UT5. Modelo de objetos del documento en JavaScript

1) Bases del Modelo de Objetos del Documento (DOM).
    1) Objetos del DOM HTML, propiedades y métodos.
    2) El árbol del DOM y tipos de nodos.
    3) Acceso a los nodos de tipo element.
    4) Modificación del contenido de nodos tipo element.
    5) Modificación del árbol DOM: insertar, añadir, borrar, reemplazar, clonar, crear nodos element
    6) Acceso a los nodos de tipo atributo.
    7) Acceso a los nodos de tipo texto.
    8) Propiedades y métodos de los objetos nodo.
2) Gestión de eventos: objeto event
    1) El objeto event.
    2) Propiedades y métodos del objeto Event.
    3) Eventos del teclado en JavaScript.
    4) Eventos del ratón en JavaScript.

1. Bases del Modelo de Objetos del Documento (DOM).

De la UT01→ Definición de DOM

DOM es una interfaz de programación de aplicaciones (**API) para documentos HTML**. Define la **estructura lógica** de los documentos y el **modo en que se accede y manipula un documento**.

Los lenguajes de programación acceden y manipulan los elementos HTML.

Para **evitar los problemas de falta de estandarización**, un organismo internacional (el W3C) definió este estándar **DOM** que **define qué elementos se considera que conforman una página web, cómo se nombran, cómo se relacionan entre sí, cómo se puede acceder a ellos, como se modifican, etc.**

1.1.Objetos del DOM HTML, propiedades y métodos.

Lista de objetos del DOM en HMTL (muchos de ellos ya los has manejado)

document HTMLElement anchor area base body button event form frame/iFrame Frameset image	input Button input Checkbox input File input Hidden Input Password Input Radio input File input Hidden Input Password Input Radio Input Reset Input Submit	Input Text Link Meta Object Option Select Style Table TableCell TableRow Textarea

1.2.- El árbol del DOM y tipos de nodos.

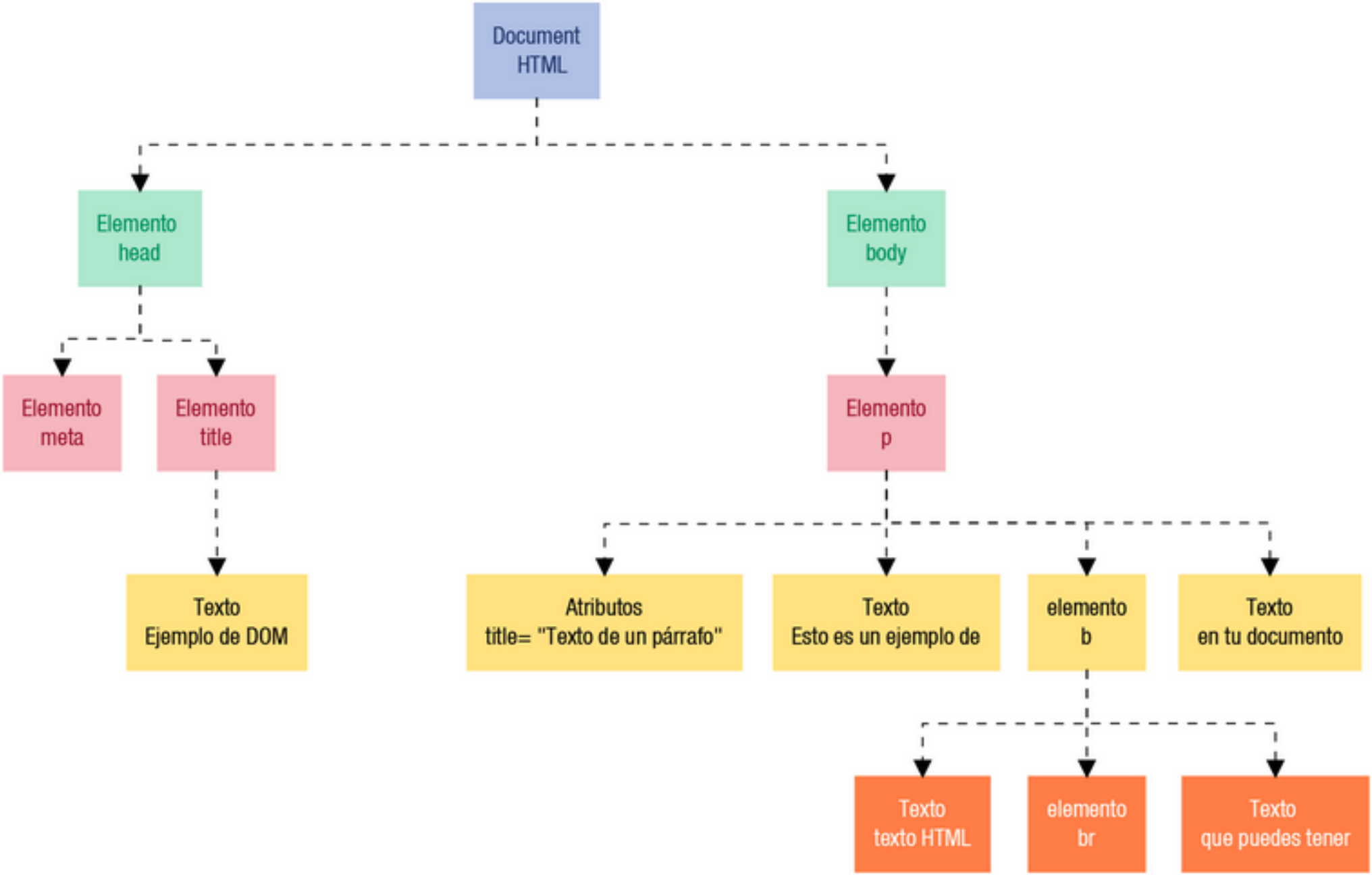
El DOM transforma **nuestro documento HTML en un conjunto de elementos, a los que llama nodos. Cada nodo es un objeto.**

Estos nodos están **conectados entre sí** en una estructura similar a un árbol, a lo que se llama árbol DOM o "**árbol de nodos**".

Ejemplo de documento HTML:

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />

<title>Ejemplo de DOM</title>

</head>

<body>

<p title="Texto de un párrafo">Esto es un ejemplo de <b>texto
HTML<br />que puedes tener</b> en tu documento.</p>

</body>

</html>
```

1.2.- El árbol del DOM y tipos de nodos.



1.2.- El árbol del DOM y tipos de nodos.

Cada **rectángulo** del gráfico representa un **nodo** del DOM.

Las líneas indican cómo se relacionan los nodos entre sí.

El árbol sigue una **estructura jerárquica**. El nodo inmediatamente superior será el nodo padre y todos los nodos que están por debajo serán nodos hijos.

La raíz del árbol de nodos es un nodo especial, denominado "**document**".

A partir de ese nodo, **cada etiqueta HTML se transformará en nodos de tipo "elemento" o "texto"**. Estos nodos se pueden crear con `createElement("p")` y `createTextNode()`

Los **nodos de tipo "texto"**, contendrán el **texto** encerrado **para esa etiqueta HTML**.

1.2.- El árbol del DOM y tipos de nodos.

Tipos de nodos:

La especificación del DOM define **12 tipos de nodos**, aunque generalmente emplearemos solamente **cuatro o cinco tipos de nodos**:

- **Document**, es el nodo raíz y del que derivan todos los demás nodos del árbol.
- **Element**, representa cada una de las etiquetas HTML. En nuestro ejemplo serían: body, head, meta, title, p, b, br. Es el único nodo que **puede contener atributos y el único del que pueden derivar otros nodos.**
- **Attr**, con este tipo de nodos **representamos los atributos de las etiquetas HTML**, es decir, un nodo por cada atributo=valor. En nuestro ejemplo de arbol sería atributo title del elemento p.
- **Text**, es el nodo que contiene el **texto encerrado por una etiqueta HTML**. En nuestro ejemplo de árbol serían todos los rectángulos llamados texto.
- **Comment**, representa los comentarios incluidos en la página HTML. No tenemos en nuestro ejemplo.

Los atributos de una etiqueta HTML aunque son un tipo de nodo, no forman parte del árbol de nodos, del arbol de nodos solo lo forman los elementos HTML (element) y los elementos texto (text)

Los otros tipos de nodos pueden ser: CdataSection, DocumentFragment, DocumentType, EntityReference, Entity, Notation y ProcessingInstruction.

1.3.- Acceso a los nodos element.

El acceder a un **nodo del árbol**, es lo equivalente a acceder a **un trozo de la página** de nuestro documento.

Una vez que hemos accedido a esa parte del documento, ya podemos modificar valores, crear y añadir nuevos elementos, moverlos de sitio, etc.

Para acceder a todos los nodos de un árbol, el árbol tiene que estar completamente construido, es decir, cuando **la página HTML haya sido cargada por completo**.

1.3.- Acceso a los nodos element.

Método `getElementById()`

Es el método más utilizado.

Nos **permite acceder directamente al elemento por el ID**. Entre paréntesis escribiremos la cadena de texto con el ID.

Es muy importante que el ID sea único para cada elemento de una misma página.

La función nos devolverá únicamente el nodo buscado. Por ejemplo:

```
var elemento= document.getElementById("apellidos");
```

Podemo accederemos a todas las celdas de la tabla con id=“datos” así:

```
var celdas=

document.getElementById("datos").getElementsByTagName("td")
;
```

1.3.- Acceso a los nodos element.

Método `getElementsByName()`

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

```
var elementos =
document.getElementsByName("apellidos");
```

Devuelve todos los elementos del documento cuyo nombre sea apellidos, devuelve una colección de elementos.

Se accede a cada uno de estos elementos **como si fuese un array. De hecho a menudo se le llama array, aunque no lo es** (no se puede utilizar pop y push).

Para acceder al **primer elemento** cuyo nombre sea apellidos haremos así:

```
document.getElementsByName("apellidos")[0];
```

En esta colección se puede utilizar el método **`forEach()`**

1.3.- Acceso a los nodos element.

Método `getElementsByName() - Ejercicio`

Crea una página HTML con un radio button con 3 opciones de deporte.

Desde Javascript haz que siempre esté seleccionada la última opción.

Desde Javascript recorre los elementos del radiobutton con el método getElementsByName y forEach() para ver cuál de ellos tiene la propiedad checked a true. El elemento seleccionado será el que tenga dicha propiedad a true.

1.3.- Acceso a los nodos element.

Método `getElementsByTagName()`

```
var elementos = document.getElementsByTagName("input");

// Este array de elementos contendrá todos los elementos
input del documento.
```

Devuelve todos los elementos input del documento, también en una colección de elementos.

Para acceder al **cuarto elemento input** haremos así:

```
var cuarto =
document.getElementsByTagName("input")[3];
```

La colección que devuelve este método no permite utilizar el método **`forEach()`** Para ello habría que pasar la colección a Array así:

```
Array.from(elementos).forEach()
```

1.3.- Acceso a los nodos element.

Método `getElementsByClassName()`

Muy útil para seleccionar elementos que tengan definida determinada clase CSS:

```
var elementos =
document.getElementsByClassName("importante");
```

La colección que devuelve este método no permite utilizar el método **`forEach()`** Para ello habría que pasar la colección a Array así:

```
Array.from(elementos).forEach()
```

Ejercicio:

Crea una página HTML con:

- **Una clase de estilos llamada “importante” que ponga color a red**
- **5 elementos div. 2 de ellos tendrán la clase “importante”.**

Desde Javascript calcula el número de elementos con la clase “importante”

1.3.- Acceso a los nodos element.

Métodos `querySelector("CSS selector")` y `querySelectorAll("CSS selector")`

En CSS los **selectores** se utilizan para seleccionar los elementos HTML de nuestra página web a los que queremos aplicar estilo. Estos CSS selectores se pueden utilizar en estos métodos para indicar qué elementos HTML queremos seleccionar.

```
<style>
.importante {    // CSS Selector es una clase. A los elementos que
                 // tengan como atributo "class=importante" se les
                 // aplica el estilo color: blue
  color: blue;
}

p,h1 {           // CSS Selector son elementos. A los elementos
                 // párrafo y encabezado 1 se les aplica este estilo
color : blue
}

#demo {      // CSS Selector es el id de los elementos. A los elementos
             //con identificado "demo" se les aplica el estilo color: blue
color : blue
}
</style>
```

1.3.- Acceso a los nodos element.

Métodos `querySelector("CSS selector")` y `querySelectorAll("CSS selector")`

El primero devuelve el primer elemento que cumple con el CSS selector.

El segundo devuelve todos los nodos que cumplen con el CSS Selector en una colección. En esta colección se puede utilizar el método **`forEach()`**

```
var elementos = document.querySelector(".importante");
//devuelve el primer elemento del documento con clase importante

var elementos = document.querySelectorAll("#nombre");
//devuelve una colección con todos los elementos con id nombre

var elementos = document.querySelectorAll("p");

//devuelve una colección con todos los elementos párrafo
```

Continuación del Ejercicio anterior (de 5 div):

- **Al 5º div ponle como identificador "5div".**
- **Obtén: el nº de elementos con clase "importante", el primer elemento con clase "importante", el nº de div, el elmento con identificador "5div".**

1.3.- Acceso a los nodos element.

Para acceder a un nodo específico (elemento HTML) lo podemos hacer empleando de **dos formas**:

- **Método de acceso directo**.

```
document.getElementById("c_madrid") o

c_madrid
```

- **A través de los nodos padre:**

```
document.getElementById("n_form").getElementById(
"c_madrid") /*n_form es el formulario
```

Más breve:

```
n_form.c_madrid
```

Modificación del contenido de un element

`element.innerHTML:` para modificar el contenido del elemento (nodo texto hijo), indicando al navegador que el texto lo interprete como código HTML. No modifica el propio elemento HTML

`element.outerHTML:` para modificar el contenido del elemento (nodo texto hijo), indicando al navegador que el texto lo interprete como código HTML, y también modifica el elemento HTML.

`element.innerText:` para modificar el contenido del elemento (nodo texto hijo), indicando al navegador que el texto lo interprete como texto plano.

Modificación del contenido de un nodo element

Ejercicio:

Tenemos una página HTML con 4 párrafos, cada uno de ellos con un id.

- Modifica el primero para que aparezca el texto "**Sergio Sánchez** es estupendo"
- Modifica el segundo párrafo para que sea sustituido por la siguiente lista (comprueba que el elemento párrafo desaparece):
  - **Adriana**
  - **Elena**
- Modifica el tercer párrafo dando el valor "Hola" a la propiedad innerText. ¿Qué ocurre?, ¿por qué?
- Modifica el primer párrafo para que aparezca una línea más en dicho párrafo con "**Adrián** es fantástico"
- Modifica el cuarto párrafo para que aparezca lo mismo que en el primero pero sin las negritas utilizando la propiedad innerText.

Acceso a los nodos element-continuación

`elem.children:` devuelve una colección con todos los nodos hijos de tipo element de un elemento (`elem`). El orden es el de aparición en la página HTML

En esta página HTML de ejemplo:

```
<div>
<p>Primera línea</p>
<p>Segunda línea</p>
<p>Tercera línea</p>
</div>
var elemDiv=document.getElementsByTagName("div")[0]
elemDiv.children[0]// es el párrafo cuyo contenido es "Primera línea"
elemDiv.children[1]// es el párrafo cuyo contenido es "Segunda línea"
elemDiv.children[elemDiv.children.length-1]// es "Tercera línea"
```

La colección que devuelve este método no permite utilizar el método **`forEach()`** Para ello habría que pasar la colección a Array así:

```
Array.from(elem.children).forEach()
```

`elem.childElementCount:` devuelve el nº de nodos hijos de tipo element de un elemento (`elem`). Equivalente a: **`elem.children.length`**

```
elemDiv.children[elemDiv.childElementCount-1]// es "Tercera línea"
```

Acceso a los nodos element.

`elem.firstElementChild:` devuelve el primer nodo hijo de tipo element de un elemento (`elem`). El orden es el de aparición en la página HTML

En esta página HTML de ejemplo:

```
<div>
<p>Primera línea</p>
<p>Segunda línea</p>
<p>Tercera línea</p>
</div>
var elemDiv= document.getElementsByTagName("div")[0]
elemDiv.firstElementChild // es el de contenido "Primera línea"
//igual elemDiv.children[0]
```

`elem.lastElementChild:` devuelve el último nodo hijo de tipo element de un elemento (`elem`)

```
elemDiv.lastElementChild // es el de contenido "Tercera línea"
// igual a elemDiv.children[elemDiv.childElementCount-1]
```

`elem.parentNode:` es el nodo padre del elemento `elem`.

```
elemDiv.firstElementChild.parentNode // es elemDiv
```

Acceso a los nodos element.

`elem.nextElementSibling`: devuelve el siguiente nodo de tipo element a `elem` que hay en el mismo nivel del árbol DOM. El orden es el de aparición en la pág. HTML

En esta página HTML de ejemplo:

```
<div>
<p>Primera línea</p>
<p>Segunda línea</p>
<p>Tercera línea</p>
</div>
var elemDiv= document.getElementsByTagName("div")[0]

elemDiv.firstElementChild.nextElementSibling()
// es el párrafo de contenido "Segunda línea"
```

`elem.previousElementSibling`: devuelve el nodo de tipo element anterior a `elem` que hay en el mismo nivel del árbol DOM. El orden es el de aparición en la página HTML

```
elemDiv.lastElementChild.previousElementSibling() //el de "Segunda línea"
elemDiv.firstElementChild.previousElementSibling() //es null
```

Ejercicio-Acceso a los nodos element.

Dada esta tabla:

```
 <table style="width:300px">
   <tbody>
   <tr>
     <td><button style="font-size:26px;width:80px">1</button></td>
     <td><button style="font-size:26px;width:80px">2</button></td>
     <td><button style="font-size:26px;width:80px">3</button></td>
   </tr>
   <tr>
     <td><button style="font-size:26px;width:80px" >4</button></td>
     <td><button style="font-size:26px;width:80px" >5</button></td>
     <td><button style="font-size:26px;width:80px">6</button></td>
   </tr>
   <tr>
     <td><button style="font-size:26px;width:80px">7</button></td>
     <td><button style="font-size:26px;width:80px">8</button></td>
     <td><button style="font-size:26px;width:80px">9</button></td>
   </tr>

</tbody>
   </table>
```

Ejercicio-Acceso a los nodos element-cont.

Crea los siguientes botones:

```
<input type="button" value="primer boton rojo" onclick="primero()">

<input type="button" value="siguiente boton al primero verde"
onclick="segundo()"><br>

<input type="button" value="ultimo boton azul" onclick="ultimo()">

<input type="button" value="anterior boton al ultimo gris"
onclick="penultimo()"><br>
```

Codifica las funciones de la siguiente manera:

`primero()`: ponga el primer botón de la tabla en rojo como color de fondo.

`segundo()`: ponga el siguiente botón al primero de la tabla en verde como color de fondo.

`ultimo()`: ponga el ultimo botón de la tabla en azul como color de fondo.

`penultimo()`: ponga el botón anterior al ultimo de la tabla en gris como color de fondo.

Modificación del árbol DOM

- **`cloneNode(true)` copia un nodo** ya existente, **pero no lo añade automáticamente a ningún nodo. Con true copia también todos sus nodos hijos si los tuviera.**

```
var nodoNuevo=nodo.cloneNode(true);
```

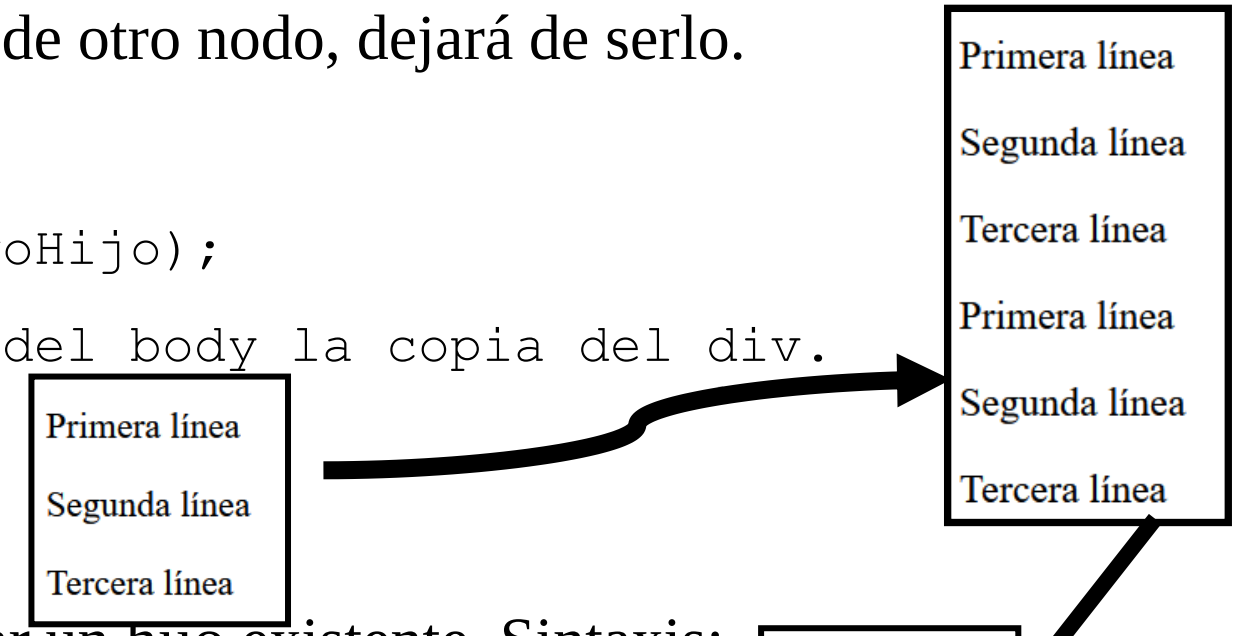
Ejercicio: crea una copia del primer párrafo del div.

- **`appenChild()`** Para insertar un hijo después del último nodo hijo. Si el hijo insertado era hijo previamente de otro nodo, dejará de serlo.

Sintaxis:

```
nodoPadre.appendChild(nuevoHijo);
```

Ejercicio: añade al final del body la copia del div.



- **`removeChild()`** para borrar un hijo existente. Sintaxis:

```
nodoPadre.removeChild(nodoHijo).
```

Ejercicio: borra la copia del div.

Modificación del árbol DOM

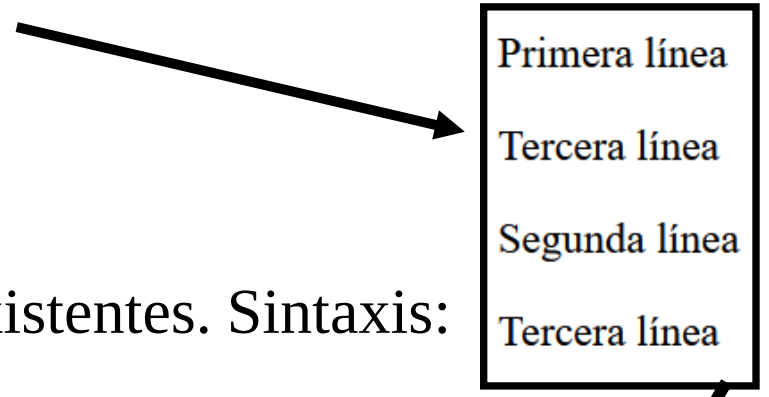
- **`insertBefore()`** Para insertar un hijo antes de otro nodo hijo. Sintaxis:

```
nodoPadre.insertBefore(nuevoHijo,refHijo);
```

Va a añadir al nodo padre un hijo delante del nodo hijo refHijo

Ejemplo: https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_node_insertbefore

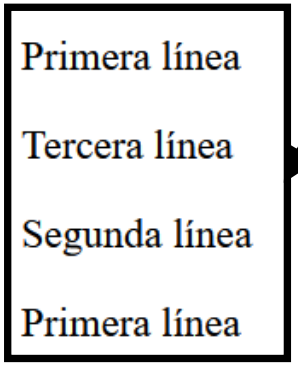
Ejercicio: crea una copia del ultimo párrafo del div e insértalo antes del segundo párrafo del div.



- **`replaceChild()`** **sobrescribe nodos** ya existentes. Sintaxis:

```
nodoPadre.replaceChild(nuevoHijo,antiguoHijo);
```

Ejercicio: crea una copia del primer párrafo del div y reemplaza el ultimo párrafo por esta copia.



Ejercicio-Modificación del árbol DOM.

En la tabla anterior añade los siguientes botones:

```
<input type="button" value="subir" onclick="subir()">
<input type="button" value="bajar" onclick="bajar()">
<input type="button" value="izquierda" onclick="izquierda()">
<input type="button" value="derecha" onclick="derecha()">
<input type="button" value="aleatorio" onclick="aleatorio()">
```

Codifica las funciones de la siguiente manera (Importante: No se puede modificar el atributo value de los botones**):**

`subir():` sube todas las filas de la tabla un nivel más arriba de forma circular, es decir, la primera fila será la última después de subir.

`bajar():` baja todas las filas de la tabla un nivel más arriba de forma circular

`izquierda():` desplaza todos los botones una posición a la izquierda de forma circular.

`derecha():` desplaza todos los botones una posición a la derecha de forma circular.

`aleatorio():` desordena todos los botones de forma aleatoria.

Creado de nodos element.

Los métodos **`createElement()` nos permite crear un elemento.** El nuevo elemento se puede añadir a la página HTML con **`appenchild()`, `replaceChild()`,`insertBefore()`**

```
<p id="parrafo">Esto es un ejemplo</p>

var nuevoParrafo = document.createElement('p');
nuevoParrafo.id=”elnuevo”;
nuevoParrafo.innerHTML='Contenido añadido.’;

parrafo.appendChild(nuevoParrafo);
```

El resultado será (se añade el nuevo párrafo como un hijo del existente)**:**

```
<p id="parrafo">Esto es un ejemplo<p
id=”elnuevo”>Contenido añadido.</p></p>
```

1.6.- Creado de nodos de tipo element

Ejemplo: creamos una tabla en div

```
var division= document.getElementsByTagName(“div”)[0];

var tabla= document.createElement('table');
tabla.style.border="2px solid";

var titulo=document.createElement("caption");
titulo.innerHTML="El titulo";
tabla.appendChild(titulo);

var fila1=document.createElement("tr");
tabla.appendChild(fila1);

var celda1=document.createElement("td");
celda1.innerHTML="Celda 1";
celda1.style.border="2px solid";
fila1.appendChild(celda1);

var celda2=document.createElement("td");
celda2.innerHTML="Celda 2";
celda2.style.border="2px solid";
fila1.appendChild(celda2);

division.appendChild(tabla);
```

1.6.- Creado de nodos de tipo element

Ejercicio:

Crea una aplicación web con:

1. Un input de tipo number llamado “columnas”.
2. Un input de tipo number llamado “filas”.
3. Un botón llamado “Crear tabla” que cree una tabla con el número de filas y el número de columnas indicados en los inputs anteriores

1.4.- Acceso a los nodos de tipo atributo.

Para referenciar a los atributos de los elementos, como por ejemplo el atributo type="text" del elemento “nombre”, emplearemos **la propiedad `attributes`:**

```
<input type="text" size="40“ id=“nombre”>
var atributos= document.getElementById(“nombre”).attributes
```

`attributes` es una colección.

Cada elemento de la colección `attributes` es **un atributo del elemento** HTML, que, a su vez, **tiene dos propiedades:**

- **`nodeName` o `name`: nombre** del atributo (`”type”,“size”,“id”`)
- **`nodeValue` o `value`: el valor** del atributo(`”text”,”40”,”nombre”`)

```
atributos[0].nodeName //vale “type”, como atributos[0].name
atributos[0].nodeValue //vale “text”, como atributos[0].value
```

La colección que devuelve este método no permite utilizar el método **`forEach()`** Para ello habría que pasar la colección a Array así: **`Array.from(elementos).forEach()`**

1.4.- Acceso a los nodos de tipo atributo.

Para modificar un atributo o crear un nuevo atributo dentro de esta colección de atributos (attributes) se emplea el método:

`setAttribute(nodeName,nodeValue);`

```
document.getElementById("nombre").setAttribute("size"
,"40");
```

Para obtener el valor de un atributo dentro de esta colección de atributos (attributes) se emplea el método:

`getAttribute("nodeName");`

```
var valor= document.getElementById("nombre").getAttribute("type");
// devuelve "text"
```

1.4.- Acceso a los nodos de tipo atributo.

Para eliminar un atributo dentro de esta colección de atributos (attributes) se emplea el método:

`removeAttribute("nodeName");`

```
document.getElementById("nombre").removeAttribute("placeholder");

//se eliminará el placeholder del input
```

Otros métodos y propiedades de los nodos

`nodo.childNodes`: propiedad que es una colección con los nodos hijos (tipo element y text) de `nodo`. `nodo.children` es una colección de nodos hijos tipo `element`.

`nodo.hasChildNodes()`: devuelve `true` si `nodo` tiene nodos hijos (tipo element o text), y `false` si no tiene.

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

demo.hasChildNodes() //devuelve false
demo.innerHTML="contenido"  // Queda así: <p id="demo">contenido</p>
demo.hasChildNodes() //devuelve true porque ya tiene un nodo hijo tipo
text
```

`nodo.nodoType`: tiene valor:

- 1 si el nodo es de tipo `Element`
- 2 si el nodo es de tipo `atributo`.
- 3 si el nodo es de tipo `text`.

```
demo.nodeType // es el nodo párrafo, su tipo de nodo vale 1
// es el atributo id del párrafo que es un nodo cuyo tipo vale 2
demo.attributes[0].nodeType
// es el nodo hijo (del párrafo) de tipo texto ("contenido") y el tipo es 3.
demo.childNodes[0].nodeType
```

Modificación de los estilos de los elementos del DOM- objeto `style`

`nodo.style:` objeto para manejar los estilos del elemento `nodo` de forma "inline"

```
<p id="demo">Contenido</p>
demo.style.color="red" //Pone Contenido en rojo
```

Con ello se está aplicando un estilo de forma inline al párrafo. Si se observa el inspector se verá el párrafo así:

```
<p id="demo" style="color: red">Contenido</p>
```

Modificación de los estilos de los elementos del DOM- propiedad `classList`

`nodo.classList.add("clase1","clase2",...)` : método `add()` para añadir una clase de estilos o varias, definida/s en una hoja de estilos, al elemento `nodo`.

`nodo.classList.remove("clase1","clase2",...)`: método `remove()` para quitar una clase de estilos o varias, definida/s en una hoja de estilos, al elemento `nodo`.

`nodo.classList.toggle("clase1")`: método `toggle()` para quitar una clase de estilos, definida en una hoja de estilos, al elemento `nodo`. si la tiene, y si no la tiene se la añade

`nodo.classList.contains("clase1")`: método `contains()` devuelve `true` si `nodo` tiene la clase `"clase1"` y `false` si no la tiene.

Ejercicio: utilizando la tabla de números anterior haz lo siguiente:

- Elimina los estilos inline que tienen los botones en la parte HTML.
- Crea la clase de estilos `boton` con `font-size: 26px; width: 80px` con la etiqueta `<style>`
- Aplica esta clase `boton` a los botones desde la parte HTML.
- Crea la clase de estilos `azul` con `background-color: cyan` con la etiqueta `<style>`
- Desde Javascript programa un botón que aplique y desaplique la clase de estilos `azul` a los botones pares utilizando `classList.` Observa cómo quedan los estilos desde el inspector.
- Desde Javascript programa un botón que pona el fondo de color verde a los botones con números impares utilizando `style`. Observa cómo quedan los estilos desde el inspector.

Eventos. Introducción

Eventos son acciones que se producen en nuestra aplicación: ya sean generadas directamente por el usuario o por nuestra aplicación. Los eventos se pueden dividir en categorías:

- **Eventos producidos por el ratón**: click, dblclick, mouseenter, mouseleave, mousedown, mouseup, mousemove, drag, dragstart, dragend, dragenter, dragleave, dragover, drop, …
- **Eventos producidos por el teclado**: keydown/keypress, keyup.
- **Eventos producidos en los elementos de un formulario**: focus, blur, submit, reset, change, input, select, …
- **Eventos relacionados con la página**: load, unload, resize, …
- **Eventos de bases de datos**: abort, blocked, complete, error, success, upgradeneeded, versionchange, …
- **Eventos de peticiones asíncronas**: readystatechange, …
- **Eventos de conexión de red**: disabled, enabled, offline, online, statuschange
- **Eventos de cambios del DOM**.
- **Eventos de medios**: canplay, durationchange, pause, play, ..

Eventos. Introducción

- **Eventos de animaciones**: animationend, animationiteration, animationstart..
- **Eventos de impression:** afterprint, beforeprint, …
- **Eventos de clipboard:** copy, paste, cut, select, …

JavaScript permite programar que se ejecute una tarea cuando se produzca un evento. Es lo que se conoce en programación como **capturar un evento**. Es decir, se indica que cuando se produzca un evento se ejecute una función.

Acción por defecto de la acción

Un evento puede generar la ejecución de una función si así se ha programado (captura del evento). Y además, ese mismo evento puede generar la ejecución de una tarea propia de ese evento, es a lo que se llama acción por defecto.

Por ejemplo cuando en un enlace se hace click se carga la página a la que apunta dicho enlace. Esto es una acción por defecto del click en un enlace. Además, si está programado que cuando se haga click en el enlace se ejecute una función (por la captura del evento click), ocurrirán las dos funciones así:

1)Se ejecutará la función asociada al evento.
2)La acción por defecto se ejecutará después (*cargarse la página a la que apunta el enlace*)

2.1. Modelo de registro de eventos en línea

La captura del evento puede ser incluido como un atributo más a la etiqueta HTML, como en este ejemplo:

```
<a id=”enlace” href="pagina.html" onclick="alertar()">Pulsa aqui</a>

function alertar() {

        alert("Has pulsado en el enlace");

}
```

No se recomienda utilizar el modelo de registro de eventos en línea. Ya que lo que se recomienda es separar completamente la programación JavaScript de la estructura HTML.

Modo de funcionamiento de los eventos

El orden de ejecución es el siguiente:

1.El script se ejecutará primero (función *`alertar()`*)

2.La acción por defecto se ejecutará después (*cargarse la página* *`pagina.html`*)

Ejercicio 10

2.1. Modelo de registro de eventos en línea

Evitar la acción por defecto desde el código HTML

Desde el código HTML se puede evitar la ejecución de la acción por defecto. En este caso sería la carga de la página HTML.

```
<a id=”enlace” href="pagina.html" onclick="alertar(); return
false">Pulsa aqui</a>
```

Con el return false estamos diciendo al navegador que no realice la acción del objeto <a>, que es cargar la página pagina.html

Ejercicio 11

Evitar la acción por defecto desde el código JavaScript

```
<a id=”enlace” href="pagina.html" onclick="return
preguntar()">Pulsa aqui</a>

function preguntar() {
     return confirm("¿Desea realmente ir a esa dirección?");
}
```

Ejercicio 12 y 13

2.2. Modelo de registro de eventos tradicional

El modelo de registro de eventos tradicional se **basa en la separación del código JavaScript de la estructura HTML.**

El evento pasa a ser una propiedad del elemento, y puede ser manejado por código JavaScript.

No está admitido por W3C, pero debido a que fue ampliamente utilizada por Netscape y Microsoft, todavía es válida hoy en día.

Una forma de hacerlo sería así:

```
<a id=”enlace” href="pagina.html">Pulsa aqui</a>

<script>
  enlace.onclick=funcion;
  function funcion () {
     alert(“Vas a acceder a una pagina web”);
  }
</script>
```

Nota importante → si se pusiera: `enlace.onclick=funcion();`

Se ejecutaría `funcion()` y el resultado se asignaría al atributo `onclick` de enlace

Ejercicio 17

Para eliminar un gestor de eventos de un elemento u objeto: `elemento.onclick=null;`

En nuestro ejemplo sería: `enlace.onclick=null;`

2.2. Modelo de registro de eventos tradicional

Además de la ventaja de separación de código, nos permite lanzar el evento de forma manual. Por ejemplo:

```
elemento.onclick();
```

Utilización del objeto this en modelo de registro de eventos tradicional:

```
<a id=”enlace” href="pagina.html">Pulsa aqui</a>

<script>
  enlace.onclick=funcion;

  function funcion () {
     alert(“Vas a acceder a la pagina web” + this.href);
  }
//this es el elemento que tiene programado el evento

</script>
```

2.3. Modelo de registro avanzado de eventos W3C

Dado que W3C no estandarizó el modelo tradicional ofreció como alternativa el modelo de registro avanzado de eventos, con el método: `addEventListener()`

Este método tiene **tres argumentos: el tipo de evento, la función a ejecutar y un valor booleano (true o false).** Este valor booleano se utiliza para indicar cuando se debe capturar el evento: en la fase de captura (true) o de burbujeo (false) (se verá más adelante la diferencia entre las dos fases)

```
elemento.addEventListener('evento', función, false|true)
```

Por ejemplo:


```
enlace.addEventListener('click', alertar, false)
function alertar() {
     alert("Te conectaremos con la página que es un poco pesada:
"+this.href);
}  //this es el objeto enlace que es quien tiene programado el evento
```

Podemos añadir varias funciones para el mismo evento:

La ventaja de este método, es que podemos añadir tantos eventos como queramos. Por ejemplo:

```
enlace.addEventListener('click',alertar,false);

enlace.addEventListener('click',avisar,false);

enlace.addEventListener('click',chequear,false);
```

Por lo tanto, cuando hagamos clic en “enlace” se disparará la llamada a las tres funciones.

2.3. Modelo de registro avanzado de eventos W3C

¿Qué eventos han sido registrados?

Si se visualiza el atributo correspondiente al evento, se verá que es null:

```
enlace.onclick→ es null
```

Esto es así, porque el método addEventListener() no modifica este atributo.

Para visualizar los eventos asociados a un objeto HTML se puede utilizar el método:

`eventListenerList`

NOTA: no es ampliamente utilizado.

¿Eliminar un evento de un elemento?

Para eliminar un evento asociado a un objeto HTML se puede utilizar el método :

```
elemento.removeEventListener('evento', función, false|true);
```

Ejemplo:

```
enlace.removeEventListener(“click”, alertar, false);
```

Ejercicio 23

2.3. Modelo de registro avanzado de eventos W3C

También se pueden usar funciones anónimas (sin nombre de función), con el modelo W3C:

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

Nota → el uso de this en modelo de registro avanzado de eventos es igual al del modelo de registro tradicional de eventos.

Función flecha en registro de eventos

- **IMPORTANTE: No se debe utilizar en la función manejadora de un evento: `this` será `window` en lugar del objeto que tiene definida la función manejadora.**

```
b.addEventListener("click",function (){console.log(this)})
// cuando se hace click en el elemento b sale el elemento b en consola.


b.addEventListener("click", ()=>{console.log(this)})
// cuando se hace click en el elemento b sale Window en consola.
```

2.3. Modelo de registro avanzado de eventos W3C

Para anular la ejecución del evento por defecto de un elemento

Para evitar que se ejecute el evento por defecto de un elemento según el modelo de registro avanzado de eventos se utiliza el método **`event.preventDefault();`**

`event` es un objeto que guarda toda la información del evento que se ha producido en el elemento.
`preventDefault` es un método del objeto **`event.`**

Hay eventos que no permiten ejecutar `preventDefault()` , por ejemplo `load`. Los eventos tienen una propiedad de solo lectura llamada **`cancelable`** que indica si la acción por defecto se puede cancelar o no.

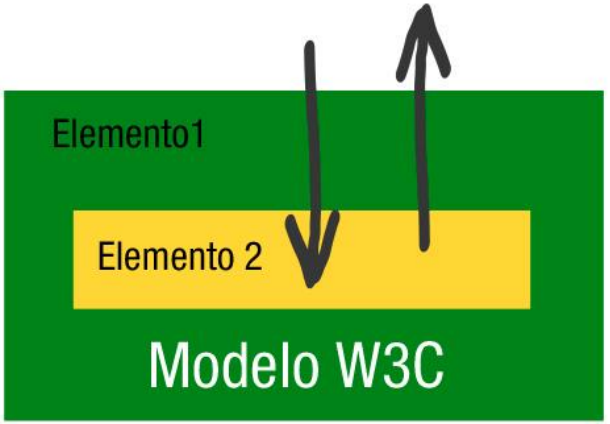
En este ejemplo se hace así (en el caso de los enlaces el evento por defecto es abrir la página indicada en href):

```
<a id="enlace" href="https://w3schools.com/">Enlace</a>

enlace.addEventListener("click", anularAccionDefecto);

function anularAccionDefecto(event){

    event.preventDefault();

}
```

Ejercicios 24 y 25

2.4. Orden de disparo de los eventos

Si tenemos dos elementos, uno contenido dentro de otro, y tenemos programado el mismo tipo de evento para ambos. Al producirse el evento en el de dentro, se va a producir también un evento en el de fuera, ¿cuál de los eventos se disparará primero?



Según el modelo W3C

Si se produce un evento en Elemento 2 también se está produciendo en Elemento 1. El modelo W3C divide el proceso de disparo en dos fases:

- **La fase captura (de fuera a dentro):** que empieza en Elemento 1 y termina en Elemento 2.
- **La fase de burbujeo (de dentro a fuera):** que empieza en Elemento 2 y termina en Elemento 1.

Tú podrás decidir cuando quieres que se ejecute la función manejadora del evento: en la fase de captura o en la fase de burbujeo. **El tercer parámetro de addEventListener te permitirá indicar si lo haces en la fase de captura (`true`), o en la fase de burbujeo(`false`).**

2.4. Orden de disparo de los eventos

Por ejemplo:

```
elemento1.addEventListener('click',hacerAlgo1,false);

elemento2.addEventListener('click',hacerAlgo2,false);
```

Si el usuario hace clic en el elemento2 se producirá el evento click de elemento2 y de elemento 1. El elemento destino es elemento2, que es en el que se ha originado el evento (el más interno).

El orden de disparo será el siguiente:

- El evento de clic comenzará en la fase de captura (de fuera al elemento destino)
  - El evento comprueba si hay algún ancestro del elemento2 que tenga el evento capturado para la fase de captura (true). El evento encuentra un elemento1 que tiene capturado el evento click para que se ejecute hacerAlgo1() en la fase de burbujeo por lo que no hace nada.
  - El evento viajará hacia el destino (elemento2) que tiene capturado el evento click para que se ejecute hacerAlgo2() en la fase de burbujeo, por lo que no hace nada.
- Entonces el evento pasa a la fase de burbujeo (del elemento destino hacia fuera)
  - El evento busca si está capturado en elemento2 el evento en fase de burbujeo y ve que sí, por lo que ejecuta su manejador que es hacerAlgo2().
  - El evento viaja hacia arriba de nuevo y chequea si algún ancestro (Elemento1) tiene programado este evento para la fase de burbujeo, ve que es así y ejecuta su manejador que es hacerAlgo1().

2.4. Orden de disparo de los eventos

Por ejemplo:

```
elemento1.addEventListener('click',hacerAlgo1,true);

elemento2.addEventListener('click',hacerAlgo2,true);
```

Si el usuario hace clic en el elemento2 se producirá el evento click de elemento2 y de elemento 1. El elemento destino es elemento2, que es en el que se ha originado el evento (el más interno)

El orden de disparo será el siguiente:

- El evento de clic comenzará en la fase de captura (de fuera al elemento destino)
  - El evento comprueba si hay algún ancestro del elemento2 que tenga el evento capturado para la fase de captura (true). El evento encuentra un elemento1 que tiene capturado el evento click para que se ejecute hacerAlgo1() en la fase de captura por lo que ejecuta hacerAlgo1()
  - El evento viajará hacia el destino (elemento2) que tiene capturado el evento click para que se ejecute hacerAlgo2() en la fase de captura, por lo que se ejecuta hacerAlgo2()
- Entonces el evento pasa a la fase de burbujeo (del elemento destino hacia fuera)
  - El evento busca si está capturado en elemento2 el evento en fase de burbujeo y ve que no, por lo que no hace nada
  - El evento viaja hacia arriba de nuevo y chequea si algún ancestro (Elemento1) tiene programado este evento para la fase de burbujeo, ve que no por lo que no hace nada.

Ejercicios 26

3 – Gestión de eventos: el objeto event

Generalmente, **los manejadores de eventos** (las funciones que se ejecutan cuando se produce un evento) **necesitan información adicional para procesar las tareas que tienen que realizar.**

Si una función procesa, por ejemplo, el evento `keypress`, lo más probable es que necesite conocer qué tecla ha sido pulsada.

Para gestionar esta información tenemos el objeto `event`.

Este **objeto `event` guarda toda la información del evento** que se ha producido.

El objeto `event` es utilizado dentro de la función que se dispara cuando se produce el evento.

3.1 Event: Cómo se accede al objeto event

Si la función manejadora está definida con un argumento de entrada, en este argumento es donde se almacenará el objeto event.

```
elemento.addEventListener('keypress',gestionar,false);

function gestionar (mievento) {

   …..

   mievento.preventDefault();

}
```

Si la función manejadora no tiene un argumento de entrada se puede utilizar dentro de ella directamente el objeto event referenciándolo así, event.

```
elemento.addEventListener('keypress',gestionar,false);

function gestionar () {

   …..

   event.preventDefault();

}
```

3.2 Propiedades del objeto event

Propiedades	Descripción
`altKey, ctrlKey, metaKey, shiftKey`	Valor booleano que indica si están presionadas alguna de las teclas `Alt`, `Ctrl`, `Meta` o `Shift` en el momento del evento.
`bubbles`	Valor booleano que indica si el evento burbujea o no.
`button`	Valor `integer` que indica que botón del ratón ha sido presionado o soltado, 0=izquierdo, 2=derecho, 1=medio.
`cancelable`	Valor booleano que indica si el evento se puede cancelar.
`charCode`	Indica el carácter Unicode de la tecla presionada.
`clientX, clientY`	Devuelve las coordenadas de la posición del ratón en el momento del evento.

Se recomienda utilizar la propiedad `key` en lugar de `charCode` o `keyCode` ya que están obsoletas, y en lugar de `which` que no es estándard

`altKey, ctrlKey, metaKey, shiftKey`→ pueden modificar el comportamiento de un evento en función de si estas teclas están pulsadas o no, por ejemplo un `click` de ratón + `Ctrl` se comporta diferente a `click` solo

La tecla especial Meta no la tienen todos los teclados.

En teclados Sun Microsystems está representado por "◆" .

En teclados Mac está representado por ("⌘") "Command/Cmd"

Recuerda que la propiedad `cancelable` es de lectura, solo se puede consultar, no modificar. Indica si el evento permite que la acción por defecto se pueda cancelar o no con `preventDefault();`

3.2. Propiedades del objeto event

currentTarget	El elemento al que se asignó el evento. Por ejemplo si tenemos un evento de click en un divA que contiene un hijo divB. Si hacemos click en divB, currentTarget referenciará a divA (el elemento dónde se asignó el evento) mientras que target devolverá divB, el elemento dónde ocurrió el evento.
eventPhase	Un valor integer que indica la fase del evento que está siendo procesada. Fase de captura (1), en destino (2) o fase de burbujeo (3).
layerX, layerY	Devuelve las coordenadas del ratón relativas a un elemento posicionado absoluta o relativamente. Si el evento ocurre fuera de un elemento posicionado se usará la esquina superior izquierda del documento.
pageX, pageY	Devuelve las coordenadas del ratón relativas a la esquina superior izquierda de una página.
relatedTarget	En un evento de "mouseover" indica el nodo que ha abandonado el ratón. En un evento de "mouseout" indica el nodo hacia el que se ha movido el ratón.
screenX, screenY	Devuelve las coordenadas del ratón relativas a la pantalla dónde se disparó el evento.
target	El elemento dónde se originó el evento, que puede diferir del elemento que tenga asignado el evento. Véase currentTarget.
timestamp	Devuelve la hora (en milisegundos desde epoch) a la que se creó el evento. Por ejemplo cuando se presionó una tecla. No todos los eventos devuelven timestamp.
type	Una cadena de texto que indica el tipo de evento "click", "mouseout", "mouseover", etc.

Ejercicio 31

`currentTarget` es `this`, es el elemento que tiene capturado el evento, es decir, el que tiene programado que se ejecute una función cuando se produzca un evento.
`target` hace referencia al elemento en el que se ha producido el evento. Ejemplo: si `body` ha capturado el evento `click`, y hago `click` en un párrafo del `body`: `target` es el `párrafo` y `currentTarget` es el `body`. El concepto de elemento destino se utiliza para el orden de disparo de funciones manejadoras, cuando tenemos varios elementos anidados con el mismo evento registrado. El elemento destino es en el que se ha originado el evento, el más interno de todos los elementos involucrados.

3.2 Propiedades currentTarget, target

Ejemplo:

```
<!DOCTYPE html> // PROBAR pinchando en Hazme click(p), justo debajo(
es decir en el body pero no en el párrafo) y al final pag (ni en el
párrafo ni en el body).

<html>
<body id="cuerpo">
<p id="demo">Hazme click.</p>
<p id="info"></p><br>

<script> // PROBAR TAMBIÉN EN BURBUJEO – FALSE – Esto es independiente

document.addEventListener("click",myFunction,true);
cuerpo.addEventListener("click",myFunction,true);
demo.addEventListener("click",myFunction,true);
function myFunction(event) {
  info.innerHTML+="";
  info.innerHTML+="<br>currentTarget es:
"+event.currentTarget.nodeName+"<br>";
  info.innerHTML+="this es: "+this.nodeName+"<br>";
  info.innerHTML+="target es: "+event.target.nodeName+"<br>";
}
</script>
</body>
</html>
```

Ejercicio target - currentTarget

Ejercicio 26-modificando:

- Eliminar el segundo bloque myDiv2 y myP2.
- Asignar al primer bloque (DIV-p) la ejecución de 3 funciones cuando se produce el evento click:
  - Una en myDiv en fase de captura.
  - Una en myP en fase de captura.
  - Una en myDiv en fase de burbujeo.
  - NOTA: No es necesario registrar el click de myP en fase de captura y burbujeo. Al ser el elemento más interno realmente no se ejecuta en ninguna de esas fases, ya que es el destino, por lo que con ponerla en una de ellas es suficiente.
- Imprimir en todas las funciones evento.target y evento.currentTarget
- Observa y saca conclusiones

3.2 Métodos del objeto event

Métodos	Descripción
`preventDefault()`	Cancela cualquier acción asociada por defecto a un evento.
`stopPropagation()`	Evita que un evento burbujee. Por ejemplo si tenemos un `divA` que contiene un `divB` hijo. Cuando asignamos un evento de click a `divA`, si hacemos click en `divB`, por defecto se dispararía también el evento en `divA` en la fase de burbujeo. Para evitar ésto se puede llamar a `stopPropagation()` en `divB`. Para ello creamos un evento de click en `divB` y le hacemos `stopPropagation()`

El método `stopPropagation()` sirve para evitar que la propagación del evento continúe tanto en la fase de captura como de burbujeo. Desde el momento en que se ejecuta stopPropagation() las funciones manejadores asociadas a ese evento siguientes a ejecutarse no se ejecutarán

Ejercicio 32

Ejercicio 26-modificando:

- Eliminar el segundo bloque myDiv2 y myP2.
- Asignar al primer bloque (DIV-p) la ejecución de 3 funciones cuando se produzca el evento click:
  - Una en myDiv en fase de captura.
  - Una en myP en fase de captura.
  - Una en myDiv en fase de burbujeo.
- Inhibir en cada una de ellas la propagación del evento con stopPropagation()
- Observar qué ocurre.

3.4 Eventos del ratón en JavaScript.

Eventos `mousedown, mouseup y click`

Cada vez que un usuario hace clic se producen los eventos:

- **`mousedown:`** se produce al presionar el botón del ratón en un elemento.
- **`mouseup:`** se produce al soltar el botón del ratón en un elemento.
- **`click:`** se produce al presionar y soltar el botón del ratón en el mismo elemento.

Ejemplo: presionamos el botón sobre un elemento A, nos desplazamos y soltamos el botón sobre otro elemento B, se detectarán los eventos:

- `mousedown` sobre A.
- `mouseup` sobre B.
- No se detectará el evento de `click`.

Evento `dblclick`:

No se recomienda registrar este evento `click` y `dblclick` a la vez sobre el mismo elemento, ya que al hacer doble-click se dispararán ambos eventos.

3.4 Eventos del ratón en JavaScript.

Ejemplo eventos mousedown, mouseup y click

```
//Pincha en boton1 y suelta en botón 2
//Pincha en boton2 y suelta en botón 1
//Pincha en boton1 y suelta en botón 1
//Pincha en boton2 y suelta en botón 2

<input type="button" id="boton1" value="boton1"><br>
<input type="button" id="boton2" value="boton2"><br>
<p id="info"></p>

<script>

boton1.addEventListener("mouseup",mostrar);
boton1.addEventListener("mousedown",mostrar);
boton1.addEventListener("click",mostrar);

boton2.addEventListener("mouseup",mostrar);
boton2.addEventListener("mousedown",mostrar);
boton2.addEventListener("click",mostrar);

function mostrar(e){
   info.innerHTML+="<br>evento: "+e.type+" en elemento: "+this.value;
}
</script>
```

3.4 Eventos del ratón en JavaScript.

Evento `mousemove`

Este evento **se produce cada vez que el ratón se mueve 1 pixel.**

Por lo que si se registra este evento asignándole una función, cada vez que el ratón se mueve 1 pixel esta función se ejecutará.

Por lo que **se recomienda utilizar este evento sólo cuando haga falta, y desactivarlo cuando hayamos terminado.**

Eventos `mouseover y mouseout`:

Se producen cuando el ratón entra en la zona del elemento o sale del elemento.

Para saber de dónde procede el ratón y hacia dónde va se utiliza la propiedad **`relatedTarget`** para estos dos eventos.

`relatedTarget` para `mouseover` contiene el elemento desde dónde viene el ratón.

`relatedTarget` para `mouseout` contiene el elemento en el que acaba de entrar.

Ejercicio 36

3.4 Eventos del ratón en JavaScript.

Para saber **qué botón del ratón se ha pulsado se utilizan la propiedad `button` de los eventos `mousedown` o `mouseup`**.

Los valores de la propiedad button pueden ser los siguientes:

- Botón izquierdo: 0
- Botón medio: 1
- Botón derecho: 2

Para conocer la **posición en la que se encuentra el ratón**, se utilizan estas propiedades:

- **`clientX, clientY`**: devuelven las coordenadas del ratón relativas a la ventana.
- **`offsetX, offsetY`**: devuelven las coordenadas del ratón relativas al objeto target (donde se ha originado el evento).
- **`pageX, pageY`**: devuelven las coordenadas del ratón relativas al documento completo (aunque no sea vea todo en la ventana).
- **`screenX, screenY`**: devuelven las coordenadas del ratón relativas a la pantalla.

Las propiedades `layerX` y `layerY` no son estándard

3.3 Eventos del teclado en JavaScript.

Respecto al teclado hay eventos. Cuando se pulsa una tecla se generan los 3:

- `keydown/keypress:` se produce al presionar la tecla y mantener pulsada. Son el mismo evento.
- `keyup:` se produce al levantar la tecla.

Y hay dos tipos de teclas:

- **Las especiales:** Mays, Alt, AltGr, Intro, etc.
- **Las teclas alfanuméricas:** las letras, números, y símbolos (.,&/ etc.)

- Las propiedades **`keyCode` y `charCode`** están **obsoletas. Para averiguar el código de la tecla pulsada** utilizar**:**
  - La propiedad: **`key`**
  - De los eventos**:** **`keydown y keyup`**