



```
if ($window.scrollTop() > header1_initialDistance /  
    header1.css('padding-top', '0px') + header1_initialPadding)  
    header1.css('padding-top', '' + $window.scrollTop() - header1_initialDistance /  
    header1.css('padding-top', '0px') + header1_initialPadding + 'px');  
}  
} else {  
    header1.css('padding-top', '' + header1_initialPadding + 'px');  
}  
  
if ($window.scrollTop() > header2_initialDistance /  
    header2.css('padding-top', '0px') + header2_initialPadding)  
    header2.css('padding-top', '' + $window.scrollTop() - header2_initialDistance /  
    header2.css('padding-top', '0px') + header2_initialPadding + 'px');
```

# TEMA 8

## INTRODUCCIÓN A

## JAVASCRIPT

Colegio Salesiano San Pedro  
Formación Profesional  
1º CFGS DAM

# ÍNDICE

1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
5. Gestión de Eventos
6. Gestión de Formularios

# 1 Introducción

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

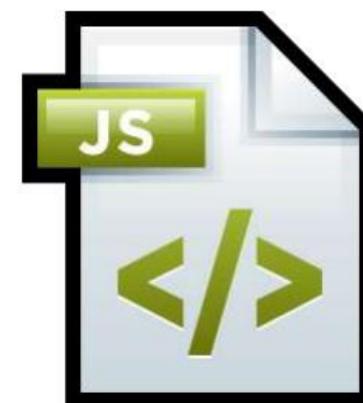
JS es un lenguaje de programación para crear páginas web dinámicas

Una web dinámica es aquella que incorpora efectos y acciones que se activan al pulsar botones y ventanas con mensajes de aviso

JS es un lenguaje INTERPRETADO

El código JS se ejecuta en el navegador, por tanto, en el lado cliente (FRONT-END)

JAVASCRIPT NO TIENE  
NADA QUE VER CON  
JAVA



# 1 Introducción

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

El código JS se incrusta en un documento HTML al igual que se incrusta el CSS

Usando una etiqueta <script>

Mediante archivo externo (preferido).

Usar la opción defer

En los elementos HTML (desaconsejado)

```
<head>
  <title></title>
  <meta charset="utf-8">
  <script>
    alert("esto es un script");
  </script>
  <script src="js/misScripts.js"></script>
</head>
<body>
  <input type="submit" name="enviar" value="Pínchame"
    onclick="alert('¡me han pinchado!');" >
</body>
```

Ya no es necesario incluir el tipo en la etiqueta script

Hay discrepancias sobre dónde es el mejor lugar para situar el código JS. [Mira este artículo](#)

1. *Introducción*
2. *Conceptos básicos*
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. *Operadores*
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 2 Conceptos básicos

## A. Sintaxis básica

La sintaxis en JS es muy parecida a la de otros lenguajes como C o Java

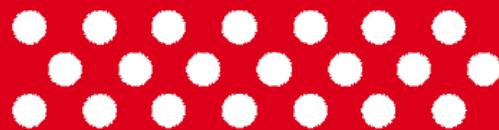
No se tienen en cuenta los espacios en blanco y las nuevas líneas

Se distingue entre mayúsculas y minúsculas

No se define el tipo de las variables (JS es un lenguaje débilmente tipado)

No es obligatorio (pero sí muy recomendable) terminar las sentencias con punto y coma (;

Se pueden escribir comentarios al estilo de Java (con `/**/` o con `//`)



1. *Introducción*
2. *Conceptos básicos*
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. *Operadores*
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 2 Conceptos básicos

## B. Variables

Una variable sirve para almacenar valores.

Se definen con la palabra reservada **var** (antigua) o con **let** (PREFERIBLE). Más adelante se explica la diferencia.

No es necesario indicar el tipo de dato

Reglas para los identificadores:

- Solo admite letras, números, el símbolo \$ y el guion bajo \_
- No puede comenzar por número
- Distingue entre mayúsculas y minúsculas
- Se suele utilizar nomenclatura camelCase

```
var edad = 5;  
let mes = "Enero";
```

Si se declara una variable pero no se inicializa, ésta toma el valor “undefined”

1. *Introducción*
2. *Conceptos básicos*
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. *Operadores*
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 2 Conceptos básicos

## B. Variables

El ALCANCE de una variable indica desde dónde puede ser accedida

En JS clásico el alcance siempre fue GLOBAL o de FUNCIÓN. Un *var* puede tener estos alcances

```
<script>
    var nombre = "Ruperto"; //nombre es GLOBAL
    //aquí puede usarse nombre

    function miFuncion(){
        var edad = 18; //edad es LOCAL
        //aquí puede usarse nombre
        //aquí puede usarse edad
    }

    //aquí no puede usarse edad
</script>
```

1. Introducción
2. Conceptos básicos
  - a. Sintaxis
  - b. Variables
  - c. Tipos
  - d. Operadores
  - e. Control de flujo
3. Funciones
4. Gestión del DOM
5. Gestión de eventos
6. Gestión de formularios

# 2 Conceptos básicos

## B. Variables

Se incluyó *let* para permitir también el alcance de BLOQUE. Por tanto, un *let* puede tener los tres alcances.

```
function miFuncion(){  
    if(true){  
        var edad = 18; //edad es LOCAL  
        let telefono = 666666666; //telefono es DE BLOQUE  
  
        //aquí se puede usar edad y telefono  
    }  
    //aquí se puede usar edad  
    //aquí no se puede usar edad telefono  
}  
  
//aquí no puede usarse edad ni telefono
```

1. *Introducción*
2. *Conceptos básicos*
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. *Operadores*
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 2 Conceptos básicos

## B. Variables

Fuera de un bloque, el comportamiento de var y let es prácticamente el mismo (aunque hay ligeras diferencias)

[Para profundizar en la diferencia entre var y let deberías leer este artículo \(muy recomendado\)](#)

En nuestro código SIEMPRE  
HAY QUE UTILIZAR LET, var  
es para código antiguo



1. Introducción
2. Conceptos básicos
  - a. Sintaxis
  - b. Variables
  - c. Tipos
  - d. Operadores
  - e. Control de flujo
3. Funciones
4. Gestión del DOM
5. Gestión de eventos
6. Gestión de formularios

# 2 Conceptos básicos

## C. Tipos de datos

JS es un lenguaje *débilmente tipado*. Es decir, el tipo no se indica en la declaración de la variable sino que se *adivina*

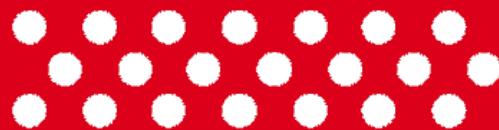
```
let iva = 21;  
let precio = 23.95;
```

Numérico Pueden ser enteros o decimales

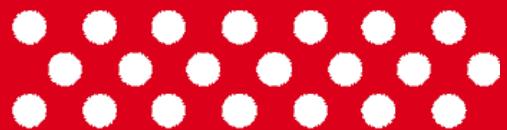
String Se pueden declarar con comillas dobles o simples

Array Colección de variables

Boolean Valores true/false



1. *Introducción*
2. *Conceptos básicos*
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. *Operadores*
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*



# 2 Conceptos básicos

## C. Tipos de datos

### String

Se pueden declarar con comillas simples o dobles, pero no se pueden mezclar

```
Let nombre = "Manolo";
Let apellido = 'Pérez';
Let texto1 = "Una frase con 'comillas simples' dentro";
Let texto2 = 'Una frase con "comillas simples" dentro';
```

| Caracteres especiales |    |
|-----------------------|----|
| Nueva línea           | \n |
| Tabulador             | \t |
| Comilla simple        | \' |
| Comilla doble         | \" |
| Barra inclinada       | \  |

1. *Introducción*
2. *Conceptos básicos*
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. *Operadores*
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 2 Conceptos básicos

## C. Tipos de datos

### Arrays

Un array es una colección de variables. Según conozcamos sus valores o no, se declara de una manera u otra

```
Let nombre_array = [valor1, valor2, ..., valorN]; //array estático  
Let miArray = new Array(); //array dinámico
```

En el array dinámico, no es necesario indicar el tamaño

Se pueden mezclar datos de diferentes tipos en un array

```
Let dias = ["Lunes", "Martes", 3, "Jueves", "Viernes", "Sábado", "Domingo"];  
Let diaSeleccionado = dias[4];
```

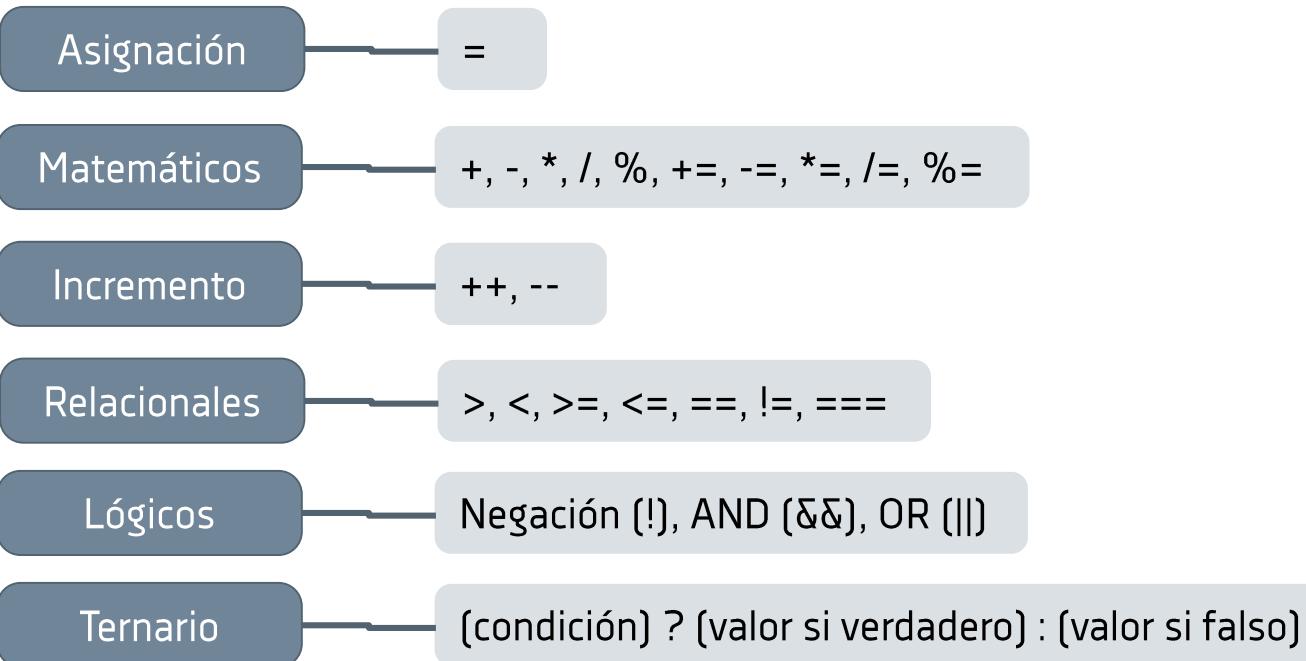
Para acceder a un elemento del array, se usa [] (recuerda que la numeración comienza en 0)



1. *Introducción*
2. ***Conceptos básicos***
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. ***Operadores***
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 2 Conceptos básicos

## D. Operadores



1. *Introducción*
2. *Conceptos básicos*
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. *Operadores*
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 2 Conceptos básicos

## D. Operadores

Debido a la ausencia de tipos, el intérprete JS “adivina” los tipos de los operandos conforme va leyendo de izquierda a derecha. Por lo que los resultados pueden variar según se escriba la sentencia

```
let resultado = 12 + 'hola';//resultado = "12hola"  
resultado = 12 + true;//resultado = 13  
resultado = "la operación es" + false;//resultado = "la operación es false"  
resultado = "hola" + 12 + 4; //resultado="hola124"  
resultado = 12 + 4 + "hola"; //resultado="16hola"
```

Cuando se convierte un booleano a número, false = 0, true = 1.

En el penúltimo ejemplo, como primero se interpreta un string, los números se convierten a string.

En el último ejemplo, como primero se interpretan números, se hace la suma, pero después, al encontrarse un string, convierte el resultado parcial a string.



1. *Introducción*
2. *Conceptos básicos*
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. *Operadores*
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

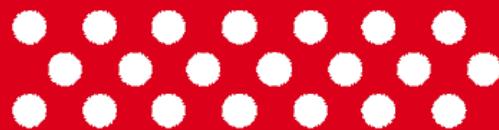
# 2 Conceptos básicos

## D. Operadores

Una comparación con == trata de convertir los dos operandos al mismo tipo

Una comparación estricta con === compara valores y tipos, por lo que deben ser ambos iguales

```
if(13=='13'){
    //este código se ejecuta
}
if(13==='13'){
    //este código no se ejecuta
}
```



1. Introducción
2. Conceptos básicos
  - a. Sintaxis
  - b. Variables
  - c. Tipos
  - d. Operadores
  - e. Control de flujo
3. Funciones
4. Gestión del DOM
5. Gestión de eventos
6. Gestión de formularios

# 2 Conceptos básicos

## E. Control de flujo

if

```
if(condicion){  
    //su código aquí  
}
```

if ...else

```
if (condicion) {  
    //su código aquí  
} else {  
    //su código aquí  
}
```

switch

```
switch(variable){  
    case 1:  
        //...  
        break;  
  
    case 2:  
        //...  
        break;  
    //...  
    case 10:  
        //...  
        break;  
    default:  
        //...  
        break;  
}
```

Para entrar en cada caso del switch se utiliza comparación estricta (==), por lo que deben coincidir tipo y valor

Si varios casos coinciden, solo se ejecuta el primero

No olvidar nunca el break al final de cada caso, ni el caso default

1. *Introducción*
2. *Conceptos básicos*
  - a. *Sintaxis*
  - b. *Variables*
  - c. *Tipos*
  - d. *Operadores*
  - e. *Control de flujo*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 2 Conceptos básicos

## E. Control de flujo

while

```
while(condicion){  
    //su código  
}
```

do while

```
do{  
    //su código  
}while(condicion);
```

for

```
for(inicializacion;condicion;actualizacion){  
    //su código  
}  
  
for(let i=0;i<10;i++){  
    alert(i);  
}
```

1. Introducción
2. Conceptos básicos
  - a. Sintaxis
  - b. Variables
  - c. Tipos
  - d. Operadores
  - e. Control de flujo
3. Funciones
4. Gestión del DOM
5. Gestión de eventos
6. Gestión de formularios

# 2 Conceptos básicos

## E. Control de flujo

for in

```
for(indice in array){  
    // ...  
}
```

El bucle for in sirve para recorrer las propiedades de un objeto

for of

```
let language = "JavaScript";  
let text = "";  
  
for (let x of language) {  
    text += x + "<br>";  
}
```

No usarlo con arrays

El bucle for of sirve para recorrer un elemento iterable (array, string, NodeList) en orden

1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. *Para strings*
  - c. *Para arrays*
  - d. *Para números*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 3 Funciones

## A. Función básica

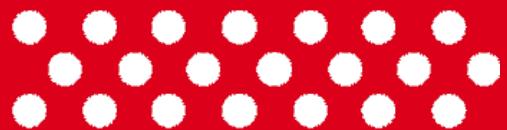
Una función es un bloque de código diseñado para realizar una tarea en concreto cuando es invocado.

En JS son al mismo tiempo funciones y procedimientos,  
por lo que pueden o no tener una sentencia return

Al no existir el tipado, la declaración es muy sencilla

```
function miFuncion(parametro1, parametro2){  
    //operaciones a realizar  
  
    return valor; //esta línea es opcional  
}
```

1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. **Para strings**
  - c. *Para arrays*
  - d. *Para números*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*



# 3 Funciones

## B. Funciones para operar con cadenas de caracteres

length

Es una propiedad que indica la longitud de la cadena

```
Let mensaje = "hola hola vecinito";  
Let numCaracteres = mensaje.length;
```

toUpperCase()

Convierten una cadena a todo mayúsculas o minúsculas

toLowerCase()

```
Let mensaje = "hola hola vecinito";  
mensaje.toUpperCase();
```

charAt(pos)

Devuelve el carácter situado en la posición indicada  
(recuerda que empieza a contar por 0)

```
Let mensaje = "hola hola vecinito";  
Let letra = mensaje.charAt(5); //letra = 'h'
```

1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. **Para strings**
  - c. *Para arrays*
  - d. *Para números*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 3 Funciones

## B. Funciones para operar con cadenas de caracteres

indexOf(c)

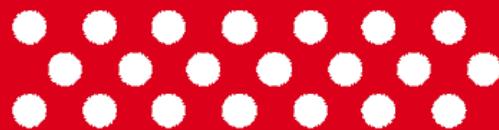
Devuelve la posición de la primera ocurrencia del carácter en la cadena. Si no está, devuelve -1

```
let mensaje = "hola hola vecinito";
let posicion = mensaje.indexOf('l');//posicion=2
```

substring[inicio,final]

Devuelve la porción de la cadena entre las posiciones indicadas. El final es opcional (si no está, final = length)

```
Let mensaje = "hola hola vecinito";
Let hola = mensaje.substring(5,8);//es el segundo 'hola'
Let holaVecinito = mensaje.substring(5);
```



1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. **Para strings**
  - c. *Para arrays*
  - d. *Para números*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

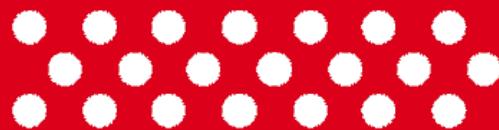
# 3 Funciones

## B. Funciones para operar con cadenas de caracteres

split(separador)

Convierte una cadena en un array, según el carácter separador que se indique. El separador se elimina del resultado

```
Let mensaje = "hola hola vecinito";
Let palabras = mensaje.split(" "); //palabras=['hola','hola','vecinito']
Let letras = mensaje.split("");
//[['h','o','l','a',' ','h','o','l','a',' ','v','e','c','i','n','i','t','o']]
Let cosaRara = mensaje.split('h');//cosaRara = ['ola ','ola vecinito']
```



1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. *Para strings*
  - c. **Para arrays**
  - d. *Para números*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 3 Funciones

## C. Funciones para operar con arrays

length

Es una propiedad que indica la longitud del array

```
let dias = ['lunes','martes','miércoles','jueves','viernes','sábado','domingo'];
let numDias = dias.length; //numDias = 7
```

concat()

Concatena los elementos de varios arrays

```
let laborables = ['lunes','martes','miércoles','jueves','viernes'];
let finde = ['sábado','domingo'];
let semana = laborables.concat(finde);
```

join(separador)

Une todos los elementos de un array para formar un string (es lo contrario a split()). Para unir los elementos se utiliza el carácter separador

```
let palabras = ['hola','hola','vecinito'];
let frase = palabras.join(' ');
```

1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. *Para strings*
  - c. **Para arrays**
  - d. *Para números*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 3 Funciones

## C. Funciones para operar con arrays

pop()

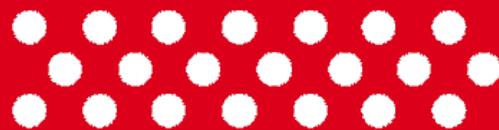
Elimina el último elemento del array y lo devuelve. El array original y su longitud disminuye en 1 elemento

```
let dias = ['lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado', 'domingo'];
let ultimo = dias.pop(); //ultimo = 'domingo'. dias.length = 6
```

push()

Agrega un elemento al final de un array, incrementando en 1 su tamaño

```
let dias = ['lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado', 'domingo'];
let ultimo = dias.pop(); //ultimo = 'domingo'. dias.length = 6
dias.push(ultimo); //la semana vuelve a estar completa
```



1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. *Para strings*
  - c. **Para arrays**
  - d. *Para números*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 3 Funciones

## C. Funciones para operar con arrays

shift()

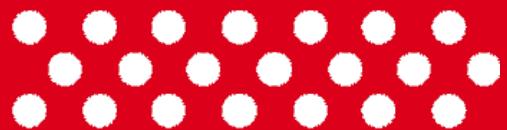
Elimina el primer elemento del array y lo devuelve. El array original y su longitud disminuye en 1 elemento

```
let dias = ['lunes','martes','miércoles','jueves','viernes','sábado','domingo'];
let primero = dias.shift(); //primero = 'lunes'. dias.length = 6
```

unshift()

Agrega un elemento al inicio de un array, incrementando en 1 su tamaño

```
let dias = ['lunes','martes','miércoles','jueves','viernes','sábado','domingo'];
let primero = dias.shift(); //primero = 'lunes'. dias.length = 6
dias.unshift(primer); //la semana vuelve a estar completa
```



1. Introducción
2. Conceptos básicos
3. **Funciones**
  - a. function
  - b. Para strings
  - c. **Para arrays**
  - d. Para números
4. Gestión del DOM
5. Gestión de eventos
6. Gestión de formularios

# 3 Funciones

## C. Funciones para operar con arrays

reverse()

Invierte el orden de los elementos de un array.

```
let dias = ['lunes','martes','miércoles','jueves','viernes','sábado','domingo'];
let said = dias.reverse();
```

splice()

Funciona como una navaja suiza para arrays. Puede hacer todo: insertar, remover y remplazar elementos

splice(start [,deleteCount, elem1,...elemN])

Comenzando en start, elimina los elementos indicados en deleteCount y añade los elementos indicados en elem1...elemN

```
let mensaje = ["Quiero","estudiar","JS","ahora","mismo"];
mensaje.splice(2,1,"Entornos"); // "Quiero estudiar Entornos ahora mismo"
mensaje.splice(0,0,"No");// "No quiero estudiar Entornos ahora mismo"
mensaje.splice(2,2,'comerme','una','pizza');// "No quiero comerme una pizza ahora mismo"
```



1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. *Para strings*
  - c. **Para arrays**
  - d. *Para números*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 3 Funciones

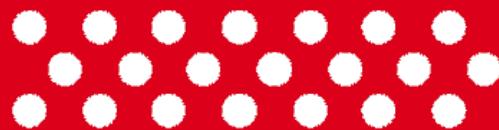
## C. Funciones para operar con arrays

indexOf()

Devuelve el índice de la primera (o última) ocurrencia de un elemento

lastIndexOf()

[Aquí puedes consultar más métodos para arrays](#)



1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. *Para strings*
  - c. *Para arrays*
  - d. **Para números**
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 3 Funciones

## D. Funciones para operar con números

isNaN()

Devuelve un boolean indicando si una variable contiene un valor NaN

Nan (NotANumber) es un valor especial que toman las variables numéricas cuando se realizan operaciones que producen indeterminaciones (por ejemplo 0/0)

```
if(isNaN(num1/num2)){  
    alert("Esta operación produce un resultado indeterminado")  
}else{  
    alert(num1/num2);  
}
```

isFinite()

Devuelve un boolean indicando si una variable contiene un valor numérico diferente de Nan y Infinity

Infinity es un valor especial que representa un número más grande que cualquier otro. Se obtiene al hacer operaciones que dan resultado infinito (por ejemplo dividir por 0). Puede ser negativo

```
if(isFinite(num1/0)){  
}else{  
    //aquí no se entra nunca  
}
```

1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. *Para strings*
  - c. *Para arrays*
  - d. **Para números**
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 3 Funciones

## D. Funciones para operar con números

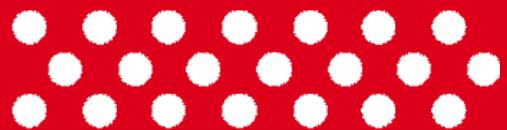
`toString(base)`

Convierte un número en un string. El número se representa en la base indicada (si no se indica, es en base decimal)

```
Let numero = 1023;  
Let cadena = numero.toString(); //cadena = "1023"  
cadena = numero.toString(16); //cadena = "3fff"  
cadena = 1023..toString(); //cadena = "1023" ¡¡OJO a los dos puntos!!
```

`toFixed(numDecimales)`

Devuelve un string con el número redondeado con la cantidad de decimales expresados en numDecimales



1. *Introducción*
2. *Conceptos básicos*
3. **Funciones**
  - a. *function*
  - b. *Para strings*
  - c. *Para arrays*
  - d. **Para números**
4. *Gestión del DOM*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 3 Funciones

## D. Funciones para operar con números

Math.floor(n)

Devuelve el entero más cercano por debajo de un número decimal

Math.ceil(n)

Devuelve el entero más cercano por encima de un número decimal

Math.round(n)

Devuelve el entero más cercano a un número decimal (redondeo)

Math.trunc(n)

Devuelve la parte entera de un número decimal (sin redondeo)

```
let num = 10.635;  
let res = Math.floor(num); //res = 10  
res = Math.ceil(num); //res = 11  
res = Math.round(num); //res = 11  
res = Math.trunc(num); //res = 10  
//si se quiere redondear a una cantidad de decimales  
res = Math.round(num*100)/100; //res = 10.64
```

[Más funciones con números aquí](#)

1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
5. Gestión de eventos
6. Gestión de formularios

# 4 Gestión del Documento (DOM)

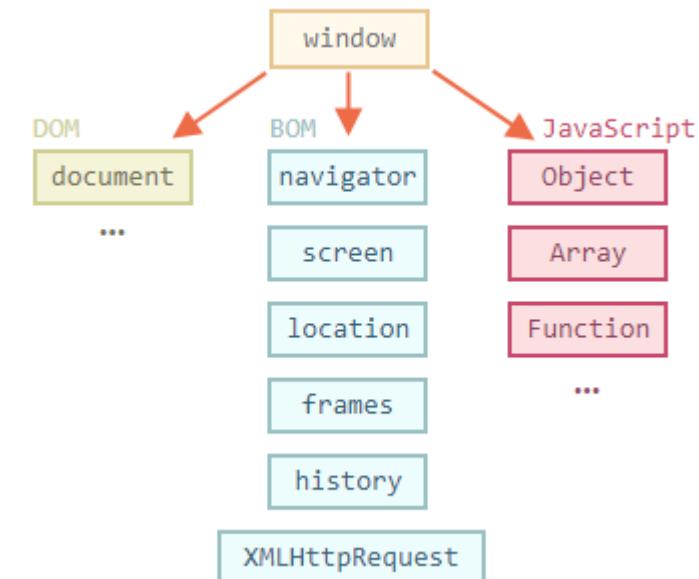
## A. Árbol del DOM

[Artículo de referencia](#)

El objeto global *window* representa la “ventana del navegador”, proporcionando métodos para controlarla

El objeto global *document* representa al contenido de la página, por lo que podemos modificar cualquier elemento HTML

Los métodos definidos en el estándar DOM permiten recorrer todo el documento para modificar el contenido (HTML) y la apariencia (CSS)



1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
  - a. *Árbol del DOM*
  - b. *Tipos de nodos*
  - c. *Acceso a los nodos*
  - d. *Crear nodos*
  - e. *Eliminar nodos*
  - f. *Acceso a atributos*
  - g. *Acceso a CSS*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 4 Gestión del Documento (DOM)

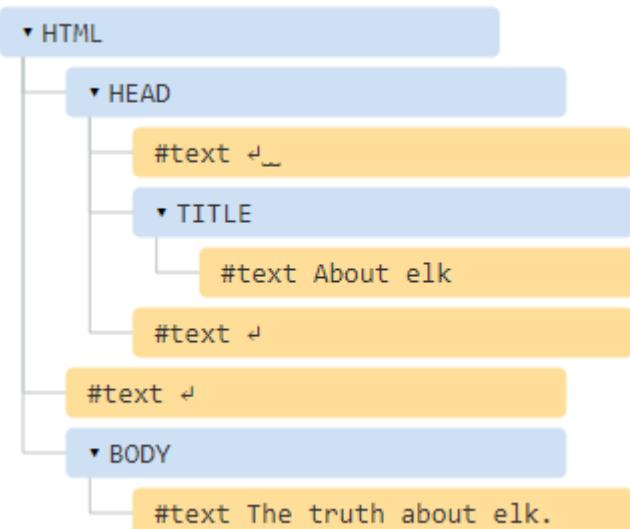
## A. Árbol del DOM

[Artículo de referencia](#)

La estructura de un documento HTML son sus etiquetas. Cada etiqueta es un objeto en JS

Por tanto el DOM puede entenderse como un árbol lleno de nodos y ramas

```
<!DOCTYPE HTML>
<html>
<head>
  <title>About elk</title>
</head>
<body>
  The truth about elk.
</body>
</html>
```



El contenido (texto) de las etiquetas también es un nodo, por lo que también es un objeto

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. ***Gestión del DOM***
  - a. *Árbol del DOM*
  - b. ***Tipos de nodos***
  - c. *Acceso a los nodos*
  - d. *Crear nodos*
  - e. *Eliminar nodos*
  - f. *Acceso a atributos*
  - g. *Acceso a CSS*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 4 Gestión del Documento (DOM)

## B. Tipos de nodos

[Artículo de referencia](#)

El estándar define 12 tipos de nodos, nosotros nos centramos en 5

Document

Es el nodo raíz, del que derivan todos los demás

Element

Representa una etiqueta HTML, por tanto puede contener otras etiquetas, atributos y texto

Attr

Representa un atributo de una etiqueta (por ejemplo href)

Text

Es el contenido (texto plano) de una etiqueta

Comment

Representa un comentario en el documento

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. ***Gestión del DOM***
  - a. *Árbol del DOM*
  - b. *Tipos de nodos*
  - c. ***Acceso a los nodos***
  - d. *Crear nodos*
  - e. *Eliminar nodos*
  - f. *Acceso a atributos*
  - g. *Acceso a CSS*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 4 Gestión del Documento (DOM)

## C. Acceso a los nodos

[Artículo de referencia](#)

A los nodos se puede acceder recorriendo el árbol o *preguntando por ellos*

`getElementById`

Devuelve el elemento con el id especificado como parámetro

OJOUIDAO: los ids deben ser únicos en el documento. Si un documento tiene varios elementos con el mismo id, es impredecible cuál de todos se devolverá

```
Let elemento = document.getElementById("primerParrafo");
```

También existen los métodos `getElementsByName()`, `getElementsByClassName()` o `getElementsByTagName()`, pero han caído en desuso. Hoy preferimos usar el método `querySelectorAll` que se explica a continuación



1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
5. Gestión de eventos
6. Gestión de formularios

# 4 Gestión del Documento (DOM)

## C. Acceso a los nodos

[Artículo de referencia](#)

### querySelectorAll

Devuelve una colección con todos los elementos que coinciden con el selector CSS indicado.

```
let elementos = document.querySelectorAll("header nav li");
let elemEnHover = document.querySelectorAll("header nav li:hover");//admite pseudoclases!!
```

Al ser una colección,  
debo recorrerla usando  
un bucle for..of

```
let elementos = document.querySelectorAll("header nav li");
for(let elem of elementos){
    alert(elem.innerHTML);
}
```

### querySelector

Busca el primer elemento que coincide con el selector CSS indicado.

Es equivalente a querySelectorAll(css)[0], pero es útil cuando sabes que solo hay un elemento que lo cumple, ya que será más rápido.



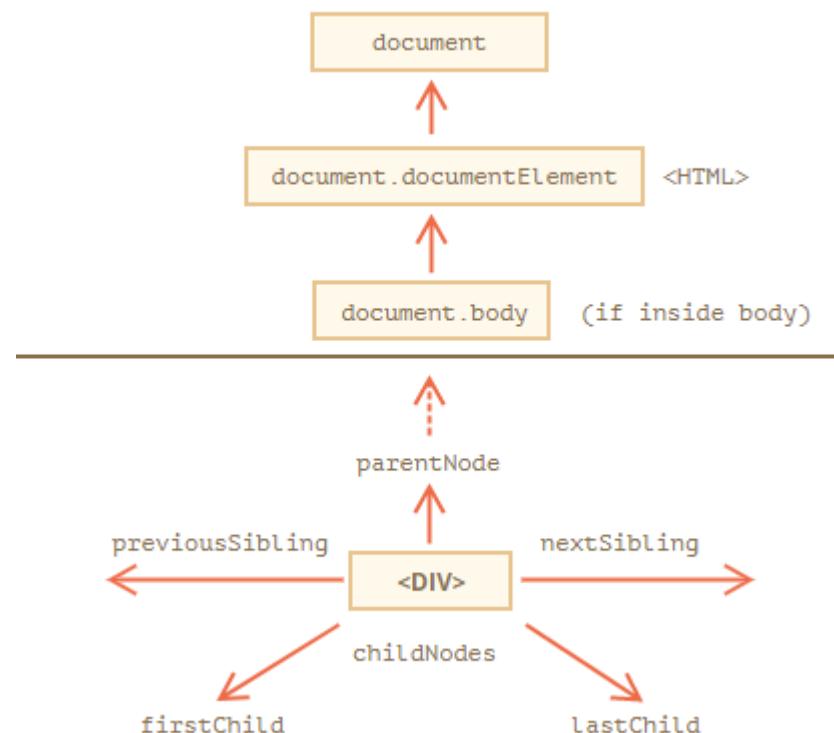
1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
5. Gestión de eventos
6. Gestión de formularios

# 4 Gestión del Documento (DOM)

## C. Acceso a los nodos

[Artículo de referencia](#)

Todos los elementos tienen una colección de sus hijos, por lo que puedo recorrerlo



Además, puedo acceder directamente al padre, a los hermanos inmediatamente anterior y siguiente, y al primer y último hijo

```
let elemento = document.getElementById("primerParrafo");
let padre = elemento.parentNode;
let hermanoAnterior = elemento.previousSibling;
let hermanoSiguiente = elemento.nextSibling;
let hijos = elemento.childNodes;
for(let hijo of hijos){
    //voy recorriendo todos los hijos
}
let primerHijo = elemento.firstChild;
let ultimoHijo = elemento.lastChild;
```

OJO CUIDAO: son propiedades, no son métodos

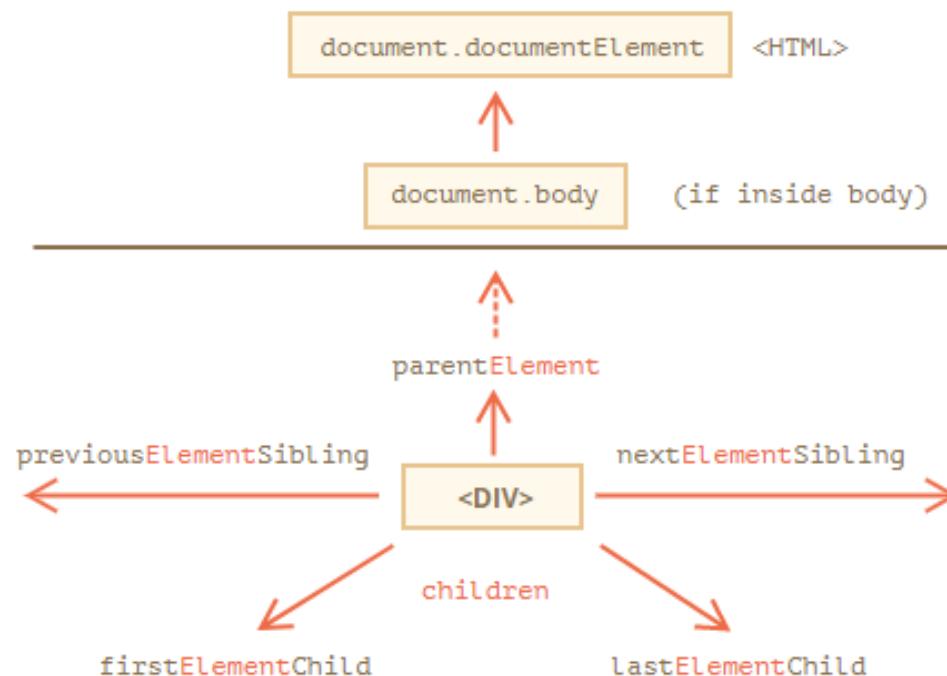
1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
5. Gestión de eventos
6. Gestión de formularios

# 4 Gestión del Documento (DOM)

## C. Acceso a los nodos

[Artículo de referencia](#)

Lo anterior devuelve cualquier tipo de nodo. Puedo navegar directamente por los nodos de tipo elemento



Así eliminamos los nodos de texto y de comentarios

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. ***Gestión del DOM***
  - a. *Árbol del DOM*
  - b. *Tipos de nodos*
  - c. *Acceso a los nodos*
  - d. ***Crear nodos***
  - e. *Eliminar nodos*
  - f. *Acceso a atributos*
  - g. *Acceso a CSS*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 4 Gestión del Documento (DOM)

## D. Creación de nodos

[Artículo de referencia](#)

Para añadir contenido al DOM hay que hacerlo en tres pasos:

1. Crear el nodo
2. Añadirle contenido al nodo
3. Añadir el nodo al documento

```
let div = document.getElementById('contenedor');
let nuevoP = document.createElement('p');
nuevoP.innerHTML = "Hola hola vecinito"; //aquí se puede escribir cualquier cosa en HTML
div.append(nuevoP);
```

Con `document.createElement(tag)` creo un elemento con la etiqueta indicada

Una vez creado, relleno el HTML interno (`innerHTML`). También existe el método `createTextNode` pero es poco utilizado

El nuevo nodo hay que añadirlo a alguien, por lo que llamo al método `append` DEL NODO PADRE (el que va a contenerlo)

1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
5. Gestión de eventos
6. Gestión de formularios

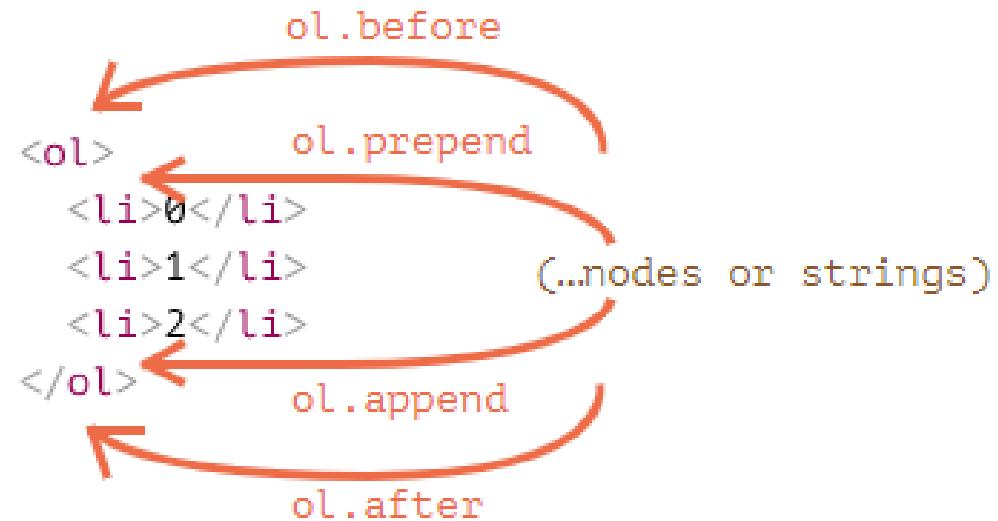
# 4 Gestión del Documento (DOM)

## D. Creación de nodos

[Artículo de referencia](#)

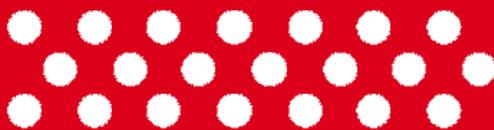
Para añadir nodos a otro nodo existen los siguientes métodos:

- append
- Prepend
- Before
- After
- replaceWith



Como parámetro puedo pasar un nodo o una cadena de caracteres.

Si paso un string, se añade texto plano, por lo que si lleva etiquetas HTML, estas son escapadas y, por tanto, ignoradas



1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
5. Gestión de eventos
6. Gestión de formularios

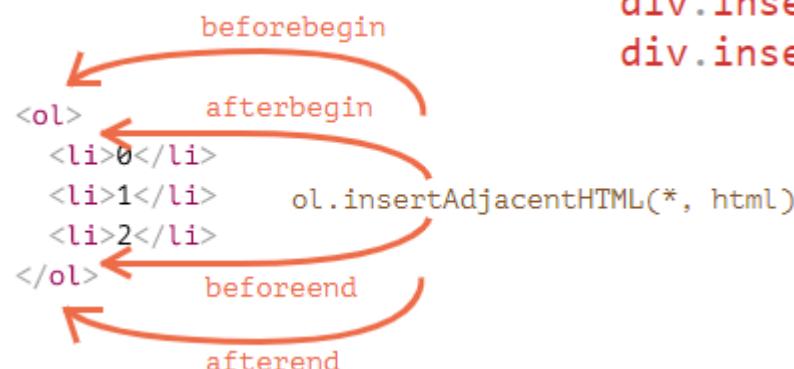
# 4 Gestión del Documento (DOM)

## D. Creación de nodos

[Artículo de referencia](#)

Otra opción es usar el método elem.insertAdjacentHTML(where, html). El primer argumento es una palabra clave que indica dónde insertar. El segundo es un string con el código HTML

- Beforebegin
- Afterbegin
- Beforeend
- afterend



```
div.insertAdjacentHTML('beforebegin', '<p>Hola</p>');
div.insertAdjacentHTML('afterend', '<p>Adiós</p>');
```

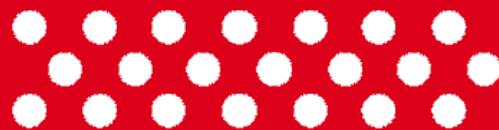
1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. ***Gestión del DOM***
  - a. *Árbol del DOM*
  - b. *Tipos de nodos*
  - c. *Acceso a los nodos*
  - d. ***Crear nodos***
  - e. *Eliminar nodos*
  - f. *Acceso a atributos*
  - g. *Acceso a CSS*
5. *Gestión de eventos*
6. *Gestión de formularios*

# 4 Gestión del Documento (DOM)

## D. Creación de nodos

[Artículo de referencia](#)

Existen métodos antiguos que son parent.appendChild(), parent.insertBefore(), parent.removeChild() y parent.replaceChild(). Pueden servirnos para comprender código antiguo, pero hoy en día no se utilizan.



1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
5. Gestión de eventos
6. Gestión de formularios

# 4 Gestión del Documento (DOM)

## D. Creación de nodos

[Artículo de referencia](#)

Otro método interesante es elem.cloneNode(boolean)

```
<div class="alert">
    <strong>Hola</strong> Tienes un mensaje por leer
</div>

<script>
    let div = document.querySelector(".alert");
    let div2 = div.cloneNode(true); // clona el div
    div2.querySelector('strong').innerHTML = 'Adiós'; // cambia
    el contenido del clon
    div.after(div2); // coloca el clon después del div
</script>
```

Si el parámetro es true, se clona al completo, incluyendo clases, nodos hijos etc.

Si el parámetro es false, el clon no incluye los nodos hijos.

1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
5. Gestión de eventos
6. Gestión de formularios

# 4 Gestión del Documento (DOM)

## E. Eliminar Nodos

[Artículo de referencia](#)

Para eliminar un nodo llamo a node.remove()

OJOUIDAO: remove() se llama desde el mismo nodo que quieras eliminar.

`div2.remove(); //esto sigue del ejemplo anterior`

Antiguamente se usaba removeChild(), que debía ser llamado desde el nodo padre

OJOUIDAO: si quieres mover un nodo de sitio, no hace falta borrarlo y añadirlo de nuevo.  
Simplemente añádelo en el nodo de destino y se borrará del origen

```
<div id="uno">Un párrafo</div>
<div id="dos">Otro párrafo</div>
<script>
  let uno = document.querySelector('#uno');
  let dos = document.querySelector('#dos');
  dos.after(uno); // toma el primero y lo coloca después del dos,
  sin necesidad de eliminarlo antes
</script>
```

1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
5. Gestión de eventos
6. Gestión de formularios

# 4 Gestión del Documento (DOM)

## F. Acceso a los atributos de un nodo

node.tagName

Con la propiedad tagName obtengo un string con el nombre de la etiqueta

```
let etiqueta = uno.tagName; //etiqueta = 'DIV'
```

node.hidden

Es una propiedad boolean que hace el mismo efecto que display:none;

```
dos.hidden = true; //ahora me ves
```

```
dos.hidden = false; //ahora no me ves
```

En general, puedo acceder a cualquier atributo de un elemento HTML como si fuera una propiedad del nodo, utilizando la notación punto

```
enlace.href="https://www.miweb.com";  
unInput.value="valor del casillero";  
unInput.placeholder="valor del placeholder";
```

OJOUIDAO: cada etiqueta HTML tiene sus atributos y, por tanto, cada nodo sus propiedades

OJOUIDAO: para acceder a la clase debo usar className

- 1. Introducción
- 2. Conceptos básicos
- 3. Funciones
- 4. Gestión del DOM
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. Acceso a CSS
- 5. Gestión de eventos
- 6. Gestión de formularios

# 4 Gestión del Documento (DOM)

## F. Acceso a los atributos de un nodo

Otra forma de acceder a los atributos es mediante los métodos de atributos en vez de mediante propiedades

elem.hasAttribute(nombre)

elem.getAttribute(nombre)

elem.setAttribute(nombre,valor)

elem.removeAttribute(nombre)

elem.attributes

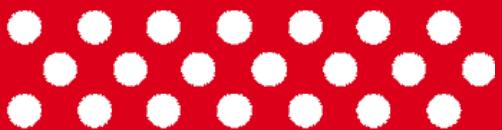
```
<a href="#inicio">Ir arriba</a>
<script>
  let enlace = document.querySelector('a');
  let urlEnlace = enlace.getAttribute('href');
  //urlEnlace='#inicio'
  let urlCompleta = enlace.href;
  //urlCompleta='https://laruta.que/fuera#inicio'
</script>
```

Estos métodos son interesantes para crear atributos propios

A veces no hay el mismo valor en el atributo que en la propiedad (por ejemplo, en href)

Otra opción es usar el dataset. Aprende la diferencia entre las tres formas en este enlace

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. **Gestión del DOM**
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. **Acceso a CSS**
5. Gestión de eventos
6. Gestión de formularios



# 4 Gestión del Documento (DOM)

## G. Acceso a las propiedades de estilo

[Artículo de referencia](#)

Recuerda que un estilo puede estar declarado en una clase o directamente en el elemento HTML (y que esto último está desaconsejado)

Lo mejor es tener declaradas varias clases en la hoja CSS y, mediante JS, añadir o quitar las clases a los elementos

elem.className

El valor del atributo 'class' de un elemento HTML.

```
<div id="cajaRoja" class="bg-danger col-3">Una caja roja</div>
<script>
  let caja = document.getElementById('cajaRoja');
  caja.className = 'color-light';
  //le cambia el color al texto pero ¡ojo! le quita el bg-danger y el col-3
  caja.className += 'color-light';//¡¡mejor así!!
</script>
```

OJO CUIDAO:  
cuando se edita directamente se machaca lo que tuviera

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. **Gestión del DOM**
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. **Acceso a CSS**
5. Gestión de eventos
6. Gestión de formularios

# 4 Gestión del Documento (DOM)

## G. Acceso a las propiedades de estilo

[Artículo de referencia](#)

elem.classList

Es una colección con todas las clases que posee. Tiene métodos para acceder a ellas. Es una manera más segura de modificar las clases

elem.classList.contains("class")

elem.classList.add("class")

elem.classList.remove("class")

elem.classList.toggle("class")

Es un objeto iterable (permite usar un for..of)

```
<div id="cajaRoja" class="bg-danger col-3">Una caja roja</div>
```

```
<script>
```

```
let caja = document.getElementById('cajaRoja');  
caja.classList.remove('bg-danger');  
caja.classList.add('bg-success');
```

```
</script>
```

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. **Gestión del DOM**
  - a. Árbol del DOM
  - b. Tipos de nodos
  - c. Acceso a los nodos
  - d. Crear nodos
  - e. Eliminar nodos
  - f. Acceso a atributos
  - g. **Acceso a CSS**
5. Gestión de eventos
6. Gestión de formularios

# 4 Gestión del Documento (DOM)

## G. Acceso a las propiedades de estilo

[Artículo de referencia](#)

Para modificar una sola propiedad CSS, puedo usar  
`elem.style.nombrePropiedad`

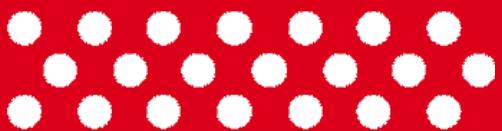
OJOUIDAO: esto modifica directamente el estilo incrustado en la etiqueta

Cuando una propiedad tiene un guion, se convierte en mayúscula  
(margin-top → marginTop)

OJOUIDAO: las propiedades con unidades se parsean como string para poder añadir la unidad

Esta técnica solo debería usarse cuando calculas dinámicamente el valor de la propiedad (por ejemplo, un tamaño)

```
caja.style.marginTop = 10; // ¡¡ESTO ESTÁ MAL!!  
caja.style.marginTop = '10px'; // ¡¡ESTO ESTÁ BIEN!!  
caja.style.fontSize = '1.5em';
```



1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. ***Gestión del DOM***
  - a. *Árbol del DOM*
  - b. *Tipos de nodos*
  - c. *Acceso a los nodos*
  - d. *Crear nodos*
  - e. *Eliminar nodos*
  - f. *Acceso a atributos*
  - g. ***Acceso a CSS***
5. *Gestión de eventos*
6. *Gestión de formularios*

# 4 Gestión del Documento (DOM)

## G. Acceso a las propiedades de estilo

[Artículo de referencia](#)

El problema de usar elem.style es que me da únicamente el estilo de la etiqueta. ¿Cómo puedo conocer los estilos provenientes de la cascada CSS?

`getComputedStyle(elemento)`

Devuelve un objeto con todas las propiedades aplicadas. Es de solo lectura

```
Let caja = document.getElementById('cajaRoja');  
Let estilosAplicados = getComputedStyle(caja);  
Let margenTop = estilosAplicados.marginTop;
```

Se le debe indicar el nombre completo de la propiedad. Por ejemplo, no puedo pedir a secas "padding", ya que no sabemos cuál de todos devolverá. Hay que indicar "paddingTop", "paddingLeft"...

Cuidado con usar width/height a secas. [Mejor usar las propiedades geométricas](#)

- 1. *Introducción*
- 2. *Conceptos básicos*
- 3. *Funciones*
- 4. *Gestión del DOM*
- 5. ***Gestión de eventos***
  - a. *Definiciones*
  - b. *Manejadores de eventos*
  - c. *Propiedades de los eventos*
  - d. *Tipos de eventos*
  - e. *Propagación y delegación*
- 6. *Gestión de formularios*

# 5 Gestión de eventos

## A. Definiciones

Un evento es una señal de que algo ocurrió en el navegador.

El evento por sí solo no sirve. Necesito capturarlo y gestionarlo con un controlador (*handler*)

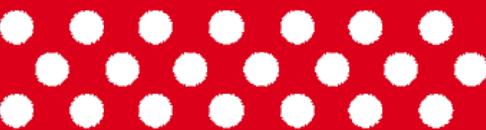
Un controlador (*handler*) es una función que es llamada cuando se lanza un evento

Los eventos más frecuentes son los que están relacionados con el ratón (moverse, click, etc.), con cualquier tipo de puntero (táctil o ratón), con el teclado (pulsaciones de teclas) y con formularios (focus y envío)

El procedimiento habitual será:

1. Crear una función *handler* para un evento
2. Vincular el controlador a un evento sobre un elemento HTML

- 1. Introducción
- 2. Conceptos básicos
- 3. Funciones
- 4. Gestión del DOM
- 5. Gestión de eventos
  - a. Definiciones
  - b. Manejadores de eventos
  - c. Propiedades de los eventos
  - d. Tipos de eventos
  - e. Propagación y delegación
- 6. Gestión de formularios



# 5 Gestión de eventos

## B. Manejadores de eventos

3 maneras de manejar un evento

Como atributo HTML (desaconsejado)

Como propiedad DOM (desaconsejado)

Con el método addEventListener() (recomendado)

Como atributo HTML (desaconsejado)

```
<script>
  function cuentaOvejitas() {
    for(let i=1; i<=3; i++) {
      alert("Ovejita número " + i);
    }
  }
</script>
```

```
<input type="button" onclick="cuentaOvejitas()" value="¡Ayúdame a dormir!">
```

No permite la separación entre contenido, presentación y JS

- 1. Introducción
- 2. Conceptos básicos
- 3. Funciones
- 4. Gestión del DOM
- 5. Gestión de eventos
  - a. Definiciones
  - b. Manejadores de eventos
  - c. Propiedades de los eventos
  - d. Tipos de eventos
  - e. Propagación y delegación
- 6. Gestión de formularios

# 5 Gestión de eventos

## B. Manejadores de eventos

Como propiedad DOM (desaconsejado)

Todos los objetos Element del DOM tienen como propiedades todos los eventos que pueden capturar

A la propiedad se le asigna como valor el nombre del método controlador (¡¡SIN PARÉNTESIS!!)

```
let boton = document.querySelectorAll('input[type="button"]')[0];
boton.onclick=cuentaOvejitas;

function cuentaOvejitas() {
  for(let i=1; i<=3; i++) {
    alert("Ovejita número " + i);
  }
}
```

Tiene dos problemas

Solo admite un manejador por evento

Puede causar valores impredecibles si no se gestiona bien la carga del script

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. ***Gestión de eventos***
  - a. *Definiciones*
  - b. ***Manejadores de eventos***
  - c. *Propiedades de los eventos*
  - d. *Tipos de eventos*
  - e. *Propagación y delegación*
6. *Gestión de formularios*

# 5 Gestión de eventos

## B. Manejadores de eventos

`elem.addEventListener(evento,handler [,opciones])`

El nombre del evento se indica sin el “on” (por ejemplo, “click”)

El nombre del método controlador va sin paréntesis

La opción más interesante es ‘once:true’

De esta forma puedo asignar varios controladores a un evento

Es la forma más segura de asignar controladores

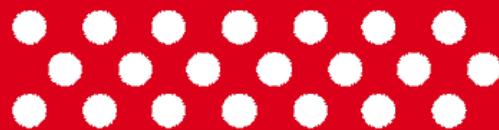
`elem.removeEventListener(evento,handler)`

Elimina un controlador de un evento

```
<input type="button" value="¡Ayúdame a dormir!">
<script>
let boton = document.querySelectorAll('input[type="button"]')[0];
boton.addEventListener('click',cuentaOvejitas,'once:true');
```



1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
5. Gestión de eventos
  - a. Definiciones
  - b. Manejadores de eventos
  - c. Propiedades de los eventos
  - d. Tipos de eventos
  - e. Propagación y delegación
6. Gestión de formularios



# 5 Gestión de eventos

## C. Propiedades de los eventos

this

En un manejador, this hace referencia al elemento DOM que lanzó el evento

event

Al handler se le puede pasar el objeto del evento. Este objeto contiene bastante información útil (quién generó el evento, la posición del cursor, la tecla pulsada...)

```
function ratonMoviéndose(event) {  
    let posX = event.clientX;  
    let posY = event.clientY;  
}
```

event.type

String con el tipo de evento (por ejemplo, 'click')

event.currentTarget

Elemento que lanzó el evento (equivale a usar this en el manejador)

Según el tipo de evento, nos encontraremos unas u otras propiedades. Los de puntero llevan información sobre las coordenadas. Los de teclado sobre la tecla pulsada.

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. ***Gestión de eventos***
  - a. *Definiciones*
  - b. *Manejadores de eventos*
  - c. *Propiedades de los eventos*
  - d. ***Tipos de eventos***
  - e. *Propagación y delegación*
6. *Gestión de formularios*

# 5 Gestión de eventos

## D. Tipos de eventos

De ratón

mousedown

mouseup

mouseover

mouseout

mousemove

mouseenter

mouseleave

click

dblclick

contextmenu

Todo sobre los eventos de ratón

Ojo al orden de los evento. Un click genera (en este orden) 3 eventos: *mousedown*, *mouseup*, *click*

Entiende los eventos de movimiento

Entiende los eventos de arrastre

De puntero

pointerdown

pointerup

pointerover

pointerout

pointermove

pointerenter

pointerleave

pointercancel

gotpointercapture

lostpointercapture

Un puntero hace referencia al ratón, a la pantalla táctil o a un lápiz digital.

Soporta el multitáctil

Mejor usar estos eventos que solo los de ratón

Todo sobre los eventos de puntero



1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. ***Gestión de eventos***
  - a. *Definiciones*
  - b. *Manejadores de eventos*
  - c. *Propiedades de los eventos*
  - d. ***Tipos de eventos***
  - e. *Propagación y delegación*
6. *Gestión de formularios*

# 5 Gestión de eventos

## D. Tipos de eventos

De teclado

keydown

keyup

keypress\*

[Todo sobre los eventos de teclado](#)

Ojo al orden de los evento. Pulsar una tecla genera (en este orden) 2 eventos: *keydown*, *keyup*

Ojo que si la tecla se deja pulsada, *keydown* se lanza repetidas veces

\*Deprecated

De desplazamiento

scroll

[Todo sobre el evento de scroll](#)

De formulario

change

input

focus

blur

copy

paste

cut

[Todo sobre los eventos de formulario](#)

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. ***Gestión de eventos***
  - a. *Definiciones*
  - b. *Manejadores de eventos*
  - c. *Propiedades de los eventos*
  - d. *Tipos de eventos*
  - e. ***Propagación y delegación***
6. *Gestión de formularios*

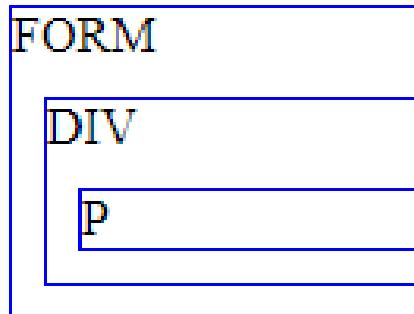


# 5 Gestión de eventos

## E. Propagación y delegación de eventos

[Artículo de referencia](#)

Los eventos se propagan. Es decir, cuando un evento ocurre en un elemento, este primero ejecuta los manejadores que tiene asignados, luego los manejadores de su padre, y así hasta otros ancestros.



Se llamará al manejador de eventos de cada ancestro hasta llegar al document

`event.target`

El elemento anidado más profundo que causó el evento es llamado elemento objetivo

`event.currentTarget`

El elemento “actual” que está manejando ahora mismo el evento (= this)

`event.stopPropagation()`

Detiene la propagación del evento hacia arriba

`event.stopImmediatePropagation()`

Detiene la propagación del evento hacia arriba, y también la ejecución de otros manejadores para este elemento

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. ***Gestión de eventos***
  - a. *Definiciones*
  - b. *Manejadores de eventos*
  - c. *Propiedades de los eventos*
  - d. *Tipos de eventos*
  - e. ***Propagación y delegación***
6. *Gestión de formularios*

# 5 Gestión de eventos

## E. Propagación y delegación de eventos

[Artículo de referencia](#)

Gracias a la propagación, podemos delegar el control de un evento a un ancestro común.

La delegación es útil cuando varios elementos van a tener un comportamiento similar.

Por ejemplo: en una tabla, que cuando se haga clic en una celda, ésta cambie de color de fondo

Podría crear un handler y asignarlo a todos los td

```
let celdas = document.querySelectorAll('td');
for(let celda of celdas){
  celda.addEventListener('click',miManejador);
}
```

Pero hay una opción mejor...



1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. ***Gestión de eventos***
  - a. *Definiciones*
  - b. *Manejadores de eventos*
  - c. *Propiedades de los eventos*
  - d. *Tipos de eventos*
  - e. ***Propagación y delegación***
6. *Gestión de formularios*

# 5 Gestión de eventos

## E. Propagación y delegación de eventos

[Artículo de referencia](#)

Mejor asignar el evento a la tabla y, en el manejador, comprobar si quien provocó el evento fue una celda

```
let tabla = document.querySelector('table');
tabla.addEventListener('click',miManejador);

function miManejador(event) {
    let celda = event.target;
    if (celda.tagName != 'TD'){
        return; // ¿no es un TD? No nos interesa
    } else{
        //...lo que toque hacer
    }
}
```

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. ***Gestión de formularios***
  - a. *Acceso al formulario*
  - b. *Campos*
  - c. *Eventos*

# 6 Gestión de formularios

## A. Acceso al formulario

[Artículo de referencia](#)

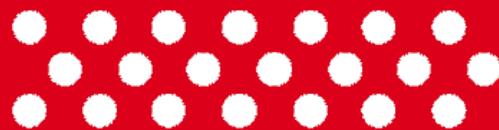
Todos los formularios de una página están disponibles en la colección `document.forms`

Puedo acceder al formulario conociendo su orden en la página

```
let miForm = document.forms[0];
```

Puedo acceder al formulario usando su nombre  
(atributo name)

```
<form name="miForm"></form>
<script>
    let miForm = document.forms.miForm;
</script>
```



- 1. Introducción
- 2. Conceptos básicos
- 3. Funciones
- 4. Gestión del DOM
- 5. Gestión de eventos
- 6. Gestión de formularios
  - a. Acceso al formulario
  - b. Campos
  - c. Eventos

# 6 Gestión de formularios

## B. Acceso a los campos del formulario

[Artículo de referencia](#)

Todos los elementos de un formulario están disponibles en la colección `form.elements`

Puedo acceder a los diferentes elementos usando su nombre (atributo name)

En el caso de que haya varios elementos con el mismo nombre (checkbox, radiobutton), `form.elements.nombre` es una colección

Da igual la estructura de etiquetas del formulario, todos los elementos están disponibles en la colección

También puedo acceder a los campos directamente como `form.nombre` (desaconsejado)

```
<form name="miForm">
  <input type="text" name="nombre">
</form>
<script>
  let miForm = document.forms.miForm
  let nombre = miForm.elements.nombre;
</script>
```

```
<form name="miForm">
  <input type="text" name="nombre">
</form>
<script>
  let miForm = document.forms.miForm
  let nombre = miForm.nombre;
</script>
```

- 1. Introducción
- 2. Conceptos básicos
- 3. Funciones
- 4. Gestión del DOM
- 5. Gestión de eventos
- 6. Gestión de formularios
  - a. Acceso al formulario
  - b. Campos
  - c. Eventos

# 6 Gestión de formularios

## B. Acceso a los campos del formulario

[Artículo de referencia](#)

Los fieldset también pertenecen a la colección. A su vez, los fieldset tienen otra colección de elementos, por lo que pueden entenderse como “sub-formularios”

```
<form name="miForm">
  <fieldset name="datosPersonales">
    <legend>info</legend>
    <input name="login" type="text">
  </fieldset>
</form>
<script>
  let miForm = document.forms.miForm
  let elFieldset = miForm.elements.datosPersonales;
  let login = elFieldset.elements.login;
  //miForm.elements.login también sería válido
</script>
```

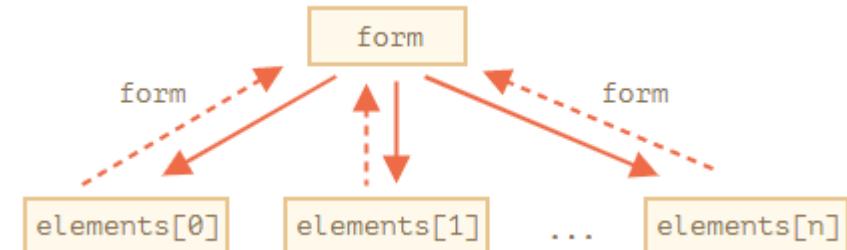
1. Introducción
2. Conceptos básicos
3. Funciones
4. Gestión del DOM
5. Gestión de eventos
6. Gestión de formularios
  - a. Acceso al formulario
  - b. Campos
  - c. Eventos

# 6 Gestión de formularios

## B. Acceso a los campos del formulario

[Artículo de referencia](#)

Al mismo tiempo, cada elemento también tiene referencia al formulario completo a través de `element.form`



`element.value`

Cada elemento de formulario tiene la propiedad `value`

Sirve tanto para leer el valor introducido por el usuario como para modificarlo

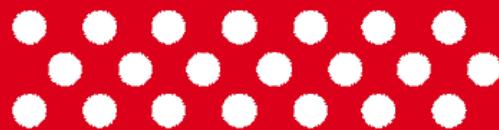
```
let nombre = miForm.elements.nombre;  
nombre.value = nombre.value.toLowerCase();
```

En el caso de los textarea también se utiliza `value` y no `innerHTML`

`element.checked`

Booleano. Similar a `value` para checkboxes y radiobuttons

- 1. *Introducción*
- 2. *Conceptos básicos*
- 3. *Funciones*
- 4. *Gestión del DOM*
- 5. *Gestión de eventos*
- 6. ***Gestión de formularios***
  - a. *Acceso al formulario*
  - b. ***Campos***
  - c. *Eventos*



# 6 Gestión de formularios

## B. Acceso a los campos del formulario

[Artículo de referencia](#)

El caso de los campos tipo select tenemos 3 propiedades importantes

select.options

Una colección con todos los elementos de tipo <option>

select.value

El valor del option que está seleccionado

select.selectedValue

El número de orden del option que está seleccionado

option.selected

Indica si el option está seleccionado.

option.text

El texto que se visualiza del option

option.index

La posición del option en el listado

new Option()

Se pueden crear nuevas opciones

`Let opcion = new Option(text,value,defaultSelected,selected);`

- 1. *Introducción*
- 2. *Conceptos básicos*
- 3. *Funciones*
- 4. *Gestión del DOM*
- 5. *Gestión de eventos*
- 6. ***Gestión de formularios***
  - a. *Acceso al formulario*
  - b. *Campos*
  - c. ***Eventos***

# 6 Gestión de formularios

## C. Eventos del formulario

Los formularios tienen una serie de eventos que ya vimos en el apartado de los eventos. Comentamos aquí algunas particularidades de ellos

focus/blur

Estos eventos no se propagan

[Todo sobre focus y blur](#)

focusin /focusout

Estos eventos son equivalentes y sí se propagan

No confundir los eventos focus/blur con los métodos `elem.focus()` y `elem.blur()`

input

Es la mejor manera para capturar cuando se escribe en un campo, ya que también capture el copia/pegar o el ingreso por reconocimiento de voz

cut/copy/paste

Los eventos de portapapeles dan acceso a los datos copiados a través de `event.clipboardData`

1. *Introducción*
2. *Conceptos básicos*
3. *Funciones*
4. *Gestión del DOM*
5. *Gestión de eventos*
6. ***Gestión de formularios***
  - a. *Acceso al formulario*
  - b. *Campos*
  - c. ***Eventos***

# 6 Gestión de formularios

## C. Eventos del formulario

[Todo sobre submit](#)

submit

El evento submit se lanza cuando se pincha en el botón de envío, pero también cuando el usuario pulsa intro en un campo

Curiosamente, cuando se pulsa intro, se genera un evento click en el botón de submit

form.submit()

Este método permite enviar el formulario desde JS sin necesidad de que se pinche en el botón

No genera evento submit, porque se supone que si lo estás haciendo por código, ya lo tienes todo controlado