

Llenguatges de Marques i Sistemes de Gestió d'Informació

UD 5.3 DOM



IES Jaume II El Just
Tavernes de la Valldigna
Departament d'Informàtica
Curs 2024-25

Índex

1	DOM (Document Object Model)	3
1.1	L'estructura del DOM	3
1.1.1	Correcció automàtica del DOM	4
1.1.2	Exploració del DOM	4
1.2	Navegació pel DOM	5
1.2.1	Accés als elements principals del document	5
1.2.2	Propietats per accedir als nodes fills	5
1.2.3	Propietats per accedir als nodes pare	5
1.2.4	Propietats per accedir als nodes germans	6
1.2.5	Propietats específiques per a elements	6
1.3	Cerca d'elements	6
1.3.1	<code>document.getElementById(id)</code>	6
1.3.2	<code>elem.querySelectorAll(css)</code>	7
1.3.3	<code>elem.querySelector(css)</code>	7
1.3.4	<code>elem.matches(css)</code>	8
1.3.5	<code>elem.getElementsByTagName(tag)</code>	8
1.3.6	<code>elem.getElementsByClassName(className)</code>	8
1.3.7	<code>document.getElementsByName(name)</code>	9
1.3.8	Resum	9
1.3.9	Observacions:	10
1.4	Propietats del node	10
1.4.1	<code>innerHTML</code>	10
1.4.2	<code>outerHTML</code>	11
1.4.3	<code>data</code> (per a nodes de text i comentaris)	11
1.4.4	<code>textContent</code>	12
1.4.5	<code>hidden</code>	13
1.4.6	<code>value</code>	13
1.5	Modificació del DOM	14
1.5.1	<code>document.createElement(tagName)</code>	14
1.5.2	<code>append(...nodesOrDOMStrings)</code>	14
1.5.3	<code>prepend(...nodesOrDOMStrings)</code>	15
1.5.4	<code>after(...nodesOrDOMStrings)</code>	15
1.5.5	<code>before(...nodesOrDOMStrings)</code>	15
1.5.6	<code>replaceWith(...nodesOrDOMStrings)</code>	15
1.5.7	<code>cloneNode(deep)</code>	16
1.5.8	<code>remove()</code>	16

1.5.9	Mètodes de la “vella escola”	16
1.6	Modificant estils	17
1.6.1	style	17
1.6.2	className	17
1.6.3	classList	18
1.6.4	Mètodes de <code>classList</code>	18
1.6.5	Combinació de <code>classList</code> i <code>style</code>	19
1.7	Referències	19

1 DOM (Document Object Model)

El navegador representa internament un document HTML mitjançant una estructura jeràrquica anomenada **DOM (Document Object Model)**. Aquesta estructura converteix cada etiqueta HTML en **nodes d'element**, i el contingut dins de les etiquetes en **nodes de text**, fent que la pàgina web siga accessible i manipulable mitjançant **JavaScript**.

1.1 L'estructura del DOM

Quan un navegador carrega un document HTML, el converteix en un arbre de nodes. Per exemple, si tenim aquest HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>My Document</title>
  </head>
  <body>
    <h1>Header</h1>
    <p>Paragraph</p>
  </body>
</html>
```

L'estructura del DOM resultant serà:

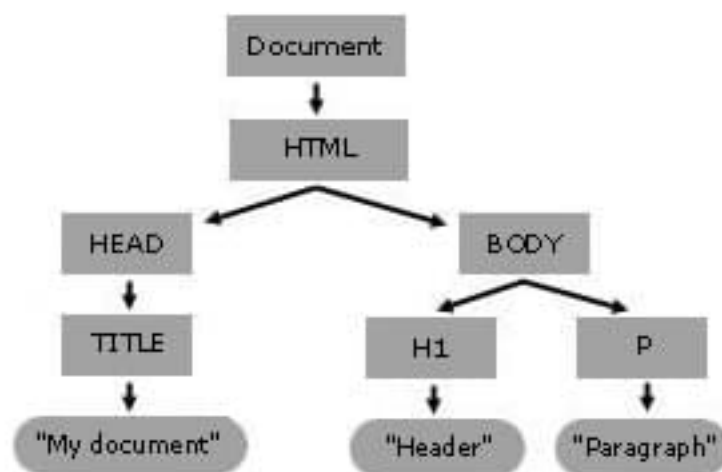


Figura 1: Arbre del document

Cada element del document es representa com un **node** en aquest arbre.

El DOM conté diferents **tipus de nodes**, cadascun amb les seues pròpies propietats i funcions:

1. **Node d'element (element node)**

- Representa una etiqueta HTML (<div>, <p>, , etc.).
- Conté altres nodes com a fills (altres elements o text).

2. **Node de text (text node)**

- Conté únicament text.
- No pot tindre fills, només representa el text dins d'un element.

3. **Node de comentari (comment node)**

- Representa un comentari HTML.
- No afecta la representació visual de la pàgina.

4. **Node d'atribut (attribute node)**

- Representa atributs HTML (class, id, src, etc.).
- No es considera part de l'arbre principal, sinó com a propietats dels nodes d'element.

1.1.1 Correcció automàtica del DOM

Els navegadors poden corregir errors en l'HTML en crear el DOM. Per exemple:

- Si falta la etiqueta <tbody> en una taula, el navegador l'afegirà automàticament.
- Si una etiqueta no es tanca correctament (<p>Text sense </p>), el navegador intentarà corregir-ho.

Aquest comportament pot causar diferències entre el codi HTML original i l'estructura final del DOM.

1.1.2 Exploració del DOM

Els navegadors ofereixen eines de desenvolupador per explorar i modificar el DOM en temps real. Per accedir-hi:

1. **Obrir la consola de desenvolupador** (F12 o Ctrl + Shift + I en Chrome/Firefox).
2. **Usar document o inspector a la consola** per veure l'estructura de la pàgina.
3. **Modificar elements en temps real** canviant els estils i continguts de la pàgina.

1.2 Navegació pel DOM

La navegació pel DOM (Document Object Model) és fonamental per manipular i accedir als elements d'una pàgina web mitjançant JavaScript. A continuació, es detallen les principals propietats i mètodes utilitzats per desplaçar-se per l'estructura del DOM

1.2.1 Accés als elements principals del document

- **document.documentElement**: Representa l'element `<html>` del document.
- **document.body**: Accedeix a l'element `<body>`.
- **document.head**: Accedeix a l'element `<head>`.

És important destacar que, si el script s'executa abans que el navegador haja carregat completament l'element `<body>`, `document.body` pot retornar `null`. Això succeirà si l'script es troba en el `<head>`.

1.2.2 Propietats per accedir als nodes fills

- **childNodes**: Retorna una col·lecció de tots els nodes fills, incloent-hi els nodes de text.
- **firstChild**: Accedeix al primer node fill.
- **lastChild**: Accedeix a l'últim node fill.

Per exemple, per iterar sobre tots els fills de `document.body`:

```
for (let i = 0; i < document.body.childNodes.length; i++) {  
  console.log(document.body.childNodes[i]);  
}
```

1.2.3 Propietats per accedir als nodes pare

- **parentNode**: Retorna el node pare immediat.

Per exemple, per obtenir el pare d'un element:

```
let element = document.querySelector('li');  
console.log(element.parentNode); // Mostra l'element <ul>
```

1.2.4 Propietats per accedir als nodes germans

- **previousSibling**: Accedeix al node germà anterior.
- **nextSibling**: Accedeix al node germà següent.

Per exemple, per obtenir el germà següent d'un element:

```
let element = document.querySelector('li');  
console.log(element.nextSibling); // Pot retornar un node de text o un  
→ element
```

1.2.5 Propietats específiques per a elements

Les propietats esmentades anteriorment consideren tots els nodes, incloent-hi els nodes de text. Per treballar exclusivament amb elements HTML, s'utilitzen les següents propietats:

- **children**: Retorna una col·lecció HTML dels elements fills.
- **firstElementChild**: Accedeix al primer element fill.
- **lastElementChild**: Accedeix a l'últim element fill.
- **parentElement**: Retorna l'element pare.
- **previousElementSibling**: Accedeix a l'element germà anterior.
- **nextElementSibling**: Accedeix a l'element germà següent.

Aquestes propietats són útils per evitar nodes de text innecessaris en la navegació del DOM.

1.3 Cerca d'elements

A continuació, es presenten els principals mètodes per buscar elements dins del DOM:

1.3.1 document.getElementById(id)

Aquest mètode retorna l'element amb l'atribut `id` especificat. És important que l'`id` siga únic dins del document; en cas contrari, el comportament pot ser imprevisible.

Exemple:

```
<div id="element">Contingut</div>  
  
<script>
```

```
let elem = document.getElementById('element');
elem.style.background = 'yellow';
</script>
```

1.3.2 elem.querySelectorAll(css)

Retorna tots els elements dins de `elem` que coincideixen amb el selector CSS proporcionat. Aquest mètode és molt versàtil, ja que permet utilitzar qualsevol selector CSS, incloent pseudoclasses com `:hover` i `:active`.

Exemple:

```
<ul>
  <li>Element 1</li>
  <li>Element 2</li>
</ul>
<ul>
  <li>Element 3</li>
  <li>Element 4</li>
</ul>

<script>
let elements = document.querySelectorAll('ul > li:last-child');
elements.forEach(elem => {
  console.log(elem.innerHTML); // "Element 2", "Element 4"
});
</script>
```

1.3.3 elem.querySelector(css)

Similar a `querySelectorAll`, però retorna només el primer element que coincideix amb el selector CSS especificat. És més eficient quan només es necessita un únic element.

Per exemple `document.querySelector('#menu')` és equivalent a `document.getElementById('menu')`

Exemple:

```
<div class="contenedor">
  <p>Paràgraf 1</p>
  <p>Paràgraf 2</p>
</div>

<script>
```



```
let parrafo = document.querySelector('.contenedor p');
console.log(parrafo.innerHTML); // "Paràgraf 1"
</script>
```

1.3.4 elem.matches(css)

Aquest mètode verifica si l'element `elem` coincideix amb el selector CSS proporcionat, retornant `true` o `false`. És útil per a filtres durant la iteració d'elements.

Exemple:

```
<a href="archivo.zip">Descarregar arxiu</a>
<a href="pagina.html">Anar a la pàgina</a>

<script>
  document.querySelectorAll('a').forEach(link => {
    if (link.matches('a[href$=".zip"]')) {
      console.log("Enllaç a arxiu ZIP: " + link.href);
    }
  });
</script>
```

1.3.5 elem.getElementsByTagName(tag)

Retorna una col·lecció d'elements amb el nom de l'etiqueta especificada. Aquest mètode és viu, és a dir, la col·lecció s'actualitza automàticament si es modifiquen els elements del DOM.

Exemple:

```
<div>
  <p>Paràgraf 1</p>
  <p>Paràgraf 2</p>
</div>

<script>
  let parrafos = document.getElementsByTagName('p');
  console.log(parrafos.length); // 2
</script>
```

1.3.6 elem.getElementsByClassName(className)

Retorna una col·lecció d'elements que tenen la classe CSS especificada. També és una col·lecció viva.

Exemple:

```
<div class="caixa">Caixa 1</div>
<div class="caixa">Caixa 2</div>

<script>
  let caixes = document.getElementsByClassName('caixa');
  console.log(caixes.length); // 2
</script>
```

1.3.7 document.getElementsByName(name)

Retorna una col·lecció d'elements amb l'atribut name especificat. S'utilitza principalment per a formularis.

Exemple:

```
<form>
  <input type="text" name="usuari">
  <input type="text" name="email">
</form>

<script>
  let inputs = document.getElementsByName('usuari');
  console.log(inputs.length); // 1
</script>
```

1.3.8 Resum

Mètode	Descripció	Retorna un únic element?	Retorna una col·lecció?
querySelector(css)	Selecciona el primer element que coincideix amb el selector CSS.	Sí	No
querySelectorAll(css)	Selecciona tots els elements que coincideixen amb el selector CSS.	No	Sí
getElementById(id)	Selecciona un element pel seu id.	Sí	No
getElementsByTagName(tag)	Selecciona tots els elements per nom de l'etiqueta.	No	Sí

Mètode	Descripció	Retorna un únic element?	Retorna una col·lecció?
<code>getElementsByClassName</code>	Selecció dels elements per classe CSS.	No	Sí
<code>getElementsByName</code>	Selecció dels elements amb l'atribut <code>name</code> .	No	Sí

1.3.9 Observacions:

- Els mètodes **`getElementById`** i **`querySelector`** retornen **un únic element**.
- Els mètodes **`getElementsByTagName`**, **`getElementsByClassName`**, **`getElementsByName`** i **`querySelectorAll`** retornen **una col·lecció** d'elements.
- **`querySelectorAll`** retorna un **`NodeList`**, mentre que els altres mètodes que retornen col·leccions utilitzen una **`HTMLCollection`** (que es pot modificar en temps real si el DOM canvia).

1.4 Propietats del node

En el Document Object Model (DOM), els elements HTML posseeixen diverses propietats que permeten accedir i modificar el seu contingut i atributs. A continuació, es detallen les propietats **`innerHTML`**, **`outerHTML`**, **`data`**, **`textContent`**, **`hidden`** i **`value`**:

1.4.1 `innerHTML`

La propietat `innerHTML` permet obtenir o establir el contingut HTML d'un element com una cadena de text. És útil per inserir o reemplaçar elements fills dins d'un element existent.

Exemple:

```
<div id="contenedor">
  <p>Paràgraf inicial</p>
</div>

<script>
  // Obtenir el contingut HTML
  console.log(document.getElementById('contenedor').innerHTML);
  // Eixida: <p>Paràgraf inicial</p>

  // Establir un nou contingut HTML
```

```
document.getElementById('contenedor').innerHTML = '<p>Nou paràgraf</p>';  
</script>
```

Consideracions:

- Quan s'estableix `innerHTML`, el contingut anterior de l'element es reemplaça.
- Si s'insereixen etiquetes `<script>` mitjançant `innerHTML`, aquestes no s'executen.

1.4.2 outerHTML

La propietat `outerHTML` proporciona el codi HTML complet de l'element, incloent-hi l'element mateix. A diferència de `innerHTML`, modificar `outerHTML` reemplaça l'element en si mateix en el DOM.

Exemple:

```
<div id="contenedor">  
  <p>Paràgraf inicial</p>  
</div>  
  
<script>  
  let element = document.getElementById('contenedor');  
  
  // Obtenir l'HTML complet de l'element  
  console.log(element.outerHTML);  
  // Eixida: <div id="contenedor"><p>Paràgraf inicial</p></div>  
  
  // Reemplaçar l'element per un altre  
  element.outerHTML = '<section id="nou-contenedor"><p>Nou  
→ contingut</p></section>';  
  
  // L'element 'element' encara fa referència al vell <div>  
  console.log(element.innerHTML);  
  // Eixida: <p>Paràgraf inicial</p>  
</script>
```

Consideracions:

- Després de reemplaçar un element amb `outerHTML`, la referència a l'element antic encara existeix, però ja no forma part del DOM.

1.4.3 data (per a nodes de text i comentaris)

La propietat `data` s'utilitza per accedir o modificar el contingut de nodes de text o comentaris.

Exemple:

```
<div id="contenedor">
  <!-- Comentari inicial -->
  Text inicial
</div>

<script>
  let element = document.getElementById('contenedor');

  // Accedir al node de text
  let textNode = element.childNodes[1];
  console.log(textNode.data);
  // Eixida: Text inicial

  // Accedir al node de comentari
  let commentNode = element.childNodes[0];
  console.log(commentNode.data);
  // Eixida: Comentari inicial
</script>
```

1.4.4 textContent

La propietat `textContent` retorna o estableix el text dins d'un element, excloent-hi qualsevol etiquetes HTML. És útil per obtenir o establir el text sense format d'un element.

Exemple:

```
<div id="contenedor">
  <p><b>Text en negreta</b> i text normal.</p>
</div>

<script>
  let element = document.getElementById('contenedor');

  // Obtenir el text sense etiquetes
  console.log(element.textContent);
  // Eixida: Text en negreta i text normal.

  // Establir un nou text
  element.textContent = 'Nou text sense HTML.';
  console.log(element.innerHTML);
```

```
// Eixida: Nou text sense HTML.  
</script>
```

Consideracions:

- A diferència de `innerHTML`, `textContent` tracta tot el contingut com a text pla, sense interpretar etiquetes HTML.
- És una manera segura d'inserir text, ja que evita l'execució de codi HTML o JavaScript potencialment maliciós.

1.4.5 hidden

La propietat `hidden` és un atribut booleà que indica si un element és visible o no. Quan es estableix a `true`, l'element es comporta com si tingués `display: none` en CSS.

Exemple:

```
<div id="contenedor">  
  Contingut visible  
</div>  
  
<script>  
  let element = document.getElementById('contenedor');  
  
  // Amagar l'element  
  element.hidden = true;  
  
  // Mostrar l'element després de 2 segons  
  setTimeout(() => {  
    element.hidden = false;  
  }, 2000);  
</script>
```

Consideracions:

- És una manera senzilla de controlar la visibilitat d'un element sense manipular directament els estils CSS.

1.4.6 value

La propietat `value` s'utilitza principalment en elements de formulari com `<input>`, `<textarea>` i `<select>` per obtenir o establir el seu valor actual.

Exemple:

```
<input type="text" id="camp" value="Valor inicial">

<script>
  let inputElement = document.getElementById('camp');

  // Obtenir el valor actual
  console.log(inputElement.value);
  // Sortida: Valor inicial

  // Establir un nou valor
  inputElement.value = 'Nou valor';
</script>
```

1.5 Modificació del DOM

La manipulació del Document Object Model (DOM) és essencial per crear pàgines web dinàmiques i interactives. A continuació, es presenten diverses funcions modernes per crear, inserir, reemplaçar i eliminar elements del DOM.

1.5.1 document.createElement(tagName)

Crea un nou element HTML amb el nom d'etiqueta especificat.

Exemple:

```
// Crear un nou element <div>
let div = document.createElement('div');
```

1.5.2 append(...nodesOrDOMStrings)

Insereix un o més nodes o cadenes de text al final de l'element. Si s'insereixen cadenes de text, es converteixen en nodes de text.

Exemple:

```
// Crea un paràgraf, amb el text 'Hola, món!', dins del div amb el id demo
let myDiv = document.querySelector('#demo');
let p = document.createElement('p');
p.textContent = 'Hola, món!';
myDiv.append(p);
```

1.5.3 prepend(...nodesOrDOMStrings)

Insereix un o més nodes o cadenes de text al principi de l'element.

Exemple:

```
//Afegeix un li com a primer element del ul amb id myUl
let ul = document.querySelector('#myUl');
let li = document.createElement('li');
li.textContent = 'Primer element';
myUl.prepend(li);
```

1.5.4 after(...nodesOrDOMStrings)

Insereix un o més nodes o cadenes de text immediatament després de l'element en el mateix nivell jeràrquic.

Exemple:

```
let p = document.createElement('p');
p.textContent = 'Paràgraf existent';
document.body.append(p);
let p2 = document.createElement('p');
p2.textContent = 'Paràgraf després del paràgraf existent';
p.after(p2);
```

1.5.5 before(...nodesOrDOMStrings)

Insereix un o més nodes o cadenes de text immediatament abans de l'element en el mateix nivell jeràrquic.

Exemple:

```
let p = document.createElement('p');
p.textContent = 'Paràgraf existent';
document.body.append(p);
p.before('Text abans del paràgraf', document.createElement('hr'));
```

1.5.6 replaceWith(...nodesOrDOMStrings)

Reemplaça l'element pel conjunt de nodes o cadenes de text especificats.

Exemple:


```
let oldElement = document.createElement('div');
oldElement.textContent = 'Element antic';
document.body.append(oldElement);
oldElement.replaceWith('Element nou', document.createElement('span'));
```

1.5.7 cloneNode(deep)

Crea una còpia de l'element en què es crida. Si el paràmetre `deep` és `true`, es clonen també tots els seus descendents; si és `false`, només es clona l'element en si.

Exemple:

```
let originalNode = document.createElement('div');
originalNode.textContent = 'Original';
let shallowClone = originalNode.cloneNode(false); // Sense fills
let deepClone = originalNode.cloneNode(true); // Amb fills
```

1.5.8 remove()

Elimina l'element del DOM.

Exemple:

```
let element = document.createElement('p');
element.textContent = 'Aquest paràgraf serà eliminat.';
document.body.append(element);
element.remove();
```

1.5.9 Mètodes de la “vella escola”

Abans de la introducció dels mètodes esmentats anteriorment, s'utilitzaven altres mètodes per manipular el DOM:

- **parentNode.insertBefore(newNode, referenceNode)**: Insereix `newNode` abans de `referenceNode` com a fill de `parentNode`.
- **parentNode.replaceChild(newNode, oldNode)**: Reemplaça `oldNode` per `newNode` com a fill de `parentNode`.
- **parentNode.removeChild(childNode)**: Elimina `childNode` de `parentNode`.
- **document.write(html)**: Agrega HTML a la pàgina abans de terminar la càrrega. Si s'executa quan s'ha carregat, esborra tot el contingut de la pàgina.

Aquests mètodes encara són vàlids i àmpliament compatibles, però les funcions modernes proporcionen una sintaxi més senzilla i llegible per a la manipulació del DOM.

1.6 Modificant estils

A continuació vorem com manipular els estils i les classes d'elements HTML amb JavaScript. Explicuem el funcionament de tres elements importants: **style**, **className** i **classList**.

1.6.1 style

El **style** és una propietat que permet modificar els estils en línia d'un element HTML mitjançant JavaScript. Aquesta propietat es tracta com un objecte, on cada propietat correspon a una regla CSS. Per exemple:

```
element.style.backgroundColor = "red"; // Canvia el color de fons a vermell
```

Característiques de style:

- Modifica només els estils en línia, és a dir, els que s'apliquen directament a l'element.
- No permet afegir estils CSS complets (com els definits en fulls d'estils externs).
- S'utilitza per canvis dinàmics de l'estil d'un element.
- Pot ser útil quan es vol canviar de manera ràpida l'estil d'un element (per exemple, al passar el ratolí per sobre o durant animacions).
- Els estils aplicats amb `style` tenen més prioritat que els estils definits en fulls d'estils CSS, però menys prioritat que els estils inline.

Exemple:

```
const element = document.getElementById("miElemento");
element.style.color = "blue"; // Canvia el color del text a blau
element.style.fontSize = "20px"; // Canvia la mida del text
```

1.6.2 className

La propietat **className** permet obtenir o establir la cadena completa de classes de l'element. És una cadena de text que conté tots els noms de les classes que s'hi han aplicat. Quan utilitzem **className**, reemplaçarem totes les classes anteriors amb la nova cadena de classes.

Característiques de className:

- Permet establir totes les classes de l'element.

- Si assignem una nova cadena de text a **className**, eliminarem totes les classes prèviament assignades i les substituïrem per la nova.
- És útil per a situacions en què volem establir un conjunt complet de classes per a un element, però pot ser menys flexible que altres mètodes si només volem modificar una classe específica.

Exemple:

```
const element = document.getElementById("miElemento");
element.className = "clase1 clase2"; // Substitueix totes les classes per
↪ "clase1" i "clase2"
```

Si volem afegir classes de manera dinàmica sense substituir les existents, hauríem d'utilitzar **classList** (veure més endavant).

1.6.3 classList

classList és una propietat que proporciona una forma més flexible i moderna de manipular les classes d'un element. A diferència de **className**, **classList** ens permet afegir, eliminar o alternar (*toggle*) classes individualment sense haver de preocupar-nos de substituir tota la cadena de classes de l'element.

1.6.4 Mètodes de classList

- **add(className)**: Afegeix una classe a l'element.

```
element.classList.add("novaClasse");
```
- **remove(className)**: Elimina una classe de l'element.

```
element.classList.remove("claseAntiga");
```
- **toggle(className)**: Alterna l'estat d'una classe. Si l'element té la classe, se'n treu; si no la té, se n'afegeix.

```
element.classList.toggle("actiu");
```
- **contains(className)**: Comprova si l'element té una classe específica.

```
if (element.classList.contains("actiu")) {
  console.log("L'element té la classe 'actiu'");
}
```

Característiques de classList:

- Ens permet manipular classes de manera més flexible, evitant la necessitat de substituir tota la cadena de classes com passa amb **className**.
- És ideal per a afegir o eliminar classes específiques sense afectar altres classes de l'element.
- **classList** és un objecte que té mètodes per manipular les classes de manera molt precisa.
- A més, és iterable, el que significa que podem iterar sobre totes les classes d'un element.

Exemple:

```
const element = document.getElementById("miElemento");
element.classList.add("highlight"); // Afegeix la classe "highlight"
element.classList.remove("oldClass"); // Elimina la classe "oldClass"
element.classList.toggle("active"); // Alterna la classe "active"
```

1.6.5 Combinació de classList i style

És possible utilitzar `classList` per gestionar classes dinàmicament mentre que els estils es gestionen amb `style`. Això permet una manipulació més senzilla i més eficaç de l'aspecte i les interaccions de l'element.

```
element.classList.add("visible");
element.style.color = "green"; // Modifica el color de l'element amb
↪ l'estil en línia
```

Amb aquests tres mètodes, podem manipular els estils i les classes de manera dinàmica, creant interaccions més riques i controlades per part de JavaScript. La millor pràctica depèn de la situació: si necessitem canvis ràpids i temporals, **style** és una bona opció, però si volem modificar classes específiques sense alterar la resta, **classList** és el mètode preferit.

1.7 Referències

- [Tutorial de DOM \(Javascript.info\)](#)
- [Tutorial DOM W3Schools](#)