Unitat 5

Manipulació del BOM i el DOM

Jorge Rodríguez Sebastià

Contingut

Normalment, quan programem en JavaScript, ens trobem dins del context d'un navegador (això no passa si desenvolupem amb NodeJS per exemple). Dins d'aquest context, tenim alguns objectes predefinits i funcions que podem usar per a interactuar amb aquest navegador i amb el document HTML.

Tota la informació d'aquests objectes, mètodes i propietats es pot consultar en W3schools.com

1. Browser Object Model (BOM)

1.1. 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 omés accedint a les seues propietats directament.

```
1   'use strict';
2   // Grandària total de la finestra (exclou la barra superior del navegador)
3   console.log(window.outerWidth + " - " + window.outerHeight);
4   window.open("https://www.google.com");
5
6   // Propietats de la pantalla
7   console.log(window.screen.width + " - " + window.screen.height); // Ample
    de pantalla i alt (Resolució)
8   console.log(window.screen.availWidth + " - " + window.screen.
    availHeight); // Excloent la barra del S.O.
9
10  // Propietats del navegador
11  console.log(window.navigator.userAgent); // Imprimeix la informació del
    navegador
12  window.navigator.geolocation.getCurrentPosition(function(position) {
13  console.log("Latitude: " + position.coords.latitude + ", Longitude: " +
    position.coords.longitude);
14  });
15
16  // Podem ometre l'objecte window (està implícit)
17  console.log(history.length); // Nombre de pàgines del history. El mateix
    que window.history.length
```

1.2. "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), i 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.

```
1   console.log(new Date().toString()); // Imprimeix immediatament la data
    actual
2   setTimeout(() => console.log(new Date().toString()), 5000); // S'executarà
    en 5 segons (5000 ms)
```

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

```
1   // setTimeout retorna un número amb el id, i a partir d'ací, podrem
    cancel·lar-lo
2   let idTime = setTimeout(() => console.log(new Date().toString()), 5000);
3   clearTimeout(idTime); // Cancel·la el timeout (abans de que s'execute)
```

- 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.

```
1   let num = 1;
2   setInterval(() => console.log(num++), 1000); // Imprimeix un número i
    l'incrementa cada segon
```

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

```
1   let num = 1;
2   let idInterval = setInterval(() => {
3       console.log(num++);
4       if(num > 10) { // Quan imprimim 10, parem el timer perquè no es
        repetisca més
5           clearInterval(idInterval);
6       }
7   }, 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); // Després de 3 segons imprimirà 35 (5*7)
```

1.3. Objecte location

L'objecte location (no confondre amb l'objecte navigator.geolocation) conté informació sobre la url actual del navegador. Podem accedir i modificar aquesta url usant aquest objecte.

```
console.log(location.href); // Imprimeix la URL actual
console.log(location.host); // Imprimeix el nom del servidor (o la IP) com "localhost" 192.168.0.34
console.log(location.port); // Imprime el número del puerto (normalmente 80)
console.log(location.protocol); // Imprimeix el protocol usat (http ó https)
location.reload(); // Recarrega la pàgina actual
location.assign("https://google.com"); // Carrega una nova pàgina. El parámetre es la nova URL
location.replace("https://google.com"); // Carrega una nova pàgina sense guardar l'actual en l'objecte history
```

Per a navegar a través de les pàgines que hem visitat en la pestanya actual, podem usar l'objecte history. Aquest objecte té mètodes bastant útils:

```
console.log(history.length); // Imprimeix el número de pàgines emmagatzemades
history.back(); // Torna a la pàgina anterior
history.forward(); // Va cap a la següent pàgina
history.go(2); // Va dues pàgines avant (-2 aniria dues pàgines cap endarrere)
```

Què ocorre si no volem recarregar la pàgina actual quan anem arrere i avant 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'historial.

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)

ejemplo 2.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="example1.js"></script>
    <button onclick="goBack()">Página anterior</button>
    <button onclick="goNext()">Página siguiente</button>
</body>
</html>
```

ejemplo 2.js

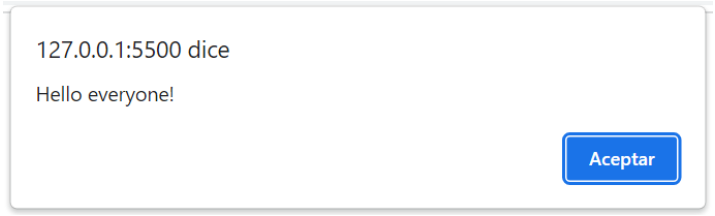
```js
'use strict';
// Esdeveniment per a capturar la navegació pel history
window.onpopstate = function (event) {
    if(event.state) {
        console.log("Estic a la pàgina " + event.state.page);
    } else { // event.state == null si es la primera pàgina
        console.log("Estic a la primera pàgina");
    }
};
let page = 1;
function goBack() {
    history.back();
}
function goNext() {
// history.state == null si es la primera página (la següent es la pàgina 2)
    let pageNum = history.state?history.state.page + 1:2; // Següent pàgina = Pàgina actual + 1.
    history.pushState({page: pageNum}, "");
    console.log("Estic a la página " + pageNum);
}
```

1.4. Diàlegs

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).

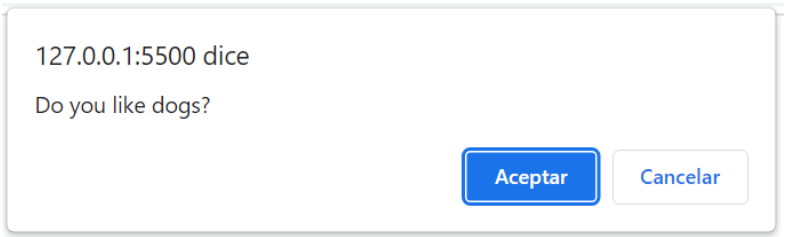
El diàleg alert, mostra un missatge (amb un botó d'Acceptar) dins d'una finestra. Bloqueja l'execució de l'aplicació fins que es tanca.

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



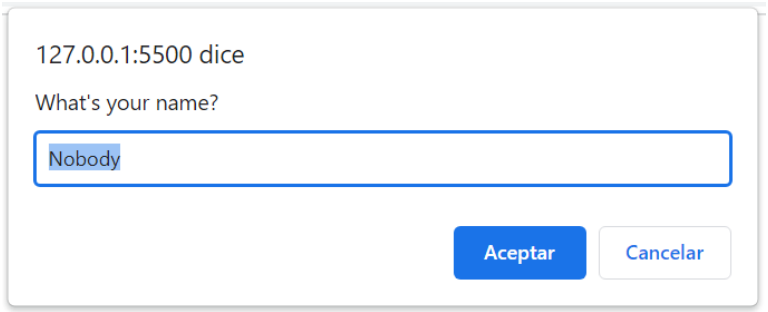
El diàleg confirm és similar, però et retorna un booleà. Té dos botons (cancel·lar → false, Acceptar → true). L'usuari triarà entre una d'aqueixes dues opcions.

```
1   if(confirm("Do you like dogs?")) {
2       console.log("You are a good person");
3       } else {
4       console.log("You have no soul");
5   }
```



El diàleg prompt mostra un input després del missatge. Ho podem usar perquè l'usuari introduïsca 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).

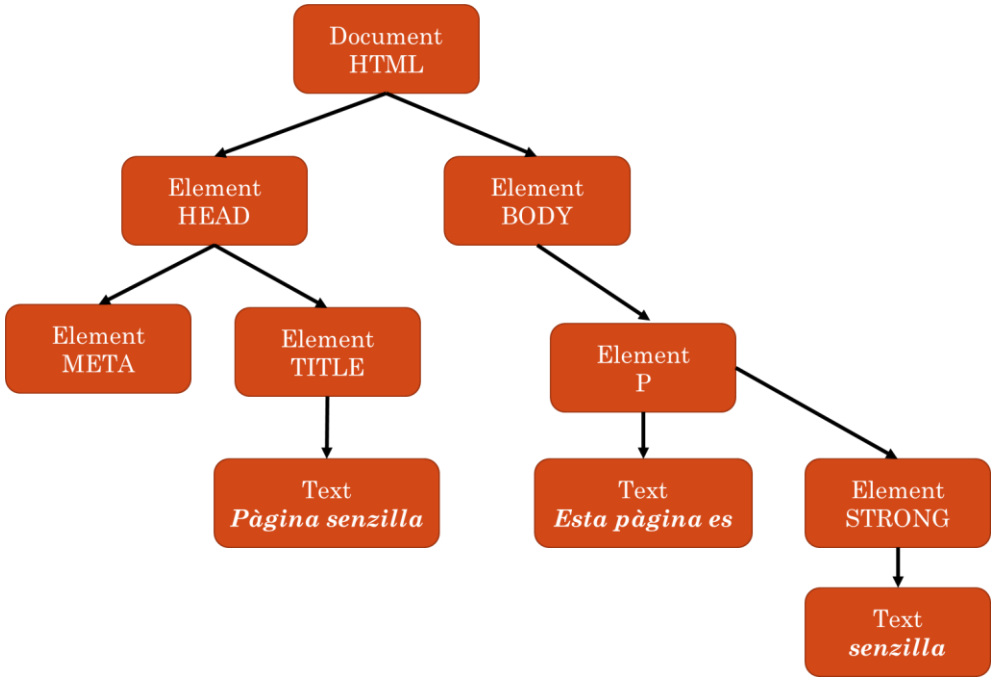
```
let nom = prompt("What's your name?", "Nobody");
if(nom !== "null") {
console.log("Your name is: " + nom);
} else {
console.log("You didn't answer!");
}
```



2. Document Object Model (DOM)

El Document Object Model 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.

```
<!DOCTYPE html>
    <html lang="es">
        <head>
            <meta charset="utf-8" />
            <title>Pàgina senzilla</title>
        </head>
        <body>
            <p>Aquesta pàgina es <strong>molt senzilla</strong></p>
        </body>
    </html>
```



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 Vue.js. Veurem alguns mètodes bàsics i propietats dels elements del DOM en JavaScript. No ho estudiarem en profunditat per a abastar més conceptes durant el curs, i perquè realment usant jQuery, Vue.js, Angular, etc. és més fàcil i pràctic.

2.1. 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'aneu especificat, o null si no existeix.
- document.getElementsByClassName("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 a dalt 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 utilitzar nextElementSibling o previousElementSibling si volem obtindre només els elements HTML.

ejemplo 2.html

```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>
```

ejemplo 2.js

```js
let firstLi = document.getElementById("firstListElement"); // Retorna <li>
console.log(firstLi.nodeName); // Imprimeix "LI"
console.log(firstLi.nodeType); // Imprimeix 1. (element -> 1, atribut -> 2, text -> 3, comentari -> 8)
console.log(firstLi.firstChild.nodeValue); // Imprimeix "Element 1". El primer (y únic) fill és un node de text
console.log(firstLi.textContent); // Imprimeix "Element 1". Una altra manera d'obtindre el contingut (text)

// IItera a través de tots els elements de la llista
let liElem = firstLi;
while(liElem !== null) {
    console.log(liElem.innerText); // Imprimeix el text de dins de l'element <li>
    liElem = liElem.nextElementSibling; // Va al següent element de la llista <li>
}

let ulElem = firstLi.parentElement; // Obté l'element <ul>. Similar a parentNode.
/* Imprimeix el codi HTML de dins de l'element <ul>:
<li id="firstListElement">Element 1</li>
<li>Element 2</li>
<li>Element 3</li> */
console.log(ulElem.innerHTML);
```

2.2. Selector Query

Una de les principals característiques que JQuery va introduir quan es va llançar (en 2006) va ser la possibilitat d'accedir als elements HTML basant-se en selectors CSS (classe, aneu, 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 selectors CSS que podem utilitzar per a trobar elements:

a → Elements amb l'etiqueta HTML <a>
.class → Elements amb la classe “class”
#id → Elements amb el id “id”
.class1.class2 → Elements que tenen totes dues classes, "class1" i "class2"
.class1,.class2 → Elements que contenen o la classe “class1”, o “class2”
.class1 p → Elements <p> dins d'elements amb la classe “class1”
.class1 > p → Elements <p> que són fills immediats amb la classe “class1”
#id + p → Element <p> que va després (següent germà) d'un element que té el id "id"
#id ~ p → Elements que són paràgrafs <p> i germans d'un element amb el id "id"
.class[attrib] → Elements amb la classe "class" i un atribut anomenat "attrib"
.class[attrib="value"] → Elements amb la classe "class" i un atribut "attrib" amb el valor "value"
.class[attrib^="value"] → Elements amb la classe "class" i l'atribut de la qual "attrib" comença amb "value"
.class[attrib*="value"] → Elements amb la classe "class" l'atribut de la qual "attrib" en el seu valor conté "value"
.class[attrib$="value"] → Elements amb la classe "class" i l'atribut de la qual "attrib" acaba amb "value"

Exemple usant querySelector() i querySelectorAll():

ejemplo 2.html

```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>
```

ejemplo 2.js

```javascript
console.log(document.querySelector("#div1 a").title); // Imprimeix "hello world"
console.log(document.querySelector("#div1 > a").title); // ERROR: No hi ha un fill immediat dins de <div id="div1"> el qual siga un enllaç <a>
console.log(document.querySelector("#div1 > p > a").title); // Imprimeix "hello world"
console.log(document.querySelector(".normalLink[title^='bye']").title); // Imprimeix "bye world"
console.log(document.querySelector(".normalLink[title^='bye'] + a").title); // Imprimeix "hello again"
let elems = document.querySelectorAll(".normalLink");
elems.forEach(function(elem) { // Imprimeix "hello world" y "bye world"
    console.log(elem.title);
});
let elems2 = document.querySelectorAll("a[title^='hello']"); // Atribut title comença per “hello...”
elems2.forEach(function(elem) { // Imprimeix "hello world" y "hello again"
    console.log(elem.title);
});
let elems3 = document.querySelectorAll("a[title='hello world'] ~ a"); // Germans de <a title="hello world">
elems2.forEach(function(elem) { // Imprimeix "bye world" y "hello again"
    console.log(elem.title);
});
```

2.3. 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 DG.

document.createTextNode("text") → Crea un node de text que podem introduir dins d'un element. Equival a element.innerText = “text”.

element.appendChild(childElement) → Afig un nou element fill al final de l'element-pare.

element.insertBefore(newChildElement, childElem) → Inserida 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):

```
let ul = document.getElementsByTagName("ul")[0]; // Obté la primera llista (ul)
let li3 = ul.children[2]; // Tercer element de la llista (li)
let newLi3 = document.createElement("li"); // Crea un nou element de llista
newLi3.innerText = "Now I'm the third element"; // Y li assigna un text
ul.insertBefore(newLi3, li3); // Ara li3 es el quart element de la llista (newLi3 s'insereix abans)
li3. innerText = "I'm the fourth element..."; // Canviem el text per a reflectir que és el quart element
```

El HTML resultant serà:

```
<ul>
    <li id="firstListElement">Element 1</li>
    <li>Element 2</li>
    <li>Now I'm the third element</li>
    <li>I'm the fourth element...</li>
</ul>
```

2.4. 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 l'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 diferència 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("clase1", "clase2"): lleva la classe "clase1" i la substitueix per la classe "clase2".
  - classList.add("clase1"): Afig la classe "clase1" a l'element.
  - classList.remove("clase1"): Li lleva la classe "clase1".
  - classList.toggle("clase1"): Si no té "clase1", l'afig. En cas contrari, la lleva.
- element.hasAttribute("attrName") → Retorna cert 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

ejemplo 2.html

```
1  <!DOCTYPE>
2  <html>
3      <head>
4          <title>JS Example</title>
5      </head>
6      <body>
7          <p><a id="toGoogle" href="https://google.es" class="normalLink">Google</a></p>
8          <script src="./example1.js"></script>
9      </body>
10 </html>
```

ejemplo 2.js

```
let link = document.getElementById("toGoogle");
link.className = "specialLink"; // Equival a: link.setAttribute("class",
"specialLink");
link.setAttribute("href", "https://twitter.com");
link.textContent = "Twitter";
if(!link.hasAttribute("title")) { // Si no tenia l'atribut title,
n'establim un
link.title = "Ara vaig a Twitter!";
}
/* Imprimeix: <a id="toGoogle" href="https://twitter.com"
class="specialLink"
title="Ara vaig a Twitter!">Twitter</a> */
console.log(link);
```

2.5. L'atribut style

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

ejemplo 2.html

```
<!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>
```

JS ejemplo 2.js ●

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