
	IES JOAN RAMIS I RAMIS Departament d'Informàtica	UD8: JavaScript
	Entorns de desenvolupament	CFGES 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.....	12
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


	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

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

de qualsevol tipus i competir amb els llenguatges clàssics de programació.

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

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

emmagatzemar diferents tipus de dades durant l'execució de l'script.

- 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>

```

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

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

```

<!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>

```

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

- **Fitxer extern** (external file)

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.

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

Existeixen dos tipus de comentaris:

- **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'
```

	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 PI = 3.1416; // Declaració de constant emprant 'const'
```

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.

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

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

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

servir cometes simples o dobles i el *backslash* per a caràcters especials (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*).

	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 messi = true;
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" }
```

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

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.

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
```

	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 multiplicacio = a * b; // 50
var divisio = a / b; // 2
var modul = a % b; // 0

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";

```

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

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

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

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

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

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;
```

	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 b = 10;

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

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"];
```


	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(nombres[1]); // "María"
```

També es poden inicialitzar d'aquesta manera:

```
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:

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

Existeixen uns quants mètodes per a treballar amb arrays:

- **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'];
```

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

```

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

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

const fruites1 = ['poma', 'plàtan'];
const fruites2 = ['taronja', 'rem'];
const combinacioFruites = fruites1.concat(fruites2);
console.log(combinacioFruites); // Output: ['poma', 'plàtan', 'taronja', 'rem']

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

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

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

fruites = ['poma', 'plàtan', 'taronja'];
const taronjaIndex = fruites.indexOf('taronja');
console.log(taronjaIndex); // Output: 2
fruites = ['poma', 'plàtan', 'taronja'];

```

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

```
const unioElements = fruites.join(', ');
console.log(unioElements); // Output: "poma, plàtan, taronja"
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'empregar **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.

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

Anem a veure exemples d'ús per a:

- **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 semblant 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
());
```

	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 (ciutats); // ['Zaragoza', 'madrid',
'Barcelona']
```

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

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

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 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",
  },
  {
```

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

```

        "municipi": "Barcelona",
        "provincia": "Barcelona",
    }
];

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;

```


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

```

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

```

	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 "aragorn":
        edat = 532;
        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;

```

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

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

	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(i);
}
// Output: 0 1
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:

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

	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 unClient = {
  nom: "Peter Jackson",
  "Adreça del client": "C/ Desconegut",
  "-+--+": "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.

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

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 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 = {
```

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

```

    descripcio: "factura de prova",
    preu: 100.0,
    iva: 5.0,
    subtotal: function () {
        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";
}

```


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

```

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

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;
    };
}

```

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

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

```
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.

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

Per a solucionar això cal fer servir *prototype*:

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

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 = {
```

	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: "Andreu",
    edat: 25,
  };
const { nom, edat } = persona;
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

```

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

```
${edat} anys.`;
```

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

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**

	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 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).

```
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)**

Les arrow functions són una sintaxi més concisa per a definir funcions en ES6. S'utilitzen fletxes (*=>*) en lloc de la paraula clau *function*.

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

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

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

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

- **Mètodes per a arrays**

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.

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

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

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:

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

	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 següent:

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

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

```
provincies.set(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

```
console.log(provincies); //Output Map(3) { 1 → "Granada",
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:

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

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

```
for (let p of provincies) {
```

```
    console.log(p);
```

```
}
```

```
//Output:
```

```
Array [ 5, "Pontevendra" ]
```

```
Array [ 7, "Madrid" ]
```

```
Array [ 22, "Balears" ]
```

```
Array [ 12, "Segovia" ]
```

```
for (let [clau, valor] of provincies) {
```

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

```
}
```

```
//Output:
```

```
5 Pontevendra
```

```
7 Madrid
```

```
22 Balears
```

```
12 Segovia
```