UT6. 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.
    4) Acceso a los nodos de tipo atributo.
    5) Acceso a los nodos de tipo texto.
    6) Creación y borrado de nodos.
    7) 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.

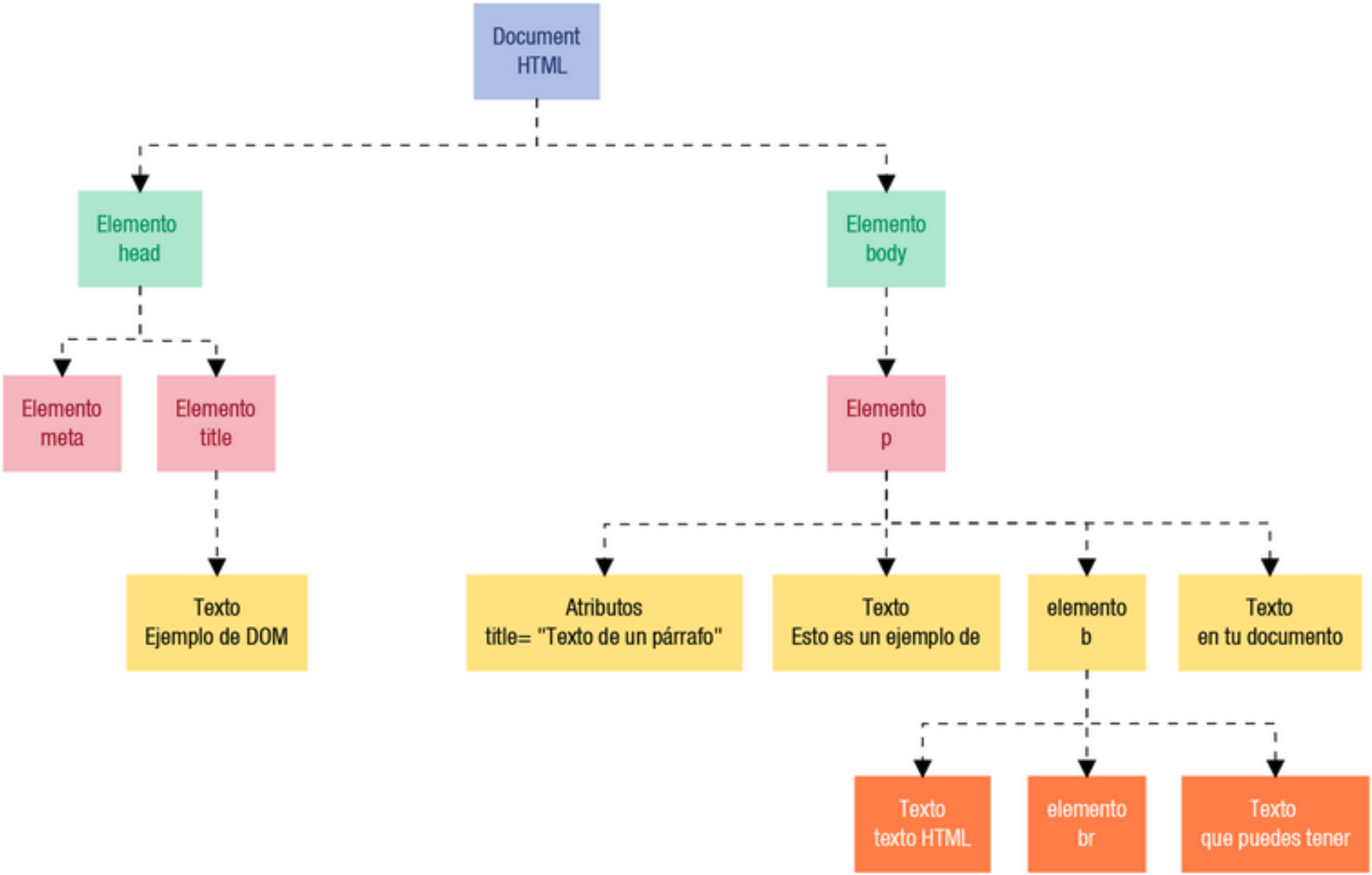
El DOM transforma **nuestro documento HTML en un conjunto de elementos, a los que llama nodos. Cada nodo es un objeto.**

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 un nodo específico (elemento HTML) lo podemos hacer empleando **dos métodos**:

- **A través de los nodos padre:**

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

- **Método de acceso directo**.

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

c_madrid
```

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 `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];
```

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];
```

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 `getElementsByClassName()`

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

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

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`:**

```
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.
- **`nodeValue o value`: el valor** del atributo.

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("value",
"Nuevo Valor");
```

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");
```

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");`

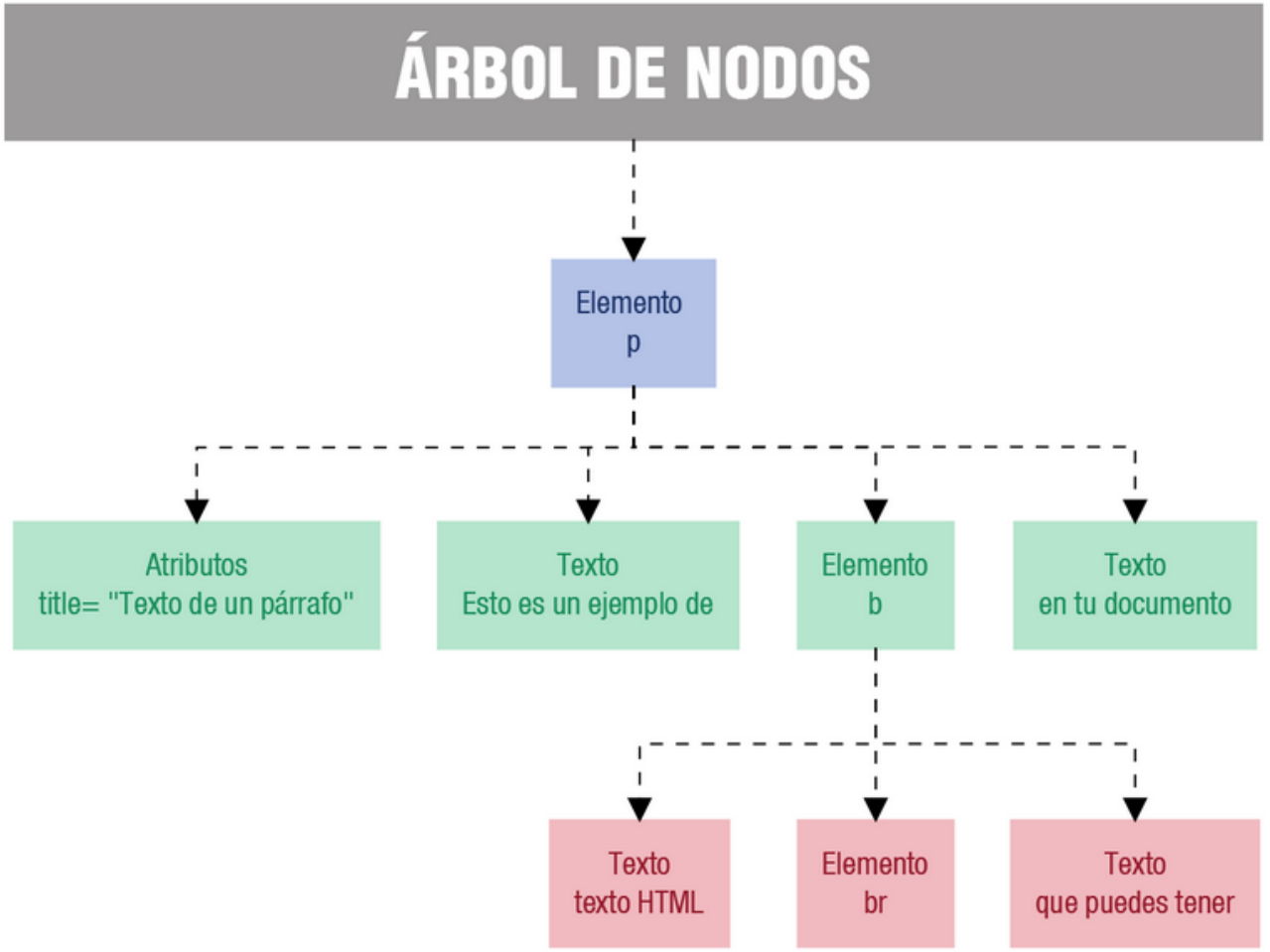
```
document.getElementById("nombre").removeAttribute("value");
```

1.5.- Acceso a los nodos de tipo texto.

Dada esta parte del documento HTML:

```
<p title="Texto de un párrafo">Esto es un ejemplo de
<b>texto HTML<br />

que puedes tener</b> en tu documento.</p>
```

Su árbol de nodos es:



1.5.- Acceso a los nodos de tipo texto.

La **colección `childNodes`** de un nodo correspondiente a un elemento **nos permite acceder a los nodos hijo** de un elemento, **ya sean de tipo element o text (no de tipo attr).**

Al texto "texto HTML" podemos acceder así:

```
document.getElementsByTagName("p")
[0].childNodes[1].childNodes[0].nodeValue;

//No existe value
```

`childNodes[1]` : selecciona el segundo hijo de <p> que sería el elemento <b> (el primer hijo es un nodo de tipo Texto "Esto es un…").

`childNodes[0]` : selecciona el primer hijo del elemento<b> que es el nodo de texto "texto HTML"

Se puede utilizar **`firstChild`** en lugar **`childNodes[0]`**.

Se puede utilizar **`lastChild`** para el último nodo de la colección.

Si se modifica el árbol DOM desde JavaScript las propiedades **`childNodes, firstChild y lastChild`** se recalculan.

1.5.- Acceso a los nodos de tipo texto.

Para modificar el contenido de un nodo, modificaremos la propiedad **`nodeValue`** del nodo:

Ejemplo:

```
document.getElementsByTagName("p")
[0].childNodes[1].childNodes[0].nodeValue=”Co
ntenido cambiado”

//no existe value
```

1.6.- Creado y borrado de nodos.

Los métodos **`createElement()`, `createTextNode()` y `appendChild()`, nos permitirán crear un elemento, crear un nodo de texto y añadir un nuevo nodo hijo.**

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

var nuevoParrafo = document.createElement('p');

nuevoParrafo.setAttribute("id","elnuevo");

var nuevoTexto = document.createTextNode('Contenido
añadido.');

nuevoParrafo.appendChild(nuevoTexto);

parrafo.appendChild(nuevoParrafo);
```

El resultado será:

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

1.6.- Creado y borrado de nodos.

Ejemplo: creamos una tabla en div

```
<div id="division"></div>
var tabla= document.createElement('table');
tabla.setAttribute(“border”,”1px”);
var titulo=document.createElement(“caption”);
var textoTitulo=document.createTextNode(“El titulo”);
titulo.appendChild(textoTitulo);
tabla.appendChild(titulo);
var fila1=document.createElement(“tr”);
tabla.appendChild(fila1);
var celda1=document.createElement(“td”);
var textoCelda1=document.createTextNode(“Celda 1”);
celda1.appendChild(textoCelda1);
fila1.appendChild(celda1);
var celda2=document.createElement(“td”);
var textoCelda2=document.createTextNode(“Celda 2”);
celda2.appendChild(textoCelda2);
fila1.appendChild(celda2);
division.appendChild(tabla);
```

1.6.- Métodos de los objetos nodos para crear y borrar nodos

Método	Descripción
`appendChild(newChild)`	Añade un hijo al final del nodo actual.
`cloneNode(deep)`	Realiza una copia del nodo actual (opcionalmente con todos sus hijos).
`hasChildNodes()`	Determina si el nodo actual tiene o no hijos (valor boolean).
`insertBefore(new, ref)`	Inserta un nuevo hijo antes de otro hijo.
`removeChild(old)`	Borra un hijo.
`replaceChild(new, old)`	Reemplaza un hijo viejo con el nuevo viejo.
`isSupported(feature, version)`	Determina cuando el nodo soporta una característica especial.

1.6.- Creado y borrado de nodos.

- **`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

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

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

- **`cloneNode(true)`** **copia un nodo** ya existente, **pero no lo añade automáticamente a la colección `childNodes`. Con true copia también todos sus hijos.**

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

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

1.7.- Propiedades y métodos de los objetos nodos

Propiedad	Valor	Descripción
`nodeName`	`String`	Varía según el tipo de nodo.
`nodeValue`	`String`	Varía según el tipo de nodo.
`nodeType`	`Integer`	Constante que representa cada tipo.
`parentNode`	`Object`	Referencia al siguiente contenedor más externo.
`childNodes`	`Array`	Todos los nodos hijos en orden.
`firstChild`	`Object`	Referencia al primer nodo hijo.
`lastChild`	`Object`	Referencia al último nodo hijo.
`previousSibling`	`Object`	Referencia al hermano anterior según su orden en el código fuente.
`nextSibling`	`Object`	Referencia al hermano siguiente según su orden en el código fuente.
`attributes`	`NodeMap`	Array de atributos de los nodos.
`ownerDocument`	`Object`	Contiene el objeto `document`.
`namespaceURI`	`String`	URI a la definición de namespace.
`Prefix`	`String`	Prefijo del `namespace`.
`localName`	`String`	Aplicable a los nodos afectados en el `namespace`.

1.7.- Propiedades y métodos de los objetos nodos

Las propiedades **`nextSibling`** y **`previousSibling`** solo funcionan con el código fuente HTML original.

Si el árbol DOM se modifica desde el código JavaScript estas propiedades no se recalculan, siguen guardando la información del código fuente HTML inicial.

Las propiedades **`childNodes, firstChild, lastChild`** sí van cambiando si cambia el árbol DOM de nuestro documento desde la parte de JavaScript.

NodeType tiene valor:

- 3 si el nodo es de tipo Text.
- 1 si el nodo es de tipo `Element`

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.

`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();`

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

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.
which	Indica el Unicode de la tecla presionada. Idéntico a charCode, excepto que esta propiedad también funciona en Netscape 4.

`currentTarget` es equivalente a `this`, es el elemento que tiene capturado el evento en ese momento. `target` hace referencia al elemento en el que se ha producido el evento. Ejemplo: si `body` tiene 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>
```

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

3.3 Eventos del teclado en JavaScript.

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

- `keydown:` se produce al presionar la tecla y mantener pulsada.
- `keypress:` se produce al presionar la tecla, es el mismo evento que `keydown`.
- `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`**

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.

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.

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