

BLOQUE DIDÁCTICO E: MANIPULACIÓN DEL MODELO DE OBJETOS DEL DOCUMENTO



TEMARIO DEL BLOQUE DIDÁCTICO E

1. El objeto WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, y otros.
2. Modelo de objetos del documento: Objeto DOCUMENT y tipos de nodos.
3. Selección de elementos del DOM.
4. Obtener y modificar DOM.
5. Temporizadores.
6. Cookies.



OBJETIVOS DEL BLOQUE E

- Analizar el funcionamiento del modelo de objetos del documento y del navegador.
- Describir la estructura y relación entre los distintos objetos del documento y el navegador.
- Reconocer el funcionamiento de los principales objetos pertenecientes al objeto window.
- Utilizar las propiedades y métodos de objetos BOM para obtener información del usuario.
- Analizar y modificar el texto y estructura de los elementos del documento.
- Recorrer los elementos del documento obteniendo información relevante.
- Analizar y modificar el formato CSS de los elementos.
- Crear y aplicar temporizadores web.

1. EL OBJETO WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, Y OTROS

BOM o Browser Objects Models → Es el modelo que permite a JS comunicarse con el navegador

¿Qué es el **DOM** y el **BOM**?

DOM o Document Objects Model → Es aquella parte del BOM que aglutina la estructura organizativa de los objetos del navegador

BOM

window

frames

history

location

navigator

screen

DOM

anchors

forms

images

links

areas

layers

.....

¿Forma parte el BOM del ECMA SCRIPT?



Pero los distintos navegadores han igualado las distintas formas de trabajar con estos objetos, por lo que se habla de una "estandarización" de los objetos del navegador

1. EL OBJETO WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, Y OTROS

OBJETO NAVIGATOR → Es un **objeto** que representa al **navegador del usuario** que está usando una aplicación web,



Posee numerosas propiedades de lectura, entre ellas **userAgent** que permite obtener información acerca del navegador, versión, motor, etc.

Este objeto posee otros métodos importantes como son:

- **clipboard** → devuelve un objeto para gestionar el código desde portapapeles.
- **cookieEnable** → devuelve booleano indicando si las cookies están activadas.
- **geolocation** → obtiene un objeto que permite la geolocalización del navegador
- **javaEnabled()** → devuelve booleano indicativo si java esta activo o no.
- **language** → devuelve **string** con el código del lenguaje del navegador
- **mimeType** → obtiene un array con los MIME aceptados por el ordenador.
- **online** → devuelve booleano indicando si el explorador está online o no.
- **plugins** → devuelve un array acerca de los plugins instalados
- **serviceWorker** → obtenemos un objeto para trabajar con ServiceWorker.
- **storage** → obtenemos un objeto StorageManager para manejar el API de servicios de almacenamiento persistente

```

Elements Console Sources Network > 3 | ⚙️ | ⚙️ | X
top | Filter Default levels ▾ 3 Issues: 3 | ⚙️ | ⚙️
> console.log(navigator.userAgent)
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
VM149:1
< undefined
>
  
```

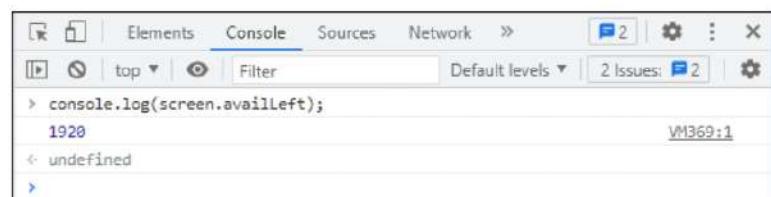
1. EL OBJETO WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, Y OTROS

OBJETO SCREEN → Es un **objeto** que **representa la pantalla del usuario** que permite obtener propiedades relacionadas con ella

Este objeto posee como propiedades importantes:



- **availTop** → devuelve la **primera coordenada superior** de la pantalla.
- **availLeft** → devuelve **primera coordenada superior izquierda** de la pantalla.
- **availHeight** → devuelve el número de pixeles de la **altura** de la pantalla.
- **availWidth** → devuelve el número de pixeles de la **anchura** de la pantalla.
- **height** → devuelve la altura completa de la pantalla del usuario en pixeles.
- **width** → devuelve la anchura completa de la pantalla del usuario en pixeles.
- **colorDepth** → devuelve el número de colores disponibles en la pantalla.
- **orientation** → devuelve un objeto ScreenOrientation que sirva para saber la orientación de la pantalla.



```
Elements Console Sources Network > 2 | Filter Default levels 2 Issues: 2 | VM369:1
> console.log(screen.availLeft);
1920
< undefined
```

1. EL OBJETO WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, Y OTROS

OBJETO LOCATION → Es un **objeto** que **representa la dirección URL** de la aplicación actual del usuario.



Este objeto posee como métodos importantes:

- **href** → devuelve la **URL completa** actual. Podemos modificarla para ir a otra url.
- **host** → devuelve de la URL **la dirección del host** con el puerto sino es estándar.
- **hostname** → igual que host pero sin el puerto.
- **search** → devuelve la **cadena de búsqueda** de la URL.
- **hash** → devuelve **el marcador**, si existe, de la URL. (ejem. js.com#link1 → **#link1**)
- **username** → devuelve el **nombre de usuario**, si lo hay, de la URL.
- **password** → devuelve la **contraseña**, si la hay, de la URL.
- **origin** → devuelve **la dirección canonical** de la URL.
- **reload(XXX)** → **recarga la página** donde XXX es un booleano que puede ser:
True → La recarga desde el servidor.
False → La recarga con la cache.

1. EL OBJETO WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, Y OTROS

OBJETO HISTORY → Es un **objeto** que representa el historial de las páginas visitadas por el usuario.



Este objeto posee como métodos importantes:

- **go(n)** → permite navegar por el historial recibiendo un número ("n") que puede ser:
 - 1 → página anterior del historial.
 - 0 → recarga la página actual.
 - 1 → página siguiente del historial.
- **length** → devuelve el **tamaño** del historial.

INFO ABOUT RIGHTS

2110259625010

www.safecreative.org/work

1. EL OBJETO WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, Y OTROS

OBJETO WINDOW → Otros métodos importantes del **objeto window** son:



- **innerWidth** → devuelve la **anchura interior** de la ventana o anchura real.
- **innerHeight** → devuelve la **altura interior** de la ventana o altura real.
- **outerWidth** → devuelve la **anchura exterior** de la ventana.
- **outerHeight** → devuelve la **altura exterior** de la ventana.
- **screenX** o **screenLeft** → devuelve la **distancia de la ventana al borde izquierdo** de la pantalla.
- **screenY** o **screenTop** → devuelve la **distancia de la ventana al borde superior** de la pantalla.
- **scrollX** → devuelve el **desplazamiento horizontal** que el usuario ha realizado (mouse o teclas).
- **scrollY** → devuelve el **desplazamiento vertical** que el usuario ha realizado (mouse o teclas).
- **scroll(x,y)** → permite el **desplazamiento absoluto** de la página a una posición concreta.
- **scrollBy(x,y)** → permite el **desplazamiento relativo** de la página una cantidad “x” e “y” de pixeles.
- **status** → referencia a la **barra de estado** de la ventana, permitiendo modificar su contenido.
- **console** → referencia a la **consola de JS**. Permite manejar todos los métodos disponibles.
- **event** → referencia al **objeto de tipo evento actual en ejecución**.

TEMARIO DEL BLOQUE DIDÁCTICO E

1. El objeto WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, y otros.
2. Modelo de objetos del documento: Objeto DOCUMENT y tipos de nodos.
3. Selección de elementos del DOM.
4. Obtener y modificar DOM.
5. Temporizadores.
6. Cookies.



INFO ABOUT RIGHTS

2110259625010

www.safecreative.org/work



OBJETIVOS DEL BLOQUE E

- Analizar el funcionamiento del modelo de objetos del documento y del navegador.
- Describir la estructura y relación entre los distintos objetos del documento y el navegador.
- Reconocer el funcionamiento de los principales objetos pertenecientes al objeto window.
- Utilizar las propiedades y métodos de objetos BOM para obtener información del usuario.
- Analizar y modificar el texto y estructura de los elementos del documento.
- Recorrer los elementos del documento obteniendo información relevante.
- Analizar y modificar el formato CSS de los elementos.
- Crear y aplicar temporizadores web.

2. MODELO DE OBJETOS DEL DOCUMENTO: OBJETO DOCUMENT Y TIPOS DE NODOS.

OBJETO DOCUMENT → Es una propiedad del **objeto window**, cuyo método principal es:

```

1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport"
6          |   content="width=device-width, initial-scale=1.0">
7      <title>Ejemplo Objeto DOCUMENT</title>
8  </head>
9  <body>
10     <section id="s1">
11         <h1>Titulo</h1>
12         <p>Este es un ejemplo</p>
13     </section>
14     <section id="s2">
15         <p>Este es otro ejemplo</p>
16     </section>
17 </body>
18 </html>
```

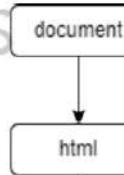


INFO ABOUT RIGHTS

2110259625010

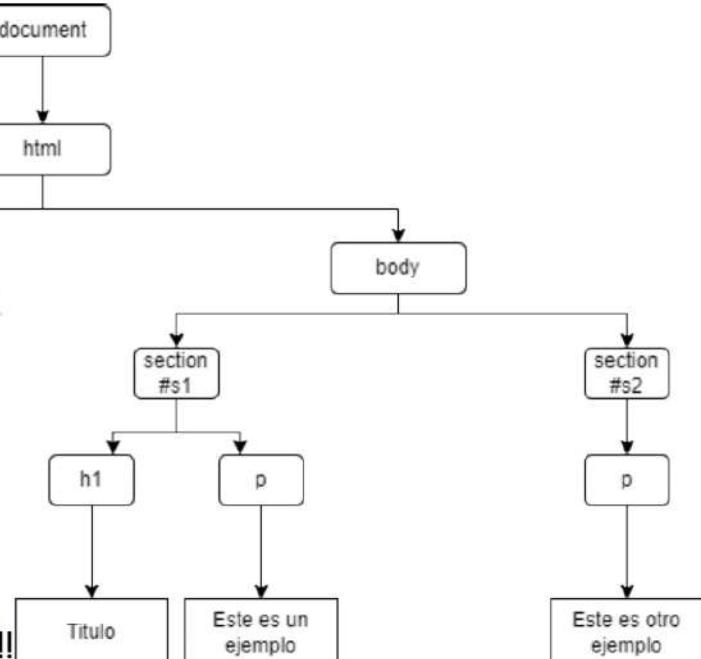
www.safecreative.org/work

document



→ var Var1 = document.getElementById("#s1");

¿Y esto que supone? ¡¡Has referenciado un nodo!!!



¿Podrías crear una variable en JS llamada "Var1" que referencia a la sección con "s1"?

2. MODELO DE OBJETOS DEL DOCUMENTO: OBJETO DOCUMENT Y TIPOS DE NODOS.

OBJETO DOCUMENT → Es una propiedad del **objeto window**, cuyo método principal es: `document.getElementById("#id");`

Todos los nodos poseen una **propiedad** llamada: `nombre_variable.nodeType` la cual devuelve un número, que puede ser:

VALOR	TIPO DE NODO	CONSTANTE RELACIONADA
1	Elemento	Node.ELEMENT_NODE
2	Atributo	Node.ATTRIBUTE_NODE
3	Texto	Node.TEXT_NODE
4	Apartado CDATA	Node.CDATA_SECTION_NODE
5	Referencia a entidad	Node.ENTITY_REFERENCE_NODE
6	Entidad	Node.ENTITY_NODE
7	Instrucción de procesado	Node.PROCESSING_INSTRUCTION_NODE
8	Comentario	Node.COMMENT_NODE
9	Documento completo	Node.DOCUMENT_NODE
10	Nodo de tipo documento	Node.DOCUMENT_TYPE_NODE
11	Nodo de fragmento de código	Node.DOCUMENT_FRAGMENT_NODE
12	Nodo de notación	Node.NOTATION_NODE

Esta propiedad es solo de **lectura**

También existe otra propiedad: `nombre_variable.nodeName`

que permite obtener el nombre del nodo en modo lectura
obteniendo como resultados:

TIPO NODO	VALOR DEVUELTO POR NODENAME
Elemento	Nombre de la etiqueta del elemento
Atributo	Nombre del atributo
Texto	#text
Comentario	#comment
Documento	#document

Por último, existe la propiedad: `nombre_variable.nodeValue`

TEMARIO DEL BLOQUE DIDÁCTICO E

1. El objeto WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, y otros.
2. Modelo de objetos del documento: Objeto DOCUMENT y tipos de nodos.
3. Selección de elementos del DOM.
4. Obtener y modificar DOM.
5. Temporizadores.
6. Cookies.



INFO ABOUT RIGHTS

2110259625010

www.safecreative.org/work



OBJETIVOS DEL BLOQUE E

- Analizar el funcionamiento del modelo de objetos del documento y del navegador.
- Describir la estructura y relación entre los distintos objetos del documento y el navegador.
- Reconocer el funcionamiento de los principales objetos pertenecientes al objeto window.
- Utilizar las propiedades y métodos de objetos BOM para obtener información del usuario.
- Analizar y modificar el texto y estructura de los elementos del documento.
- Recorrer los elementos del documento obteniendo información relevante.
- Analizar y modificar el formato CSS de los elementos.
- Crear y aplicar temporizadores web.

3. SELECCIÓN DE ELEMENTOS DEL DOM.

ELEMENTOS DEL DOM → Existen 4 métodos de selección de elementos:



- 1. MEDIANTE IDENTIFICADOR** → `let/var/const Nom_Var = document.getElementById("nom_id");`
- 2. POR ETIQUETA** → `let/var/const Nom_Eti = document.getElementsByTagName("etiqueta_html");`
La variable guarda todos los elementos de esa etiqueta en una lista
¿Cómo sabemos cuantos elementos hay? → `Nom_Eti.length`
¿Cómo accedemos a un elemento? → `Nom_Eti[k]` siendo k el índice del elemento
→ `let/var/const Nom_Eti2 = Nom_Var.getElementsByTagName("etiqueta_html");`
Esta manera permite afinar mas en un sub espacio de búsqueda por id
- 3. MEDIANTE CLASES** → `let/var/const Nom_Class = document.getElementsByClassName("clase_css");`
La variable guarda todos los elementos esa clase de css en un array
- 4. MEDIANTE SELECTOR** → `let/var/const Variable = document.querySelectorAll("xxx");`
Es el objeto más potente porque me permite seleccionar cualquier cosa, por ejemplo `xxx="ul li"`, elemento li dentro de ul.

TEMARIO DEL BLOQUE DIDÁCTICO E

1. El objeto WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, y otros.
2. Modelo de objetos del documento: Objeto DOCUMENT y tipos de nodos.
3. Selección de elementos del DOM.
4. Obtener y modificar DOM.
5. Temporizadores.
6. Cookies.



OBJETIVOS DEL BLOQUE E

- Analizar el funcionamiento del modelo de objetos del documento y del navegador.
- Describir la estructura y relación entre los distintos objetos del documento y el navegador.
- Reconocer el funcionamiento de los principales objetos pertenecientes al objeto window.
- Utilizar las propiedades y métodos de objetos BOM para obtener información del usuario.
- Analizar y modificar el texto y estructura de los elementos del documento.
- Recorrer los elementos del documento obteniendo información relevante.
- Analizar y modificar el formato CSS de los elementos.
- Crear y aplicar temporizadores web.

4. OBTENER Y MODIFICAR DOM.

MANIPULACIÓN DE ATRIBUTOS → Existen 4 métodos de modificación de atributos:



1. Obtener el valor → `Nom_Var.getAttribute("atributo_css");`

A partir del siguiente código en HTML5, recupera todos los elementos de la lista no ordenada y muestra solo aquellos que hayan sido afectados por una clase.

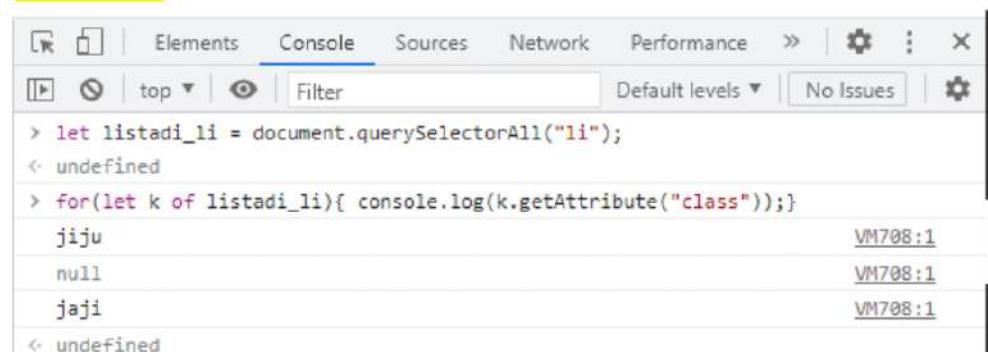
5min

```

1 <!DOCTYPE html>
2 <html lang="es" dir="ltr">
3   <head>
4     <meta charset="UTF-8">
5     <title>Obtener valor</title>
6     <style>
7       #lolo{color: blue}
8       .jiju{text-decoration: line-through;}
9       .jaji{text-decoration: underline;}
10      </style>
11    </head>
12    <body>
13      <h1>Obtener Valor</h1>
14      <p>Sea la lista de la compra:</p>
15      <ul>
16        <li class="jiju">Pan</li>
17        <li id="lolo">Chetos</li>
18        <li class="jaji">Cerveza</li>
19      </ul>
20    </body>
21  </html>
```

Solución

www.safecreative.org/work



```

Elements Console Sources Network Performance > | ⚙ ; ×
top ▾ Filter Default levels ▾ No Issues ⚙
> let listadi_li = document.querySelectorAll("li");
< undefined
> for(let k of listadi_li){ console.log(k.getAttribute("class")); }
jiju VM708:1
null VM708:1
jaji VM708:1
< undefined

```

4. OBTENER Y MODIFICAR DOM.

MANIPULACIÓN DE ATRIBUTOS → Existen 4 métodos de modificación de atributos:

2. Modificar el valor → `Nom_Var.setAttribute("atributo_css");`

A partir del siguiente formulario en HTML5, se pide anular el texto mostrado en *placeholder* ya que es un error no subsanado en la programación del formulario.



INFO ABOUT RIGHTS

Usuario:

Password:

2110259625010

www.safecreative.org/work

Usuario:

Password:

Solución

```
Elements Console Sources Network »
top Filter Default levels ▾ No Issues
> const userInput = document.querySelector('input');
< undefined
> userInput.setAttribute('placeholder', '');
< undefined
```

4. OBTENER Y MODIFICAR DOM.

MANIPULACIÓN DE ATRIBUTOS → Existen 4 métodos de modificación de atributos:



3. Eliminar el valor → `Nom_Var.removeAttribute("atributo_css");`



A partir del siguiente formulario de HTML5, se pide corregir el campo anulado ya que es un error no subsanado en la programación del formulario.

Solución

```
let problema = document.querySelector('#mail');
problema.removeAttribute('disabled');
```

```
1 <!DOCTYPE html>
2 <html lang="es" dir="ltr">
3 <head>
4   <meta charset="UTF-8">
5   <title>Eliminar Atributo</title>
6 </head>
7 <body>
8   <h1>Eliminar Atributo</h1>
9   <form action="">
10    <fieldset>
11      <legend>Datos Personales</legend>
12      <label for="nom">Nombre</label>
13      <input type="text" id="nom" name="nom"><br/>
14      <label for="ape">Apellidos</label>
15      <input type="text" id="ape" name="ape"><br/>
16      <label for="naci">Fecha Nacimiento</label>
17      <input type="date" id="naci" name="naci"><br/>
18      <label for="mail">Email</label>
19      <input type="email" id="mail" name="mail" disabled="disabled"><br/>
20      <input type="submit">
21    </fieldset>
22  </form>
23 </body>
24 </html>
```

Eliminar Atributo

Datos Personales

Nombre

Apellidos

Fecha Nacimiento dd/mm/aaaa

Email

4. OBTENER Y MODIFICAR DOM.

MANIPULACIÓN DE ATRIBUTOS → Existen 4 métodos de modificación de atributos:



4. Consultar existencia de atributos



Nom_Var.hasAttribute("atributo_css");

Se está analizando la accesibilidad web de un portal para discapacitados visuales, y nos han pedido desde el departamento de calidad la creación de un artefacto que permita validar de manera automática la accesibilidad de las imágenes.

Solución



```

Elements Console Sources > | ⚙ | X
top ▾ Filter Default levels ▾ | ⚙
No Issues
> let img = document.querySelectorAll("img");
< undefined
> for (let k of img){console.log(k.getAttribute("alt"));}
true VM270:1
false VM270:1
true VM270:1
< undefined
  
```

INFO ABOUT RIGHTS

2110259625010

```

1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="UTF-8">
5     <title>Existencia de atributos</title>
6   </head>
7   <body>
8     
10
11    
12
13    
15  </body>
16 </html>
  
```

4. OBTENER Y MODIFICAR DOM.

MANIPULACIÓN DEL **CONTENIDO** → Existen 2 propiedades que permiten **obtener** y **modificar** el contenido de un elemento:

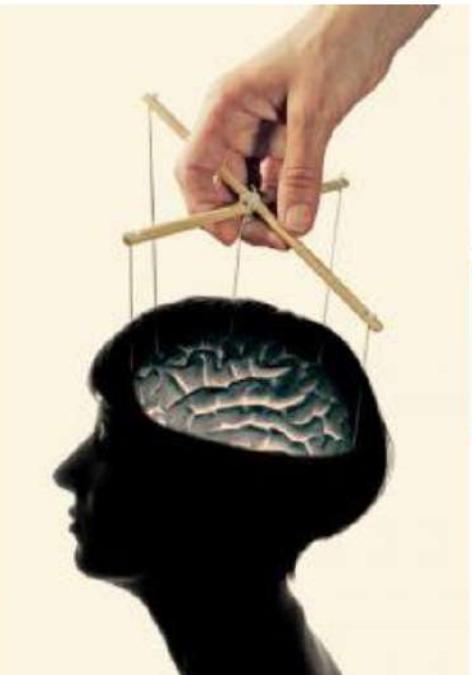
A. Propiedad `textContent` `Nom_Var.textContent;`

B. Propiedad `innerHTML` `Nom_Var.innerHTML;`

La propiedad B puede recuperar el contenido existente en etiquetas anidadas, cosa que la propiedad A no.

Se parte del siguiente código HTML5 y se pide extraer el contenido de los párrafos con el identificador `main` y sustituirlo por diez "X".

Solución



INFO ABOUT RIGHTS

2110259625010

```

Elements Console Sources Network Performance > | ⚙ : ×
top ▾ Filter Default levels ▾ No Issues ⚙
> console.log(document.getElementById("main").textContent);
AAAAAAA
<- undefined
> console.log(document.getElementById("main").textContent="XXXXXXXXXX");
XXXXXXXXXX
<- undefined
  
```

```

1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <title>Extraer Contenido</title>
6      <style>
7          .notice{color: ■ black}
8          #main{color: ■ blue}
9      </style>
10     </head>
11     <body>
12         <h1>Extraer contenido</h1>
13         <p id="main">AAAAAAA</p>
14         <p class="notice">BBB</p>
15         <p>CCCCCCCCC</p>
16         <p class="notice">DDD</p>
17     </body>
18 </html>
  
```

Extraer contenido

XXXXXXXXXX

BBB

CCCCCCCCC

DDD

4. OBTENER Y MODIFICAR DOM.

MANIPULACIÓN DEL CSS → Podemos modificar los **atributos de style** y **class** mediante el método **setAttribute**

Tan solo tenemos que utilizar el método **Nom_Var.style.propiedad_css="nueva_propiedad"**

También existe el método en el objeto **window** llamado **getComputedStyle(objeto)** que permite consultar las propiedades CSS que se están aplicando.



Se parte de la siguiente tabla de HTML5 y se pide extraer que se añadan mediante estilos CSS una línea continua sólida y de color negro con ancho de un píxel, con el fin de visualizar correctamente la tabla.

Solución



```
let tabla = document.getElementsByTagName("td");
for (let k of tabla){k.style.border="1px solid black";}
'1px solid black'
```



Modificar Estilo

NOMBRE APELLIDO

Mario	Benedetti
Federico	Garcia

Modificar Estilo

NOMBRE APELLIDO

Mario	Benedetti
Federico	Garcia

4. OBTENER Y MODIFICAR DOM.

MANIPULACIÓN DEL CSS → Podemos modificar los **atributos de style** y **class** mediante el método **setAttribute**



Para objetos con una sola clase existe la propiedad **className**, la cual permite consultar la clase de CSS existente, y en caso de querer añadir una lo realizaremos mediante **className = "nombre_clase"**

Cuando existe más de una clase aplicada en un objeto, usaremos la propiedad **classList** que permite consultar las propiedades CSS que se están aplicando. Esta propiedad dispone de las subpropiedades **add("nom_clase")** para añadir una clase a un objeto, **remove("nom_clase")** para eliminarlas, y **replace("clase_old", "clase_new")** para remplazar las clases.

Se tiene una página web HTML5 y CSS, cuyos elementos de tipo párrafo disponen de varias clases aplicadas de manera simultánea. Se pide obtener el código JS que permita visualizar dichas clases.

```

1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <title>Varias clases</title>
6      <style>
7          .clase1{color: blue}
8          .clase2{text-decoration: underline;}
9      </style>
10     </head>
11     <body>
12         <h1>Varias clases</h1>
13         <p class="clase1 clase2">Esto es el parrafo.</p>
14     </body>
15     </html>
```

www.safecreative.org/work

Solución

```

> let parrafo = document.getElementsByTagName("p");
< undefined
> for (let k of parrafo[0].classList){console.log(k);}
    clase1
    clase2
< undefined
VM260:1
VM260:1
```

4. OBTENER Y MODIFICAR DOM.

NAVEGAR POR EL DOM → Podemos obtener información acerca de la estructura del DOM, y así manipularlo



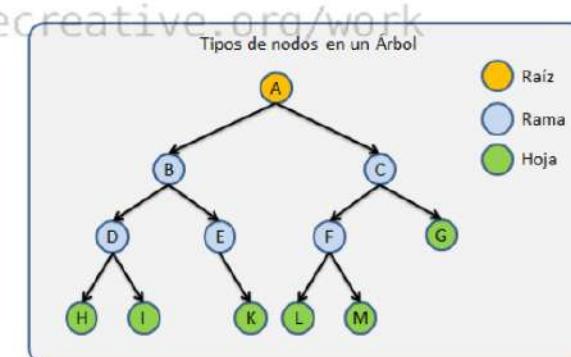
Todos los nodos cuentan con la propiedad llamada `childNodes` que devuelve la lista de nodos hijos.

También está la propiedad `children` que solo devuelve los elementos que sean hijos.

¿Qué propiedades existen?

PROPIEDAD	USO
<code>firstChild</code>	Selecciona el 1º nodo hijo del actual
<code>lastChild</code>	Selecciona el último nodo hijo
<code>nextSibling</code>	Selecciona el siguiente hermano respecto del nodo actual
<code>previousSibling</code>	Obtiene el hermano anterior al nodo actual
<code>parentNode</code>	Obtiene el padre del nodo actual
<code>firstElementChild</code>	Obtiene el primer hijo que sea un elemento del nodo
<code>lastElementChild</code>	Obtiene el último hijo que sea un elemento del nodo
<code>nextElementSibling</code>	Obtiene el siguiente hermano del anterior que sea un elemento
<code>previousElementSibling</code>	Obtiene el elemento hermano anterior al nodo actual
<code>childElementCount</code>	Número de elementos hijos del nodo actual

INFO ABOUT RIGHTS
2110259625010



4. OBTENER Y MODIFICAR DOM.

NAVEGAR POR EL DOM

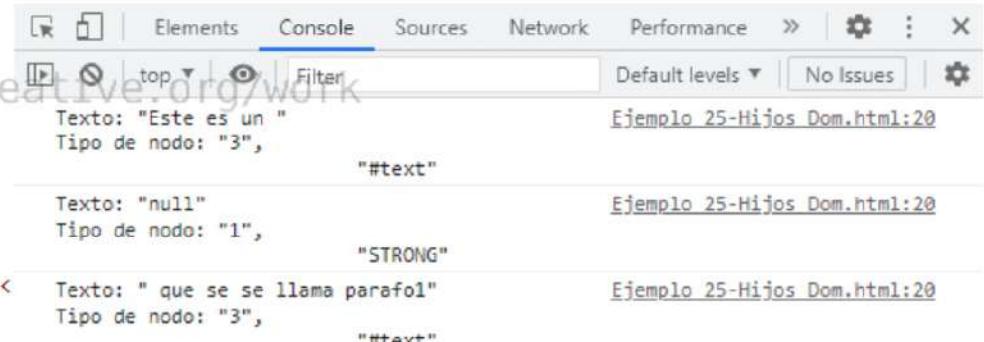


Dado el siguiente código html, se pide obtener tanto el contenido relativo al “párrafo1” y así como el tipo de nodo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hijos DOM</title>
</head>
<body>
  <section id="s1">
    <h1 id="titulo">Hijos DOM</h1>
    <p id="parrafo1">Este es un <strong>nodo</strong> que se se llama parrafo1</p>
    <p id="parrafo1">Yo soy el párrafo2</p>
    <p id="parrafo1">Y yo, el párrafo3</p>
  </section>
</body>
</html>
```

```
17   <script type="text/javascript">
18     let p1 = document.getElementById("parrafo1");
19     for (let hijo of p1.childNodes){
20       console.log(`Texto: ${hijo.nodeValue}\n`+
21           `Tipo de nodo: ${hijo.nodeType},`+
22           `${hijo.nodeName}`);
23     }
24   </script>
```

2110259625010



Output	File
Texto: "Este es un " Tipo de nodo: "3", "#text"	Ejemplo_25-Hijos_Dom.html:20
Texto: "null" Tipo de nodo: "1", "STRONG"	Ejemplo_25-Hijos_Dom.html:20
Texto: " que se se llama parrafo1" Tipo de nodo: "3", "#text"	Ejemplo_25-Hijos_Dom.html:20

4. OBTENER Y MODIFICAR DOM.

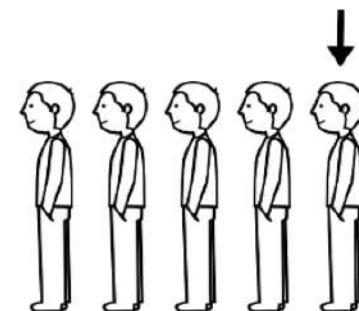
AÑADIR ELEMENTOS AL DOM

→ A partir de la **información** de la **estructura** del DOM, podemos añadir etiquetas
En el objeto **document** se dispone del método **createElement("etiqueta")** que **permite crear elementos** y mediante la propiedad **innerHTML** **podemos añadir contenido a la etiqueta creada**. Por último, faltaría añadirlo al DOM con el método **appendChild(nombre_elemento)**

También existe en el objeto **document** el método **createTextNode("texto")** que **permite crear nodos de tipo texto** y el cual deberemos añadirlo con la propiedad **innerHTML**

¿Qué problema existe con esta metodología de adición de elementos?

¡¡ Si hay elementos siempre se colocan los últimos!!



4. OBTENER Y MODIFICAR DOM.

NAVEGAR POR EL DOM



Dado el siguiente código html, se pide insertar un nodo h1 en el div preparado llamado "capa" con el texto "Añadiendo contenido".

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Añadir contenido</title>
8 </head>
9 <body>
10  <div id="capa"></div>
11  <p>Ya hemos aprendido</p>
12 </body>
13 </html>
```



```
> var capa = document.getElementById("capa");
  var h1 = document.createElement("h1");
  h1.innerHTML = "Añadiendo contenido";
  capa.appendChild(h1);
<- <h1>Añadiendo contenido</h1>
```

2110259625010

Añadiendo contenido

Ya hemos aprendido

4. OBTENER Y MODIFICAR DOM.

AÑADIR ELEMENTOS AL DOM → Podemos añadir etiquetas en **POSICIONES DETERMINADAS** o eliminarlas

```

9   <body>
10  <div id="especial">
11    <p>Primer parrafo</p>
12    <p>Segundo parrafo</p>
13    <p>Tercer parrafo</p>
14  </div>
15 </body>
```

Primer parrafo

Segundo parrafo

Tercer parrafo

Añadir capas

```

15  <script>
16    let capa = document.getElementById("especial"); //busco el bloque
17    let newP = document.createElement("p"); //creo etiqueta
18    newP.textContent="Me pongo donde quiero"; //relleno etiqueta
19    let Pdos = document.querySelectorAll("#especial p:nth-of-type(2)")[0]; //posiciono
20    capa.insertBefore(newP,Pdos); //inserto
21  </script>
```

Primer parrafo

Me pongo donde quiero

Segundo parrafo

Eliminar capas

```

15  <script>
16    let capa = document.getElementById("especial"); //busco el bloque
17    let Pdos = document.querySelectorAll("#especial p:nth-of-type(2)")[0]; //posiciono
18    capa.removeChild(Pdos); //elimino
19    capa.appendChild(Pdos); //Añado
20  </script>
```

Tercer parrafo

Primer parrafo

Tercer parrafo

Segundo parrafo

INFO ABOUT RIGHTS

2110259625010

www.safecreative.org/work

TEMARIO DEL BLOQUE DIDÁCTICO E

1. El objeto WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, y otros.
2. Modelo de objetos del documento: Objeto DOCUMENT y tipos de nodos.
3. Selección de elementos del DOM.
4. Obtener y modificar DOM.
- 5. Temporizadores.**
6. Cookies.



OBJETIVOS DEL BLOQUE E

- Analizar el funcionamiento del modelo de objetos del documento y del navegador.
- Describir la estructura y relación entre los distintos objetos del documento y el navegador.
- Reconocer el funcionamiento de los principales objetos pertenecientes al objeto window.
- Utilizar las propiedades y métodos de objetos BOM para obtener información del usuario.
- Analizar y modificar el texto y estructura de los elementos del documento.
- Recorrer los elementos del documento obteniendo información relevante.
- Analizar y modificar el formato CSS de los elementos.
- Crear y aplicar temporizadores web.

5. TEMPORIZADOR.

TEMPORIZADORES → El objeto `window` tiene varios métodos que permiten trabajar con el tiempo para desencadenar un evento



¿En qué se
diferencian?



`setTimeout(fx, time)` → Establece una cuenta atrás única para desencadenar el evento

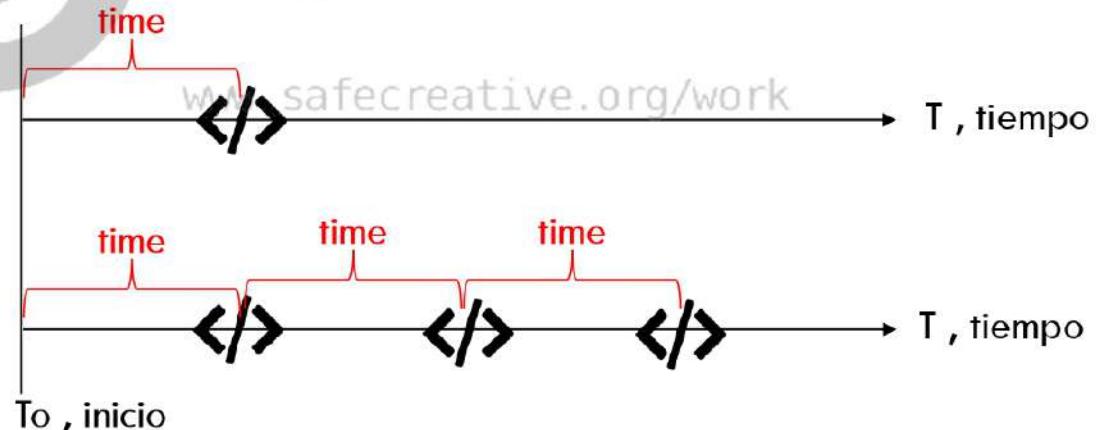
`setInterval(fx, time)` → Establece una secuencia temporal de aparición de eventos

`clearTimeout(variable)`
`clearInterval(variable)`

- `fx` → es el código que ha de ejecutarse
- `time` → es la cantidad de tiempo en milisegundos

`setTimeout()`

`setInterval()`



5. TEMPORIZADOR.



Se pide crear un código en JS que produzca una alerta transcurridos cinco segundos desde la ejecución.



Se pide crear un código en JS que produzca alertas cada cinco segundos desde la ejecución.

INFO ABOUT RIGHTS

2110259625010

www.safecreative.org/work

```

1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Alerta tras 5 segundos</title>
8  </head>
9  <body>
10     <h1>Alerta tras 5 segundos</h1>
11     <p>El código solo genera una alerta tras los 5 segundos</p>
12
13     <script type="text/javascript">
14         var comienzo = setTimeout(()=>alert("Hola"),5000) //creamos
15         comienzo //ejecutamos alerta
16     </script>
17 </body>
18 </html>
```

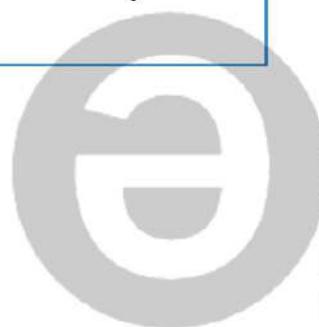
```

1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Alerta cada 2 segundos</title>
8  </head>
9  <body>
10     <h1>Alerta cada 2 segundos</h1>
11     <p>El código solo alertas cada 2 segundos</p>
12
13     <script type="text/javascript">
14         var comienzo = setInterval(()=>alert("Hola"),2000) //creamos
15         comienzo //ejecutamos alerta
16     </script>
17 </body>
18 </html>
```

5. TEMPORIZADOR.



Se pide crear un código en JS que produzca alertas cada 2 segundos y transcurridos 6 segundos ya no agobie más.



```
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Alerta cada 2 segundos</title>
8  </head>
9  <body>
10     <h1>Alerta cada 2 segundos</h1>
11     <p>El código genera alertas cada 2 segundos y transcurrido 6 lo cancela</p>
12
13     <script type="text/javascript">
14         var comienzo = setInterval(()=>alert("Hola"),2000) //creamos
15             comienzo //ejecutamos alerta
16         var fin = setTimeout(()=>clearInterval(comienzo),6000) //creamos
17             fin //ejecutamos alerta
18     </script>
19  </body>
20  </html>
```

TEMARIO DEL BLOQUE DIDÁCTICO E

1. El objeto WINDOW: BOM, NAVIGATOR, SCREEN, LOCATION, HISTORY, y otros.
2. Modelo de objetos del documento: Objeto DOCUMENT y tipos de nodos.
3. Selección de elementos del DOM.
4. Obtener y modificar DOM.
5. Temporizadores.
6. Cookies.



OBJETIVOS DEL BLOQUE E

- Analizar el funcionamiento del modelo de objetos del documento y del navegador.
- Describir la estructura y relación entre los distintos objetos del documento y el navegador.
- Reconocer el funcionamiento de los principales objetos pertenecientes al objeto window.
- Utilizar las propiedades y métodos de objetos BOM para obtener información del usuario.
- Analizar y modificar el texto y estructura de los elementos del documento.
- Recorrer los elementos del documento obteniendo información relevante.
- Analizar y modificar el formato CSS de los elementos.
- Crear y aplicar temporizadores web.

6. COOKIES.

¿QUÉ SON LAS **COOKIES**?

→ Archivos de texto generados por las aplicaciones web donde escriben datos y los guardan en el cliente, para después volver a leerlos.



¿Y existen tipos de cookies? → **Si**, las estrictamente necesarias y las no necesarias
www.safecreative.org/work

¿En qué se diferencian?

→ Las estrictamente necesarias son las “buenas” ya que permiten hacer funcionar correctamente la aplicación web.
Las no necesarias son las “malas” ya que rastrean lo que haces durante tu uso en la web.

¿Existe regulación?

→ **Si**, Regulación Europea. Actualmente existen formas de evadirla como es el almacenamiento en la API localStorage de google

6. COOKIES.

TRABAJAR CON COOKIES → El objeto **document** dispone de la **propiedad cookie** que es la que controla las cookies



Por defecto, las cookies se borran al finalizar la sesión → **cookies efímeras**

Existe la posibilidad de hacer **cookies permanentes**, es decir, que no se borren.

- Tan solo tenemos que especificarlo con **expires** seguido de la fecha de expiración en formato UTC



Tipo	Sintaxis	Ejemplo
Fecha: año	YYYY	1997
Fecha: año y mes	YYYY-MM	1997-07
Fecha: año, mes y día	YYYY-MM-DD	1997-07-16
Fecha más horas y minutos	YYYY-MM-DDThh:mmTZD	1997-07-16T19:20+01:00
Fecha más horas, minutos y segundos	YYYY-MM-DDThh:mm:ssTZD	1997-07-16T19:20:30+01:00
Fecha más horas, minutos, segundos y segundos fraccionarios	YYYY-MM-DDThh:mm:ss.sTZD	1997-07-16T19:20:30.30.45+01:00

- YYYY = un año de cuatro dígitos
- MM = un mes de dos dígitos, de 01 a 12
- DD = un día del mes de dos dígitos, de 01 a 31
- T = un valor literal que sigue a la fecha y presenta la hora
- hh = dos dígitos para la hora, de 00 a 23
- mm = dos dígitos para los minutos, de 00 a 59
- ss = dos dígitos para los segundos, de 00 a 59
- s = uno o varios dígitos que representan una fracción decimal de un segundo
- TZD = designador de zona horaria (Z o +hh:mm o -hh:mm)
 - Z para hora UTC
 - +hh:mm para una zona horaria local anterior a UTC
 - -hh:mm para una zona horaria local posterior a UTC

La lectura de las cookies se realiza con la propiedad **path** la cual indica la localización de la cookie para ser leída, y **domain** relativo al dominio.

6. COOKIES.



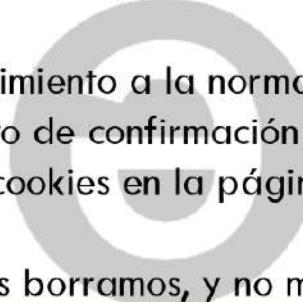
Se pide crear un código en JS de una cookie persistente, a consecuencia de navegar por la página formateya.com la cual ha de ser guardada en el directorio raíz (/). Esta ha de grabar el nombre de usuario, por ejemplo, "José", y también ha de expirar su vigencia transcurrido una semana.

```
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Cookie</title>
8  </head>
9  <body>
10     <h1>Cookie</h1>
11     <p>Atención que te meto la cookie.</p>
12     <script type="text/javascript">
13         let hoy = new Date();
14         let caducidad_en_ms = hoy.getTime() + 1000*60*60*24*7
15         let caducidad = new Date(caducidad_en_ms)
16         document.cookie='Cookie_Usuario=JOSE;
17                                         expires=${caducidad.toUTCString()};
18                                         path=/;
19                                         domain=formateya.com'
20     </script>
21  </body>
22  </html>
```



Calificable

Crear una aplicación que nos muestre el número de veces que hemos entrado. Para ello emplear una cookie que ha de caducar en un año.

Con el objetivo de dar cumplimiento a la normativa vigente, también se pide que dicha aplicación avise con un cuadro de confirmación previo (que ha de aparecer una sola vez por sesión) que vamos a grabar cookies en la página.  **2110259625010**

Si el usuario no las acepta las borramos, y no mostraremos las visitas.

www.safecreative.org/work

Entrega: **Cookie**

Formato entrega: **.word**

Margen: (vertical: 2) + (horizontal: 3)

Fuente: Calibri / Arial - 11 ptos - Justificado

Interlineado: 1,5 ptos

JOSE-CARLOS-ROMAN-HERRERA.Entrega-**cookie**.word

Román-Herrera, J.C.

