```

JavaScript

Objeto this

Objeto this

Forma 1

```
var fnc = obj.metodo;  
fnc();
```



Forma 2

```
obj.metodo();
```

- Accede a la propiedad “método” de **obj**.
- Supondremos que es una función y se invoca
- No hay receptor
- Envía el mensaje “método” a **obj**.
- Si existe, **obj** se encarga de ejecutar la función
- **obj** es el receptor.

05

Excepciones



MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



Control de errores

- Mecanismo try / catch idéntico a otros lenguajes, como Java.



```
<!DOCTYPE html>
<html>
<body>
  <p id="ejemplo"></p>

  <script>
    try {
      aleeeeeeeeeert("Hola mundo!");
    }
    catch(err) {
      document.getElementById("demo").innerHTML = err.message;
    }
  </script>
</body>
</html>
```

Control de errores

- Existen cinco tipos de errores primarios:
 - **EvalError**. Se causa cuando se usa incorrectamente la función eval().
 - **RangeError**. Se lanza cuando una variable numérica se desborda.
 - **ReferenceError**. Sucede cuando se accede a una referencia inválida.
 - **SyntaxError**. Ocurre al producirse un error sintáctico en la lectura del código.
 - **TypeError**. Se origina cuando el tipo de la variable no es el esperado.
- Para crear nuevos errores podemos hacer:

```
throw new Error("Mensaje de error");
```



Control de errores

```
new EvalError([message[,fileName[,lineNumber]]])
```

```
new RangeError([message,[fileName[,lineNumber]]])
```

```
new ReferenceError([message,[fileName[,lineNumber]]])
```

```
new SyntaxError([message,[fileName[,lineNumber]]])
```

```
new TypeError([message,[fileName[,lineNumber]]])
```

El error devuelto es un Objeto con varias propiedades

Ejemplo :

```
throw new EvalError('Error de sintaxis,'fichero.js',12);
```



Ejercicio13

Crear una plantilla Html5 llamada **ejercicio13.html** que use un Script Javascript **js/ejercicio13.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Generar un bloque de código con la sentencia **try ... catch**
- b) Lanzar un error del tipo **EvalError** con el texto '**Se ha producido un error**' , fichero donde se ha producido el error ejercicio13.js y línea **100** , dentro del bloque **try**

Creando Objetos

Literales

```
{  
    atributo: "valor",  
    nombre: "Sema",  
    categoria: "SN",  
    nivel2: true,  
    getPosition: function() {  
        console.log(this.nombre + " es " +  
            this.categoria + "-" +  
            (this.nivel2) ? "N2" : "N1");  
    }  
}
```



Construidos

```
function Empleado(attr, nom, cat, niv) {  
    this.atributo = attr;  
    this.nombre = nom;  
    this.categoría = cat;  
    this.nivel2 = niv;  
    this.getPosition = function() {  
        console.log(this.nombre + " es " +  
            this.categoría + "-" +  
            ((this.nivel2) ? "N2" : "N1"));  
    };  
}
```

Creando Objetos

Literales

- A partir de la clase Object
- Sencillos y ligeros



Construidos

- A partir de prototype o de una función constructora
- Utilizan un constructor

Creando Objetos

Cuando queremos crear una copia de un Objeto con sus propiedades y métodos necesitamos usar los constructores. Un constructor es una función que define un nuevo objeto y lo devuelve. Hay varias formas de hacerlo:

- Una función que cree un Objeto con sus propiedades y métodos dentro de la misma y lo devuelva. Cada Objeto creado de esta manera se almacena en un espacio diferente en la memoria y por lo tanto tiene sus propias propiedades y valores, pero también se le Puede asignar el mismo Objeto a varias variables (Usar `==` para comparar propiedades), también podemos comprobar si dos variables referencian el mismo Objeto usando el método `is` del Objeto global `Object (Object.is(objeto1,objeto2))`
- Con el operador `new`, estos tipos de constructores necesitan que las propiedades y métodos del Objeto se identifiquen con la palabra clave `this`
- A través de la herencia. Heredan las propiedades y métodos, la herencia en Javascript se realiza a través de los `prototypes`. Para usar esta opción se usa el método `create` del Objeto global llamado `Object (var nuevoObjeto=Object.create(objeto))`

Creando Objetos

- Deberíamos usar **prototype** para crear Objectos a partir de otros Objectos. Los Objectos creados son totalmente independientes pero comparten el enlace **prototype del + Objeto Padre**, con el que podemos modificar/añadir nuevas propiedades y métodos a todos los Objectos que se hayan creado del mismo Objeto Padre

Ejemplo :

```
var objetoPadre={  
edad:null,  
nombre:null,  
apellidos:null,  
muestraDatos:function(){  
console.log(this.nombre+' '+this.apellidos+' tiene una edad de '+this.edad);  
}  
};
```

Creando Objetos

```
/* Creamos hijo1 a partir de objetoPadre */
var hijo1=Object.create(objetoPadre);
hijo1.edad=21;
hijo1.nombre='Antonio';
hijo1.apellidos='Martinez Garcia';

/* Creamos hijo2 a partir de objetoPadre */
var hijo2=Object.create(objetoPadre);
hijo2.edad=30;
hijo2.nombre='Pepe';
hijo2.apellidos='Martinez Garcia';

hijo1.muestraDatos();
hijo2.muestraDatos();

/*hijo1.verEdad(); hijo2.verEdad();*/

/* Creamos una función nueva del Objeto Padre que será heredada por hijo1 e hijo2 a través del prototype */
objetoPadre['verEdad']=function(){
  console.log(this.edad);
}
hijo1.verEdad();
hijo2.verEdad();
```

Ejercicio14

Crear una plantilla Html5 llamada **ejercicio14.html** que use un Script Javascript **js/ejercicio14.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Implementar el ejemplo anterior

06

Dom



MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



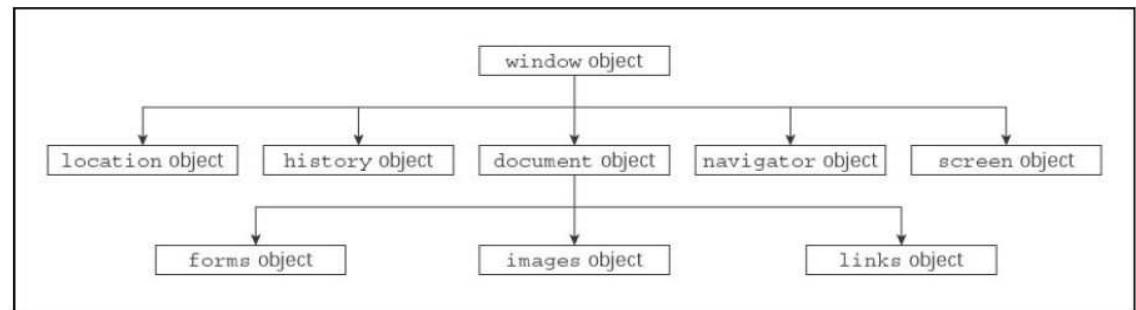
Árbol de elementos

- BOM
- Trabajando con el DOM
 - Tipos de nodos, enlaces entre nodos
- Trabajando con CSS
 - Estilos, clases
- Animaciones
- Eventos
 - Flujo de eventos, tipos (carga, foco, ratón, teclado)
- Trabajando con Formularios



Árbol de elementos

- *Browser Object Model*
- Define una serie de objetos que nos permiten interactuar con el navegador



JavaScript

Dom

window

- Objeto global
- `window.open()` → abre una nueva ventana del navegador
 - En los navegadores actuales, si intentamos abrir más de una ventana mediante `window.open` el navegador bloqueará su apertura (p.ej. ventanas emergentes de publicidad)
 - Devuelve un nuevo objeto `window` → será global para el script que corre sobre dicha ventana
 - Conteniendo todas las propiedades comunes a los objetos (constructor `Object`, objeto `Math`)
 - La mayoría de los navegadores no permiten consultar sus propiedades → modo `sandbox`.
- **Modo sandbox** → implica que el navegador sólo nos mostrará la información relativa al mismo dominio, y si abrimos una página de un dominio diferente al nuestro no tendremos control sobre las propiedades privadas del objeto `window`.



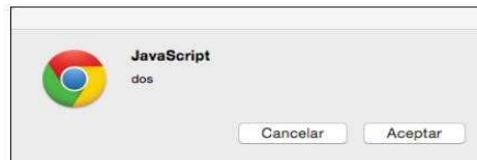
- `window.close()`
- `window.alert(mensaje)`, `window.confirm(mensaje)` y `window.prompt(mensaje [, valorPorDefecto])`. → mensajes

JavaScript

Dom

Mensajes

```
alert("uno");
confirm("dos");
var resp = prompt("tres");
```



Ejercicio15

Crear una plantilla Html5 llamada **ejercicio15.html** que use un Script Javascript **js/ejercicio15.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Solicitar por pantalla un valor y guardarlo en una variable llamada **nombre** , mostrar el valor introducido por **console.log**. El mensaje que tiene que aparecer al pedir el nombre es '**Introduzca el nombre :**'

- b) Solicitar por pantalla una confirmación y guardarla en una variable llamada **continuar** , mostrar el valor por **console.log**. El mensaje que tiene que aparecer al pedir la confirmación es, '**¿Desea continuar?**'

JavaScript

Dom

navigator

- Permite acceder a propiedades de información del navegador, tales como su nombre y versión.

```
console.log(navigator.language); // "es-es"
console.log(navigator.cookieEnabled); // true
console.log(navigator.appName); // "Netscape"
console.log(navigator.appVersion); // "5.0 (Macintosh; Intel Mac OS X 10_10_1)
AppleWebKit/600.2.5 (KHTML, like Gecko) Version/8.0.2 Safari/600.2.5"
console.log(navigator.product); // "Gecko"
console.log(navigator.userAgent); // "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1)
AppleWebKit/600.2.5 (KHTML, like Gecko) Version/8.0.2 Safari/600.2.5"
```



Ejercicio16

Crear una plantilla Html5 llamada **ejercicio16.html** que use un Script Javascript **js/ejercicio16.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- Mostrar por consola todas las propiedades y valores de las mismas del Objeto **navigator**. Mostrarlas con el siguiente formato
"nombre_propiedad = valor_propiedad"

document, document.location

- Objeto que representa el documento mostrado
- Propiedad `location` → información sobre la URL
 - `href`: cadena que representa la URL completa
 - `protocol`: protocolo de la URL
 - `host`: nombre del host
 - `pathname`: trayectoria del recurso
 - `search`: parte que contiene los parámetros, incluido el símbolo ?

```
http://localhost:63342/Pruebas/bom/location.html?alfa=beta&gama=delta
console.log("href:" + location.href); // http://localhost:63342/Pruebas/bom/location.html?
alfa=beta&gama=delta
console.log("protocol:" + location.protocol); // http:
console.log("host:" + location.host); // localhost:63342
console.log("pathname:" + location.pathname); // /Pruebas/bom/location.html
console.log("search:" + location.search); // ?alfa=beta&gama=delta
```

- Si a `location.href` le asignamos una nueva URL, el navegador realizará una petición a dicha URL y el navegador cargará el nuevo documento.



JavaScript

Dom

document, document.write

- `document.write(texto)`
- Permite escribir contenido HTML en el documento.

```
<html>
<head><title>La hora</title></head>
<body>
  <p>Son las
    <script type="text/javascript">
      var time = new Date();
      document.write(time.getHours() + ":" + time.getMinutes());
    </script>
  </p>
</body>
</html>
```



Ejercicio17

Crear una plantilla Html5 llamada **ejercicio17.html** que use un Script Javascript **js/ejercicio17.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- Escribir en el documento Html el texto en negrita ***"Mi primera línea de texto con write"***

Trabajando con el Dom

- DOM: *Document Object Model* - Modelo de objetos de documento
- El API DOM permite interactuar con el documento HTML
 - modificar el contenido y la estructura, estilos CSS y gestionar los eventos mediante *listeners*.
- El *DOM* es un modelo que representa en forma de árbol un documento HTML, formado por nodos.
 - *DOM Level 0* (Legacy DOM): define las colecciones `forms`, `links` e `images`.
 - *DOM Level 1* (1998): introduce el objeto `Node` y a partir de él, los nodos `Document`, `Element`, `Attr` y `Text`. Además, las operaciones `getElementsByName`, `getAttribute`, `removeAttribute` y `setAttribute`
 - *DOM Level 2*: facilita el trabajo con XHTML y añade los métodos `getElementById`, `hasAttributes` y `hasAttribute`
 - *DOM Level 3*: añade atributos al modelo, entre ellos `textContent` y el método `isEqualNode`.
 - *DOM Level 4* (2004): supone el abandono de HTML por XML e introduce los métodos `getElementsByClassName`, `prepend`, `append`, `before`, `after`, `replace` y `remove`.



Trabajando con el Dom

- El nodo raíz y parente de todos los nodos es `window`.
- En `JavaScript`, al declarar una variable tiene una alcance global, y todas las variables globales forman parte del objeto `window`.

```
var batman = "Bruce Wayne";
console.log(window.batman);
```

- Normalmente no referenciamos al objeto `window` directamente.
- Sólo cuando desde una función queremos acceder a una variable global de manera única.



```
var superheroe = "Batman";
var mejorSuperheroe = function () {
    var superheroe = "Superman";

    if (window.superheroe != superheroe) {
        superheroe = window.superheroe;
    }
}
```

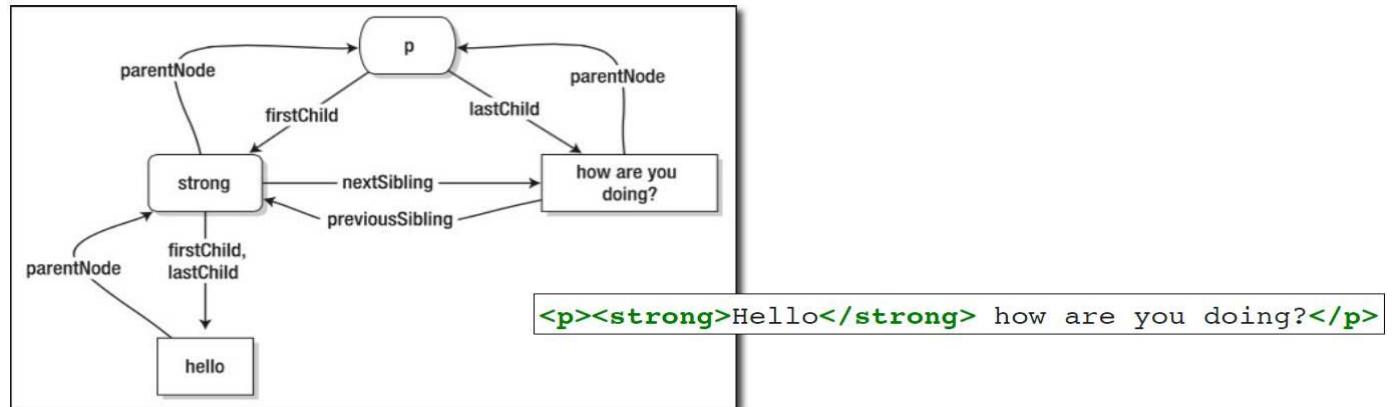
Trabajando con el Dom

- A partir de `document` podemos acceder a los elementos que forman la página mediante una estructura jerárquica.
- Al objeto que hace de raíz del árbol, el nodo `html`, se puede acceder mediante la propiedad `document.documentElement`.
- Si en vez de a `html` necesitamos acceder al elemento `body` → `document.body`
- Notación de `.` para navegar por el DOM



Trabajando con el Dom

- Cada elemento exceptuando el elemento `<html>` forma parte de otro elemento, que se le conoce como *padre (parent)*.
- Un elemento a su vez puede contener elementos hijos (*child*) y/o hermanos (*siblings*)

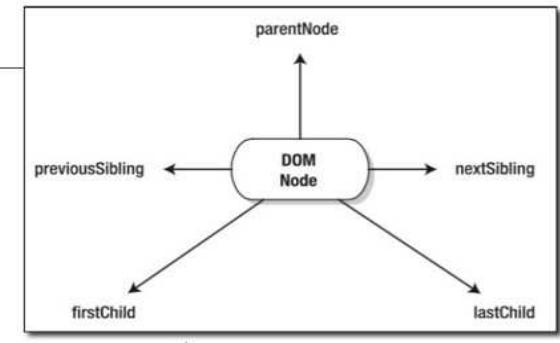


Trabajando con el Dom



```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Ejemplo DOM</title>
<meta charset="utf-8" />
</head>
<body>
<h1>Encabezado uno</h1>
<p>Primer párrafo</p>
<p>Segundo párrafo</p>
<div><p id="tres">Tercer párrafo dentro de un div</p></div>
<script src="dom.js" charset="utf-8"></script>
</body>
</html>
```

<http://jsbin.com/bopije/1/edit?html>

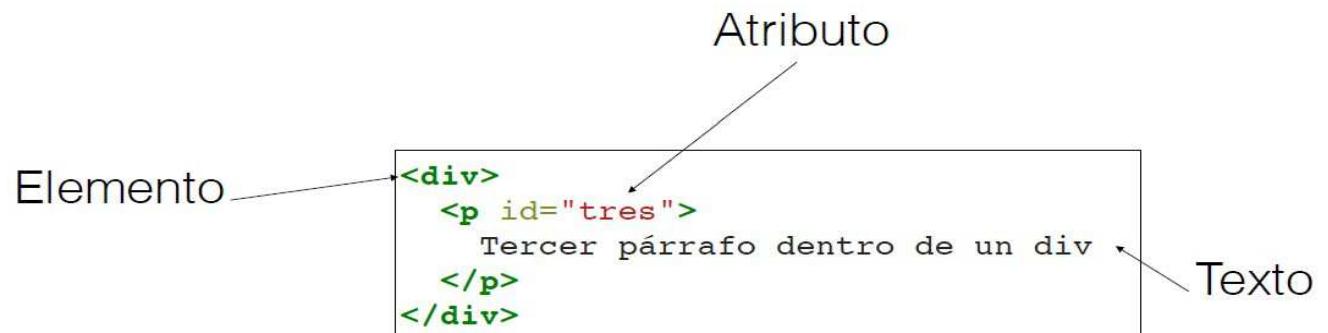


```
var encabezado = document.body.firstChild;
var scriptJS = document.body.lastChild;
var parrafos = encabezado.nextSibling;
var capa = scriptJS.previousSibling;
```

<http://jsbin.com/bopije/1/edit?js>

Trabajando con el Dom : tipos de nodos

- **Element**: nodo que contiene una etiqueta HTML
- **Attr**: nodo que forma parte de un elemento HTML
- **Text**: nodo que contiene texto y que no puede tener hijos



Trabajando con el Dom : tipos de nodos

- Para averiguar si un nodo representa un texto o un elemento → propiedad `nodeType`
 - Devuelve un número, por ejemplo: 1 si es un elemento (nodo HTML), 3 si es de texto, 8 comentario.

```
function esNodoTexto(nodo) {  
    return nodo.nodeType == document.TEXT_NODE;  
}  
esNodoTexto(document.body); // false  
esNodoTexto(document.body.firstChild.firstChild); // true
```

- Los elementos contienen la propiedad `nodeName` que indica el tipo de etiqueta HTML que representa (siempre en mayúsculas).
- Los nodos de texto contienen `nodeValue` que obtiene el texto contenido.

```
document.body.firstChild.nodeName; // H1  
document.body.firstChild.firstChild.nodeValue; // Encabezado uno
```



Trabajando con el Dom : recorriendo el Dom

- Búsqueda recursiva

```
function buscarTexto(nodo, cadena) {  
    if (nodo.nodeType == document.ELEMENT_NODE) {  
        for (var i=0, len=nodo.childNodes.length; i<len; i++) {  
            if (buscarTexto(nodo.childNodes[i], cadena)) {  
                return true;  
            }  
        }  
        return false;  
    } else if (nodo.nodeType == document.TEXT_NODE) {  
        return nodo.nodeValue.indexOf(cadena) > -1;  
    }  
}
```



Trabajando con el Dom : seleccionar elementos

- Grupal: `document.getElementsByTagName(nombreDeTag)` → devuelve un array con los nodos cuya etiqueta sea *nombreDeTag*
- Individual: `document.getElementById(nombreDeId)` → devuelve un nodo cuyo id coincida con *nombreDeId*

```
(function() {
    var pElements = document.getElementsByTagName("p"); // NodeList
    console.log(pElements.length); // 3
    console.log(pElements[0]); // Primer párrafo

    var divpElement = document.getElementById("tres");
    console.log(divpElement);
}());
```



Trabajando con el Dom : seleccionar elementos

- Selector API (2013)
 - Soportado por todos los navegadores actuales (soporte parcial en IE8)
 - Utilizan un selector CSS
 - Ofrece mucha flexibilidad
 - Empleado por jQuery
- `querySelector(selector)` → devuelve el 1^{er} elemento que cumple el selector
- `querySelectorAll(selector)` → devuelve una lista estática con todos los elementos que cumplen el selector
- `getElementById` es entre 3 y 4 veces más rápido que `querySelector`



```
var pElements = document.querySelectorAll("p");
var divpElement = document.querySelector("div p");
var tresElement = document.querySelector("#tres");
```

Ejercicio18

Crear una plantilla Html5 llamada **ejercicio19.html** que use un Script Javascript **js/ejercicio19.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Crear una plantilla Html5 con la siguiente estructura dentro del **<body>**:

```
<header>CABECERA</header>
<main><section>SECCION1<article>ARTICULO1
</article></section></main><footer>PIE</footer>
```

- b) Buscar el elemento **<header>** y mostrar por consola el elemento encontrado. Realizar la búsqueda de 3 maneras diferentes :
`getElementsByName`, `getElementById`, `querySelector`

Trabajando con el Dom : seleccionar elementos

- Las referencias con `getElementsBy*` están **vivas** y siempre contienen el estado actual del documento,
- Mediante `querySelector*` obtenemos las referencias existentes en el momento de **ejecución**, sin que cambios posteriores en el DOM afecten a las referencias obtenidas.



```
(function() {
    var getElements = document.getElementsByTagName("p"),
        queryElements = document.querySelectorAll("p");
    console.log("Antes con getElements:" + getElements.length); // 3
    console.log("Antes con querySelector:" + queryElements.length); // 3

    var elem = document.createElement("p");
    elem.innerHTML = "getElements vs querySelector";
    document.body.appendChild(elem);

    console.log("Después con getElements:" + getElements.length) // 4
    console.log("Después con querySelector:" + queryElements.length) // 3
}());
```

Trabajando con el Dom : operaciones CRUD con elementos

- `appendChild(nuevoElemento)` → el nuevo nodo se incluye inmediatamente después de los hijos ya existentes (si hay alguno) y el nodo padre cuenta con una nueva rama.
- `insertBefore(nuevoElemento, elementoExistente)` → permiten elegir un nodo existente del documento e incluir otro antes que él.
- `replaceChild(nuevoElemento, elementoExistente)` → reemplazar un nodo por otro
- `removeChild(nodoABorrar)` → elimina un nodo
- `cloneNode()` → permite clonar un nodo, permitiendo tanto el elemento como el elemento con su contenido (parámetro a `true`)
- Propiedad `innerHTML` → permite añadir el contenido de un elemento. Parsea el contenido incluido



Trabajando con el Dom : operaciones CRUD con elementos

- `appendChild(nuevoElemento)` → el nuevo nodo se incluye inmediatamente después de los hijos ya existentes (si hay alguno) y el nodo padre cuenta con una nueva rama.
 - `insertBefore(nuevoElemento, elementoExistente)` → permiten elegir un nodo existente del documento e incluir otro antes que él.
 - `replaceChild(nuevoElemento, elementoExistente)` → reemplazar un nodo por otro
 - `removeChild(nodoABorrar)` → elimina un nodo
 - `cloneNode()` → permite clonar un nodo, permitiendo tanto el elemento como el elemento con su contenido (parámetro a `true`)
-
- Propiedad `innerHTML` → permite añadir el contenido de un elemento. Parsea el contenido incluido



JavaScript

Dom

Trabajando con el Dom : operaciones CRUD con elementos

```
(function() {
    var doc = document,
        elem = doc.createElement("p"),
        contenido = doc.createTextNode("<strong>Nuevo párrafo creado dinámicamente</strong>"),
        pTres = doc.getElementById("tres");

    elem.appendChild(contenido);
    elem.id = "conAppendChild";

    pTres.parentNode.appendChild(elem); // o insertBefore, replaceChild
}());
```

```
(function() {
    var
        doc = document,
        elem = doc.createElement("p"),
        pTres = doc.getElementById("tres");

    elem.innerHTML = "<strong>Nuevo párrafo reemplazado dinámicamente</strong>";
    elem.id = "conInner";

    pTres.parentNode.replaceChild(elem, pTres);
}());
```



Encabezado uno

Primer párrafo
Segundo párrafo
Nuevo párrafo reemplazado dinámicamente

Q: Elements Network Sources Timeline Profiles Resources Audits Console

```
<html>
  <head>
    <title>Prueba</title>
  </head>
  <body>
    <p>Primer párrafo</p>
    <p>Segundo párrafo</p>
    <p>Nuevo párrafo reemplazado dinámicamente</p>
  </body>
</html>
```

Ejercicio19

Crear una plantilla Html5 llamada **ejercicio20.html** que use un Script Javascript **js/ejercicio20.js** y realizar las siguientes tareas :
Hay que mostrar por consola todos el resultado de todos los apartados:

- a) Crear dinámicamente un elemento **<p>** directamente dentro del **<body>** del documento, con el texto **Este texto está añadido dinámicamente**

Trabajando con el Dom : operaciones con atributos

- Operaciones

- `getAttribute(nombreAtributo)`
- `setAttribute(nombreAtributo, valorAtributo)`

```
var pTres = document.getElementById("tres");
pTres.setAttribute("align", "right");
```

- Propiedad

- `elemento.getAttribute()`

```
var pTres = document.getElementById("tres");
pTres.align = "right";
```

- El uso de atributos para definir la apariencia del documento está desaconsejado → CSS



Ejercicio20

A partir del código fuente del **ejercicio19.js** y **ejercicio19.html**, generar **ejercicio20.html** y **ejercicio20.js** y realizar las siguientes tareas :

Hay que mostrar por consola todos el resultado de todos los apartados:

- Añadir al elemento **<p>** un atributo ***id*** con valor ***parrafo1***

Trabajando con el Dom : operaciones con CSS

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8" />
  <title></title>
  <style>
    #batman { }
    .css-class {
      color: blue;
      border : 1px solid black;
    }
  </style>
</head>
<body>
  <div style="font-size:xx-large" id="batman">Batman siempre gana.</div>
  <script src="css.js"></script>
</body>
</html>
```



Trabajando con el Dom : operaciones con CSS

- Permite obtener /modificar los estilos

```
var divBatman = document.getElementById("batman");
divBatman.style.color = "blue";
divBatman.style.border = "1px solid black";
```

- Si la propiedad CSS contiene un guión, para usarla mediante JavaScript, se usa la notación *camelCase*.
 - `background-color` pasará a usarse como `backgroundColor`.



Trabajando con el Dom : operaciones con CSS (clases)

- Agrupan varios estilos
- Propiedad `className` (`class` es una keyword de JavaScript)

```
var divBatman = document.getElementById("batman");
divBatman.className = "css-class";
// divBatman.className = ""; -> elimina la clase CSS
```

- Para añadir más de una clase → separarlas con espacios o utilizar la propiedad `classList`
 - Permite añadir clases → método `add`
 - Para eliminar una clase → `remove`
 - Para cambiar una clase por otra → `toggle`

```
var divBatman = document.getElementById("batman");
divBatman.classList.remove("css-class");
divBatman.classList.add("css-class2");
```



Trabajando con el Dom : mostrar / ocultar contenido

- Propiedad `style.display`
 - `none` → no se muestra el elemento
 - cadena vacía → se muestra

```
var divBatman = document.getElementById("batman");
divBatman.style.display = "none"; // oculta
divBatman.style.display = ""; // visible
```



Trabajando con el Dom : estilo calculado

- Para averiguar el estilo de una determinada propiedad, podemos acceder a la propiedad de `window.getComputedStyle(elem, null).getPropertyValue(cssProperty)`.
- Si el navegador no la soporta (sólo IE antiguos), hay que usar el array `currentStyle`.

```
var divBatman = document.getElementById("batman");
var color = window.getComputedStyle(divBatman, null).getPropertyValue("color");
var colorIE = divBatman.currentStyle["color"];
```



Trabajando con el Dom : animaciones

- Movimiento y manipulación de contenido
- Animación → llamadas sucesivas a una función, con un límite de ejecuciones mediante *Timers*

```
var velocidad = 2000,
    i = 0;
miFuncion = function() {
    console.log("Batman vuelve " + i);
    i = i + 1;
    if (i < 10) {
        setTimeout(miFuncion, velocidad);
    }
};
setTimeout(miFuncion, velocidad);
```

```
var velocidad = 2000,
    i = 0;
miFuncion = function() {
    console.log("Batman vuelve " + i);
    i = i + 1;
    if (i > 9) {
        clearInterval(timer);
    }
};
var timer = setInterval(miFuncion, velocidad);
```



Trabajando con el Dom : animaciones



```
(function() {
    var velocidad = 10,
        mueveCaja = function(pasos) {
            var el = document.getElementById("caja"),
                izq = el.offsetLeft;

            if ((pasos > 0 && izq > 399) || (pasos < 0 && izq < 51)) {
                clearTimeout(timer);
                timer = setInterval(function() {
                    mueveCaja(pasos * -1);
                }, velocidad);
            }

            el.style.left = izq + pasos + "px";
        };

    var timer = setInterval(function () {
        mueveCaja(3);
    }, velocidad);
}());
```

```
<style>
#caja {
    position: absolute;
    left: 50px;
    top: 50px;
    background-color: blue;
    height: 100px;
    width: 100px;
}
</style>

<div id="caja"></div>
```

Trabajando con el Dom : eventos

- Asocian un comportamiento a una acción
 - Pulsar un botón, pasar el ratón por encima de un elemento, cargar la página, etc..
- 3 formas:
 1. `elemento.onEvento = manejador` → sólo un manejador
 2. atributo `on*` (`onclick`, `onmouseover`, etc...) de un elemento
 3. `addEventListener(evento, manejador, flujoEvento)` → múltiples manejadores



```
var el = document.getElementById("caja");  
1 el.onclick = function() {  
    this.style.backgroundColor = "red";  
};
```

```
2 <button onclick="this.style.backgroundColor='red';">Incrustado</button>
```

Trabajando con el Dom : eventos



```
<head>
  <style>
    .normal {
      background-color: white;
      color: black;
    }
    .contrario {
      background-color: black;
      color: white;
    }
  </style>
</head>
<body class="normal">
  <h1>Hola Eventos</h1>
  <p><a href="http://es.wikipedia.org/wiki/Batman">Batman</a> Forever</p>

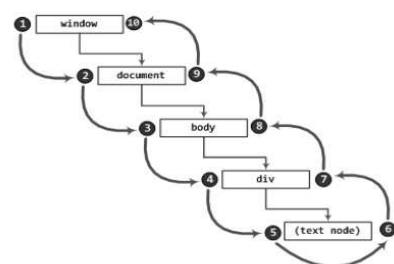
  <button>Normal</button>
  <button>Contrario</button>
</body>
```

```
(function() {
  var botones = document.getElementsByTagName("button");

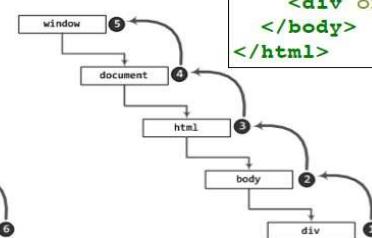
  for (var i=0, len=botones.length; i<len; i=i+1) {
    botones[i].onclick = function() {
      var className = this.innerHTML.toLowerCase();
      document.body.className = className;
    };
    // botones[i].onclick = function() {};
  }
})();
```

Trabajando con el Dom : eventos

- **Captura de eventos:** al pulsar sobre un elemento, se produce una evento de arriba a abajo, desde el elemento `window`, pasando por `<body>` hasta llegar al elemento que lo captura.
- **Burbujeo de eventos (event bubbling):** el evento se produce en el elemento de más abajo y va subiendo hasta llegar al `window`.



Captura



```
<html onclick="procesaEvento()">
  <head><title>Ejemplo de flujo de eventos</title></head>
  <body onclick="procesaEvento()">
    <div onclick="procesaEvento()">Pincha aqui</div>
  </body>
</html>
```

Burbujeo

Trabajando con el Dom : eventos

- Uso de atributo `on*`
- O `addEventListener(evento, funciónManejador, flujoEvento)`
 - `flujoEvento: true` (captura de eventos), `false` (*event bubbling - recomendado*)
- IE8 → `attachEvent`

```
var botones = document.getElementsByTagName("button");
for (var i=0, len=botones.length; i<len; i=i+1) {
    botones[i].onclick = function() {
        var className = this.innerHTML.toLowerCase();
        document.body.className = className;
    };
    // botones[i].onclick = function() {};
}
var botonClick = function() {
    var className = this.innerHTML.toLowerCase();
    document.body.className = className;
}
var botones = document.getElementsByTagName("button");
for (var i=0, len=botones.length; i<len; i=i+1) {
    botones[i].addEventListener("click", botonClick, false);
    // botones[i].removeEventListener("click", botonClick, false);
}
```



JavaScript

Trabajando con el Dom : información de eventos

- Se recibe como parámetro en el manejador
- Propiedades:
 - type → evento del que proviene
 - target → elemento sobre el cual está registrado

```
var botonClick = function(evt) {
    var className = this.innerHTML.toLowerCase();
    document.body.className = className;
    console.log(evt.type + " - " + evt.target); // click - HTMLButtonElement
}
```

- Para cancelar un evento → preventDefault()

```
var enlaceClick = function(evt) {
    evt.preventDefault();
}
```



Trabajando con el Dom : delegación de eventos

- Se basa en el flujo de eventos ofrecidos por *event bubbling*
- Delega un evento desde un elemento inferior en el DOM que va a subir como una burbuja hasta el exterior.

```
(function() {
    document.addEventListener("click", function(evt) {
        var tag = evt.target.tagName;
        console.log("Click en " + tag);

        if ("A" == tag) {
            evt.preventDefault();
        }
    }, false);
})());
```



Trabajando con el Dom : evento de foco

- `focus` → al tomar el foco
- `blur` → al perder el foco



```
var campoNombre = document.getElementById("nom");
campoNombre.value = "Escribe tu nombre";

campoNombre.onfocus = function() {
    if (campoNombre.value == "Escribe tu nombre") {
        campoNombre.value = "";
    }
};

campoNombre.onblur = function() {
    if (campoNombre.value == "") {
        campoNombre.value = "Escribe tu nombre";
    }
};
```

```
<form name="miForm">
    Nombre: <input type="text" name="nombre" id="nom" tabindex="10" />
    Apellidos: <input type="text" name="apellidos" id="ape" tabindex="20" />
</form>
```

Trabajando con el Dom : evento de carga

- Para poder asignar un *listener* a un elemento del *DOM*, éste debe haberse cargado.
- Se recomienda incluir el código *JavaScript* al final de la página *HTML*, justo antes de cerrar el *body*.
- `window.onload` → se lanza cuando el documento ha cargado completamente (incluye las imágenes)



```
function preparandoManejadores() {
    var miLogo = document.getElementById("logo");
    miLogo.onclick() {
        alert("Has venido al sitio adecuado.");
    }
}

window.onload = function() {
    preparandoManejadores();
}
```

Ejercicio21

Crear una plantilla Html5 llamada ***ejercicio21.html*** que use un Script Javascript ***js/ejercicio21.js***, una hoja de estilos ***css/ejercicio22.css*** y realizar las siguientes tareas :

- a) Crear un regla que tenga como selector una clase llamada ***parrafo*** y establezca en el texto un color ***rojo***, subrayado y un tamaño de fuente de ***2em***
- b) Crear una regla que tenga como selector una clase llamada ***aplicado*** y establezca en el color de fondo el color ***yellow***
- c) Crear dinámicamente un elemento ***<p>*** directamente dentro del ***<body>*** del documento, con el texto ***Párrafo1*** y con el atributo ***id="p1"***
- d) Crear dinámicamente un elemento ***<button>*** con ***id="boton1"*** con el texto ***"Aplicar estilo a párrafo"***, añadirlo dentro del elemento ***<p>*** creado anteriormente
- e) Asignar un evento ***onclick*** al botón con ***id="boton1"*** que cuando se realice un click sobre el botón realice lo siguiente :
 1. Si el elemento ***<p>*** no tiene aplicada la clase ***parrafo*** aplicarla y cambiar el texto del botón a ***"Quitar estilo a párrafo"*** y añadir la clase ***aplicado*** al botón
 2. Si el elemento ***<p>*** tiene aplicada la clase ***párrafo*** quitarla y cambiar el texto del botón a ***"Aplicar estilo a párrafo"*** y quitar la clase ***aplicado*** al botón

<https://developer.mozilla.org/es/docs/Web/API/Element/classList>

Trabajando con el Dom : eventos de ratón

- Al hacer click: `mousedown` → `mouseup` → `click`
- Si sucede dos veces de manera consecutiva → `dblclick`
- Coordenadas: `clientX`, `clientY`
- Movimiento: `mousemove`
- Entrar y salir de un elemento: `mouseover`, `mouseout`
 - `target`: referencia el nodo que ha lanzado el evento
 - `relatedTarget`: indica el nodo de donde viene el ratón (para `mouseover`) o adonde va (para `mouseout`)
- Cuidado con el *event bubbling*
 - Al asociar un manejador a un botón, lo normal es que sólo nos interese si sólo ha hecho click.
 - Si asociamos el manejador a un nodo que tiene hijos, al hacer click sobre los hijos el evento "burbujea" hacia arriba, por lo que nos interesaría averiguar que hijo ha sido el responsable (propiedad `target`)



Trabajando con el Dom : eventos de ratón

```
var miLogo = document.getElementById("logo");
miLogo.onclick() {
    alert("Has venido al sitio adecuado.");
}
```



```
miParrafo.addEventListener("mouseover", function(event) {
    if (event.target == miParrafo)
        console.log("El ratón ha entrado en mi párrafo");
}, false);
```

Trabajando con el Dom : eventos de teclado

- **keydown**: al pulsar una tecla; también se lanza si se mantiene pulsada
- **keyup**: al soltar una tecla
- **keypress**: tras soltar la tecla, pero sin las teclas de modificación; también se lanza si se mantiene pulsada
- Secuencia de eventos:
 - Carácter alfanumérico: keydown → keypress → keyup.
 - Otro tipo de tecla: keydown, keyup.
 - Carácter alfanumérico pulsado: se repiten de forma continua los eventos keydown y keypress
 - Otro tipo de tecla pulsada: se repite el evento keydown de forma continua.
- Usaremos keydown y keyup para averiguar que tecla se ha pulsado, por ejemplo los cursores.
- Si estamos interesado en el carácter pulsado, entonces usaremos keypress.



Trabajando con el Dom : teclas especiales

- Al usar los eventos `keydown` y `keyup`, podemos consultar a partir del evento las propiedades:
 - `keyCode`: obtiene el código ASCII del elemento pulsado
 - `altKey`: devuelve true/false si ha pulsado la tecla ALT (option en MAC)
 - `ctrlKey`: devuelve true/false si ha pulsado la tecla CTRL
 - `shiftKey`: devuelve true/false si ha pulsado la tecla SHIFT
 - `metaKey`: devuelve true/false si ha pulsado la tecla command de MAC



```
document.addEventListener("keydown", function(evt) {  
    var code = evt.keyCode;  
    var ctrlKey = evt.ctrlKey;  
  
    if (ctrlKey && code === 66) {  
        console.log("Ctrl+B");  
    }  
}, false);
```

Trabajando con el Dom : formularios

- Para interactuar con un formulario, es conveniente asignar un `id`
- Si no tiene `id`, pero si `name` → `document.forms.nombreDelFormulario`
- Para los campos, bien por su `id` o si tienen nombre →
`document.forms.nombreDelFormulario.nombreDelCampo`

```
<form name="formCliente" id="frmClnt">
<fieldset id="infoPersonal">
  <legend>Datos Personales</legend>
  <p><label for="nombre">Nombre</label>
    <input type="text" name="nombre" id="nom" /></p>
  <p><label for="correo">Email</label>
    <input type="email" name="correo" id="email" /></p>
</fieldset>
<!-- ...Dirección ... -->
</fieldset>
</form>
```

```
// Mediante el atributo name
var formulario = document.forms.formCliente;
var correo = formulario.correo;
// Mediante el atributo id
var formuId = document.getElementById("frmClnt");
var correoId = document.getElementById("email");
```



Trabajando con el Dom : eventos de teclado

```
<input type="text" name="cajaTexto" id="cajaTexto" />
```

```
(function() {
    var caja = document.getElementById("cajaTexto");
    document.addEventListener("keypress", function(evt) {

        var ascii = evt.charCode;

        if (ascii >= 65 && ascii <=90) {
            // solo dejamos mayúsculas
            // las minúsculas van del 97 al 122
        } else {
            evt.preventDefault();
        }
    }, false);
}());
```



Trabajando con el Dom : validación de formularios

- Al enviar un formulario, se produce el evento `submit`
 - Dentro del manejador, devolvemos `true` si las validaciones son correctas



```
function preparandoManejadores() {
    document.getElementById("frmClnt").onsubmit = function() {
        var ok = false;
        // validamos el formulario
        if (ok) {
            return true; // se realiza el envío
        } else {
            return false;
        }
    };

    window.onload = function() {
        preparandoManejadores();
    };
}
```

Trabajando con el Dom : campos de texto

- type="text" (o "password" o "hidden")
- Contenido mediante propiedad **value**
- Los eventos que puede lanzar son: focus, blur, change, keypress, keydown y keyup.



```
var correoId = document.getElementById("email");
if (correoId.value == "") {
    alert("Por favor, introduce el correo");
}
```

Trabajando con el Dom : desplegables

- Etiqueta `select`
- Propiedad `type` → indica si se trata de una lista de selección única (`select-one`) o selección múltiple (`select-multiple`).
- Al seleccionar un elemento, se lanza el evento `change`.
- Para acceder al elemento seleccionado
 - Selección única: propiedad `selectedIndex` (de 0 a n-1).
 - Selección múltiple: recorrer el array de `options` y consultar la propiedad `selected` → `options[i].selected`.
- Una vez tenemos un elemento/opción (objeto `Option`), podemos acceder a la propiedad
 - `value` → obtiene el valor
 - `text` → texto



Trabajando con el Dom : desplegables

```
<form name="formCliente" id="frmClnt">
<!-- ... Datos Personales ... -->

<fieldset id="direccion">
  <legend>Dirección</legend>
  <p><label for="tipoVia">Tipo de Vía</label>
    <select name="tipoVia" id="tipoViaId">
      <option value="calle">Calle</option>
      <option value="avda">Avenida</option>
      <option value="pza">Plaza</option>
    </select>
  </p>
  <p><label for="domicilio">Domicilio</label>
    <input type="text" name="domicilio" id="domi" /></p>
</fieldset>
</form>
```



```
var tipoViaId = document.getElementById("tipoViaId");

tipoViaId.onchange = function() {
  var indice = tipoViaId.selectedIndex; // 1
  var valor = tipoViaId.options[indice].value; // avda
  var texto = tipoViaId.options[indice].text; // Avenida
};
```

Trabajando con el Dom : desplegables

- Cada elemento es un objeto `Option`
- Para añadir elementos a la lista de manera dinámica → `add(option, lugarAntesDe)`
 - Si no se indica el lugar se insertará al final de la lista
- Para eliminar un elemento → `remove(indice)`

```
var op = new Option("Camino Rural", "rural");
tipoviaId.add(op, tipoviaId.options[3]);
```



Trabajando con el Dom : radio y check

- Tanto los *radio* como los *checkboxes* tienen la propiedad `checked` que nos dice si hay algún elemento seleccionado (`true` o `false`).
- Para averiguar cual es el elemento marcado, tenemos que recorrer el array de elementos que lo forman.
- Eventos → `click`, `change`

```
color.checked = true;

var colorElegido = "";

for (var i = 0, l = color.length; i < l; i = i + 1) {
  if (color[i].checked) {
    colorElegido = color[i].value;
  }
}
```



Ejercicio22

: crear una plantilla Html5 llamada *ejercicio23.html* que use un Script Javascript *js/ejercicio23.js*, una hoja de estilos *css/ejercicio23.css* y realizar las siguientes tareas :

- a) Crear un formulario como el mostrado en la imagen

Datos Personales

Nombre:

Email:

Provincia:
Comunidad Valenciana
ALICANTE
CASTELLON
Comunidad Murciana
MURCIA
TOTANA

Enviar

- b) Cuando se haga *click* sobre el botón *Enviar* se tienen que realizar las siguientes comprobaciones

1. El campo **Nombre** no puede ser mayor a 20 caracteres y debe contener la cadena de texto **ANTONIO** al inicio del texto
2. El resto de campos tienen que estar rellenos
3. Mostrar debajo del recuadro **Datos Personales** los valores introducidos en el formulario
4. Si no se cumple ni el punto 1 ni el punto 2 mostrar un mensaje debajo del recuadro de **Datos Personales** el siguiente mensaje **"Los datos del formulario no son correctos"**, este mensaje deberá desaparecer cuando se edite otra vez cualquier campo del formulario.

<https://developer.mozilla.org/es/docs/Web/API/Element/classList>

JavaScript

Definición de clases ECMAScript 2015

Clases

- En la versión ESCMAScript 2015 se introduce la definición de las clases muy al estilo Java

```
class Persona {  
    constructor(nombre, apellidos) {  
        this.nombre = nombre;  
        this.apellidos = apellidos;  
    }  
}
```

JavaScript

Definición de clases ECMAScript 2015

Clases

- Uso de la palabra reservada **class** seguida del nombre de la clase

```
class Persona {  
    constructor(nombre, apellidos) {  
        this.nombre = nombre;  
        this.apellidos = apellidos;  
    }  
}
```

- Constructor de la clase, con la palabra reservada **constructor**

Clases

- Para declarar las variables miembro de la clase podemos usar la palabra reservada `this.<propiedad>` dentro del constructor

```
class Persona {  
    constructor(nombre, apellidos) {  
        this.nombre = nombre;  
        this.apellidos = apellidos;  
    }  
}
```

JavaScript

Definición de clases ECMAScript 2015

Clases

- Funciones `get` y `set` personalizadas

```
get nombreCompleto() {  
    return `El nombre completo del alumno es ${this.nombre} ${this.apellidos}`;  
}  
  
set nombreCompleto(nombreCompleto) {  
    var arr = nombreCompleto.split(" ");  
    this.nombre = arr[0];  
    this.apellidos = arr[1];  
}
```

```
let a1 = new Alumno(['Antonio', 'Martínez']);  
console.log(a1.nombreCompleto);  
a1.nombreCompleto = 'Ramón Meseguer';  
console.log(a1.nombreCompleto);
```

JavaScript

Definición de clases ECMAScript 2015

Clases

- Herencia , usando la palabra reservada **extends**

```
class Persona {  
    constructor(nombre, apellidos) {  
        this.nombre = nombre;  
        this.apellidos = apellidos;  
    }  
}  
  
class Alumno extends Persona {  
  
    constructor(nombre, apellidos) {  
        super();  
        this.nombre = nombre;  
        this.apellidos = apellidos;  
    }  
  
    get nombreCompleto() {  
        return `El nombre completo del alumno es ${this.nombre} ${this.apellidos}`;  
    }  
  
    set nombreCompleto(nombreCompleto) {  
        var arr = nombreCompleto.split(" ");  
        this.nombre = arr[0];  
        this.apellidos = arr[1];  
    }  
}
```

JavaScript

Fetch API y XMLHttpRequest

Para realizar llamadas a servicios REST

- XMLHttpRequest

```
var xhr = new XMLHttpRequest();
xhr.open('GET', '/server', true);

xhr.onload = function () {
    // Request finished. Do processing here.
};

xhr.send(null);
// xhr.send('string');
// xhr.send(new Blob());
// xhr.send(new Int8Array());
// xhr.send(document);
```

```
var xhr = new XMLHttpRequest();
xhr.open("POST", '/server', true);

//Send the proper header information along with the request
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.onreadystatechange = function() { // Call a function when the state changes.
    if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
        // Request finished. Do processing here.
    }
}
xhr.send("foo=bar&lorem=ipsum");
// xhr.send(new Int8Array());
// xhr.send(document);
```

JavaScript

Fetch API y XMLHttpRequest

Para realizar llamadas a servicios REST

- Request (Fetch API), GET

```
const request = new Request('https://www.mozilla.org/favicon.ico');

const URL = request.url;
const method = request.method;
const credentials = request.credentials;

fetch(request)
  .then(response => response.blob())
  .then(blob => {
    image.src = URL.createObjectURL(blob);
  });
}
```

JavaScript

Fetch API y XMLHttpRequest

Para realizar llamadas a servicios REST

- Request (Fetch API), POST

```
const request = new Request('https://example.com', {method: 'POST', body: '{"foo": "bar"}'});

const URL = request.url;
const method = request.method;
const credentials = request.credentials;
const bodyUsed = request.bodyUsed;

fetch(request)
  .then(response => {
    if (response.status === 200) {
      return response.json();
    } else {
      throw new Error('Something went wrong on api server!');
    }
  })
  .then(response => {
    console.debug(response);
    // ...
  })
  .catch(error => {
    console.error(error);
  });
});
```

JavaScript

Funciones Arrow con ECMAScript 2015

Cambio en la definición de las funciones

- Podemos simplificar la definición de las funciones evitando la palabra clave *function*

```
function Suma(valor1, valor2) {  
    return valor1 + valor2;  
}  
  
var suma = (valor1, valor2) => {  
    return valor1 + valor2;  
}  
  
console.log(Suma(12, 12));  
console.log(suma(12, 12));
```

24
24

JavaScript

Objetos de interés en ECMAScript 2015

Array, Map y Set

Todos estos Objetos tiene multiples funciones que facilitan en gran medida el manejo de los mismos

- **Array** : Objeto usado para la manipulación de *Arrays*.

```
var array1=['Pepe','Antonio'];
```

- **Map** : Objeto usado para la manipulación de *Maps*

```
const map=new Map();
map.set("nombre","Antonio");
```

- **Set**: Objeto usado para la manipulación de *Sets* (*Como Array pero no permite elementos duplicados*)

```
const set=new Set();
set.add("Antonio");
set.add("Pepe");
```

JavaScript

Otras funcionalidades de ECMAScript 2015

Array.from(x) -> crea un array a partir del argumento que se le pasa como parámetro

```
console.log(Array.from('EOI'));  
  
console.log(Array.from([1, 2, 3], x => x + x));
```

```
[ 'E', 'O', 'I' ]  
[ 2, 4, 6 ]
```

JavaScript

Otras funcionalidades de ECMAScript 2015

Array.of(x) -> crea un array a partir del argumento que se le pasa como parámetro

```
console.log(Array.of(12));
console.log(Array.of([1, 2, 3]));
```

```
[ 12 ]
[ 1, 2, 3 ]|
```

JavaScript

Otras funcionalidades de ECMAScript 2015

`Array.fill(valor,desde,hasta)` -> setea a *valor* los elementos del array en las posiciones *desde..hasta* , si no se indica nada setea todas las posiciones del array a ese *valor*

```
const array1 = [1, 2, 3, 4];

console.log(array1.fill(0, 1, 2));
| 

console.log(array1.fill(5, 1));

console.log(array1.fill(6));
```

```
[ 1, 0, 3, 4 ]
[ 1, 5, 5, 5 ]
[ 6, 6, 6, 6 ]
```



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



JavaScript

Otras funcionalidades de ECMAScript 2015

`Array.find(x => condición) -> Devuelve el primer elemento del array que cumple la condición`

```
const array1 = ["Hola", "soy", "un", "alumno de la EOI"];

const found = array1.find(element => element.indexOf('EOI') > 0);

console.log(found);
```

alumno de la EOI

JavaScript

Otras funcionalidades de ECMAScript 2015

Array.findIndex(x => condición) -> Devuelve la posición del primer elemento del array que cumple la condición

```
const array1 = ["Hola", "soy", "un", "alumno de la EOI"];

const un = (element) => { return element == "un"; };

console.log(array1.findIndex(un));
```

2

JavaScript

Otras funcionalidades de ECMAScript 2015

Array.entries() -> Devuelve el iterador del array, conteniendo cada iteración el par clave/valor del elemeto correspondiente del array

```
const array1 = ["Hola", "soy", "un", "alumno de la EOI"];

var iterador = array1.entries();

for (var item of iterador) {
    console.log(item);
}
```

```
[ 0, 'Hola' ]
[ 1, 'soy' ]
[ 2, 'un' ]
[ 3, 'alumno de la EOI' ]
```

JavaScript

Otras funcionalidades de ECMAScript 2015

Map.entries() -> Devuelve el iterador del Map, conteniendo cada iteración el par clave/valor del elemeto correspondiente del Map

```
const map1 = new Map();
map1.set('nombre', 'Mi nombre');
map1.set('datosContacto', { direccion: 'Mi direccion', telefono: 'Mi telefono' });

var iterador = map1.entries();

for (var item of iterador) {
    console.log(item);
}
```

```
[ 'nombre', 'Mi nombre' ]
[ 'datosContacto',
  { direccion: 'Mi direccion', telefono: 'Mi telefono' } ]
```

JavaScript

Otras funcionalidades de ECMAScript 2015

Array.keys() -> Devuelve el iterador con las keys del array

```
const array1 = ["Hola", "soy", "un", "alumno de la EOI"];  
  
var iteradorKeys = array1.keys();  
|  
for (var key of iteradorKeys) {  
    console.log(key);  
}
```

0
1
2
3

JavaScript

Otras funcionalidades de ECMAScript 2015

Map.keys() -> Devuelve el iterador con las keys del Map

```
const map1 = new Map();
map1.set('nombre', 'Mi nombre');
map1.set('datosContacto', { direccion: 'Mi direccion', telefono: 'Mi telefono' });

var iteradorKeys = map1.keys();

for (var key of iteradorKeys) [
  console.log(key);
]
```

nombre
datosContacto

JavaScript

Otras funcionalidades de ECMAScript 2015

Array.values() -> Devuelve el iterador con los valores del array

```
const array1 = ["Hola", "soy", "un", "alumno de la EOI"];

var iteradorValues = array1.values();

for (var value of iteradorValues) {
    console.log(value);
}
```

```
Hola
soy
un
alumno de la EOI
```

JavaScript

Otras funcionalidades de ECMAScript 2015

Map.values() -> Devuelve el iterador con los valores del Map

```
const map1 = new Map();
map1.set('nombre', 'Mi nombre');
map1.set('datosContacto', { direccion: 'Mi direccion', telefono: 'Mi telefono' });

var iteradorValues = map1.values();

for [var value of iteradorValues] {
    console.log(value);
}
```

```
Mi nombre
{ direccion: 'Mi direccion', telefono: 'Mi telefono' }
```

JavaScript

Otras funcionalidades de ECMAScript 2015

Array.map(función) -> Devuelve un nuevo array resultado de aplicar la función pasada como parámetro

```
const array1 = ["Hola", "soy", "un", "alumno de la EOI"];  
  
var array2 = array1.map(elemento => {  
    return "<EOI>" + elemento  
});  
  
console.log(array2);  
|
```

```
[ '<EOI>Hola', '<EOI>soy', '<EOI>un', '<EOI>alumno de la EOI' ]
```

JavaScript

Otras funcionalidades de ECMAScript 2015

Array.reduce(función) -> Devuelve el resultado de aplicar la función pasada como parámetro a todos los elementos del array, el resultado de cada operación en cada nodo se le pasa como parámetro al siguiente (Es acumulativo)

```
const array1 = [10, 20, 30, 40, -10];

var resultado = array1.reduce((acumulado, actual) => acumulado + actual), 10);

console.log(resultado);
```

100

JavaScript

Otras funcionalidades de ECMAScript 2015

`Array.join(separador)` -> Devuelve una cadena de texto con todos los elementos del array separador por *separador*

```
const array1 = ["Hola", "soy", "un", "alumno de la EOI"];
console.log(array1.join());
console.log(array1.join(' '));
console.log(array1.join(','));
console.log(array1.join(' '));
```

```
Hola,soy,un,alumno de la EOI
Holasoyunalumno de la EOI
Hola,soy,un,alumno de la EOI
Hola soy un alumno de la EOI
```

JavaScript

Otras funcionalidades de ECMAScript 2015

Array.filter(funcionFiltro) -> Devuelve un array con los resultados de aplicar el filtro pasado como parámetro al array

```
const array1 = ["Hola", "soy", "un", "alumno de la EOI"];  
  
const nuevoArray = array1.filter(element => element.length >= 2 && element.length <= 3);  
  
console.log(nuevoArray);
```

['soy', 'un']

JavaScript

Otras funcionalidades de ECMAScript 2015

`Array.forEach(función)` -> Ejecuta para cada elemento del array la función pasada como parámetro

```
const array1 = ["Hola", "soy", "un", "alumno de la EOI"];

array1.forEach(elemento => {
    console.log(elemento.toUpperCase());
});
```

HOLA
SOY
UN
ALUMNO DE LA EOI

JavaScript

Otras funcionalidades de ECMAScript 2015

Map.forEach(función) -> Ejecuta para cada elemento del Mapa la función pasada como parámetro

```
const map1 = new Map();
map1.set('nombre', 'Mi nombre');
map1.set('datosContacto', { direccion: 'Mi direccion', telefono: 'Mi telefono' });

map1.forEach((valor, clave) => {
  console.log(clave + " " + JSON.stringify(valor));
});
```

```
nombre "Mi nombre"
datosContacto {"direccion":"Mi direccion","telefono":"Mi telefono"}
```

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

