Interacción con el usuario

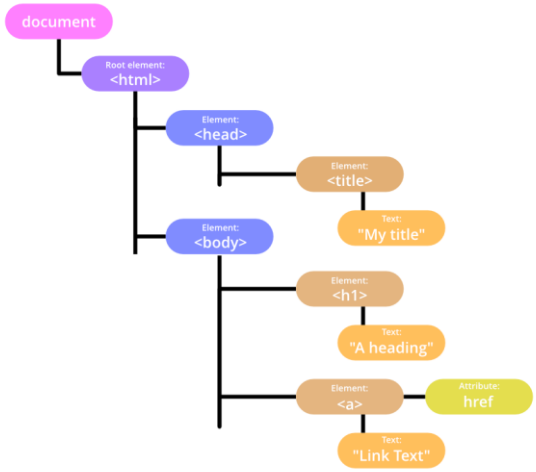
UD1: Introducción a JS

Javier G. Pisano (javiergpi@educastur.org)

Al acabar la lección..

- Conocerás que es el DOM y sus funciones básicas
- Conocerás los principales eventos que pueden desencadenarse al interactuar con una página web

Indice



Introducción al DOM (Document Object Model)

Indice

DOM (*Document Object Model*)

- Habitualmente utilizamos JavaScript para acceder al contenido de la página Web
- Permite acceder a una estructura de datos que representa (en forma de árbol) todo el contenido de una página Web y realizar operaciones de:
  - Lectura
  - Modificación
  - Inserción / eliminación de elementos

Objeto `document`

- Representa el documento cargado en una ventana del navegador
- Permite el **acceso a todas las etiquetas HTML** dentro de una página
- Forma parte del objeto `window`, luego puede ser accedido mediante `window.document` o directamente `document`

Accediendo al DOM
Métodos de `document`

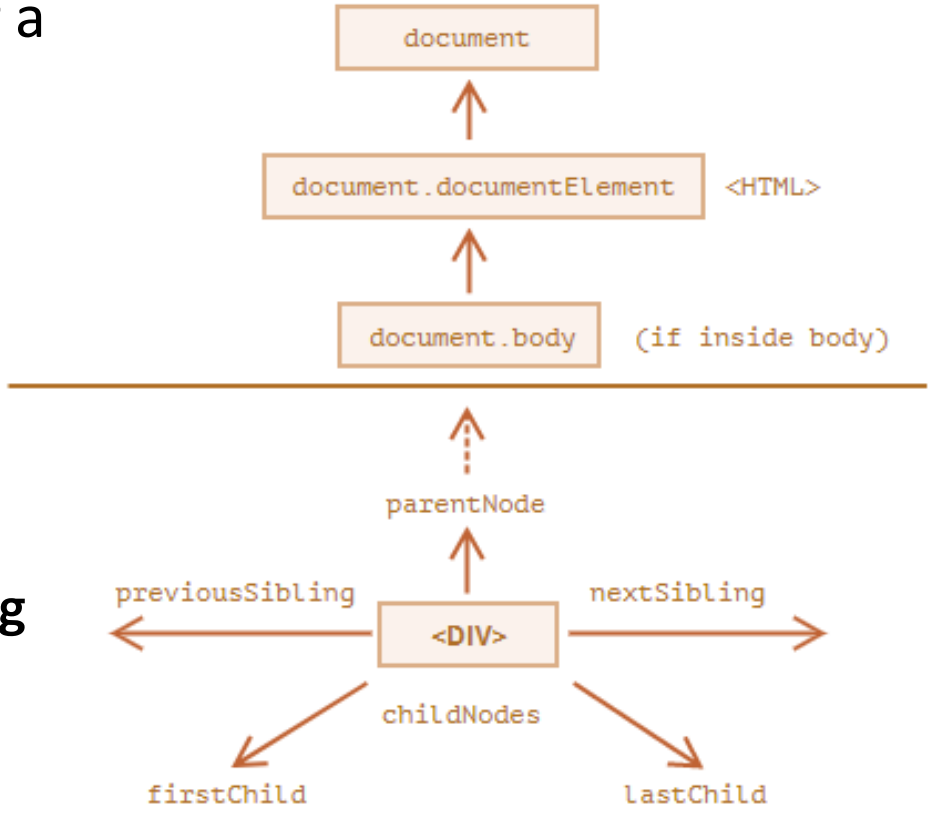
`getElementById(id)`	Devuelve el elemento con id **`id`**
`getElementsByClassName(class)`	Devuelve un array de elementos que tienen clase **`class`**
`getElementsByTagName(tagname)`	Devuelve un array de elementos cuya etiqueta es **`tagname`**
`querySelector(query)`	Devuelve el primer elemento que concuerda con el grupo de selectores CSS especificados entre paréntesis
`querySelectorAll(query)`	Devuelve un array de elementos que concuerdan con el grupo de selectores CSS especificados entre paréntesis

Accediendo al DOM: Ejemplos

```
var elemento1=document.getElementById("id-1");
var elementosClase=document.getElementsByClassName("clase-1");

/* Accedemos a cada elemento de manera secuencial */
for(var i=0;i<elementosClase.length;i++);
var elementoClase1=elementosClase[i];

/* Equivale a la primera sentencia */
elemento1=document.querySelector("#id-1");

/* Equivale a la segunda sentencia */
elementos=document.querySelectorAll(".clase-1");

/* Obtendría elementos[0] */
 var elemento1=document.querySelector(".clase-1");
```

Accediendo al DOM: Parentescos

- Desde un nodo puedo acceder a los nodos relacionados:
  - A los hijos con la propiedad **`childNodes`**.
    - **`firstChild:`** el primero
    - **`lastChild:`** el último
  - A su padre con la propiedad **`parentNode`**
  - A sus hermanos con las propiedades **`previousSibling`** y **`nextSibling`**.



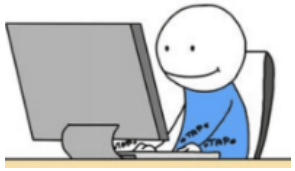
Cambiar el contenido

**`elemento.innerHTML = `*`nuevo valor`*	Contenido de cualquier etiqueta HTML. Es posible consultarla y modificarla.
**`elemento.`*`atributo`*` = `*`nuevo valor`* *`elemento.setAttribute(nuevoValor)`*	Dos formas para modificar el contenido de un atributo.

Se desaconsejan, más ineficientes

`open()`	Abre el flujo de escritura para poder usar **`write()`**
`close()`	Cierra el flujo de escritura abierto con **`open()`**
`write(cadena)` **`writeln(cadena)`**	Permite escribir expresiones HTML dentro de un documento (sin o con salto de línea al final)

Ejemplo 1



- Realiza un script que muestre la **fecha actual**
  - Utiliza una hoja de estilos que vincularás desde la página HTML para mejorar la apariencia del calendario:

```
<link rel="stylesheet" href="./css/estilos.css">
```

  - Para poner el contenido en la página web:

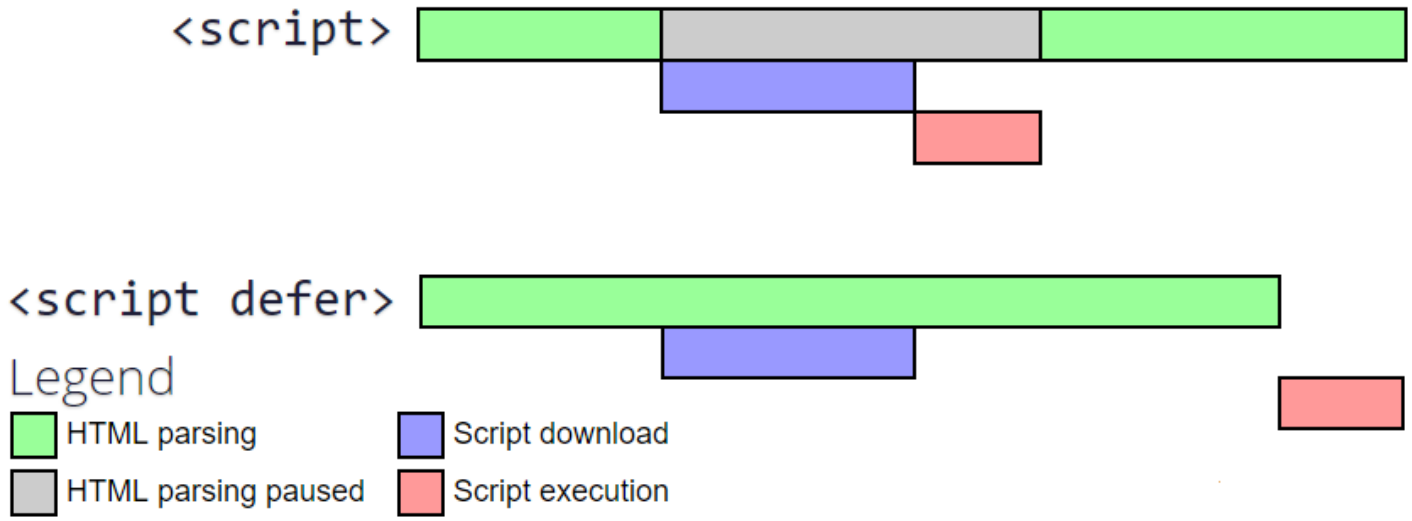
```
document.getElementById("calendario").innerHTML=…
```

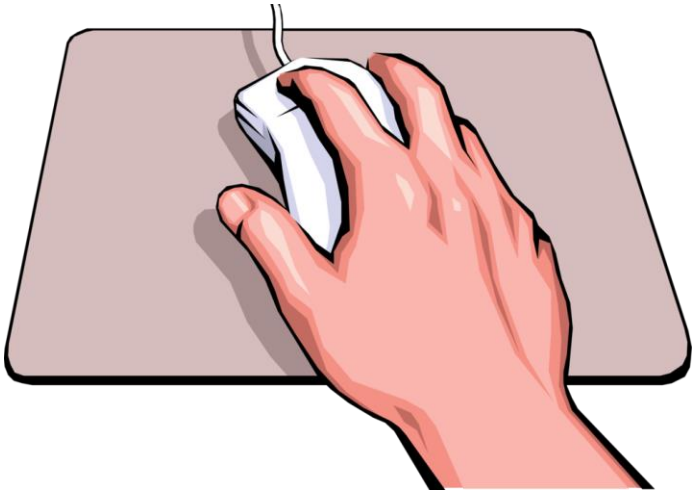
Ten en cuenta…

- Ten en cuenta…
  - Para acceder al DOM la página tiene que haber terminado de cargarse
  - OPCIONES:
    - Usar eventos: Luego hablamos de ello.
    - Palabra reservada **`defer`**

```
<script defer src="scripts/calendario.js">
```

Funcionamiento de defer





Eventos

Indice

Eventos

- Los scripts mostrados hasta ahora se ejecutan secuencialmente
  - Con las estructuras de control de flujo y funciones cambiamos dicho comportamiento ligeramente
- Con JavaScript podemos utilizar un modelo de funcionamiento **basado en eventos**
  - Los scripts se ejecutan una vez que el usuario haya "*hecho algo*" (pulsar un botón, mover el ratón, cerrar la ventana..), es decir, cuando se produzca un **evento**
  - Puedo asignar una función a cada uno de dichos eventos. Este tipo de funciones se llaman **manejadores de eventos**

Modelos de eventos

- La mayor parte de incompatibilidades entre navegadores se produce en el modelo de eventos
  - El **modelo básico de eventos** es el que cumplen la mayoría
- Cada elemento HTML define su propia lista de posibles eventos que se le pueden asignar
  - Un mismo evento (ej: pulsar el ratón) puede estar definido para varios elementos HTML distintos, y un elemento puede tener asociados varios eventos distintos

Principales eventos I

Evento	Descripción	Elementos para los que está definido
`click`	Pinchar y soltar el ratón	Todos los elementos
`dblclick`	Pinchar dos veces seguidas con el ratón	Todos los elementos
`focus`	Seleccionar un elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`keydown`	Pulsar una tecla (sin soltar)	Elementos de formulario y `<body>`
`keypress`	Pulsar una tecla	Elementos de formulario y `<body>`
`keyup`	Soltar una tecla pulsada	Elementos de formulario y `<body>`
`load`	La página se ha cargado completamente	`<body>`
`mousedown`	Pulsar (sin soltar) un botón del ratón	Todos los elementos
`blur`	Deseleccionar el elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`change`	Deseleccionar un elemento que se ha modificado	`<input>`, `<select>`, `<textarea>`

Principales eventos II

Evento	Descripción	Elementos para los que está definido
`mousemove`	Mover el ratón	Todos los elementos
`mouseout`	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
`mouseover`	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
`mouseup`	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
`reset`	Inicializar el formulario (borrar todos sus datos)	`<form>`
`resize`	Se ha modificado el tamaño de la ventana del navegador	`<body>`
`select`	Seleccionar un texto	`<input>, <textarea>`
`submit`	Enviar el formulario	`<form>`
`unload`	Se abandona la página (por ejemplo al cerrar el navegador)	`<body>`

Manejadores de eventos

- Contienen el código que se ejecutará cuando sucede un evento
- Existen varias maneras de indicar los manejadores de eventos
  - El W3C propone una manera de registrar los eventos sobre cualquier objeto ("**Registro de eventos avanzado**") usando el método **`addEventListener()`**

addEventListener (tipo, función,[modo])

- El método recibe 3 argumentos:
  - Tipo de evento (“click”, “resize”, etc.)
  - Función a ejecutar (manejador del evento)
  - Opcionalmente el modo de transmisión de eventos entre elementos anidados
    - **true**: Registro el evento en la fase de **captura**.
    - **false**: Registro el evento en la fase de **burbujeo**.

Registro de eventos avanzado: Ejemplo

```
document.getElementById("enlace").addEventListener('click’, alertar, false);

function alertar(){
 alert("Te conectaremos con la página: "+this.href);
}
```

Función manejadora

```
element.addEventListener('click', function () {
            this.style.backgroundColor = '#cc0000';
            }, false)
```

En el calendario…

- Para que el script se ejecute al cargar la página (alternativa al defer):

```
window.addEventListener("load", principal);

function principal(){
//esto se ejecuta al cargar la página
}
```

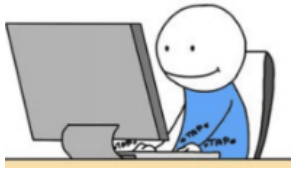
- Aunque también es muy habitual:

```
window.onload = principal;

function principal(){
//esto se ejecuta al cargar la página
}
```

Otras operaciones con eventos

- De la misma forma se pueden:
  - Eliminar eventos:
    - `removeEventListener(evento, manejador)`
  - Cancelar la ejecución de eventos:
    - `preventDefault()`

Ejemplo 2

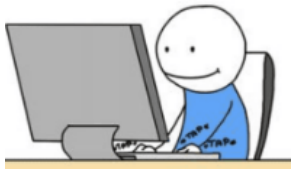


- Tenemos una página con un botón, un párrafo y una caja de texto.

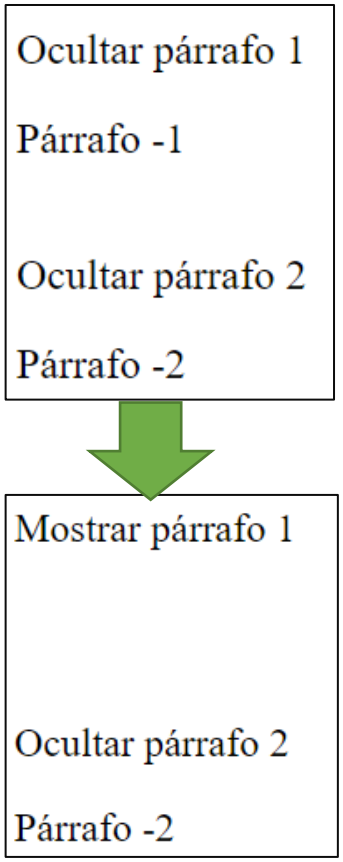
```
<textarea id="taEntrada" placeholder="Escribe aquí"></textarea>
    <button id="btnBoton">Púlsame</button>
    <div id="divSalida">
        <p id="pSalida">Hola</p>
    </div>
```

- Queremos que al pulsar el botón se muestre en el párrafo el texto de la caja de texto

Ejemplo 3



- Dada una página con un enlace y un párrafo:
  - Queremos que al pulsar sobre el enlace el párrafo se oculte/muestre de manera alternativa
  - El texto del enlace debería cambiarse (Ocultar/mostrar)
- Modificar el script para que el método sirva para N conjuntos enlace+ párrafo, donde N es un número escogido por el usuario



Información sobre el evento (I)

- Distinguir **qué** evento ha ocurrido:

```
document.getElementById("eventos").addEventListener("mouseover", manejador);
document.getElementById("eventos").addEventListener("mouseout", manejador);
function manejador(e) {
        switch (e.type) {
        case "mouseover":
            this.style.color = "purple";
            break;
        case "mouseout":
            this.style.color = "yellow";
            break;
        }
    }
```

La función manejadora puede recibir como parámetro el evento en sí (no importa el nombre)

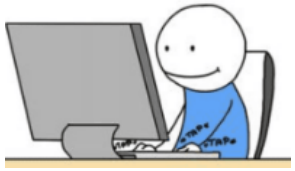
Información sobre el evento (II)

- Distinguir **sobre qué** elemento ha ocurrido:

```
document.getElementById("parrafo1").addEventListener("click", saludo);
document.getElementById("parrafo2").addEventListener("click", saludo);

 function saludo(e) {
      if (e.target.id == "parrafo1")
               alert("Has pulsado el primer párrafo");
      else if (e.target.id == "parrafo2")
               alert("Has pulsado el segundo párrafo");
      alert("Has pulsado el " + e.target.id);
      }
```

Ten en cuenta...

- **El elemento anidado más profundo que causó el evento es llamado elemento objetivo, accesible como `event.target`**
- Nota la diferencia:
  - `event.target` – es el elemento "objetivo" que inició el evento, no cambia a través de todo el proceso de propagación
  - `this (=event.currentTarget)` – es el elemento "actual", el que tiene un manejador ejecutándose en el momento



Ejemplo 4

- Tenemos una página HTML (**`ejemplo4.html`**) con N fotos de perros y gatos, al lado de cada una de las cuales hay una etiqueta, además de haber otra etiqueta general.
- Queremos que:
  - Al pulsar sobre cada una de las imágenes la etiqueta muestre el número de pulsaciones sobre su imagen correspondiente.
  - La etiqueta general muestre el nombre del último animal sobre el cual hemos pinchado.

La función manejadora **recibe como parámetro el evento**, y se puede usar para saber el elemento pulsado

Manipulación del DOM

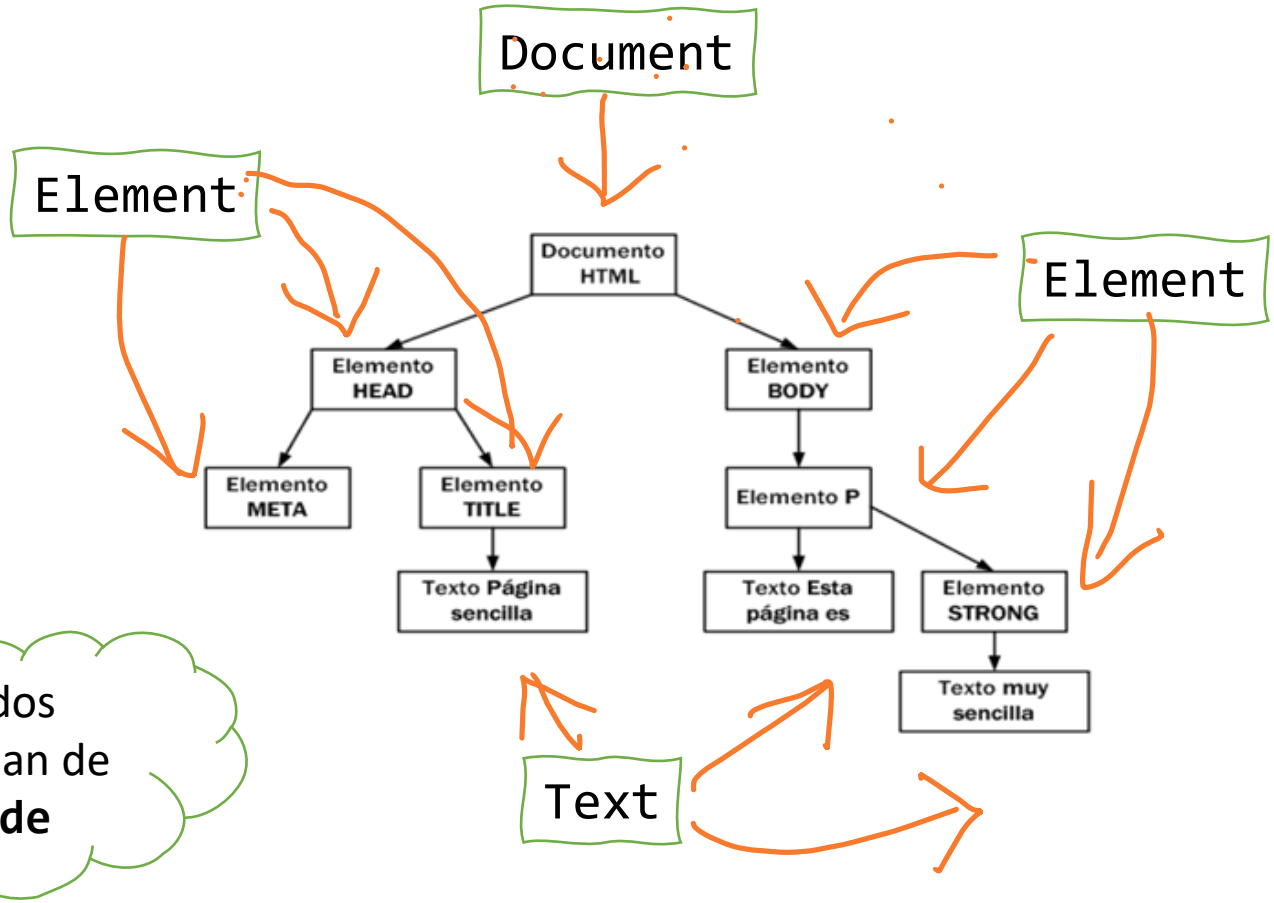
Indice

Nodos del DOM (Node)

- Cada nodo del DOM es un objeto de un tipo derivado de la interfaz **Node**
- Existen 12 tipos de nodos, pero normalmente manejaremos los siguientes:

Document	Raíz del que derivan el resto. Se instancia un objeto de esta clase llamado **document** mediante el cual puedo acceder al contenido del documento.
Element	Etiqueta HTML. Puede contener atributos y otros nodos.
Attr	Atributos de las etiquetas
Text	Texto encerrado por una etiqueta HTML
Comment	Comentarios

Ejemplo

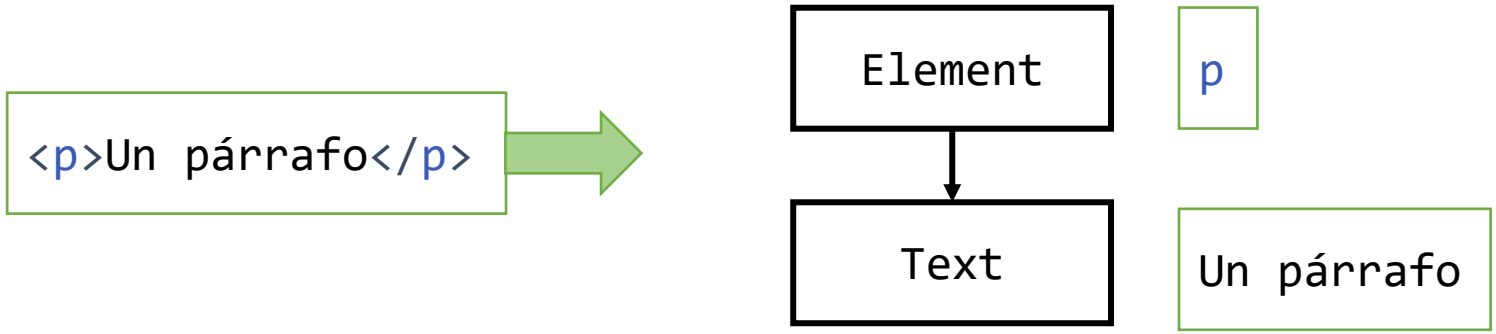

Document
Element
Element
Documento HTML
Elemento HEAD
Elemento BODY
Elemento META
Elemento TITLE
Elemento P
Texto Página sencilla
Texto Esta página es
Elemento STRONG
Texto muy sencilla
Text
Todos heredan de **Node**

Modificando el DOM: Métodos de Node

`appendChild(nodo)`	Añade un nodo al final de la lista de nodos hijos.
`removeChild(nodo)`	Elimina el nodo cuya referencia le proporcionamos de la lista de nodos hijos.
`replaceChild(nuevo,reemplazado)`	Reemplaza el nodo viejo por el nuevo, siendo **`nuevo`** y **`reemplazado`** referencias a nodos.
`insertBefore(nuevo,anterior)`	Inserta el nodo **`nuevo`** antes de **`anterior`**, siendo ambos referencias a nodos.

Creación de elementos (I)

- Un elemento HTML genera dos nodos:
  - Un nodo de tipo **`Element`** que representa la propia etiqueta
  - Un nodo de tipo **`Text`** que representa el texto de la etiqueta (lo que hay entre la apertura y el cierre de la misma.
    - No se genera si la etiqueta es de tipo sencillo (como **`<img />`**)



Modificando el DOM : Métodos de `document`

`createElement(tagname)`	Crea el elemento HTML cuya etiqueta indica **`tagname`**
`createTextNode(texto)`	Crea el nodo de texto con el contenido **`texto`**.
`createAttribute(name)`	Crea el atributo llamado **name** Posteriormente hay que agregarlo al elemento
`createComment(comentario)`	Crea el comentario con el contenido **`comentario`**.

Creación de elementos (II)

- La creación de un elemento consta de 4 pasos:
  1. Creación del nodo de tipo **Element**
  2. Creación del nodo de tipo **Text**
  3. Añadir el nodo **Text** como nodo hijo del nodo **Element**
  4. Añadir el nodo **Element** a la página, como hijo del nodo donde quiero insertarlo

También se puede modificar directamente la propiedad **`innerHTML`** del nodo padre

- *Ejemplo: Añadir un párrafo al final de la página:*

```
var parrafo=document.createElement("p");
var contenido=document.createTextNode("Contenido del párrafo");

parrafo.appendChild(contenido);
document.body.appendChild(parrafo);
```

Eliminación de nodos

- Para eliminar un nodo llamamos a la función **`removeChild`**() del nodo padre del que queremos borrar, pasando el propio nodo a borrar

```
var padre=document.getElementById("listado")
var elementoBorrar==document.getElementById("borrame");
padre.removeChild(elementoBorrar);
```

- Lo más común es acceder al padre del elemento a borrar mediante la propiedad **`parentNode`** de este

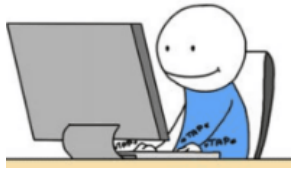
```
var elementoBorrar==document.getElementById("borrame");
elementoBorrar.parentNode.removeChild(elementoBorrar);
```

Reemplazo de nodos

- Podemos reemplazar un nodo llamando al método **replaceChild()**

```
/* Reemplazamos el primer elemento */
var primerHijo=listado.firstChild;

var nuevoHijo=document.createElement("li");
var texto=document.createTextNode("Nuevo texto");
nuevoHijo.appendChild(texto);

listado.replaceChild(nuevoHijo, primerHijo);
```

Ejemplo 5



- Crear un código html con un div que contenga un enlace y un párrafo con el texto *“Me van a cambiar”.*
- Al cargar la página debe:
  - Mostrar mensaje con alert que indique que iniciamos los cambios – nos permitirá ver el html inicial.
  - Añadir un párrafo al final del body con el texto: “Contenido párrafo”.
  - Borrar el enlace.
  - Cambiar el párrafo por otro cuyo contenido sea vuestro nombre.
  - Añadir un párrafo con la fecha actual antes del div.

Nodos Attribute

- Los nodos tienen **atributos** que representan los atributos HTML

createAttribute (nombre)	Crea un nodo tipo atributo nuevo con el nombre especificado en el parámetro.
hasAttribute (nombre)	Devuelve si el elemento actual tiene un atributo con el nombre especificado o no.
removeAttribute (nombre)	Elimina el atributo con el nombre especificado del elemento actual.
setAttribute (nombre, valor)	Añade un atributo a un elemento con el nombre y valor especificados.
getAttribute (nombre)	Devuelve el valor del atributo con el nombre especificado para el elemento actual.

Acceso a los atributos

```
<a id="enlace" href="http://www.iesnaranco.es">IES Naranco</a>
```

- Poniendo el nombre del atributo en minúsculas
  - A excepción del atributo **`class`**, al que se accede con **`className`**

```
var enlace=document.getElementById("enlace");
console.log(enlace.href); //muestra http://www.iesnaranco.es
```

- Usando los métodos de la transparencia anterior:

```
enlace.setAttribute("href","http://www.iesnaranco.es");
console.log(enlace.getAttribute("href"));
```

- A través de su propiedad **`attributes`**

```
var atributos=elemento.attributes; //Acceso a todos sus atributos
```

Acceso al estilo

- También es posible acceder al estilo de los elementos (atributo **style**)
  - El nombre de las propiedades CSS compuestas se forma eliminando todos los guiones intermedios (-) y escribiendo en mayúscula la letra siguiente.
    - **`font-weight`** se transforma en **`fontWeight`**
    - **`border-top-style`** se transforma en **`borderTopStyle`**
    - Etc.

Acceso al estilo: Ejemplo de uso

```
elemento.style.backgroundColor="red";
elemento.style.fontSize="1.5em";
elemento.style.backgroundImage=="url('fondo.png')";
```

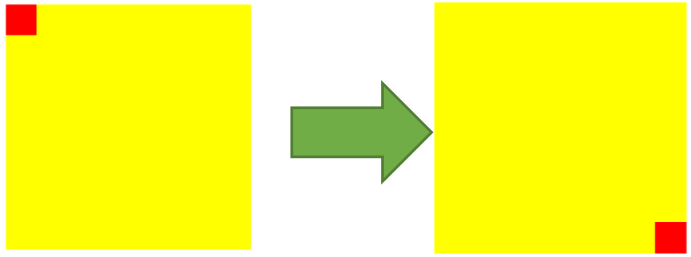
- En lugar de asignar todas las propiedades CSS individualmente, puede ser útil:
  - Agrupar las propiedades CSS que queremos asignar en una clase definida en el archivo .css
  - Asignar la clase al elemento cuyo estilo queremos cambiar (o quitarla si queremos quitar el estilo)

```
estilo-chulo{
  background-color:"red";
  font-size:1.5em;
  background-image:url('fondo.png');
}
```

```
elemento.className="estilo-chulo";
```

Animaciones

- Poder acceder al estilo nos va a permitir hacer animaciones:



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

```
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";
      }
    }
  }
```

Manejo de tablas desde el DOM

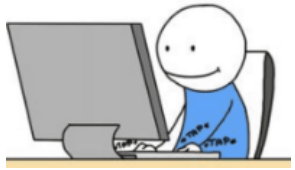
- DOM proporciona métodos específicos para trabajar con tablas:
  - Elemento `table`

`rows`	Array con las filas de la tabla
`insertRow(posicion)`	Inserta una nueva fila en la posición indicada
`deleteRow(posicion)`	Elimina una fila de la posición indicada

  - Elemento `celda`

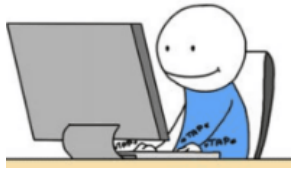
`cells`	Devuelve un array con las columnas de la fila.
`insertCell(posicion)`	Inserta una nueva columna en la posición indicada
`deleteCell(posicion)`	Elimina la columna de la posición indicada

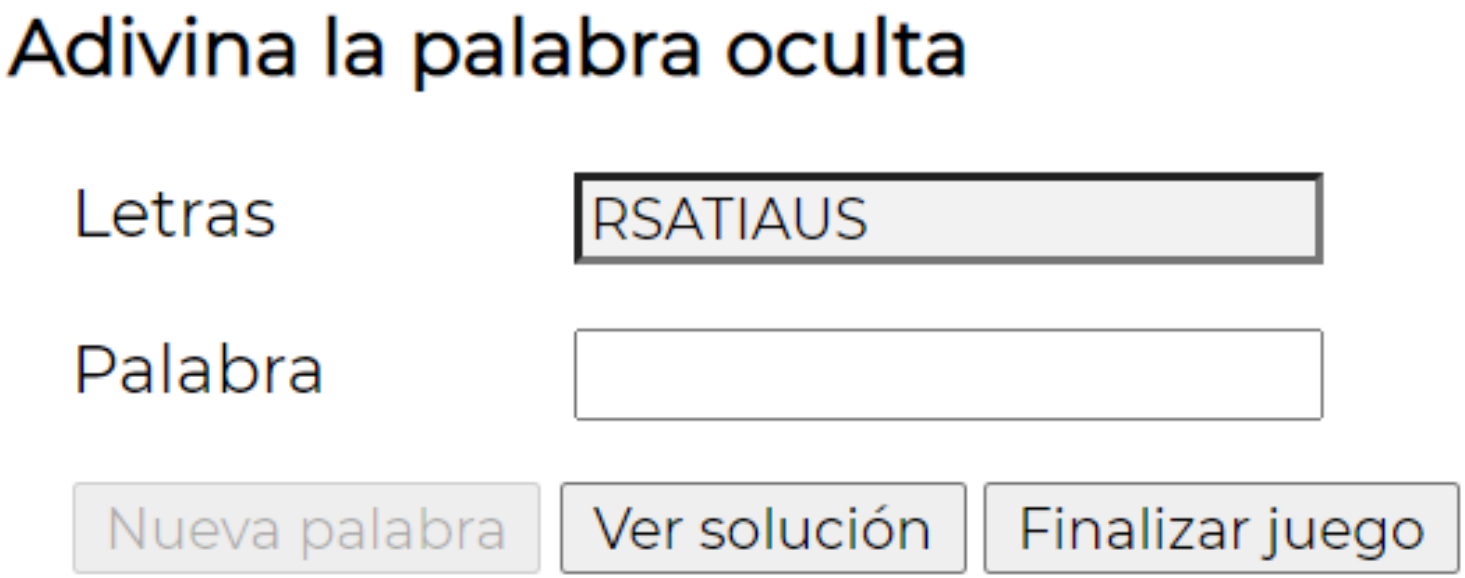
Ejemplo 6



- Crea un formulario HTML que tenga:
  - Una caja de texto
  - Un botón *Añadir.* Al pulsar el mismo, se añadirá a la página HTML un elemento de lista (<li>) cuyo contenido será el valor de la caja de texto
    - Si es la primera vez que se pulsa, deberá crearse el listado
  - Un botón *Quitar.* Al pulsar el mismo, se eliminará del listado **todos** aquellos valores cuyo texto coincida con el valor introducido en la caja de texto

UD1 ACT3: Palabra oculta





Valoración

Indice

Vanilla JS y frameworks

- Ya sabemos las bases de Vanilla JS
  - A excepción del uso de AJAX que es clave en aplicaciones actuales)
- ¿Qué aporta el uso de un framework como Angular?

JS sin framework = Vanilla JS

Ventajas de Vainilla JS

- Control completo sobre nuestro código
- Más ligero que cargar un framework completo
- Aprendizaje mas rápido
- Flexibilidad
  - Yo eligo qué librerías y bibliotecas usar
- Mejor rendimiento (*)

Ventajas un framework

- Proporciona una estructura y convenios, lo que facilita organizar el código y trabajar en equipo
- Funcionalides avanzadas que facilitan la solución a muchos problemas
- Desarrollo más rápido en aplicaciones complejas
- Mantenimiento y soporte más sencillo

Conclusiones

- Si estamos construyendo una aplicación Web más liviana o una página simple, Vanilla JS puede ser suficiente
- Si estamos trabajando en una aplicación Web más compleja, con un equipo grande o necesitamos características avanzadas, Angular puede ser una buena opción