

SOC

Servei d'Ocupació  
de Catalunya



Generalitat  
de Catalunya



Unió Europea  
Fons social europeu  
L'FSE inverteix en el teu futur



## MÓDULO 1. MF0951\_2 INTEGRAR COMPONENTES SOFTWARE EN PÁGINAS WEB

### UNIDAD FORMATIVA 1. UF1305 INTEGRAR COMPONENTES SOFTWARE EN PÁGINAS WEB.



- HTML DOM**
- 5.1 Introducció al DOM**
  - \_El HTML DOM (Modelo de objeto de documento)
  - \_¿Qué es el HTML DOM?
- 5.2 Métodos DOM**
  - \_La interfaz de programación DOM
- 5.3 Documento DOM**
  - \_El objeto de documento HTML DOM



5.4

## HTML DOM

### Elementos DOM

- \_Encontrar elementos HTML
- \_Encontrar elemento HTML por ID
- \_Búsqueda de elementos HTML por nombre de etiqueta
- \_Encontrar elementos HTML por nombre de clase
- \_Búsqueda de elementos HTML mediante selectores de CSS
- \_Búsqueda de elementos HTML por colecciones de objetos HTML

5.5

## DOM HTML

- \_Cambio de contenido HTML
- \_Cambiar el valor de un atributo
- \_Contenido HTML dinámico
- \_document.write()



## HTML DOM

### 5.6

#### Formularios DOM

- \_Validación de formulario JavaScript
- \_JavaScript puede validar la entrada numérica
- \_Validación automática de formularios HTML
- \_Validación de datos
- \_Validación de restricciones HTML

### 5.7

#### DOM CSS

- \_Cambio de estilo HTML
- \_Uso de eventos



## HTML DOM

5.8

### Animaciones DOM

- \_Una página web básica
- \_Crear un contenedor de animación
- \_Dale estilo a los elementos
- \_Código de animación
- \_Crear la animación completa usando JavaScript

5.9

### Eventos DOM

- \_Reaccionar a los eventos
- \_Asignar eventos utilizando el HTML DOM
- \_Los eventos onload y onunload
- \_El evento onchange
- \_Los eventos onmouseover y onmouseout
- \_Los eventos onmousedown, onmouseup y onclick
- \_Más ejemplos



## 5.10

### Oyente de eventos DOM

- \_El método `addEventListener()`
- \_Agregar un controlador de eventos a un elemento
- \_Agregue muchos controladores de eventos al mismo elemento
- \_Agregue un controlador de eventos al objeto de la ventana
- \_Pasando Parámetros
- \_bubbling de eventos o captura de eventos?
- \_El método `removeEventListener()`



## 5.11

### Navegación DOM

\_Nodos DOM

\_Relaciones de nodos

\_Navegación entre nodos

\_Nodos secundarios y valores de nodos

\_InnerHTML

\_DOM Root Nodes

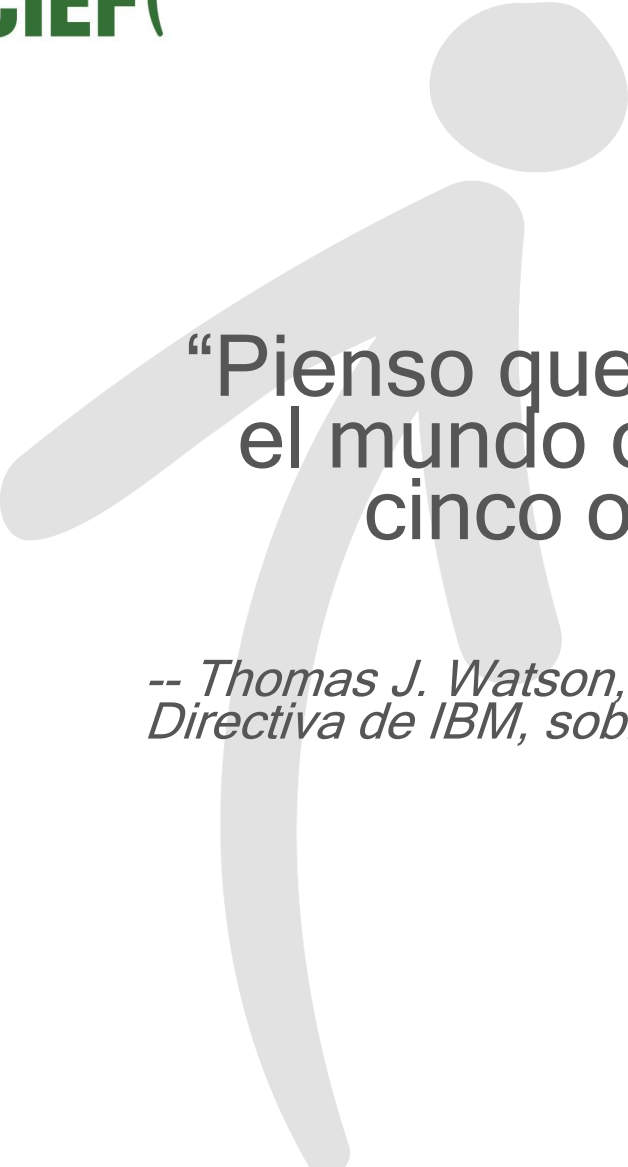
\_The nodeName Property

\_The nodeValue Property

\_The nodeType Property





A large, light grey, stylized figure of a person with arms raised, similar to the CIEF logo icon, serves as a background for the quote.

“Pienso que hay mercado en el mundo como para unos cinco ordenadores”

*-- Thomas J. Watson, Presidente de la Junta Directiva de IBM, sobre 1948*





# Javascript. IV



## Introducción al DOM. (**M**odelo de **O**bjeto de **D**ocumento)

## 5.1. Introducció. El HTML DOM (Modelo de Objeto de Documento)

Para que el navegador pueda representar la página, debe construir los árboles del DOM y el CSSOM. En consecuencia, debemos asegurarnos de proporcionar lenguaje de marcado HTML y CSS al navegador lo más rápido posible.

El modelo HTML DOM se construye como un árbol de objetos :

Comencemos con el caso más sencillo: una página HTML estándar, con un poco de texto y una sola imagen. ¿Cómo el navegador procesa esta página?

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div></div>
  </body>
</html>
```

## 5.1. Introducció. El HTML DOM (Modelo de Objeto de Documento)

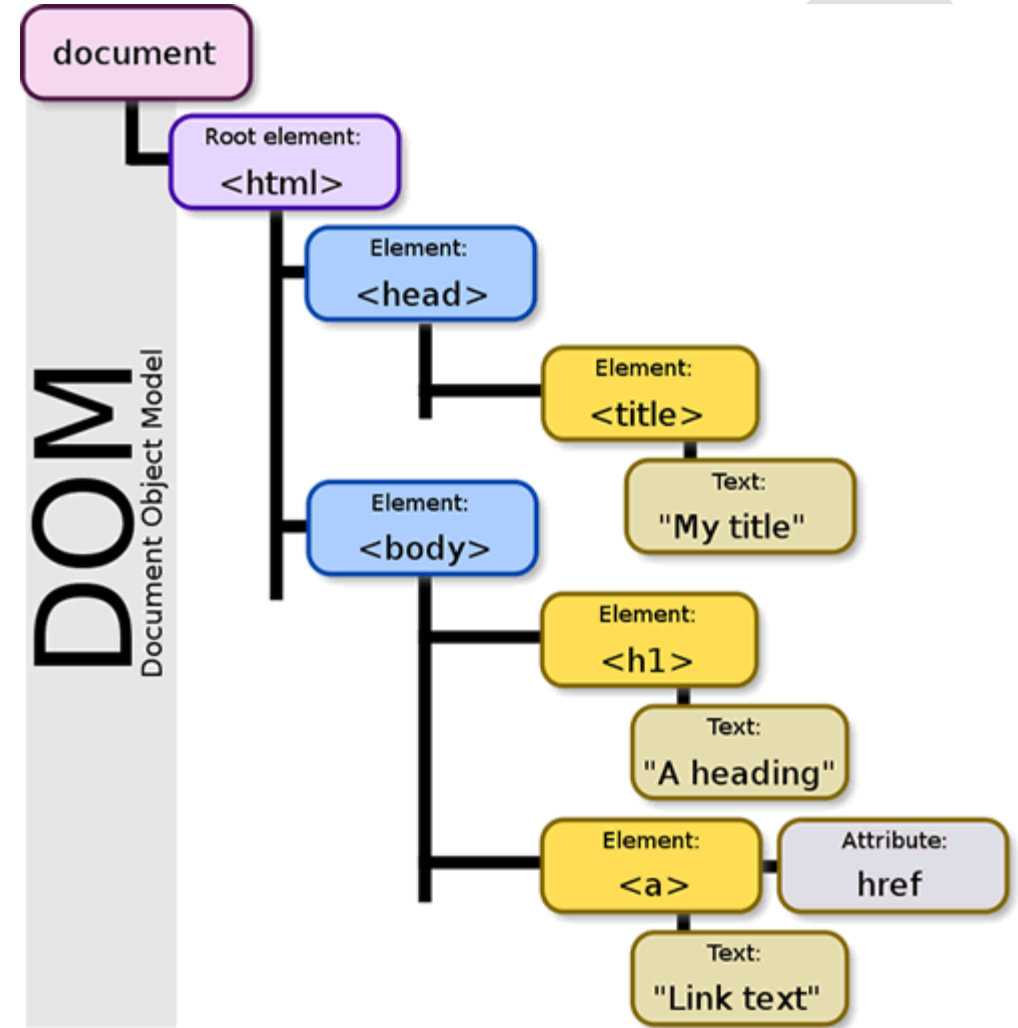
Cuando se carga una página web, el navegador crea un **modelo de objeto** de documento de la página.

El modelo HTML DOM se construye como un árbol de **objetos** :

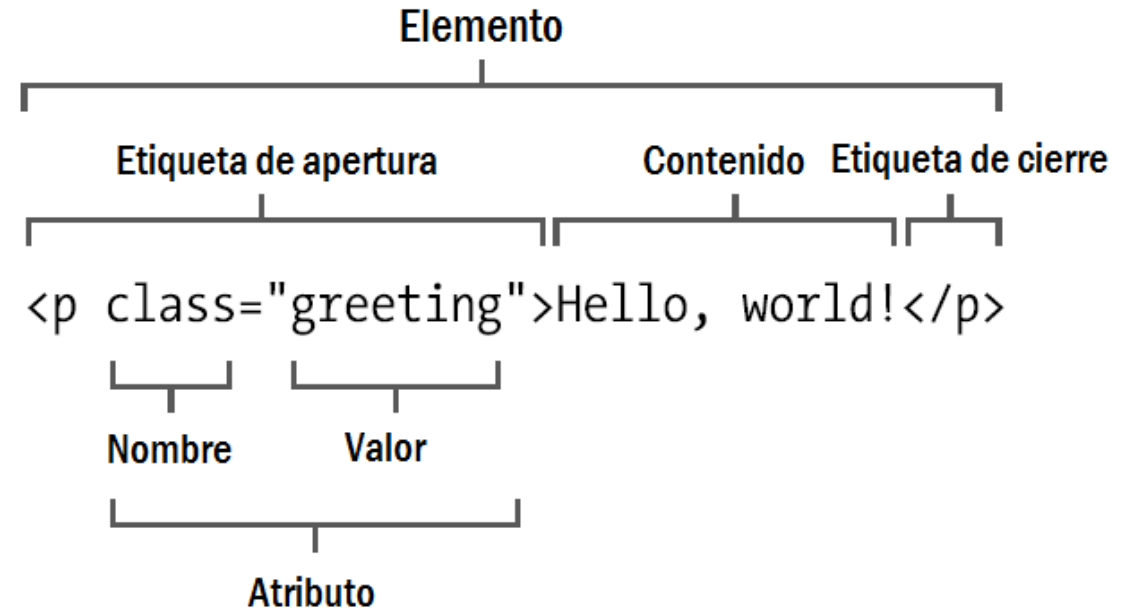
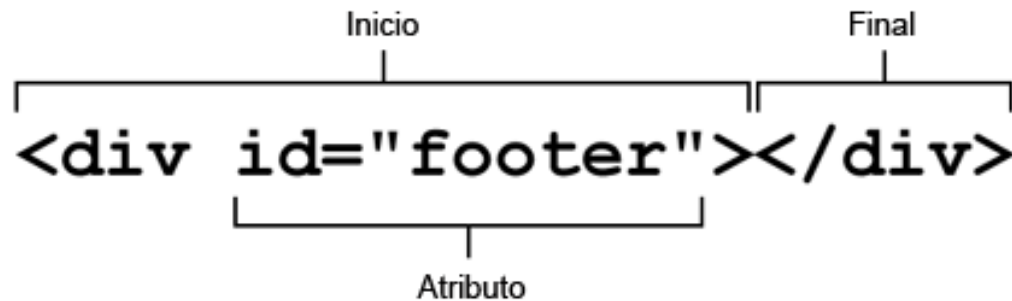
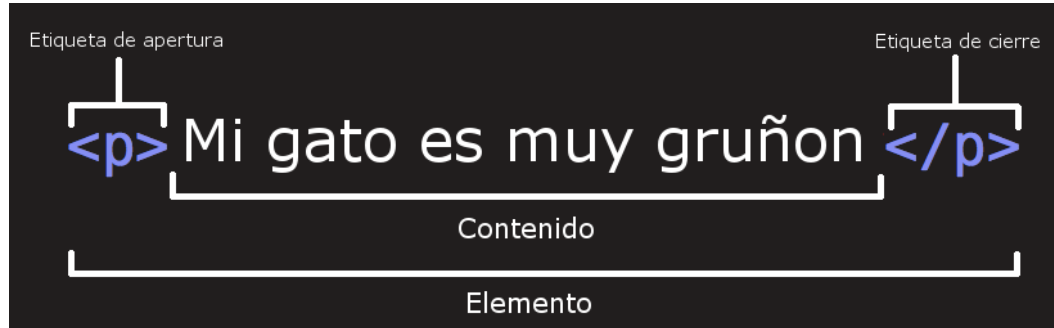
Con el modelo de objetos, JavaScript obtiene todo el poder que necesita para crear HTML dinámico:

- JavaScript puede **cambiar** todos los **elementos HTML** en la página
- JavaScript puede cambiar todos los **atributos HTML** en la página
- JavaScript puede cambiar todos los **estilos CSS** en la página
- JavaScript puede **eliminar** elementos y atributos HTML existentes
- JavaScript puede **agregar** nuevos elementos y atributos HTML
- JavaScript puede **reaccionar** a todos los eventos HTML existentes en la página
- JavaScript puede **crear nuevos eventos** HTML en la página

¿Sabrías decir un ejemplo de cada?



## 5.1. Introducció. El HTML DOM (Modelo de Objeto de Documento)



## 5.1. Introducció. ¿Qué es el HTML DOM?



[www.w3.org/](http://www.w3.org/)

El DOM es un estándar W3C (World Wide Web Consortium).

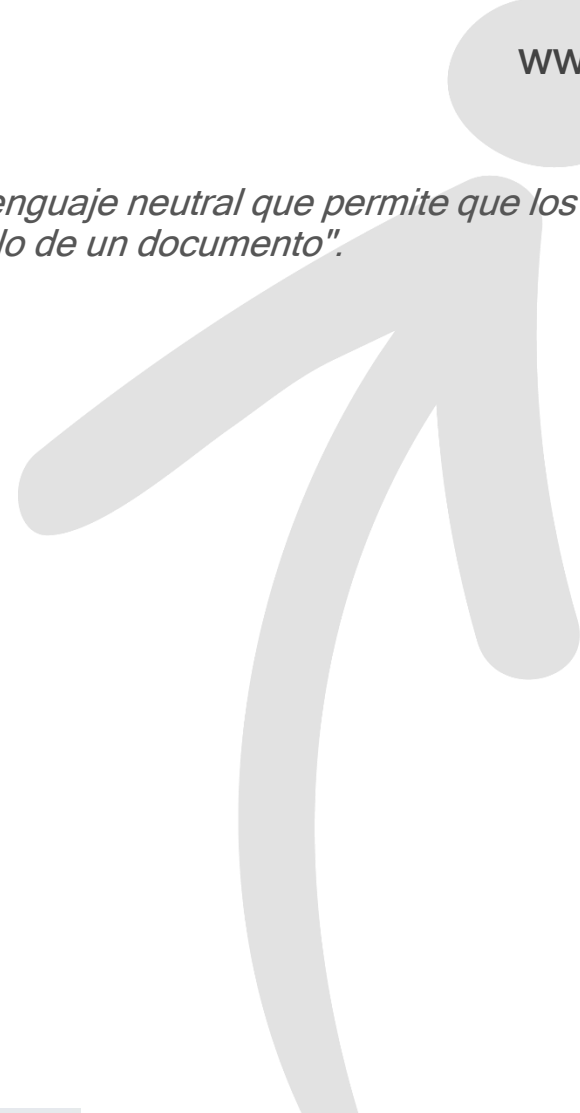
El DOM define un estándar para acceder a los documentos:

*"El Modelo de Objetos de Documento (DOM) del W3C es una plataforma y una interfaz de lenguaje neutral que permite que los programas y scripts accedan y actualicen dinámicamente el contenido, la estructura y el estilo de un documento".*

El estándar W3C DOM se divide en 3 partes diferentes:

- Core DOM: modelo estándar para todos los tipos de documentos
- XML DOM - modelo estándar para documentos XML
- HTML DOM - modelo estándar para documentos HTML

:



## Métodos DOM. (Modelo de Objeto de Documento)





## 5.2. Mètodes DOM. La interface de programació DOM



Los métodos HTML DOM son **acciones** que puede realizar (en elementos HTML).

Las propiedades HTML DOM son **valores** (de elementos HTML) que puede **establecer o cambiar**.

### La interfaz de programación DOM

Se puede acceder al HTML DOM con JavaScript (y con otros lenguajes de programación).

En el DOM, todos los elementos HTML se definen como **objetos**.

La interfaz de programación son las propiedades y métodos de cada objeto.

- Una **propiedad** es un valor que puede obtener o establecer (como cambiar el contenido de un elemento HTML).
- Un **método** es una acción que puede realizar (como agregar o eliminar un elemento HTML)..

El siguiente ejemplo cambia el contenido (el innerHTML) del <p>elemento con id="demo":

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Page</h2>

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

**My First Page**

Hello World!

## 5.2. Mètodes DOM. La interface de programació DOM



La forma **más común** de acceder a un elemento HTML es usar el **id del elemento**.

En el ejemplo anterior, el **getElementById** método utilizado **id="demo"** para encontrar el elemento.

La forma más sencilla de obtener el contenido de un elemento es mediante la **innerHTML** propiedad.

La **innerHTML** propiedad es útil para obtener o reemplazar el contenido de los elementos HTML.

La **innerHTML** propiedad se puede usar para obtener o cambiar cualquier elemento HTML, incluidos `<html>` y `<body>`.

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Page</h2>

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

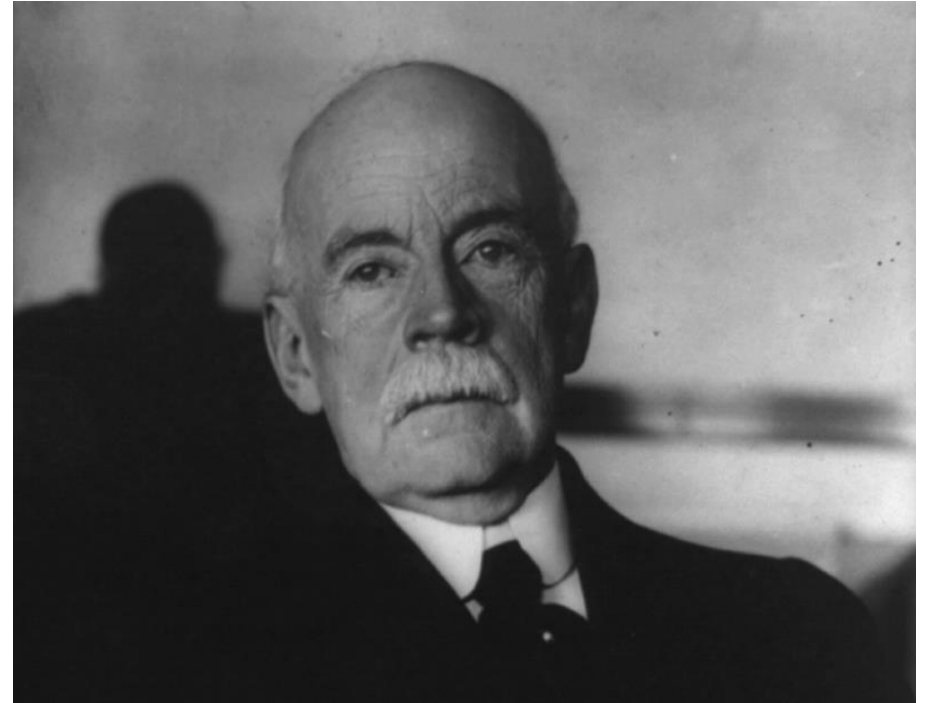
## 5.2. Mètodes DOM. La interface de programació DOM



camino	descripció
getElementById ()	Devuelve el elemento con el ID especificado.
getElementsByTagName ()	Devuelve una lista de nodos (matriz de conjuntos / nodo) que contiene todos los elementos con el nombre de la etiqueta especificada.
getElementsByTagName ()	Devuelve una lista de todos los elementos con el nodo especificado contiene el nombre de la clase.
appendChild ()	Para añadir un nuevo nodo secundario al nodo especificado.
removeChild ()	Elimina el nodo secundario.
replaceChild ()	Vuelva a colocar los nodos secundarios.
insertBefore ()	Insertar un nuevo nodo secundario frente al nodo secundario especificado.
createAttribute ()	Crear nodo de atributo.
createElement ()	Crear un nodo elemento.
createTextNode ()	Crear un nodo de texto.
getAttribute ()	Devuelve los valores de los atributos especificados.
setAttribute ()	Para establecer o modificar el valor especificado de la propiedad especificada.

“Todo lo que puede ser inventado ha sido ya inventado”

-- *Charles H. Duell, Comisario de oficina de Patentes en EEUU, en 1899*



## Documento DOM. (**M**odelo de **O**bjeto de **D**ocumento)

## 5.3. Documento DOM. \_El objeto de documento HTML DOM

- El objeto del documento representa su página web.
- Si desea acceder a cualquier elemento en una página HTML, siempre comienza accediendo al objeto del documento.
- A continuación se muestran algunos ejemplos de cómo puede utilizar el objeto de documento para acceder y manipular HTML.

Method	Description
<code>document.getElementById(<i>id</i>)</code>	Find an element by element id
<code>document.getElementsByTagName(<i>name</i>)</code>	Find elements by tag name
<code>document.getElementsByClassName(<i>name</i>)</code>	Find elements by class name

## Elementos DOM. (**M**odelo de **O**bjeto de **D**ocumento)



## 5.4. Elementos DOM. \_Encontrar elemento HTML

Encontrar elementos HTML por id

Encontrar elementos HTML por nombre de etiqueta

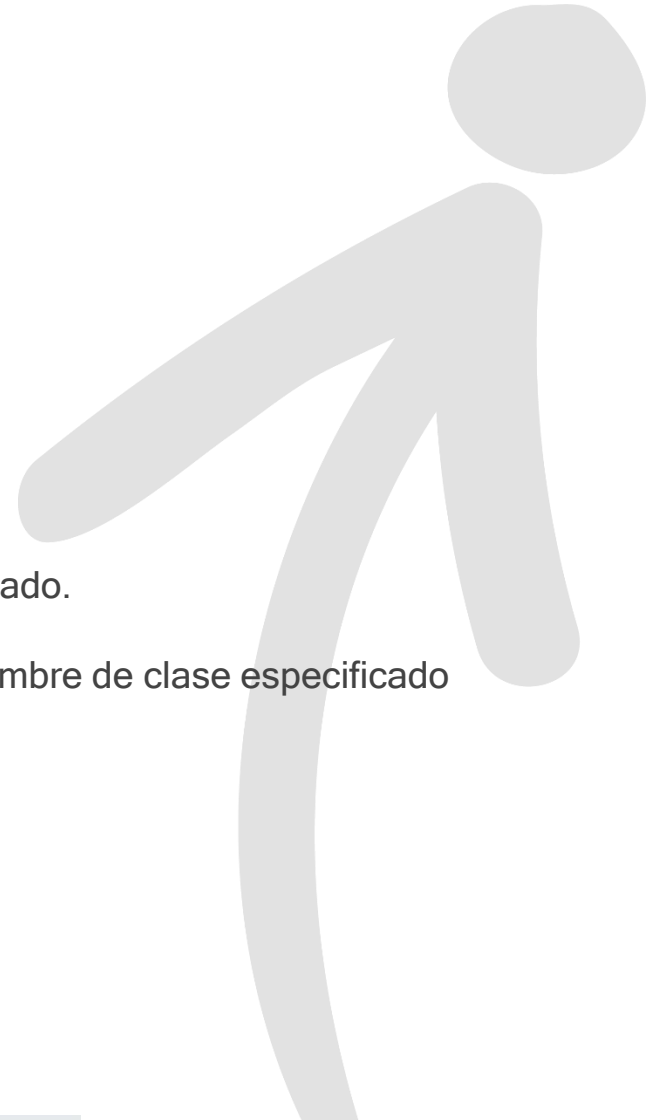
Encontrar elementos HTML por nombre de clase

### Método:

`getElementById()` : Devuelve el elemento con el ID especificado

`getElementsByTagName()` : Devuelve una colección de objetos con el nombre especificado.

`getElementsByName()` : Devuelve una lista de nodos con todos los elementos con el nombre de clase especificado





document.**getElementById()**

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p id="intro">Finding HTML Elements by Id</p>
<p>This example demonstrates the <b>getElementsById</b> method.</p>

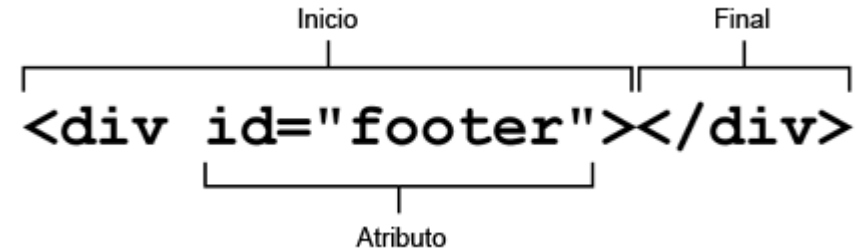
<p id="demo"></p>

<script>
const element = document.getElementById("intro");

document.getElementById("demo").innerHTML =
"The text from the intro paragraph is: " + element.innerHTML;

</script>

</body>
</html>
```



## JavaScript HTML DOM

### Finding HTML Elements by Id

This example demonstrates the `getElementsById` method.

The text from the intro paragraph is: Finding HTML Elements by Id

Index\_502



document.**getElementsByTagName()**

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Tag Name.</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>

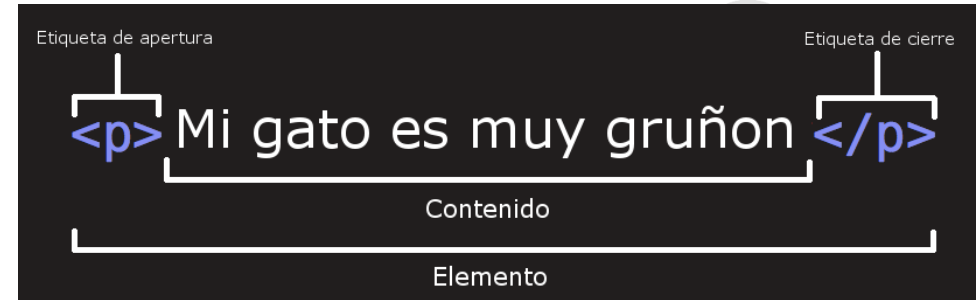
<p id="demo"></p>

<script>
const element = document.getElementsByTagName("p");

document.getElementById("demo").innerHTML = 'The text in first paragraph (index 0)
is: ' + element[0].innerHTML;

</script>

</body>
</html>
```



## JavaScript HTML DOM

Finding HTML Elements by Tag Name.

This example demonstrates the `getElementsByTagName` method.

The text in first paragraph (index 0) is: `Finding HTML Elements by Tag Name.`



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<div id="main">
<p>Finding HTML Elements by Tag Name</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
<h1>Titulo</h1>
</div>

<p id="demo0" style="color:Tomato;"></p>
<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");

document.getElementById("demo0").innerHTML = 'inside "main" is: ' + x.innerHTML;

document.getElementById("demo1").innerHTML = 'The first paragraph (index 0) inside "main" is: ' + y[0].innerHTML;

document.getElementById("demo2").innerHTML = 'The second paragraph (index 1) inside "main" is: ' + y[1].innerHTML;

document.getElementById("demo3").innerHTML = 'The second paragraph (index 2) inside "main" is: ' + y[2].innerHTML;
/*ERROR no existe tercera etiqueta <p>*/

</script>

</body>
</html>
```

Este ejemplo encuentra el elemento con `id="main"` y luego encuentra todos los `<p>` elementos dentro `"main"`:

### JavaScript HTML DOM

Finding HTML Elements by Tag Name

This example demonstrates the `getElementsByTagName` method.

### Titulo

inside "main" is:

Finding HTML Elements by Tag Name

This example demonstrates the `getElementsByTagName` method.

### Titulo

The first paragraph (index 0) inside "main" is: Finding HTML Elements by Tag Name

The second paragraph (index 1) inside "main" is: This example demonstrates the `getElementsByTagName` method.



## document.getElementsByClassName()

```
<!DOCTYPE html>
<html>
<body>

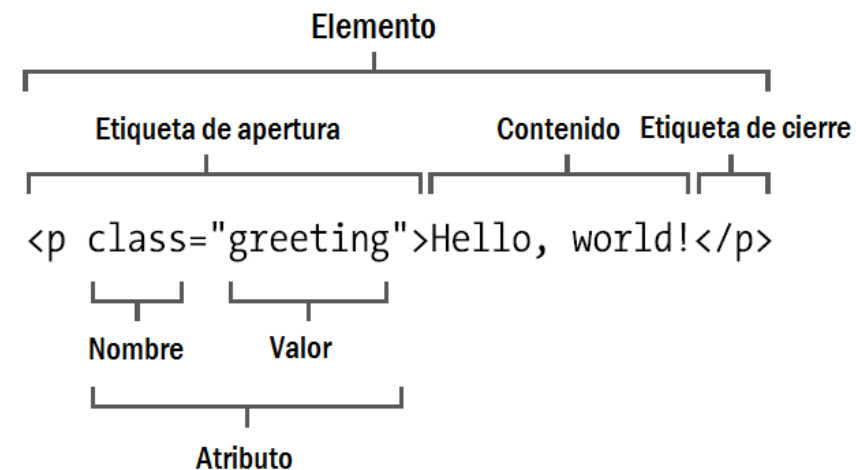
<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Class Name.</p>
<p class="intro">Hello World!</p>
<p class="intro">This example demonstrates the <b>getElementsByClassName</b>method.</p>

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

<script>
const x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>

</body>
</html>
```



### JavaScript HTML DOM

Finding HTML Elements by Class Name.

Hello World!

This example demonstrates the **getElementsByClassName** method.

The first paragraph (index 0) with class="intro" is: Hello World!



document.querySelectorAll()

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Query Selector</p>
<p class="intro">Hello World!.</p>
<p class="intro">This example demonstrates the <b>querySelectorAll</b> method.</p>

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

<script>
const x = document.querySelectorAll("p.intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>

</body>
</html>
```

## JavaScript HTML DOM

Finding HTML Elements by Query Selector

Hello World!.

This example demonstrates the **querySelectorAll** method.

The first paragraph (index 0) with class="intro" is: Hello World!.

## 5.4. Elementos DOM. \_Encontrar elemento HTML por colección objetos

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<p>Encontrar elementos usando <b>document.forms</b>.</p>

<form id="frm1" action="/action_page.php">
  First name: <input type="text" name="fname" value="Donald"><br>
  Last name: <input type="text" name="lname" value="Duck"><br><br>
  <input type="submit" value="Submit">
</form>

<p>Estos son los valores de cada elemento del formulario:</p>

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

<script>
const x = document.forms["frm1"];
let text = "";

for (let i = 0; i < x.length ;i++) {
  text +=x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

document.forms

### JavaScript HTML DOM

Encontrar elementos usando **document.forms**.

First name:   
Last name:

Estos son los valores de cada elemento del formulario:

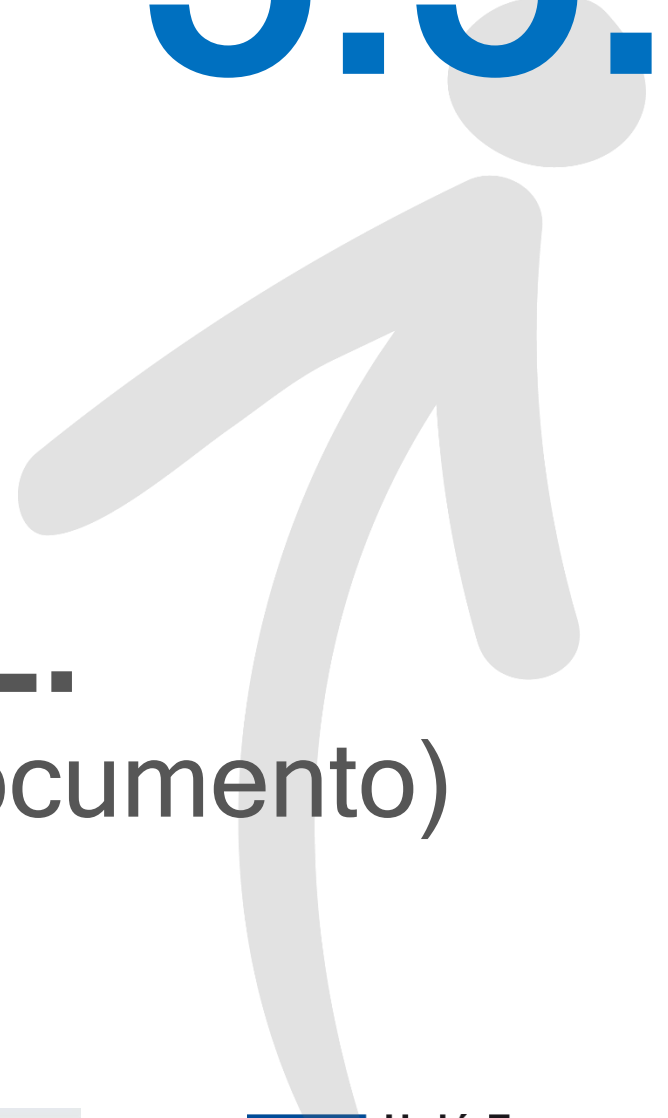
Donald  
Duck  
Submit





## DOM HTML.

(**M**odelo de **O**bjeto de **D**ocumento)





El HTML DOM permite que JavaScript cambie el contenido de los elementos HTML.

La forma más sencilla de modificar el contenido de un elemento HTML es mediante la `innerHTML` propiedad. Para cambiar el contenido de un elemento HTML, utilice esta sintaxis:

```
document.getElementById(id).innerHTML = new HTML
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript can Change HTML</h2>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

<p>The paragraph above was changed by a script.</p>

</body>
</html>
```

## JavaScript can Change HTML

New text!

The paragraph above was changed by a script.



```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Old Heading</h1>

<script>
const element = document.getElementById("id01");
element.innerHTML = "New Heading";
</script>

<p>JavaScript changed "Old Heading" to "New Heading".</p>

</body>
</html>
```

### New Heading

JavaScript changed "Old Heading" to "New Heading".

Ejemplo explicado:

- El documento HTML anterior contiene un `<h1>` elemento con `id="id01"`
- Usamos el HTML DOM para obtener el elemento con `id="id01"`
- Un JavaScript cambia el contenido ( `innerHTML` ) de ese elemento a "Nuevo encabezado"

## 5.5. DOM HTML. \_ Cambiar el valor de un atributo



Para cambiar el valor de un atributo HTML, utilice esta sintaxis:

```
document.getElementById(id).attribute = new value
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>


<script>
document.getElementById("image").src = "landscape.jpg";
</script>

<p>The original image was smiley.gif, but the script changed it to landscape.jpg</p>

</body>
</html>
```

### JavaScript HTML DOM



The original image was smiley.gif, but the script changed it to landscape.jpg



JavaScript puede crear contenido HTML dinámico:

```
<!DOCTYPE html>
<html>
<body>

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

<script>
document.getElementById("demo").innerHTML = "Date : " + Date();
</script>

</body>
</html>
```

Date : Tue Mar 08 2022 15:00:23 GMT+0100 (hora estándar de Europa central)

## 5.5. DOM HTML. \_document.write()



En JavaScript, **document.write()** se puede usar para escribir directamente en el flujo de salida HTML:

```
<!DOCTYPE html>
<html>
<body>

<p>Bla, bla, bla</p>

<script>
document.write(Date());
</script>

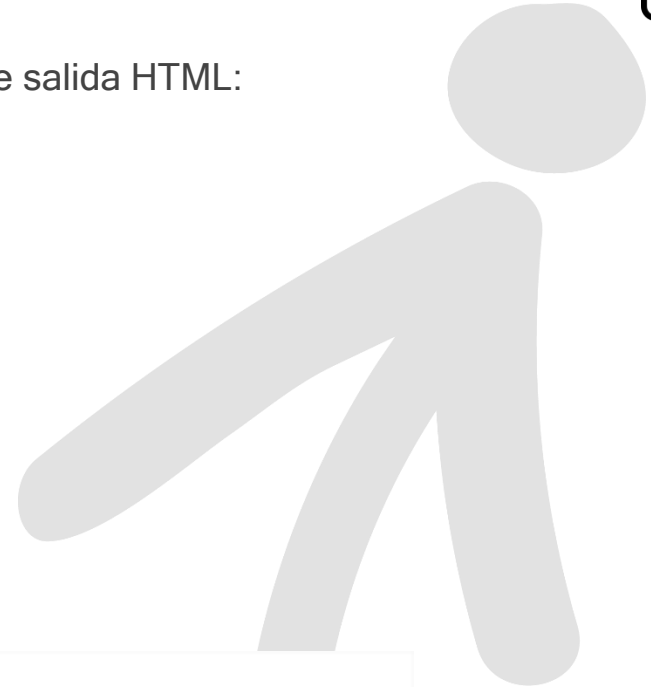
<p>Bla, bla, bla</p>

</body>
</html>
```

Bla, bla, bla

Tue Mar 08 2022 15:01:48 GMT+0100 (hora estándar de Europa central)

Bla, bla, bla



"Podría parecer que hemos llegado a los límites alcanzables por la tecnología informática, aunque uno debe ser prudente con estas afirmaciones, pues tienden a sonar bastante tontas en cinco años"

-- John Von Neumann, sobre 1949





## Formularios DOM. (**M**odelo de **O**bjeto de **D**ocumento)



La validación de formularios HTML se puede realizar mediante JavaScript.

Si un campo de formulario (fname) está vacío, esta función alerta con un mensaje y devuelve falso para evitar que se envíe el formulario:

### JavaScript Validation

Name:

#### ¿Qué es == en JavaScript?

Doble igual (==) es un operador de comparación, que transforma los operandos que tienen el mismo tipo antes de la comparación.

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>

<h2>JavaScript Validation</h2>

<form name="myForm" action="/action_page.php" onsubmit="return validateForm()"
method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

## 5.6. Formularios DOM \_ JavaScript puede validar la entrada numérica

JavaScript se usa a menudo para validar la entrada numérica;

**JavaScript Validation**

Please input a number between 1 and 10:

Input OK

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Validation</h2>

<p>Please input a number between 1 and 10:</p>

<input id="numb">

<button type="button" onclick="myFunction()">Submit</button>

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

<script>
function myFunction() {
  // Get the value of the input field with id="numb"
  let x = document.getElementById("numb").value;
  // If x is Not a Number or less than one or greater than 10
  let text;
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input not valid";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```



## 5.6. Formularios DOM \_ Validación automática de formularios HTML



El navegador puede realizar automáticamente la validación del formulario HTML:  
Si un campo de formulario (fname) está vacío, el **required** atributo impide que se envíe este formulario:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Validation</h2>

<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>

<p>If you click submit, without filling out the text field,
your browser will display an error message.</p>

</body>
</html>
```



## 5.6. Formularios DOM \_ Validación de datos

La validación de datos es el proceso de garantizar que la entrada del usuario sea limpia, correcta y útil.

Las tareas típicas de validación son:

- ¿Ha rellenado el usuario todos los campos obligatorios?
- ¿El usuario ha ingresado una fecha válida?
- ¿Ha ingresado el usuario texto en un campo numérico?

Muy a menudo, el propósito de la validación de datos es garantizar la entrada correcta del usuario.

La validación se puede definir mediante muchos métodos diferentes y se puede implementar de muchas maneras diferentes.

**La validación del lado del servidor** la realiza un servidor web, después de que la entrada se haya enviado al servidor.

**La validación del lado del cliente** la realiza un navegador web, antes de enviar la entrada a un servidor web.

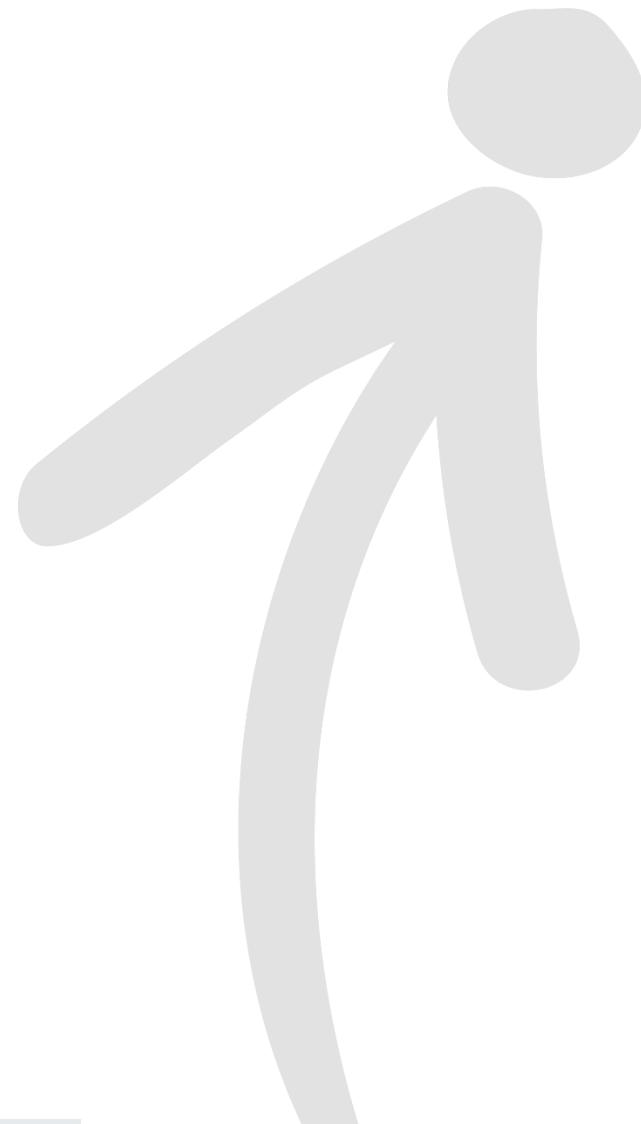
¿Cuáles son?

### Atributos de entrada HTML de validación de restricciones

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element



## 5.6. Formularios DOM \_Validación de restricciones HTML



# DOM CSS.

(**M**odelo de **O**bjeto de **D**ocumento)



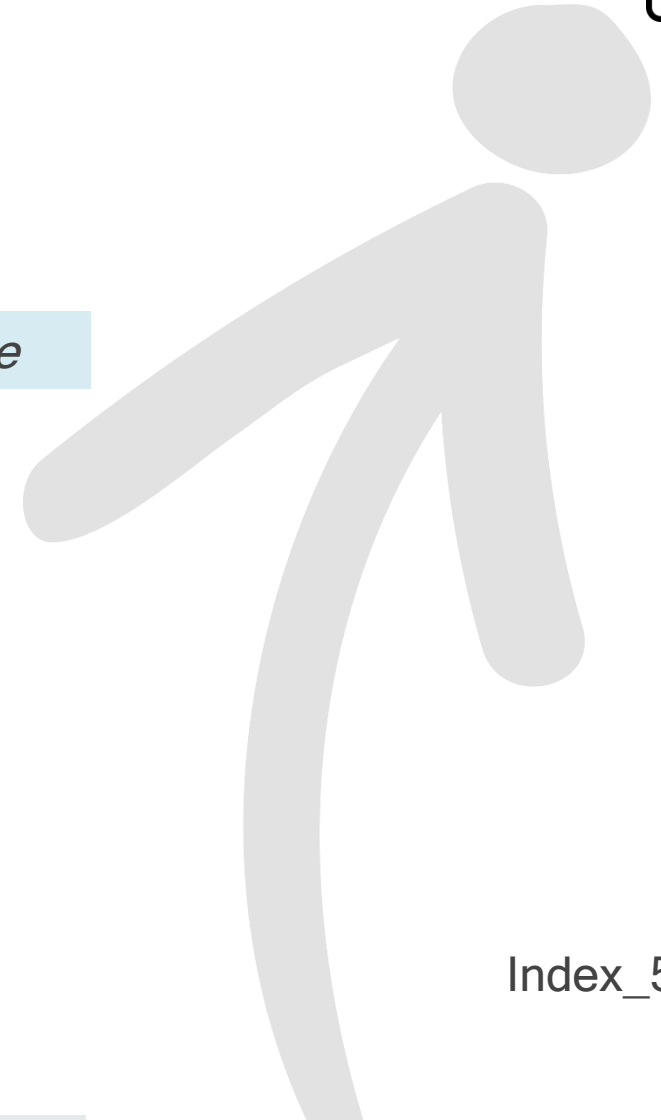
## 5.7. DOM CSS \_Cambio de estilo HTML



El HTML DOM permite que JavaScript cambie el estilo de los elementos HTML.

Para cambiar el estilo de un elemento HTML, use esta sintaxis:

```
document.getElementById(id).style.property = new style
```



Index\_503





El HTML DOM le permite ejecutar código cuando ocurre un evento.

Los eventos son generados por el navegador cuando "le suceden cosas" a los elementos HTML:

- Se hace clic en un elemento
- La página ha cargado
- Los campos de entrada se cambian

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

**My Heading 1**

Click Me!

Index\_504

## 5.7. DOM CSS \_Cambio de estilo HTML



El HTML DOM permite que JavaScript cambie el estilo de los elementos HTML.

### JavaScript HTML DOM

Changing the HTML style:

Hello World!

Hello World!

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<p>Changing the HTML style:</p>

<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
</script>

</body>
</html>
```

"Windows NT podr  direccionar 2GB de RAM,  
que es m s de lo que  
cualquier aplicaci n va a  
necesitar jam s"

*-- Microsoft, durante el desarrollo de  
Windows NT, en 1992*



# Animaciones DOM.

(**M**odelo de **O**bjeto de **D**ocumento)

## 5.8. Animaciones DOM \_Una página web básica

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript Animation</h1>

<div id="animate">My animation will go here</div>

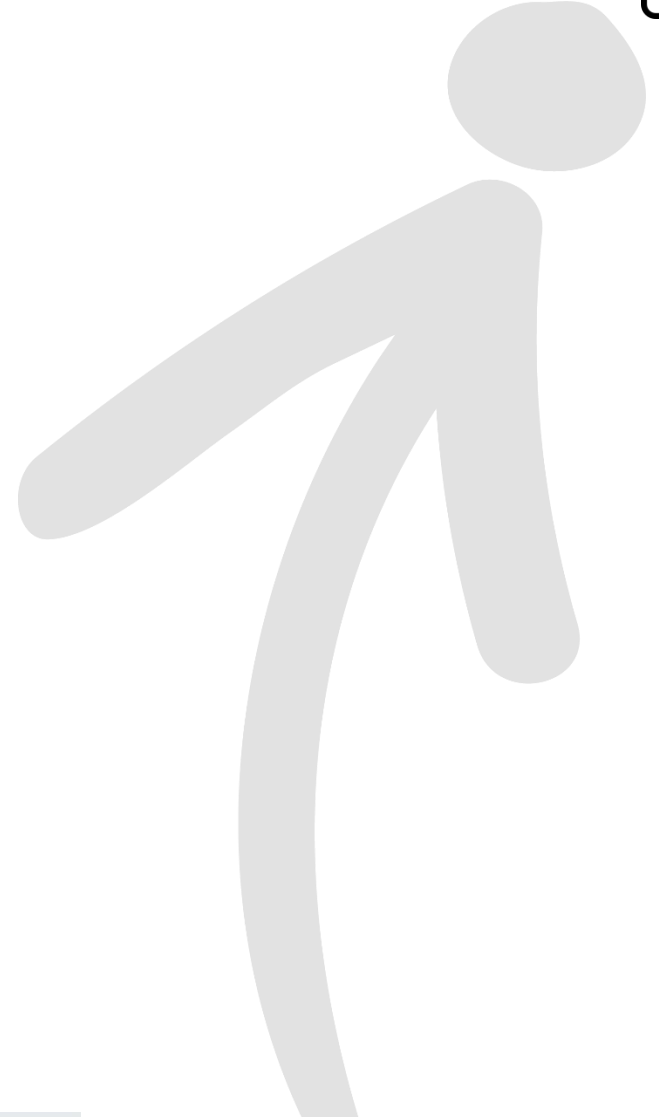
</body>
</html>
```

## 5.8. Animaciones DOM \_Crear un contenedor de animación



```
<div id="container">  
  <div id="animate">My animation will go here</div>  
</div>
```

```
<!DOCTYPE html>  
<html>  
  <body>  
  
    <h1>My First JavaScript Animation</h1>  
  
    <div id="animation">My animation will go here</div>  
  
  </body>  
</html>
```



## 5.8. Animaciones DOM \_Dale estilo a los elementos

El elemento contenedor debe realizarse con el style = "position: relative".

El elemento de animación debe realizarse con el style = "position: absolute".

```
#container {  
  width: 400px;  
  height: 400px;  
  position: relative;  
  background: yellow;  
}  
#animate {  
  width: 50px;  
  height: 50px;  
  position: absolute;  
  background: red;  
}
```



## 5.8. Animaciones DOM \_Dale estilo a los elementos



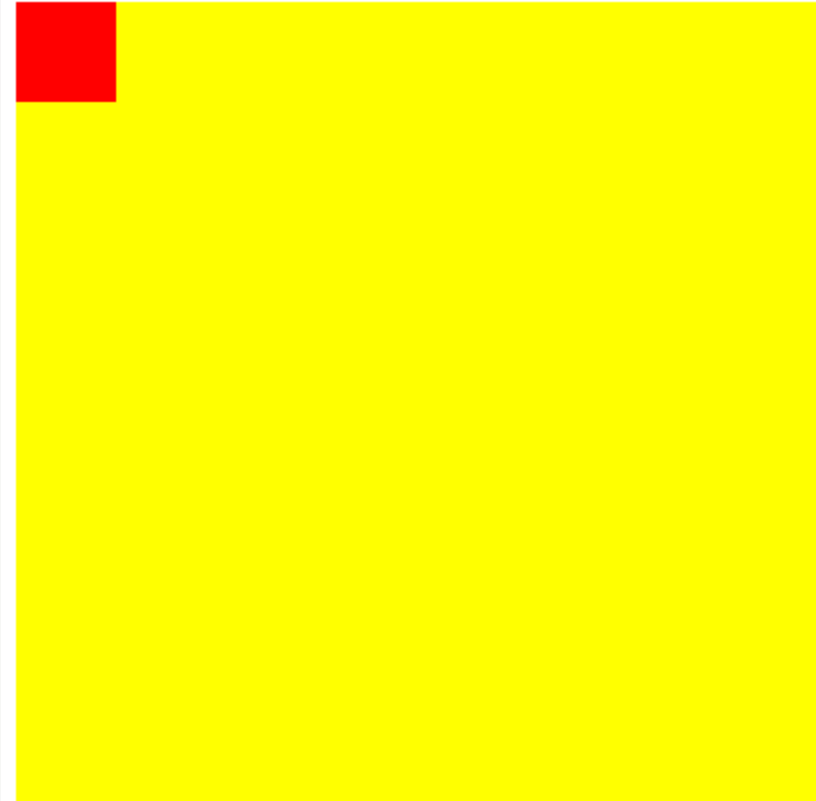
```
<!Doctype html>
<html>
<style>
#container {
  width: 400px;
  height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background: red;
}
</style>
<body>

<h2>My First JavaScript Animation</h2>

<div id="container">
<div id="animate"></div>
</div>

</body>
</html>
```

### My First JavaScript Animation



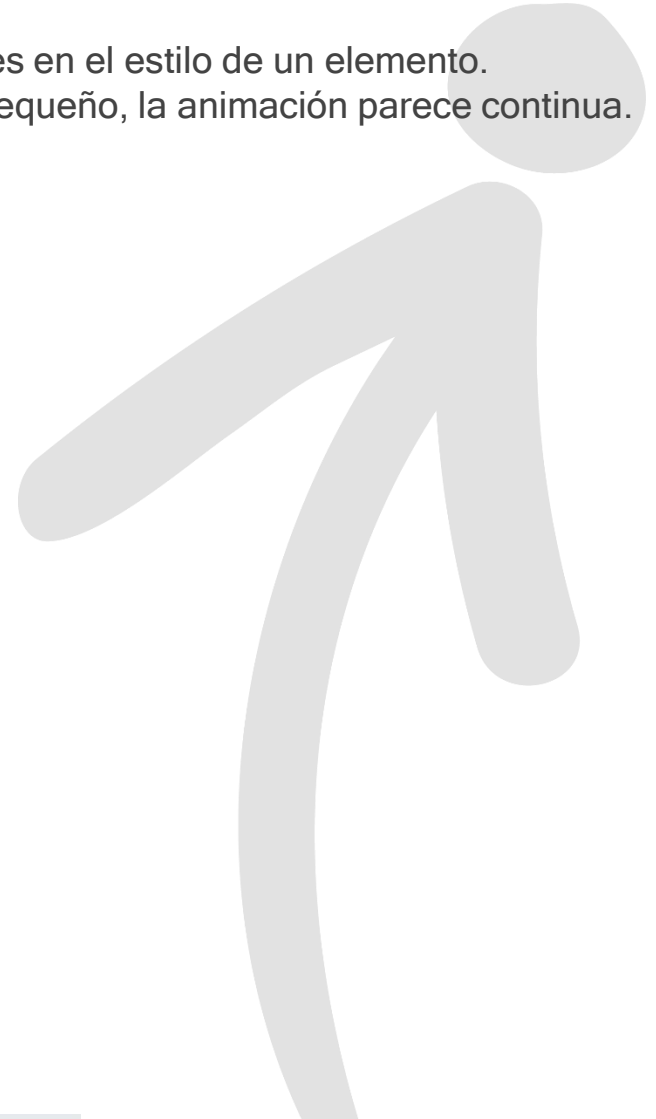


## 5.8. Animaciones DOM \_ Código de animación

Las animaciones de JavaScript se realizan mediante la programación de cambios graduales en el estilo de un elemento. Los cambios son llamados por un temporizador. Cuando el intervalo del temporizador es pequeño, la animación parece continua. El código básico es:

```
var id = setInterval(frame, 5);

function frame() {
  if (/* test for finished */) {
    clearInterval(id);
  } else {
    /* code to change the element style */
  }
}
```



## 5.8. Animaciones DOM \_ Crear la animación completa usando JS



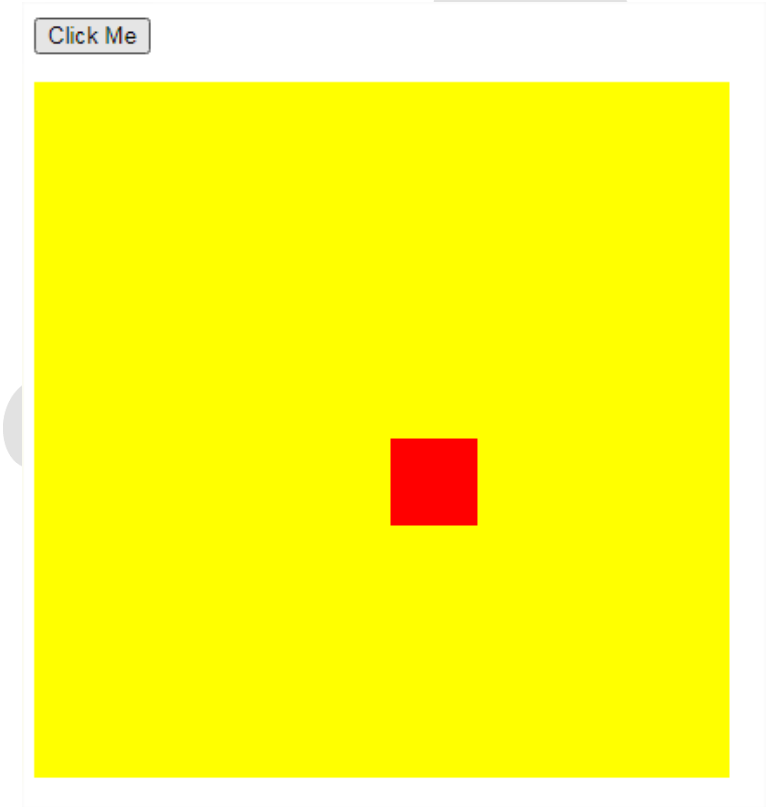
```
<!DOCTYPE html>
<html>
<style>
#container {
  width: 400px;
  height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background-color: red;
}
</style>
```

```
<body>

<p><button onclick="myMove()">Click Me</button></p>

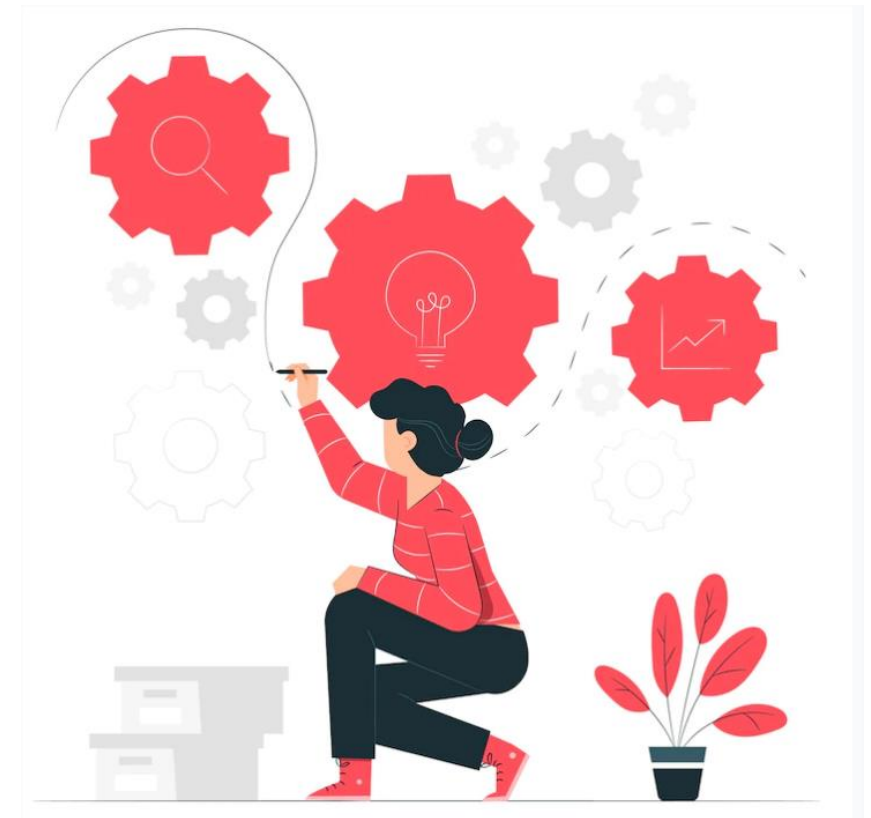
<div id ="container">
  <div id ="animate"></div>
</div>

<script>
function myMove() {
  let id = null;
  const elem = document.getElementById("animate");
  let pos = 0;
  clearInterval(id);
  id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + "px";
      elem.style.left = pos + "px";
    }
  }
}
</script>
</body>
</html>
```



"Pero, ¿para qué puede valer eso?"

*-- Ingeniero en la división de sistemas informáticos avanzados de IBM, hablando sobre los microchips, en 1968*



## Eventos DOM. (Modelo de Objeto de Documento)

## 5.9 Eventos DOM\_Reaccionar a los eventos



Un JavaScript se puede ejecutar cuando ocurre un evento, como cuando un usuario hace clic en un elemento HTML. Para ejecutar código cuando un usuario hace clic en un elemento, agregue código JavaScript a un atributo de evento HTML:

Ejemplos de eventos HTML:

- Cuando un usuario hace clic con el mouse
- Cuando una página web ha cargado
- Cuando se ha cargado una imagen
- Cuando el mouse se mueve sobre un elemento
- Cuando se cambia un campo de entrada
- Cuando se envía un formulario HTML
- Cuando un usuario pulsa una tecla

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<h2 onclick="this.innerHTML='Ooops!'">Click on this text!</h2>

</body>
</html>
```

**JavaScript HTML Events**

**Click on this text!**

**JavaScript HTML Events**

**Ooops!**

Index\_506

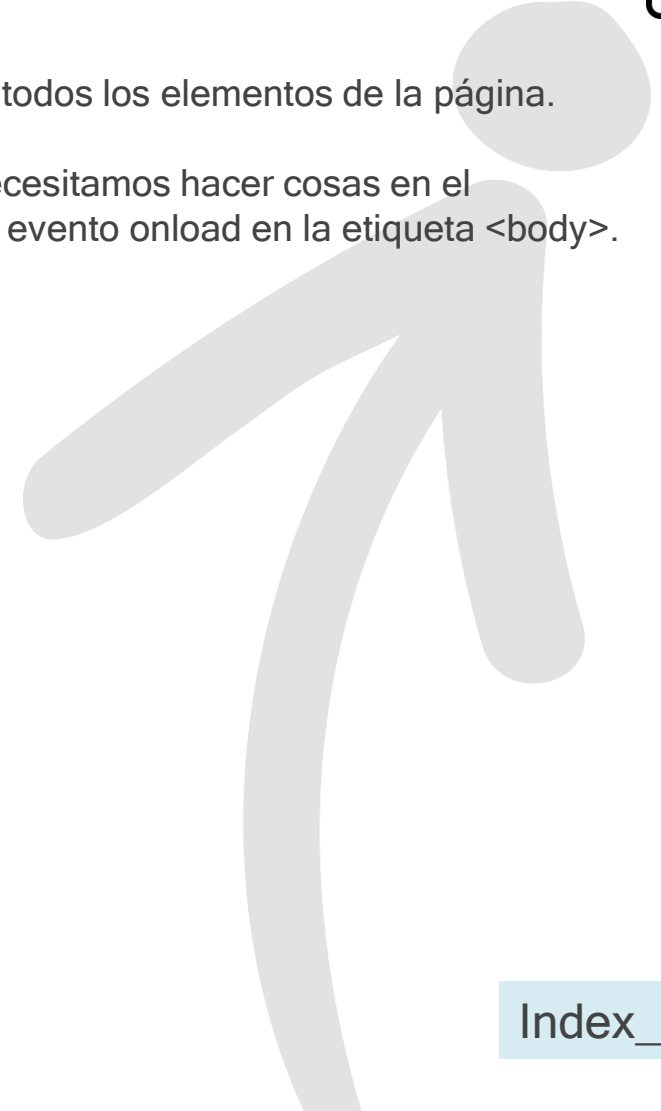


Con el evento onload podemos ejecutar acciones justo cuando se han terminado de cargar todos los elementos de la página.

El **evento onload de Javascript** se activa cuando se termina de cargar la página. Cuando necesitamos hacer cosas en el momento en el que la página haya terminado de cargar, podemos asociar un manejador de evento onload en la etiqueta <body>.

```
<body onload="pedirVoto()">
```

```
<body onload="init();">
```



Index\_506



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<h2 onclick="changeText(this)">Click on this text!</h2>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

**JavaScript HTML Events**

**Click on this text!**

**JavaScript HTML Events**

**Ooops!**

## 5.9 Eventos DOM\_Reaccionar a los eventos. **onclick()**



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<p>Click the button to display the date.</p>

<button onclick="displayDate()">The time is?</button>

<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

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

</body>
</html>
```

### JavaScript HTML Events

Click the button to display the date.

The time is?

Tue Mar 08 2022 15:33:51 GMT+0100 (hora estándar de Europa central)



## 5.9 Eventos DOM\_Asignar eventos utilizando el HTML DOM



El HTML DOM le permite asignar eventos a elementos HTML usando JavaScript:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<p>Click "Try it" to execute the displayDate() function.</p>

<button id="myBtn">Try it</button>

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

<script>
document.getElementById("myBtn").onclick = displayDate;

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```

### JavaScript HTML Events

Click "Try it" to execute the displayDate() function.

Try it

Tue Mar 08 2022 15:35:36 GMT+0100 (hora estándar de Europa central)

## 5.9 Eventos DOM\_Los eventos onload y onunload



Los eventos **onload** y **onunload** se activan cuando el usuario entra o sale de la página.

El **onload** evento se puede utilizar para comprobar el tipo de navegador y la versión del navegador del visitante, y cargar la versión adecuada de la página web en función de la información.

Los eventos **onload** y **onunload** se pueden utilizar para tratar con las cookies.

### JavaScript HTML Events

Cookies are enabled.

```
<!DOCTYPE html>
<html>
<body onload="checkCookies()">

<h2>JavaScript HTML Events</h2>

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

<script>
function checkCookies() {
  var text = "";
  //BOM
  if (navigator.cookieEnabled == true) {
    text = "Cookies are enabled.";
  } else {
    text = "Cookies are not enabled.";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

## 5.9. Eventos DOM\_El evento onchange

El **onchange** evento se usa a menudo en combinación con la validación de campos de entrada.

A continuación se muestra un ejemplo de cómo utilizar onchange. La `toUpperCase()` función se llamará cuando un usuario cambie el contenido de un campo de entrada.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
Enter your name: <input type="text" id="fname" onchange="toUpperCase()">
<p>When you leave the input field, a function is triggered which transforms the input
text to upper case.</p>

<script>
function upperCase() {
  const x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>

</body>
</html>
```

### JavaScript HTML Events

Enter your name:

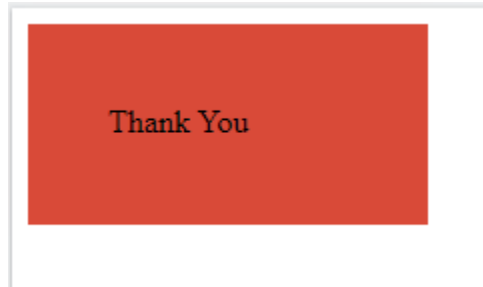
When you leave the input field, a function is triggered which transforms the input text to upper case.

Index\_501

## 5.9. Eventos DOM\_Los eventos onmouseover y onmouseout



Los eventos **onmouseover** y **onmouseout** se pueden usar para activar una función cuando el usuario pasa el mouse sobre o fuera de un elemento HTML:



```
<!DOCTYPE html>
<html>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver(obj) {
  obj.innerHTML = "Thank You"
}

function mOut(obj) {
  obj.innerHTML = "Mouse Over Me"
}
</script>

</body>
</html>
```



Los eventos **onmousedown**, **onmouseup** y **onclick** son todos partes de un clic del mouse.

Primero, cuando se hace clic en un botón del mouse, se activa el evento **onmousedown**, luego, cuando se suelta el botón del mouse, se activa el evento **onmouseup**, finalmente, cuando se completa el clic del mouse, se activa el evento **onclick**.



```
<!DOCTYPE html>
<html>
<body>

<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-color:#D94A38;width:90px;height:20px;padding:40px;">
Click Me</div>

<script>
function mDown(obj) {
  obj.style.backgroundColor = "#1ec5e5";
  obj.innerHTML = "Release Me";
}

function mUp(obj) {
  obj.style.backgroundColor="#D94A38";
  obj.innerHTML="Thank You";
}
</script>

</body>
</html>
```

Index\_510



Cambiar una imagen cuando un usuario mantiene presionado el botón del mouse.

Otros dos eventos relacionados con el mouse son **mousedown** y **mouseup**.

El evento **mousedown** se dispara inmediatamente luego que presionamos con la flecha del mouse un elemento HTML que tiene registrado dicho evento.

El evento **mouseup** se ejecuta luego de soltar el botón del mouse estando dentro del control HTML.

Para probar estos eventos implementaremos una página que contenga dos div, a un div le asignaremos el evento **mousedown** y al otro el evento **mouseup**. Cuando ocurra el evento procederemos a cambiar el texto contenido dentro del div. **index\_511.html**

En la función inicio registramos los eventos **mousedown** y **mouseup** para los dos div, al div recuadro1 procedemos a registrar el evento **mousedown** y al div recuadro2 procedemos a registrar el evento **mouseup**:

```
function inicio() { document.getElementById('recuadro1').addEventListener('mousedown',presion1,false);  
document.getElementById('recuadro2').addEventListener('mouseup',presion2,false); }
```

El método que se dispara para el primer div procedemos a modificar todo el contenido del div accediendo a la propiedad innerHTML

```
function presion1() { document.getElementById('recuadro1').innerHTML='Se presione el mouse y todavía no se soltó'; }
```

De forma similar codificamos la función que modifica el segundo div:

```
function presion2() { document.getElementById('recuadro2').innerHTML='Se presione el mouse y se soltó'; }
```



Dos eventos que podemos capturar de cualquier elemento HTML son cuando un usuario hace un clic o un doble clic sobre el mismo. El evento click se produce cuando el usuario hace un solo clic sobre el elemento HTML y suelta inmediatamente el botón del mouse en el mismo lugar y el dblclick cuando presiona en forma sucesiva en la misma ubicación.

Para probar como registramos estos eventos implementaremos una página que muestre dos div y definiremos el evento click para el primero y el evento dblclick para el segundo.. **index\_512.html**

Primero registramos mediante la llamada al método addEventListener el evento load de la página (recordemos que esta forma de registrar los eventos no funciona en el IE8 o anteriores, pero en muy poco tiempo la cantidad de dispositivos con dichos navegadores serán ínfimas), y dentro de la función inicio registramos los eventos click y dblclick para los dos div que definimos en la página HTML:

```
addEventListener('load', inicio, false);  
function inicio() {  
  document.getElementById('recuadro1').addEventListener('click', presion1, false);  
  document.getElementById('recuadro2').addEventListener('dblclick', presion2, false); }
```

Como vemos lo único que hacemos en los métodos que se disparan es mostrar un mensaje:

```
function presion1() {  
  alert('se hizo click');  
}  
  
function presion2() {  
  alert('se hizo doble click');  
}
```



El evento `mouseover` se dispara cuando ingresamos con la flecha del mouse a un control HTML que tiene registrado dicho evento, y `mouseout` se dispara cuando sacamos la flecha del mouse del control.

Es muy común utilizar estos dos eventos para producir cambios en el elemento HTML cuando ingresamos la flecha del mouse y retornar al estado anterior cuando se saca la flecha del mouse.

Es importante hacer notar que estos eventos se disparan sin tener que presionar la flecha del mouse, solo con desplazarla al interior del elemento HTML se dispara el evento `mouseover`.

Implementaremos un pequeño ejemplo que muestre un cuadrado, el mismo mostrará los bordes redondeados cuando ingresemos la flecha del mouse en su interior y volverá al estado anterior cuando retiremos la flecha. [index\\_513.html](#)

Definimos en el HTML un div de color azul de 200 píxeles de lado:

```
<div style="width:200px;height:200px;background:#0000ff" id="recuadro1"> </div>
```

En la función inicio registramos los eventos `mouseover` y `mouseout`:

```
function inicio() {  
    document.getElementById('recuadro1').addEventListener('mouseover',entrada,false);  
    document.getElementById('recuadro1').addEventListener('mouseout',salida,false); }  

```

El método `entrada` se ejecuta cuando ingresemos la flecha del mouse en el div llamado `recuadro1` y procedemos en el mismo a modificar la propiedad `borderRadius` del estilo de este elemento (indicamos que el redondeo de los vértices del div sea de 30 píxeles):

```
function entrada() { document.getElementById('recuadro1').style.borderRadius='30px'; }
```

Cuando sale la flecha del mouse del div se ejecuta la función `salida` donde fijamos con 0 la propiedad `borderRadius`:

```
function salida() { document.getElementById('recuadro1').style.borderRadius='0px'; }
```





El evento `mousemove` se dispara cada vez que desplazamos la flecha del mouse sobre el elemento HTML que esta escuchando este evento.

Este tipo de evento hay que utilizarlo con cuidado ya que puede sobrecargar el navegador, ya que este evento se dispara cada vez que desplazamos aunque sea solo un pixel.

Implementaremos para ver su funcionamiento una página que muestre un div que capture el evento `mousemove` y como acción incrementaremos un contador para saber la cantidad de veces que se disparó el evento.. [index\\_514.html](#)

Disponemos un div y un párrafo donde mostramos el número 0:

```
<div style="width:200px;height:200px;background:#0000ff" id="recuadro1">
</div> <p id="cantidad">0</p>
```

Capturamos el evento `mousemove` para el div:

```
function inicio() { document.getElementById('recuadro1').addEventListener('mousemove',mover,false); }
```

Cada vez que se emite el evento `mousemove` se llama el método `mover` donde extraemos el valor del párrafo, lo incrementamos en uno y lo volvemos a actualizar:

```
function mover() {
  var x=parseInt(document.getElementById('cantidad').innerHTML);
  x++;
  document.getElementById('cantidad').innerHTML=x;
}
```



### Eventos: keydown, keyup y keypress

Estos tres eventos son similares a los eventos del mouse: mousedown, mouseup y click pero orientados al teclado del equipo. El evento keydown se dispara cuando presionamos cualquier tecla del teclado, el evento keyup cuando soltamos una tecla. En cuanto el evento keypress en un principio procesa tanto cuando se la presionó y soltó, el único y gran inconveniente es que la mayoría de los navegadores no dispara el evento keypress para todas las teclas del teclado.

Para probar el evento keyup implementaremos un programa que permita solo ingresar 140 caracteres y nos informe con un mensaje la cantidad de caracteres disponibles para seguir escribiendo. [Index\\_515.html](#)

### Evento: change

El evento change se lo puede asociar a distintos elementos de formularios HTML y su comportamiento depende de cada uno. Cuando asociamos el evento change a un control de tipo checkbox el mismo se dispara inmediatamente después que lo chequeamos o sacamos la selección. Para los controles de tipo radio también se dispara luego que cambia el estado de selección del mismo.

Para los controles select se dispara el evento change cuando cambiamos el item seleccionado.

Para los controles text y textarea se dispara cuando pierde el foco del control (porque seleccionamos otro control) y hemos cambiando el contenido del mismo.

**Problema** Implementaremos una aplicación que muestre un alert cada vez que sucede el evento change para los controles checkbox, radio, select, text y textarea. . [Index\\_515.html](#)



### Eventos: focus y blur

El evento focus se dispara cuando se activa el control o toma foco y el evento blur se dispara cuando pierde el foco el control. Podemos capturar el evento focus y blur de un control de tipo text, textarea, button, checkbox, file, password, radio, reset y submit.

#### Problema

Confeccionar un formulario que muestre dos controles de tipo text. El que está con foco mostrar su texto de color rojo y aquel que no está seleccionado el texto se debe mostrar de color negro. [Index\\_517.html](#)

### Evento: change

El evento change se lo puede asociar a distintos elementos de formularios HTML y su comportamiento depende de cada uno. Cuando asociamos el evento change a un control de tipo checkbox el mismo se dispara inmediatamente después que lo chequeamos o sacamos la selección. Para los controles de tipo radio también se dispara luego que cambia el estado de selección del mismo.

Para los controles select se dispara el evento change cuando cambiamos el item seleccionado.

Para los controles text y textarea se dispara cuando pierde el foco del control (porque seleccionamos otro control) y hemos cambiando el contenido del mismo.

**Problema** Implementaremos una aplicación que muestre un alert cada vez que sucede el evento change para los controles checkbox, radio, select, text y textarea. . [Index\\_515.html](#)

"640K deberían ser suficientes  
para todo el mundo"

-- Bill Gates, 1981



5.10.

# Listener DOM.

(**M**odelo de **O**bjeto de **D**ocumento)

## 5.10. Oyente de eventos DOM\_El método addEventListener()

Detector de eventos que se active cuando un usuario haga clic en un botón:

Hasta ahora siempre hemos asociado una función distinta para procesar los eventos de distintos elementos HTML. Pero veremos que está permitido asociar una única función a varios eventos de distintos objetos. Debemos definir un parámetro en la función a implementar que llega la referencia del objeto que emitió el evento. Implementaremos un panel con un conjunto de botones que nos permitan ingresar un valor numérico presionando botones.

Asociaremos el click de cada botón con una única función

Puede eliminar fácilmente un detector de eventos utilizando el `removeEventListener()` método.

### Sintaxis

```
element.addEventListener(event, function, useCapture);
```

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

El primer parámetro es el tipo de evento (como "click" o "mousedown" o cualquier otro [evento HTML DOM](#) ).

El **segundo parámetro** es la **función** que queremos llamar cuando ocurra el evento.

El tercer parámetro es un valor booleano que especifica si se debe usar el burbujeo de eventos o la captura de eventos. Este parámetro es **opcional**.

## 5.10. Oyente de eventos DOM\_El método addEventListener() EventListener



Agregue un detector de eventos que se active cuando un usuario haga clic en un botón:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to attach a click event to a button.</p>

<button id="myBtn">Try it</button>

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

<script>
document.getElementById("myBtn").addEventListener("click", displayDate);

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```

### JavaScript addEventListener()

This example uses the addEventListener() method to attach a click event to a button.

Try it

Tue Mar 08 2022 15:45:13 GMT+0100 (hora estándar de Europa central)

## 5.10. Oyente de eventos \_Agregar un controlador de eventos a un elemento

```
<!DOCTYPE html>
<html>
<body>

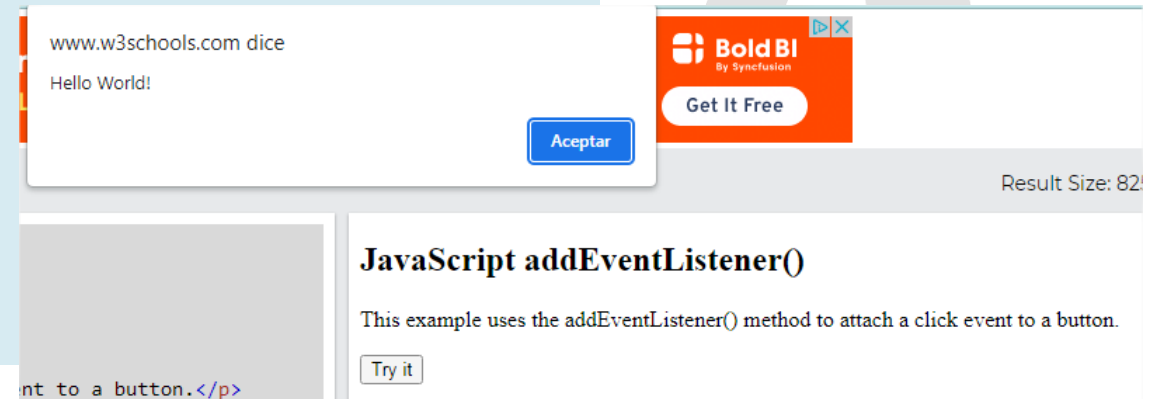
<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to attach a click event to a button. </p>

<button id="myBtn">Try it</button>

<script>
document.getElementById("myBtn").addEventListener("click", function() {
  alert("Hello World!");
});
</script>

</body>
</html>
```





## 5.10. Oyente de eventos \_Agregar un controlador de eventos a un elemento

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

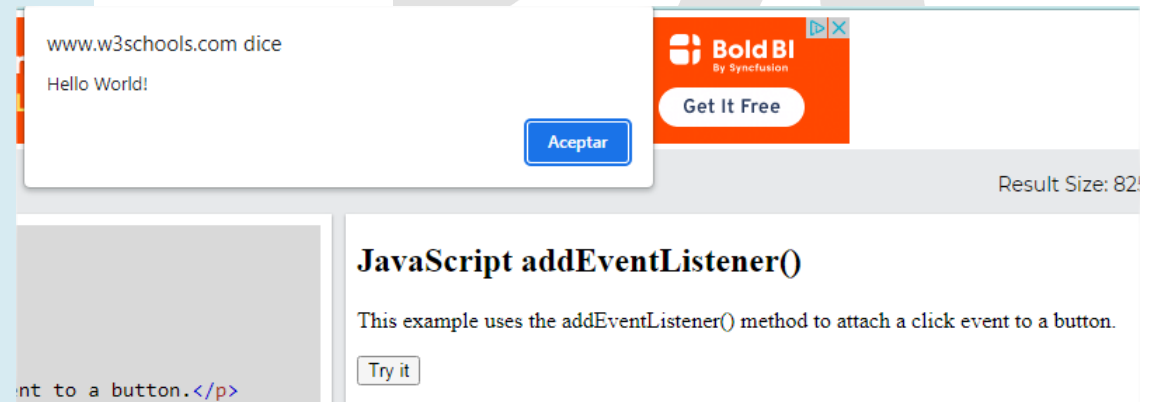
<p>This example uses the addEventListener() method to execute a function when a
user clicks on a button.</p>

<button id="myBtn">Try it</button>

<script>
document.getElementById("myBtn").addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
</script>

</body>
</html>
```



## 5.10. Oyente de eventos \_Agregar muchos controladores de eventos a un elemento

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add two click events to the
same button.</p>

<button id="myBtn">Try it</button>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);

function myFunction() {
  alert ("Hello World!");
}

function someOtherFunction() {
  alert ("This function was also executed!");
}
</script>

</body>
</html>
```

### JavaScript addEventListener()

This example uses the addEventListener() method to add two click events to the same button.

Try it



## 5.10. Oyente de eventos \_Agregar muchos controladores de eventos a un elemento

### JavaScript addEventListener()

This example uses the addEventListener() method to add many events on the same button.

Try it

Moused over!  
Moused out!  
Moused over!  
Moused out!  
Moused over!  
Moused out!  
Moused over!  
Moused out!  
Moused over!  
Moused out!  
Moused over!  
Moused out!  
Moused over!  
Clicked!  
Clicked!  
Moused out!

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add many events on the same button.</p>

<button id="myBtn">Try it</button>

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

<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);

function myFunction() {
  document.getElementById("demo").innerHTML += "Moused over!<br>";
}

function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!<br>";
}

function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused out!<br>";
}
</script>

</body>
</html>
```





## 5.10. Oyente de eventos \_Pasando parámetros

### JavaScript addEventListener()

This example demonstrates how to pass parameter values when using the addEventListener() method.

Click the button to perform a calculation.

Try it

35

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example demonstrates how to pass parameter values when using the
addEventListener() method.</p>

<p>Click the button to perform a calculation.</p>

<button id="myBtn">Try it</button>

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

<script>
let p1 = 5;
let p2 = 7;
document.getElementById("myBtn").addEventListener("click", function() {
  myFunction(p1, p2);
});

function myFunction(a, b) {
  document.getElementById("demo").innerHTML = a * b;
}
</script>

</body>
</html>
```

## 5.10. Oyente de eventos \_bubbling de eventos o captura de eventos?

Hay dos formas de propagación de eventos en HTML DOM, bubbling y captura.

La propagación de eventos es una forma de definir el orden de los elementos cuando ocurre un evento. Si tiene un elemento `<p>` dentro de un elemento `<div>` y el usuario hace clic en el elemento `<p>`, ¿qué evento de "clic" de elemento debe manejarse primero?

Al *burbujear* (bubbling), el evento del elemento más interno se maneja primero y luego el externo: el evento de clic del elemento `<p>` se maneja primero, luego el evento de clic del elemento `<div>`.

Al *capturar* el evento del elemento más externo se maneja primero y luego el interno: el evento de clic del elemento `<div>` se manejará primero, luego el evento de clic del elemento `<p>`.



Index\_531

Index\_533

## 5.10. Oyente de eventos \_bubbling de eventos o captura de eventos?

**Bubbling** quiere decir un evento que se propaga desde el elemento que ejecuto el evento(event.target), hasta el elemento mas lejano en la jerarquia que disponga del mismo evento. Osea, desde la fuente del evento hasta el ancestro mas lejano:

```
form, div, p{  
  border:solid red 1px;  
  padding:2px;  
}
```

```
<form onclick="alert('form')">FORM  
  <div onclick="alert('div')">DIV  
    <p onclick="alert('p')">P</p>  
  </div>  
</form>
```

Fijate como primero se ejecuta el evento p, luego div y por ultimo form. Eso es bubbling. Para detener el bubbling se utiliza el metodo [event.stopPropagation\(\)](#):

```
function formFn() {  
  alert('form'); }  
  
function divFn() {  
  alert('div'); }  
  
function pFn(e) {  
  alert('p'); e.stopPropagation(); }
```

```
form, div, p{  
  border:solid red 1px;  
  padding:2px;  
}
```

```
<form onclick="alert('form')">FORM  
  <div onclick="alert('div')">DIV  
    <p onclick="alert('p')">P</p>  
  </div>  
</form>
```



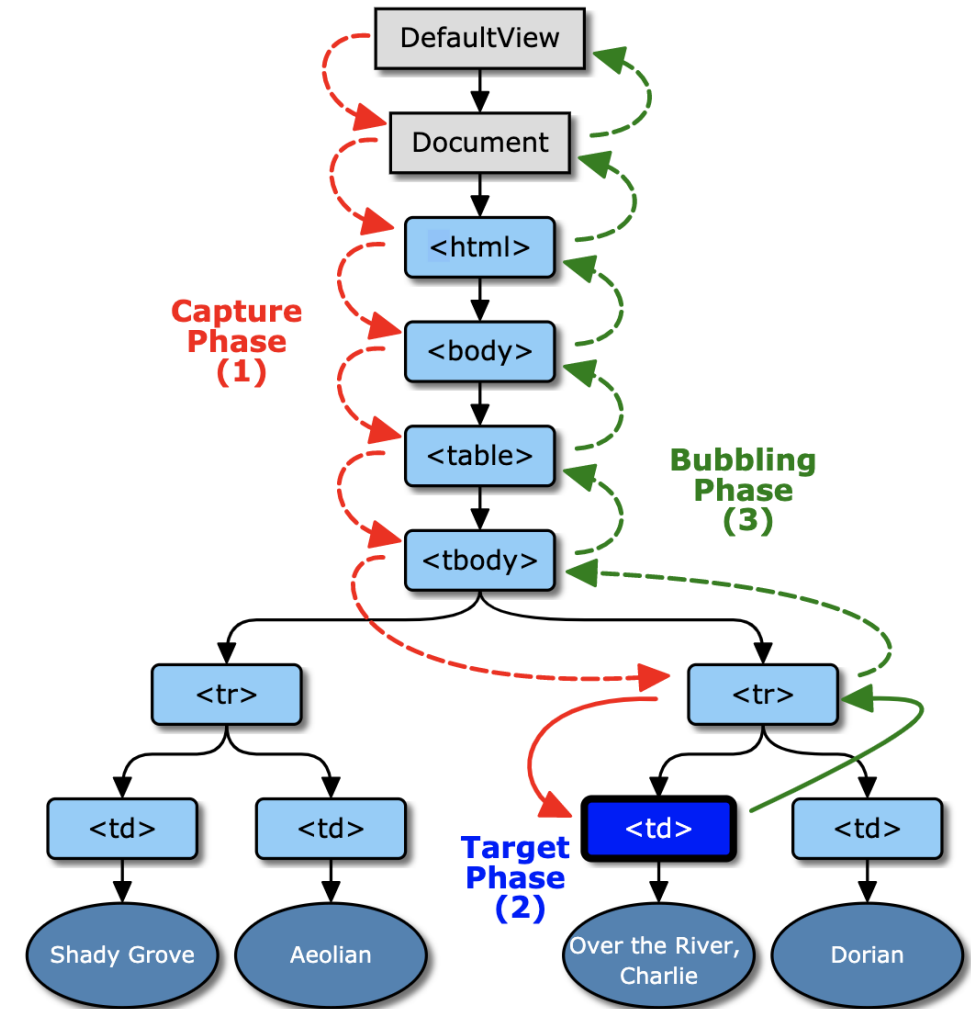
## 5.10. Oyente de eventos \_bubbling de eventos o captura de eventos?

En cuanto a **capturing** es lo contrario a bubbling: que en vez de ejecutar los eventos desde la fuente hasta el ancestro, seria desde el ancestro hasta la fuente.

Por ejemplo mira como aquí hay definidos 2 eventos, 1 para el **form** y otro para el **p**. Nota como al hacer clic sobre el elemento **p** primero se ejecuta el evento del **form** y luego es que va a **p**, mientras que si haces clic sobre **form**, solo se ejecuta su evento.

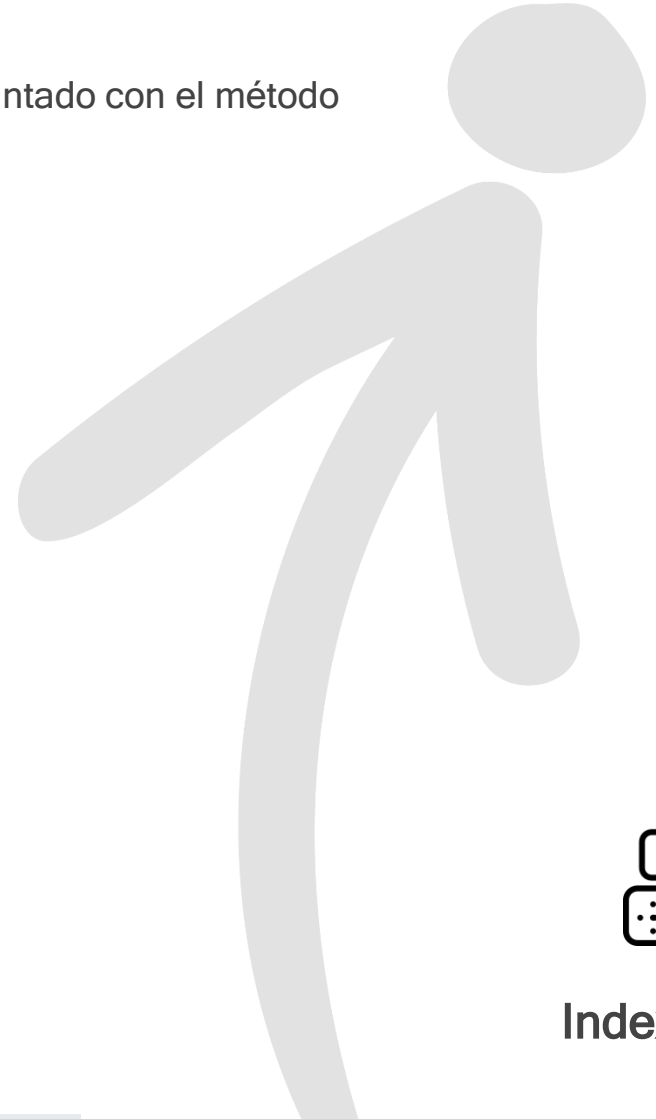
Como ya habrás notado, bubbling es el evento por defecto. Si quieres utilizar capturing pues solo tendrías que pasarle true como ultimo parámetro al método `addEventListener`.

En cuanto a los casos de usos, siempre se recomienda utilizar bubbling por que es la forma mas fácil de descubrir la posición del elemento que lanzo el evento. Nunca me he visto la necesidad e utilizar capturing y seguro tendrás sus razones que por defecto es bubbling.



## 5.10. Oyente de eventos \_El método removeEventListener()

El **removeEventListener()** método elimina los controladores de eventos que se han adjuntado con el método `addEventListener()`:



Index\_532



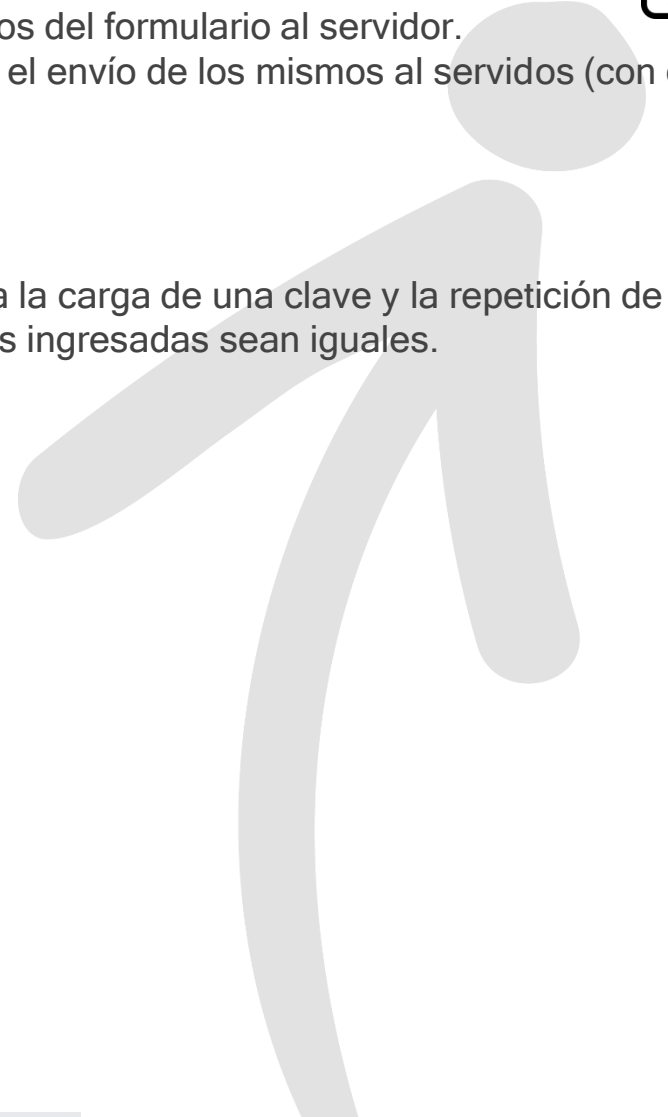



Todo formulario se le puede capturar el evento submit que se dispara previo a enviar los datos del formulario al servidor. Uno de los usos más extendidos es la de validar los datos ingresados al formulario y abortar el envío de los mismos al servidor (con esto liberamos sobrecargas del servidor)

El evento submit se dispara cuando presionamos un botón de tipo type="submit".

Para probar el funcionamiento del evento submit implementaremos un formulario que solicita la carga de una clave y la repetición de la misma. Luego cuando se presione un botón de tipo "submit" verificaremos que las dos claves ingresadas sean iguales.

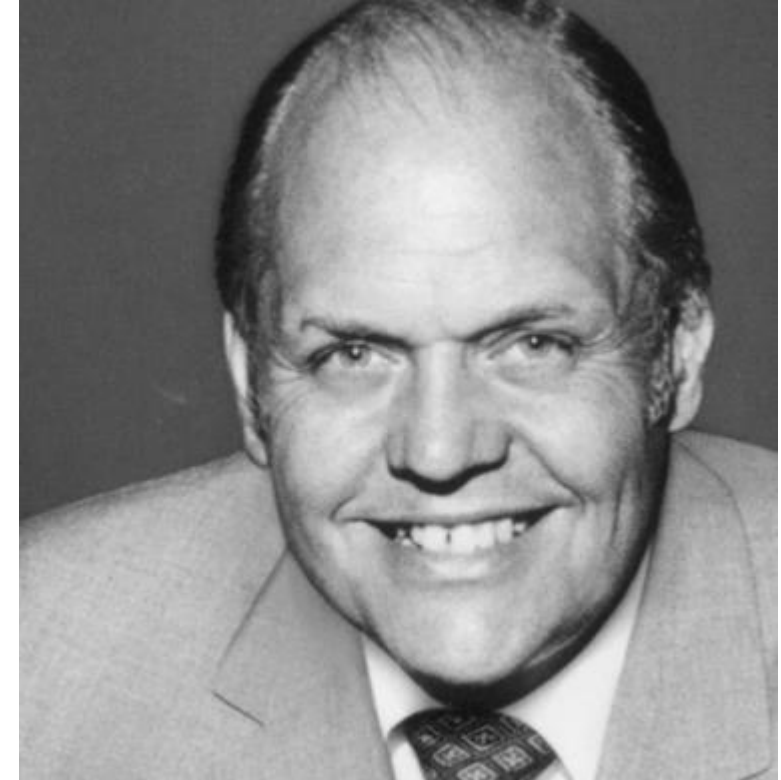
[index\\_534.html](#)



A large, light grey stick figure is positioned behind the main text, with its arms spread wide and legs apart.

"No hay ninguna razón para que  
un individuo tenga un ordenador  
en su casa"

-- Ken Olson, Presidente de Digital Equipment Corporation,  
en 1977

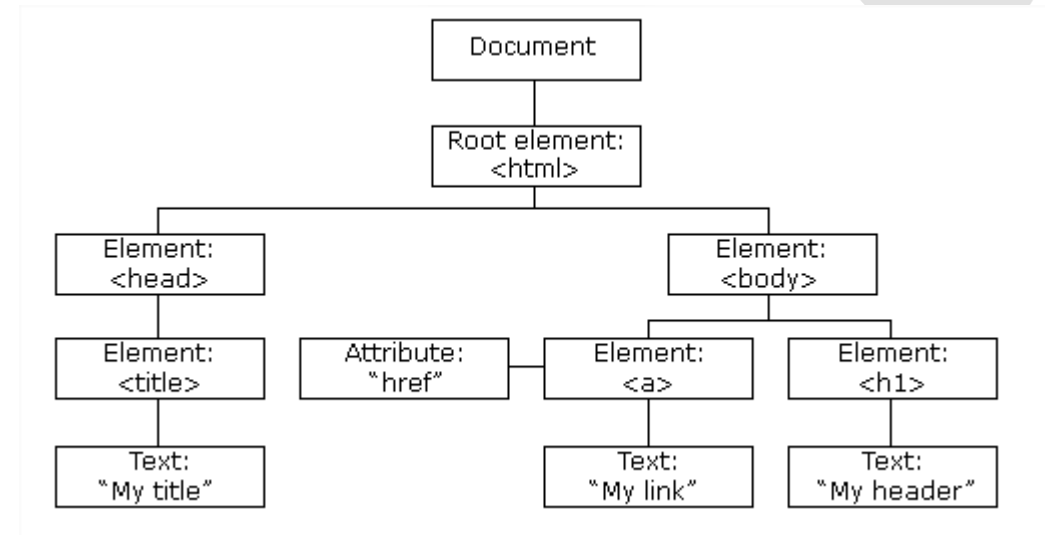


## Navegación DOM. (Modelo de Objeto de Documento)

## 5.11. Navegación DOM\_Nodos DOM

De acuerdo con el estándar W3C HTML DOM, todo en un documento HTML es un nodo:

- Todo el documento es un nodo de documento.
- Cada elemento HTML es un nodo de elemento
- El texto dentro de los elementos HTML son nodos de texto.
- Cada atributo HTML es un nodo de atributo (obsoleto)
- Todos los comentarios son nodos de comentarios.



Con el HTML DOM, JavaScript puede acceder a todos los nodos del árbol de nodos. Se pueden crear nuevos nodos y todos los nodos se pueden modificar o eliminar.

## 5.11. Navegación DOM\_Relación de Nodos

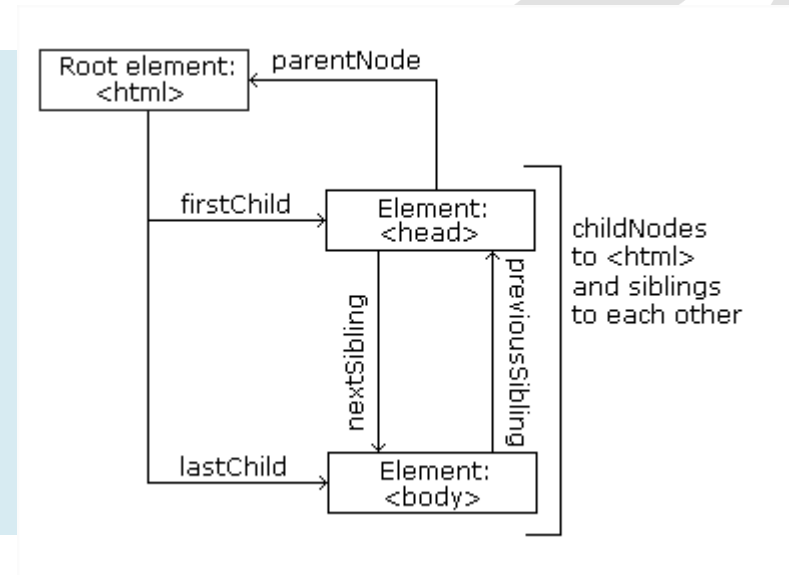
- Los nodos del árbol de nodos tienen una relación jerárquica entre sí.
- Los términos padre, hijo y hermano se utilizan para describir las relaciones.
- En un árbol de nodos, el nodo superior se llama raíz (o nodo raíz)
- Cada nodo tiene exactamente un padre, excepto la raíz (que no tiene padre)
- Un nodo puede tener varios hijos.
- Los hermanos (hermanos o hermanas) son nodos con el mismo padre

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



## 5.11. Navegación DOM\_Relación de Nodos

Desde el HTML anterior se puede leer:

- `<html>` es el nodo raíz
- `<html>` no tiene padres
- `<html>` es el padre de `<head>` y `<body>`
- `<head>` es el primer hijo de `<html>`
- `<body>` es el último hijo de `<html>`

y:



- `<head>`
  - tiene un hijo: `<title>`
- `<title>`
  - tiene un hijo (un nodo de texto): "Tutorial DOM"
- `<body>`
  - tiene dos hijos: `<h1>` y `<p>`
- `<h1>`
  - tiene un hijo: "Lección uno de DOM"
- `<p>`
  - tiene un hijo: "¡Hola mundo!"
- `<h1>`
  - y `<p>` son hermanos

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

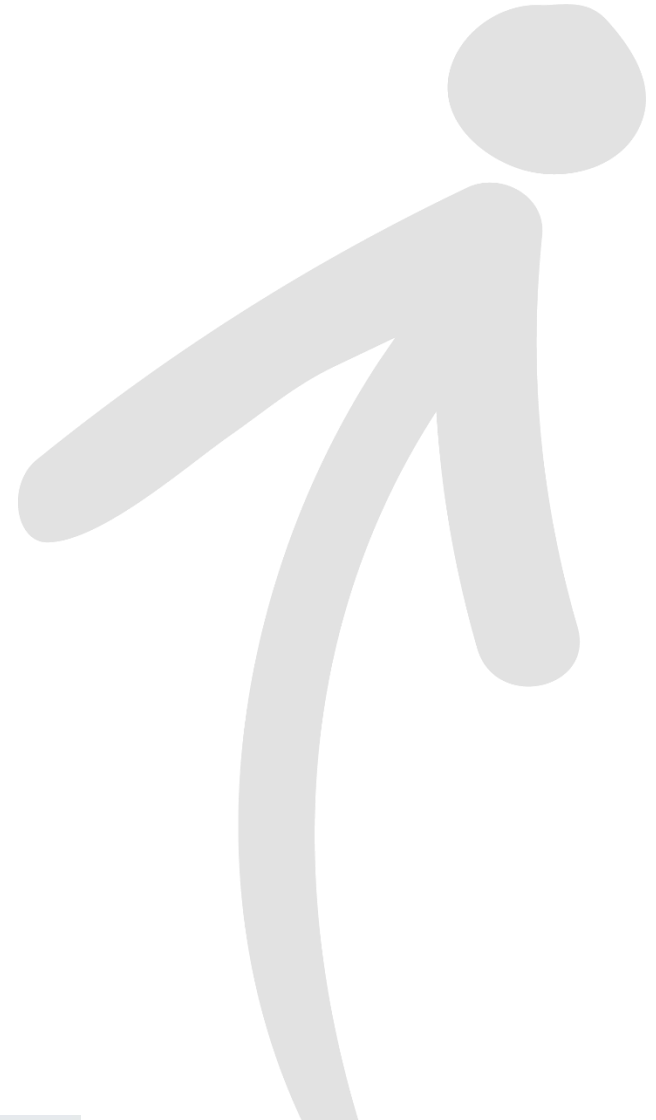
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```

## 5.11. Navegación DOM\_Navegación entre Nodos

Puede usar las siguientes propiedades de nodo para navegar entre nodos con JavaScript:

- parentNode
- childNodes[*nodenumber*]
- firstChild
- lastChild
- nextSibling
- previousSibling



## 5.11. Navegación DOM\_Nodos secundarios y valores de nodos

Un error común en el procesamiento de DOM es esperar que un nodo de elemento contenga texto.

```
<title id="demo">DOM Tutorial</title>
```

El nodo del elemento `<title>` (en el ejemplo anterior) **no** contiene texto.

Contiene un **nodo de texto** con el valor "Tutorial DOM".

Se puede acceder al valor del nodo de texto mediante la `innerHTML` propiedad del nodo:

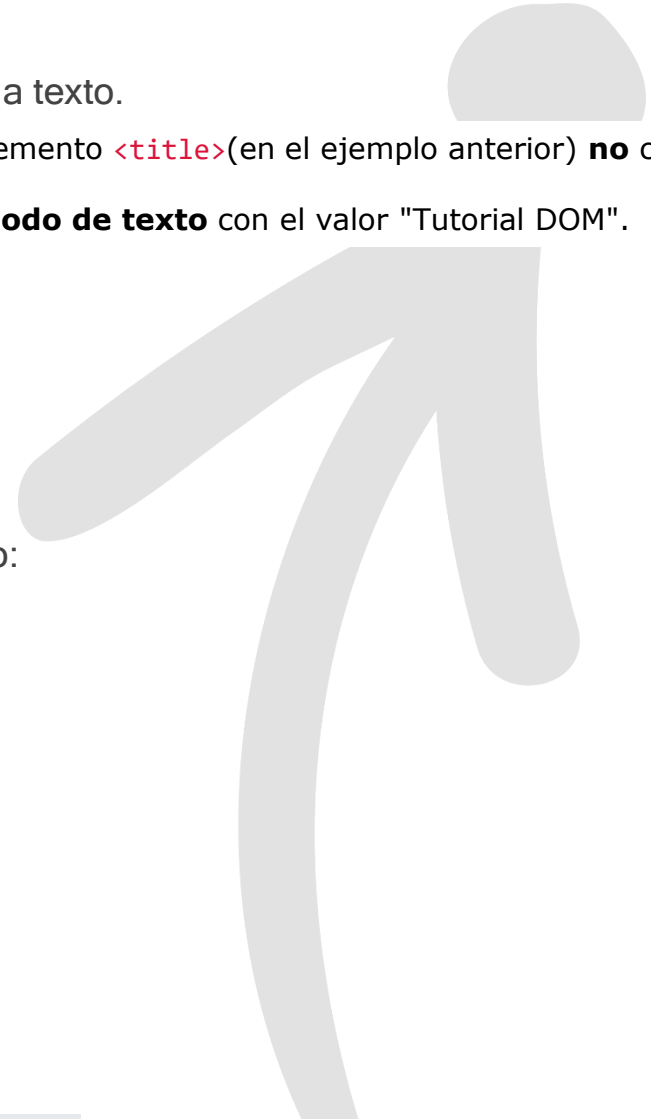
```
myTitle = document.getElementById("demo").innerHTML;
```

Acceder a la propiedad `innerHTML` es lo mismo que acceder a la `nodeValue` del primer hijo:

```
myTitle = document.getElementById("demo").firstChild.nodeValue;
```

El acceso al primer hijo también se puede hacer así:

```
myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```





## 5.11. Navegación DOM\_Nodos secundarios y valores de nodos



Todos los ejemplos siguientes recuperan el texto de un <h1>elemento y lo copian en un <p>elemento:

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").innerHTML;
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").firstChild.nodeValue;
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").childNodes[0].nodeValue;
</script>

</body>
</html><!DOCTYPE html>
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").childNodes[0].nodeValue;
</script>

</body>
</html>
```

## 5.11. Navegación DOM\_InnerHTML - Nodos raíz del DOM



### InnerHTML

En este tutorial usamos la propiedad innerHTML para recuperar el contenido de un elemento HTML. Sin embargo, aprender los otros métodos anteriores es útil para comprender la estructura de árbol y la navegación del DOM.

### Nodos raíz DOM

Hay dos propiedades especiales que permiten el acceso al documento completo:

- document.body- El cuerpo del documento.
- document.documentElement- El documento completo

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTMLDOM</h2>
<p>Displaying document.body</p>

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

<script>
document.getElementById("demo").innerHTML = document.body.innerHTML;
</script>

</body>
</html>
```

### JavaScript HTMLDOM

Displaying document.body

### JavaScript HTMLDOM

Displaying document.body

## 5.11. Navegación DOM\_InnerHTML - Nodos raíz del DOM



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTMLDOM</h2>
<p>Displaying document.documentElement</p>

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

<script>
document.getElementById("demo").innerHTML = document.documentElement.innerHTML;
</script>

</body>
</html>
```

### JavaScript HTMLDOM

Displaying document.documentElement

### JavaScript HTMLDOM

Displaying document.documentElement

## 5.11. Navegación DOM\_La propiedad nodeName



La nodeName propiedad especifica el nombre de un nodo.

- nodeName es de solo lectura
- nodeName de un nodo de elemento es el mismo que el nombre de la etiqueta
- nodeName de un nodo de atributo es el nombre del atributo
- nodeName de un nodo de texto siempre es #text
- nodeName del nodo del documento siempre es #document

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").nodeName;
</script>

</body>
</html>
```

**My First Page**

H1

## 5.11. Navegación DOM\_Creación de nuevos elementos HTML (nodos)



### appendChild()

Para agregar un nuevo elemento al HTML DOM, primero debe crear el elemento (nodo de elemento) y luego agregarlo a un elemento existente.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<p>Add a new HTML Element.</p>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);
const element = document.getElementById("div1");
element.appendChild(para);
</script>

</body>
</html>
```

### JavaScript HTML DOM

Add a new HTML Element.

This is a paragraph.

This is another paragraph.

This is new.

### Ejemplo explicado

Este código crea un nuevo `<p>` elemento:

```
const para = document.createElement("p");
```

Para agregar texto al `<p>` elemento, primero debe crear un nodo de texto. Este código crea un nodo de texto:

```
const node = document.createTextNode("This is a new paragraph.");
```

Luego debe agregar el nodo de texto al `<p>` elemento:

```
para.appendChild(node);
```

Finalmente, debe agregar el nuevo elemento a un elemento existente.

Este código encuentra un elemento existente:

```
const element = document.getElementById("div1");
```

Este código agrega el nuevo elemento al elemento existente:

```
element.appendChild(para);
```

## 5.11. Navegación DOM\_Insertar nuevos elementos HTML (nodos)



### insertBefore()

El appendChild() método del ejemplo anterior, agregó el nuevo elemento como el último hijo del padre.

Si no quieres eso, puedes usar el insertBefore() método:.

#### JavaScript HTML DOM

Add a new HTML Element.

This is new.

This is a paragraph.

This is another paragraph.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<p>Add a new HTML Element.</p>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);

const element = document.getElementById("div1");
const child = document.getElementById("p1");
element.insertBefore(para,child);
</script>

</body>
</html>
```

## 5.11. Navegación DOM\_Eliminación de elementos HTML (nodos)



### Eliminación de elementos HTML existentes

Para eliminar un elemento HTML, utilice el remove() método:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<h3>Remove an HTML Element.</h3>

<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<button onclick="myFunction()">Remove Element</button>

<script>
function myFunction() {
document.getElementById("p1").remove();
}
</script>

</body>
</html>
```

### JavaScript HTML DOM

#### Remove an HTML Element.

This is a paragraph.

This is another paragraph.

Remove Element

### JavaScript HTML DOM

#### Remove an HTML Element.

This is another paragraph.

Remove Element



### Eliminación de un nodo secundario

Para los navegadores que no admiten el remove() método, debe encontrar el nodo principal para eliminar un elemento:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<p>Remove Child Element</p>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
const parent = document.getElementById("div1");
const child = document.getElementById("p1");
parent.removeChild(child);
</script>

</body>
</html>
```

### JavaScript HTML DOM

Remove Child Element

This is another paragraph.



## 5.11. Navegación DOM\_Sustitución elementos HTML (nodos)



### Sustitución de elementos HTML

Para reemplazar un elemento en HTML DOM, use el replaceChild() método:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<h3>Replace an HTML Element.</h3>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is a paragraph.</p>
</div>

<script>
const parent = document.getElementById("div1");
const child = document.getElementById("p1");
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);
parent.replaceChild(para,child);
</script>

</body>
</html>
```

#### JavaScript HTML DOM

##### Replace an HTML Element.

This is new.

This is a paragraph.

## 5.11. Navegación DOM\_El objeto HTMLCollection



El `getElementsByName()` método devuelve un `HTMLCollection` objeto.  
Un `HTMLCollection` objeto es una lista similar a una matriz (colección) de elementos HTML.  
El siguiente código selecciona todos los `<p>` elementos en un documento:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Hello World!</p>

<p>Hello Norway!</p>

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

<script>
const myCollection = document.getElementsByTagName("p");

document.getElementById("demo").innerHTML = "The innerHTML of the second
paragraph is: " + myCollection[1].innerHTML;

</script>

</body>
</html>
```

### JavaScript HTML DOM

Hello World!

Hello Norway!

The innerHTML of the second paragraph is: Hello Norway!

## 5.11. Navegación DOM\_HTMLCollection longitud



La length propiedad define el número de elementos en un HTMLCollection:

### JavaScript HTML DOM

Hello World!

Hello Norway!

Click the button to change the color of all p elements.

Try it

### JavaScript HTML DOM

Hello World!

Hello Norway!

Click the button to change the color of all p elements.

Try it

¡Una HTMLCollection NO es una matriz!

Una HTMLCollection puede parecer una matriz, pero no lo es. Puede recorrer la lista y hacer referencia a los elementos con un número (como una matriz). Sin embargo, no puede usar métodos de matriz como valueOf(), pop(), push() o join() en una HTMLCollection.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Hello World!</p>

<p>Hello Norway!</p>

<p>Click the button to change the color of all p elements.</p>

<button onclick="myFunction()">Try it</button>

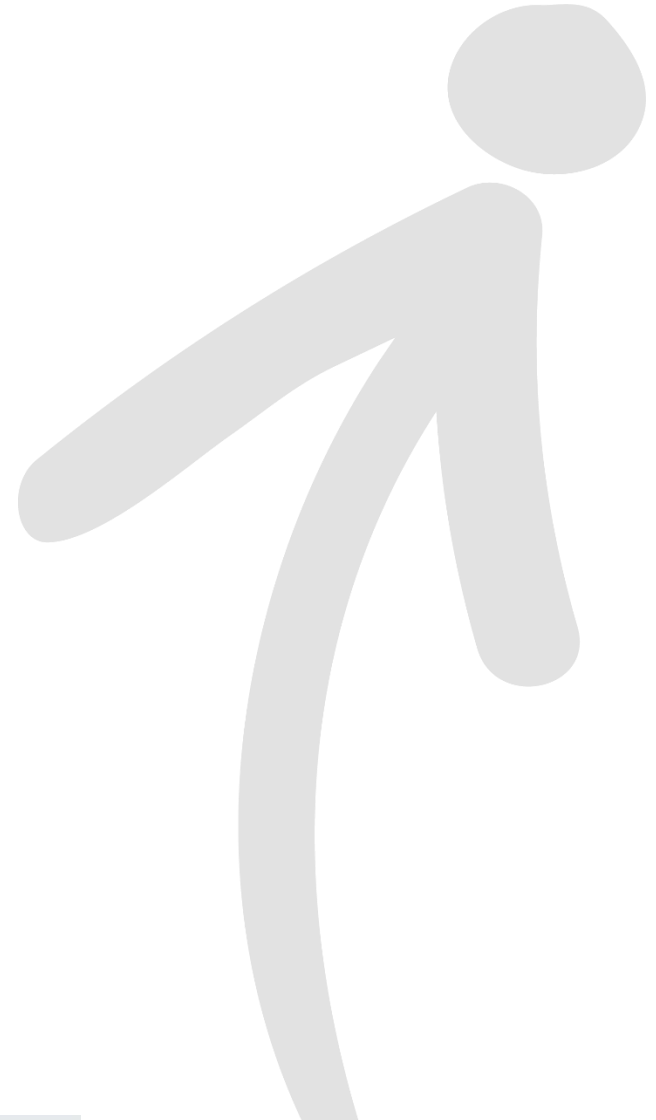
<script>
function myFunction() {
  const myCollection = document.getElementsByTagName("p");
  for (let i = 0; i < myCollection.length; i++) {
    myCollection[i].style.color = "red";
  }
}
</script>

</body>
</html>
```



## 5.11. Navegación DOM\_NodeList

El objeto HTML DOM NodeList



## 5.11. Navegación DOM\_Longitud de la lista de nodos HTML DOM

Longitud de la lista de nodos HTML DOM



## 5.11. Navegación DOM\_ Listas de nodos JavaScript HTML DOM

La diferencia entre una HTMLCollection y una NodeList



## 5.12. Ejercicios DOM\_ DOM Children 5123

Mira esta página:

```
<html>
  <body>
    <div>Users:</div>
    <ul>
      <li>John</li>
      <li>Pete</li>
    </ul>
  </body>
</html>
```

- ¿El nodo <div> del DOM?
- ¿El nodo <ul> del DOM?
- El segundo <li> (con Pete)?

Hay muchas maneras, por ejemplo:  
El nodo <div> del DOM:

```
document.body.firstChild
```

```
document.body.children[0]
```

```
// (el primer nodo es un espacio, así que tomamos el segundo)
document.body.childNodes[1]
```

El nodo <ul> del DOM:

```
document.body.lastElementChild
```

```
document.body.children[1]
```

El segundo <li> (con Pete):

```
// obtener <ul>, y luego obtener su último elemento hijo
document.body.lastElementChild.lastElementChild
```

## 5.12. Ejercicios DOM\_ Seleccionar todas las celdas diagonales 5124

Escribe el código para pintar todas las celdas diagonales de rojo.  
Necesitarás obtener todas las <td> de la <table> y pintarlas usando el código:

```
// td debe ser la referencia a la celda de la tabla
td.style.backgroundColor = 'red';
```

El resultado debe ser:

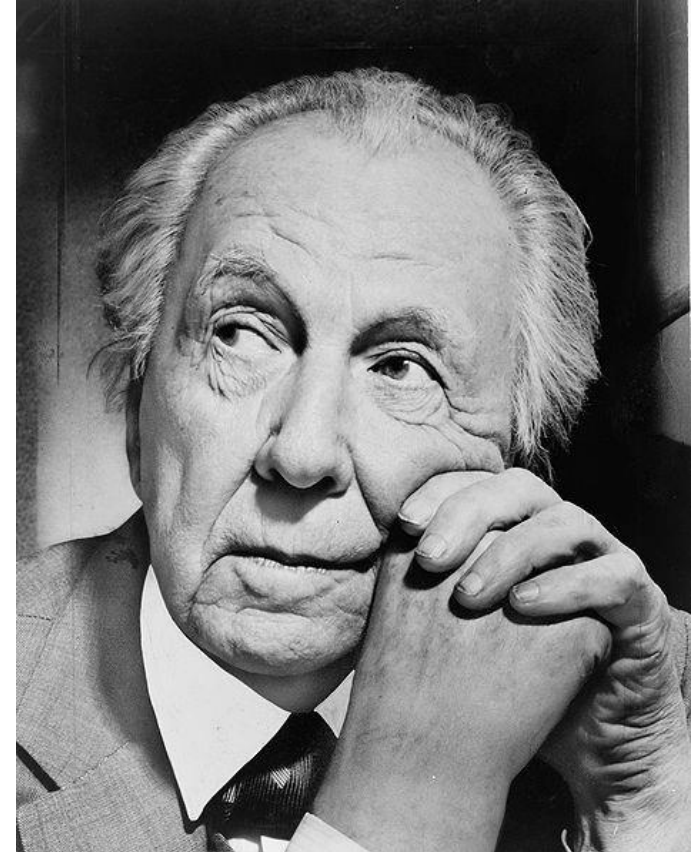
1:1	2:1	3:1	4:1	5:1
1:2	2:2	3:2	4:2	5:2
1:3	2:3	3:3	4:3	5:3
1:4	2:4	3:4	4:4	5:4
1:5	2:5	3:5	4:5	5:5

Usaremos las propiedades de las filas y las celdas para acceder a las celdas de la tabla diagonal



"Si las cosas siguen así, al hombre  
se le atrofiarán todas sus  
extremidades excepto los dedos de  
pulsar los botones"

-- Frank Lloyd Wright





#### **Barcelona**

Francesc Tàrraga 14  
08027 Barcelona  
93 351 78 00

#### **Madrid**

Campanar 12  
28028 Madrid  
91 502 13 40

#### **Reus**

Alcalde Joan Bertran 34-38  
43202 Reus  
977 31 24 36

[info@grupcief.com](mailto:info@grupcief.com)

[www.grupcief.com](http://www.grupcief.com)

