Unidad 6
Modelo de objetos del cliente

CESUR
Tu Centro Oficial de FP

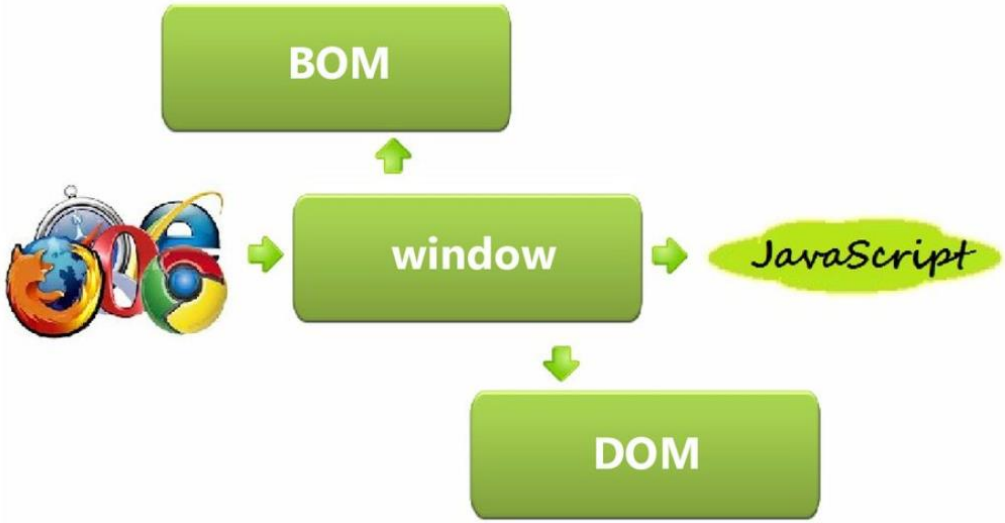
1. Introducción

BOM (modelo de objetos del navegador) – objeto en el que se representa el navegador, no sólo el documento. Mediante el BOM se puede acceder a datos como el historial de navegación, dimensiones de la ventana, etc.

DOM (modelo de objetos del documento) – plataforma e interfaz de un documento que permite a los scripts y programas acceder y modificar dinámicamente su contenido, estructura y estilo.

JS tiene la ventaja de poder acceder a ambos objetos, lo cual consigue que las aplicaciones sean interactivas.

Accediendo al DOM y al BOM, el programador va a poder conocer la potencia de la programación de lado del cliente. Esto permitirá que mucha de la lógica de las aplicaciones web corra en este lado dedicándose el servidor a temas más específicos de back-end.

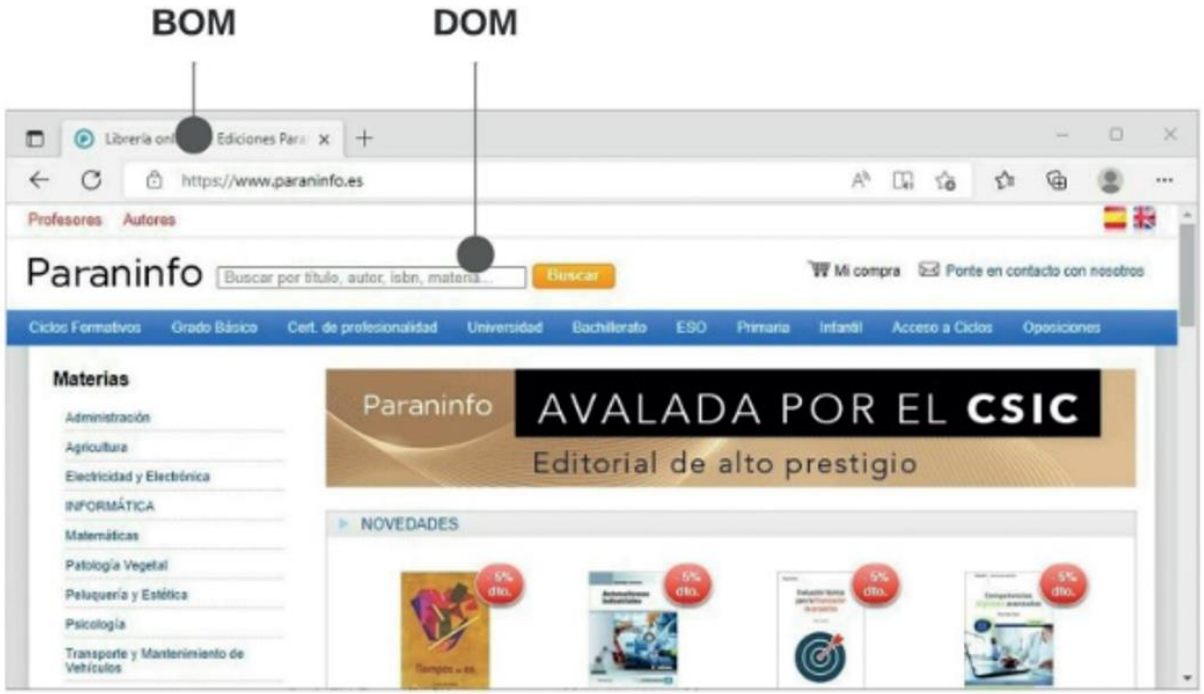


1. Introducción

Muchas veces cuando hablamos del modelo de objetos de cliente se piensa únicamente en el **DOM** (modelo de objetos del documento).

Pero el **DOM** no lo es todo, sobre todo teniendo en cuenta que el **DOM** es una parte del **BOM** (modelo de objetos del navegador), ya que el navegador es toda la interfaz que tienen los usuarios a mano para usar una aplicación web.

El **BOM** da acceso a todo lo que tiene que ver con el navegador, y el **DOM** con el sitio web que en cada momento se representa en el navegador.



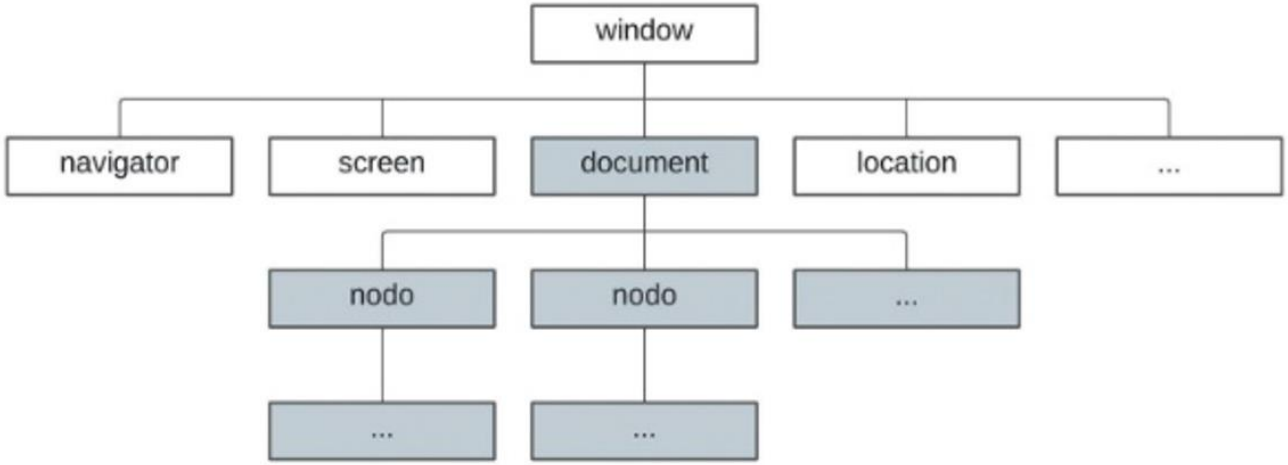
2. Modelo de objetos del navegador (BOM)

El **BOM** permite que JS se comunique con el navegador que utilizan los usuarios para usar una aplicación web.

No existen estándares oficiales que lo normalicen, sin embargo, debido a que una extensa generalidad de los navegadores modernos ha implementado casi todos los métodos y propiedades de JS, con frecuencia se denominan **propiedades y métodos del BOM**.

Los elementos que vamos a ver son:

1) Objeto window
2) Objeto navigator
3) Objeto screen
4) Objeto location
5) Objeto history
6) Temporizadores



2. Modelo de objetos del navegador (BOM)

1) OBJETO WINDOW

Representa la ventana del navegador.

Todos los objetos, funciones y variables globales de JS se convierten automáticamente en miembros de este objeto, de tal forma que:

- Las variables serán las propiedades
- Las funciones los métodos

Hasta el DOM es una propiedad del objeto window.

Una de las principales utilidades que ofrece este objeto es poder obtener el tamaño de la ventana del navegador:

```
let ancho = window.innerWidth;
let alto = window.innerHeight;
console.log(`Ancho x Alto (en píxeles): ${ancho} x ${alto}`);
```

```
Ancho x Alto (en píxeles): 1186 x 151
>
```

Los métodos más útiles del objeto window:

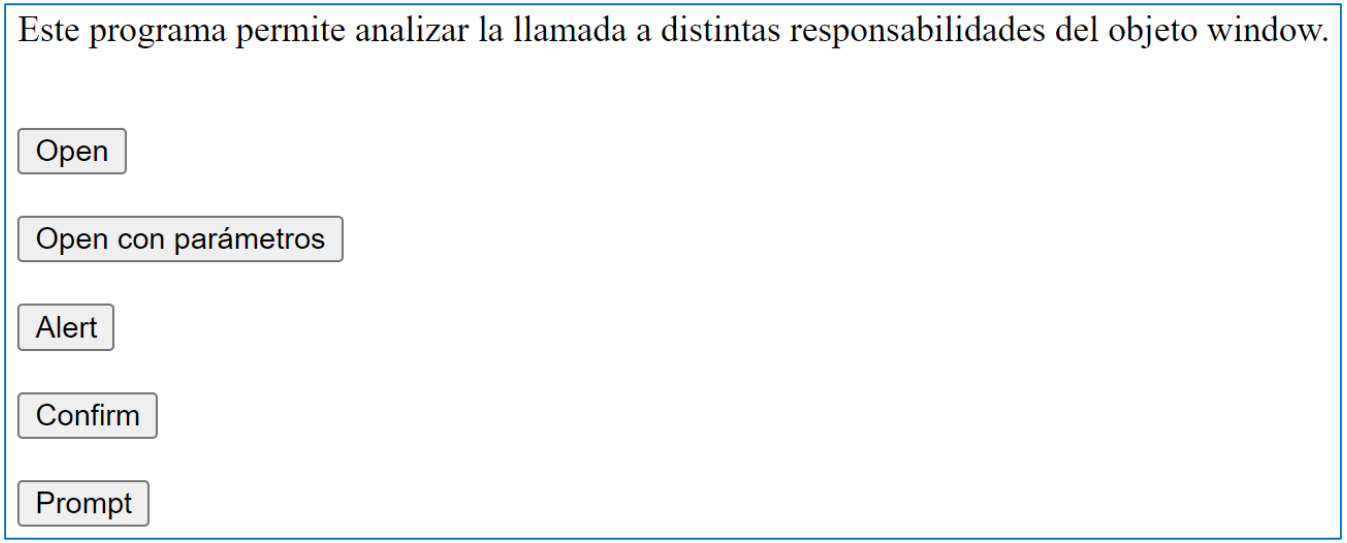
Método	Utilidad
alert()	Abre una ventana de alerta con un mensaje y un botón para aceptar.
blur()	Retira el foco de una ventana.
close()	Cierra la ventana.
confirm()	Abre una ventana de diálogo con dos botones: **Aceptar** y **Cancelar.** Devuelve **true** si se pulsa **Aceptar,** y **false** en caso contrario.
focus()	Asigna el foco a una ventana.
moveBy()	Mueve la ventana la cantidad de píxeles que se indique desde su posición actual.
moveTo()	Mueve la ventana a las coordenadas en píxeles que se le indiquen.
open()	Abre una URL en una nueva pestaña.
print()	Abre el cuadro de diálogo de impresión para imprimir una ventana.
prompt()	Abre un cuadro de diálogo para que el usuario introduzca datos y dos botones: **Aceptar** y **Cancelar.** Devuelve el valor introducido por el usuario si se pulsa en **Aceptar,** y **null** en otro caso.
resizeBy()	Redimensiona la ventana usando los incrementos o decrementos que se indican en píxeles.
resizeTo()	Redimensiona la ventana al tamaño indicado.
scrollBy()	Desplaza el contenido de la ventana el número de píxeles que se le indique.
scrollTo()	Desplaza el contenido de la ventana hasta la posición en píxeles que se indica.
stop()	Cancela la carga de la página.

Objeto window
https://developer.mozilla.org/es/docs/Web/API/Window

Ejemplo 1: OBJETO WINDOW

Escribe un programa que incluya los siguientes botones:

1) Open – Abre una nueva pestaña/ventana del navegador
2) Open con parámetros – abre una nueva ventana de 400x250 px
3) Alert – abre una ventana con un botón Aceptar
4) Confirm – abre una ventana con dos botones, Aceptar y Cancelar
5) Prompt – abre una ventana que nos pide una cadena y luego nos muestra los datos introducidos



CESUR
Tu Centro Oficial de F

Ejemplo 1: MÉTODOS DEL OBJETO WINDOW

```
<body>
    <p>Este programa permite analizar la llamada a distintas responsabilidades del objeto window.</p><br>
        <form>
            <input type="button" value="Open" onClick="abrir()"><br><br>
            <input type="button" value="Open con parámetros" onClick="abrirParametros()"><br><br>
            <input type="button" value="Alert" onClick="mostrarAlerta()"><br><br>
            <input type="button" value="Confirm" onClick="confirmar()"><br><br>
            <input type="button" value="Prompt" onClick="cargarCadena()">
        </form>
        <script>
            function abrir() {
                let ventana = open();
                ventana.document.write("Estoy escribiendo en la nueva ventana<br>");
                ventana.document.write("Segunda linea");
            }

            function abrirParametros() {
                let ventana = open('', '', 'status=yes,width=400,height=250,menubar=yes');
                ventana.document.write("Esto es lo primero que aparece<br>");
            }

            function mostrarAlerta() {
                alert("Esta ventana de alerta ya la utilizamos en otros problemas.");
            }

            function confirmar() {
                let respuesta = confirm("Presione alguno de los dos botones");
                if (respuesta == true)
                    alert("presionó aceptar");
                else
                    alert("presionó cancelar");
            }

            function cargarCadena() {
                let cad = prompt("cargue una cadena:");
                alert("Usted ingreso " + cad);
            }
        </script>
</body>
```

2. Modelo de objetos del navegador (BOM)

2) OBJETO NAVIGATOR

Objeto de window que contiene información sobre el navegador que está utilizando el usuario para interactuar con la aplicación web.

NOTA: puede utilizarse directamente sin utilizar el prefijo window.

Algunas de las propiedades más útiles del objeto navigator son:

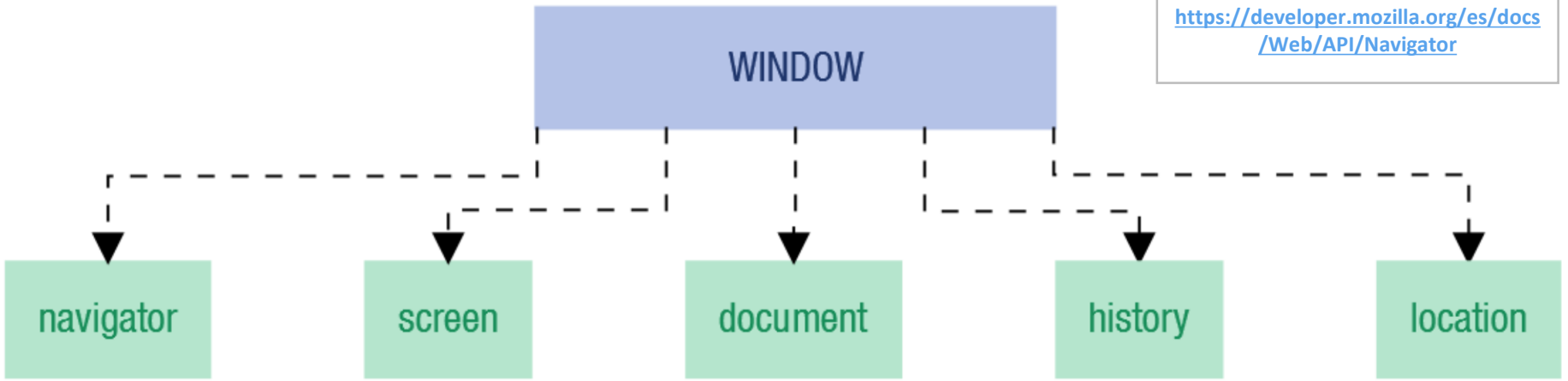
Propiedad	Utilidad
clipboard	Devuelve un objeto para trabajar con el portapapeles.
cookieEnabled	Devuelve **true** si las *cookies* están habilitadas.
geolocation	Devuelve un objeto con la localización del usuario.
language	Devuelve el idioma del navegador.
onLine	Devuelve **true** si el navegador está *online*.
plugins	Devuelve información sobre los *plugins* instalados.
storage	Devuelve un objeto que permite trabajar con la API de almacenamiento de datos persistentes.
userAgent	Devuelve información sobre el navegador del usuario.

2. Modelo de objetos del navegador (BOM)

Otras propiedades de navigator son:

- appName : almacena el nombre oficial del navegador.
- appVersion : almacena la versión del navegador.
- cookieEnabled : almacena si las cookies están activas en el navegador.
- platform : almacena la plataforma donde el navegador se está ejecutando.
- plugins : almacena un array de los plugin cargados en el navegador.

Objeto navigator
https://developer.mozilla.org/es/docs/Web/API/Navigator



2. Modelo de objetos del navegador (BOM)

Ejemplo 2: OBJETO NAVIGATOR

Escribe un programa que incluyendo las propiedades del objeto navigator obtenga la siguiente salida:

VALORES DE LAS PROPIEDADES DEL OBJETO NAVIGATOR:

Nombre oficial del navegador (Propiedad appName):Netscape

Versión del navegador (Propiedad appVersion):5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36

Están las cookies activas (Propiedad cookieEnabled):true

Plugins cargados en el navegador (Propiedad plugins):5

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Actividad 1</title>
    </head>
    <body>
        <script>
            document.write('VALORES DE LAS PROPIEDADES DEL OBJETO NAVIGATOR:<br><br>');
            document.write('Nombre oficial del navegador (Propiedad appName):' + navigator.appName + '<br><br>');
            document.write('Versión del navegador (Propiedad appVersion):' + navigator.appVersion + '<br><br>');
            document.write('Están las cookies activas (Propiedad cookieEnabled):' + navigator.cookieEnabled + '<br><br>');
            document.write('Plugins cargados en el navegador (Propiedad plugins):' + navigator.plugins.length + '<br><br>');
        </script>
    </body>
</html>
```

2. Modelo de objetos del navegador (BOM)

3) OBJETO SCREEN

Objeto de window que contiene información sobre la pantalla del usuario. Hay que tener en cuenta que los navegadores determinan cuál es la pantalla actual detectando qué pantalla tiene el navegador centrado.

Algunas de las propiedades más útiles del objeto screen son:

Propiedad	Utilidad
availHeight	Especifica la altura de la pantalla, en píxeles.
availWidth	Especifica la anchura de la pantalla, en píxeles.
colorDepth	Devuelve la profundidad de color de la pantalla.
height	Altura completa de la pantalla del usuario.
orientation	Devuelve un objeto con información sobre la orientación de la pantalla. Es una propiedad experimental que continuará, pero podría modificar su funcionamiento en el futuro próximo.
pixelDepth	Devuelve la profundidad de bits de la pantalla.
width	Anchura completa de la pantalla del usuario.

Objeto screen
https://developer.mozilla.org/es/docs/Web/API/Screen

2. Modelo de objetos del navegador (BOM)

Ejemplo 3: OBJETO SCREEN

Escribe un programa que incluyendo las propiedades del objeto screen obtenga la siguiente salida:

VALORES DE LAS PROPIEDADES DEL OBJETO SCREEN:

Altura de la pantalla en píxeles (Propiedad availHeight): 690 px

Anchura de la pantalla en píxeles (Propiedad availWidth): 1175 px

Altura completa de la pantalla del usuario (Propiedad height): 734 px

Anchura completa de la pantalla del usuario (Propiedad width): 1175 px

Profundidad de color de la pantalla (Propiedad colorDepth): 24

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>
<body>
    <script>
        document.write('VALORES DE LAS PROPIEDADES DEL OBJETO SCREEN:<br><br>');
        document.write('Altura de la pantalla en píxeles (Propiedad availHeight): ' + screen.availHeight + ' px<br><br>');
        document.write('Anchura de la pantalla en píxeles (Propiedad availWidth): ' + screen.availWidth + ' px<br><br>');
        document.write('Altura completa de la pantalla del usuario (Propiedad height): ' + screen.height + ' px<br><br>');
        document.write('Anchura completa de la pantalla del usuario (Propiedad width): ' + screen.width + ' px<br><br>');
        document.write('Profundidad de color de la pantalla (Propiedad colorDepth): ' + screen.colorDepth);
    </script>
</body>
</html>
```

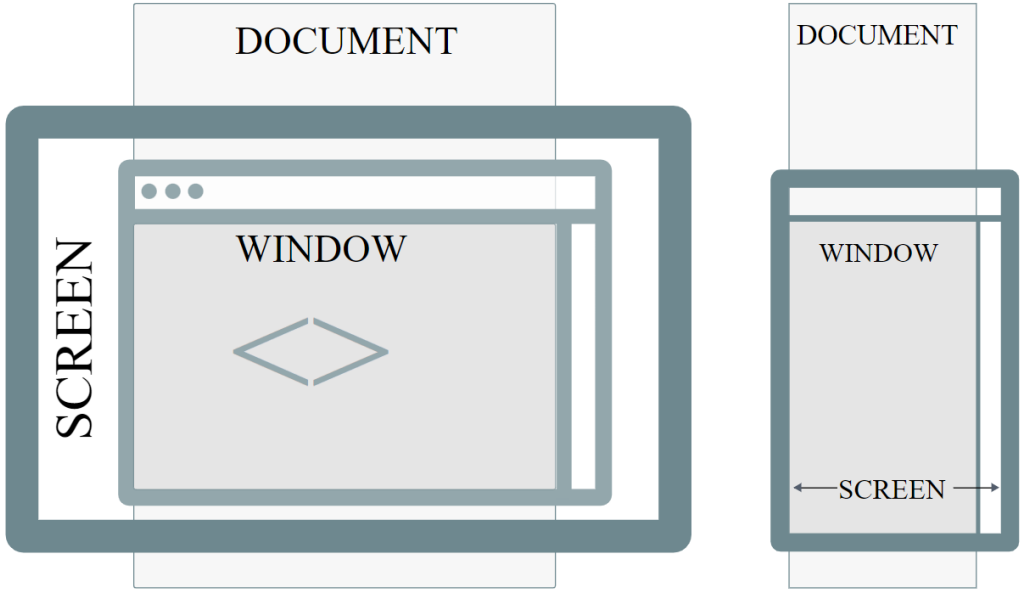
2. Modelo de objetos del navegador (BOM)

4) OBJETO LOCATION

Objeto de window que representa la localización (URL) de la aplicación web. A esta propiedad se puede acceder desde el objeto window y desde el objeto document, ya que ambas son referencias al mismo objeto.

La propiedad más importante es href que devuelve la url de la aplicación. También redirecciona a la URL indicada asignando una nueva dirección:

```
console.log(location.href); // Devuelve la URL de la aplicación
location.href = "https://www.google.es"; // Carga google.es
```



Objeto location
https://developer.mozilla.org/es/docs/Web/API/Window/location

2. Modelo de objetos del navegador (BOM)

Algunas de las propiedades más útiles del objeto location son:

da los paráme tros a partir de una URL

Propiedad	Utilidad
hash	Obtiene # seguido del marcador indicado en la URL. Por ejemplo, si la URL es https://paraninfo.es/libros/tecnicos#javascript devolvería **#javascript.**
host	Una cadena que contiene *nombrehost:puerto* de la URL. Por ejemplo: **paraninfo.es:886.**
origin	Contiene la forma canónica del origen de una URL: protocolo, puerto y nombre de *host*.
password	Almacena la contraseña que vaya en la URL, si la hay.
pathname	Saca la ruta de ficheros a partir del *host* en la URL: por ejemplo, **/libro/html/modelo.html.**
port	Obtiene el número del puerto de la URL.
protocol	Captura el esquema del protocolo de la URL, incluyendo los dos puntos: por ejemplo, **http:.**
search	Contiene la cadena de consulta de una URL. Por ejemplo: si la URL es https://paraninfo.es/buscar.php?autor=Gonzalez&keyword=js, almacenaría **?autor=Gonzalez&keyword=js.**
username	Almacena el nombre de usuario que vaya en la URL, si lo hay.

2. Modelo de objetos del navegador (BOM)

Ejemplo 4: OBJETO LOCATION

Escribe un programa que incluye un botón que al pulsarlo ingrese en Youtube, pero primero muestre una ventana de confirmación preguntando si queremos acceder:

a) Si el usuario pulsa sí, le redirecciona a Youtube

b) En caso contrario mostrará un mensaje

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>
<body>
    <button onClick="verificarEntrada()">Ingresar a Youtube</button>
    <script>
        function verificarEntrada() {
            if (window.confirm('Desea salir del sitio e ingresar a Youtube?')) {
                window.location = 'https://www.youtube.com/';
            } else {
                window.alert('No hay problema');
            }
        }
    </script>
</body>

</html>
```

2. Modelo de objetos del navegador (BOM)

5) OBJETO HISTORY

Objeto destinado a almacenar el historial de páginas visitadas.

Algunas de las propiedades más útiles del objeto history son:

- **back() –** retrocede a la página anterior. Es similar a la flecha de atrás del navegador.
- **forward()** – avanza a la página siguiente almacenada en la caché de la máquina. Es similar a la flecha de avance de navegador.
- **go()** que permite moverse por el historial.

  Por ejemplo:

  go(-1) llevaría a la página anterior, es similar a back()

  go(1) llevaría a la página siguientes, es similar a forward()

Objeto location
https://developer.mozilla.org/es/docs/Web/API/Window/history

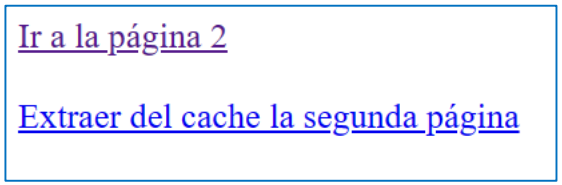
Manipulando el historial de navegación
https://developer.mozilla.org/es/docs/Web/API/History_API

2. Modelo de objetos del navegador (BOM)

Ejemplo 5: OBJETO HISTORY I

Crea dos páginas de tal forma que:

1) La primera debe tener dos links
   I. El primer link debe ser un enlace normal a la segunda página
   II. El segundo link, tiene dos posibilidades:
      a) Si no hay ninguna página en la caché muestra un alert avisándolo
      b) Si hay alguna página en la caché, va a esa página
2) En la segunda página sólo hay un link que sirve para retroceder a la primera página utilizando el método go



Página 1

Página 2

Página 1

```html
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>
<body>

    <a href="pagina2.html">Ir a la página 2</a><br><br>
    <a href="javascript:avanzar()">Extraer del cache la segunda página</a>
    <script>
        function avanzar() {
            if (window.history.length>1){
                window.history.go(1);
            } else {
                alert('No hay otra página en la cache hacia adelante');
            }
        }
    </script>
</body>
</html>
```

Página 2

```html
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
</head>
<body>
    </head>
    <body>
        <a href="javascript:retroceder()">Extraer del cache la primera página</a>
        <script>
            function retroceder() {
                window.history.go(-1);
            }
        </script>
    </body>
</html>
```

2. Modelo de objetos del navegador (BOM)

6) TEMPORIZADORES

El objeto window también ofrece algunos métodos que nos permiten gestionar el tiempo en el navegador.

Tenemos cuatro métodos:

1. setTimeout()
2. setInterval()
3. clearTimeout()
4. clearInterval()

1.- setTimeout()

Método que permite ejecutar un código una vez que expira el intervalo de tiempo dado. Como entrada recibe el código a ejecutar y el tiempo en milisegundos. Puede incluir parámetros adicionales.

```
function greet(name){
     console.log("Welcome to my blog " + name)
  }

setTimeout(greet, 5000, "Elon")
```

Imprimirá “Welcome to my blog Elon” pero tardará 5 segundos (5000 milisegundos) en hacerlo (después de ejecutar el código).

Además, incluye un parámetro adicional donde se almacena el nombre.

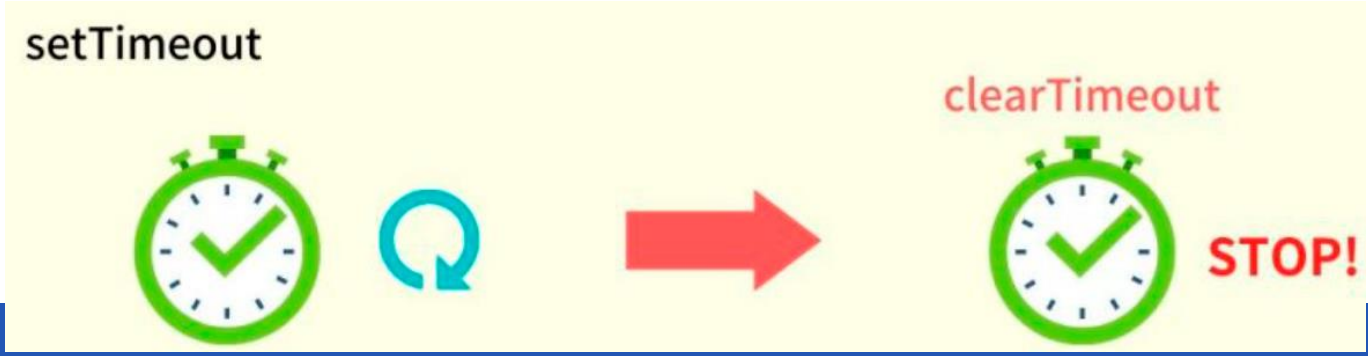
2. Modelo de objetos del navegador (BOM)

2.- clearTimeout()

Método que evita que se ejecute la función que proporcionamos en setTimeout, es decir borra la función dada en el setTimeout antes de que expire el temporizador.

```
const timeoutID = setTimeout(myGreeting, 5000);

function myGreeting() {
  console.log("Good Morning")
}

function stopFunction() {
  clearTimeout(timeoutID);
}

//stopFunction()
```

Imprimirá “Good Morning” después de 5 segundos. Sin embargo, no se imprimirá nada si descomentamos stopFunction().



2. Modelo de objetos del navegador (BOM)

3.- setInterval()

Método que permite ejecutar una función repetidamente. El intervalo de tiempo entre cada repetición lo proporciona setInterval.

```
function greet(){
     console.log("Welcome to my blog")
  }

setInterval(greet, 1000)
```

Imprimirá “Welcome to my blog” cada 1 segundo.

4.- clearInterval()

Método que cancela una acción repetitiva definida con setInterval()

Imprimirá una cuenta regresiva de 6 a o y en 0 imprimirá BangBang.

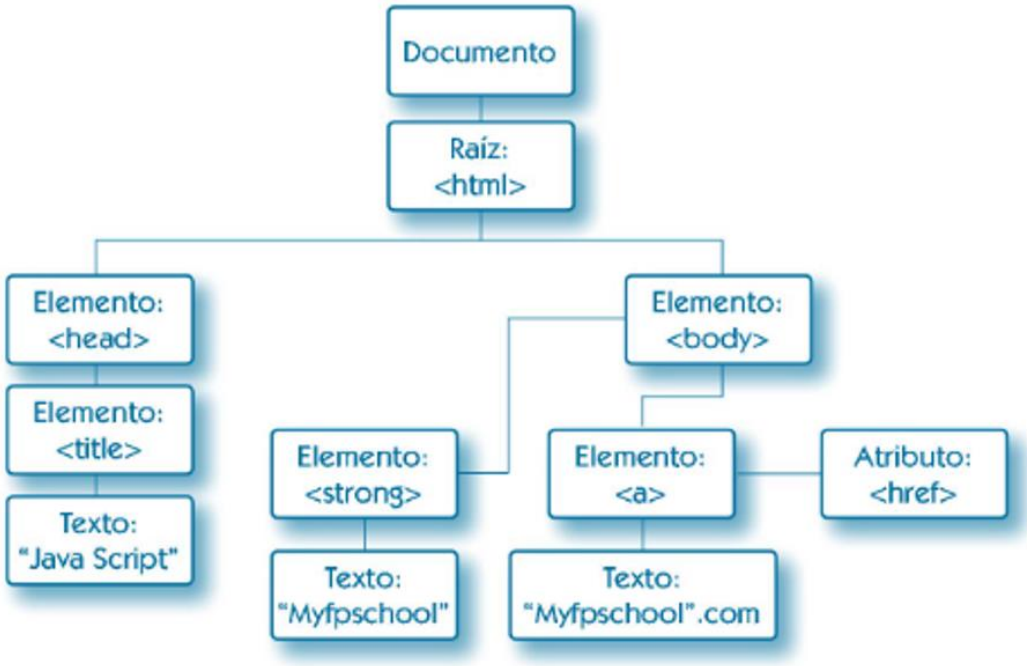
```
let countDown = num => {intervalID = setInterval(() =>
{num = num - 1
if(num === 0){
console.log("BangBang")}
else if(num === -1){
clearInterval(intervalID)}
else{
console.log(num)}}, 1000)}

countDown(6)
```

3. Modelo de objetos del documento (DOM)

El **DOM** permite que los programas accedan y actualicen dinámicamente el contenido, la estructura y el estilo de un documento.

Dentro de una página HTML, JS permite:

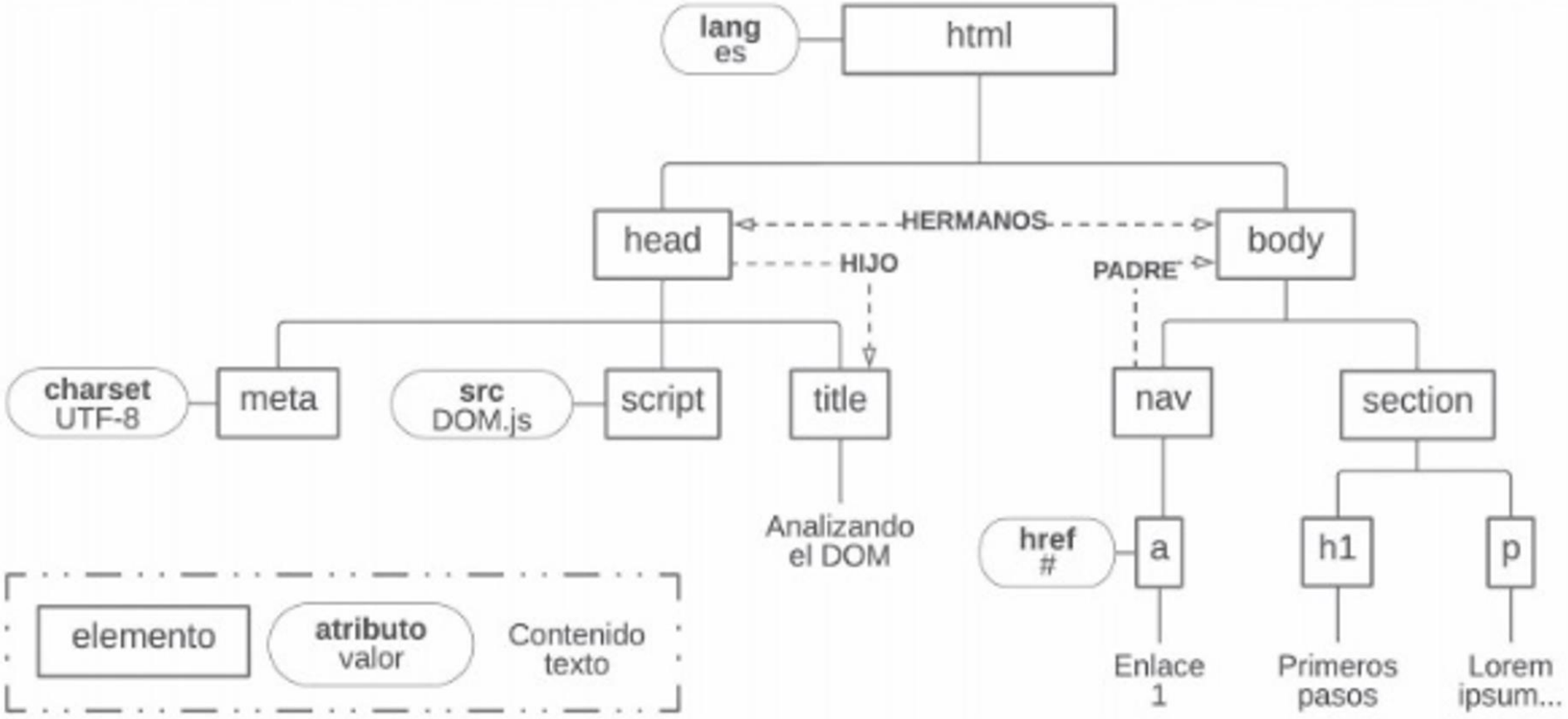
1) Cambiar todos los elementos
2) Cambiar todos los atributos
3) Cambiar todos los estilos CSS
4) Eliminar elementos y atributos existentes
5) Agregar nuevos elementos y atributos
6) Reaccionar a todos los eventos
7) Crear nuevos eventos



El DOM tiene una estructura de árbol, el objeto JS que contiene toda la información es **document**, que será el punto de partida para seleccionar y modificar elementos de las páginas.

3. Modelo de objetos del documento (DOM)

Árbol con relaciones de parentesco entre sus nodos:



Vemos que existen distintos tipos de nodos. Para conocer qué tipo de nodo se está tratando se utiliza la propiedad **nodeType**:

```
console.log(document.nodeType); // Escribe 9
```

3. Modelo de objetos del documento (DOM)

Los posibles valores de **nodeType** son:

Valor	Nodo	Constante asociada
1	Elemento	ELEMENT_NODE
2	Atributo*	ATTRIBUTE_NODE
3	Texto	TEXT_NODE
4	CDATA*	CDATA_SECTION_NODE
5	Referencia a entidad*	ENTITY_REFERENCE_NODE
6	Entidad*	ENTITY_NODE
7	Instrucción de procesado	PROCESSING_INSTRUCTION_NODE
8	Comentario	COMMENT_NODE
9	Documento	DOCUMENT_NODE
10	Tipo de documento	DOCUMENT_TYPE_NODE
11	Fragmento de código	DOCUMENT_FRAGMENT_NODE
12	Anotación*	NOTATION_NODE

*Actualmente obsoletos, no se recomienda su uso en sitios web nuevos.

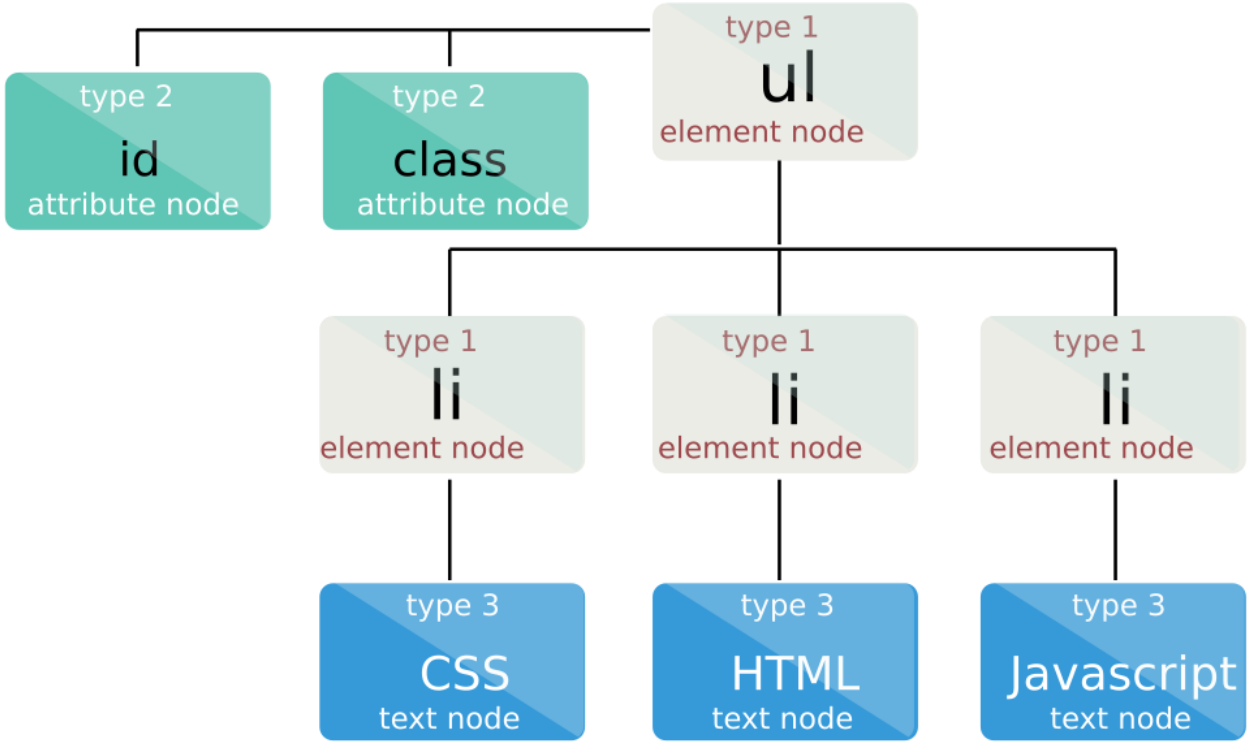
3. Modelo de objetos del documento (DOM)

Asociadas a **nodeType** hay las siguientes propiedades:

- **nodeName** – nombre del nodo
- **nodeValue** – valor del nodo. Devuelve null cuando el nodo es el documento entero, un fragmento, el tipo del documento, un elemento, una referencia a entidad o una anotación.

El objeto **document** ofrece múltiples formas para seleccionar nodos del DOM:

a) Identificadores
b) Etiquetas
c) Clases
d) Selectores CSS



3. Modelo de objetos del documento (DOM)

a) IDENTIFICADORES

getElementById – permite seleccionar nodos por medio del atributo id de sus elementos. Si lo encuentra, devuelve el nodo y si no, null.

getElementById
https://developer.mozilla.org/es/docs/Web/API/Document/getElementById

HTML

```
<html>
  <head>
    <title>Ejemplo getElementById</title>
  </head>
  <body>
    <p id="para">Cualquier texto acá</p>
    <button onclick="changeColor('blue');">Azul</button>
    <button onclick="changeColor('red');">Rojo</button>
  </body>
</html>
```

JS

```
function changeColor(newColor) {
  var elem = document.getElementById("para");
  elem.style.color = newColor;
}
```

3. Modelo de objetos del documento (DOM)

b) ETIQUETAS

getElementByTagName – permite seleccionar nodos por medio del nombre de su etiqueta. Devuelve una lista de nodos (NodeList).

getElementByTagName
https://developer.mozilla.org/es/docs/Web/API/Document/getElementsByTagName

```
let equipo = document.getElementsByTagName("span");
console.info(equipo);
console.log(equipo.length);
console.log(equipo[1]);
```

```
                                                          modelo.js:2
▼HTMLCollection(3) [span#CEO, span#CTO, span#CIO, CEO: span#CEO, CTO:
 span#CTO, CIO: span#CIO] ℹ
  ▶0: span#CEO
  ▶1: span#CTO
  ▶2: span#CIO
  ▶CEO: span#CEO
  ▶CIO: span#CIO
  ▶CTO: span#CTO
    length: 3
  ▶[[Prototype]]: HTMLCollection
3                                                         modelo.js:3
<span id="CTO">Lucía Valdivia.</span>                     modelo.js:4
>
```

El contenido de un NodeList, se le pueden aplicar algunas utilidades de los arrays: equipo.length cantidad de elemento, acceder a los elementos con corchetes.

No se puede utilizar slice, join o sort. Para hacerlo, se puede convertir un NodeList en un array puro mediante …(propagación)

3. Modelo de objetos del documento (DOM)

c) CLASES

getElementByClassName – permite seleccionar nodos por el valor de su atributo class. Devuelve una lista de nodos (NodeList).

getElementByTagName
https://developer.mozilla.org/es/docs/Web/API/Document/getElementsByClassName

```
<span id="CEO" class="jefatura">Joaquín Luque.</span>
<span id="CTO" class="direccion">Lucía Valdivia.</span>
<span id="CIO" class="direccion">Marisa Pons.</span>
```

```
let directores = document.getElementsByClassName("direccion");
// devuelve el segundo y tercer span
```

3. Modelo de objetos del documento (DOM)

d) SELECTORES CSS

Existen varias alternativas para seleccionar nodos usando selectores CSS, pero el más importante es **querySelectorAll**. Hay que indicarle un selector CSS como parámetro y siempre volverá una lista de nodos que cumplen con los criterios de la selección.

querySelectorAll
https://developer.mozilla.org/es/docs/Web/API/Document/querySelectorAll

```
<span id="CEO" class="jefatura">
        <a class="god_link" href="#">Joaquín Luque.</a>
</span>
<span id="CTO" class="direccion">
        <a class="top_link" href="#">Lucía Valdivia.</a>
</span>
<span id="CIO" class="direccion">
        <a class="top_link" href="#">Marisa Pons.</a>
</span>
```

```
let enlaces = document.querySelectorAll("span > a");
let primerDirector = document.querySelectorAll("a.top_link")[0];
let errorInducido = document.querySelectorAll("span > p");
console.info(enlaces);
console.info(primerDirector);
console.info(errorInducido);
```

```
▶ NodeList(3) [a.god_link, a.top_link, a.top_link]
  <a class="top_link" href="#">Lucía Valdivia.</a>
▶ NodeList []
>
```

3. Modelo de objetos del documento (DOM)

MOVERSE POR EL DOM

Todos los elementos del DOM tienen asociada la propiedad **childNodes** que permite acceder a la lista de sus nodos hijos.

Las propiedades más interesantes para navegar por el DOM son:

childNodes
https://developer.mozilla.org/es/docs/Web/API/Node/childNodes

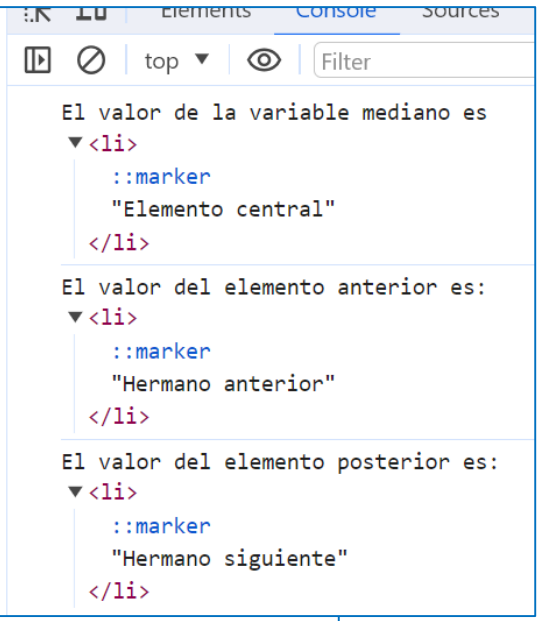
Propiedad	Utilidad
parentNode	Selecciona el nodo padre.
childElementCount	Devuelve el número de elementos hijos.
firstChild	Selecciona el primer nodo hijo.
firstElementChild	Selecciona el primer elemento hijo.
lastChild	Selecciona el último nodo hijo.
lastElementChild	Selecciona el último elemento hijo.
previousSibling	Selecciona el nodo hermano anterior.
previousElementSibling	Selecciona el elemento hermano anterior.
nextSibling	Selecciona el nodo hermano siguiente.
nextElementSibling	Selecciona el elemento hermano siguiente.

3. Modelo de objetos del documento (DOM)

Ejemplo 6: HERMANOS

Crea una lista ordenada de tres elementos, selecciona el elemento del medio y muestra en consola su hermano anterior y siguiente.

1. Hermano anterior
2. Elemento central
3. Hermano siguiente




```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <title>Ejemplo de JavaScript</title>
5       <meta charset="UTF-8">
6   </head>
7       <body>
8           <ol>
9               <li>Hermano anterior</li>
10              <li>Elemento central</li>
11              <li>Hermano siguiente</li>
12          </ol>
13          <script>
14              let mediano = document.getElementsByTagName("li")[1];
15              console.log('El valor de la variable mediano es ',mediano);
16              console.log('El valor del elemento anterior es: ',mediano.previousElementSibling);
17              console.log('El valor del elemento posterior es: ',mediano.nextElementSibling);
18          </script>
19      </body>
20  </html>
```

3. Modelo de objetos del documento (DOM)

MANIPULACIÓN DE ELEMENTOS

Cuando se obtiene el contenido de los elementos, hay que diferenciar entre las propiedades de:

- **textContent** – obtiene el texto sin tener en cuenta las otras posibles etiquetas que contiene.
- **innerHTML** – obtiene el contenido completo, incluyendo las etiquetas que contenga.

```
<section>
        Manipular elementos <i>parece</i> sencillo<br />
        pero <strike>debe</strike> puede complicarse bastante.
</section>
```

```
console.log(document.getElementsByTagName("section")[0].textContent);
console.log(document.getElementsByTagName("section")[0].innerHTML);
```

```
        Manipular elementos parece sencillo
        pero debe puede complicarse bastante.

        Manipular elementos <i>parece</i> sencillo<br>
        pero <strike>debe</strike> puede complicarse bastante.
>
```

NOTA: esto también hay que tenerlo en cuenta al modificar el contenido, ya que si usamos textContent las etiquetas no se considerarán HTML, sin embargo, si utilizamos innerHTML sí.

3. Modelo de objetos del documento (DOM)

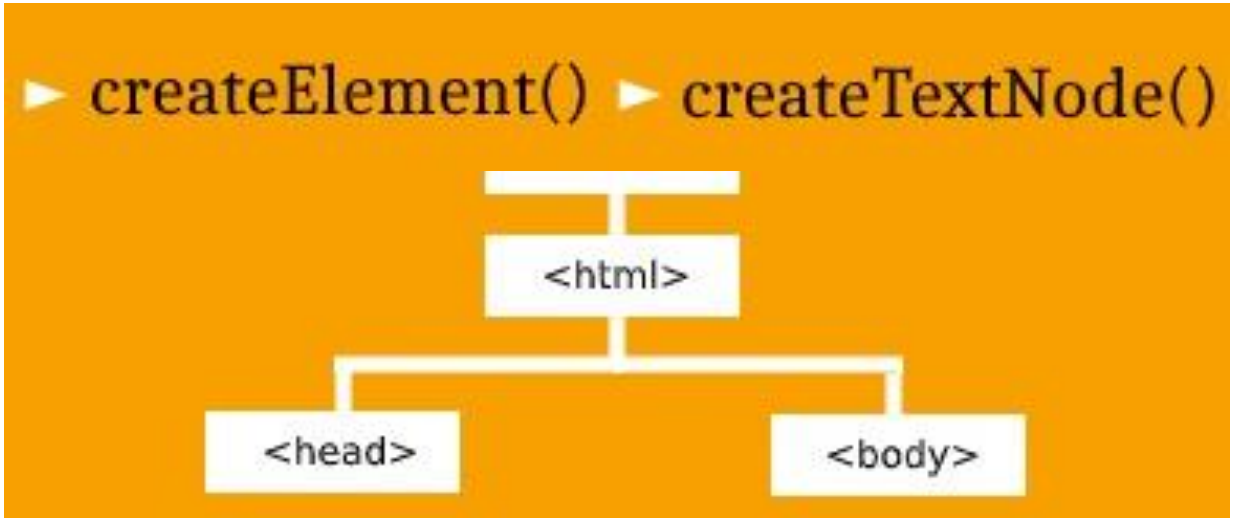
CREACIÓN DE ELEMENTOS

Los métodos que permiten crear elementos nuevos son:

- **createElement** – crea nodos de tipo elemento.
- **createTextNode** – crea nuevos nodos de texto.

Para poder visualizar ambos en el DOM, hay que indicar en qué posición del árbol hay que colocarlos:

```
let elementoNuevo = document.createElement("p");
let nodoTextoNuevo = document.createTextNode("Texto del nodo");
```



3. Modelo de objetos del documento (DOM)

Para insertar un nuevo nodo en el árbol se puede hacer de varias formas:

1) **appendChild** – añade el nodo al final de los hijos
2) **insertBefore** – añade el nodo en una posición determinada. Incluye dos parámetros, el primero el nodo a añadir y el segundo el nodo hijo que quedaría por debajo.
3) **replaceChild** – inserta el nodo reemplazando al que ocupa esa posición

```
<ul id="lista">
        <li>Primer elemento hijo</li>
        <li>Segundo elemento hijo</li>
</ul>
```

```
let nuevoElemento = document.createElement("li");
nuevoElemento.innerHTML = "Tercer elemento";
document.getElementById("lista").appendChild(nuevoElemento);
```

- Primer elemento hijo
- Segundo elemento hijo
- Tercer elemento

```
nodoPadre.replaceChild(nuevoElemento,nodoReferencia);
```

- Primer elemento hijo
- Último elemento insertado

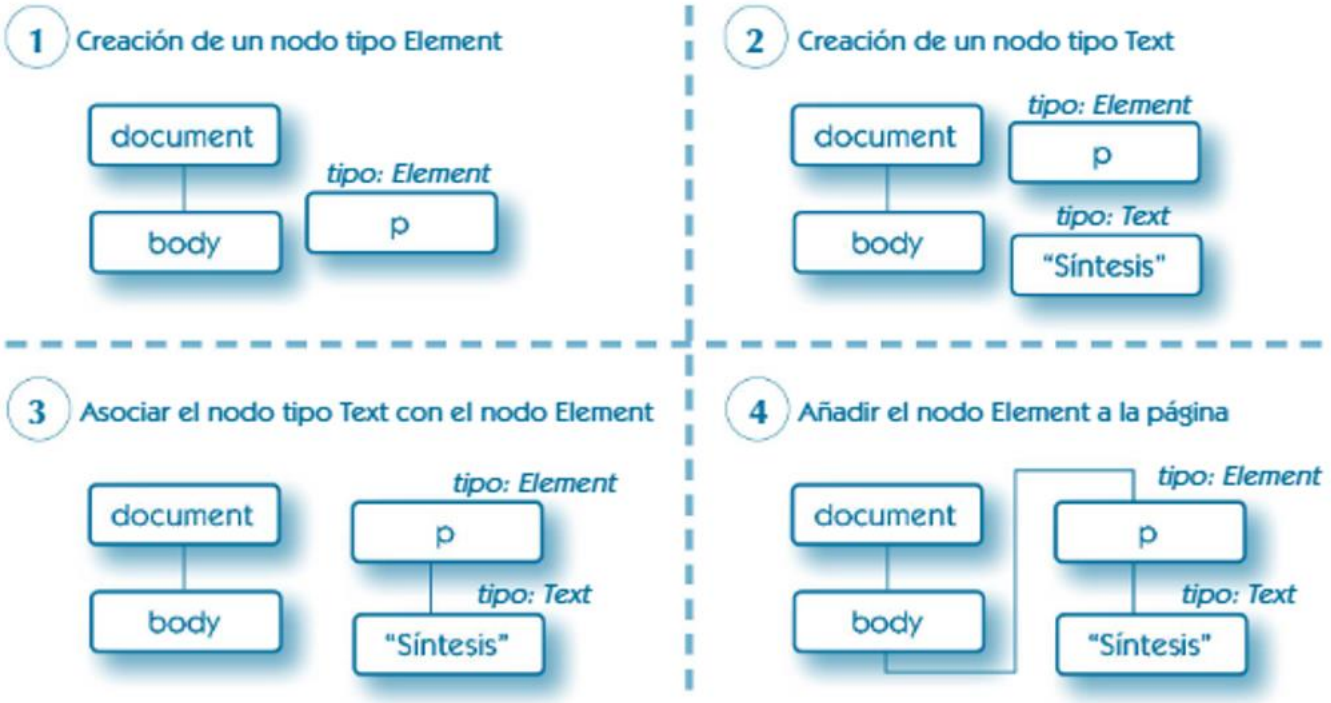
```
// crear el nodo
let nuevoElemento = document.createElement("li");
nuevoElemento.innerHTML = "Último elemento insertado";
// obtener referencia a la posición
let nodoReferencia = document.querySelectorAll("li:nth-of-type(2)")[0];
// obtener el nodo padre
let nodoPadre = nodoReferencia.parentNode;
// insertar el elemento
nodoPadre.insertBefore(nuevoElemento,nodoReferencia);
```

- Primer elemento hijo
- Último elemento insertado
- Segundo elemento hijo

3. Modelo de objetos del documento (DOM)

Pasos para crear un nuevo nodo:

1) **PASO 1 -** Crear un nuevo nodo de tipo Element
2) **PASO 2 -** Crear un nuevo nodo de tipo Text (alberga el contenido del elemento anterior)
3) **PASO 3 -** Vincular el nodo Text al nodo Element
4) **PASO 4 -** Vincular el nodo Element a la página de tal manera que el nodo esté en el lugar correspondiente donde se quiera insertar el elemento



3. Modelo de objetos del documento (DOM)

El código es:

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");
// Crear nodo de tipo Text
var contenido = document.createTextNode("Síntesis");
// Vincular el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);
// Vincular el nodo Element como hijo de la página
document.body.appendChild(parrafo);
```

En este ejemplo, se han utilizado las siguientes funciones del DOM:

- *createElement()*. Para crear un nodo de tipo Element.
- *createTextNode()*. Para crear un nodo de tipo Text.
- *appendChild()*. Para vincular un nodo hijo a un nodo padre.

3. Modelo de objetos del documento (DOM)

ELIMINAR NODOS

Para eliminar nodos se utiliza **removeChild**, indicando el nodo a eliminar:

```
nodoPadre.removeChild(nodoReferencia);
```

Element
https://developer.mozilla.org/es/docs/Web/API/element

MANIPULAR ATRIBUTOS

- **hasAttribute** – comprueba si un atributo existe. Devuelve true si existe y false en caso contrario.

```
document.getElementsByTagName("ul")[0].hasAttribute("id"); // true
document.getElementsByTagName("ul")[0].hasAttribute("class"); // false
```

- **attributes** – muestra una lista completa de atributos de un elemento. Devuelve un mapa de nodos con nombre (**NamedNodeMap**), donde cada elemento del mapa contiene el nombre y el valor del atributo, no es un array, pero es posible recorrerlo como si lo fuera.

```
<img src="#" alt="texto alternativo" title="titulo" />
```

```
let atributos = document.getElementsByTagName("img")[0].attributes;
for (let atributo of atributos) {
        console.log(`[Nombre,Valor]:[${atributo.name},${atributo.value}]`);
}
```

```
[Nombre,Valor]:[src,#]
[Nombre,Valor]:[alt,texto alternativo]
[Nombre,Valor]:[title,titulo]
```

3. Modelo de objetos del documento (DOM)

- **getAttribute** – permite acceder al valor de un atributo utilizando su nombre.

```
document.getElementsByTagName("img")[0].getAttribute("title");
// devuelve título
```

- **setAttribute** – permite modificar el valor de un atributo. Hay que indicar el nombre del atributo y el nuevo valor.

```
document.getElementsByTagName("img")[0].setAttribute("title","nuevo título");
```

- **removeAttribute** – permite eliminar un atributo indicando el nombre:

```
document.getElementsByTagName("img")[0].removeAttribute("title");
```

- **toggleAttribute** – permite eliminar, añadir y volver a eliminar un atributo y así sucesivamente en función de las acciones que realice el usuario en la aplicación web.

  Por ejemplo: **input.toggleAttribute(“disable”)** ocultará el atributo **disabled** si está presente, o lo insertará si no lo está.

3. Modelo de objetos del documento (DOM)

MODIFICAR ESTILOS

- **style** – propiedad que se asocia a un elemento y que contiene todas sus propiedades CSS. De esta forma, se pueden modificar todos los estilos de cualquier elemento HTML. No permite obtener las propiedades CSS que se aplican a un elemento desde hojas de estilo externas.

```
let ul = document.getElementById("lista");
ul.style.border = "1px solid lightblue";
```

- **getComputedStyle** – permite ver los estilos que se le están aplicando a un elemento desde una hoja de estilos externa, pero no permite modificarlos.

```
console.log(window.getComputedStyle(ul).border);
// escribe: 1px solid rgb(173, 216, 230)
```

NOTA: hay que tener cuidado con la modificación de propiedades CSS desde JS ya que genera un código difícilmente mantenible.

- **className** – permite asignar clases CSS a un elemento.

```
let ul = document.getElementById("lista");
ul.className = "bordeCeleste";
```

3. Modelo de objetos del documento (DOM)

- **classList** – permite obtener una lista de todas las clases que se aplican a un elemento. Proporciona un objeto que se puede recorrer como un array, aunque no lo es. incluye métodos como:
  - **add –** para añadir clases
  - **remove –** para eliminar clases
  - **toggle –** si la clase existe la elimina y devuelve false, si no, la añade y devuelve true.
  - **contains –** comprueba si la clase existe en el atributo de clase del elemento.
  - **replace –** reemplaza una clase existente por otra nueva.

```
let ul = document.getElementById("lista");
ul.className = "bordeCeleste letraBlanca fondoGris";
for (let clase of ul.classList)
        console.log(clase);
```

classList
https://developer.mozilla.org/es/docs/Web/API/Element/classList

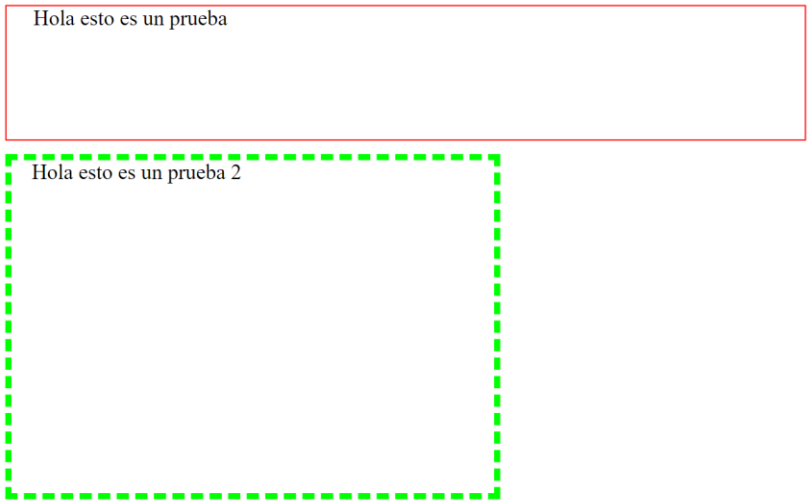
3. Modelo de objetos del documento (DOM)

Ejemplo 7: CSS

Crea un HTML que incluya dos divs, al primero se le debe aplicar la siguiente CSS:

```
div{
    width: 50%;
    height:100px;
    border: 1px solid #FF0000;
    padding-left: 20px;
    margin-bottom: 10px;
}
```

El segundo debe modificar el estilo usando sólo JS y debe tener los siguientes valores: 30%, 250px, 5px dashed #00FF00, 15px y 25px.




```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
    <style>
        div{
            width: 50%;
            height:100px;
            border: 1px solid #FF0000;
            padding-left: 20px;
            margin-bottom: 10px;
        }
    </style>
</head>
    <body>
        <div>Hola esto es un prueba</div>
        <div id="mi_Caja">Hola esto es un prueba 2</div>
        <script>
            //Obtener la caja
            const miCaja = document.getElementById("mi_Caja")
            //Modificar estilos
            miCaja.style.width = "30%";
            miCaja.style.height = "250px";
            miCaja.style.border = "5px dashed #00FF00"
            miCaja.style.paddingLeft = "15px";
            miCaja.style.marginBottom = "25px";
        </script>
    </body>
</html>
```

3. Modelo de objetos del documento (DOM)

MANIPULACIÓN DE ATRIBUTOS DATA

ATRIBUTOS DATA – atributos creados por los programadores en un elemento para almacenar cualquier dato que deba utilizarse con JS.

```
<ul id="precios" data-divisa="euro" data-iva="21">
        <li>14.95</li>
        <li>95.99</li>
</ul>
```

En este caso se utilizan los data para mejorar la presentación de los datos, ya que recoge no sólo el contenido de los elementos li, sino el % de iva que se aplica y la moneda en la que están expresados los precios.

Los atributos de data, aunque sean personalizados por los programadores podemos recogerlos, recorrerlos y conocer sus valores:

- **dataset –** contiene una lista de propiedades con los nombres de cada atributo data y su valor (su estructura es un mapa).

```
let ul = document.getElementById("precios");
console.info(ul.dataset);
console.log(ul.dataset.divisa);
console.log(ul.dataset.iva);
ul.dataset.iva = 10;
console.log(ul.dataset.iva);
```

```
▸ DOMStringMap {divisa: 'euro', iva: '21'}
euro
21
10
>
```

En este ejemplo se muestra la estructura interna de dataset, el acceso a los atributos data y la modificación de su valor.

3. Modelo

Ejemplo 8: ENLACES

Crea una página que contenga un menú básico de navegación con 5 enlaces.

Asigna a dos de ellos atributos class="enlace" y al resto no.

Luego recorre todo el menú y comprueba si existe el atributo class.

Si existe, debes cambiar el valor del atributo a "nuevoEnlace".

Si no existe debes ponerle el valor "enlace".

Las CSS de enlace y nuevoEnlace son:

```css
.enlace{
    color: red;
    font-weight: bold;
}
.nuevoEnlace{
    color: green;
    font-weight: bold;
    font-style: italic;
}
```

Enlace 1 *Enlace 2* Enlace 3 *Enlace 4* Enlace 5

```html
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de JavaScript</title>
    <meta charset="UTF-8">
    <style>
        .enlace{
            color: red;
            font-weight: bold;
        }
        .nuevoEnlace{
            color: green;
            font-weight: bold;
            font-style: italic;
        }
    </style>
</head>
<body>
    <nav>
        <a href="#">Enlace 1</a>
        <a href="#" class="enlace">Enlace 2</a>
        <a href="#">Enlace 3</a>
        <a href="#" class="enlace">Enlace 4</a>
        <a href="#">Enlace 5</a>
    </nav>
    <script>
        let enlaces = document.getElementsByTagName("a");
        for (enlace of enlaces){
            if (enlace.hasAttribute('class')){
                enlace.setAttribute("class","nuevoEnlace");
            }
            else{enlace.setAttribute("class","enlace");}
        }
    </script>
</body>
</html>
```

4. Cookies

COOKIES – pequeños ficheros (≤ 4kB) que las aplicaciones web almacenan en el navegador del usuario para que en futuras sesiones se pueden recuperar los datos que almacenan.

Las cookies sirven para recordar información del usuario.

Http es un protocolo sin estado, es decir, no conserva información entre peticiones. Por tanto, si no se utilizan las cookies, habría cierta información para la usabilidad de la aplicación que habría que solicitar constantemente al usuario, por ejemplo: idioma de la web, esquema de color, fecha de la última visita, etc.

- **document.cookie** – objeto JS para trabajar con cookies. Permite definir y obtener cookies propias (máximo 20 por dominio). Para crear una cookie en el navegador del usuario, sólo hay que asignar al objeto **document.cookie** una cadena de caracteres con la forma **clave=valor**:

```
document.cookie = "idioma=es";
document.cookie = "esquemaColor=dark";
document.cookie = "haVotadoEncuesta=si";
```

4. Cookies

El valor de una cookie siempre tiene que estar correctamente codificado, puesto que se envía en la cabecera HTTP. Para evitar problemas se debería hacer:

```
var usuarioCookie="Dimas"
document.cookie="usuario="+ encodeURIComponent(usuarioCookie);
```

- **encodeURIComponent()** – codifica en HTML los caracteres especiales como , / ? : @ & = + $ #
- **decodeURIComponent()** – decodifica en HTML los caracteres especiales como , / ? : @ & = + $ #

Normalmente, las cookies se eliminan cuando se cierra el navegador, pero podemos hacer que una cookie persista:

- **max-age** – permite que las cookies perduren un tiempo determinado expresado en segundos (la cuenta en segundos comienza en el momento en el que se crea la cookie).

```
let limiteEnSegundos = 60*60*24*365; //dentro de un año
document.cookie = `esquemaColor=dark;max-age=${limiteEnSegundos}`;
```

- **expire** – permite que las cookies perduren hasta una fecha determinada (formato GMT)

```
document.cookie="usuario=Dimas;expires=Sat, 6 Aug 2020 12:15:00 GMT";
```

4. Cookies

NOTA: las cookies sólo son accesibles por aplicaciones del mismo dominio donde se crearon y siempre que estén en el mismo directorio. Lo que se puede indicar es la ruta del dominio donde se quiere que se pueda acceder a la cookie, expresando la ruta como valor de la clave path:

```
document.cookie = "esquemaColor=dark;path=/";
```

BORRAR COOKIES

Para borrar una cookie, es suficiente con indicar como fecha de expiración cualquier momento del pasado

```
document.cookie = "usuario=; max-age=0";
```

Los navegadores por defecto imposibilitan la opción de crear cookies en local y trabajar con ellas. Para probar el funcionamiento de las cookies ir a:
https://playcode.io/javascript

Cookies
https://developer.mozilla.org/es/docs/Web/API/Document/cookie

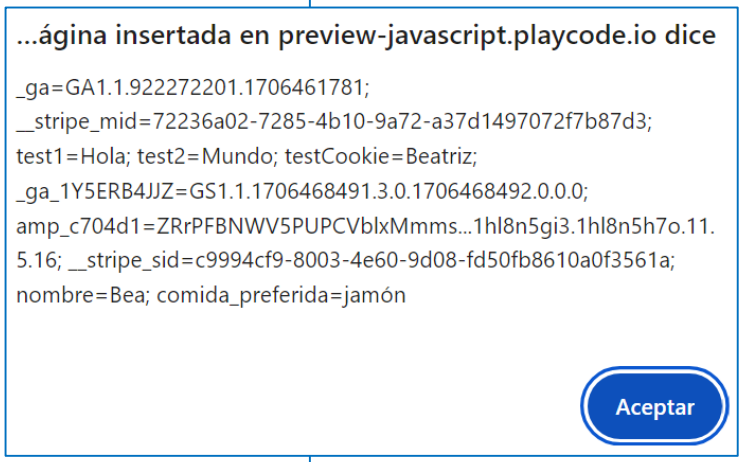
Ejemplo 9: COOKIES

Crea una página que contenga un botón llamado Mostrar cookies, al pulsarlo nos mostrará las cookies.

Se deben crear dos cookies que contengan la siguiente información:

Nombre = Bea Comida_preferida = jamón

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="src/style.css">
  </head>
  <body>
    <h1 id="header"></h1>
    <script>
        document.cookie = "nombre=Bea";
        document.cookie = "comida_preferida=jamón";
        // visualizar: nombre=Bea;comida_favorita=jamón
        function alertCookie() {
          alert(document.cookie);
        }
    </script>
    <button onclick="alertCookie()">Mostrar cookies</button>
  </body>
</html>
```



Gracias