


	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Entorns de desenvolupament	CFGs DAW/DAM - 1r curs

UD8: JavaScript

Índex

8.1 Introducció.....	2
8.1.1 Inclusió de codi JavaScript als documents HTML.....	4
8.2 Sintaxi bàsica.....	6
8.2.1 Comentaris en JavaScript.....	6
8.2.2 Variables.....	7
8.2.3 Tipus de dades bàsics.....	9
8.2.4 Operadors.....	11
8.2.5 Arrays.....	16
8.2.6 Estructures de control condicionals.....	24
8.2.7 Estructures de control iteratives.....	26
8.2.8 Funcions.....	28
8.2.9 Objectes.....	29
8.3 ECMAScript 6.....	35
8.4 El BOM i el DOM.....	45
8.4.1 L'objecte document.....	47
8.4.2 Tipus de nodes.....	49
8.4.3 Selecció d'elements del DOM.....	49
8.4.4 Manipulació d'elements.....	51
8.4.5 Manipulació del contingut dels elements.....	54
8.4.6 Manipulació d'atributs.....	55
8.4.7 Modificar el CSS.....	57
8.4.8 Navegar pel DOM.....	60
8.4.9 Obtenir atributs data.....	61
8.5 Gestió d'esdeveniments.....	62
8.5.1 Captura d'esdeveniments.....	62
8.5.2 L'objecte d'esdeveniment.....	67
8.5.3 Tipus d'esdeveniments.....	73
8.5.4 Formularis.....	78
8.6 LocalStorage.....	83
8.6.1 Creació, lectura, actualització, eliminació i neteja de registres.....	85
8.6.2 Com guardar valors que no són cadenes de caràcters amb JSON.....	86
8.6.3 Altres opcions.....	87


	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

8.1 Introducció

JavaScript (JS) és un llenguatge de programació dinàmic que quan s'aplica a un document HTML proporciona interactivitat a la pàgina web. Per exemple, dona feedback quan es clica un botó o s'emplena una informació en un formulari, s'apliquen estils dinàmics, s'afegeix animació, es poden crear jocs. Tècnicament, JavaScript és un **llenguatge de programació interpretat**. És a dir, no és necessari compilar els programes per executar-los ni necessita processos intermedis, ja que poden executar-se directament des del navegador.

JavaScript va néixer als anys noranta, quan s'iniciava el desenvolupament de les primeres aplicacions web i es començaven a incloure formularis cada cop més complexos. Al mateix temps, com que la velocitat de navegació era lenta, va sorgir la **necessitat d'un llenguatge de programació que s'executés del costat del client per evitar el temps d'espera de resposta del servidor**. La primera versió de JavaScript va ser un èxit. Per evitar guerres tecnològiques, es va decidir estandarditzar el llenguatge. Tanmateix, el 1997 l'especificació JavaScript 1.1 va ser enviada a l'ECMA (European Computer Manufacturers Association), que va crear un comitè amb l'objectiu d'estandarditzar un llenguatge d'script multiplataforma i independent de qualsevol empresa. Al llarg dels anys, JavaScript ha evolucionat i ha derivat en un llenguatge multiparadigma, multiplataforma i omnipresent en aplicacions web. S'han realitzat diverses actualitzacions a l'estàndard del llenguatge. Una de les actualitzacions més significatives va ser la introducció d'**ECMAScript 6** (ES6) al 2015, que va portar moltes característiques noves i millores.

Un tema interessant a comentar és que l'èxit de JavaScript va ajudar a desenvolupar un software anomenat **Node.js** (o Node) al 2009. La idea era crear un entorn independent del navegador, capaç d'interpretar codi JavaScript en el costat del servidor. Això també va tenir molt èxit, ja que implicava poder crear aplicacions de qualsevol tipus i competir amb els llenguatges clàssics de programació.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs


L'impuls donat per Node.js amb ajuda de llibreries i frameworks ha fet que JavaScript sigui capaç de:

- Crear aplicacions d'escriptori amb ajuda de frameworks com **Electron** o **NW.js**.
- Programar dispositius hardware amb ajuda de frameworks com Jhonny-Five, que té molt èxit en la programació de plaques **Arduino**, **Cylon.js** o **Node-Red**.
- Generar aplicacons mòbils natives. Avui en dia és possible programar una aplicació HTML5 emprant frameworks de JavaScript que permeten manipular el hardware del dispositiu. Alguns dels més populars són: **PhoneGap/Cordova**, **ionic**, **Flutter** o **ReactNative**.
- Ser el llenguatge de manipulació de Sistemes de Bases de Dades, com passa per exemple en els sistemes **MongoDB**.

Com a estàndard, JavaScript defineix un conjunt de regles que s'han de seguir per escriure el codi font correctament.

La sintaxi a seguir compleix els següents aspectes:

- No es tenen en compte els espais en blanc i les noves línies: l'interpret de JavaScript ignora qualsevol espai en blanc sobrant, de manera que el codi es pot ordenar de forma adequada per entendre-ho millor (tabulant les línies, afegint espais, creant noves línies, etc.)
- **Hi ha distinció entre majúscules i minúscules**: si s'intercanvien majúscules i minúscules, l'script no funciona.
- **No es defineix el tipus de les variables**: no cal indicar el tipus de dada que emmagatzema una variable. D'aquesta manera, una mateixa variable pot emmagatzemar diferents tipus de dades durant l'execució de l'script.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

- No cal acabar cada sentència amb el caràcter de punt i coma (;): encara que en JavaScript no és obligatori acabar cada sentència amb el caràcter de punt i coma (;), **és convenient** seguir la tradició.
- Permet incloure **comentaris**: els comentaris s'utilitzen per afegir informació en el codi font del programa. Tot i que el contingut dels comentaris no es visualitza per pantalla, sí que s'envia al navegador de l'usuari juntament amb la resta de l'script. Cal extremar les precaucions sobre la informació inclosa en els comentaris.

8.1.1 Inclusió de codi JavaScript als documents HTML

Hi ha tres maneres d'incloure codi JavaScript en un document HTML:

- **Incrustació de codi** (embedding code)

Per a afegir el codi JavaScript en les pàgines HTML, podem utilitzar l'etiqueta `<script>...</script>` per a envoltar el codi JavaScript dins de l'HTML.

Es pot fer dins del `<body>`:

```

<!DOCTYPE html>
<html>
<head>
  <title>JavaScript</title>
</head>
<body>
  <script>
    document.write("JavaScript al body");
  </script>
  <p>Incrustació de codi</p>
</body>
</html>

```

O en el `<head>`, depenent de l'estructura de la pàgina web que s'utilitzi:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```

<!DOCTYPE html>
<html>
<head>
  <title>JavaScript</title>
  <script>
    document.write("JavaScript en el tag head");
  </script>
</head>
<body>
  <p>Incrustació de codi</p>
</body>
</html>

```

- **Codi en línia** (inline code)

Normalment aquest mètode s'utilitza quan hem de cridar a una funció en els atributs d'esdeveniments HTML. Hi ha molts casos (o esdeveniments) en els quals podem afegir codi JavaScript directament, per exemple, l'esdeveniment **onClick**, sense emprar l'etiqueta `<script>...</script>`.

```

<!DOCTYPE html>
<html>
<head>
  <title>JavaScript</title>
</head>
<body>
  <p>
    <a href="#" onClick="alert('Hola!');">Fes clic aquí!</a>
  </p>
  <p>Codi JavaScript en línia.</p>
</body>
</html>

```

- **Fitxer extern** (external file)

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

També podem crear un fitxer independent per a guardar el codi de JavaScript amb l'**extensió (.js)** i posteriorment incorporar-lo en el nostre document HTML utilitzant l'atribut **src** de l'etiqueta `<script>`. Això resulta molt útil si volem fer servir el mateix codi en diversos documents HTML. També ens estalvia la tasca d'escriure el mateix codi una vegada i una altra i facilita el manteniment de les pàgines web. Encara que no reutilitzem codi, en general sempre tindrem el codi en un fitxer extern.

```
<html>
<head>
  <title>JavaScript</title>
</head>
<body>
  <form>
    <input type="button" value="Saluda" onclick="display()" />
  </form>
  <script src="hello.js"></script>
</body>
</html>
```


Aquest serà el fitxer JavaScript independent hola.js

```
function display() {
  alert("Hola!");
}
```

8.2 Sintaxi bàsica

8.2.1 Comentaris en JavaScript

Els comentaris en JavaScript s'empren per a afegir notes explicatives al codi. Existeixen dos tipus de comentaris:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

- **Comentaris d'una línia:** s'inicien amb `//` i tot el que ve després d'ells en la mateixa línia es considera un comentari.

```
// Això és un comentari d'una línia
```

- **Comentaris de diverses línies:** s'inicien amb `/*` i es tanquen amb `*/`. Tot el que es troba entre ells es tracta com un comentari.

```
/*
Això és un comentari
de diverses línies
*/
```

8.2.2 Variables

Les variables han de seguir les següents regles:


- Han de començar per una lletra, guió baix (`_`) o el símbol del dòlar (`$`).
- Després es poden emprar lletres, nombres o el guió baix.
- Es poden emprar lletres Unicode, tot i que no es trobin dins el codi ASCII.
- Es distingeixen majúscules de minúscules.

En JavaScript, les variables s'utilitzen per a emmagatzemar dades. Abans d'emprar una variable has de declarar-la... **o no!** En no haver-hi tipus no es necessita especificar res i, per tant, es pot definir una variable directament assignant un valor.

Declaracions

Per a declarar una variable en JavaScript, s'utilitza la paraula clau **var**, **let** o **const**, seguida del nom de la variable.

```
var nom; // Declaració d'una variable emprant 'var'
let edat; // Declaració d'una variable emprant 'let'
const PI = 3.1416; // Declaració de constant emprant 'const'
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

Les variables declarades amb **var** tenen un abast de funció, mentre que les variables declarades amb **let** i **const** tenen un abast de bloc. Alerta! Si s'omet **let** (o **var**), la variable es converteix en global:

```
numero = 42; // global!!
```

Diferències entre let i var

Anem a veure un parell d'exemples per entendre millor la diferència entre **let** i **var**.

```
{
    let x = 9;
}

console.log(x); //Uncaught ReferenceError: x is not defined
```

Si executam aquest codi, apareixerà aquest error: *Uncaught ReferenceError: x is not defined*. Això passa perquè la variable **x** s'ha definit dins un bloc amb la paraula clau **let** i no es pot emprar fora d'aquest bloc. Només existeix la variable dins l'àmbit del bloc on s'ha declarat.


Si en comptes de fer servir **let**, fem servir **var**, el resultat per consola serà **9**. En el cas de **var**, l'àmbit d'aplicació és major, tot i que també té restriccions quant al seu àmbit d'ús.

```
function f ()
    var x = 9;
}

console.log(x); //Uncaught ReferenceError: x is not defined
```

En executar aquest codi, tornam a tenir un error de variable no definida. Els blocs de codi que van dins una funció són més restrictius, les variables definides amb **var** no es poden fer servir fora.

Una segona diferència és la **possibilitat de tornar a declarar una variable dins un**

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGs DAW/DAM - 1r curs

mateix àmbit.

```
var x = 10
var x = 20; // Cap error
```

```
let y = 10
let y = 20; // Uncaught SyntaxError: redeclaration of let y
```

La paraula clau **var** permet tornar a declarar una variable dins un mateix àmbit, mentre que **let** no.

Per altra banda, la paraula reservada **const** es sol fer servir per a declarar constants. L'àmbit d'aplicació és igual que **let** i no es pot modificar el seu valor quan es tracta de tipus de dades primitius (nombre, cadena de text, booleà), però si és un objecte o un array, **les propietats o elements interns poden ser modificats**, però no es pot reassignar l'objecte o l'array complet.

Existeix una controvèrsia que porta a una d'aquestes eternes discussions de programadors, com el debat entre tabulacions i espais: en JavaScript no és imprescindible posar el ";" al final de cada sentència, tret que es vulguin posar diverses sentències en una mateixa línia. Així i tot, és aconsellable posar el ";" al final.

```
let nombre = 123
let nombre = 123; let comptador = 3
```

8.2.3 Tipus de dades bàsics

Vegem els tipus de dades bàsics en JavaScript:

- **Cadenes** (Strings): S'utilitzen per a representar text. Es poden declarar fent servir cometes simples o dobles i el *backslash* per a caràcters especials

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

(tabulacions `\t`, salts de línia `\n`, cometes dobles `\"`, cometes simples `'`, el caràcter ***backslash*** `\\`, etc.).

```
var nom = "Joan";
var missatge = 'Hola, com estàs?';
let cuento = "Quan es va despertar, el dinosaure encara
hi era.";

let amics = "Serem \"amics\"";
```

- **Nombres** (Numbers): S'utilitzen per a representar valors numèrics.

```
var edat = 25;
var preu = 10.99;
let ajudaArbitralAlMadrid = Infinity;
let vermellesPerACasemiro = -Infinity;
```

- **null**: És un valor existent en altres llenguatges també, que indica una cosa definida, però buit o de valor nul.

```
var valor = null;
```

- **undefined**: Amb un matís diferent, *undefined* indica que alguna cosa ni s'ha definit. Si es tracta d'emprar una variable que no ha estat definida o declarada, l'interpret de JavaScript dirà que no està definida.

```
var dada;

console.log(dada); // Output: undefined
```

- **NaN** (Not a Number): Es fa servir per a representar un valor que no és un número vàlid.

```
var resultat = "Hola" / 2;

console.log(resultat); // Output: NaN
```

- **Boolean**: S'empra per a representar valors de veritat (*true* o *false*).

```
var messi = true;
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
var penaldo = false;
```

Existeixen alguns valors que també es consideren fals i (o que JavaScript considera false), la qual cosa resulta útil per a crear expressions condicionals:

- null
- "" (String buit)
- undefined
- 0
- NaN

Existeixen dues funcions relacionades amb els tipus de dades: **typeof** i **delete**. *typeof* s'utilitza per a obtenir el tipus de dada d'una variable, i *delete* s'utilitza per a eliminar el contingut d'una propietat d'un objecte.

```
var nombre = 10;
console.log(typeof nombre); // Output: "number"
delete nombre; // Això no té cap efecte
console.log(typeof nombre); // Output: "number"
nombre = undefined; // Per a simular que "esborram" el valor
```

delete elimina propietats d'objectes:

```
var persona = {
  nom: "Joan",
  edat: 25
};
delete persona.edat;
console.log(persona); //Output: { nom: "Joan" }
```

8.2.4 Operadors

Els operadors s'empren per a realitzar operacions a variables i valors. Existeixen diferents tipus d'operadors: aritmètics, de comparació i booleans.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

Aritmètics

Els operadors aritmètics s'utilitzen per a realitzar operacions matemàtiques.

TIPUS	OPERADOR	EXEMPLE
Aritmètics	+ (suma)	$a + b$
	- (resta)	$a - b$
	* (multiplicació)	$a * b$
	/ (divisió)	a / b
	% (mòdul)	$a \% b$
Unaris	++ (increment)	$a++$, $++a$
	-- (decrement)	$a--$, $--a$
Assignació	+= (suma i assigna)	$a += b$ // $a = a + b$
	-= (resta i assigna)	$a -= b$ // $a = a - b$
	*= (multiplica i assigna)	$a *= b$ // $a = a * b$
	/= (divideix i assigna)	$a /= b$ // $a = a / b$
	%= (mòdul i assigna)	$a \% = b$ // $a = a \% b$
Canvi de signe	- (canvia el signe)	$a = 5$; $b = -a$; // $b = -5$

```
var a = 10;
```

```
var b = 5;
```

```
var suma = a + b; // 15
```

```
var resta = a - b; // 5
```

```
var multiplicacio = a * b; // 50
```

```
var divisio = a / b; // 2
```

```
var modul = a % b; // 0
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
console.log(suma, resta, multiplicacio, divisio, modul);
```

```
var x = 5;
x++; // x s'incrementa en 1 (x ara és 6)
console.log(x);
```

```
var y = 10;
y--; // y es redueix en 1 (i ara és 9)
console.log(y);
```

A més, també es pot fer una conversió de tipus. Per exemple, passar d'una cadena a un nombre. Existeixen diferents mètodes per a fer-ho: **parseInt** i **Math.trunc** (per a obtenir la part sencera d'un nombre en forma de cadena que tengui decimals), **parseFloat** (per a nombres amb decimals) i **Number** (que permet convertir tant nombres enters com decimals).

```
a = "5"; // Nota: és una cadena, no un número
let resultat = parseInt(a, 10); // La base és 10 per treballar amb decimals
console.log(resultat); // 5 (com a número)
```

```
b = "4.22";
resultat = parseFloat(b);
```

```
a = "3";
resultat = Number(a);
```

```
b = "6.7";
resultat = Math.trunc(b); // resultat = 6 (com a número)
```

De forma abreujada es pot aconseguir el mateix posant el símbol + per davant:

```
a = + "5";
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```
console.log(a); // "5";
console.log(+a); // 5
```

De comparació

Els operadors de comparació s'utilitzen per a comparar dos valors i retornar un resultat booleà (true o false).

OPERADOR	DESCRIPCIÓ	EXEMPLE
>	major que	5 > 4 // true
<	menor que	5 < 4 // false
>=	major o igual que	5 >= 4 // true
<=	menor o igual que	5 <= 4 // false
==	igual que	5 == 4 // false
!=	diferent de	5 != 4 // true
===	igual en valor i tipus	5 === 4 // false
!==	diferent en tipus	5 !== // true

```
var a = 5;
var b = 10;
```

```
console.log(a == b); // false
console.log(a != b); // true
console.log(a === b); // false
console.log(a !== b); // true
console.log(a > b); // false
console.log(a < b); // true
console.log(a >= b); // false
console.log(a <= b); // true
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```
console.log("2" == 2); // true
console.log("2" === 2); // false
```

```
console.log("true" == true); // false
console.log(1 == true); // true
```

```
console.log(undefined == null); // true
console.log(undefined === null); // false
console.log(undefined == false); // false
```

Booleans

Els operadors booleans permeten comparar expressions booleanes amb les quals es construeixen condicions que es poden aplicar en les funcions, bucles, etc.

OPERADOR	SÍMBOL	DESCRIPCIÓ	EXEMPLE
AND	&&	Només retorna true quan els dos operands ho són.	true && true // true true && false // false false && true // false false && false // false
OR		Retorna true si qualsevols dels dos operands ho és.	true true // true true false // true false true // true false false // false
NOT	!	Operador unari que retorna el contrari de l'operand.	!true // false !false // true

Es poden utilitzar operadors de comparació per a **inicialitzar variables booleanes**.

```
var a = 5;
var b = 10;

var esMajor = a > b; // false
var esMenor = a < b; // true
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
console.log(esMajor, esMenor);
```

Funció isNaN()

Hi ha problemes en JavaScript per a determinar si una expressió no és numèrica. Tot i que el valor **NaN** és un valor vàlid en JavaScript, i, per exemple, **"Hola" * 4** produeix un valor no numèric, la comparació (**"Hola" * 4**) == **NaN** dona un resultat fals.

Per això s'empra la funció `isNaN`. Aquesta funció rep una expressió i retorna vertader si l'expressió no és numèrica.

```
console.log(isNaN(NaN)); // true
console.log(isNaN("Hola" * 5)); // true
console.log(isNaN("3" * 5)); // false
```

8.2.5 Arrays

Un array és una estructura de dades que s'utilitza per a emmagatzemar múltiples valors en una sola variable. Pots accedir als elements d'un array fent servir el seu índex, que comença des de 0.

Inicialització d'arrays:

```
var numeros = [1, 2, 3, 4, 5];
console.log(numeros[0]); // 1
console.log(numeros[2]); // 3
var nombres = ["Joan", "María", "Pere"];
console.log(nombres[1]); // "María"
```

També es poden inicialitzar d'aquesta manera:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
let arrNumeros = new Array ();
arrNumeros[0] = 3;
arrNumeros[1] = 5;
arrNumeros[2] = 7;
arrNumeros[3] = 38;
arrNumeros[4] = 0;
arrNumeros[5] = -4;
arrNumeros[4] = 3;
```

```
let arrNoms = new Array ();
arrNoms[0] = 'Alicia';
arrNoms[1] = 'Jade';
arrNoms[2] = 'Aike';
```

O fins i tot així:

```
let arrNoms = new Array ('Alicia', 'Jade', 'Aike');
let arrPesos = new Array (34.5, 24.76, 45.6, 20.56, 45.4);
```

També es poden definir amb una grandària concreta. Si es passa un número, sense fer *new*, es genera un *array* d'elements buits amb aquesta grandària:

```
let arrNumeros = Array(5); // arrNumeros = [,,, ,]
```

O el que és millor, els arrays es poden inicialitzar amb una grandària i uns valors concrets:

```
let arrNumeros = Array(5).fill(0); // arrNumeros = [0, 0, 0, 0, 0]
```

No és necessari que els valors de cada posició de l'array siguin del mateix tipus, però sol ser així.

Mètodes d'arrays:

Existeixen uns quants mètodes per a treballar amb arrays:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

- **push()**: afegeix un o més elements al final de l'array.
- **pop()**: elimina l'últim element de l'array i el retorna.
- **shift()**: elimina el primer element de l'array i el retorna.
- **unshift()**: afegeix un o més elements al començament de l'array.
- **concat()**: combina dos o més arrays i retorna un nou array.
- **slice()**: retorna una còpia superficial d'un troç de l'array.
- **splice()**: canvia el contingut d'un array eliminando, reemplaçant o afegint elements.
- **indexOf()**: retorna el primer índex en el qual es troba un element donat en l'array, o -1 si no es troba.
- **join()**: uneix tots els elements d'un array en una cadena, utilitzant un separador especificat.
- **sort()**: ordena els elements d'un array alfabèticament (per a cadenes) o numèricament (per a números).

```
const fruites = ['poma', 'plàtan', 'taronja'];
fruites.push('rem');
console.log(fruites); // Output: ['poma', 'plàtan', 'taronja', 'rem']
```

```
const fruites = ['poma', 'plàtan', 'taronja'];
const eliminarPop = fruites.pop();
console.log(fruites); // Output: ['poma', 'plàtan']
console.log(eliminarPop); // Output: 'taronja'
```

```
fruites = ['poma', 'plàtan', 'taronja'];
const eliminarShift = fruites.shift();
console.log(fruites); // Output: ['plàtan', 'taronja']
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGs DAW/DAM - 1r curs

```
console.log(eliminarShift); // Output: 'poma'
```

```
fruits = ['poma', 'plàtan', 'taronja'];
fruits.unshift('rem');
console.log(fruits); // Output: ['rem', 'poma', 'plàtan',
'taronja']
```

```
const fruits1 = ['poma', 'plàtan'];
const fruits2 = ['taronja', 'rem'];
const combinacioFruits = fruits1.concat(fruits2);
console.log(combinacioFruits); // Output: ['poma', 'plàtan',
'taronja', 'rem']
```

```
fruits = ['poma', 'plàtan', 'taronja', 'rem'];
const copiaFruits = fruits.slice(1, 3);
console.log(copiaFruits); // Output: ['plàtan', 'taronja']
```

```
fruits = ['poma', 'plàtan', 'taronja'];
fruits.splice(1, 0, 'rem', 'kiwi');
console.log(fruits); // Output: ['poma', 'rem', 'kiwi',
'plàtan', 'taronja']
```

```
fruits.splice(2, 2, 'mango');
console.log(fruits); // Output: ['poma', 'rem', 'mango',
'taronja']
```

```
fruits = ['poma', 'plàtan', 'taronja'];
const taronjaIndex = fruits.indexOf('taronja');
console.log(taronjaIndex); // Output: 2
fruits = ['poma', 'plàtan', 'taronja'];
const unioElements = fruits.join(', ');
console.log(unioElements); // Output: "poma, plàtan, taronja"
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```
fruites = ['poma', 'plàtan', 'taronja', 'rem'];
fruites.sort();
console.log(fruit); // Output: ['plàtan', 'poma', 'rem',
'taronja']
```

Emprar correctament el mètode sort()

Atenció! El mètode **sort** utilitza el valor unicode de l'element a ordenar.

```
const numeros = [10, 5, 8, 3, 1];
numeros.sort();
console.log(numeros); // Output: [1, 10, 3, 5, 8]
```

Això també afecta a cadenes de text:

```
const ciutats = ['Zaragoza', 'madrid', 'Barcelona'];
ciutats.sort();
console.log(ciutats); //['Barcelona', 'Zaragoza', 'madrid']
```

Com ordenem correctament un array de números? I de cadenes, si hi ha majúscules, accents, etc...?

Per a solucionar això hem d'emprar **paràmetres** en el mètode sort.

El mètode `.sort()` té un únic paràmetre opcional que permet ajudar a aquest mètode per a realitzar l'ordenació del contingut. Aquesta és la clau perquè aquest mètode es comporti com ens interessa en cada cas.

Aquesta funció rep dos valors a comparar i com a resultat ha de:

- Retornar un valor positiu (1) si el primer valor és superior al segon.
- Retornar un valor negatiu (-1) si el primer valor és inferior al segon.
- Retornar un valor zero (0) si els dos valors són iguals o equivalents per a l'ordenació.

Anem a veure exemples d'ús per a:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

- **Emprar sort amb nombres.**

```
var numeros = [10, 5, 8, 3, 1]

// Sintaxi ES5
console.log(numeros.sort(function (a, b) {
    return a - b;
})))

// Sintaxis ES2015
console.log(numeros.sort((a, b) => a - b ))
```

La funció utilitza un petit truc i en restar un valor a un altre aconseguim que es retorni un valor positiu si a és major que b, un valor negatiu si a és menor que b, i 0 si tenen el mateix valor, per la qual cosa es compleix el requisit que ens imposen a l'hora de retornar valors en la funció de suport al mètode .sort().

- **Emprar sort amb cadenes de text**

Si volem fer alguna cosa semblant per a corregir el problema de majúscules i minúscules que hem vist, podem estar temptats a utilitzar alguna cosa d'aquest tipus:

- Si volem fer alguna cosa semblança per a corregir el problema de majúscules i minúscules que vam veure abans, podem estar temptats a utilitzar alguna cosa d'aquest tipus:

```
const ciutats = ["Zaragoza", "madrid", "Barcelona"];
ciutats.sort ((a, b) => a.toLowerCase () > b.toLowerCase
());
console.log (ciutats); // ['Zaragoza', 'madrid',
'Barcelona']
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

El resultat no és correcte perquè la nostra precipitada funció retorna true o false que és el que s'obté d'una comparació amb un operador major (>) i hem de recordar que la funció de suport a .sort() espera que retornem -1, 1 o 0. Perquè funcioni hauríem de fer alguna cosa d'aquest estil:

```
const ciutats = ["Zaragoza", "madrid", "Barcelona"];
ciutats.sort ((a, b) =>
  a.toLowerCase() > b.toLowerCase() ? 1 :
  a.toLowerCase() < b.toLowerCase() ? -1:
  0
);
console.log (ciutats);
// ['Barcelona', 'madrid', 'Zaragoza']
```

En realitat no hem acabat, ja que, si incorporem alguna lletra accentuada, tornem a tenir un resultat no desitjat:

```
const ciutats = ["Zaragoza", "madrid", "Barcelona",
  "Ávila"];
ciutats.sort ((a, b) =>
  a.toLowerCase() > b.toLowerCase() ? 1 :
  a.toLowerCase() < b.toLowerCase() ? -1:
  0
);
console.log (ciutats); // ['Barcelona', 'madrid',
  'Zaragoza', 'Ávila']
```

Seguim comparant sobre la base de la posició de cada lletra en el codi Unicode, que és el que aconseguim en fer servir l'operador major (>) amb cadenes. Una solució bastant senzilla i, en general, bastant desconeguda, és utilitzar el mètode ***String.prototype.localeCompare()*** que permet comparar dues cadenes tenint en compte accents i altres característiques específiques de cada idioma per a l'ordenació. El millor de tot és que aquesta funció

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

retorna -1, 1 o 0 segons si és major, menor o igual, que és exactament el que necessitem:

```
const ciutats = ["Zaragoza", "Ávila", "madrid",
"Barcelona"];
ciutats.sort((a, b) => a.localeCompare(b));
console.log(ciutats); // ['Ávila', 'Barcelona',
'madrid', 'Zaragoza' ]
```

L'ús de **.localeCompare()** és fonamental per a tenir una ordenació correcta i hem d'emprar-lo sempre que vulguem ordenar cadenes de text.

Ordenació de matrius amb objectes

Si la matriu conté objectes, podem emprar les propietats dels objectes per a fer la comparació, utilitzant **.localeCompare()**.

```
const ciutats = [
  {
    "municipi": "Zaragoza",
    "provincia": "Zaragoza",
  },
  {
    "municipi": "Ávila",
    "provincia": "Ávila",
  },
  {
    "municipi": "madrid",
    "provincia": "madrid",
  },
  {
    "municipi": "Barcelona",
    "provincia": "Barcelona",
  }
]
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```
];
```

```
ciutats.sort((a, b) => {
  return a.municipi.localeCompare(b.municipi)
});
console.log(ciutats);
```

Això mateix podem fer-ho amb dates i qualsevol altre tipus d'objecte que tinguem en les nostres estructures de dades.

8.2.6 Estructures de control condicionals

Les estructures de control s'utilitzen per a controlar el flux d'execució d'un programa.

if

L'estructura *if* es fa servir per a executar un bloc de codi si es compleix una condició. La condició es posa entre parèntesi i el bloc de codi dins claus. Si només hi ha una sentència, es poden ometre.

```
var edat = 18;

if (edad >= 18) {
  console.log("Ets major d\'edat");
}
```

if-else

L'estructura if-else s'empra per a executar un bloc de codi si es compleix una condició, i un altre bloc de codi si no es compleix la condició.

```
var edat = 16;

if (edat >= 18) {
  console.log("Ets major d\'edat");
} else {
```


	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```

        console.log("Ets menor d\'edat");
    }

```

if-else-if

L'estructura if-else-if s'utilitza per a avaluar múltiples condicions en ordre i executar el bloc de codi corresponent a la primera condició que es compleix. Convé utilitzar les claus (i indentació) per a evitar confusions.

```

var hora = 14;

if (hora < 12) {
    console.log("Bon dia");
} else if (hora < 18) {
    console.log("Bones tardes");
} else {
    console.log("Bona nit");
}

```

switch case

Quan es vol fer una estructura condicional segons el valor que tingui una variable o expressió, es pot utilitzar una estructura *switch-case*. A diferència d'altres llenguatges, en JavaScript es pot fer el switch-case sobre més tipus, a part dels nombres:

```

var nom = "frodo";

switch (nom) {
    case "gandalf":
        edat = 1230;
        break;
    case "aragorn":
        edat = 532;
}

```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```

        break;
    case "frodo":
    case "sam":
        edat = 34;
        break;
    default:
        edat = 1;
        break;
}

console.log("Nom: " + nombre + "\nEdat: " + edat);

```

Operador ternari

L'operador ternari `?` s'utilitza per a avaluar una condició i retornar un valor en funció de la condició, en un única línia.

```

var edat = 20;
var missatge = (edat >= 18) ? "Ets major d\'edat" : "Ets menor d\'edat";
console.log(missatge);

```

8.2.7 Estructures de control iteratives

Les estructures de control iteratives s'utilitzen per a repetir una porció de codi diverses vegades.

while

L'estructura *while* s'empra per a repetir un bloc de codi mentre es compleixi una condició.

```

var comptador = 0;

while (comptador < 5) {
    console.log(comptador);
}

```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```

    comptador++;
}

```

do-while

L'estructura *do-while* s'utilitza per a repetir un bloc de codi almenys una vegada i després mentre es compleixi una condició.

```

var comptador = 0;

do {
    console.log(comptador);
    comptador++;
} while (comptador < 5);

```

for

L'estructura *for* s'empra per a repetir un bloc de codi un nombre específic de vegades.

```

for (var i = 0; i < 5; i++) {
    console.log(i);
}

```

break / continue

La instrucció *break* s'empra per a aturar l'execució d'un bucle, mentre que la instrucció *continue* s'utilitza per a saltar a la següent iteració del bucle.

```

for (var i = 0; i < 5; i++) {
    if (i === 2) {
        break; // Atura el bucle quan i és igual a 2
    }
    console.log(i);
}
// Output: 0 1

```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
for (var i = 0; i < 5; i++) {
    if (i === 2) {
        continue; // Salta a la següent iteració quan i és
                    // igual a 2
    }
    console.log(i);
}
// Output: 0 1 3 4
```

8.2.8 Funcions

En JavaScript es pot agrupar codi en funcions. A diferència d'altres llenguatges, en JavaScript només hi ha funcions, no hi ha distinció de procediment i funcions. Això sí, les funcions no tenen per què retornar res. Poden tenir paràmetres i, igual que amb les variables, no s'indiquen els tipus.


```
function diguesHola () {
    console.log('Hola món');
}
```

També es poden emprar paràmetres:

```
function ambParametres (param1, param2) {
    let local = 0;
    local = param1 + param2;
    console.log("El resultat és: " + local);
}
```

I poden tenir retorn:

```
function multiplica (x, y, z) {
    let local = 0;
    local = x * y * z;
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```

    return local;
}

```

Per a cridar a les funcions, simplement posem el seu nom i els paràmetres, si els té, entre parèntesis:

```

function calcula () {
    let a = 4;
    let b = 45;
    let c = 2;
    let total = 0;
    total = multiplica(a, b, c);
}

```

8.2.9 Objectes

En la versió de JavaScript anterior a ES6 no existeixen les classes, però sí els objectes, els quals són estructures que es creen entre claus `{ }` i que s'utilitzen per a emmagatzemar múltiples valors relacionats com a propietats i mètodes. Les propietats són **parells de clau-valor**, i els mètodes són funcions que pertanyen a un objecte.

Claus i cometes

Les claus de les propietats d'un objecte poden ser cadenes de text (strings) o identificadors vàlids de JavaScript. Per a les claus es poden ometre les cometes, tret que necessitem utilitzar caràcters no ASCII, caràcters especials, espais en blanc, etc. En el següent exemple es mostra com accedir a diferents atributs d'un objecte de dues maneres diferents i com tractar els objectes niats. Com es pot veure, es pot jugar amb diverses combinacions.

```

let unClient = {
    nom: "Peter Jackson",
    "Adreça del client": "C/ Desconegut",
}

```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```

    "-+--+": "boquepasa",
    pagament: {
        tipus: "Visa",
        targeta: "33442324234",
        "data de caducitat": "mai"
    },
};
console.log(unClient);

unClient["nombre"] = "";
unClient["-+--+"] = "requete";
unClient.pagament["tipus"] = "compte";
unClient["pagament"].targeta = "666";
unClient["pagament"]["data de caducitat"] = 0;

console.log(unClient);

```

Cada element d'un objecte JSON pot ser de diversos tipus:

- number
- String
- boolean
- array
- Object (es pot niar)
- Function

Es poden niar i complicar tant com faci falta.

Mètodes com a dades

Si es necessita que un objecte JavaScript es comporti com els objectes d'altres llenguatges, se li ha d'afegir funcions o mètodes. És molt senzill, ja que una funció

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

és com una altra tipus de dada:

```
let estudiant = {
  id: 2,
  nom: "Peter Jackson",
  diguesHola: function () {
    return "Hola";
  },
};
console.log(estudiant);
console.log(estudiant.diguesHola()); // Hola
```

Afegir atributs i mètodes

És molt senzill afegir aquests elements, n'hi ha prou amb definir-los de la manera següent:

```
// Afegir noves propietats i mètodes:
estudiant.edat = 28;
estudiant.diAdeu = function () {
  return "Adéu";
};

console.log(estudiant.diAdeu());
```

this

Com en altres llenguatges, *this* fa referència a l'objecte actual. Resulta útil quan necessitem referir-nos a les nostres propietats en les funcions de l'objecte.

```
let factura = {
  descripcio: "factura de prova",
  preu: 100.0,
  iva: 5.0,
  subtotal: function () {
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```

        return this.preu;
    },
    total: function () {
        return this.preu + (this.preu * this.iva) / 100;
    },
};
console.log(factura);
console.log(factura.total());

```

En realitat, el *this* de JavaScript inclou molts més matisos que en altres llenguatges. Ho veurem més endavant.

Constructors

Bé, fins al moment s'havien utilitzat instàncies o objectes únics, però com es fa en JavaScript per a crear diferents instàncies d'un mateix objecte? Fins a ECMAScript 6 no existeix la paraula clau **class**, encara que sí que es pot definir una funció constructor i cridar-la utilitzant la paraula reservada **new** per a crear una nova instància.

El següent exemple mostra una espècie de classe en JavaScript. És una funció el nom de la qual comença en majúscules, la qual cosa indica, als ulls acostumats a altres llenguatges, a reconèixer la nomenclatura d'una classe: és una convenció que deixa clar que no es tracta d'una funció qualsevol, sinó un constructor. Dins d'ella simplement afegim atributs i mètodes.

```

function Web() {
    this.url = "http://localhost";
    this.nom = "Localhost";
    this.mostraInformacio = function () {
        return this.url + ": " + this.nom;
    };
}

```


	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGs DAW/DAM - 1r curs

```
let unaWeb = new Web();
unaWeb.url = "https://www.fcbarcelona.cat/ca/";
unaWeb.nom = "Més que un club";
```

```
console.log(unaWeb);
console.log(unaWeb.mostraInformacio());
```

```
let altraWeb = new Web();
altraWeb.url = "https://www.iesjoanramis.org/";
altraWeb.nom = "IES Joan Ramis i Ramis";
```

```
console.log(altraWeb);
console.log(altraWeb.mostraInformacio());
```

Constructors amb paràmetres

Els constructors també poden tenir paràmetres per a inicialitzar les propietats dels objectes.

```
function Web(url, nom) {
    this.url = url;
    this.nom = nom;
    this.mostraInformacio = function () {
        return this.url + ": " + this.nom;
    };
}
```

```
let unaWeb = new Web("https://www.fcbarcelona.cat/ca/", "Home  
sweet home");
```

```
console.log(unaWeb);
console.log(unaWeb.mostraInformacio());
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
let altraWeb = new Web(
    "https://www.iesjoanramis.org/",
    "IES Joan Ramis i Ramis"
);
console.log(altraWeb);
console.log(altraWeb.mostraInformacio());
```

Afegir atributs i mètodes a un constructor

Ja hem vist que JavaScript és un llenguatge interpretat i feblement tipat, però atenció: no s'està tractant amb classes, sinó amb objectes. Per tant, si es volen afegir atributs o funcions, s'ha d'emprar la seva propietat *prototype*.

Si fem:

```
Web.visites = 2;
console.log(unaWeb.visites);
```

Obtindrem un undefined.

De la mateixa manera, si fem:

```
Web.laMevaFuncio = function () {
    return "Hola";
}
console.log(unaWeb.laMevaFuncio());
```

Obtindrem un bonic error.

És igual si ho afegim al final del codi o just després de crear el constructor.

Per a solucionar això cal fer servir *prototype*:

```
Web.prototype.visites = 2;
console.log(unaWeb.visites);
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
Web.prototype.laMevaFuncio = function () {
    return "Hola";
};

console.log(unaWeb.laMevaFuncio ());
```

8.3 ECMAScript 6

ES6 (ECMAScript 2015) és una versió de JavaScript que introdueix noves característiques i millores al llenguatge. Hi ha actualitzacions posteriors, però aquesta versió va introduir canvis significatius. Vegem alguns conceptes clau d'ES6:

- **Declaració de variables**

En ES6, es van introduir noves maneres de declarar variables utilitzant les paraules clau **let** i **const**. **let** s'utilitza per a declarar variables amb abast de bloc, mentre que **const** s'utilitza per a declarar variables constants.

- **Desestructuració: extracció de valors**

La desestructuració és una característica d'ES6 que permet extreure valors d'arrays o objectes i assignar-los a variables individuals.

```
// Desestructuració d'un array
const numeros = [1, 2, 3];
const [a, b, c] = numeros;

console.log(a, b, c); // 1, 2, 3

// Desestructuració d'un objecte
const persona = {
    nom: "Andreu",
    edat: 25,
};

const { nom, edat } = persona;
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
console.log(nom, edat); // Andreu, 25
```

- **Drecera per a l'assignació de propietats d'objectes**

En ES6, es va introduir una sintaxi més concisa per a assignar propietats a objectes quan les variables tenen el mateix nom que les propietats.

```
const nom = "Josep";
const edat = 25;
```

```
const persona = {
    nom,
    edat,
};
```

```
console.log(persona.nom); // Josep
console.log(persona.edat); // 25
```

Veiem com les propietats *nom* i *edat* s'assignen automàticament a partir de les variables amb els mateixos noms.

- **Cadenes com a plantilles**

Les plantilles de cadenes (template strings) són una característica d'ES6 que permet incloure expressions dins de cadenes utilitzant l'operador d'interpolació `${}`.

```
const nom = "Bob";
const edat = 32;
```

```
const missatge = `Hola, el meu nom és ${nom} i tenc
${edat} anys.`;

console.log(missatge); // Hola, el meu nom és Bob i tenc
32 anys.
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

Les variables *nom* i *edat* s'insereixen dins de la cadena utilitzant l'operador d'interpolació.

- **Operador spread**

L'operador de propagació (spread operator) és una característica d'ES6 que permet expandir un array en múltiples elements.

```
const nombres = [1, 2, 3];
const nousNombres = [...nombres, 4, 5];

console.log(nousNombres); // [1, 2, 3, 4, 5]
```

Veiem com l'operador spread ... s'utilitza per a expandir l'array *nombres* amb cadascun dels elements de *nousNombres*.

- **Paràmetres per defecte en funcions**

En ES6, es va introduir la possibilitat de definir valors per defecte per als paràmetres d'una funció.

```
function saludar(nom = "Convidat") {
    console.log(`Hola, ${nom}!`);
}

saludar(); // Hola, Convidat!
saludar("Anna"); // Hola, Anna!
```

Si no es passa un valor per al paràmetre *nom*, s'utilitzarà el valor per defecte "Convidat".

- **Paràmetres rest**

El paràmetre rest (rest parameter) és una característica d'ES6 que permet capturar un nombre variable d'arguments en una funció com un array (és a dir, passar diversos paràmetres en un de sol).

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
function sumar(...numeros) {
    let resultat = 0;
    for (let numero of numeros) {
        resultat += numero;
    }
    return resultat;
}
```

```
console.log(sumar(1, 2, 3)); // 6
console.log(sumar(4, 5, 6, 7)); // 22
```

Veiem en l'exemple com el paràmetre rest *...numeros* captura tots els arguments proporcionats a la funció com un array anomenat *numeros*.

- **Funció fletxa (arrow function) o funció anònima**

Les arrow functions o funcions anònimes són una sintaxi més concisa per a definir funcions en ES6. S'utilitzen fletxes (**=>**) en lloc de la paraula clau *function*. La fletxa separa els paràmetres del cos de la funció. A més es sobreentén la paraula *return*.

```
// Funció tradicional
function sumar(a, b) {
    return a + b;
}

// Arrow function equivalent
const sumar = (a, b) => a + b;

console.log(sumar(2, 3)); // 5
```

- **Mètodes per a arrays**

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

ES6 va introduir nous mètodes per a manipular i treballar amb arrays de forma més senzilla.

- **forEach()**: executa una funció proporcionada una vegada per cada element de l'array.

```
const numeros = [1, 2, 3, 4, 5];
numeros.forEach((numero) => {
    console.log(numero);
});
// Output:
// 1
// 2
// 3
// 4
// 5
```

- **map()**: crea un nou array amb els resultats d'aplicar una funció a cada element de l'array original.

```
const numeros = [1, 2, 3, 4, 5];
const doblaNumeros = numeros.map((numero) => {
    return numero * 2;
});
console.log(doblaNumeros); // Output: [2, 4, 6, 8, 10]
```

- **filter()**: crea un nou array amb tots els elements que compleixin una condició determinada.

```
const numeros = [1, 2, 3, 4, 5];
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```
const numerosParells = numeros.filter((numero) => {
    return numero % 2 === 0;
});
```

```
console.log(numerosParells); // Output: [2, 4]
```

- **reduce()**: aplica una funció a un acumulador i a cada element de l'array (d'esquerra a dreta) per a reduir-lo a un únic valor.

```
const numeros = [1, 2, 3, 4, 5];
const suma = numeros.reduce((acumula, numero) => {
    return acumula + numero;
}, 0);
```

```
console.log(suma); // Output: 15
```

- **find()**: retorna el primer element de l'array que compleixi una condició determinada.

```
const numeros = [1, 2, 3, 4, 5];
const numeroTrobat = numeros.find((numero) => {
    return numero > 3;
});
```

```
console.log(numeroTrobat); // Output: 4
```

- **findIndex()**: retorna l'índex del primer element de l'array que compleix amb una funció de prova proporcionada, o -1 si no es troba.

```
const numeros = [1, 2, 3, 4, 5];
const indexTrobat = numeros.findIndex((numero) => {
    return numero > 3;
});
```


	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
console.log(indexTrobat); // Output: 3
```

- **some()**: comprova si almenys un element de l'array compleix una condició determinada.

```
const numeros = [1, 2, 3, 4, 5];
const teNumeroParell = numeros.some((numero) => {
    return numero % 2 === 0;
});
console.log(teNumeroParell); // Output: true
```

- **every()**: comprova si tots els elements de l'array compleixen una condició determinada.

```
const numeros = [1, 2, 3, 4, 5];
const totsSonNumerosParells = numeros.every((numero) => {
    return numero % 2 === 0;
});
console.log(totsSonNumerosParells); // Output: false
```

- **Iteradors**

Amb JavaScript hi ha diverses maneres d'iterar col·leccions d'elements. Ja hem vist la estàndard:

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}
```

Però n'hi ha d'altres:

```
var lletres = ['a', 'b', 'c', 'd', 'e'];
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
for (let i in lletres) {
    console.log(`Índex ${i}: ` + lletres[i]);
}

for (let lletra of lletres) {
    console.log(lletra);
}
```

Si tenim un array sense valors definits, emprant el bucle `for...in` ens saltem aquestes posicions, mentre que amb `for...of` no.

També s'empra aquesta fórmula:

```
var lletres = ['a', 'b', 'c', 'd', 'e'];
lletres.forEach((value) => console.log(value));
[1,2,3].forEach((value) => console.log(value));
```

- **Set**

Els Sets (o conjunts) són una estructura de dades (objectes en realitat) que permeten, de manera semblant als arrays, emmagatzemar dades.

A diferència dels arrays, no admeten valors duplicats.

```
const llista = new Set ();

llista.add(8);
llista.add(6);
llista.add(5);
llista.add(6);

console.log(llista); //Output Set(3) [8, 6, 5]
```

Donat que el mètode `add` retorna una referència al conjunt, es pot fer també el següent:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
llista.add(6).add(3).add(5);
```

També podem inicialitzar la llista a partir d'un array:

```
const llista = new Set([1, 5, 3, 9, 5, 6]);
```

O a partir d'una cadena de caràcters:

```
const llista = new Set('Conjunt'); //Output: Set(6) [ "C",  
"o", "n", "j", "u", "t" ]
```

Existeixen diferents mètodes per a treballar amb Sets:

```
console.log(llibra.size);  
llibra.delete('t');  
llibra.clear(); //Output Set {}  
console.log(llibra.has('t')); //Output false
```

- **Map**

Map és una estructura que permet emmagatzemar estructures de tipus **clau-valor**, on les claus no es poden repetir i tenen associat un valor. Tant les claus com els valors poden ser de qualsevol tipus.

```
const provinces = new Map();  
provinces.set(1, 'Sevilla');  
provinces.set(5, 'Pontevedra');  
provinces.set(7, 'Madrid');  
  
console.log(provinces); //Output Map(3) { 1 → "Sevilla", 5  
→ "Madrid", 7 → "Pontevedra" }
```

Si tornem a afegir un element amb la mateixa clau, el nou valor substituirà l'anterior.

```
provinces.set(1, 'Granada');  
console.log(provinces); //Output Map(3) { 1 → "Granada",
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```
5 → "Madrid", 7 → "Pontevendra" }
```

També podem emprar un array, on cada element sigui un altre array amb la parella clau-valor.

```
const persona = new Map([['nom', 'Jose'], ['l·linatges', 'Pons García'], ['edat', 25]]);
```

```
console.log(persona);
```

Es poden fer operacions sobre les estructures map, com obtenir el valor d'una clau amb el mètode get(), cercar una clau en un mapa, esborrar valors...

```
console.log(provincies.get(1));
```

```
console.log(provincies.has(1)); //Output true
```

```
provincies.delete(1);
```

```
console.log(provincies.has(1));
```

Es poden obtenir objectes iterables amb .keys() i values(), separant claus i valors.

```
let claus = provincies.keys();
```

```
for (let clau of claus) {
```

```
    console.log(clau);
```

```
}
```

```
let valors = provincies.values();
```

```
for (let valor of valors) {
```

```
    console.log(valor);
```

```
}
```

Anem a veure un parell més d'exemples de com iterar mapes:

```
provincies.set(22, 'Balears').set(12, 'Segovia');
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```

for (let p of provincias) {
    console.log(p);
}

//Output:
Array [ 5, "Pontevedra" ]
Array [ 7, "Madrid" ]
Array [ 22, "Balears" ]
Array [ 12, "Segovia" ]

for (let [clau, valor] of provincias) {
    console.log(clau, valor);
}


//Output:
5 Pontevedra
7 Madrid
22 Balears
12 Segovia

```

8.4 El BOM i el DOM

El BOM (Browser Object Model) i el DOM (Document Object Model) són dos conceptes importants en JavaScript que permeten interactuar amb el navegador i manipular l'estructura i contingut d'una pàgina web.

El **BOM** és el model d'objectes del navegador. Es tracta d'una interfície que permet a JavaScript interactuar amb el **navegador** mateix i amb parts que no formen part del contingut HTML de la pàgina, com:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

- La finestra del navegador (**window**): L'objecte global de JavaScript que representa la finestra del navegador.
- Historial (**history**): Permet navegar cap enrere o endavant entre pàgines (`history.back()`, `history.forward()`).
- Ubicació (**location**): Conté informació sobre l'URL actual i permet canviar-la (`location.href`).
- Navegador (**navigator**): Proporciona informació sobre el navegador, com ara el userAgent o el mode en línia/desconnectat.
- Pantalla (**screen**): Conté informació sobre la pantalla del visitant.
- Temps: Funcions com `setTimeout()` i `setInterval()`.

El BOM és menys estandarditzat que el DOM i pot variar lleugerament entre navegadors.

El **DOM** és una representació estructurada de la pàgina web (HTML o XML). Es tracta d'un model d'objectes que permet accedir, modificar i manipular el contingut i l'estructura de la **pàgina web**.

El DOM permet:

- **Accedir a elements de la pàgina:** Per exemple, amb `document.getElementById('id')`, `document.querySelector('.classe')`.
- **Modificar contingut:** Com canviar el text d'un element (`element.textContent = "Nou text"`).
- **Afegir o eliminar elements:** Crear nous elements (`document.createElement('div')`) i inserir-los al DOM (`parentNode.appendChild(nouElement)`).
- **Controlar esdeveniments:** Afegir esdeveniments com clics (`element.addEventListener('click', funció)`).

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

El DOM és una interfície estandarditzada pel W3C, per la qual cosa funciona igual en la majoria de navegadors moderns.

8.4.1 L'objecte document

Quan un document HTML es carrega en un navegador web, obtenim un objecte **document**. L'objecte document és el **node arrel** d'un document HTML, i d'ell pengen tots els altres nodes: **nodes d'elements, de text, d'atributs, de comentaris**.

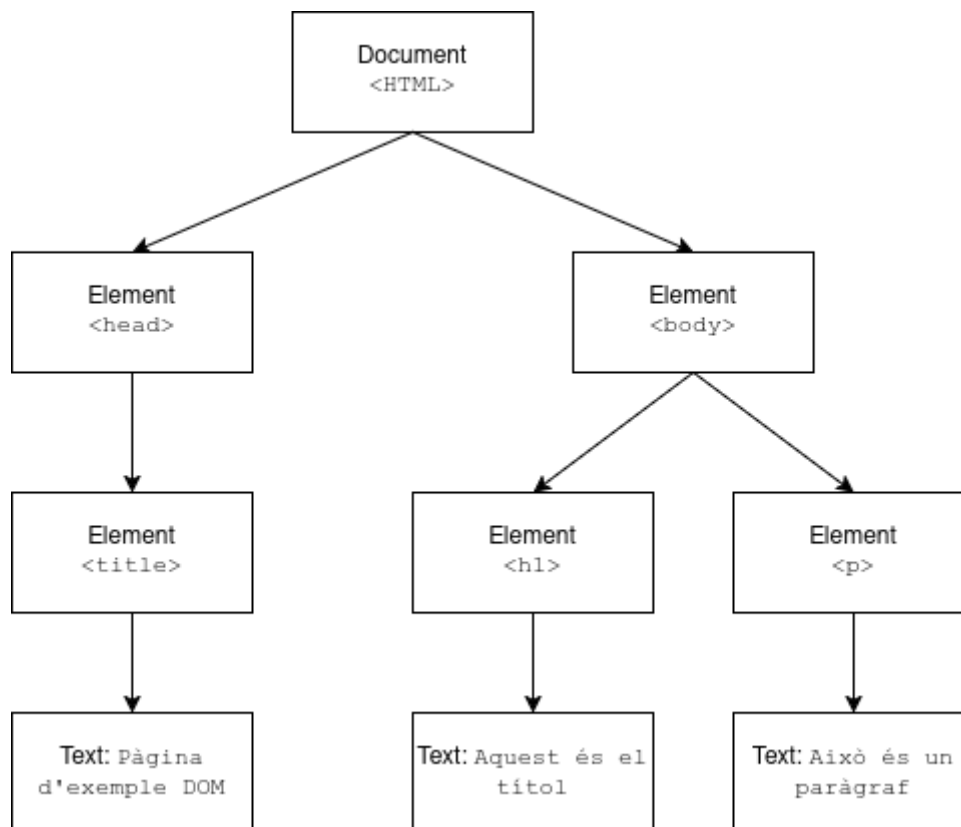
Per exemple, si tenim aquest document HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Pàgina d'exemple DOM</title>
</head>
<body>
  <h1>Aquest és el títol</h1>

  <p>Això és un paràgraf</p>
</body>
</html>
```

Aquesta seria la representació de l'estructura del DOM:


	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs



Recordeu que el *document* és part de l'objecte *window* i, per tant, s'hi pot accedir amb *window.document*, tot i que també s'hi pot accedir directament amb *document*.

Les propietats de *document* són:

- **location**: conté la URL del document.
- **title**: títol del document HTML.
- **lastModified**: data de l'última modificació.
- **cookie**: cadena de caràcters que conté la cookie associada al recurs representat per l'objecte.
- Les propietats **anchors**, **forms**, **images**, **links** retornen un objecte amb el qual podem accedir a cadascun dels elements (àncores, formularis, imatges, enllaços) presents en el document.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

Els mètodes més importants de l'objecte document són:

- **write()**: escriu codi HTML en el document.
- **writeln()**: idèntic a write(), però inserta un retorn de carro al final.
- **open**(tipus MIME): obre un buffer per recollir el que retorna els mètodes write() i writeln(). Els tipus MIME que es poden utilitzar són: text/html, text/plain, text/jpeg, etc.
- **close()**: tanca el buffer.
- **clear()**: esborra el contingut del document.

8.4.2 Tipus de nodes

Encara que existeixen més tipus de nodes, en realitat, per a manipular les pàgines web només necessitem els cinc següents:

- **document**: node arrel del qual deriven els altres.
- **element**: cadascuna de les etiquetes HTML.
- **attr**: cadascun dels atributs de les etiquetes HTML.
- **text**: text que es troba dins les etiquetes HTML.
- **comment**: comentaris de la pàgina HTML.

8.4.3 Selecció d'elements del DOM

Per a accedir a la lectura dels elements de la pàgina, els mètodes que es descriuen a continuació van precedits sempre de *document*.

getElementById ("id")	Retorna l' element amb l'id donat. Si no existeix el mètode retorna <i>null</i> .
getElementsByClassName ("classe")	Retorna una col·lecció HTML d'elements amb una classe especificada.
getElementsByTagName ("nom")	Retorna un NodeList d'elements amb el

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

	nom especificat a l'atribut <i>name</i> .
<code>getElementsByTagName("etiqueta")</code>	Retorna una col·lecció HTML amb tots els elements amb l'etiqueta especificada.
<code>querySelector("selector")</code>	Retorna el primer element que coincideix amb un selector CSS.
<code>querySelectorAll("selector")</code>	Retorna tots els elements que coincideixen amb un selector CSS donat en una llista de nodes (NodeList).

Una **col·lecció HTML** és una col·lecció d'elements HTML que representen una vista en viu dels nodes dins del DOM. Això significa que si el DOM canvia després de crear la HTMLCollection, la col·lecció es manté actualitzada automàticament.

Algunes característiques:

- Només conté elements HTML (no altres tipus de nodes com textos o comentaris).
- És una vista en viu: reflecteix automàticament els canvis en el DOM.
- Es pot accedir als elements per índex, com si fos un array.
- No suporta mètodes d'array com *forEach* directament, però es pot iterar amb un bucle *for* clàssic. També podem convertir aquest tipus d'objecte amb un array amb l'operador de progradació (*spread*).

Un **NodeList** és una llista d'objectes **Node**, que pot contenir qualsevol tipus de node (elements, text, comentaris, etc.). No sempre és una vista en viu.

Algunes característiques:

- Pot contenir nodes no HTML, com nodes de text.
- És estàtica quan prové de *querySelectorAll*, però pot ser en viu en altres casos, com amb *childNodes*.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

- Suporta alguns mètodes d'array com *forEach*.
- Es pot accedir als elements per índex, com si fos un array.

8.4.4 Manipulació d'elements

Crear un nou element HTML

Per a crear una nova etiqueta crearem els nodes que la contenen. Per a això seguirem els següents passos:


Pas	Codi i explicació	
1 ^r pas	Codi	<code>var text = "Nou text.";</code>
	Explicació	Crear variable amb el nou text
2 ⁿ pas	Codi	<code>var nouElement = document.createElement("nom_tag");</code>
	Explicació	Crear l'element (node) de l'etiqueta HTML donada.
3 ^r pas	Codi	<code>var textNode = document.createTextNode(text);</code>
	Explicació	Crear un node de tipus text i assignar-li el contingut de la variable "text" creada en el 1 ^r pas. També es pot assignar text directament entre cometes.
4 ^t pas	Codi	<code>nouElement.appendChild(textNode);</code>
	Explicació	Inserir el node de tipus text com a fill del node <i>etiqueta</i> .

Hem creat un node (etiqueta) que conté un node de text, és a dir una nova etiqueta amb text, però aquesta no està integrada en la pàgina, sinó que la tenim definida en una variable. Ens queda encara integrar l'etiqueta en la pàgina.

Inserir o reemplaçar un node en la pàgina

Després de crear el node l'hem d'inserir en la pàgina o reemplaçar-lo per un altre ja existent. Una vegada creat el node, seguint els quatre passos anteriors, tenim diverses opcions:

- Inserir-lo després d'un element (*appendChild*).

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs


Pas	Codi i explicació	
5 ^è pas	Codi	<code>document.getElementById("caixa1").appendChild(nouElement);</code>
	Explicació	Especifiquem el lloc de la pàgina on volem introduir l'element - <code>document.getElementById("...")</code> – i inserim l'element com a fill del node especificat. El nou element es col·locarà darrere dels ja existents dins d'aquest node.

- Inserir-lo abans d'un element (*insertBefore*).

Pas	Codi i explicació	
5 ^è pas	Codi	<code>var referencia = document.getElementById("ref");</code>
	Explicació	En una variable guardem el node de referència per a inserir-lo abans d'aquest.
6 ^è pas	Codi	<code>var pare = referencia.parentNode;</code>
	Explicació	En una variable guardem el node pare del node de referència des del mètode parentNode .
7 ^è pas	Codi	<code>pare.insertBefore(nuevoElemento, referencia)</code>
	Explicació	Des de l'element pare inserim el nou element mitjançant el mètode <i>insertBefore()</i> ; on el primer paràmetre serà el nou element a inserir i el segon, l'element de referència).

- Reemplaçar un element (*replaceChild*).

Pas	Codi i explicació	
5 ^è pas	Codi	<code>var elementAntic = document.getElementById("antic");</code>
	Explicació	En una variable guardem el node que volem reemplaçar.
6 ^è pas	Codi	<code>var pare = elementAntic.parentNode;</code>
	Explicació	Mitjançant una variable accedim a l'element pare del node que volem reemplaçar
7 ^è pas	Codi	<code>pare.replaceChild(nuevoElemento, elementAntic);</code>
	Explicació	Des de l'element pare reemplaçem el nou element mitjançant el mètode <i>replaceChild</i> (nou, vell); on el primer paràmetre serà el nou element a inserir, i el segon l'element a eliminar).

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

Eliminar un node

Per a eliminar un node ja existent, utilitzarem el mètode *removeChild()* amb un únic paràmetre: el node a eliminar.

Pas	Codi i explicació	
1 ^r pas	Codi	<code>var eliminar = document.getElementById("paragraf");</code>
	Explicació	En una variable guardem l'element que volem eliminar.
2 ⁿ pas	Codi	<code>eliminar.parentNode.removeChild(eliminar);</code>
	Explicació	Cridem al seu element pare - <i>parentNode</i> - i des d'aquí l'eliminem mitjançant el mètode <i>removeChild</i> .

Canviar un node de lloc

Per a canviar un node de lloc seguirem els mateixos passos que per a eliminar-lo. En els passos anteriors hem guardat el node en una variable. Això permet tornar a inserir-lo en qualsevol punt de la pàgina mitjançant el mètode *appendChild()*.

Pas	Codi i explicació	
1 ^r pas	Codi	<code>var moure = document.getElementById("paragraf");</code>
	Explicació	En una variable guardem l'element que volem moure.
2 ⁿ pas	Codi	<code>moure.parentNode.removeChild(moure);</code>
	Explicació	Eliminar l'element que es vol moure de la posició original.
3 ^r pas	Codi	<code>document.getElementById("contenedorFinal").appendChild(moure)</code>
	Explicació	Col·locar l'element guardat en un altre node.

Copiar un node en un altre lloc

Per a copiar un node en un altre lloc de la pàgina conservant l'original utilitzarem el mètode *cloneNode(true)*.

Pas	Codi i explicació	
1 ^r pas	Codi	<code>var element = document.getElementById("copia");</code>
	Explicació	Guardar en una variable l'element que es vol copiar.
2 ⁿ pas	Codi	<code>var copiar = element.cloneNode(true);</code>

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

	Explicació	Per a copiar-ho hem d'aplicar-li el mètode <code>cloneNode(true)</code> ; La còpia la guardem en una variable.
3r pas	Codi	<code>document.getElementById("pare").appendChild(copiar);</code>
	Explicació	Procedim com quan inserim un nou element, col·locant l'element des de la seva etiqueta pare (també podem emprar el mètode <code>insertBefore</code> per a inserir-lo abans d'un altre element).

8.4.5 Manipulació del contingut dels elements

Propietat `textContent`

La propietat `textContent` d'un element permet obtenir i modificar el text que conté aquell element i de tots els seus descendents.

Com funciona?

- No interpreta ni analitza HTML. Només treballa amb text.
- Si s'estableix un valor amb `textContent`, tots els nodes fills existents es reemplaçaran amb el text proporcionat.
- Si l'element no té contingut, retorna una cadena buida ("").

Propietat `innerHTML`

La propietat `innerHTML` és emblant a l'anterior, però aquesta sí llegeix i manipula etiquetes HTML.

Com funciona?

- Analitza i interpreta el contingut com a codi HTML.
- Permet inserir HTML dinàmicament dins d'un element.
- Si s'estableix amb codi HTML mal format, pot provocar errors inesperats o comportaments no desitjats.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

La lectura es faria d'aquesta manera:

Pas	Codi i explicació	
1r pas	Codi	<code>var lectura = document.getElementById("llegeix").innerHTML;</code>
	Explicació	Guardar en una variable l'element que es vol copiar.

Mentre que l'escriptura seguiria aquest format:

Pas	Codi i explicació	
1r pas	Codi	<code>document.getElementById("escriu").innerHTML = "Anar a Google";</code>
	Explicació	Cridar a l'element de destí: <code>document.getElementById("escriu")</code> , aplicar la propietat <code>.innerHTML</code> , signe igual (=), escriure el codi nou entre cometes. També podem escriure una variable on hàgim guardat prèviament el codi.

Aquesta operació esborra el contingut antic que tenia l'element i el reemplaça pel nou. Si el nou contingut és un element buit "" s'esborrarà el contingut de l'element.

Podem afegir més contingut a un element sense esborrar el que ja hi ha de la manera següent:

Pas	Codi i explicació	
1r pas	Codi	<code>var contingut = document.getElementById("element");</code>
	Explicació	Guardam l'element en una variable.
2n pas	Codi	<code>contingut.innerHTML += "<p>Afegir un altre paràgraf</p>";</code>
	Explicació	Apliquem la propietat <code>innerHTML</code> i la suma i assignació += del nou valor. El nou valor assignat s'afegirà al ja existent.

8.4.6 Manipulació d'atributs

Per a accedir o modificar els atributs d'un element ho podem fer de dues maneres diferents:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

- Accedint a través la **propietat**. L'atribut es converteix en una propietat del seu element, per la qual cosa simplement podem cridar a l'element i a continuació afegir un punt i el nom de l'atribut.

Lectura:

Pas	Codi i explicació	
1r pas	Codi	<code>var valor = document.getElementById("element").href;</code>
	Explicació	Llegir el valor de l'atribut: cridem a l'element i li posem darrere la referència de l'atribut precedida d'un punt.

Esriptura:

Pas	Codi i explicació	
1r pas	Codi	<code>document.getElementById("element").href = "https://www.google.com"</code>
	Explicació	Cridar al node que conté l'atribut, un punt i el nom de l'atribut. Amb el signe igual = li assignem un nou valor, bé sigui directament entre cometes o mitjançant una variable.


- Amb els mètodes **getAttribute** i **setAttribute**

Lectura:

Pas	Codi i explicació	
1r pas	Codi	<code>var valor = document.getElementById("element").getAttribute("class");</code>
	Explicació	Cridem a l'element, i posem darrere el mètode <i>getAttribute()</i> , el paràmetre d'aquest mètode serà el nom de l'atribut, bé entre cometes o guardat en una variable.

Esriptura:

Pas	Codi i explicació	
1r pas	Codi	<code>document.getElementById("element").setAttribute("class", "petit");</code>
	Explicació	Cridem a l'element, i posarem darrere el mètode <i>setAttribute()</i> , on

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

		el primer paràmetre serà el nom de l'atribut i el segon el seu valor. També podem escriure variables que continguin aquests valors.
--	--	---

Eliminar un atribut

Per a eliminar un atribut existeix el mètode ***removeAttribute***, el qual té un paràmetre al que li passem el nom de l'atribut.

Afegir o treure un atribut de manera ràpida

El mètode ***toggleAttribute*** permet afegir a l'element l'atribut que s'indiqui com a paràmetre si no s'ha afegit ja, o treure'l en cas que hi sigui.

Saber si un element té atributs

El mètode ***hasAttribute*** retorna vertader si l'element té l'atribut que se li passa i fals si no el té.

Obtenir tots els atributs d'un element

La propietat ***attributes*** d'un element permet obtenir tots els atributs d'un element. Es retorna un objecte de tipus **NamedNodeList** que no és un array, però sí que podem accedir per l'índex i podem emprar la propietat **length**.


Cada element de la llista que retorna aquesta propietat té una propietat **name** que conté el nom de l'atribut i una propietat **value** que conté el seu valor.

8.4.7 Modificar el CSS

Hi ha diverses maneres de modificar l'estil de manera dinàmica fent servir JavaScript.

Propietat style

Els elements tenen una propietat anomenada ***style*** que permet accedir a les propietats CSS d'un element. El que realment es fa és modificar l'atribut ***style*** de

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

l'element, modificant els valors d'aquesta. La idea és que *style* és un objecte que té com a propietats totes les propietats CSS en format **CamelCase**.

```
let paragraf = document.getElementsByTagName("p")[0];
paragraf.style.backgroundColor = "#CCC";
```

No obstant això, la majoria de navegadors actuals accepta també el format idèntic a CSS a través de claudàtors:

```
paragraf.style["text-align"] = "center";
```

Hem de tenir en compte alguns problemes a l'hora de fer servir *style* per a modificar el CSS:


- Manipular la propietat dels elements *style* dona prioritat a qualsevol altre CSS (ja que es modifica la propietat *style* d'HTML), però no sempre és el desitjable.
- És més fàcil de mantenir el codi si s'empren classes per a aplicar CSS. Especialment quan volem modificar diverses propietats CSS alhora.
- No podem consultar a través d'aquesta propietat les propietats CSS que té un element al qual se li ha donat format a través de fulls d'estil externs.

Per a pal·liar l'últim problema disposem d'un mètode en l'objecte *window* anomenat **getComputedStyle**. Aquest mètode retorna un objecte (idèntic al de la propietat *style*) en el qual es poden consultar les propietats CSS que s'estan aplicant a un element concret. Aquest mètode només permet veure les propietats de l'element, però no modificar-les.

```
let cssParagraf = window.getComputedStyle(paragraf);
console.log(cssParagraf.fontFamily);
```

Manipular les classes CSS

Els elements disposen d'una propietat anomenada **className**. Mitjançant aquesta

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

propietat podem assignar una classe CSS a un element de la pàgina. Per exemple:

```
paragraf.className = "remarcar";
```

Els problemes sorgeixen quan tenim més d'una classe assignada a un element, atès que, si volem afegir o eliminar una classe s'ha de tenir en compte que hem de treballar amb cadenes de caràcters amb espais i aquesta manera de treballar és incòmoda.

```
console.log(paragraf.className);  
//Output: remarcar anotacio
```


Per això existeix la propietat **classList**. Es tracta d'un objecte que representa les classes aplicades a un element. Novament, tenim un objecte que sembla un array i es pot accedir a cada element amb el seu índex o emprar la propietat length per a saber el nombre de classes assignades a un objecte.

Així, per a obtenir el nom de les classes d'un element mitjançant classList es faria, per exemple, d'aquesta manera:

```
for (let classe of paragraf.classList) {  
    console.log(classe);  
}  
//Output:  
remarcar  
anotacio
```

La propietat **classList** disposa d'aquestes interessants propietats:

- **add(nomClasse1 [, nomClasse2]):** Permet afegir una classe a l'element. N'hi ha prou amb indicar el nom de la classe entre cometes. Admet indicar diverses classes si se separen amb comes.
- **remove(nomClasse1 [, nomClasse2]):** Mètode contrari a l'anterior. Retira de

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

l'element la classe o classes que s'indiquin.

- **toggle**(*nomClasse1* [, *forçar*]): Si s'empra un sol paràmetre, serà el nom d'una classe. El mètode fa que si l'element no té assignada aquesta classe, s'assigna. Si l'element ja tenia aquesta classe, la treu.

El segon paràmetre (**forçar**) és un valor booleà que si val **true** llavors afegeix la classe, independentment de si l'element ja la tenia o no. Si val **false**, llavors treu la classe de l'element. És a dir, aquest mètode és capaç de fer el que feien els altres dos.

- **contains**(*nomClasse*): Retorna **true** si l'element té assignada la classe, el nom de la qual s'indica en el paràmetre.
- **replace**(*nomAntic*, *nomNou*): Permet canviar una classe per una altra en l'element.

8.4.8 Navegar pel DOM

A continuació veurem alguns mètodes que permeten obtenir informació de l'estructura del DOM del document.


Obtenir els fills d'un element

Partim d'aquest codi HTML:

```
<p id="paragraf">Aquest és el tema de <strong>JavaScript</strong>,
el darrer d'aquest curs</p>
```

Tots els elements tenen una propietat **childNodes**, que retorna una llista de nodes. Podem obtenir els fills de la següent manera:

```
let paragraf = document.getElementById("paragraf");
for (let fill of paragraf.childNodes){
  console.log(`Text: ${fill.nodeValue}\n`
    `Tipus de node: ${fill.nodeType}, ${fill.nodeName}`);
}
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

Hi ha una altra propietat molt semblant que es diu **children**, però aquesta només obté els fills que siguin elements, no els nodes de tipus text.

Altres propietats

A més del mètode que obté tots els fills, tenim les següents propietats que permeten referenciar nodes relacionats amb l'element actual:

Propietat	Ús
firstChild	Selecciona el primer node fill de l'element actual.
lastChild	Selecciona el darrer node fill de l'element actual.
nextSibling	Selecciona el següent germà respecte del node actual.
previousSibling	Selecciona el germà anterior respecte del node actual.
parentNode	Obté el pare del node actual.
firstElementChild	Obté el primer fill del node actual que sigui un element.
lastElementChild	Obté el darrer fill del node actual que sigui un element.
nextElementSibling	Obté el següent germà que sigui un element.
previousElementSibling	Obté l'anteriorgermà que sigui un element.
childElementCount	Nombre d'elements fills del node actual.

Totes les propietats tenen dues versions: una sense la paraula **Element** i una altra amb ella. Solen ser més interessants les que tenen la paraula Element, ja que només realitzen recorreguts a través de nodes que siguin elements de la pàgina.

8.4.9 Obtenir atributs data

En els documents HTML es poden crear atributs **data**. Es tracta d'un tipus d'atributs creats pels desenvolupadors. S'empren per a emmagatzemar informació que pugui ser manipulada des de JavaScript.

Els atributs data comencen per data seguit d'un guió i després el nom que es desitgi. Per exemple:

```
<p id="p1" data-tipus="Començament del llibre" data-llibre="Don Quijote" data-autor-principal="Miguel de Cervantes">En un lugar de la Mancha...</p>
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

La propietat **dataset** és l'encarregada de manipular aquests atributs. Hem de tenir en compte que accedim als noms amb CamelCase d'aquesta manera tant per llegir com per escriure:

```
let p1 = document.getElementById("p1");
console.log(p1.dataset.llibre);
console.log(p1.dataset.autorPrincipal);
console.log(p1.dataset.tipus);

p1.dataset.publicacio = "Segle XVII"
```

8.5 Gestió d'esdeveniments


Els esdeveniments (**events**) són el mecanisme principal de comunicació entre l'aplicació i l'usuari. En aquesta interacció, l'aplicació variarà en funció de les accions que faci l'usuari. Així doncs, un esdeveniment és el resultat d'un acte de l'usuari o per altres raons.

L'aplicació ha de ser capaç de detectar o capturar un esdeveniment, així com executar codi associat a aquest. L'encarregat de gestionar els esdeveniments és el DOM. Cada esdeveniment s'associa a un element del DOM.

La clau dels esdeveniments en JavaScript és la seva capacitat **asíncrona**. Podem estar esperant que l'usuari faci clic en un botó, però no es pot pausar la resta de codi a causa d'aquesta espera. L'espera es dona en segon pla; és un procés especial que es diu **listener** (oient) que està esperant a rebre una notificació del clic (o un altre tipus d'esdeveniment) de l'usuari.

8.5.1 Captura d'esdeveniments

Fa anys, en les primeres versions de JavaScript, la captura d'un esdeveniment es feia des del codi HTML. Encara funciona aquest mètode però no és recomanable fer-lo servir perquè complica el manteniment del codi.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

Aquest mètode empra atributs en les etiquetes dels elements de la pàgina que comencen amb la paraula **on** seguida del nom de l'esdeveniment. Per exemple, **onclick**. Com a valor de l'atribut s'indica el nom de la funció (que també pot ser anònima) que conté el codi a executar.

```
<p id="paragraf" onclick="alert('Has fet clic sobre el paràgraf')">
Sóc clicable</p>
<p>Jo no</p>
```

El primer paràgraf empra l'atribut onclick que permet llançar codi JavaScript quan l'usuari faci clic sobre el paràgraf. El segon paràgraf no reacciona a fer-hi clic sobre ell, ja que no s'ha indicat cap esdeveniment.

Això es podria emprar directament en codi JavaScript independent d'HTML llançant l'esdeveniment des d'una variable de la següent manera, però tampoc és la manera recomanable de fer-ho.

```
let paragraf = document.getElementById("paragraf");
paragraf.onclick={() => {alert('Has fet clic sobre el paràgraf')}};
```

S'ha assignat com a funció callback una funció fletxa que mostra el mateix missatge que abans.

addEventListener

El mètode que es recomana fer servir és crear un esdeveniment amb el mètode **addEventListener** que està disponible per a tots els elements. A aquest mètode se li ha d'indicar com a primer paràmetre el tipus d'esdeveniment i com a segon paràmetre la funció callback a executar.

```
let paragraf = document.getElementById("paragraf");
paragraf.addEventListener("click", () => {alert('Has fet clic sobre
el paràgraf')}});
```

Propagació d'esdeveniments

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

Una qüestió fonamental en la captura d'esdeveniments és com es propaguen els esdeveniments sobre els contenidors dels elements. És a dir, suposem que tenim un contenidor de tipus **div**, en ell un **paràgraf** i dins del paràgraf un **botó**. Fem clic sobre el botó. El paràgraf podria capturar el clic? I el contenidor?

Anem a veure aquest exemple:

```
<div>
  <p>
    <button>Fes clic aquí</button>
  </p>
</div>
```

Si tenim aquest codi CSS:

```
div{
  position: fixed;
  left: 0;
  top: 0;
  height: 50%;
  width: 100%;
  background-color: gray;
}

p {
  height: 50%;
  margin: 0;
  background-color: lightgray;
}
```

I aquest codi javascript:

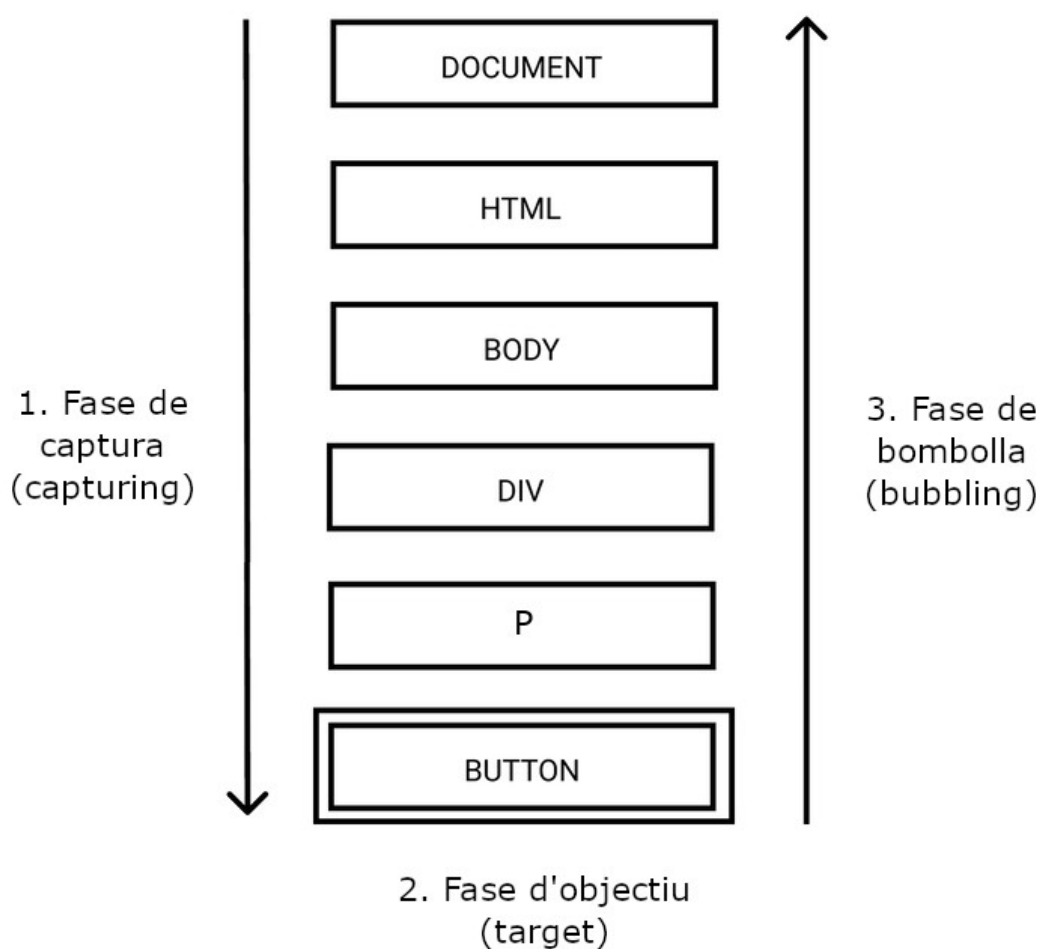
```
let div = document.querySelector("div");
let p = document.querySelector("p");
let button = document.querySelector("button");
div.addEventListener("click", ()=> console.log("click div"));
p.addEventListener("click", ()=> console.log("click p"));
button.addEventListener("click", ()=> console.log("click button"));
```

Si executem el codi i fem clic en el botó, per consola veurem aquest text:


	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGs DAW/DAM - 1r curs

```
click button
click p
click div
```

L'esdeveniment es propaga des del botó (element més interior) fins a l'element de tipus div (element més exterior, el més pròxim a l'arrel del DOM).



Per defecte, la funció associada a l'esdeveniment es llança en la base de bombolla, per això en el codi de l'exemple hem pogut veure primer el missatge del botó, a continuació el del paràgraf i després el del contenidor div.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

Podem modificar aquest comportament, si indicam que es llanci en la fase de captura. Això es pot fer afegint un tercer paràmetre a **addEventListener**. Es tracta d'un valor booleà i si es posa a **true**, el codi de captura es llança en la fase de captura, si es posa a **false** (valor per defecte) es llança en la fase de bombolla.

Anul·lar esdeveniments

Imaginem que volem que, en fer un clic a un element concret, se'ns mostri un missatge, però volem que aquest missatge aparegui una sola vegada. Per defecte, l'esdeveniment es captura indefinidament, però existeix un mètode contrari a **addEventListener** que té els mateixos paràmetres. Es tracta de **removeEventListener**.

```
<p>Fes clic aquí</p>
<script>
  let p = document.querySelector("p");

  function mostrarMissatge() {
    alert("Has fet clic sobre el paràgraf");
    p.removeEventListener("click", mostrarMissatge);
  }

  p.addEventListener("click", mostrarMissatge);
</script>
```

Hem de tenir en compte que només podem eliminar funcions de captura que tinguin nom, no podem anul·lar funcions anònimes associades a esdeveniments.

Captura d'esdeveniments en elements dinàmics

A l'hora d'implementar captures d'esdeveniments, cal tenir en compte que els *listeners* s'associen als elements del DOM que existeixin en aquest moment. Això pot donar lloc a problemes si no es va amb compte.

Suposem que volem crear un document en el qual aconseguim que, en fer clic en tots els paràgrafs (<p>), es mostri en consola el text del paràgraf.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```

<p>1</p>
<p>2</p>
<p>3</p>
<p>4</p>

<script>
  const paragraphs = document.querySelectorAll("p");

  for (let p of paragraphs) {
    p.addEventListener("click", ()=> console.log(p.textContent));
  }
</script>

```

Hem de recórrer tots els paràgrafs i assignar-li un esdeveniment a cadascun d'ells de manera individual.

No obstant això, si afegim un cinquè paràgraf mitjançant JavaScript:

```

let nouParagraf = document.createElement("p");
nouParagraf.textContent = "Cinc";
document.body.appendChild(nouParagraf);

```

Aquest paràgraf apareixerà al final dels altres quatre, però en fer clic sobre ell no passarà res. La raó és lògica. S'ha creat després del bucle i per això no s'ha executat el mètode **addEventListener** sobre ell. Hauríem d'afegir el codi que permeti capturar l'esdeveniment en el nou element.


```

nouParagraf.addEventListener("click",
  ()=>console.log(nouParagraf.textContent));

```

8.5.2 L'objecte d'esdeveniment

Quan es produeix un esdeveniment, el navegador crea automàticament un objecte les propietats del qual poden ser molt útils als desenvolupadors. La informació fonamental que graven són les coordenades del cursor del ratolí, si hi ha tecles premudes, l'element que ha produït l'esdeveniment, etc.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

Ja hem vist que, quan es produeix un esdeveniment, s'invoca automàticament el codi d'una funció. Doncs bé, aquesta funció pot tenir un paràmetre que serà una referència a l'objecte d'esdeveniment. Gràcies a aquest paràmetre podrem llegir la informació de l'esdeveniment.

Una de les propietats que posseeix aquest objecte es diu **target** i és una referència a l'element que ha provocat l'esdeveniment.

```
nouParagraf.addEventListener("click", (e)=>{
  console.log(e.target.textContent);
});
```


Seguint amb l'exemple anterior, podríem obtenir per exemple el text de l'element que ha causat l'esdeveniment.

Obtenir coordenades

Els objectes d'esdeveniment tenen dues propietats per a obtenir les coordenades del ratolí en el moment de l'esdeveniment: **clientX** i **clientY**. La primera obté la posició horitzontal en píxels i la segona la vertical. Ambdues ho fan utilitzant com a referència la cantonada superior esquerra de la finestra del navegador, que serà el punt (0,0) de coordenades.

Hi ha dues propietats semblants: **screenX** i **screenY**. La diferència és que l'origen de coordenades no és la cantonada del navegador, sinó de la pantalla. Coincideixen moltes vegades, però si la finestra no està maximitzada no coincidiran.

Altres coordenades que s'emmagatzemen en l'objecte d'esdeveniment són **pageX** i **pageY**. Funcionen com **clientX** i **clientY**, però no sols prenen les de la finestra, tenen en compte el desplaçament realitzat en l'element. És a dir, si hem avançat dues pantalles de 500 píxels d'alt usant les barres de desplaçament i el cursor està a meitat de pantalla, **clientY** ens diria 250, mentre que **pageY** ens diria 1250. Depèn del que desitgem ens vindran millor les unes o les altres coordenades.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs


Obtenir la tecla premuda

En esdeveniments de teclat, l'objecte d'esdeveniment té informació sobre la tecla premuda. Són importants aquestes dades en aplicacions com la programació de jocs o en el control de l'entrada pel teclat.

La propietat més important és **key** que obté el text que indica la tecla premuda. És el més còmode de programar perquè és fàcil d'entendre. Així els seus valors que es retornen són:

- Si és una tecla normal (**lletres o nombres**), ens retorna el caràcter en minúscula o majúscula o el nombre. Per exemple: **"a", "J", "5", "2"**.
- Amb **Shift**, quan prems una tecla juntament amb Shift, `event.key` retorna el caràcter modificat:
 - Ex: Si prems **Shift + a**, `event.key` retornarà **"A"**.
 - Si prems **Shift + 5** en un teclat QWERTY, normalment retornarà **"%"**.
- Amb **AltGr** (Alt Dret) o Alt Esquerre: és una combinació de Alt + Ctrl i s'utilitza per accedir a caràcters especials en alguns teclats (per exemple, el símbol @ o €).
 - Ex: Si prems **AltGr + 2** en un teclat espanyol, `event.key` retornarà **"@"**.
 - Alt Esquerre: Si prems només Alt Esquerre amb una tecla alfanumèrica, no sol modificar el valor retornat per `event.key`.
- Per tecles de control com Shift, Control, Alt, etc., `event.key` retorna el nom de la tecla, no un caràcter: **"Control", "F5", "End"**, etc.

A vegades és necessari saber si s'ha premut alhora que la tecla, les tecles de control especial: **Ctrl**, **Alt** o **Shift**. Tenim tres propietats relacionades amb aquestes tecles que retornaran *true* si la tecla en qüestió s'ha premut. Són: **AltKey**, **CtrlKey**, **ShiftKey** i **MetaKey** (tecla dels ordinadors Mac).

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

Obtenir els botons del ratolí

En els esdeveniments del ratolí hi ha propietats comunes amb les tecles. Per exemple, també tenim les propietats **AltKey**, **CtrlKey**, **ShiftKey** i **MetaKey**, per a saber si la pulsació d'alguna d'aquestes tecles acompanya l'esdeveniment de ratolí.

Hi ha altres propietats de coordenades molt interessants. Així, **movementX** i **movementY** ens retornen la diferència en píxels de les coordenades X i Y respecte a l'últim moviment del ratolí.

A més, la propietat **button** retorna el botó de ratolí premut en el moment de l'esdeveniment. Els valors possibles per a aquesta propietat són:

- 0: botó principal del botó.
- 1: botó central del ratolí (en molts casos, el botó de la roda).
- 2: botó secundari del ratolí (en el cas de persones destres, és el botó dret).
- 3: quart botó. En molts ratolins és el de retrocedir la pàgina.
- 4: cinquè botó. En molts ratolins és el d'avançar pàgina.

Anul·lar comportaments predeterminats

A més de propietats, els objectes d'esdeveniment posseeixen mètodes. Un dels més importants és **preventDefault**. No té paràmetres i el que fa és aconseguir que si aquest esdeveniment produïa en l'element un comportament concret per defecte, que aquest comportament no es produeixi.

És més fàcil d'entendre la idea amb un exemple. Suposem que hem posat un enllaç i desitgem que l'usuari confirmi que desitja anar al destí d'aquest enllaç, de manera que si no el confirma, no ens mourem de la pàgina actual. El codi podria ser aquest:

```
<a href="https://www.google.com">Fes clic aquí</a>
<script>
  const link = document.querySelector('a');
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

```

link.addEventListener('click', function() {

    if(confirm("Segur que vols sortir de la pàgina?")){
        window.location = "https://www.google.com";
    }
});
</script>

```

Si executam aquest codi en un navegador ens apareix el model on se'ns demana confirmació, però independentment que acceptem o no, l'acció d'anar a la pàgina de destí es fa de totes maneres, ja que el comportament per defecte de l'etiqueta <a> és aquest. El codi per evitar navegar a una altra pàgina serà el següent:

```

<a href="https://www.google.com">Fes clic aquí</a>
<script>
    const link = document.querySelector('a');
    link.addEventListener('click', function(e) {
        // Evitem el comportament predeterminat (navegar a Google)

        if(!confirm("Segur que vols sortir de la pàgina?")){
            e.preventDefault();
        }
    });
</script>

```

Cancel·lar propagació

Abans hem vist com funciona la propagació d'esdeveniments. Hem vist que els esdeveniments es llancen primer en l'element més interior, després en el següent i així, fins que al final es llancen els relacionats amb els elements més exteriors.

Podem cancel·lar la propagació mitjançant un mètode anomenat **stopPropagation**. Seguint amb l'exemple on tenim un botó dins un paràgraf i aquest dins un contenidor i cada element té un esdeveniment click on es fa una acció, recordem que per defecte, després de pitjar sobre el botó, els dos esdeveniments del paràgraf i el

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

contenidor també es llançaran.

Ho podem evitar d'aquesta manera:

```
button.addEventListener("click", (e)=> {
  console.log("click button");
  e.stopPropagation();
});
```

Llançar esdeveniments

JavaScript permet llançar esdeveniments a voluntat. La idea es basa a crear objectes d'esdeveniment propis i mitjançant el mètode **dispatchEvent** dels elements, enviar l'esdeveniment creat.

La creació de l'esdeveniment es basa en l'objecte **Event** (o en els seus descendents més específics com **MouseEvent** per exemple), el qual funciona indicant el tipus d'esdeveniment en la creació. Per exemple:


```
const customEvent = new Event('customEvent');
```

Després llançaríem l'esdeveniment a qualsevol element que estigui capturant aquest esdeveniment. Un exemple d'ús seria el que proporciona aquest codi:

```
<button id="triggerButton">Llança l'esdeveniment
personalitzat</button>
<div id="output"></div>
<script>
  const triggerButton = document.getElementById('triggerButton');
  const output = document.getElementById('output');

  // Creem un esdeveniment personalitzat
  const customEvent = new Event('customEvent');

  output.addEventListener('customEvent', ()=> {
    output.textContent = 'L\'esdeveniment personalitzat s\'ha
    llançat!';
  });
```


	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```
// Llançar l'esdeveniment quan es fa clic al botó
triggerButton.addEventListener('click', ()=> {
    output.textContent = ''; // Neteja el contingut del contenidor
    output.dispatchEvent(customEvent); // Llança l'esdeveniment
});
</script>
```

Aquest enfocament és útil quan necessites comunicar canvis o estats entre components sense un vincle directe, com ara en arquitectures d'esdeveniments per a interfícies complexes o en aplicacions SPA (Single Page Applications).

8.5.3 Tipus d'esdeveniments

En la majoria d'exemples hem treballat amb l'esdeveniment click, però en realitat hi ha molts altres tipus d'esdeveniments que podem capturar.


Esdeveniments de ratolí

Els esdeveniments de ratolí els produeix aquest dispositiu. Però també els dispositius tàctils poden produir alguns d'aquests esdeveniments. Per exemple, l'esdeveniment clic ocorre quan l'usuari prem i amolla el botó principal del ratolí sobre l'objecte que posseeix el **listener**, però també ocorre si l'usuari, en un dispositiu tàctil, colpeja i amolla el dit sobre aquest objecte.

D'altra banda, encara que els esdeveniments es refereixen al ratolí, realment valen per a qualsevol dispositiu apuntador capaç de moure el cursor per la pantalla.

La taula completa dels esdeveniments de ratolí és la següent:

Esdeveniment	Explicació
click	L'usuari fa clic sobre el botó principal del dispositiu apuntador (o dona un cop amb el dit en un dispositiu tàctil).
dblclick	L'usuari colpeja dues vegades ràpides sobre el botó principal del dispositiu apuntador (o amb el dit en un dispositiu tàctil).
mousedown	L'usuari prem un botó del dispositiu apuntador. Es produeix aquest esdeveniment just quan l'usuari prem el botó, abans que el deixi anar.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs


mouseup	Es produeix quan l'usuari amolla el botó del dispositiu apuntador que havia premut prèviament.
mousenter	Es produeix quan l'usuari mou el cursor del dispositiu apuntador dins de l'element que captura l'esdeveniment. Només es produeix si el cursor estava fora de l'element i l'usuari l'acaba de moure dins.
mouseleave	Es produeix quan l'usuari mou el cursor del dispositiu apuntador fora de l'element que captura l'esdeveniment. Només es produeix si el cursor es trobava dins de l'element i l'usuari l'acaba de moure fora.
mousemove	Es produeix cada vegada que l'usuari mou el cursor dins de l'element (estant prèviament dins).
mouseover	Es produeix quan el cursor apuntador entra dins de l'element o de qualsevol dels fills de l'element.
mouseout	Es produeix quan el cursor apuntador abandona l'element o de qualsevol dels fills de l'element.
contextmenu	Es produeix quan l'usuari acaba de demanar el menú de context. Normalment aquest menú apareix en prémer el botó secundari del dispositiu apuntador. En dispositius tàctils sol ocórrer quan l'usuari deixa el dit atapeït sobre l'element 2 o 3 segons almenys.

Esdeveniments de teclat

Esdeveniment	Explicació
keypress	Es produeix quan un usuari prem i amolla una tecla.
keydown	Es produeix just quan l'usuari ha premut la tecla (abans que la deixi anar).
keyup	Es produeix quan l'usuari amolla la tecla.

Esdeveniments de moviment en la finestra

Esdeveniment	Explicació
scroll	Ocorre quan s'ha desplaçat la finestra a través de les barres de desplaçament o usant el dispositiu tàctil.
resize	Esdeveniment de window que es produeix quan es canvia la grandària de la finestra.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

Esdeveniments sobre càrrega i descàrrega d'elements

Esdeveniment	Explicació
load	S'ha acabat la càrrega de l'element. És un dels elements més importants a capturar per a assegurar que el codi següent funciona amb la seguretat que està carregat el que necessitem. Quan s'aplica a l'element window , es produeix quan tots els elements del document s'han carregat.
DOMContentLoaded	Semblant a l'esdeveniment load . Es produeix quan el document HTML s'ha carregat. A diferència de load , es dispara sense esperar que s'acabin de carregar les fulles d'estils, imatges i elements en segon pla. En general, per a JavaScript, és més convenient aquest esdeveniment, ja que el temps de càrrega és més ràpid.
abort	Es produeix quan s'anul·la la càrrega d'un element.
error	Succeeix si hi ha hagut un error en la càrrega.
progress	Es produeix si la càrrega està en procés.
readystatechange	Ocorre quan s'ha modificat l'estat de l'atribut <i>readystatechange</i> , la qual cosa ocorre quan s'ha modificat l'estat de càrrega i descàrrega.

Altres esdeveniments

La part d'esdeveniments relacionats amb els formularis es troben a un apartat propi. Hi ha més esdeveniments encara. En aquest apartat n'anotarem alguns, que tot i no ser molt habituals, són específics i ens poden servir en algun moment donat.

- **Esdeveniments sobre l'historial**

Esdeveniment	Explicació
popstate	Es produeix si es canvia l'historial

- **Esdeveniments relacionats amb la reproducció de mitjans**

Esdeveniment	Explicació
waiting	Es dona quan el vídeo s'ha aturat per falta de dades.
playing	El mitjà està llest per a la seva reproducció després que s'hagués aturat per falta de dades o altres causes.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

canplay	Es produeix quan es detecta que un vídeo (o un altre element multimèdia) ja es pot reproduir (tot i que no s'hagi carregat del tot).
canplaythrough	Es produeix si s'estima que el vídeo ha carregat prou dades per a poder-se reproduir sense haver d'esperar que n'arribin més.
pause	El mitjà de reproducció s'ha pausat.
play	Es dona quan el mitjà de reproducció s'ha començat a reproduir després d'una pausa.
ended	Ocorre si la reproducció del vídeo o àudio s'ha aturat, sigui per haver arribat al final o perquè no hi ha més dades disponibles.
loadeddata	Es produeix si s'ha carregat el frame actual (normalment el primer).
suspend	Es llança si s'ha suspès la càrrega del mitjà.
emptied	Es llança si s'ha buidat el mitjà per un nou intent de càrrega per part de l'usuari o per altres motius.
stalled	Succeeix davant una fallada en la càrrega, però sense que aquesta s'aturi.
seeking	Ocorre quan s'ha iniciat una feina de cerca en el mitjà.
seeked	Ocorre quan s'ha acabat la feina de cerca.
loadedmetadata	S'han carregat les metadades del mitjà.
durationchange	Es produeix si es modifica l'atribut de durada del mitjà.
timeupdate	Ocorre si s'ha modificat l'atribut <i>currentTime</i> del mitjà
ratechange	Es produeix quan la ràtio del vídeo s'ha modificat.
volumechange	Es llança si el volum s'ha modificat.


- **Esdeveniments d'arrossegament**

Estan relacionats amb la interfície d'arrossegament que és part d'HTML5. Perquè un element pugui ser arrossegable ha de tenir l'atribut **draggable** col·locat amb valor **true**:

```
<div id="divArrossegable" draggable="true">Sóc arrossegable</div>
```

En aquesta mena d'operacions hi ha dues capes protagonistes: una capa arrossegable (la que realment s'arrossega) i un possible destí de l'arrossegament.

La capa que s'arrossega pot generar aquests esdeveniments:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

Esdeveniment	Explicació
dragstart	Es produeix quan l'usuari comença a arrossegar l'element.
drag	Ocorre, una vegada començat l'arrossegament, cada vegada que es continua arrossegant l'element (cada vegada que el movem).
dragstop	Es produeix quan l'arrossegament acaba.

Mentre que els esdeveniments que es produeixen en l'element destí de l'arrossegament són els següents:


Esdeveniment	Explicació
dragenter	Es produeix quan l'element que s'està arrossegant entra en l'element destí.
dragover	Ocorre cada vegada que es continua, una vegada entrat, arrossegant l'element d'origen sobre el destí.
dragleave	Es llança quan l'element que s'arrossega surt del destí.
drop	Ocorre quan l'element origen s'amolla dins el destí. Perquè aquest esdeveniment es pugui capturar s'ha d'eliminar el comportament per defecte de l'esdeveniment dragover .

- **Esdeveniments sobre animacions i transicions**

Esdeveniment	Explicació
animationstart	Es produeix quan una animació comença sobre l'element.
animationinteraction	Es produeix just quan es repeteix l'animació.
animationend	Es produeix quan acaba l'animació.
transitionrun	Es llança quan l'element està preparat per a començar la transició.
transitionstart	Ocorre quan s'inicia una transició.
transitionend	Ocorre en acabar la transició.

- **Esdeveniments del porta-retalls**

Esdeveniment	Explicació
cut	Es produeix quan l'usuari intenta retallar contingut de l'element.
copy	Es produeix quan l'usuari intenta copiar contingut de l'element.
paste	Es produeix quan l'usuari intenta aferrar contingut.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

- **Esdeveniments especials**

Esdeveniment	Explicació
offline	Només funciona per l'objecte window i es dona si el navegador es desconnecta d'Internet.
online	Només funciona per l'objecte window i es produeix si el navegador es torna a connectar a la xarxa després d'haver estat desconnectat.
fullscreenchange	Es produeix quan un element passa a mode de pantalla completa.
fullscreenerror	Succeeix si hi ha un error en passar un element al mode de pantalla completa.
message	Esdeveniment que s'associa a molts elements d'enviament de missatges com els que es produeixen a través d'APIs WebSockets , WebWorkers i d'altres.

8.5.4 Formularis


El formulari com a objecte del DOM

Sens dubte, els formularis són el component fonamental de captura d'esdeveniments en una aplicació web. Ens han estat molt útils els mètodes **alert**, **prompt** i **confirm**, però la realitat és que les aplicacions professionals no els utilitzen mai. Si cal mostrar missatges es fan en elements HTML normals i la introducció de dades per part de l'usuari es fa sempre amb formularis.

Els formularis són la base de la comunicació de les aplicacions web amb l'usuari. L'objecte **document** disposa d'una propietat anomenada **forms** que retorna una col·lecció amb tots els formularis del document. Així, **document.forms[0]** farà referència al primer formulari del document. Però és possible, i més recomanable de fet, accedir al formulari mitjançant altres mètodes, per exemple, el seu identificador.

L'objecte de formulari (**form**) té aquestes propietats i mètodes:

Propietat o mètode	Ús
--------------------	----

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

elements	Retorna una col·lecció que conté tots els controls del formulari.
length	Obté el nombre de controls del formulari.
action	Retorna el contingut de la propietat <i>action</i> , que marca la URL de destí de les dades del formulari. Permet la seva modificació.
method	Retorna el contingut de la propietat <i>method</i> del formulari, que marca el mètode d'enviament de dades (get o post). Permet la seva modificació.
enctype	Obté o canvia la manera de codificar les dades del formulari.
acceptCharset	Obté o modifica el conjunt de caràcters del formulari.
submit() requestSubmit()	Envia les dades del formulari al seu destí.
reset()	Deixa els valors dels controls del formulari al seu estat per defecte.

Esdeveniments de formulari

- **Esdeveniment submit**

Es tracta de l'esdeveniment que produeix l'enviament de les dades del formulari al seu destí. És molt habitual validar les dades prèviament al seu enviament perquè l'usuari detecti les fallades abans de realitzar l'enviament. Aquest esdeveniment es produeix just abans de l'enviament i s'associa al propi formulari (també es pot capturar en el botó de tipus **submit** del formulari). Després de la seva captura podem validar les dades, però tenint la precaució que si no cancel·lem l'acció per defecte d'aquest esdeveniment (mitjançant el **preventDefault** de l'objecte d'esdeveniment), l'acció es durà a terme.

Com a exemple, suposem que tenim un formulari en el qual demanem a l'usuari el seu nom d'usuari i que aquest nom només poden ser lletres de l'alfabet anglès i amb un mínim de 6 caràcters. No enviarem les dades si no es compleix aquesta premissa i quan detectem la fallada, avisarem de l'error i no permetrem l'enviament. El codi podria ser aquest:

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```

<form id="form" action="url-to-server/file.php">
  <input type="text" id="usuari" name="usuari">
  <button type="submit">Envia</button>
</form>
<div id="missatge"></div>
<script>
  const form = document.getElementById("form");
  const missatge = document.getElementById("missatge");

  form.addEventListener("submit", (e)=>{
    let exp=/^[a-zA-Z]{6,}$/;
    if(exp.test(form["usuari"].value) == false){
      e.preventDefault();
      missatge.innerHTML = "<p>Error: nom d'usuari no vàlid</p>";
    }
  });
</script>

```


Explicació expressió regular:

- `^`: Comença a verificar des del principi de la cadena.
- `[a-zA-Z]`: Accepta només lletres majúscules (A-Z) i minúscules (a-z).
- `{6,}`: Requereix un mínim de 6 lletres (sense límit màxim).
- `$`: Indica que ha de coincidir fins al final de la cadena.

La captura de l'esdeveniment permet aturar l'enviament de dades si no es compleix l'expressió regular en el valor del quadre de text en el qual s'escriu el nom de l'usuari.

- **Esdeveniment reset**

Aquest esdeveniment es produeix quan s'ha ordenat, a través d'un botó de tipus **reset** per exemple, que el formulari mostri els valors inicials i esborri el que l'usuari hagués escrit. La captura de l'esdeveniment ens permetrà matisar aquesta operació o afegir accions a aquesta.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

- **Esdeveniment de focus**

Són esdeveniments associats als controls de formulari en els quals l'usuari pot escriure o triar opcions. Un control obté el **focus** quan en fer servir el teclat manegem aquest control. L'enfocament s'aconsegueix gràcies a la interacció amb el dispositiu apuntador (un clic de ratolí per exemple) o al canvi de control de formulari gràcies a la tecla del tabulador.

Un control de formulari llança l'esdeveniment **focus** quan obté el focus i llança **blur** quan el perd. També existeixen les variants **focusin** i **focusout**, on els esdeveniments es propaguen cap als elements pares.

- **Esdeveniments d'entrada**

L'esdeveniment **change** és, segurament, un dels esdeveniments més importants dels formularis. Es produeix quan modifiquem el valor de qualsevol control de formulari. Exemple d'això són:


- Canviar l'estat d'un botó d'opció (radio button) o d'una casella de verificació (checkbox).
- Modificar el valor d'un quadre de text o contrasenya.
- Triar un altre valor d'una llista d'opcions.

Realment, l'**esdeveniment es produeix quan s'ha modificat el valor i, a més, s'ha perdut el focus sobre aquest control**. És a dir, durant el canvi no es llança l'esdeveniment.

L'esdeveniment **input** es dispara cada vegada que el valor d'un camp d'entrada canvia (en temps real).

L'esdeveniment **select** es dispara quan l'usuari selecciona text en un camp d'entrada o text.

- **Altres esdeveniments útils**

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

L'esdeveniment **invalid** es dispara quan un camp no compleix la validació especificada (com **required**, **pattern**, etc.).


Propietats dels controls

Per a modificar les propietats dels controls de formulari podem treballar sobre els seus atributs com fem amb qualsevol altre element. Per exemple, per a obtenir el valor d'un control de formulari podem fer el següent:

```
console.log(control.getAttribute("value"));
```

Suposant que **control** és el nom d'un control de formulari, efectivament apareix el valor. Però, el valor és una cosa que es canvia de manera dinàmica en els formularis. Si l'usuari ha modificat, per exemple, el valor d'un quadre de text perquè ha escrit alguna cosa en ell, **getAttribute("value")** no reflecteix aquests canvis. Per això és millor emprar propietats que sí que reflecteixin els canvis que fa l'usuari. Aquestes propietats són:

Propietat	Ús	Controls
name	Nom del control	Casi tots
type	Valor de l'atribut <i>type</i> .	Controls de tipus input
value	Retorna el valor actual del control	Casi tots
checked	Pot valer <i>true</i> o <i>false</i> .	Control de tipus "radio button" o "checkbox".
defaultChecked	Indica l'estat inicial de la propietat <i>checked</i> en el control.	Control de tipus "radio button" o "checkbox".
disabled	Indica, amb un booleà, si el control està deshabilitat o no.	Casi tots
readonly	Indica, amb un booleà, si el control és de només lectura o no.	Casi tots
required	Indica, amb un booleà, si és obligatori o no canviar el valor del control.	Controls d'entrada de text o llistes.
maxLength	Permet veure i modificar la propietat que defineix la longitud màxima de text.	Controls d'entrada de text
min	Valor mínim possible pel control	Input de tipus numèric o

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

		data.
max	Valor màxim possible pel control.	Input de tipus numèric o data.
step	Mínim valor de canvi de control. Si el valor és 1, els valors avancen d'un en un com a mínim.	Input de tipus numèric o data.
selectionStart	En controls d'entrada de text indica l'índex dins del text general en el que comença la selecció actual sobre el text. Si no hi ha res seleccionat, indica la posició del cursor en el text.	Controls d'entrada de text
selectionEnd	En controls d'entrada de text indica l'índex dins del text general en el que acaba la selecció actual sobre el text. Si no hi ha res seleccionat, indica la posició del cursor en el text.	Controls d'entrada de text


Mètodes dels controls

Els controls també ofereixen mètodes amb els que realitzar operacions que simulin l'acció de l'usuari. La llista és la següent:

Mètode	Ús	Controls
focus()	Força que el control obtingui el focus.	Casi tots
blur()	Provoca la pèrdua de focus en el control.	Casi tots
select()	Selecciona tot el text en el control i també deixa el focus en ell.	Controls d'entrada de text
setSelectionRange(inici, final)	Selecciona el text que va des de la posició d'inici fins la posició final (sense incloure la darrera).	Controls d'entrada de text

8.6 LocalStorage

Per tal d'emmagatzemar informació persistent a través del navegador, HTML5 va introduir algunes millores i novetats que van molt més enllà del que abans era

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

permès, encara que amb certs límits: **emmagatzemar informació en local**, en el client. Amb l'emmagatzematge web, les aplicacions web poden emmagatzemar dades localment dins del navegador de l'usuari.

Per què hauríem de voler guardar dades en el client? Un dels usos possibles és crear aplicacions i programes que funcionin fins i tot sense necessitat de connectar-se a Internet. HTML5 pot treballar en tots els sistemes i dispositius, de manera que la possibilitat de desar les dades sobre el client faciliten el desenvolupament d'aquestes aplicacions, en les quals és bàsic.

Abans d'HTML5, les dades de l'aplicació s'havien d'emmagatzemar en **cookies**, incloses en cada sol·licitud del servidor. L'emmagatzematge web és més segur, i grans quantitats de dades es poden emmagatzemar localment, sense afectar el rendiment del lloc web. A diferència de les cookies, el límit d'emmagatzematge és molt més gran (almenys 5 MB, 10 MB per a la majoria dels navegadors).

L'emmagatzematge web és per origen (per domini i protocol). Totes les pàgines, d'un origen, poden emmagatzemar i accedir a les mateixes dades.


L'emmagatzematge web HTML té dos objectes per guardar dades al client:

- **window.localStorage** - emmagatzema dades sense data de caducitat.
- **window.sessionStorage** - emmagatzema dades per a una sessió (les dades es perden quan es tanca la pestanya del navegador).

Les cookies encara són útils, especialment quan es tracta d'autenticació, però hi ha moments en què l'ús de localStorage o sessionStorage pot ser una alternativa millor.

La sintaxi de localStorage i sessionStorage és la mateixa.

Dins de JavaScript tenim aquests dos objectes que bàsicament no són més que matrius relacionals o taules de resum, és a dir, matrius els índexs de les quals no són numèrics de 0... a qualsevol cosa, sinó paraules clau que no es poden repetir.

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGs DAW/DAM - 1r curs

La diferència entre tots dos és que el que es desa a la `localStorage` està disponible malgrat tancar el navegador, mentre que en el cas de la sessió `Storage` no ho està.

No podem emmagatzemar objectes directament, però podem passar els objectes a JSON.

8.6.1 Creació, lectura, actualització, eliminació i neteja de registres

Es poden **crear** registres per a l'objecte `localStorage` utilitzant el mètode **`setItem()`**. El mètode `setItem()` pren dos paràmetres, la clau i el valor corresponent:

```
let key = 'Item 1';
localStorage.setItem(key, 'Value');
```

Per a **llegir** registres, es fa servir el mètode **`getItem()`**. El mètode `getItem()` pren un paràmetre que ha de ser la clau. Aquesta funció retornarà el valor corresponent com a cadena:

```
let myItem = localStorage.getItem(key);
```

Aquest codi estableix el *myItem* igual a «*Value*», que és el valor corresponent per a la clau.


L'**actualització** d'un registre es fa amb el mètode **`setItem()`**. De nou, calen dos paràmetres. El paràmetre *key* serà una clau existent mentre que el paràmetre *value* serà un valor nou:

```
localStorage.setItem(key, 'New Value');
```

Ara, el valor de `localStorage` per a la clau és 'New value' en lloc de 'Value'.

Podem crear, llegir i actualitzar registres a l'objecte `localStorage`. També podem eliminar registres individuals i netejar tots els registres del `localStorage`.

Es pot **eliminar** un registre amb el mètode **`removeItem()`**. El mètode `removeItem()`

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

pren un argument que serà una clau de l'objecte localStorage:

```
localStorage.removeItem(key);
```

També pots **esborrar tots els elements** del localStorage. Això es pot fer amb el mètode **clear()**:

```
localStorage.clear();
```

Aquests mètodes ens donen més capacitat per eliminar i netejar elements de localStorage ràpidament. No obstant això, hi ha alguns límits a localStorage. **Tant localStorage com sessionStorage només poden emmagatzemar cadenes**. Per treballar al voltant d'això, haurem d'emprar **mètodes JSON**.

8.6.2 Com guardar valors que no són cadenes de caràcters amb JSON

localStorage només pot emmagatzemar valors de cadena. Si volem emmagatzemar objectes o arrays com a valors a localStorage, podem utilitzar **JSON.stringify()** per a convertir-los en cadenes. En crear o actualitzar parelles clau/valor a localStorage, utilitzeu JSON.stringify() amb l'objecte o array com a argument:

```
let myObj = { name: 'Skip', breed: 'Labrador' };
localStorage.setItem(key, JSON.stringify(myObj));
```

Tot i que myObj és un objecte, JSON.stringify(myObj) el converteix en una cadena. Així que myObj és ara un valor vàlid d'emmagatzematge local.

Per a **llegir i retornar** valors de cadena, farem servir el mètode **JSON.parse()**. El mètode JSON.parse() pren cadenes JSON i les converteix en objectes JavaScript. JSON.parse() pren .getItem() com a paràmetre:

```
let item = JSON.parse(localStorage.getItem(key));
```

Ara l'element s'assigna igual al valor de la clau. En el fragment de codi anterior, el valor de la clau s'ha establert a la versió de cadena del myObj. L'ús de JSON.parse

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGES DAW/DAM - 1r curs

converteix el myObj de nou en un objecte. Així que l'element és igual a myObj com a objecte, no com a cadena.

Ser capaç de convertir arrays i objectes en cadenes utilitzant JSON.stringify i convertir-los de nou utilitzant JSON.parse és molt útil. També és necessari saber com comprovar si localStorage està buit o no.

8.6.3 Altres opcions

Es poden utilitzar expressions if per comprovar si localStorage conté elements o si està buit. Per fer-ho, comproveu la longitud de localStorage:

```
if (localStorage.length > 0) {
    // ...
}
```

Si localStorage.length és més gran que 0, llavors hi ha elements dins de l'objecte localStorage. Inclou un *else* en cas que no hi hagi elements a localStorage.

Es pot incloure codi que s'aplicarà a les expressions if i else. És possible que vulguis iterar sobre els elements a localStorage.

Els objectes localStorage i sessionStorage no suporten el mètode forEach(). Per a iterar sobre els elements a localStorage, utilitzeu un bucle *for*. El mètode key() pren un enter com a argument i retorna la clau corresponent. Amb un bucle *for*, passa i com a enter per a key():

```
for (let i = 0; i < localStorage.length; i++){
    let key = localStorage.key(i);
}
```

Es fa servir el mètode getItem per a retornar el valor corresponent per a la clau o crear una sentència console.log per a imprimir la clau i el valor a la pantalla:

```
for (let i = 0; i < localStorage.length; i++){
```

	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Llenguatge de marques i sistemes de gestió d'informació	CFGS DAW/DAM - 1r curs

```

let key = localStorage.key(i);
let value = localStorage.getItem(key);
console.log(key, value);
}

```

La `key()` és molt útil per a iterar a través de `localStorage` utilitzant bucles *for*. No tots els navegadors admeten `localStorage`.

Podeu provar la compatibilitat amb `localStorage` comprovant si està disponible a l'objecte `window` utilitzant una declaració `if`:

```

if (window.localStorage) {
    // localStorage supported
}

```