

04. BOM i DOM JAVASCRIPT

IES ESTACIÓ CURS 2021- 2022

Índex

- ▶ 1. Treballant amb el BOM i el DOM.
 - ▶ 1.1 Browser Object Model (BOM).
 - ▶ 1.2 Document Object Model (DOM).

1. Treballant amb el BOM i el DOM

- ▶ Normalment, quan programem en Javascript, ens trobem dins del context d'un navegador.
- ▶ Dins d'aquest context, tenim alguns objectes predefinits i funcions que podem usar per a interactuar amb aquest navegador i amb el document HTML.
- ▶ **1.1 Browser Object Model (BOM).**
 - ▶ **Objecte window**
 - ▶ L'objecte window representa el navegador i és l'objecte principal.
 - ▶ Tot està contingut dins de window (variables globals, l'objecte document, l'objecte location, l'objecte navigation, etc.).
 - ▶ L'objecte window pot ser omès accedint a les seues propietats directament.
 - ▶ No tenen una grandària fixa, per tant podem inicialitzar-lo amb una grandària i després afegir-li més elements.
 - ▶ **Exemples:** *window object, location object, document object, history object, navigator object, screen object.*
 - ▶ https://www.w3schools.com/jsref/obj_window.asp

1. Treballant amb el BOM i el DOM.

► 1.1 Browser Object Model (BOM).

► Objecte window

► Exemples:

```
'use strict';  
// Tamaño total de la ventana (excluye la barra superior del navegador)  
console.log(window.outerWidth + " - " + window.outerHeight);  
window.open("https://www.google.com");  
  
// Propiedades de la pantalla  
console.log(window.screen.width + " - " + window.screen.height); // Ancho de pantalla y alto (Resolución)  
console.log(window.screen.availWidth + " - " + window.screen.availHeight); // Excluyendo la barra del S.O.  
  
// Propiedades del navegador  
console.log(window.navigator.userAgent); // Imprime la información del navegador  
window.navigator.geolocation.getCurrentPosition(function(position) {  
    console.log("Latitude: " + position.coords.latitude + ", Longitude: " + position.coords.longitude);  
});  
  
// Podemos omitir el objeto window (está implícito)  
console.log(history.length); // Número de páginas del history. Lo mismo que window.history.length
```

1. Treballant amb el BOM i el DOM.

▶ 1.1 Browser Object Model (BOM).

▶ Timers (avisadors)

- ▶ Hi ha dos tipus de “timers” que podem crear en Javascript per a executar algun tros de codi en el futur (especificat en mil·lisegons), timeouts i intervals.
- ▶ El primer s'executa només una vegada (hem de tornar a crear-ho manualment si volem que es repetisca alguna cosa en el temps).
- ▶ El segon es repeteix cada X mil·lisegons sense parar (o fins que siga cancel·lat).
- ▶ ***timeout(funció, mil·lisegons)*** → Executa una funció passats un nombre de mil·lisegons.

```
console.log(new Date().toString()); // Imprime inmediatamente la fecha actual  
setTimeout(() => console.log(new Date().toString()), 5000); // Se ejecutará en 5 segundos (5000 ms)
```

- ▶ ***cleartimeout(timeoutId)*** → Cancel·la un timeout (abans de ser anomenat).

```
// setTimeout devuelve un número con el id, y a partir de ahí, podremos cancelarlo  
let idTime = setTimeout(() => console.log(new Date().toString()), 5000);  
clearTimeout(idTime); // Cancela el timeout (antes de que se ejecute)
```

1. Treballant amb el BOM i el DOM.

► 1.1 Browser Object Model (BOM).

► Timers (avisadors)

- **`setInterval(funció, mil·lisegon)`** → La diferència amb `timeout` és que quan el temps acaba i s'executa la funció, es reinicialitza i es repeteix cada X mil·lisegons automàticament fins que nosaltres el cancel·lem.

```
let num = 1;  
setInterval(() => console.log(num++), 1000); // Imprime un número y lo incrementa cada segundo
```

- **`clearInterval(idInterval)`** → Cancel·la un interval (no es repetirà més).

```
let num = 1;  
let idInterval = setInterval(() => {  
  console.log(num++);  
  if(num > 10) { // Cuando imprimimos 10, paramos el timer para que no se repita más  
    clearInterval(idInterval);  
  }  
, 1000);
```

- **`setInterval/setTimeout(nomFunció, mil·lisegons, arguments...)`** → Podem passar-li un nom funció existent. Si es requereixen paràmetres podem establir-los després dels mil·lisegons.

```
function multiply(num1, num2) {  
  console.log(num1 * num2);  
}  
  
setTimeout(multiply, 3000, 5, 7); // Después de 3 segundos imprimirá 35 (5*7)
```

1. Treballant amb el BOM i el DOM.

► 1.1 Browser Object Model (BOM).

► Objecte location

- L'objecte location conté informació sobre la url actual del navegador. Podem accedir i modificar aquesta url usant aquest objecte.

```
console.log(location.href); // Imprime la URL actual
console.log(location.host); // Imprime el nombre del servidor (o la IP) como "localhost" 192.168.0.34
console.log(location.port); // Imprime el número del puerto (normalmente 80)
console.log(location.protocol); // Imprime el protocolo usado (http ó https)

location.reload(); // Recarga la página actual
location.assign("https://google.com"); // Carga una nueva página. El parámetro es la nueva URL
location.replace("https://google.com"); // Carga una nueva página sin guardar la actual en el objeto history
```

► Objecte history

- Per a navegar a través de les pàgines que hem visitat en la pestanya actual, podem usar aquest objecte. History té mètodes bastant útils:

```
console.log(history.length); // Imprime el número de páginas almacenadas

history.back(); // Vuelve a la página anterior
history.forward(); // Va hacia la siguiente página
history.go(2); // Va dos páginas adelante (-2 iría dos páginas hacia atrás)
```

1. Treballant amb el BOM i el DOM.

▶ 1.1 Browser Object Model (BOM).

▶ Objecte history

- ▶ ¿Què ocorre si no volem recarregar la pàgina actual quan anem arrere i avance en el **history**?. Per exemple, la nostra pàgina és controlada per mètodes de Javascript i volem desfer o refer coses quan un usuari prem el botó de darrere o avançar.
- ▶ Per a aconseguir això, hem d'usar **history.pushState()**. Aquests mètodes usen objectes JSON com primer paràmetre amb informació sobre els canvis, i el títol (normalment és ignorat) com segon paràmetre. Podem usar **replaceState()** si no volem que s'emmagatzeme en l'història.
- ▶ En la nostra pàgina web la propietat **window.onpopstate** haurà de ser assignada a una funció. Aquesta funció serà anomenada cada vegada que avancem de pàgina o retrocedim en l'objecte **history**. Rebrà un paràmetre que representa l'esdeveniment de navegació, aquest té una propietat anomenada **state** que conté les dades JSON que es van assignar a l'estat actual. Podem accedir a aqueix estat a través del **history.state** (La primera pàgina tindrà com a estat null).

1. Treballant amb el BOM i el DOM.

► 1.1 Browser Object Model (BOM).

► Objecte history

File: example1.html

```
<!DOCTYPE>
<html>
  <head>
    <title>JS Example</title>
  </head>
  <body>
    <script src="example1.js"></script>
    <button onclick="goBack()">Pàgina anterior</button>
    <button onclick="goNext()">Pàgina siguiente</button>
  </body>
</html>
```

File: example1.js

```
'use strict';
```

```
// Evento para capturar la navegación por el history
window.onpopstate = function (event) {
  if(event.state) {
    console.log("Estoy en la página " + event.state.page);
  } else { // event.state == null si es la primera página
    console.log("Estoy en la primera página");
  }
};
```

```
let page = 1;
```

```
function goBack() {
  history.back();
}
```

```
function goNext() {
  // history.state == null si es la primera página (la siguiente es la página 2)
  let pageNum = history.state?history.state.page + 1:2; // Siguiente página = Página actual + 1.
  history.pushState({page: pageNum}, "");
  console.log("Estoy en la página " + pageNum);
}
```

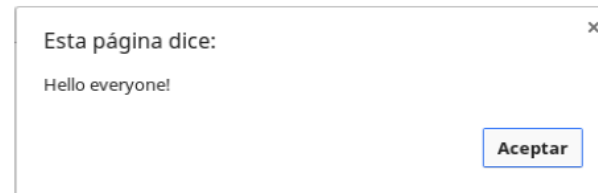
1. Treballant amb el BOM i el DOM.

▶ 1.1 Browser Object Model (BOM).

▶ Dialegs

- ▶ En cada navegador, tenim un conjunt de **diàlegs** per a interactuar amb l'usuari. No obstant això, aquests no són personalitzables i per tant cada navegador implementa el seu a la seua manera. Per això no és recomanable usar-los en una aplicació en producció (uniformitat). En canvi, són una bona opció per a fer proves (en producció hauríem d'usar diàlegs construïts amb HTML i CSS).
- ▶ El **diàleg alert**, mostra un missatge (amb un botó d'Acceptar) dins d'una finestra. Bloqueja l'execució de l'aplicació fins que es tanca.

```
alert("Hello everyone!");
```



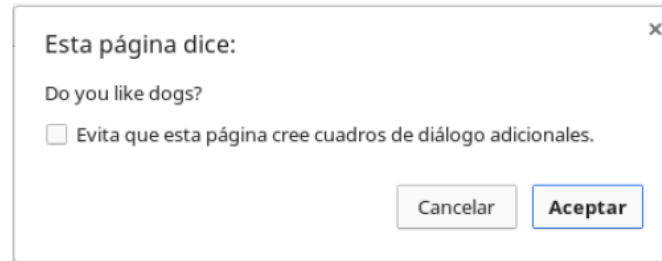
1. Treballant amb el BOM i el DOM.

► 1.1 Browser Object Model (BOM).

► Dialogs

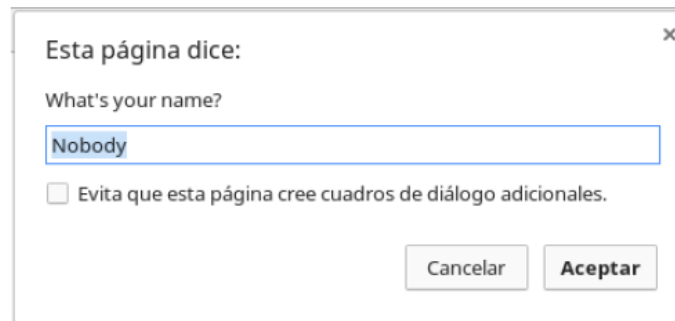
- El **diàleg confirm** és similar, però et retorna un booleà. Té dos botons (Cancel·lar → false, Acceptar → *true). L'usuari triarà entre una de les dues opcions.

```
if(confirm("Do you like dogs?")) {  
  console.log("You are a good person");  
} else {  
  console.log("You have no soul");  
}
```



- El **diàleg prompt** mostra un input després del missatge. Ho podem usar perquè l'usuari introduísca algun valor, retornant un string amb el valor introduït. Si l'usuari prem el botó de Cancel·lar o tanca el diàleg retornarà null. Es pot establir un valor per defecte (segon paràmetre).

```
let name = prompt("What's your name?", "Nobody");  
  
if(name !== "null") {  
  console.log("Your name is: " + name);  
} else {  
  console.log("You didn't answer!");  
}
```



1. Treballant amb el BOM i el DOM.

▶ 1.2 Document Object Model (DOM).

- ▶ El **DOM** és una estructura en arbre que conté una representació de tots els nodes de l'HTML incloent els seus atributs.
- ▶ En aquest arbre, tot es representa com a node i podem afegir, eliminar o modificar-los.
- ▶ L'objecte principal del **DOM** és document. Aquest és un objecte global del llenguatge.
- ▶ Cada node HTML contingut dins del document és un **objecte element**, i aquests elements contenen altres nodes, atributs i estil.
- ▶ Manipular el **DOM** usant només Javascript és una mica més complicat que amb l'ajuda de llibreries com **JQuery** o frameworks com a **Angular**. Veurem alguns mètodes bàsics i propietats dels elements del **DOM** en Javascript.

```
<html>
  ▶ <head>...</head>
  ▼ <body>
    ▼ <div>
      <p>Hello</p>
      ▼ <p>
        "Go to "
        <a href="https://google.es">Google</a>
      </p>
    </div>
    <script src="example1.js"></script>
  </body>
</html>
```

1. Treballant amb el BOM i el DOM.

▶ 1.2 Document Object Model (DOM).

▶ Navegant a través del DOM.

- ▶ **`document.documentElement`** → Retorna l'element `<html>`.
- ▶ **`document.head`** → Retorna l'element `<head>`.
- ▶ **`document.body`** → Retorna l'element `<body>`.
- ▶ **`document.getElementById("id")`** → Retorna l'element que té l'id especificat, o null si no existeix.
- ▶ **`document.getElementsByTagName("class")`** → Retorna un array d'elements que tinguen la classe especificada. En cridar a aquest mètode des d'un node (en lloc de document), buscarà els elements a partir d'aquest node.
- ▶ **`document.getElementsByTagName("HTML tag")`** → Retorna un array amb els elements amb l'etiqueta HTML especificada. Per exemple "p" (paràgrafs).
- ▶ **`element.childNodes`** → Retorna un array amb els descendents (fills) del node. Això inclou els nodes de tipus text i comentaris.
- ▶ **`element.chidren`** → Igual que l'anterior però exclou els comentaris i les etiquetes de text (només nodes HTML). Normalment és el recomanat.
- ▶ **`element.parentNode`** → Retorna el node pare d'un element.
- ▶ **`element.nextSibling`** → Retorna el següent node del mateix nivell (el germà). El mètode **`previousSibling`** fa just l'oposat. És recomanable usar **`nextElementSibling`** o **`previousElementSibling`** si volem obtenir només els elements HTML.

1. Treballant amb el BOM i el DOM.

► 1.2 Document Object Model (DOM).

File: example1.html

```
<!DOCTYPE>

<html>
  <head>
    <title>JS Example</title>
  </head>
  <body>
    <ul>
      <li id="firstListElement">Element 1</li>
      <li>Element 2</li>
      <li>Element 3</li>
    </ul>
    <script src="/example1.js"></script>
  </body>
</html>
```

File: example1.js

```
let firstLi = document.getElementById("firstListElement"); // Devuelve <li>

console.log(firstLi.nodeName); // Imprime "LI"
console.log(firstLi.nodeType); // Imprime 1. (elemento -> 1, atributo -> 2, texto -> 3, comentario -> 8)
console.log(firstLi.firstChild.nodeValue); // Imprime "Element 1". El primer (y único) hijo es un nodo de texto
console.log(firstLi.textContent); // Imprime "Element 1". Otra forma de obtener el contenido (texto)

// Itera a través de todos los elementos de la lista
let liElem = firstLi;
while(liElem !== null) {
  console.log(liElem.innerText); // Imprime el texto de dentro del elemento <li>
  liElem = liElem.nextElementSibling; // Va al siguiente elemento de la lista <li>
}

let ulElem = firstLi.parentElement; // Obtiene el elemento <ul>. Similar a parentNode.
/* Imprime el código HTML de dentro del elemento <ul>:
   <li id="firstListElement">Element 1</li>
   <li>Element 2</li>
   <li>Element 3</li> */
console.log(ulElem.innerHTML);
```

1. Treballant amb el BOM i el DOM.

► 1.2 Document Object Model (DOM).

► Selector Query.

- Una de les principals característiques que JQuery va introduir quan es va llançar va ser la possibilitat d'accedir als elements HTML basant-se en selectors CSS (classe, id, atributs, ...). Des de fa anys, els navegadors han implementat aquesta característica de manera nativa (selector query) sense la necessitat d'usar jQuery
- ***document.querySelector ("selector")*** → Retorna el primer element que coincideix amb el selector.
- ***document.querySelectorAll ("selector")*** → Retorna una col·lecció amb tots els elements que coincideixen amb el selector.
- Exemples de selectores CSS que podem usar per trobar elements.

a → Elementos con la etiqueta HTML <a>

.class → Elementos con la clase "class"

#id → Elementos con el id "id"

.class1.class2 → Elementos que tienen ambas clases, "class1" y "class2"

.class1, .class2 → Elementos que contienen o la clase "class1", o "class2"

.class1 p → Elementos <p> dentro de elementos con la clase "class1"

.class1 > p → Elementos <p> que son hijos inmediatos con la clase "class1"

#id + p → Elemento <p> que va después (siguiente hermano) de un elemento que tiene el id "id"

#id ~ p → Elementos que son párrafos <p> y hermanos de un elemento con el id "id"

.class[attrib] → Elementos con la clase "class" y un atributo llamado "attrib"

.class[attrib="value"] → Elementos con la clase "class" y un atributo "attrib" con el valor "value"

.class[attrib^="value"] → Elementos con la clase "class" y cuyo atributo "attrib" comienza con "value"

.class[attrib*="value"] → Elementos con la clase "class" cuyo atributo "attrib" en su valor contiene "value"

.class[attrib\$="value"] → Elementos con la clase "class" y cuyo atributo "attrib" acaba con "value"

1. Treballant amb el BOM i el DOM.

▶ 1.2 Document Object Model (DOM).

▶ Selector Query.

▶ Exemples utilitzant `querySelector()` i `querySelectorAll()`.

File: example1.html

```
<!DOCTYPE>
<html>
  <head>
    <title>JS Example</title>
  </head>
  <body>
    <div id="div1">
      <p>
        <a class="normalLink" href="hello.html" title="hello world">Hello World</a>
        <a class="normalLink" href="bye.html" title="bye world">Bye World</a>
        <a class="specialLink" href="helloagain.html" title="hello again">Hello Again World</a>
      </p>
    </div>
    <script src="/example1.js"></script>
  </body>
</html>
```

File: example1.js

```
console.log(document.querySelector("#div1 a").title); // Imprime "hello world"
console.log(document.querySelector("#div1 > a").title); // ERROR: No hay un hijo inmediato dentro de <div id="div1"> el cual sea un enlace <a>
console.log(document.querySelector("#div1 > p > a").title); // Imprime "hello world"
console.log(document.querySelector(".normalLink[title^='bye']").title); // Imprime "bye world"
console.log(document.querySelector(".normalLink[title^='bye'] + a").title); // Imprime "hello again"

let elems = document.querySelectorAll(".normalLink");
elems.forEach(function(elem) { // Imprime "hello world" y "bye world"
  console.log(elem.title);
});

let elems2 = document.querySelectorAll("a[title^='hello']"); // Atributo title empieza por "hello..."
elems2.forEach(function(elem) { // Imprime "hello world" y "hello again"
  console.log(elem.title);
});

let elems2 = document.querySelectorAll("a[title='hello world'] ~ a"); // Hermanos de <a title="hello world">
elems2.forEach(function(elem) { // Imprime "bye world" y "hello again"
  console.log(elem.title);
});
```


1. Treballant amb el BOM i el DOM.

▶ 1.2 Document Object Model (DOM).

▶ Manipulant el DOM.

- ▶ **`document.createElement("tag")`** → Crea un element HTML. Encara no estarà en el DOM, fins que ho inserim (usant **`appendChild`**, per exemple) dins d'un altre element del DOM.
- ▶ **`document.createTextNode("text")`** → Crea un node de text que podem introduir dins d'un element. Equival a **`element.innerText = "text"`**.
- ▶ **`element.appendChild(childElement)`** → Afegix un nou element fill al final de l'element pare.
- ▶ **`element.insertBefore(newChildElement, childElem)`** → Inserix un nou element fill abans de l'element fill rebut com a segon paràmetre.
- ▶ **`element.removeChild(childElement)`** → Elimina el node fill que rep per paràmetre.
- ▶ **`element.replaceChild(newChildElem, oldChildElem)`** → Reemplaça un node fill amb un nou node.
- ▶ Exemple de manipulació del DOM (mateix HTML que abans, amb una llista):

1. Treballant amb el BOM i el DOM.

▶ 1.2 Document Object Model (DOM).

▶ Atributs.

- ▶ Dins dels elements HTML hi ha atributs com name, id, href, src, etc. Cada atribut té nom (name) i valor (value), i aquest pot ser llegit o modificat.
- ▶ ***element.attributes*** → Retorna el array amb els atributs d'un element.
- ▶ ***element.className*** → S'usa per a accedir (llegir o canviar) a l'atribut **class**. Altres atributs als quals es pot accedir directament són: **element.id**, **element.title**, **element.style** (propietats CSS),
- ▶ ***element.classList*** → array de classes CSS de l'element. A diferencia de **className**, que és una cadena amb les classes separades per espai, aquest atribut te les retorna en forma de array o llista. Té mètodes molt útils per a consultar i modificar classes com:
 - ▶ ***classList.contains("classe")***: true si té la classe.
 - ▶ ***classList.replace("classe1","classe2")***: Quita la classe "classe1" i la substitueix per la classe "classe2".
 - ▶ ***classList.add("classe1")***: Afegeix la classe "classe1" a l'element.
 - ▶ ***classList.remove("classe1")***: Elimina la classe "classe1".
 - ▶ ***classList.toggle("classe1")***: Si no té "classe1", l'afegeix. En cas contrari, la quita.
- ▶ ***element.hasAttribute("attrName")*** → Retorna true si l'element té un atribut amb el nom especificat.
- ▶ ***element.getAttribute("attrName")*** → Retorna el valor de l'atribut.
- ▶ ***element.setAttribute("attrName", "newValue")*** → Canvia el valor.

1. Treballant amb el BOM i el DOM.

► 1.2 Document Object Model (DOM).

► Atributs.

File: example1.html

```
<!DOCTYPE>
<html>
  <head>
    <title>JS Example</title>
  </head>
  <body>
    <p><a id="toGoogle" href="https://google.es" class="normalLink">Google</a></p>
    <script src="./example1.js"></script>
  </body>
</html>
```

File: example1.js

```
let link = document.getElementById("toGoogle");
link.className = "specialLink"; // Equivale a: link.setAttribute("class", "specialLink");
link.setAttribute("href", "https://twitter.com");
link.textContent = "Twitter";
if(!link.hasAttribute("title")) { // Si no tenía el atributo title, establecemos uno
  link.title = "Ahora voy a Twitter!";
}

/* Imprime: <a id="toGoogle" href="https://twitter.com" class="specialLink"
  title="Ahora voy a Twitter!">Twitter</a> */
console.log(link);
```

1. Treballant amb el BOM i el DOM.

► 1.2 Document Object Model (DOM).


► L'atribut style.

- L'atribut style permet modificar les propietats CSS. La propietat CSS a modificar ha d'escriure's amb el format **camelCase**, mentre que en CSS s'utilitza el format **snake-case**. Per exemple, a l'atribut **background-color** (CSS), s'accedeix a partir de **element.style.backgroundColor**. El valor establert a una propietat serà un string que contindrà un valor CSS vàlid per a l'atribut.

```
<!DOCTYPE>
<html>
  <head>
    <title>JS Example</title>
  </head>
  <body>
    <div id="normalDiv">I'm a normal div</div>
    <script src="./example1.js"></script>
  </body>
</html>
```

File: example1.js

```
let div = document.getElementById("normalDiv");
div.style.boxSizing = "border-box";
div.style.maxWidth = "200px";
div.style.padding = "50px";
div.style.color = "white";
div.style.backgroundColor = "red";
```



I'm a normal
div