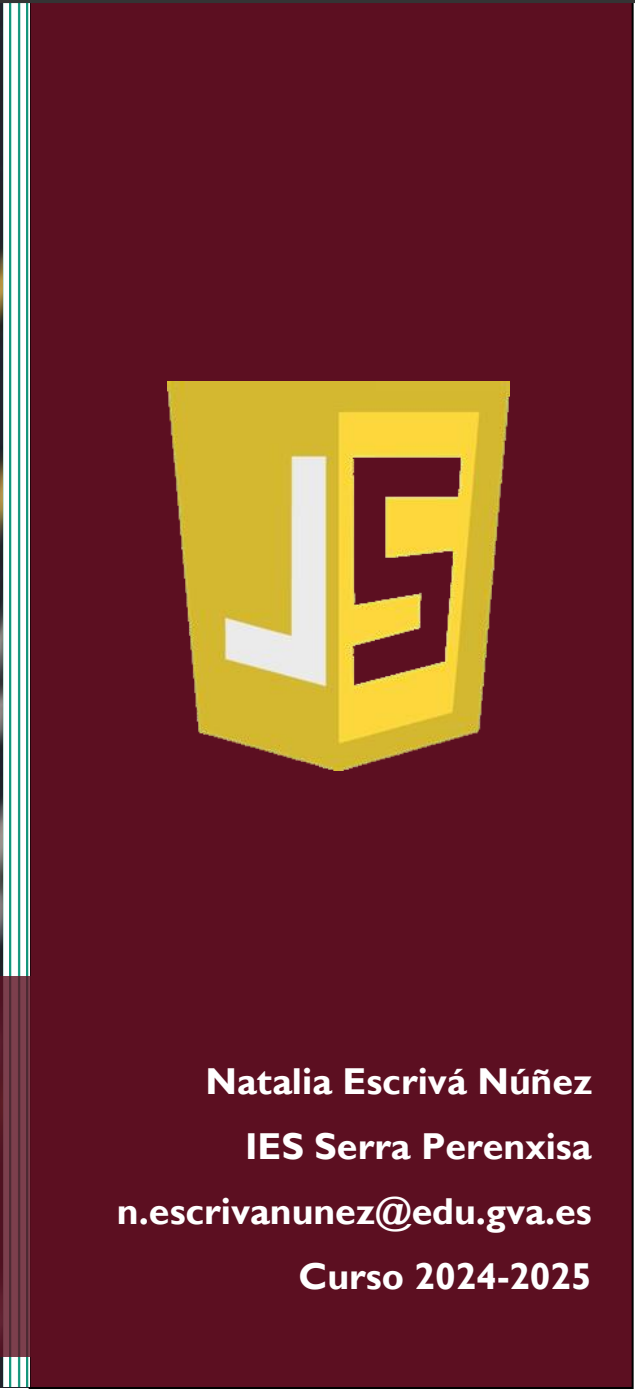


DESARROLLO WEB ENTORNO CLIENTE

2º DAW CFGs

4. JS: MANEJO BOM/DOM



Natalia Escrivá Núñez
IES Serra Perenxisa
n.escrivanunez@edu.gva.es
Curso 2024-2025

IES Serra Perenxisa
n.escrivanez@edu.gva.es
Curso 2024-2025

n.escrivanunez@edu.gva.es

Curso 2024-2025

Curso 2024-2025

CONTENIDO

Contenido	3
1. OBJETOS DEL NAVEGADOR	4
2. OBJETO WINDOW	5
2.1. Propiedades básicas del objeto window	5
2.2. Métodos del objeto window.....	6
3. DOM: OBJETO DOCUMENT	8
3.1. Selección de los elementos del DOM	10
3.2. Modificar textos o imágenes.....	16
3.3. Modificar CSS con JavaScript	19
3.4. Traversing the DOM	25
3.5. Eliminar elementos del DOM.....	30
3.6. Crear elementos en el DOM	32
4. ANEXO: bom.....	37
4.1. Objeto Navigator.....	37
4.2. Objeto Screen.....	38
4.3. Objeto History	38
4.4. Objeto Location.....	39

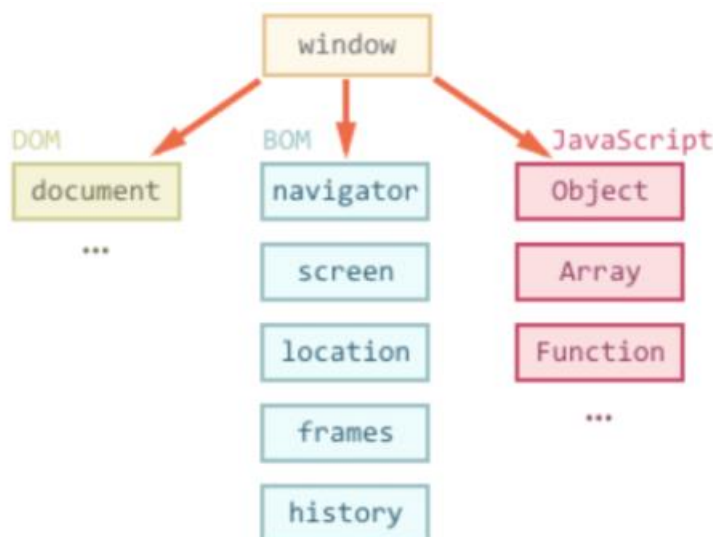


I. OBJETOS DEL NAVEGADOR

Hasta ahora hemos trabajado con objetos nativos (y datos primitivos).

Existen otros objetos que son propios del navegador.

Todos los objetos derivan del **objeto Window**.

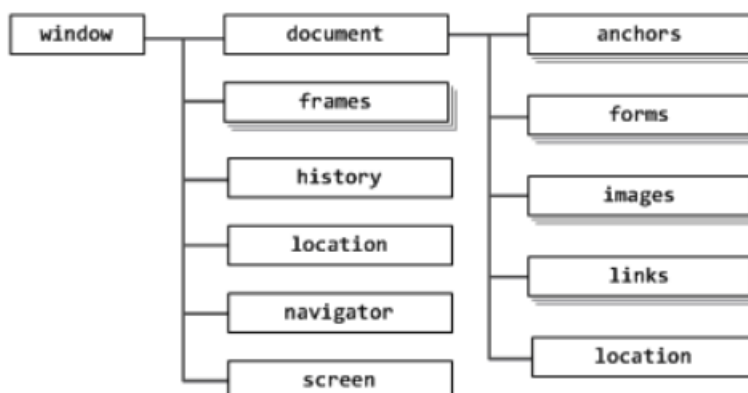


En el esquema anterior se muestran los **frames**. No los vamos a trabajar ya que ya **NO** se crean páginas web a través de **frames**.

A partir de las versiones 3.0 de los navegadores Internet Explorer y Netscape Navigator, se introdujo el concepto de **Browser Object Model o BOM**.

Si bien el BOM **NO** forma parte del lenguaje oficial del lenguaje JavaScript, lo cierto es que los distintos navegadores han igualado la forma de trabajar con estos objetos.

Del objeto Window dependen varios objetos relacionados entre sí.



En el esquema anterior, los objetos mostrados con varios recuadros superpuestos indica que son **arrays**. El resto de los objetos, representados con un único rectángulo, indica que son **objetos simples**.



2. OBJETO WINDOW

Representa una ventana abierta del navegador.

Mediante este objeto, podremos mover, redimensionar y manipular la ventana actual del navegador. Incluso es posible abrir y cerrar nuevas ventanas de navegador (`window.alert`, `window.prompt` o `window.confirm`).

2.1. Propiedades básicas del objeto window

PROPIEDAD	SIGNIFICADO
<code>window.name</code>	Permite conocer el nombre de la ventana
<code>window.outerWidth</code> <code>window.outerHeight</code>	Permiten conocer el tamaño exterior de la ventana CON toolbar y scrollbar
<code>window.innerWidth</code> <code>window.innerHeight</code>	Permiten conocer el tamaño interior de la ventana SIN toolbar y scrollbar: tamaño real disponible en el área de contenido de la ventana teniendo en cuenta el tamaño real
<code>window.scrollX</code> <code>window.scrollY</code>	Permiten conocer el scroll en píxeles en un momento determinado, tanto el horizontal como el vertical.
<code>window.screenX</code> <code>window.screenLeft</code>	Permiten conocer la distancia de la ventana al borde izquierdo de la pantalla: número de píxeles horizontal
<code>window.screenY</code> <code>window.screenTop</code>	Permiten conocer la distancia de la ventana al borde superior de la pantalla: número de píxeles vertical
<code>window.closed</code>	Devuelve un booleano indicando si la ventana ha sido cerrada o no.

```

window.name = "BOM: OBJETO WINDOW";
let texto = "";
//Nombre de la ventana
texto += "<br>Nombre: "+window.name;
//Tamaño de la ventana exterior (con toolbar y scrollbar)
texto += "<br>Ancho externo: "+window.outerWidth;
texto += "<br>Alto externo: "+window.outerHeight;
//Tamaño de la ventana interior (sin toolbar ni scrollbar)
texto += "<br>Ancho interno: "+window.innerWidth;
texto += "<br>Alto interno: "+window.innerHeight;
//Scroll horizontal y vertical --> scrollX y scrollY
texto += "<br>Scroll horizontal: "+window.pageXOffset;
texto += "<br>Scroll vertical: "+window.pageYOffset;
//Distancia de la esquina superior izquierda
texto += "<br>Distancia desde la izquierda: "+window.screenX;
texto += "<br>Distancia desde arriba: "+window.screenY;

document.write(texto);

```

```

Nombre: OBJETO WINDOW
Ancho externo: 772
Alto externo: 740
Ancho interno: 676
Alto interno: 916
Scroll horizontal: 0
Scroll vertical: 0
Distancia desde la izquierda: 4
Distancia desde arriba: 8

```



2.2. Métodos del objeto window

Aquí encontramos distintos tipos de métodos que nos ayudarán a trabajar con nuestras ventanas.

Se puede consultar el funcionamiento en los ficheros de la unidad.

2.2.1. MÉTODOS GENERALES

MÉTODO	SIGNIFICADO
open()	Permite abrir una ventana nueva. Parámetros, aquí
close()	Permite cerrar la ventana concreta.
resizeBy(<pix>,<pix>)	Permite redimensionar una ventana el n° de píxeles indicados. La ventana se “estirará” desde la esquina inferior derecha.
resizeTo(<pix>,<pix>)	Permite redimensionar la ventana al n° de píxeles indicados respecto a su posición actual.
moveBy(<pix>,<pix>)	Permite mover una ventana el número de píxeles indicados respecto a su posición actual.
moveTo(<pix>,<pix>)	Permite mover una ventana a la posición concreta según los píxeles indicados.
focus()	Permite dar el foco a la ventana
print()	Permite imprimir el contenido de la ventana concreta
stop()	Permite parar la carga de la página concreta.

2.2.2. MÉTODOS DE CUADROS DE TEXTO

En este apartado entran los tres métodos que ya hemos ido utilizando como herramientas para interactuar con el usuario.

MÉTODO	SIGNIFICADO
alert()	Muestra un mensaje al usuario, sin recoger valores
prompt()	Muestra un mensaje al usuario y, además, permite que el usuario introduzca un valor en su interior (String). Además, podemos incluir como segundo parámetro un texto por defecto.



confirm()	Muestra un mensaje al usuario y dos botones. Si pulsa "Aceptar", devuelve true y si pulsa "Cancelar" o cierra la ventana, devuelve false.
------------------	---

2.2.3. MÉTODOS RELACIONADOS CON TEMPORIZADORES

El manejo del tiempo es fundamental en la programación de videojuegos, creación de animaciones, publicidad y otras áreas.

MÉTODO	SIGNIFICADO
setTimeout(<función>,ms)	Ejecuta la función pasado el tiempo indicado en el parámetro.
clearTimeout(<variable>)	Si el método setTimeout ha sido asignada a una variable, la podremos parar antes de su ejecución.
setInterval(<función>,ms)	Repite la función pasada como argumento cada intervalo de tiempo pasado en milisegundos.
clearInterval(<variable>)	Cancela la repetición de la función.

```
<button onclick="saludar()">Saludo Ventana</button>
<button onclick="clearTimeout(saludo)"> Cancelar Saludo </button>
```

```
function saludar(){
    miVentana.document.write("<h2>Hola</h2>");
}
```

```
<button onclick="saludoRepe = setInterval(saludar2,3000)">Saludo Repe Ventana</button>
<button onclick="clearInterval(saludoRepe)"> Cancelar Saludo </button>
```

```
function saludar2(){
    miVentana.document.write("<h3>Hola</h3>")
}
```



3. DOM: OBJETO DOCUMENT

DOM (Document Object Model), es el objeto generado por el navegador en el momento en que el documento HTML de una web se carga.

Ese objeto (que desciende del objeto Window), será el que usaremos como base para manipular cualquier página web.

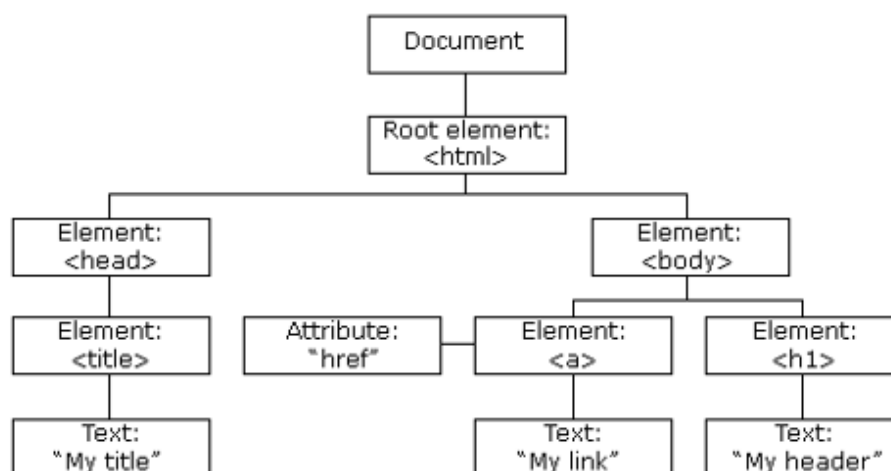
El DOM define la **página web como una estructura organizada que forma un árbol de recorrido**: la raíz sería el HTML y sus elementos estarían distribuidos en **HTML nodes** ("ramas" o hijos). El primer nodo, padre de todos los demás, será **node <html>**, **root node** o **root element**.

Existen **dos tipos de nodos**:

- **Element Nodes**: Se refiere a los <head>, <title>, <body>, <header>, <nav>, <div>, <p>, <button>, <a>,...
- **Text Nodes**: Se refiere al contenido de algunos element nodos (p, h2, button...). Aquí incluimos los saltos de línea. Distinguiremos con los saltos de línea entre:
 - Los encerrados entre etiquetas (
)
 - Los propios de la creación de un documento HTML

```
<body>
  <div id="main">
    <p id="parrafo1">Este es el primer párrafo</p>
    <p id="parrafo2">Este es el segundo párrafo</p>
    <p id="parrafo3">Este es el tercer párrafo</p>
  </div>
  <br/>
```

Los **Element Nodes** pueden tener **atributos** (nos referimos a id, class, src, href...)



En el ejemplo de este apartado (ficheros unidad) encontramos este HTML:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>MiViaje.com</title>
    <link href="https://fonts.googleapis.com/css?family=Lato:400,700,900" rel="stylesheet">
    <link rel="stylesheet" href="css/fontawesome-all.min.css">
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <div class="hero">
      <header class="header contenedor">
        <div class="logo">
          
        </div>
        <nav class="navegacion">
          <a href="#">Vender</a>
          <a href="#">Ayuda</a>
          <a href="#">Registro</a>
          <a href="#">Iniciar Sesión</a>
        </nav>
      </header>

      <div class="contenido-hero contenedor">
        <h1>Encuentra <span>alojamiento</span> para tus próximas vacaciones</h1>

        <form action="/buscador" method="POST" class="formulario formulario-buscar" id="
          <input type="text" name="busqueda" class="busqueda" placeholder="New York, L
          <input type="submit" value="Buscar" id="btn-submit">
        </form>
      </div>
```

Podemos ver que la raíz son las etiquetas `<html>` y sus dos grandes ramas: `<head>` y `<body>`.

Dentro de `<head>` vemos ramas o hijos con la misma jerarquía y, no ocurre igual con el apartado `<body>`, que tiene hijos que, a su vez, tienen otras ramas de jerárquicamente inferiores.

Podremos, partiendo del HTML, **crear, modificar, borrar elementos** del document y también **modificar sus atributos** y, todo esto, de forma **dinámica**.

Para interactuar con la página, tenemos que hacerlo a través de los nodos, obteniendo una referencia a ellos. De esa forma podremos modificar (elementos o atributos) y borrarlos.

También podremos crear y borrar elementos, a través del **DOM Scripting**.

La **referencia** a document sería **window.document** pero podremos hacer **directamente referencia al objeto document y sus elementos**.



3.1. Selección de los elementos del DOM

Tenemos a nuestro alcance varios métodos para seleccionar elementos del DOM.

Debemos tener en cuenta que los elementos del DOM son objetos.

Sabemos que para seleccionar elementos haremos referencia siempre al objeto document.

Los **resultados de la selección** de elementos se presentan como:

- **HTMLCollection:** es como un array de elementos HTML que corresponden a esa selección
- **NodeList:** es como un array de nodos (incluye saltos de línea...) que corresponden a esa selección.

3.1.1. SELECCIÓN POR CLASE

Seleccionaremos elementos del documento que tengan asignada una clase determinada de CSS.

El selector que utilizaremos es `getElementsByClassName()` que recibe como argumento un string con el valor que buscamos.

Si la clase NO existe, devuelve un HTMLCollection vacío, NO dará error.

NO es el selector más actual.

Ejemplo → Seleccionamos la clase “contenedor”

```
<!DOCTYPE html>
<html lang="en">
<head> ...
</head>
<body>

  <div class="hero">
    <header class="header contenedor"> ...
    </header>

    <div class="contenido-hero contenedor"> ...
    </div>
    <!--.hero-->

    <main class="contenido contenedor"> ...
    </main>

    <footer id="footer" class="footer">
      <div class="contenedor"> ...
      </div>
    </footer>
    <a href="#footer" class="btn-flotante">Idioma y Moneda</a>

    <script src="js/01-scripts.js"></script>
    <script src="js/02-scripts.js"></script>
    <script src="js/03-scripts.js"></script>

  </body>
</html>
```

```
const contenedor = document.getElementsByClassName('contenedor');
console.log(contenedor);
```

```
HTMLCollection(4) [header.header.contenedor, div.contenido-hero.contenedor,
main.contenido.contenedor, div.contenedor]
  ▶ 0: header.header.contenedor
  ▶ 1: div.contenido-hero.contenedor
  ▶ 2: main.contenido.contenedor
  ▶ 3: div.contenedor
  length: 4
  [[Prototype]]: HTMLCollection
```



3.1.2. SELECCIÓN POR ETIQUETA

Seleccionaremos elementos del documento que tengan una etiqueta HTML concreta.

El selector que utilizaremos es `getElementsByTagName()` que recibe como argumento un string con el valor que buscamos.

Si la clase NO existe, devuelve un HTMLCollection vacío, NO dará error.

NO es el selector más actual.

Ejemplo → Seleccionamos los párrafos

```
const parrafosTodos = document.getElementsByTagName('p');
console.log(parrafosTodos);
```

```
01-scripts.js:14
HTMLCollection(37) [p.categoria.concierto, p.titulo,
p.precio, p.categoria.concierto, p.titulo, p.precio,
p.categoria.clase, p.titulo, p.precio,
p.categoria.paseo, p.titulo, p.precio,
p.categoria.hospedaje, p.titulo, p.precio,
▶ p.categoria.hospedaje, p.titulo, p.precio,
p.categoria.hospedaje, p.titulo, p.precio, p.titulo,
p.titulo, p.titulo, p.titulo, p.categoria.clase,
p.titulo, p.precio, p.categoria.concierto, p.titulo,
p.precio, p.categoria.clase, p.titulo, p.precio,
p.categoria.paseo, p.titulo, p.precio]
```

3.1.3. SELECCIÓN POR IDENTIFICADOR

Seleccionaremos UN NODO del documento a través del valor del atributo `id`.

El selector que utilizaremos es `getElementById()` que recibe como argumento un string con el valor que buscamos.

Nos **devolverá el primer elemento** que coincida con ese id. Tendremos en cuenta que las buenas prácticas hacen que nunca se repitan los ids en un HTML.

Si el id NO existe, devuelve **null**, NO dará error.

NO es el selector más actual.

Ejemplo → Seleccionamos el elemento con id = "formulario"

```
<form action="/buscador" method="POST" class="formulario formulario-buscar" id="formulario" >
  <input type="text" name="busqueda" class="busqueda" placeholder="New York, Londres, Roma, Guadalajara">
  <input type="submit" value="Buscar" id="btn-submit">
</form>
```

```
const formulario = document.getElementById('formulario');
console.log(formulario);
```

```
01-scripts.js:12
▶ <form action="/buscador" method="POST" class="formulario formulario-buscar" id="formulario"> ...
  </form> flex
```



3.1.4. SELECCIÓN POR SELECTORES CSS: UN ELEMENTO

Los tres selectores anteriores han sido los utilizados hasta el momento, pero ya NO es la forma más actual de seleccionar elementos, aunque debemos conocerlas ya que veremos ese código en muchos proyectos.

El selector que utilizaremos en este caso es `querySelector()` que recibe como argumento un string con el **valor de cualquier selector válido para CSS**, de hecho, su sintaxis es muy parecida a la de las hojas de estilos de CSS.

Nos **devolverá el primer elemento** que coincida con ese selector.

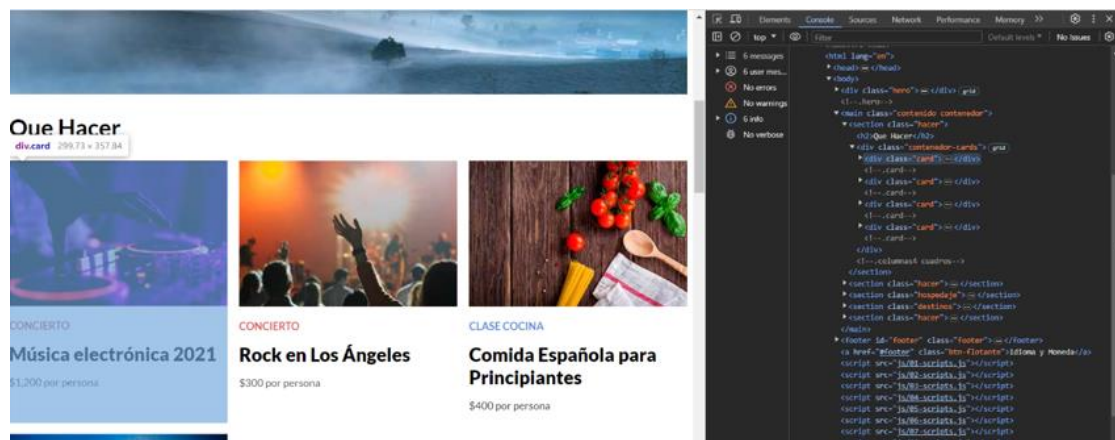
Si el selector NO existe, NO devuelve nada, sin dar error.

Usaremos:

- El punto para hacer referencia a las clases
- El prefijo hash para hacer referencia a los identificadores
- Su nombre para hacer referencia a las etiquetas

Ejemplo 1 → Varios elementos: solo devuelve el primero.

En nuestro proyecto ejemplo, existen varios elementos cuya clase es “card”. Encontramos hasta 15.



```
const card = document.querySelector('.card');
console.log(card);
```

Si nos fijamos en la sintaxis vemos el punto delante de la clase (tal y como la referenciamos en un documento de estilos de CSS).

Ejemplo 2 → Seleccionamos un elemento hijo.

En nuestro proyecto ejemplo, vemos un `div class = “contenedor-cards premiun”` con un elemento hijo:

- `div class = “info”`, que, a su vez, tiene dos elementos hijos:
 - Texto en `<h3>`
 - Link `class = “boton btn-mi-viaje”`





Seleccionamos los tres elementos que tienen class (dejaremos el `<h3>` sin seleccionar).

```
const premium = document.querySelector('section.hacer .premium');
console.log(premium);
const premiumInfo = document.querySelector('.premium .info');
console.log(premiumInfo);
const enlaceInfo = document.querySelector('.premium .info .btn-mi-viaje');
console.log(enlaceInfo);
```

```
><div class="contenedor-cards premium">...</div> grid
```

```
><div class="info">...</div> flex
```

```
<a href="#" class="boton btn-mi-viaje">Explorar alojamientos </a>
```

Ejemplo 3 → Seleccionamos elementos concretos de entre los que tienen el mismo nombre de clase (primero, segundo y último)

En nuestro proyecto ejemplo, vemos que la sección con class “Hospedaje” hay dos elementos hijos:

- Texto `<h2> Hospedaje </h2>`
- Un div con class “contenedor-cards” que, a su vez tiene 3 divs con class “card” cada una.



CASA COMPLETA - 2 CAMAS

Casa completa con todos los servicios y 2 recamaras

\$3,200 por noche



1 CUARTO CON 2 CAMAS

1 Cuarto con 2 camas y alberca

\$2,200 por noche



CABAÑA COMPLETA - 4 CAMAS

Cabaña en Bosque para 6 personas

\$2,500 por noche

```
> document
  < #document
    <[DOCTYPE html]
    <[html lang="en"]
    <[head]
    <[body]
      <[div class="hero"]
      <[main class="contenido contenedor"]
        <[section class="hacer"]
        <[section class="hacer"]
        <[section class="hospedaje"]
          <[h2>Hospedaje/h2]
            <[div class="contenedor-cards"]
              <[div class="card"]
              <[div class="card"]
              <[div class="card"]
            <[div]
            <[!--.columnas4 cuadros--]
          </section>
        <[section class="destinos"]
        <[section class="hacer"]
      </main>
      <[footer id="footer" class="footer"]
    </body>
  </html>
```



Vamos a seleccionar los elementos por separado (sabemos que este selector únicamente devuelve un elemento).

```
const primerCard = document.querySelector('section.hospedaje .card:first-of-type');
console.log(primerCard);
const segunCard = document.querySelector('section.hospedaje div.card:nth-child(2)');
console.log(segundCard);
const tercerCard = document.querySelector('section.hospedaje div.card:last-of-type');
console.log(tercerCard);
```

```
<div class="card">
  
  <div class="info"> ... </div>
</div>

<div class="card">
  
  <div class="info"> ... </div>
</div>

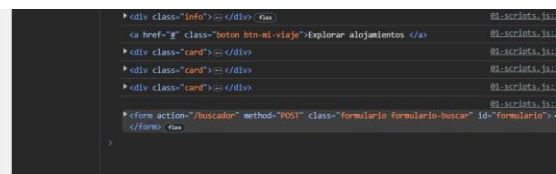
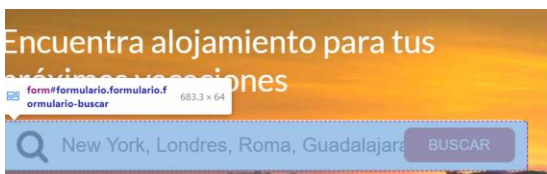
<div class="card">
  
  <div class="info"> ... </div>
</div>
```

Podemos ver la información de los cards seleccionados: el primero, segundo y último div con clase “card” de la sección “Hospedaje”.

Ejemplo 4 → Seleccionamos un elemento a través de su id (buscamos el formulario)

```
<form action="/buscador" method="POST" class="formulario formulario-buscar" id="formulario" >
  <input type="text" name="busqueda" class="busqueda" placeholder="New York, Londres, Roma, Guadalajara">
  <input type="submit" value="Buscar" id="btn-submit">
</form>
```

```
const formularioQuery = document.querySelector('#formulario');
console.log(formularioQuery);
```



Podemos ir “ajustando” la selección haciendo referencia a clases/elementos padres e hijos.

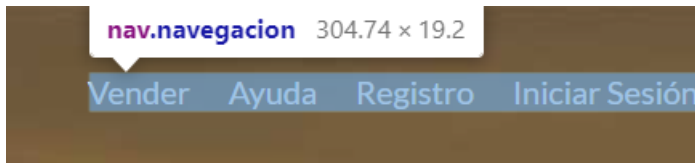
Podemos también seleccionar elementos concretos del **HTMLCollection**.

Podemos incluso mezclar clases e id (aunque no tenga mucho sentido, ya hemos dicho que los id deben ser únicos).

```
const formularioQuery = document.querySelector('.contenido-hero #formulario');
console.log(formularioQuery);
```



Ejemplo 5 → Seleccionamos un elemento según su etiqueta HTML y sus hijos.
En nuestro proyecto ejemplo, tenemos un `<nav>` con 4 menús de navegación.



```
<nav class="navegacion">
  <a href="#">Vender</a>
  <a href="#">Ayuda</a>
  <a href="#">Registro</a>
  <a href="#">Iniciar Sesión</a>
</nav>
```

```
<div class="hero">
  <header class="header contenedor">
    <div class="logo">...
    </div>
    <nav class="navegacion">
      <a href="#">Vender</a>
      <a href="#">Ayuda</a>
      <a href="#">Registro</a>
      <a href="#">Iniciar Sesión</a>
    </nav>
  </header>
```

Vemos que el elemento `<nav>` tiene una clase, pero podemos seleccionarlo directamente por la etiqueta.

```
const navegacion = document.querySelector('nav');
console.log(navegacion);
```

Podemos también seleccionar el primer y último nodo hijo:

```
const primNav = navegacion.firstElementChild;
console.log(primNav);
const ultNav = navegacion.lastElementChild;
console.log(ultNav);
```

```
<a href="#">Vender</a>
<a href="#">Iniciar Sesión</a>
```

3.1.5. SELECCIÓN POR SELECTORES CSS: VARIOS ELEMENTOS

A través de este selector podremos seleccionar TODOS los elementos del DOM que coincidan con el/los selector/es de CSS.

El selector que usaremos será `querySelectorAll()`

Nos devolverá un **NodeList**, una lista de nodos que coinciden con el argumento del método selector.

Si el elemento NO existe, devolverá un **NodeList** vacío, NO dará error.

También en este caso podemos especificar los selectores indicando padres e hijos.



Ejemplo 1 → Seleccionamos todos los elementos con clase 'card'

En nuestro proyecto de ejemplo, seleccionamos todos los elementos que tienen el atributo **class = 'card'**

```

<main class="contenido contenedor">
  <section class="hacer">
    <h2>¿Te apetece?</h2>
    <div class="contenedor-cards">
      <div class="card">...
    </div>
    <div class="card">...
    </div>
    <div class="card">...
    </div>
    <div class="card">...
    </div>
  </div> <!--.columnas4 cuadros-->
</section>

<section class="hacer">...
</section>

<section class="hospedaje">
  <h2>Hospedaje</h2>
  <div class="contenedor-cards">
    <div class="card">...
    </div>
    <div class="card">...
    </div>
    <div class="card">...
    </div>
  </div> <!--.columnas4 cuadros-->
</section>

```

```

const cardTodos = document.querySelectorAll('.card');
console.log(cardTodos);

```

NodeList(15) [div.card, div.card, div.card, div.card, div.card, div.card, div.card, div.card, div.card, div.card, div.card, div.card, div.card, div.card, div.card]

Este resultado (**NodeList**), lo podremos tratar como un array:

```

for (let card of cardTodos){
  console.log(card);
}

```

3.2. Modificar textos o imágenes

Una vez seleccionados los elementos del DOM, vamos a ver cómo modificarlos a través de JavaScript.

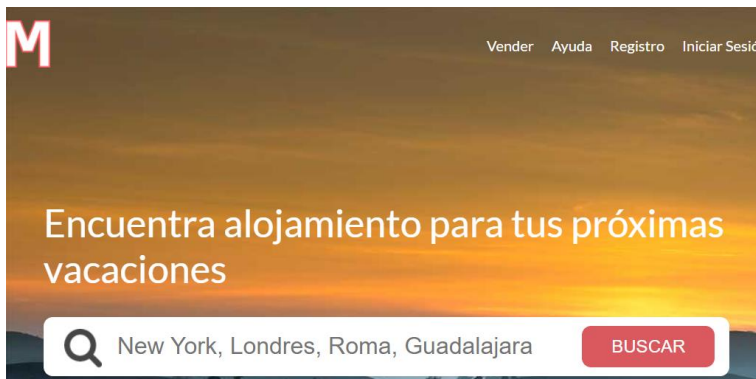
3.2.1. MODIFICAR TEXTOS

Vamos a modificar el texto del encabezado: “Encuentra alojamiento para tus próximas vacaciones”.



Podemos hacerlo de tres formas:

- **innerText** → recoge el texto plano, pero NO literal. Si en CSS la propiedad `visibility: hidden`, NO lo encuentra.
- **textContent** → recoge el texto plano literal. Si en CSS la propiedad `visibility: hidden`, SÍ lo encuentra.
- **innerHTML** → recoge el HTML



```
<div class="contenido-hero contenedor">
  <h1>Encuentra <span> alojamiento </span> para tus próximas vacaciones</h1>
  <form action="/buscador" method="POST" class="formulario formulario-buscar" >
  </form>
</div>
```

Primero seleccionamos el elemento a modificar:

```
const encabezado = document.querySelector('.contenido-hero h1');
console.log(encabezado);
```

Vamos a ver cómo se recoge el texto:

```
console.log(encabezado.innerText);
console.log(encabezado.textContent);
console.log(encabezado.innerHTML);
```

```
Encuentra alojamiento para tus próximas vacaciones
Encuentra  alojamiento  para tus próximas vacaciones
Encuentra <span> alojamiento </span> para tus próximas vacaciones
```

Vemos que en el caso de **innerText** el texto aparece con un formato correcto.

En el caso de **textContent** el texto aparece con dos espacios delante y detrás de la palabra “alojamiento”. Esto es porque en el HTML esos espacios están.

En el caso de **innerHTML** lo que aparece es el propio HTML.



Vamos a modificar el texto:

Ejemplo 1 → innerText

```
document.querySelector('h1').innerText = "Prepara tus vacaciones con innerText";
```

Prepara tus vacaciones con
innerText

Ejemplo 2 → textContent

```
encabezado.textContent = "Busca un HOSPEDAJE con textContent";
```

Busca un HOSPEDAJE con
textContent

Ejemplo 3 → innerHTML

```
encabezado.innerHTML = "<h1>Encuentra <strong> alojamiento </strong> con innerHTML</h1>";
```

Encuentra alojamiento con
innerHTML

The screenshot shows a web application interface with a header and a search bar. The header text is "Encuentra alojamiento con innerHTML". Below the header is a search bar with the text "New York, Londres, Roma, Guadala..." and a "BUSCAR" button. To the right of the interface is a Chrome DevTools console showing the DOM tree. The console highlights the

element, and its innerHTML is shown as "<h1>Encuentra alojamiento con innerHTML</h1>".



3.2.2. MODIFICAR IMÁGENES

Al igual que los textos, también vamos a poder modificar imágenes.

Vamos a acceder a la primera imagen del proyecto de muestra:

```
const imagen = document.querySelector('.card img');
console.log(imagen);
```

Modificamos la **propiedad src** del elemento:

```
imagen.src = 'img/hacer2.jpg';
```



3.3. Modificar CSS con JavaScript

Podemos, a través de JavaScript agregar o quitar clases, modificar el css o los estilos de un elemento.

Como ejemplo, en los formularios cuando dejamos algún campo vacío, y queda resaltado en rojo.

3.3.1. PROPIEDAD STYLE

Debemos saber que, cuando las propiedades de CSS son compuestas (separadas por un guión), en JavaScript se elimina el guión y seguimos la norma de Lower Camel Case:

- background-color → backgroundColor
- justify-content → justifyContent



Para modificar estilos utilizaremos la palabra reservada **style**, seguido de un punto y el estilo a modificar.

Ejemplo → Cambiamos varios atributos de estilo de un elemento

Vamos a cambiar el color del <h1> de nuestro proyecto de muestra. También modificamos el tipo de letra y lo pondremos en mayúsculas.

```
const encabezadoCSS = document.querySelector('h1');
encabezadoCSS.style.backgroundColor = "red";
encabezadoCSS.style.fontFamily = "Georgia";
encabezadoCSS.style.textTransform = "upperCase";
```



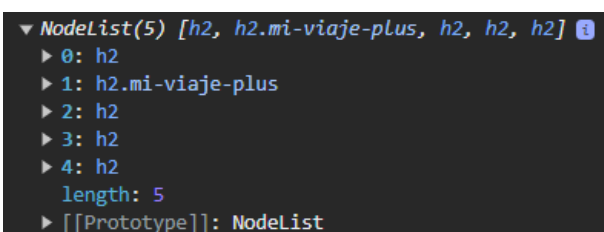
3.3.2. GETATTRIBUTE() / HASATTRIBUTE()

Estos métodos nos ayudan a averiguar detalles sobre los atributos de los elementos:

- **getAttribute()** obtiene el valor de un atributo concreto
- **hasAttribute()** nos devolverá true si el elemento tiene el atributo pasado como argumento.

Ejemplo → Seleccionamos los elementos <h2> y solo mostramos aquellos que tienen el atributo 'class'

```
const h2Todos = document.querySelectorAll('h2');
console.log(h2Todos);
```



Vamos a recorrer el **NodeList** y mostraremos información únicamente de aquellos que tienen la propiedad 'class'.

```
h2Todos.forEach(h2 => {
  if (h2.hasAttribute('class')){
    console.log(`El elemento: "${h2.innerHTML}",
               tiene la clase: ${h2.getAttribute('class')}`);
  }
});
```

```
El elemento: "Presentamos Mivaje.com Plus",
             tiene la clase: mi-viaje-plus
```

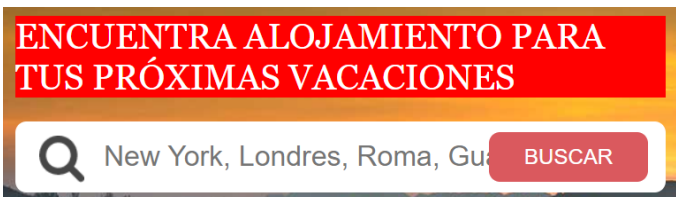
3.3.3. REMOVEATTRIBUTE()

Con el método **removeAttribute()** podremos **eliminar un atributo o propiedad**.

Ejemplo → Eliminamos varios estilos de un elemento.

Si recordamos los estilos que habíamos dado al encabezado...

```
const encabezadoCSS = document.querySelector('h1');
encabezadoCSS.style.backgroundColor = "red";
encabezadoCSS.style.fontFamily = "Georgia";
encabezadoCSS.style.textTransform = "upperCase";
```



Eliminamos la propiedad "style":

```
encabezadoCSS.removeAttribute("style");
```



3.3.4. CREAR Y BORRAR CLASES

El método anterior está bien conocerlo, pero deberemos usarlo en contadas ocasiones.

Esto es porque debemos respetar la independencia de código, y de los estilos se encargan las hojas de estilos, es decir, los ficheros CSS.

Una buena opción es modificar las clases. De esa forma, los estilos se los estaremos dando desde la propia hoja de estilos y mantendremos la coherencia (asociaremos estilos a clases y no a elementos).

Si seleccionamos un elemento de DOM que tiene asignada una clase, podemos ver que tiene dos propiedades:

- **classList**: devuelve TODAS las clases como si fuese un array
- **className**: devuelve el nombre de la/s clase/s (como string)

```
> const main = document.querySelector('main')
> main.classList
< ▶ DOMTokenList(2) ['contenido', 'contenedor', value: 'contenido contenedor']
> main.className
< 'contenido contenedor'
```

En el caso de **classList**, detallamos los distintos **métodos** que podemos usar:

- ✓ **add** (<"nomClaseI">,...,["nomClasen"]): Permite añadir una clase al elemento. Se pueden añadir varias separadas por comas.
- ✓ **remove** (<"nomClaseI">,...,["nomClasen"]): Igual que el anterior pero eliminando las clases.
- ✓ **toggle** (<"Clase">,[forzar]): Con solo un parámetro, se considera la clase. Si el elemento no tiene una clase asignada, se la asigna y, si ya la tiene, se la quita. El segundo parámetro recibe un true que añadirá la clase tanto si el elemento la tiene como si no. Por el contrario, si es false, quita la clase del elemento.
Este método consigue hacer lo que hacen los dos anteriores.
- ✓ **contains** (<"Clase">): Devuelve true si el elemento tiene asignada la clase del parámetro.
- ✓ **replace** (<"vieja">,<"nueva">): Permite cambiar una clase por otra en el elemento.



Ejemplo → Añadir y borrar clases

Seleccionamos el primer elemento cuya clase es “card” para añadir y borrar posteriormente clases.

```
const card1 = document.querySelector('.card');
console.log(card1.classList);
console.log(card1.className);
```

```
▼ DOMTokenList ['card', value: 'card'] ⓘ
  0: "card"
  length: 1
  value: "card"
  ► [[Prototype]]: DOMTokenList
card
```

Añadimos dos clases:

```
card1.classList.add('nueva-clase', 'segunda-clase');
console.log(card1.classList);
console.log(card1.className);
```

```
DOMTokenList(3) ['card', 'nueva-clase', 'segunda-clase',
value: 'card nueva-clase segunda-clase'] ⓘ
  0: "card"
  1: "nueva-clase"
  2: "segunda-clase"
  length: 3
  value: "card nueva-clase segunda-clase"
  ► [[Prototype]]: DOMTokenList
card nueva-clase segunda-clase
```

Borramos ahora la clase “card”:

```
card1.classList.remove('card');
console.log(card1.classList);
console.log(card1.className);
```

```
▼ DOMTokenList(2) ⓘ
  0: "nueva-clase"
  1: "segunda-clase"
  length: 2
  value: "nueva-clase segunda-clase"
  ► [[Prototype]]: DOMTokenList
nueva-clase segunda-clase
```

Vamos a ver la comparativa con y sin la clase “card”:

¿Te apetece?



CONCIERTO
Música electrónica 2021
\$1,200 por persona



CONCIERTO
Rock en Los Ángeles
\$300 por persona

¿Te apetece?



CONCIERTO
Música electrónica 2021
\$1,200 por persona



CONCIERTO
Rock en Los Ángeles
\$300 por persona

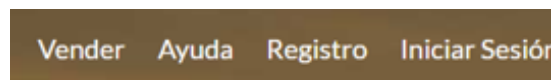
3.3.5. CLASSLIST.TOGGLE()

Este método de **classList** permite añadir al elemento el atributo que se indica si no lo tiene, o lo quitará si ya lo tiene.

La primera vez que se usa, lo añade, la siguiente, la elimina, y repite así el proceso.

Ejemplo → Vamos a modificar la clase a los elementos de menú de navegación superior.

```
const hrefTodas = document.querySelectorAll('.navegacion a');
```



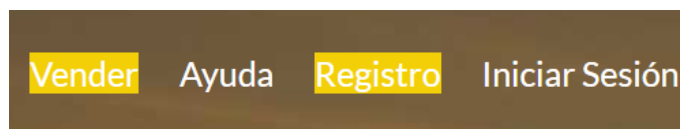
Les vamos a añadir a todos los elementos la clase ".fondoAmarillo":

```
hrefTodas.forEach(element => {
  element.classList.toggle("fondoAmarillo");
});
```



Ahora vamos a quitarle esa clase al segundo y cuarto elemento.

```
for(let i=1; i<=hrefTodas.length; i++){
  if (i === 1 || i === 3){
    hrefTodas[i].classList.toggle("fondoAmarillo");
  }
}
```



3.4. Traversing the DOM

Sabemos que TODOS los elementos del DOM están considerados como nodos y que están conectados, unos dependen de otros.

Hemos visto cómo seleccionar elementos, cómo modificar su aspecto o sus atributos.

Los nodos tienen varias propiedades, entre ellas, destacamos:

- `nodeType`: Nos indica el tipo de nodo:
 - De tipo elemento (1) → las etiquetas HTML
 - De tipo texto (3)
 - De tipo comentario (8)
- `nodeName`: Nos da el tipo de nodo HTML (etiqueta)

Vamos a ver cómo podemos “navegar” a través del DOM o recorrerlo, como si se tratase de un mapa.

3.4.1. DE PADRES A HIJOS

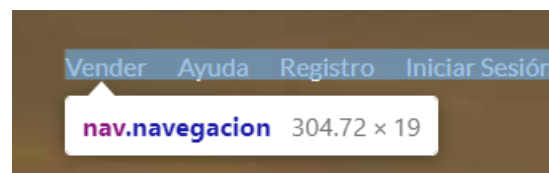
Contamos con dos propiedades que nos van a servir para acceder o trabajar con los nodos hijos de un elemento.

Utilizaremos *childNodes* y *children*

Ejemplo 1 → Acceder a nodos “hijos” de un elemento (nav)

Recorremos la parte superior del proyecto ejemplo. Lo seleccionamos:

```
<header class="header contenedor">
  <div class="logo">...
</div>
  <nav class="navegacion">
    <a href="#">Vender</a>
    <a href="#">Ayuda</a>
    <a href="#">Registro</a>
    <a href="#">Iniciar Sesión</a>
  </nav>
</header>
```



```
const navDOM = document.querySelector('nav');
console.log(navDOM);
```



Si vemos en la consola este elemento, vemos que tiene 4 enlaces (nodos) “hijos”. Si queremos acceder a ellos podemos usar **childNodes**:

```
console.log(navDOM.childNodes);
```

```
▶ NodeList(9) [text, a, text, a, text, a, text, a, text]
```

```
<nav class="navegacion">
  <a href="#">Vender</a>
  <a href="#">Ayuda</a>
  <a href="#">Registro</a>
  <a href="#">Iniciar Sesión</a>
</nav>
```

Nos devuelve un **NodeList** con 9 elementos pero en nuestro HTML SOLO hay 4 links.

Veamos el nombre y tipo del primer elemento del **NodeList**:

```
console.log(navDOM.childNodes[0].nodeType); 3
console.log(navDOM.childNodes[0].nodeName); #text
```

Esto es porque cada salto de línea se considera un elemento de texto. Para que no aparecieran todos, en el HTML tendríamos que poner los links seguidos unos de otros.

```
<nav class="navegacion">
  <a href="#">Vender</a><a href="#">Ayuda</a><a href="#">Registro</a><a href="#">Iniciar Sesión</a>
</nav>
```

```
▶ NodeList(6) [text, a, a, a, a, text]
```

Esto nos desordenaría el código HTML, por lo que podemos utilizar **children**, que nos devolverá los elementos HTML hijos.

```
console.log(navDOM.children);
```

Nos va a devolver un **HTMLCollection** con los 4 elementos reales:

```
▶ HTMLCollection(4) [a, a, a, a]
```

Veamos el nombre y tipo del primer elemento del **HTMLCollection**:

```
console.log(navDOM.children[0].nodeType); 1
console.log(navDOM.children[0].nodeName); A
```

Ejemplo 2 → Acceder a nodos “hijos” de un elemento (card)

En este caso queremos acceder a una card concreta (la foto de la bici)

```
const cardBici = document.querySelectorAll('.card')[3];
console.log(cardBici);
```





PASEO EN BICI

Paseo en las Montañas

\$200 por persona

```
<div class="card">...</div>
<div class="card">...</div>
<div class="card">
  
  <div class="info">
    <p class="categoria paseo">Paseo en Bici</p>
    <p class="titulo">Paseo en las Montañas</p>
    <p class="precio">$200 por persona</p>
  </div>
</div>
<div class="card">...</div>
<div class="card">...</div>
<div class="card">...</div>
<div class="card">...</div>
```

Vamos a ver sus hijos:

```
console.log(cardBici.children);
```

```
HTMLCollection(2) [img, div.info]
  0: img
  1: div.info
  length: 2
```

Veamos los hijos de su hijo "div":

```
console.log(cardBici.children[1].children);
```

```
HTMLCollection(3) [p.categoria.paseo, p.titulo, p.precio]
  0: p.categoria.paseo
  1: p.titulo
  2: p.precio
  length: 3
  [[Prototype]]: HTMLCollection
```

Vamos a navegar hasta el título ("Paseo en las Montañas") y de paso, lo modificamos:

```
cardBici.children[1].children[1].textContent = "Traversing the DOM";
```

Modificamos ahora la foto también recorriendo el DOM:

```
cardBici.children[0].src = 'img/hacer3.jpg';
```



PASEO EN BICI

Traversing the DOM

\$200 por persona



PASEO EN BICI

Traversing the DOM

\$200 por persona



3.4.2. DE HIJOS A PADRES

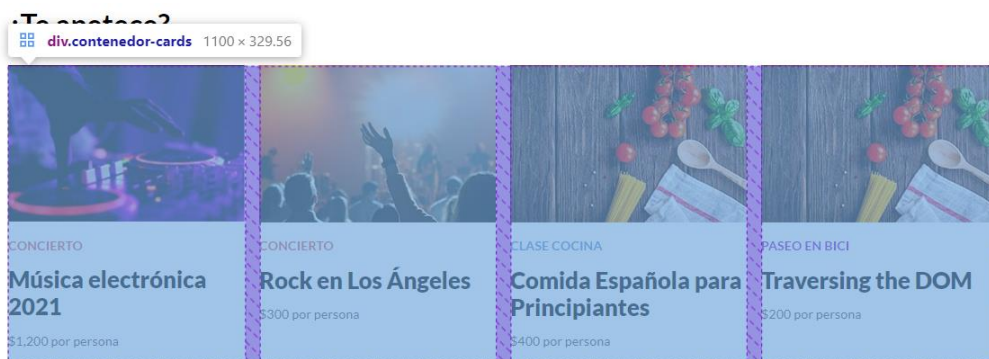
Contamos con dos propiedades que nos van a servir para acceder o trabajar con el nodo padre de un elemento.

Utilizaremos `parentNode` y `parentElement`

Ejemplo I → Acceder al nodo “padre” de un nodo

Seguimos con nuestro proyecto de muestra y vamos a acceder al padre del primer card.

```
const padreCardMusica = document.querySelector('.card').parentElement;
console.log(padreCardMusica);
```

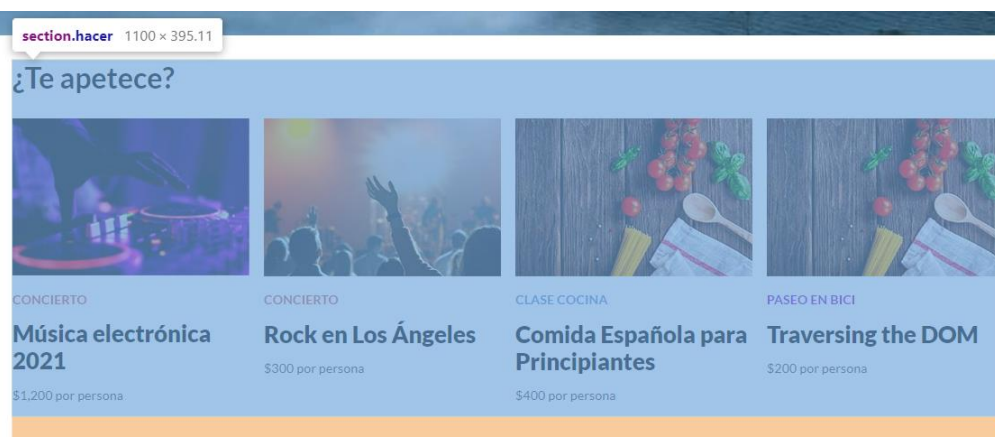


Efectivamente, hemos llegado a su nodo padre:

```
<div class="contenedor-cards"> grid
  > <div class="card"> ... </div>
  > <div class="card"> ... </div>
  > <div class="card"> ... </div>
  > <div class="card"> ... </div>
</div>
```

Vamos a ver el padre del padre:

```
console.log(document.querySelector('.card').parentElement.parentElement);
```



Lo verificamos:

```
<section class="hacer">
  <h2>¿Te apetece?</h2>
  <div class="contenedor-cards">
    <div class="card">...</div>
    <div class="card">...</div>
    <div class="card">...</div>
    <div class="card">...</div>
  </div>
  <!--.columnas4 cuadros-->
</section>
```

Si seguimos seleccionando el elemento padre del elemento padre, llegamos hasta el HTML, el origen:

```
<main class="contenido contenedor">...</main>
<body>...</body>
<html lang="en">
  <head>...</head>
  <body>...</body>
</html>
```

3.4.3. ENTRE HERMANOS

También vamos a poder seleccionar, trabajar y navegar a través de los nodos del mismo rango, es decir, los hermanos.

Para ello usaremos *nextElementSibling* y *previousElementSibling*

Ejemplo I → Acceder al siguiente hermano

Retomamos el ejemplo del primer card y llegamos hasta su siguiente hermano:

```
const hermano1 = document.querySelector('.card').nextElementSibling;
console.log(hermano1);
```

¿Te apetece?



CONCIERTO

**Música electrónica
2021**

\$1,200 por persona

div.card 257 x 324.92



CONCIERTO

Rock en Los Ángeles

\$300 por persona

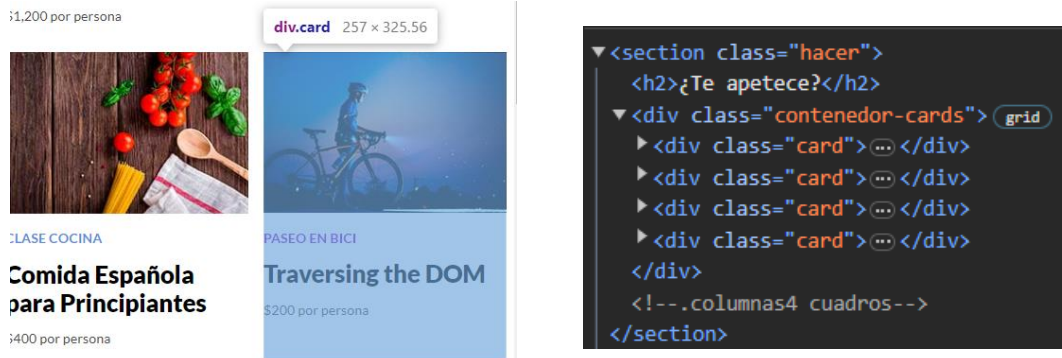
```
<div class="card">
  
  <div class="info">...</div>
</div>
```



Ejemplo 2 → Acceder al hermano anterior

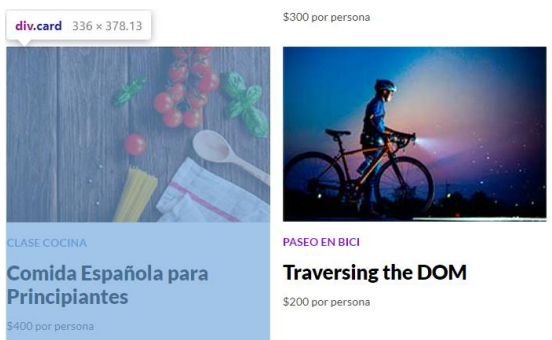
Vamos a seleccionar el cuarto card de la primera sección.

```
const hermano = document.querySelector('section.hacer div.card:nth-child(4)');
console.log(hermano);
```



Seleccionamos ahora el hermano anterior:

```
const hermanoAnt = hermano.previousElementSibling;
console.log(hermanoAnt);
```



3.5. Eliminar elementos del DOM

A veces vamos a tener que eliminar elementos de la página como una foto, un “like” equivocado, un artículo de un carrito...

Podemos eliminar elementos del DOM con dos métodos:

- Eliminando directamente el elemento.
- Eliminando el elemento a través de la referencia del padre.



3.5.1. ELIMINAR EL ELEMENTO DIRECTAMENTE

Vamos a eliminar el primer elemento de los enlaces de navegación.

Primero lo seleccionamos:

```
const primerEnlace = document.querySelector('a');
console.log(primerEnlace);
```

Vender Ayuda Registro Iniciar Sesión

```
<header class="header contenedor"> flex
  <div class="logo"> ... </div>
  <nav class="navegacion"> == $0
    <a href="#">Vender</a>
    <a href="#">Ayuda</a>
    <a href="#">Registro</a>
    <a href="#">Iniciar Sesión</a>
  </nav>
</header>
```

Eliminamos el elemento:

```
primerEnlace.remove();
```

Ayuda Registro Iniciar Sesión

```
<header class="header contenedor"> flex
  <div class="logo"> ... </div>
  <nav class="navegacion"> == $0
    <a href="#">Ayuda</a>
    <a href="#">Registro</a>
    <a href="#">Iniciar Sesión</a>
  </nav>
</header>
```

3.5.2. ELIMINAR EL ELEMENTO A TRAVÉS DE SU PADRE

Podemos eliminar un elemento a través de la referencia de su padre.

Seguimos con el menú de navegación.

Seleccionamos la navegación:

```
const navegacion2 = document.querySelector('.navegacion');
console.log(navegacion2.children);
```

```
▼ HTMLCollection(3) [a, a, a] ⓘ
  ▶ 0: a
  ▶ 1: a
  ▶ 2: a
  length: 3
  ▶ [[Prototype]]: HTMLCollection
```

Borramos el último enlace ("Iniciar Sesión"):

```
navegacion2.removeChild(navegacion2.children[2]);
```

Ayuda Registro

```
<header class="header contenedor"> flex
  <div class="logo"> ... </div>
  <nav class="navegacion"> == $0
    <a href="#">Ayuda</a>
    <a href="#">Registro</a>
  </nav>
</header>
```



3.6. Crear elementos en el DOM

En ocasiones nos podemos encontrar con que tenemos que **generar HTML desde JS**. Por ejemplo, si hacemos un comentario, escribiremos en una cajita de texto y, al clicar en el botón se unirá al resto de comentarios que, a su vez podrán ser comentados.

Esto es solo un ejemplo.

Podremos utilizar distintos métodos.

3.6.1. CREAR UN ELEMENTO AL FINAL DEL ELEMENTO

Este método colocará el nuevo elemento al final de los elementos hijos.

Utilizaremos el método `appendChild()`.

Ejemplo → Creamos un enlace y lo insertamos al final del menú de navegación superior.

Creamos el enlace usando el método `createElement()`. Como argumento le pasaremos el nombre de etiqueta que vamos a crear. Después le añadimos el texto y su link.

```
const newEnlace = document.createElement('a');
newEnlace.textContent = 'Nuevo Enlace';
newEnlace.href = '#';
console.log(newEnlace);
```

```
<a href="#">Nuevo Enlace</a>
```

Se lo añadimos a la navegación superior de la página.

Primero seleccionamos el elemento al que se lo queremos añadir.

```
const navegacion3 = document.querySelector('.navegacion');
console.log(navegacion3);
```

Vender Ayuda Registro Iniciar Sesión

```
<nav class="navegacion">
  <a href="#">Vender</a>
  <a href="#">Ayuda</a>
  <a href="#">Registro</a>
  <a href="#">Iniciar Sesión</a>
</nav>
```



Ahora se lo añadimos.

```
navegacion3.appendChild(newEnlace);
console.log(navegacion3);
```

Vender Ayuda Registro Iniciar Sesión Nuevo Enlace

```
<nav class="navegacion">
  <a href="#">Vender</a>
  <a href="#">Ayuda</a>
  <a href="#">Registro</a>
  <a href="#">Iniciar Sesión</a>
  <a href="#">Nuevo Enlace</a>
</nav>
```

3.6.2. CREAR ELEMENTO EN UBICACIÓN CONCRETA

Este método nos permite controlar más la ubicación del elemento que incorporamos al DOM.

El método que usaremos será `insertBefore()`, que recibe dos argumentos:

- El elemento que queremos insertar
- Donde queremos insertarlo (antes de qué elemento queremos insertarlo).

Ejemplo 1 → Modificamos la posición de un enlace del menú de navegación superior.

Partimos del mismo elemento nav. Vemos sus elementos hijos:

```
console.log(navegacion3.children);
```

Vender Ayuda Registro Iniciar Sesión Nuevo Enlace

```
▼ HTMLCollection(5) [a, a, a, a, a] ⓘ
  ▶ 0: a
  ▶ 1: a
  ▶ 2: a
  ▶ 3: a
  ▶ 4: a
  length: 5
  ▶ [[Prototype]]: HTMLCollection
```

Decidimos cambiar de sitio el nuevo enlace en la segunda posición:

```
navegacion3.insertBefore(newEnlace, navegacion3.children[1]);
console.log(navegacion3);
```

Vender Nuevo Enlace Ayuda Registro Iniciar Sesión

```
<nav class="navegacion">
  <a href="#">Vender</a>
  <a href="#">Nuevo Enlace</a>
  <a href="#">Ayuda</a>
  <a href="#">Registro</a>
  <a href="#">Iniciar Sesión</a>
</nav>
```



Ejemplo 2 → Incluimos un enlace en una ubicación concreta en el menú de navegación.

Creamos otro enlace:

```
const otroEnlace = document.createElement('a');
otroEnlace.textContent = 'Otro';
otroEnlace.href = '#';
navegacion3.insertBefore(otroEnlace, navegacion3.children[3]);
```

Vender Nuevo Enlace Ayuda Otro Registro Iniciar Sesión

3.6.3. CREAR ESTRUCTURA DE ELEMENTOS

Las estructuras que se incorporan al DOM de manera dinámica son muy habituales cuando hacemos consultas a nuestra base de datos o usamos una API.

Vamos a crear un card dentro de nuestro proyecto de muestra.

Este elemento tiene, a su vez, elementos hijos, que también los crearemos.

La estructura es la siguiente:

```
<div class="contenedor-cards">
  <div class="card">
    
    <div class="info">
      <p class="categoria concierto">concierto</p>
      <p class="titulo">Música electrónica 2021</p>
      <p class="precio">$1,200 por persona</p>
    </div>
  </div>
```

Vemos que el elemento tiene:

- Una imagen
- Un div con tres párrafos

Creamos los tres párrafos. Vamos a asignar textos las clases correspondientes a cada párrafo para que los estilos se mantengan.



```
const parrafo1 = document.createElement('p');
parrafo1.textContent = 'CONCIERTO';
parrafo1.classList.add('categoria', 'concierto');
console.log(parrafo1);

const parrafo2 = document.createElement('p');
parrafo2.textContent = 'Festival de Música Rock';
parrafo2.classList.add('titulo');
console.log(parrafo2);

const parrafo3 = document.createElement('p');
parrafo3.textContent = '80€ por persona';
parrafo3.classList.add('precio');
console.log(parrafo3);
```

```
<p class="categoria concierto">CONCIERTO</p>
<p class="titulo">Festival de Música Rock</p>
<p class="precio">$800 por persona</p>
```

Vemos que los tres párrafos están dentro de un `<div class = 'info'>`

Creamos un div con esa característica:

```
const info = document.createElement('div');
info.classList.add('info');
console.log(info);
```

```
<div class="info"></div>
```

Ahora vamos a incluir en el div, los tres párrafos:

```
info.appendChild(parrafo1);
info.appendChild(parrafo2);
info.appendChild(parrafo3);
console.log(info.children);
```

```
HTMLCollection(3) [p.categoria.concierto, p.titulo, p.precio]
  ▶ 0: p.categoria.concierto
  ▶ 1: p.titulo
  ▶ 2: p.precio
  length: 3
  [[Prototype]]: HTMLCollection
```

Vamos a crear la imagen para el card:

```
const imagen = document.createElement('img');
imagen.src = 'img/hacer2.jpg';
console.log(imagen);
```

```

```

Lo que nos falta es el elemento “padre”, el card, que es un `<div class = 'card'>`

```
const divCard = document.createElement('div');
divCard.classList.add('card');
console.log(divCard);
```

```
<div class="card"></div>
```

Por último, debemos añadir imagen y el div ‘info’ con los 3 párrafos al card



```
divCard.appendChild(imagen);
divCard.appendChild(info);
console.log(divCard.children);
```

```
▼ HTMLCollection(2) [img, div.info] ⓘ
  ▶ 0: img
  ▶ 1: div.info
  length: 2
  ▶ [[Prototype]]: HTMLCollection
```

Lo asignamos en el HTML en el primer bloque de cards del proyecto:

```
const padreCard1 = document.querySelector('.contenedor-cards');
console.log(padreCard1);
```

Lo añadimos al inicio, como novedad...

```
padreCard1.insertBefore(divCard, padreCard1.children[0]);
```

¿Te apetece?



CONCIERTO

Festival de Música Rock

80€ por persona



CONCIERTO

Música electrónica 2021

\$1,200 por persona



CONCIERTO

Rock en Los Ángeles

\$300 por persona



CLASE COCINA

Comida Española para Principiantes

\$400 por persona



PASEO EN BICI

Paseo en las Montañas

\$200 por persona

```
▼ <section class="hacer">
  <h2>¿Te apetece?</h2>
  ▼ <div class="contenedor-cards"> grid
    ▶ <div class="card"> ... </div> == $0
    ▶ <div class="card"> ... </div>
    ▶ <div class="card"> ... </div>
    ▶ <div class="card"> ... </div>
    ▶ <div class="card"> ... </div>
  </div>
  <!--.columnas4 cuadros-->
</section>
```

Muchos frameworks y librerías hacen este trabajo de forma automática pero es importante que dominemos estas acciones.



4. ANEXO: BOM

Este apartado de anexos incluirá la explicación y detalle de propiedades y métodos básicos de algunos objetos que se relacionan con BOM.

4.1. Objeto Navigator

Objeto que representa al navegador del usuario que está utilizando la aplicación web.

4.1.1. PROPIEDADES DEL OBJETO NAVIGATOR

MÉTODO	SIGNIFICADO
platform	Devuelve la plataforma del navegador. <code>console.log(navigator.platform); //Win32</code>
onLine / offLine	Devuelve true o false si el usuario está conectado a Internet <code>console.log(navigator.onLine); //true</code>
language	Devuelve el idioma en el que está configurado el navegador <code>console.log(navigator.language); //es-ES</code>
cookieEnabled	Devuelve true o false si están activadas o no las cookies <code>console.log(navigator.cookieEnabled); //true</code>
userAgent()	Permite obtener la cadena de información del navegador <code>console.log(navigator.userAgent)</code> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:106.0) Gecko/20100101 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (Chrome/107.0.0.0 Safari/537.36)
geolocation	Devuelve un objeto que nos servirá para acceder a la posición GPS del usuario (usando la API de geolocalización)
Storage	Devuelve un objeto que nos servirá para trabajar el almacenamiento de datos persistentes (usando la API correspondiente).



4.2. Objeto Screen

Objeto que hace referencia a la pantalla de la ventana que se está visualizando.

4.2.1. PROPIEDADES DEL OBJETO SCREEN

MÉTODO	SIGNIFICADO
width	Devuelve el ancho de la pantalla
height	Devuelve el alto de la pantalla
availWidth	Devuelve el ancho de la pantalla sin la barra de tareas
availHeight	Devuelve el alto de la pantalla sin la barra de tareas
colorDepth	Indica la profundidad en bits de los colores de la pantalla. El número de colores disponibles en la pantalla será 2 elevado a la cantidad devuelta.

4.3. Objeto History

Guarda las URL visitadas por el usuario dentro de una ventana del explorador. NO lo podemos confundir con el historial del navegador.

4.3.1. PROPIEDAD DEL OBJETO SCREEN

PROPIEDAD	SIGNIFICADO
length	Devuelve el número de URLs en el historial de la página.



4.3.2. MÉTODOS DEL OBJETO SCREEN

MÉTODO	SIGNIFICADO
back()	Carga la URL anterior en el historial
forward()	Carga la URL siguiente en el historial
go(<num> <URL>)	<p>Va a una página concreta del historial.</p> <p>Si el número es positivo, irá hacia adelante, el número de páginas indicadas. Lo mismo si es negativo.</p> <p>Si lo que indicamos es una URL, irá a esa URL</p>

4.4. Objeto Location

Este objeto representa la dirección URL de la propia aplicación o página que se está visitando.

4.4.1. PROPIEDADES DEL OBJETO LOCATION

PROPIEDAD	SIGNIFICADO
href	Devuelve la URL de la página
hostname host	<p>Devuelve el nombre del host de la página.</p> <p>Devuelve el nombre del host de la página con el puerto</p>
pathname	Devuelve la ruta de directorios a partir del host de la URL.
protocol	Devuelve el protocolo de la página
hash	Devuelve el ancla o marcador, si lo tiene, de la URL.
origin	Devuelve el protocolo, hostname y puerto.
search	Permite extraer la querystring de la página.



4.4.2. MÉTODOS DEL OBJETO LOCATION

PROPIEDAD	SIGNIFICADO
assign(<url>)	Asigna un nuevo documento a la página.
reload()	Recarga la página.
replace(<url>)	Sustituye la página por otra (desaparece su historial).

