

UD4.1

Javascript

LLENGUATGES DE MARQUES I SISTEMES
DE GESTIÓ DE LA INFORMACIÓ – 1º DAM



José Javier Segura Martínez



Index

1. Introducció

1.1 Que es un framework

1.2 Que es una llibreria

1.3 Javascript en HTML

2. Codi Javascript

3. Sintaxi Javascript

4. Funcions

5. DOM. Arbre de nodes

6. innerHTML i outerHTML

7. Invocar funcions senzilles amb href

8. Nous mètodes selectors per a Javascript

9. Esdeveniments. Events

10. Noves funcionalitats

11. Bibliografia

1. Introducció (I)

Javascript és un llenguatge **no compilat** que funciona gràcies a un **intèrpret** de codi (el **navegador Web** en incorpora un). Així, no podrem executar un script en aquest llenguatge si no és mitjançant un navegador o una ferramenta que dispose d'un intèrpret.

Amb JS podrem donar funcionalitat a la nostra pàgina.

A dia de hui es pot utilitzar tant per a entorn servidor com per a entorn client (tant al backend com al frontend).

1. Introducció (II)



“Necessite un llenguatge de scripts per a poder executar-los en el meu navegador i així automatitzar i fer les pàgines més dinàmiques”

“Tens deu dies per a desenvolupar un prototip funcional”

1. Introducció (III)

Creat el **1995** per **Brendan Eich**. En l'empresa Netscape Communications, que és la que va desenvolupar els primers navegadors web comercials.

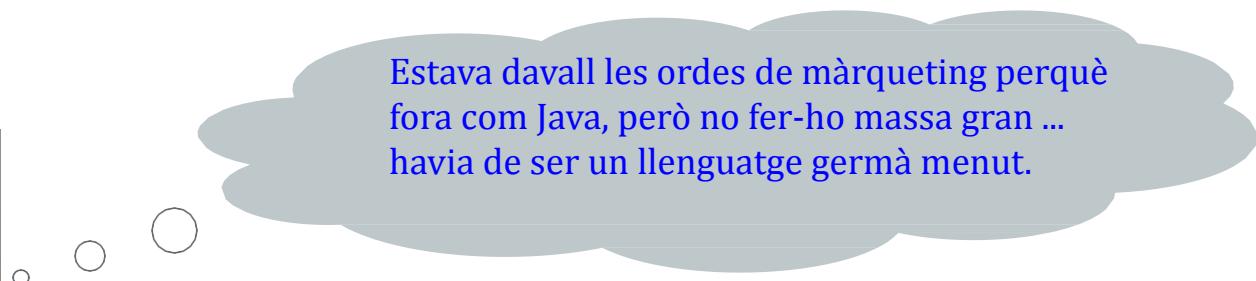
Va aparèixer per primera vegada en el producte de Netscape anomenat **Netscape Navigator 2.0**

La **W3C** va dissenyar l'estàndard **Document Object Model** (DOM, o Model d'Objectes del Document).

1. Introducció (V)

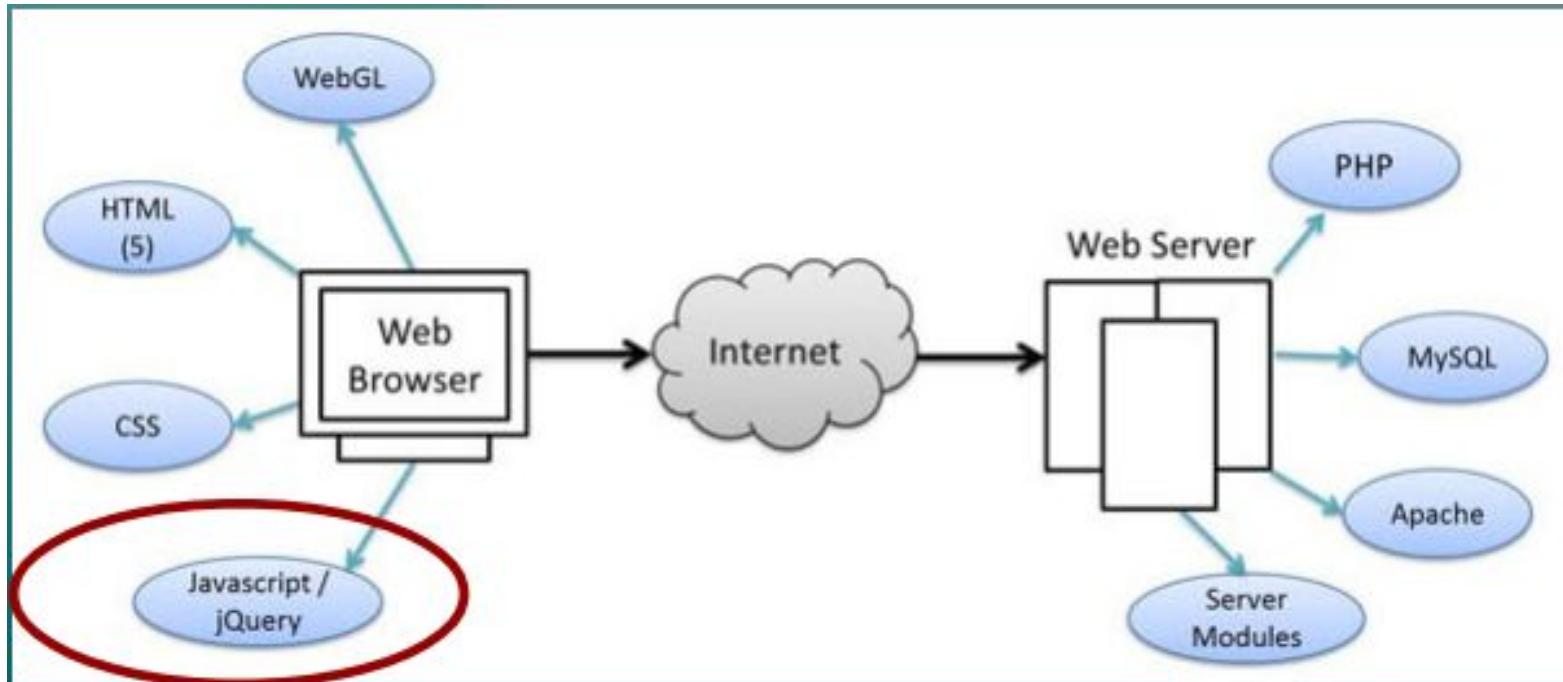
Es va anomenar JavaScript en referència al llenguatge **JAVA** (***encara que no té cap relació amb JAVA***), ja que comercialment pretenia ser la competència directa de Microsoft.

El navegador Netscape oferia suport per a Java i per al seu nou llenguatge JavaScript, que no té res a vore amb el primer.



Estava davall les ordes de màrqueting perquè
fora com Java, però no fer-ho massa gran ...
havia de ser un llenguatge germà menut.

1. Introducció (V)



1. Introducció (VI)

JavaScript s'ha mantingut als navegadors durant quasi una dècada, fins que es va iniciar la **revolució AJAX**: permetre que JavaScript recuperara dades dels servidors i actualitzara documents HTML sense necessitat d'una pàgina completa amb el cicle de petició-resposta.

Amb esta innovació, la funcionalitat de la interfície d'usuari ha arribat a ser **molt interactiva en el navegador** (GMail o Google Maps)

No ha deixat de créixer, desenvolupant-se **frameworks** per al seu ús tant en el costat de **client** en un inici, i ara també en el **servidor**.

1. Introducció (VII)

MEAN STACK

VS

LAMP STACK



MongoDB



Express.js



AngularJS



Node.js



Linux (OS)



Apache



My SQL



PHP

1.1 Que es un framework

És una caixa de ferramentes basada en un llenguatge de programació. Aglutinen **biblioteques** de codi i **patrons de disseny** avançats, i serveixen com a **base per al desenvolupament** d'aplicacions diverses.

Concretament, JavaScript té alguns Frameworks bastant coneguts i utilitzats: Vue, React, Angular, Ember, Backbone...

1.2 Que es una llibreria

Una llibreria serveix de suport a la nostra pàgina per incloure-hi alguna **dependència externa**. Com ara un slider, un calendari, entre d'altres.

JQuery és una llibreria que serveix com a suport per a l'escriptura de menys codi, simplificant la sintaxi del codi a utilitzar, ja que a més compatibilitza tots els navegadors.

1.3 Javascript en HTML

Pots fer pràcticament qualsevol cosa amb Javascript. Pots començar amb respostes a les pulsacions de botons, xicotetes coses com carrusels, galeries d'imatges,... Encara que també es poden crear jocs, animacions 2D i gràfics 3D, aplicacions integrades basades en BD.

Nosaltres l'utilitzarem per **afegir característiques interactives** al Lloc Web (esdeveniments que ocorren quan els botons són pressionats, en introduir dades als formularis, efectes d'estil dinàmics...)

2. Codi Javascript (I)

- Dins una pàgina web HTML, el codi JavaScript pot col·locar-se a diferents llocs, de manera similar a CSS.
 - En un fitxer independent.
 - Entre etiquetes <script> </script>.
 - Dins d'una etiqueta, anomenat script en línia (poc recomanable).

2. Codi Javascript (II)

■ Script en línia.

```
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <title>Primer exemple per a JavaScript</title>
</head>
<body>
    <p onclick="alert('Hola, què hi ha de nou?')">Estem estudiant JavaScript</p>
</body>
</html>
```

onclick – És un esdeveniment (event) que és el desencadenant de l'acció.

funció alert: Obre finestra amb missatge, valors,...

Argument = Missatge – acció a executar. S'expressa entre cometes. (Per què son simples les cometes internes?)

2. Codi Javascript (III)

■ Script en linea.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Attribute</h2>

<button onclick="document.getElementById('demo').innerHTML=Date()">The time
is?</button>

<p id="demo"></p>

</body>
</html>
```

2. Codi Javascript (IV)

■ Javascript dins de l'HTML.

- El més habitual és que l'script estiga abans que el CSS o al final del body
- En este cas, s'executarà només obrir el navegador, ja que no respon a cap esdeveniment.
- S'interpreta de dalt a baix. Atura l'execució del flux del programa.

```
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <title>Primer exemple per a JavaScript</title>
    <script>
        alert("Hola, què hi ha de nou?");
    </script>
</head>
<body>
    <p>Estem estudiant JavaScript</p>
</body>
</html>
```

2. Codi Javascript (V)

■ Javascript en fitxer extern.

Exercici:

- Còpia el codi de la diapositiva anterior.
- Després del primer missatge, mostra un segon que diga “Sóc el primer Script”
- Passa tot el codi JS al fitxer script.js i que continue funcionant.
- Afig algun comentari

```
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <title>Primer exemple per a JavaScript</title>
    <script src="script.js"></script>
</head>
<body>
    <p>Estem estudiant JavaScript</p>
</body>
</html>
```

2. Codi Javascript (VI)

■ Recordatori.

Només hem d'inserir al **<head>** el **codi imprescindible** i necessari, per tenir-lo carregat abans de carregar el cos de la pàgina.

Sempre que siga possible, per no afectar la velocitat de càrrega de la pàgina, evitant bloquejos, **es recomana** afegir l'element **script** just abans del tancament de l'etiqueta `</body>`

2. Codi Javascript (VII)

■ Càrrega asíncrona.

Tenim la possibilitat de carregar Javascript de manera asíncrona. Com? Molt senzill. Simplement has d'afegir l'atribut **async** al l'etiqueta script.

```
<script src="https://exemple.com/script.js" async
```

Això fa **que la pàgina carregue molt més ràpida** ja que el navegador carregarà simultàniament tant el fitxer HTML com el JavaScript.

2. Codi Javascript (VIII)

■ Evitar bloqueig del renditzar.

Async – permet a l'analitzador d'HTML descarregar el JavaScript mentre segueix analitzant la resta d'HTML. És a dir, no deixa completament d'analitzar mentre el fitxer es descarrega. No obstant això, farà una **pausa** a l'analitzador d'HTML per **executar el script una vegada que es descarregue el script**.

Defer – deixa que l'analitzador HTML descarregue el JavaScript mentre analitza la resta de l'HTML i espera a **executar** l'script fins que **l'anàlisi HTML s'acabe**.

2. Codi Javascript (IX)

■ **noscript**.

Alguns navegadors no disposen de suport complet de JavaScript, altres navegadors permeten **bloquejar-lo** parcialment i fins i tot alguns usuaris bloquegen completament l'ús de JavaScript perquè creuen que així naveguen de forma més segura.

En estos casos, si la pàgina web requereix JavaScript per al funcionament correcte, cal incloure un missatge d'avís a l'usuari indicant que hauria d'activar JavaScript per gaudir completament de la pàgina.

2. Codi Javascript (X)

■ **noscript.**

El llenguatge HTML defineix l'etiqueta **<noscript>** per mostrar un missatge a l'usuari quan el navegador no pot executar JavaScript. El codi següent mostra un exemple de l'ús de l'etiqueta **<noscript>**.

```
<body>
  <noscript>
    <p>Benvingut al meu lloc web.</p>
    <p>La pàgina que visualitzes requereix l'ús de JavaScript per al seu correcte
       funcionament</p>
  </noscript>
</body>
```

2. Codi Javascript (XI)

■ Hola Mundo!!.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Hola mundo en JavaScript en el propio documento</title>
<script type="text/javascript">
    alert("Hola mundo!");
</script>
</head>

<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```

3. Sintaxi Javascript (I)

■ Conjunt de regles.

- No es tenen en compte els espais en blanc i les noves línies: com passa amb HTML, l'intèrpret de JavaScript ignora qualsevol espai en blanc sobrant, per la qual cosa el codi es pot ordenar de manera adequada (tabulant les línies,...)
- Es distingeixen les majúscules i minúscules: Si en JavaScript s'intercanvien majúscules i minúscules, l'script no funciona.

3. Sintaxi Javascript (II)

■ Conjunt de regles.

- No es defineix el tipus de variables: en crear una variable, no cal indicar el tipus de dada que emmagatzemarà. D'esta manera, una mateixa variable pot emmagatzemar diferents tipus de dades durant l'execució de l'script.
- Tot i que JavaScript no obliga a fer-ho, és convenient finalitzar les sentències amb punt i coma (;).
- Es recomana incloure comentaris al codi.

```
// A continuació es mostra un missatge  
alert("Missatge de prova");
```

```
/* Els missatges amb varies línies  
són molt útils quan volem incloure  
molta informació dins del comentari */
```

3. Sintaxi Javascript (III)

¿CONOCES LA SINTAXIS DE JAVASCRIPT?

```
1 let language = 'JavaScript'  
2 let company = {  
3     name: 'EDteam',  
4     slogan: 'Nunca te detengas',  
5     founded: 2015  
6 }  
7 console.log(company.name)  
8 // 'EDteam'  
9 const getMajorNumber = (a,b) => {  
10    if (a > b) { return a }  
11    else { return b }  
12 }  
13 getMajorNumber(4,6)  
14 // 6
```



Las variables se declaran con let (no hay que indicar el tipo de dato)

Los objetos encierran entre llaves parejas con el formato propiedad: valor separadas por comas.

console.log() imprime en consola la expresión entre los paréntesis.

Para obtener el valor de una propiedad de un objeto se usa objeto.propiedad

Condicional (if / else)

Comentarios (líneas 8 y 14)

Ejecución de la función

Definición de función (se recomienda usar constantes con const)



A diferencia de Python, los saltos de línea e indentación no son parte de la sintaxis.

3. Sintaxi Javascript (V)

■ Declaracions: comunicació amb l'usuari.

```
alert("Hola!");  
document.write("Hola!");  
  
confirm("Vols continuar?");  
  
prompt("Introdueix la teua edat");  
  
console.log("Escriu en la consola");
```

document.title

3. Sintaxi Javascript (V)

■ Finestra alert.

- S'usa per mostrar un missatge en un quadre, després de realitzar alguna acció, que pot ser pressionar un enllaç, un botó, fer clic a un contenidor o qualsevol altre esdeveniment.
- Tenen la propietat modal: s'obre en primer pla i no es pot manipular res del que estiga en segon pla. Segons el navegador podem moure la finestra o no.

```
<button type="button" onclick="alert('Això és un quadre d'alerta')">Provar alerta</button>
```

- prova un "function" si no te funciona:

```
<button onclick="myFunction()">Provar alerta</button>

<script>
function myFunction() {
  alert("Això és un quadre d'alerta");
}
</script>
```

3. Sintaxi Javascript (VI)

■ Finestra confirm.

- La funció confirm és pareguda a les alertes, un xicotet quadre de diàleg amb un quadre de confirmació.
- La diferència és que ofereix dos opcions: "Acceptar" per confirmar el missatge i "Cancelar" si no s'està d'acord amb la petició expressada.
- En tots dos casos s'estableix una acció que cal fer.

3. Sintaxi Javascript (VII)

■ Finestra confirm.

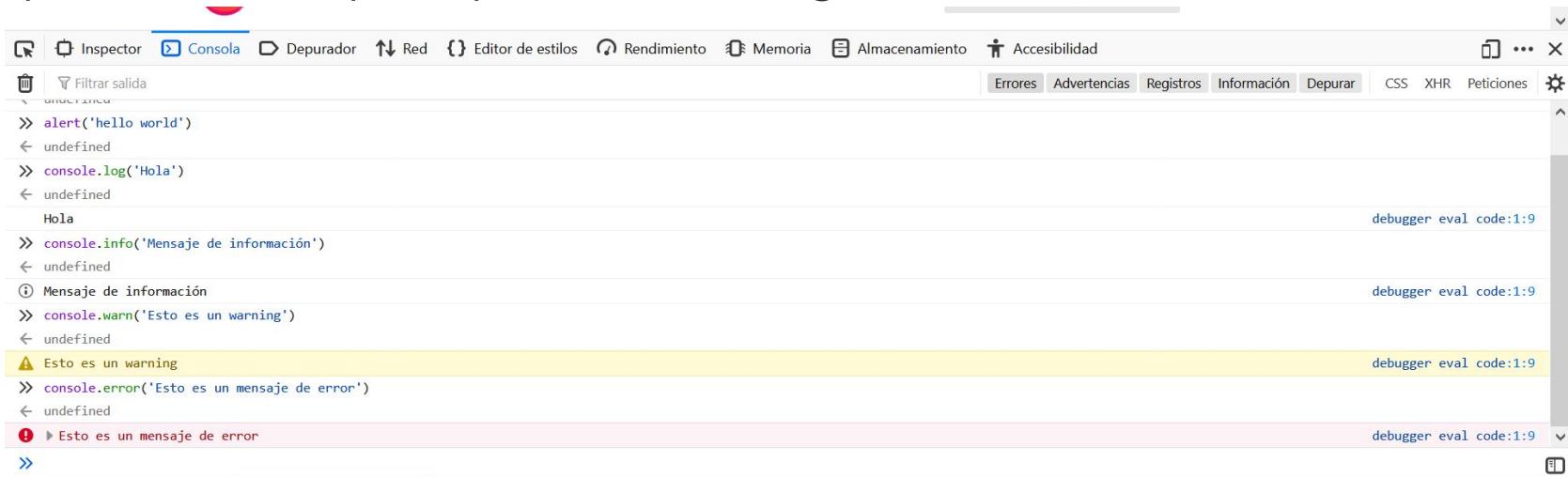
```
<script>
function confirmacio() {
    var pregunta = confirm("Voleu visitar la pàgina principal?")
    if (pregunta){
        alert("T'envie allà ràpidament")
        window.location = "index.html"
    }
    else{
        alert("Potser en un altre moment...\n Gràcies igualment!")
    }
}
</script>

<button type="button" onclick="confirmacio()">Confirmar!</button>
```

3. Sintaxi Javascript (VIII)

■ Consola – veure-la amb F12.

Des de qualsevol navegador, tenim opció d'utilitzar la consola per executar codi JavaScript i comprovar-ne el funcionament. A més, podem filtrar per tipus de missatge:



The screenshot shows the Firefox developer tools interface with the "Console" tab selected. The console output is displayed in three columns: Errors, Advertencias, and Registro. The "Registro" column contains the following entries:

- » alert('hello world')
← undefined
- » console.log('Hola')
← undefined
Hola
- » console.info('Mensaje de información')
← undefined
Mensaje de información
- » console.warn('Esto es un warning')
← undefined
⚠ Esto es un warning
- » console.error('Esto es un mensaje de error')
← undefined
❗ Esto es un mensaje de error
- »

Each entry includes a timestamp "debugger eval code:1:9". The "Registro" column has a yellow background for the warning and error messages.

3. Sintaxi Javascript (IX)

■ Tipus de missatges.

- **alert**. Mostra un missatge en una finestra emergent.
- **console**.
 - **log**. Es fa servir per a missatges de log.
 - **warn**. Missatges d'avertiment (warning).
 - **error**. Missatges d'error.
 - **info**. Missatges d'informació.
- **debugger**. Servirà per a depurar, aturant l'execució del nostre codi JavaScript.

Nota: Només utilitzarem el console per les pràctiques, a producció mai deixarem codi amb console...

3. Sintaxi Javascript (X)

■ Elements de codi. Comentaris

// per a una línia

/* per a
Diverses
línies */

3. Sintaxi Javascript (XI)

■ Elements de codi. Variables

Les **variables** són contenidors on podem emmagatzemar valors. Primer hem de declarar la variable amb la paraula clar let (o var), seguida del nom que vulguem donar-li.

```
let x = 5;  
let y = 6;  
let z = x + y;
```

```
let suma;  
suma = 5 + 4;
```

• Tipus

- **Globals.** Es defineixen per a l'entorn global.
- **Locals.** Definides dins d'un entorn concret.

3. Sintaxi Javascript (XII)

■ Elements de codi. Variables

• Declaracions

- **var.** Ús per a les variables generals i scope (entorn d'execució) i s'hereda per referència, per això pot ser usada per a variables globals.
- **let.** Ús per a variables generals i només pot ser usada en el bloc en el que s'ha declarat.
- **const.** Ús per a les dades que no poden ser sobreescrites (constants).

3. Sintaxi Javascript (XIII)

■ Elements de codi. Variables

```
» let missatge = 'hola mon'  
← undefined  


---

» let alumnes = 'Els alumnes de DAW són molt atents'  
← undefined  


---

» missatge  
← "hola mon"  


---

» alumnes  
← "Els alumnes de DAW són molt atents"  


---

» const benvinguts = "Este missatge és un " + missatge  
← undefined  


---

» benvinguts  
← "Este missatge és un hola mon"
```

3. Sintaxi Javascript (XM)

■ Elements de codi. Variables

- **Definicions**
- **camelCase**. Quan la variable fa referència a diverses paraules.
 - Ex: alumnatPrimer, alumnatSegon
- **lowercase**. Quan la variable es refereix a una paraula només.
 - Ex: alumnat
- Les constants se solen definir en majúscules
 - const EXEMPLE. Per a una paraula.
 - const EXEMPLE_VARIABLES. Per a diverses paraules.

3. Sintaxi Javascript (XV)

■ Elements de codi. Variables

- Exemples

```
// VARIABLES NUMÈRIQUES
let descompte = 21;
let preuTotal = 234.65;

// VARIABLES BOOLEANES
let clientRegistrat = false;
let majorEdat = true;

// VARIABLES DE CADENES DE TEXT
let missatge = 'Benvinguts a la web';
let nomProducte = 'Teclat SX32';
let text = "Cometes 'simples' dins de dobles";
let missatge2 = `Producte ${nomProducte} esgotat`;
let lletra = 'c';
```

```
// VARIABLES DE TIPUS LLISTA O VECTORS
let nom = "Pep";
let vector = ["Maria", nom, 33, true];

// OBJECTES
let persona = {
    nom: nom,
    cognoms: "Gil Berenguer",
    aficions: ["html", "css", "javascript"],
    inscrit: 1
}
```

3. Sintaxi Javascript (XVI)

- Elements de codi. Variables
- Exercici classe:

Nom	Escape
Cometa simple	\'
Cometa doble	\\"
Barra invertida	\
Nova línia	\n
Tabulador horizontal	\t
Tabulador vertical	\v
Salt de pàgina	\f
Retorn de carro	\r
Retrocés	\b

- Fes que el navegador mostre este missatge. Primer guarda el missatge en una variable i després mostra el missatge.



3. Sintaxi Javascript (XVII)

■ Tipus de dades.

- **Numbers.** Nombres positius o negatius, decimals o no. Podem utilitzar els operadors aritmètics o de comparació.
- **Strings.** Textos (Cadenes de text).
- **Booleans.** True/false.
- **Undefined.** No definit.
- **(*)Null.** Valor nul.
- **Arrays**

```
<script>
    var mimatriz = ["rojo", "verde", "azul"];
    alert(mimatriz[0]);
</script>
```

```
let nombreAlumnes = 22;
let treballs = 8.23;
let examens = 7.15;
let mitjana = (treballs + examens) / 2;
```

3. Sintaxi Javascript (XVIII)

■ Operacions.

//+ sumar	console.log(2+2)
//- restar	console.log(3-1)
// / dividir	console.log(4/2)
// * multiplicar	console.log(3 * 2)
// < menor que	console.log(10 < 20)
// > més gran que	console.log(10 > 20)
// <= menor o igual	console.log(20 <= 20)
// >= major o igual que	console.log(10 >= 20)
// == igual que	console.log(10 == 20)
// != different de	console.log(10 != 20)
// === valors i el tipus	console.log(40 === 40)

3. Sintaxi Javascript (XIX)

■ Operaciones

Operadores	Sintaxis	Significado	Ejemplo
=	a=b	Asigna a a el valor de b	a=5
+=	a+=b equivale a a=a+b	Asigna a a el valor de a+b	a+=5
-=	a-=b equivale a a=a-b	Asigna a a el valor de a-b	a-=5
=	a=b equivale a a=a*b	Asigna a a el valor de a*b	a*=5
/=	a/=b equivale a a=a/b	Asigna a a el valor de a/b	a/=5
+	a+b	Evaluá al valor de a+b	2+2
-	a-b	Evaluá al valor de a-b	2-2
*	a*b	Evaluá al valor de a*b	2*2
/	a/b	Evaluá al valor de a/b	2/2
++	++a	Suma 1 a a y evalúa al valor de a+1	++2
	a++	Suma 1 a a y evalúa al valor de a	2++
--	--a	Resta 1 a a y evalúa al valor de a-1	--2
	a--	Resta 1 a a y evalúa al valor de a	2--
-	-a	Evaluá al valor de -a	-2
&&	a&&b	Y lógico, evalúa a true si ambos valores son verdaderos y a false en caso contrario.	
	a b	O lógico, evalúa a false si ambos valores son falsos y a true en caso contrario	
!	!a	No lógico, evalúa a false si el operando es verdadero y a true en caso contrario.	
==	a==b	Evaluá a true si a y b son iguales y a false en caso contrario.	true==!false
>	a>b	Evaluá a true si a es estrictamente mayor que b y a false en caso contrario.	edad>65
>=	a>=b	Evaluá a true si a es mayor o igual que b y a false en caso contrario.	edad>=18
<	a<b	Evaluá a true si a es estrictamente menor que b y a false en caso contrario.	volumen<25.7
<=	a<=b	Evaluá a true si a es menor o igual que b y a false en caso contrario.	x<=y
!=	a!=b	Evaluá a true y a y b son distintos y a false en caso contrario.	anno!=2000

3. Sintaxi Javascript (XX)

■ Altres operacions.

- A més de les operacions aritmètiques i lògiques, a JavaScript tenim les operacions següents:
 - `typeof`. Retorna el tipus de variable.
 - `NaN`. Torna una cosa que no siga un número.
 - `isNaN()`. Comprova si no és un número.
 - `toString()`. Converteix un número en una cadena.
 - `.toFixed()`. Donem format específic al número. Arrodoneix i posa els decimals que nosaltres especifiquem.

3. Sintaxi Javascript (XXI)

■ Exemples pràctics. Exercici 1.

En un sopar d'amics arriba el compte i voleu dividir entre tots. Quant pagareu cadascú?

- a. Total: 102 €
- b. Persones: 6

3. Sintaxi Javascript (XXII)

■ Exemples pràctics. Exercici 1.

```
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="utf-8">
    <title>Exemple amics en JavaScript</title>
</head>
<body>
    <script type="text/javascript">
        /*Exemple sopar d'amics:
         Total: 102€
         Amics: 6
        */
        const TOTAL = 102
        const PERSONES = 6
        const TOTAL_PER_PERSONA = TOTAL / PERSONES
        console.log(TOTAL_PER_PERSONA)
    </script>
</body>
</html>
```

*Si no us funciona
proveu sense el type
de l'script
només <script>...

3. Sintaxi Javascript (XXIII)

■ Exemples pràctics. Exercici 2.

En un carro de la compra l'usuari ha triat diferents productes que has de sumar. I, pel fet de ser la primera compra, hauràs d'aplicar al total un 10% de descompte.

Mòbil (300€) + Cascos (30€) + Funda (10€)

Descompte 10%

3. Sintaxi Javascript (XXIV)

■ Exemples pràctics. Exercici 3.

Volem crear un convertidor d'unitats informàtiques de capacitat. Introduirem per teclat (prompt) la quantitat de MiB i ens ha de mostrar (document.write) la quantitat de KiB, bytes i bits a què equival.

Convertidor de MiB a KiB, bytes i bits

3.5MiB equivalen a 3584KiB

3.5MiB equivalen a 3670016 bytes

3.5MiB equivalen a 29360128 bits

3. Sintaxi Javascript (XXV)

■ Strings i operacions amb Strings.

Un String és una cadena de text.

Es poden concatenar cadenes de text usant l'operador +

Es poden realitzar operacions de comparació de cadenes.

- Concatenar (+)
- template literals. concatenar text amb variables dinàmiques.
- typeof
- .length
- .includes('an')
- .slice(start,end)
- .replace('este','per este')
- .trim()
- .split(',')

3. Sintaxi Javascript (XXVI)

■ Strings i operacions amb Strings.

```
<body>
  <script type="text/javascript">
    //concatenar text
    console.log('Hola +' Don Pepito')
    //template literals
    const NOM='Jose'
    console.log('Hola ' + NOM + ', bon dia')
    console.log(`Hola ${NOM}, bon dia`)
    //typeof
    console.log(typeof NOM)
    //length
    console.log(NOM.length)
    //includes()
    console.log(NOM.includes('s'))
    //slice(start,end)
    console.log(NOM.slice(2,3))
    //replace('esta','per esta')
    console.log(NOM.replace('os','an'))
    //trim()
    const TEXT='      aaaa dd aa ss aa oo '
    console.log(TEXT.trim())
    //split('')
    const DIRECCIO='Carrer major,n.7,4F'
    console.log(DIRECCIO.split(','))

  </script>
</body>
```

3. Sintaxi Javascript (XXVII)

■ Boolean.

Només admet dos valors: true o false. (true=1, false=0)

Molt útil per a comprovar certs estats de la nostra aplicació.

Aconsellable declarar-los en text positiu. Ex: userIsLogged

3. Sintaxi Javascript (XXVIII)

■ Boolean.

```
<body>
  <script type="text/javascript">
    //true
    console.log(Boolean(1))
    console.log(Boolean("el que siga"))
    console.log(Boolean(3.1))
    console.log(Boolean(5>4))
    console.log(Boolean(1<100))
    console.log(Boolean('1'==1))
    //false
    console.log(Boolean(0))
    console.log(Boolean(-0))
    console.log(Boolean(' '))
    console.log(Boolean(NaN))
    console.log(Boolean(null))
    console.log(Boolean(undefined))
    console.log(Boolean(1==='1'))
  </script>
</body>
```

3. Sintaxi Javascript (XXIX)

■ Boolean. Exercici classe 1.

Fes un programa que escriga en la pàgina un missatge diferent segons si l'usuari està connectat o no (una constant).

Si no està connectat: "T'has de registrar per a llegir l'article".

Si està connectat: "Fes clic aquí per a vore el contingut".

3. Sintaxi Javascript (XXX)

■ Boolean. Exercici classe 1.

```
<body>
  <p class="text"></p>
  <script type="text/javascript">
    const USER_LOGGED=true;
    const TEXT=document.querySelector('.text')
    if(USER_LOGGED){
      TEXT.innerHTML='<p>Fes clic <a href="#">aquí</a> per a vore el contingut'
    }else{
      TEXT.innerHTML="<p>T'has de registrar per a llegir l'article</p>"
    }
  </script>
</body>
```

3. Sintaxi Javascript (XXXI)

- **Null i undefined.**

undefined. Absència de valor.

Exemple: `console.log(a)`

```
console.log(typeof a) //undefined
```

null. Té valor, però és nul.

Exemple: `var c=null`

```
console.log(c)
```

```
console.log(typeof c) //object
```

3. Sintaxi Javascript (XXXII)

■ Arrays.

push(valor)—Aquest mètode agrega un nou valor al final de la matriu. L' atribut valor és el valor a ser agregat.

shift()—Aquest mètode remou el primer valor de la matriu i el retorna per ser processat .

pop()—Aquest mètode remou l'últim valor de la matriu i el retorna per ser processat .

```
<script>
    var mimatrix = [ "rojo" , 32 ];
    mimatrix.push('coche');
    alert(mimatrix[2]);
    alert(mimatrix.shift());
</script>
```

3. Sintaxi Javascript (XXXIII)

■ Condicionals i bucles.

```
<script>
    var mivariable = 9;
    if(mivariable < 10) {
        alert("El valor es menor que 10");
    }
</script>

<script>
    var mivariable = 9;
    if(mivariable < 10) {
        alert("El valor es menor que 10");
    }else{
        alert("El valor es igual que 10 o mayor");
    }
</script>
```

3. Sintaxi Javascript (XXXIV)

■ Condicionals i bucles.

```
<script>
    var mivariable = 9;
    switch(mivariable) {
        case 5:
            alert("El valor es cinco");
            break;
        case 8:
            alert("El valor es ocho");
            break;
        case 10:
            alert("El valor es diez");
            break;
        default:
            alert("El valor es " + mivariable);
    }
</script>
```

3. Sintaxi Javascript (XXXV)

■ Condicionals i bucles.

```
<script>
    var mivariable = 9;
    for(var f = 0; f < mivariable; f++) {
        alert("El valor actual es " + f);
    }
</script>
```

```
<script>
    var mivariable = 9;
    while(mivariable < 100) {
        mivariable++;
    }
    alert("El valor final de mivariable es " + mivariable);
</script>
```

```
<script>
    var mivariable = 9;
    do{
        mivariable++;
    }while(mivariable > 100);
    alert("El valor final de mivariable es " + mivariable);
</script>
```

3. Sintaxi Javascript (XXXVI)

■ Objectes.

Els objectes són com grans variables capaces de contenir altres variables (anomenades **propietats**) així com funcions (anomenades **mètodes**). Per declarar objectes, podem optar per diverses alternatives, tot i que el més simple és fer servir notació literal

```
<script>
  var miobjeto = {
    nombre: "Juan",
    edad: 30
  };
  alert("Mi nombre es " + miobjeto.nombre);
  alert("Yo tengo " + miobjeto['edad'] + " años de edad");
</script>
```

3. Sintaxi Javascript (XXXVII)

■ Objectes.

```
<script>
  var miobjeto = {
    nombre: "Juan",
    edad: 30
  };
  miobjeto.nombre = "Jorge";
  miobjeto.trabajo = "Programador";
  alert(miobjeto.nombre + " " + miobjeto.edad + " " + miobjeto.trabajo);
</script>
```

3. Sintaxi Javascript (XXXVIII)

■ Objectes.

```
<script>
var miobjeto = {
    nombre: "Juan",
    edad: 30,
    moto: {
        modelo: "Susuki",
        fecha: 1981
    },
    mostrarcoche: function(){
        alert(miobjeto.nombre + " posee una " + miobjeto.moto.modelo);
    }
};
miobjeto.mostrarcoche();
</script>
```

3. Sintaxi Javascript (XXXIX)

■ Objectes.

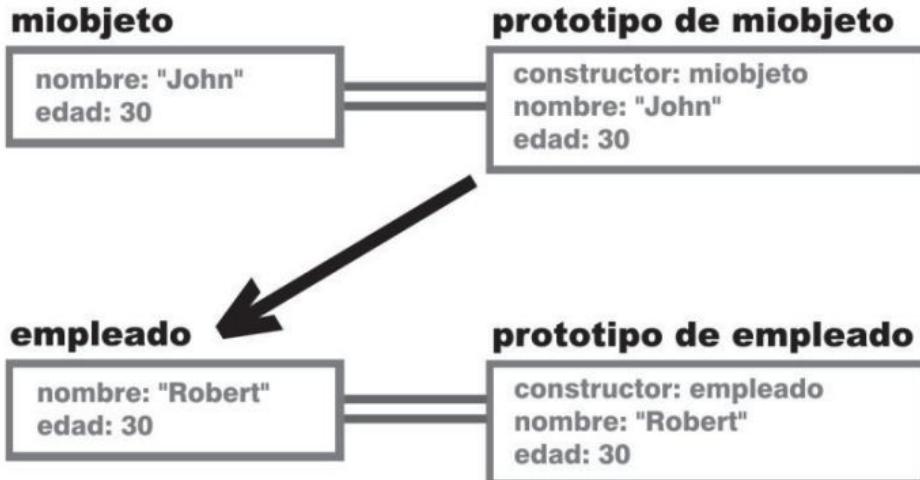


Figura 4-1: Herencia de prototipos

```
<script>  
var miobjeto = {  
    nombre: "Juan",  
    edad: 30,  
    mostrarnombre: function(){  
        alert(this.nombre);  
    },  
    cambiarnombre: function(nuevonombre){  
        this.nombre = nuevonombre;  
    }  
};  
var empleado = Object.create(miobjeto);  
empleado.cambiarnombre('Roberto');  
empleado.mostrarnombre();  
miobjeto.mostrarnombre();  
</script>
```

3. Sintaxi Javascript (XL)

■ Objectes.

```
<script>
    var miobjeto = {
        nombre: "Juan",
        edad: 30,
        mostrarnombre: function(){
            alert(this.nombre);
        },
        cambiarnombre: function(nuevonombre){
            this.nombre = nuevonombre;
        }
    };
    var empleado1 = Object.create(miobjeto);
    var empleado2 = Object.create(empleado1);
    var empleado3 = Object.create(empleado2);

    empleado2.mostraredad = function(){
        alert(this.edad);
    };
    empleado3.edad = 24;
    empleado3.mostraredad();
</script>
```

3. Sintaxi Javascript (XLI)

■ Objectes. Constructors

```
<script>
    function Miobjeto(nombreinicial, edadinicial){
        this.nombre = nombreinicial;
        this.edad = edadinicial;
        this.mostrarnombre = function(){
            alert(this.nombre);
        };
        this.cambiarnombre = function(nuevonombre){
            this.nombre = nuevonombre;
        };
    }
    var empleado = new Miobjeto("Roberto", 55);
    empleado.mostrarnombre();
</script>
```

4. Funcions (I)

- Una funció és una tasca que engloba la lògica que has definit. Determina un scope (àmbit).
- Les variables que es defineixen dins l'àmbit de la funció seran variables locals, encara que també es podran fer servir variables globals definides a l'àmbit superior.
- Es poden utilitzar paràmetres encara que no és obligatori, tant per rebre valors com per retornar-los. A més, poden tenir un valor per defecte.

4. Funcions (II)

```
<script type="text/javascript">
    //definició de la funció
    const MISSATGE='Hola món'
    function HolaMon (){
        console.log(MISSATGE)
    }
    //execució de la funció
    HolaMon()

    function suma(){
        const NUMERO=2
        console.log(NUMERO+NUMERO)
    }
    suma()
</script>
```

```
<script type="text/javascript">
    //funcions amb paràmetres
    function suma(a,b){
        console.log(a+b)
    }
    suma(2,5)
    suma(10,6)
    //funcions amb valor per defecte
    function dirHola(text='Món'){
        console.log('Hola ' + text)
    }
    dirHola()
    dirHola('alumnes')
    //funcions amb retorn
    function dirHola2(nom='Món'){
        return `Hola ${nom}`
    }
    console.log(dirHola2('classe de DAW'))
</script>
```

4. Funcions (III)

No tenen nom i es poden assignar a una variable.

```
<script type="text/javascript">
  var hola=function(nom='món'){
    return `Hola ${nom}`
  }
  console.log(hola())
  console.log(hola('alumnes'))
</script>
```

```
<body>
  <p id="text">Hola, això es un paràgraf de la meua web.</p>
  <script type="text/javascript">
    var comptador = 1;
    let paragraf = document.getElementById('text');
    paragraf.style.fontSize = comptador + "rem";
    paragraf.addEventListener("dblclick", function(){
      comptador++;
      paragraf.style.fontSize = comptador + "rem";
    })
  </script>
</body>
```

4. Funcions (V)

- No tenen nom. Són anònimes.
- La sintaxi és reduïda i mantenen l'àmbit (closure).
- Es pot utilitzar this. Poden tenir cos o no.

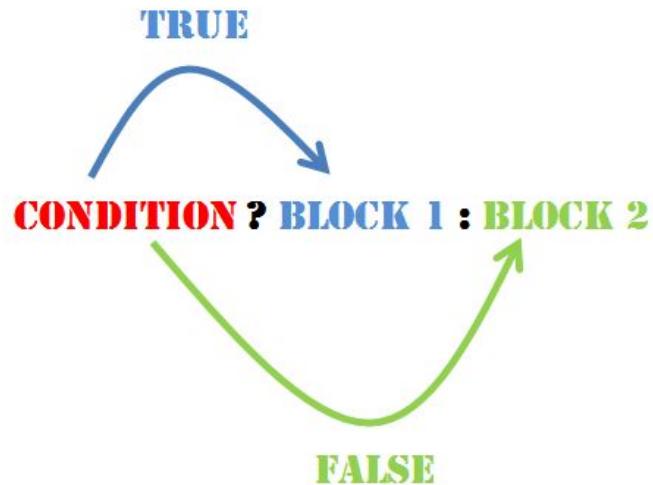
```
<script type="text/javascript">
    let dirHola=(nom)=>`Hola ${nom}`
    let dirAdeu=(nom)=>{return `Adéu ${nom}`}
    console.log(dirHola('classe de DAW'))
    console.log(dirAdeu('classe de DAM'))
</script>
```

```
<body>
    <p id="text">Hola, això es un paràgraf de la meua web.</p>
    <script type="text/javascript">
        var comptador = 1;
        let paragraf = document.getElementById('text');
        paragraf.style.fontSize = comptador + "rem";
        paragraf.addEventListener("dblclick", () => {paragraf.style.fontSize = ++comptador + "rem"})
    </script>
</body>
```

4. Funcions (V)

- Operadors ternaris:

Comprova una condició:



4. Funcions (VI)

- **Operadors ternaris:**

Comprova una condició:

```
<body>
  <script>
    const VOL = 180
    const HOTEL = 394
    const TOUR = 30
    const DIETA = 300
    const TOTAL = VOL + HOTEL + TOUR + DIETA
    const PRESSUPOST = 1000;
    const RESULTAT = PRESSUPOST >= TOTAL
      ? "Sí, me'n vaig de viatge"
      : "No, no me'n vaig de viatge"
    console.log(RESULTAT);
  </script>
</body>
```

```
<body>
  <script>
    const VOL = 180
    const HOTEL = 394
    const TOUR = 30
    const DIETA = 300
    const TOTAL = VOL + HOTEL + TOUR + DIETA
    const PRESSUPOST = 1000;
    console.log(PRESSUPOST >= TOTAL
      ? "Sí, me'n vaig de viatge"
      : "No, no me'n vaig de viatge")
  </script>
</body>
```

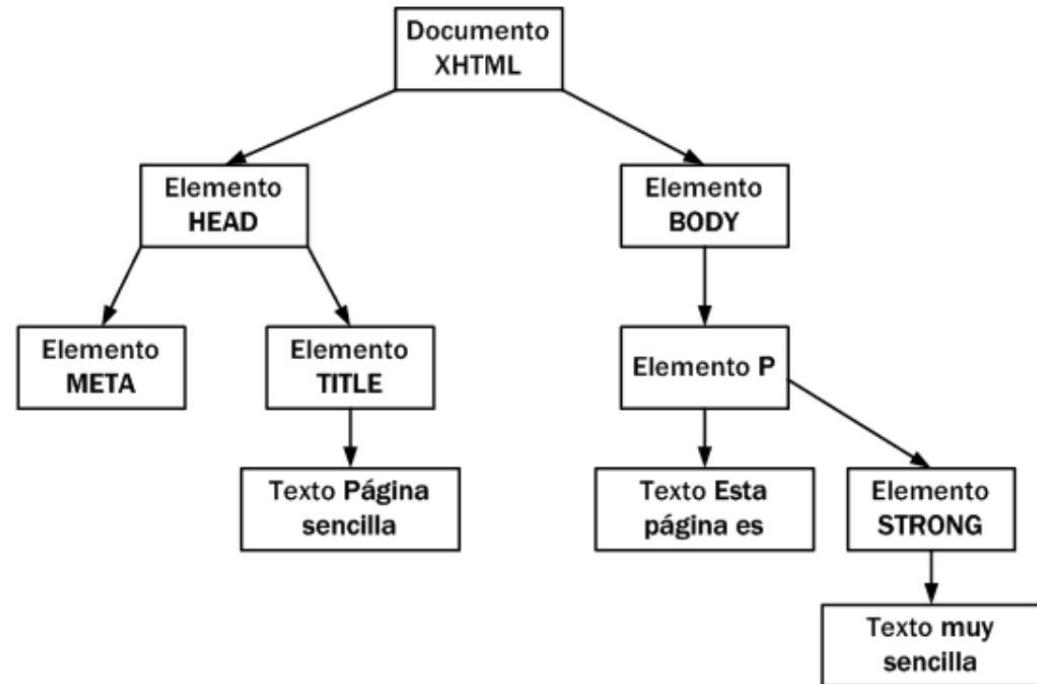
5. DOM - Arbre de nodes (I)

DOM permet als programadors web accedir i manipular les pàgines HTML com si foren documents XML

Tot i els seus orígens, DOM s'ha convertit en una utilitat disponible per a la majoria de llenguatges de programació (Java, PHP, JavaScript...) i les úniques diferències es troben en la forma d'implementar-lo.

5. DOM - Arbre de nodes (II)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Página sencilla</title>
</head>
<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

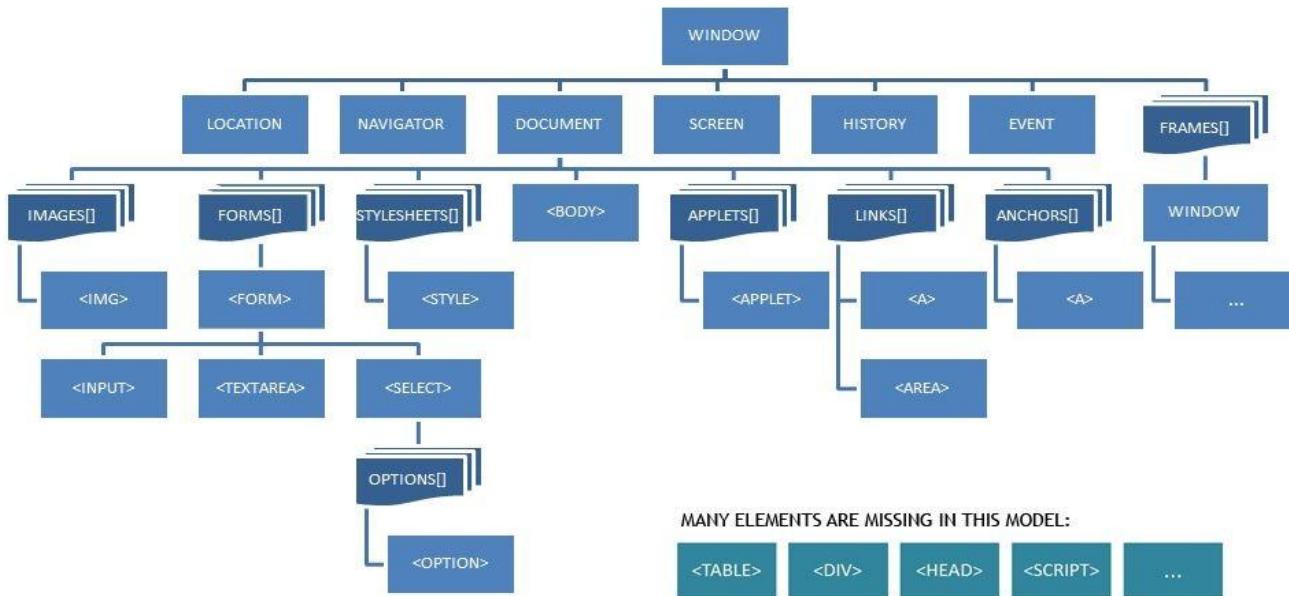


Cada rectangle representa un node DOM. Dins de cada node, se n'ha inclòs el tipus i el contingut

5. DOM - Arbre de nodes (III)

L'arrel de l'arbre de nodes de qualsevol pàgina HTML sempre és la mateixa: un node de tipus especial anomenat "**Document**". A partir d'eixe node arrel, cada etiqueta HTML es transforma en un node de tipus "Element". La conversió d'etiquetes en nodes es realitza de forma jeràrquica. D'esta manera, del node arrel només poden derivar els nodes HEAD i BODY. A partir d'esta derivació inicial, cada etiqueta HTML es transforma en un node que deriva del node corresponent a la seua "etiqueta pare".

5. DOM - Arbre de nodes (N)



```
<button onclick="window.history.go(-1)">Anar arrere</button>
<button onclick="window.location='https://www.mozilla.org'">Web Mozilla</button>
<button onclick="document.body.style.backgroundColor='red'">Canvia de color</button>
```

5. DOM - Arbre de nodes (V)

- Accés directe als elements:

```
document.getElementsByTagName("valor")
```

```
document.getElementsByName("valor")
```

```
document.getElementsByClassName("valor")
```

```
document.getElementById("valor")
```

5. DOM - Arbre de nodes (VI)

□ **document.getElementsByTagName("valor"):**

- Per afegir codi JavaScript i que este codi siga el que identifique els elements.
- L'objecte document s'utilitza per a identificar els elements.
- El mètode **getElementsByName(name)**: Obté tots els elements de la pàgina HTML que tinguen el nom d'etiqueta igual al paràmetre que es passa a la funció i els emmagatzema en un array per a poder accedir a la posició de l'etiqueta que volem.
 - L'exemple següent mostra com obtindre tots els paràgrafs d'una pàgina HTML:
 - `let paragraphs = document.getElementsByTagName("p");`

5. DOM - Arbre de nodes (VII)

□ `document.getElementsByTagName("valor")`:

- Si volem identificar tots els elements d'un mateix tipus, cal recórrer l'array (amb un bucle for).

```
<body>
  <p>Primer paràgraf</p>
  <p>Segon paràgraf</p>
  <p>Tercer paràgraf</p>
  <script>
    // Guardem l'array d'elements "p"
    let paragrafs = document.getElementsByTagName("p")
    // Recorrem els elements de l'array
    for(let i=0; i<paragrafs.length; i++){
      console.log(paragrafs[i].innerHTML)
    }
  </script>
</body>
```

```
<body>
  <p>Primer paràgraf</p>
  <p>Segon paràgraf</p>
  <p>Tercer paràgraf</p>
  <script>
    // Guardem l'array d'elements "p"
    let paragrafs = document.getElementsByTagName("p")
    // Recorrem els elements de l'array
    for(let paragraf of paragrafs){
      paragraf.onclick = alert('Contingut: ' + paragraf.innerHTML)
    }
  </script>
</body>
```

5. DOM - Arbre de nodes (VIII)

□ **document.getElementById("identificador"):**

- La funció **getElementById()** és de les funcions més utilitzades quan es desenvolupen aplicacions web dinàmiques. Es tracta de la funció per accedir directament a un node i per llegir o modificar les seues propietats

Com que l'atribut id ha de ser únic per a cada element d'una mateixa pàgina, la funció només retorna el node desitjat.

Podem utilitzar elements amb l'atribut id per identificar elements en JavaScript.

Ex: **document.getElementById('important').style.color='blue'**

5. DOM - Arbre de nodes (IX)

- `document.getElementById("identificador")`:

```
<body>
    <p id="primer">Primer paràgraf</p>
    <p id="segon">Segon paràgraf</p>
    <p id="tercer">Tercer paràgraf</p>
    <button onclick="canvia()">Canvia el segon</button>
    <script>
        function canvia(){
            let segon = document.getElementById('segon')
            if(segon.style.color == 'red'){
                segon.style.color = 'black'
            }
            else{
                segon.style.color = 'red'
            }
        }
    </script>
</body>
```

5. DOM - Arbre de nodes (X)

- `document.getElementById("identificador")`:

```
<body>
    <p id="primer">Primer paràgraf</p>
    <p id="segon">Segon paràgraf</p>
    <p id="tercer">Tercer paràgraf</p>
    <button onclick="canvia()">Canvia el segon</button>
    <script>
        function canvia(){
            let segon = document.getElementById('segon')
            if(segon.style.color == 'red'){
                segon.style.color = 'black'
            }
            else{
                segon.style.color = 'red'
            }
        }
    </script>
</body>
```

5. DOM - Arbre de nodes (X)

□ **document.getElementsByName("name"):**

La funció **getElementsByName()** obté tots els elements de la pàgina HTML l'atribut name dels quals coincidísca amb el paràmetre que es passa a la funció.

Normalment l'atribut name és únic per als elements HTML que l'inclouen, per la qual cosa és un mètode molt pràctic per accedir directament al node desitjat als formularis. En el cas dels elements HTML radiobutton, l'atribut name és comú a tots els radiobutton, per la qual cosa la funció torna una col·lecció d'elements.

5. DOM - Arbre de nodes (XI)

□ **document.getElementsByClassName**:

- La funció **getElementsByClassName()** obté tots els elements de la pàgina HTML l'atribut class dels quals coincidís amb el paràmetre que es passa a la funció.
- Normalment l'atribut class és comú a diversos elements HTML, per això és un mètode molt pràctic per accedir directament als nodes que comparteixen una classe.

5. DOM - Arbre de nodes (XII)

□ `document.getElementsByClassName`:

```
<body>
  <p class="verd">Primer paràgraf</p>
  <p class="verd destacat">Segon paràgraf</p>
  <p class="destacat">Tercer paràgraf</p>
  <button onclick="canvia()">Canvia els verds</button>
  <script>
    function canvia() {
      let verds = document.getElementsByClassName("verd")
      for (let i=0; i<verds.length; i++) {
        if (verds[i].style.color != 'red') {
          verds[i].style.color = 'red'
        }
        else {
          verds[i].style.color = 'green'
        }
      }
    }
  </script>
</body>
```

5. DOM - Arbre de nodes (XIII)

□ Objecte Window:

setTimeout(función, milisegundos)—Aquest mètode executa la funció especificada en el primer argument després del temps en mil·lisegons especificat pel segon argument. Podem assignar-li una variable i després cancel·lar aquest procés usant el mètode clearTimeout() amb el nom de la variable entre parèntesis (per exemple, **var temps = setTimeout(function(){ alert("Hola"); }, 5000);**).

setInterval(función, milisegundos)—Aquest mètode és similar a l'economia, però cridarà a la funció sense parar fins que el procés és cancellat pel mètode clearInterval() (per exemple, **var temps = setInterval(function(){ alert("Hola"); }, 2000);**).

5. DOM - Arbre de nodes (XM)

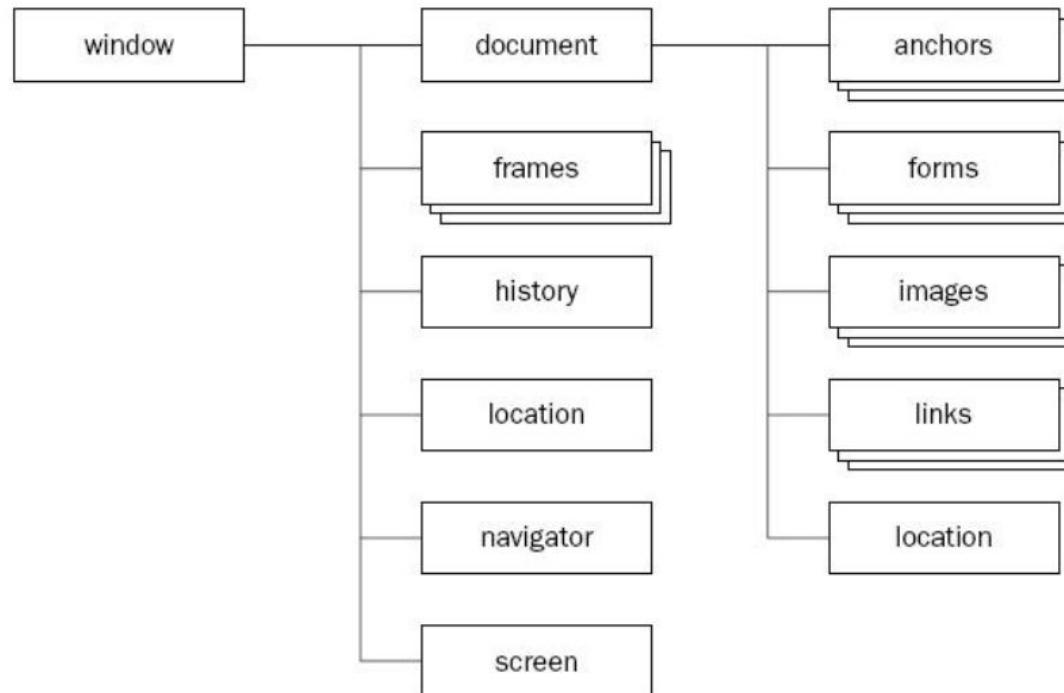
□ Objecte Window:

location—Aquest objecte inclou diverses propietats per retornar informació sobre l' adreça URL del document actual. Pot també ser usat com una propietat per declarar o retornar la URL del document (per exemple, **window.location** = "http://www.formasterminds.com").

history—Aquest objecte inclou mètodes per navegar a través de l' historial de la finestra. Alguns d'aquests mètodes són **back()**, per carregar el document anterior, **forward()**, per carregar un document més recent, i **go()**, per carregar qualsevol document en l'historial de la finestra.

5. DOM - Arbre de nodes (XV)

❑ Objecte Window:



6. innerHTML i outerHTML (I)

Element.innerHTML Retorna el contingut HTML, en format text, dels descendents de l'element.

Ex. alert(getElementById('primer').innerHTML)

També podrem modificar el contingut canviant el valor de l'element amb l'innerHTML.

Ex. getElementById('primer').innerHTML = "<label>holà</label>"

<https://developer.mozilla.org/es/docs/Web/API/Element/innerHTML>

6. innerHTML i outerHTML (II)

Element.outerHTML Retorna el contingut HTML, en format text, d'un element HTML, incloent-hi el mateix element HTML.

Ex. alert(getElementById('primer').outerHTML)

<https://developer.mozilla.org/es/docs/Web/API/Element/outerHTML>

6. innerHTML i outerHTML (III)

Exemple:

```
<div id="un"><strong>Això</strong> és un <a href="#">Exemple</a></div>
```

- **innerHTML** de l'element #un ens tornaria l'html *interior* de l'etiqueta: `Això` és un `Exemple`
- **outerHTML** ens retornaria l'HTML complet de l'element: `<div id="un">Això és un Exemple</div>`
- **innerText** ens tornaria el text interior de l'element: Això és un exemple
- **outerText** no és un atribut estàndard, però habitualment ens tornaria el text interior (com ara `innerText`). La diferència és que, si se li assigna alguna cosa, reemplaçarà tot l'element en comptes de només el contingut.

7. Invocar funcions senzilles amb href:

- Les funcions quan són senzilles, també es poden invocar amb un enllaç, incloent javascript i la funció en la propietat **href**.
- Alguns exemples. Prova'ls.
- ` Missatge`
`<!-- Missatge amb alerta -->`
- `Imprimir`
- `Refrescar`

8. Nous mètodes selectors per a Javascript (I)

- Utilitza selectors CSS3 per a identificar elements HTML5.
 - **querySelector()**. Retorna el primer element que correspon al grup de selectors indicat entre parèntesis.

```
<body>
  <p class="destacat">Primer paràgraf</p>
  <section>
    <p class="destacat">Segon paràgraf</p>
    <p class="destacat">Tercer paràgraf</p>
  </section>
  <button onclick="canvia()">Executa</button>
  <script>
    function canvia(){
      let element = document.querySelector("section > .destacat")
      element.style.backgroundColor = "pink";
    }
  </script>
```

8. Nous mètodes selectors per a Javascript (II)

- Aquest exemple es basa en els selectors d'herència.

```
<body>
    <p>Primer paràgraf</p>
    <section id="principal">
        <p>Segon paràgraf</p>
        <p>Tercer paràgraf</p>
    </section>
    <button onclick="canvia()">Executa</button>
    <script>
        function canvia(){
            let element = document.querySelector("#principal p:last-child")
            element.style.backgroundColor = "pink";
        }
    </script>
</body>
```

8. Nous mètodes selectors per a Javascript (III)

- Aquest exemple es basa en els selectors d'herència.

```
<body>
    <p>Primer paràgraf</p>
    <section id="principal">
        <p>Segon paràgraf</p>
        <p>Tercer paràgraf</p>
    </section>
    <button onclick="canvia()">Executa</button>
    <script>
        function canvia(){
            let element = document.querySelector("#principal p:last-child")
            element.style.backgroundColor = "pink";
        }
    </script>
</body>
```

8. Nous mètodes selectors per a Javascript (N)

- **querySelectorAll()**. Retorna tots els elements que corresponen al grup de selectors indicat entre parèntesis. El valor tornat és un Array contenint els elements en el mateix ordre en què apareixen al document.

```
<body>
  <p class="destacat">Primer paràgraf</p>
  <section>
    <p class="destacat">Segon paràgraf</p>
    <p class="destacat">Tercer paràgraf</p>
  </section>
  <button onclick="canvia()">Executa</button>
  <script>
    function canvia(){
      let elements = document.querySelectorAll("section > .destacat")
      for(let element of elements){
        element.style.backgroundColor = "pink"
      }
    }
  </script>
</body>
```

8. Nous mètodes selectors per a Javascript (V)

```
<body>
    <p class="destacat">Primer paràgraf</p>
    <section>
        <p class="destacat">Segon paràgraf</p>
        <p class="destacat">Tercer paràgraf</p>
    </section>
    <button onclick="canvia()">Executa</button>
    <script>
        function canvia(){
            let r = Math.floor(Math.random() * 256)
            let g = Math.floor(Math.random() * 256)
            let b = Math.floor(Math.random() * 256)
            let elements = document.querySelectorAll("section > .destacat")
            for(let element of elements){
                element.style.backgroundColor = `rgb(${r},${g},${b})`
            }
        }
    </script>
</body>
```

8. Nous mètodes selectors per a Javascript (VI)

- **Selectors de grup.** Per poder executar més d'un tipus de selector, podem fer servir una coma i així podem apuntar a diversos elements alhora.

```
<body>
    <p class="destacat">Primer paràgraf</p>
    <section>
        <p class="destacat">Segon paràgraf</p>
        <p class="destacat">Tercer paràgraf</p>
    </section>
    <p class="verd">Paràgraf final.</p>
    <button onclick="canvia()">Executa</button>
    <script>
        function canvia(){
            const elements = document.querySelectorAll("section, .verd")
            for(let element of elements){
                element.style.outline="2px dashed purple"
            }
        }
    </script>
</body>
```

9. Esdeveniments. Events (I)

- Definició d'esdeveniment: desencadenant de l'acció.
- Qui desencadena l'acció?
 - L'usuari (per exemple en fer clic – onclick).
 - El sistema (per exemple en carregar un document – window.onload)
- Manejador d'esdeveniments (**event handler**): codi que processa l'esdeveniment. Per exemple button.onclick
- Escoltador d'esdeveniments (**event listener**): mètode que espera a que un esdeveniment tinga lloc i diu quines accions tindran lloc. Per exemple button.addEventListener

9. Esdeveniments. Events (II)

Manejadors desdeveniments en línia.

```
<body>
  <button onclick="mostrar()">Executa</button>
  <script>
    function mostrar(){
      alert("alerta!")
    }
  </script>
</body>
```

Manejadors d'esdeveniments com a propietats.

```
<body>
  <button id="boto">Executa</button>
  <script>
    document.getElementById("boto").onclick = () => {
      alert("alerta!")
    };
  </script>
</body>
```

9. Esdeveniments. Events (III)

- Mètode **addEventListener()**. Nou en especificació HTML5 (l'opció més recomanada). És capaç de desencadenar diverses accions baix diferents esdeveniments

```
<body>
    <button id="boto">Executa</button>
    <script>
        function mostrarAlerta(){
            alert("Alerta!")
        }
        document.getElementById("boto").addEventListener("click", mostrarAlerta)
    </script>
</body>
```

9. Esdeveniments. Events (V)

```
<head>
    <meta charset="UTF-8">
    <title>Exemple</title>
    <style>
        img {width: 100px; height: 100px; border: 1px solid blue;}
    </style>
</head>

<body>
    
    <script>
        function creixer() {
            document.getElementsByTagName("img")[0].style.width = "400px"
            document.getElementsByTagName("img")[0].style.height = "400px"
        }
        function minvar() {
            document.getElementsByTagName("img")[0].style.width = "200px"
            document.getElementsByTagName("img")[0].style.height = "200px"
        }
        document.getElementsByTagName("img")[0].addEventListener("mouseover", creixer)
        document.getElementsByTagName("img")[0].addEventListener("mouseout", minvar)
    </script>
</body>
```

9. Esdeveniments. Events (V)

Efecte per a diverses imatges.

Haurem d'incloure un bucle for per a assignar a totes les imatges un escoltador d'esdeveniments.

```
<head>
    <meta charset="UTF-8">
    <title>Exemple</title>
    <style>
        img {width: 200px; height: 200px; border: 1px solid black; border-color: blue; border-style: solid; border-width: 1px; border-radius: 50%;}
    </style>
</head>
<body>
    
    
    
<script>
    let imatges = document.getElementsByTagName("img");
    for(let i=0; i<imatges.length; i++){
        imatges[i].addEventListener("mouseover", () => {
            imatges[i].style.width = "400px";
            imatges[i].style.height = "400px";
        })
        imatges[i].addEventListener("mouseout", () => {
            imatges[i].style.width = "200px";
            imatges[i].style.height = "200px";
        })
    }
</script>
</body>
```

9. Esdeveniments. Events (VI)

Efectes en éléments niuats.

Farem servir un tercer paràmetre de la funció addEventListener per a donar prioritat a l'execució d'un esdeveniment sobre un altre.

```
<body>
  <section>
    
    <p>Un text del section</p>
  </section>
  <script>
    let section = document.querySelector("section");
    let imatge = document.querySelector("img");
    section.addEventListener("click", socSection, true)
    imatge.addEventListener("click", socImatge, false)

    function socSection(){
      alert("Sóc el section")
    }
    function socImatge(){
      alert("Sóc la imatge")
    }
  </script>
</body>
```

9. Esdeveniments. Events (VII)

Events de ratolí.

Evento	Descripción
click	Se produce cuando se pulsa el botón izquierdo del ratón. También se produce cuando el foco de la aplicación está situado en un botón y se pulsa la tecla ENTER
dblclick	Se produce cuando se pulsa dos veces el botón izquierdo del ratón
mousedown	Se produce cuando se pulsa cualquier botón del ratón
mouseout	Se produce cuando el puntero del ratón se encuentra en el interior de un elemento y el usuario mueve el puntero a un lugar fuera de ese elemento
mouseover	Se produce cuando el puntero del ratón se encuentra fuera de un elemento y el usuario mueve el puntero hacia un lugar en el interior del elemento

9. Esdeveniments. Events (VIII)

Events de ratolí i teclat.

Evento	Descripción
mouseup	Se produce cuando se suelta cualquier botón del ratón que haya sido pulsado
mousemove	Se produce (de forma continua) cuando el puntero del ratón se encuentra sobre un elemento
contextmenu	Se produce cuando se pulsa el botón derecho

Evento	Descripción
keydown	Se produce cuando se pulsa cualquier tecla del teclado. También se produce de forma continua si se mantiene pulsada la tecla
keypress	Se produce cuando se pulsa una tecla correspondiente a un carácter alfanumérico (no se tienen en cuenta telas como SHIFT, ALT, etc.). También se produce de forma continua si se mantiene pulsada la tecla
keyup	Se produce cuando se suelta cualquier tecla pulsada

9. Esdeveniments. Events (IX)

Events de HTML.

Evento	Descripción
load	Se produce en el objeto window cuando la página se carga por completo. En el elemento cuando se carga por completo la imagen. En el elemento <object> cuando se carga el objeto
unload	Se produce en el objeto window cuando la página desaparece por completo (al cerrar la ventana del navegador por ejemplo). En el elemento <object> cuando desaparece el objeto.
abort	Se produce en un elemento <object> cuando el usuario detiene la descarga del elemento antes de que haya terminado
error	Se produce en el objeto window cuando se produce un error de
	JavaScript. En el elemento cuando la imagen no se ha podido cargar por completo y en el elemento <object> cuando el elemento no se carga correctamente
select	Se produce cuando se seleccionan varios caracteres de un cuadro de texto (<input> y <textarea>)

9. Esdeveniments. Events (X)

Events de HTML.

Evento	Descripción
change	Se produce cuando un cuadro de texto (<input> y <textarea>) pierde el foco y su contenido ha variado. También se produce cuando varía el valor de un elemento <select>
submit	Se produce cuando se pulsa sobre un botón de tipo submit (<input type="submit">)
reset	Se produce cuando se pulsa sobre un botón de tipo reset (<input type="reset">)
resize	Se produce en el objeto window cuando se redimensiona la ventana del navegador
scroll	Se produce en cualquier elemento que tenga una barra de scroll, cuando el usuario la utiliza. El elemento <body> contiene la barra de scroll de la página completa
focus	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento obtiene el foco
blur	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento pierde el foco

9. Esdeveniments. Events (XI)

L' objecte event.

[El objeto event | Introducción a AJAX | LibrosWeb.es \(um.es\)](#)

Tipus d'events

[Tipos de eventos | Introducción a AJAX | LibrosWeb.es \(um.es\)](#)

10. Noves funcionalitats (I)

- Sabem que CSS suporta **animacions** però les de JS poden arribar a extrems que amb CSS no pot...

↗ Handle CSS3 Animations

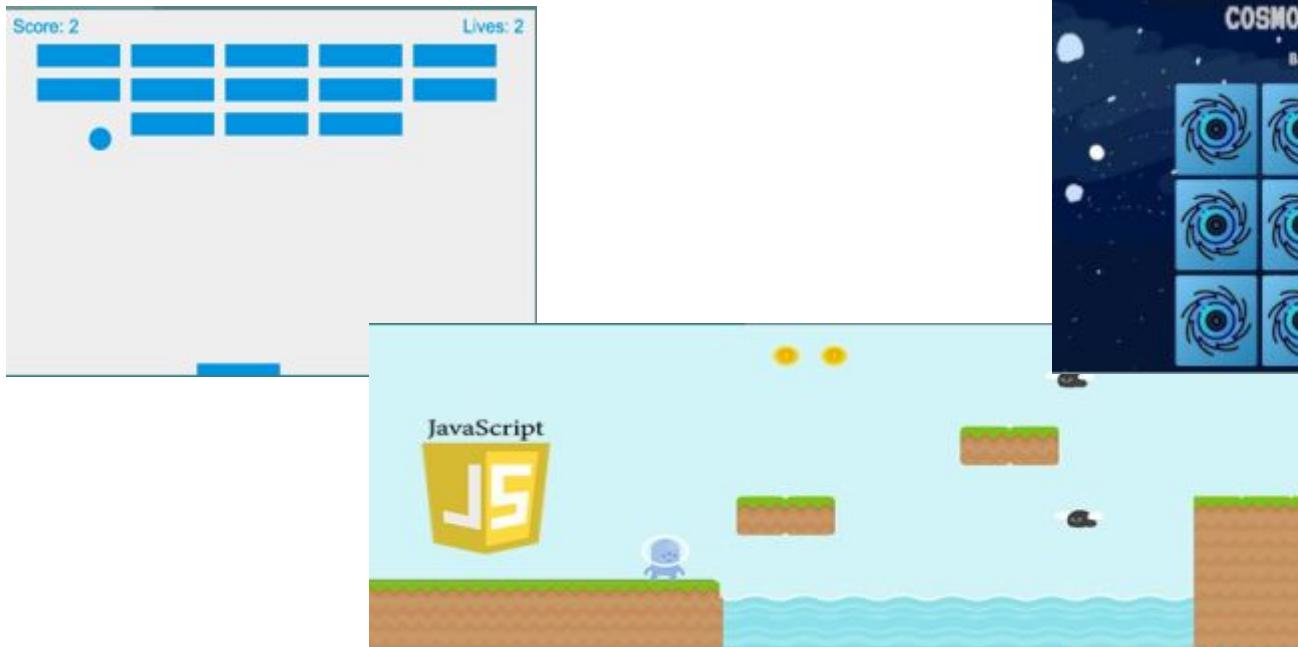
* If click, On Square, Do fade animated To .container-box

```
div[data-mouse="if: click, do: fade animated, to: .container-box"]
```



10. Noves funcionalitats (II)

- **Jocs.** Des de l'aparició del component JS anomenat **Canvas** és possible renderitzar em pantalla el codi que dibuixi elements.



11. Bibliografía

[1] Referències web en cada apartat de la presentació.

A continuació oferim en ordre alfabètic els autors que han fet aportacions a aquest document:

- Silvia Amorós Hernández
- Miguel Ángel Tomás
- José Más Davó
- José Javier Segura Martínez