
Tutorial de JavaScript

Índex

1. Introducció	1
2. On inserir JavaScript	1
2.1. Dins d'una etiqueta <code><script></code> al mateix HTML	1
2.2. A la capçalera <code><head></code>	2
2.3. Fitxer extern (recomanat)	2
3. Sortida de dades	2
4. Sintaxi bàsica	3
4.1. Instruccions	3
4.2. Comentaris	3
4.3. Sensibilitat a majúscules	3
5. Variables: <code>var</code>, <code>let</code> i <code>const</code>	4
5.1. <code>var</code> (antiga, evitar)	4
5.2. <code>let</code> (àmbit de bloc, recomanada)	4
5.3. <code>const</code> (constant)	4
5.4. Noms de variables	4
6. Tipus de dades	4
6.1. Comprovació de tipus amb <code>typeof</code>	5
6.2. Conversió de tipus	5
7. Operadors	6
7.1. Aritmètics	6
7.2. Assignació	6
7.3. Comparació	6
7.4. Lògics	7
7.5. Operador ternari	7
8. Estructures condicionals	7
8.1. <code>if</code> / <code>else if</code> / <code>else</code>	7
8.2. <code>switch</code>	7
9. Bucles	8
9.1. <code>for</code>	8
9.2. <code>while</code>	8
9.3. <code>do...while</code>	8
9.4. <code>for...of</code> (per a iterables: arrays, strings...)	8
9.5. <code>for...in</code> (per a propietats d'objectes)	9
9.6. <code>break</code> i <code>continue</code>	9
10. Cadenes de text (Strings)	9
10.1. Creació	9
10.2. Propietats i mètodes bàsics	9
10.3. Template literals	10
11. Números	10
11.1. Creació i operacions	10

11.2. Mètodes útils	10
12. Arrays	11
12.1. Creació i accés	11
12.2. Mètodes principals	11
13. Funcions	12
13.1. Declaració clàssica	12
13.2. Expressió de funció	12
13.3. Funció fletxa (Arrow function, ES6)	12
13.4. Paràmetres per defecte	12
13.5. Rest parameters i Spread	13
13.6. Funcions d'ordre superior (Higher-order functions)	13
14. Objectes	13
14.1. Creació i accés	13
14.2. Destructuring	14
14.3. Spread i Object.assign	14
14.4. Mètodes d'Object	14
14.5. Constructors i new	14
15. Àmbit (Scope) i Hoisting	15
15.1. Àmbit global i local	15
15.2. Àmbit de bloc amb let i const	15
15.3. Hoisting	15
15.4. Mode estricte ("use strict")	16
16. El DOM (Document Object Model)	16
16.1. Selecció d'elements	16
16.2. Modificar contingut i atributs	16
16.3. Modificar estils	17
16.4. Crear i eliminar elements	17
17. Esdeveniments (Events)	17
17.1. Mètode addEventListener	17
17.2. Esdeveniments comuns	18
17.3. Exemple pràctic: formulari	18
18. Classes i POO	18
18.1. Definir una classe	18
18.2. Herència amb extends	19
18.3. Getters, setters i mètodes estàtics	19
19. Promeses i programació asíncrona	20
19.1. Callbacks	20
19.2. Promeses (Promises)	20
19.3. async / await (ES2017, recomanat)	21
19.4. fetch API (peticions HTTP)	21
19.5. Promise.all i Promise.race	21
20. Mòduls ES6	22
20.1. Exportar	22

20.2. Importar	22
20.3. Ús en HTML	22
Exemples: 3 programes senzills	23
Resum de bones pràctiques	23
Referències	23
Índex de figures	24

1. Introducció

JavaScript (JS) és el **llenguatge de programació del web**. Permet afegir interactivitat a les pàgines HTML: respondre a clics, validar formularis, actualitzar contingut dinàmicament, comunicar-se amb servidors, etc.



Figura 1: JavaScript logo

Característiques principals:

- **Interpretat**: s'executa directament al navegador, sense compilació prèvia.
- **Dinàmic**: els tipus de variables es resolen en temps d'execució.
- **Orientat a objectes i funcional**: admet múltiples paradigmes de programació.
- **Multiplataforma**: funciona en tots els navegadors moderns i també al servidor (Node.js).

JavaScript **no té res a veure** amb Java. Malgrat el nom, són llenguatges completament diferents.

2. On inserir JavaScript

JavaScript es pot incloure a una pàgina HTML de tres maneres:

2.1. Dins d'una etiqueta `<script>` al mateix HTML

```
<html>
<body>
  <h1>La meva primera pàgina</h1>
  <script>
    document.write("Hola, món!");
  </script>
</body>
</html>
```

2.2. A la capçalera <head>

```
<head>
  <script>
    function saluda() {
      alert("Hola!");
    }
    saluda();
  </script>
</head>
```

2.3. Fitxer extern (recomanat)

```
<script src="script.js"></script>
```

Separar el JavaScript en un fitxer .js extern és la pràctica recomanada perquè:

- Millora la llegibilitat del codi.
- Permet la memòria cau del navegador.
- Facilita el manteniment.

3. Sortida de dades

JavaScript disposa de diversos mètodes per mostrar informació:

Mètode	Descripció
<code>document.write()</code>	Escriu directament al document HTML
<code>console.log()</code>	Mostra a la consola del navegador (per a depuració)
<code>alert()</code>	Mostra un quadre de diàleg emergent
<code>innerHTML</code>	Modifica el contingut d'un element HTML

```
// Consola del navegador (F12)
console.log("Valor de la variable:", x);

// Modificar un element HTML
document.getElementById("resultat").innerHTML = "Hola!";

// Quadre d'alerta
alert("Operació completada");
```

4. Sintaxi bàsica

4.1. Instruccions

Cada instrucció acaba amb punt i coma ; (opcional però recomanat):

```
let x = 5;  
let y = 10;  
let z = x + y;  
console.log(z); // 15
```

4.2. Comentaris

```
// Comentari d'una sola línia  
  
/*  
    Comentari  
    de múltiples línies  
*/
```

4.3. Sensibilitat a majúscules

JavaScript diferencia entre majúscules i minúscules (*case-sensitive*):

```
let nom = "Anna";  
let Nom = "Bernat"; // variable diferent!
```

5. Variables: var, let i const

5.1. var (antiga, evitar)

```
var x = 10;  
var x = 20; // Permet redeclarar (problemàtic)
```

5.2. let (àmbit de bloc, recomanada)

```
let edat = 25;  
edat = 26; // Es pot reassignar  
// let edat = 30; // ERROR: no es pot redeclarar
```

5.3. const (constant)

```
const PI = 3.14159;  
// PI = 3; // ERROR: no es pot modificar
```

Regla pràctica: usa const per defecte i let quan necessitis reassignar el valor. Evita var.

5.4. Noms de variables

- Poden contenir lletres, dígit, _ i \$.
- Han de començar per lletra, _ o \$.
- No poden ser paraules reservades (if, for, function...).
- Convenció: **camelCase** (nomComplet, totalFactura).

6. Tipus de dades

JavaScript té 8 tipus de dades bàsics:

```
// Primitius  
let text    = "Hola";           // String  
let enter   = 42;               // Number  
let decimal = 3.14;            // Number  
let cert     = true;           // Boolean  
let res      = undefined;      // Undefined  
let buit    = null;           // Null  
let gran     = 9007199254740991n; // BigInt  
let simbol   = Symbol("id");    // Symbol  
  
// Compost
```

```
let objecte = { nom: "Anna", edat: 30 }; // Object
```

6.1. Comprovació de tipus amb typeof

```
typeof "Hola"      // "string"  
typeof 42           // "number"  
typeof true        // "boolean"  
typeof undefined   // "undefined"  
typeof null        // "object" (comportament històric de JS)  
typeof {}          // "object"  
typeof []          // "object"  
typeof function(){} // "function"
```

6.2. Conversió de tipus

```
// A número  
Number("42")      // 42  
Number("3.14")     // 3.14  
Number(true)      // 1  
Number(false)     // 0  
parseInt("10px")   // 10  
parseFloat("3.5em") // 3.5  
  
// A cadena  
String(42)        // "42"  
(42).toString()   // "42"  
  
// A booleà  
Boolean(0)        // false  
Boolean("")       // false  
Boolean(null)     // false  
Boolean(1)        // true  
Boolean("hola")   // true
```

7. Operadors

7.1. Aritmètics

```
let a = 10, b = 3;
console.log(a + b); // 13 (suma)
console.log(a - b); // 7 (resta)
console.log(a * b); // 30 (multiplicació)
console.log(a / b); // 3.333... (divisió)
console.log(a % b); // 1 (mòdul / residu)
console.log(a ** b); // 1000 (potència)
console.log(++a); // 11 (increment)
console.log(--b); // 2 (decrement)
```

7.2. Assignació

```
let x = 10;
x += 5; // x = x + 5 → 15
x -= 3; // x = x - 3 → 12
x *= 2; // x = x * 2 → 24
x /= 4; // x = x / 4 → 6
x %= 4; // x = x % 4 → 2
x **= 3; // x = x ** 3 → 8
```

7.3. Comparació

```
5 == "5" // true (igualtat amb conversió de tipus)
5 === "5" // false (igualtat estricta, sense conversió)
5 != "5" // false
5 !== "5" // true
5 > 3 // true
5 < 3 // false
5 >= 5 // true
5 <= 4 // false
```

CONSELL

Bona pràctica: usa sempre `===` i `!==` per evitar errors de conversió inesperats.

7.4. Lògics

```
true && false    // false (AND)
true || false    // true  (OR)
!true            // false (NOT)
```

7.5. Operador ternari

```
let edat = 20;
let acces = (edat >= 18) ? "Permès" : "Denegat";
console.log(acces); // "Permès"
```

8. Estructures condicionals

8.1. if / else if / else

```
let nota = 7;

if (nota >= 9) {
  console.log("Excel·lent");
} else if (nota >= 7) {
  console.log("Notable");
} else if (nota >= 5) {
  console.log("Aprovat");
} else {
  console.log("Suspès");
}
```

8.2. switch

```
let dia = "dilluns";

switch (dia) {
  case "dilluns":
  case "dimarts":
  case "dimecres":
  case "dijous":
  case "divendres":
    console.log("Dia laborable");
    break;
  case "dissabte":
  case "diumenge":
    console.log("Cap de setmana");
    break;
}
```

```
default:
  console.log("Dia desconegut");
}
```

9. Bucles

9.1. for

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

9.2. while

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

9.3. do...while

```
let i = 0;
do {
  console.log(i); // S'executa almenys una vegada
  i++;
} while (i < 5);
```

9.4. for...of (per a iterables: arrays, strings...)

```
let fruites = ["poma", "pera", "maduixa"];
for (let fruita of fruites) {
  console.log(fruita);
}
```

9.5. for . . . in (per a propietats d'objectes)

```
let persona = { nom: "Anna", edat: 30, ciutat: "Barcelona" };
for (let clau in persona) {
    console.log(clau, ":", persona[clau]);
}
```

9.6. break i continue

```
for (let i = 0; i < 10; i++) {
    if (i === 5) break;    // Atura el bucle
    if (i % 2 === 0) continue; // Salta els parells
    console.log(i); // Mostra: 1, 3
}
```

10. Cadenes de text (Strings)

10.1. Creació

```
let s1 = "Hola";           // cometes dobles
let s2 = 'Món';            // cometes simples
let s3 = `Hola, ${s2}!`;   // template literal (ES6)
```

10.2. Propietats i mètodes bàsics

```
let text = "JavaScript és genial";

text.length;           // 21
text.toUpperCase();     // "JAVASCRIPT ÉS GENIAL"
text.toLowerCase();     // "javascript és genial"
text.indexOf("és");     // 11
text.includes("genial"); // true
text.startsWith("Java"); // true
text.endsWith("al");    // true
text.slice(0, 10);      // "JavaScript"
text.substring(11, 13); // "és"
text.replace("genial", "increïble"); // "JavaScript és increïble"
text.split(" ");        // ["JavaScript", "és", "genial"]
text.trim();            // Elimina espais als extrems
text.padStart(25, "*"); // "*****JavaScript és genial"
text.repeat(2);         // "JavaScript és genialJavaScript és
↪  genial"
```

10.3. Template literals

```
let nom = "Bernat";
let edat = 22;
let msg = `Em dic ${nom} i tinc ${edat} anys.
Nació: ${2026 - edat}`; // Admeten múltiples línies i expressions
```

11. Números

11.1. Creació i operacions

```
let n1 = 42;
let n2 = 3.14;
let n3 = 1e6;      // 1000000 (notació científica)
let n4 = 0xFF;     // 255 (hexadecimal)
let n5 = Infinity; // Infinit
let n6 = NaN;      // Not a Number
```

11.2. Mètodes útils

```
let num = 3.14159;

num.toFixed(2);      // "3.14" (arrodonit, retorna string)
num.toPrecision(4);  // "3.142"
Number.isInteger(42); // true
Number.isNaN(NaN);   // true
Number.isFinite(Infinity); // false

Math.round(4.7);     // 5
Math.floor(4.9);     // 4
Math.ceil(4.1);      // 5
Math.abs(-7);        // 7
Math.max(3, 7, 2);   // 7
Math.min(3, 7, 2);   // 2
Math.pow(2, 8);       // 256
Math.sqrt(144);      // 12
Math.random();       // Número aleatori entre 0 i 1

// Número aleatori enter entre 1 i 100
let aleatori = Math.floor(Math.random() * 100) + 1;
```

12. Arrays

12.1. Creació i accés

```
let fruites = ["poma", "pera", "maduixa"];
console.log(fruites[0]); // "poma"
console.log(fruites.length); // 3
fruites[3] = "kiwi"; // Afegir element
```

12.2. Mètodes principals

```
let a = [1, 2, 3, 4, 5];

// Afegir / eliminar
a.push(6); // Afegeix al final → [1,2,3,4,5,6]
a.pop(); // Elimina el darrer → [1,2,3,4,5]
a.unshift(0); // Afegeix al principi → [0,1,2,3,4,5]
a.shift(); // Elimina el primer → [1,2,3,4,5]
a.splice(2, 1); // Elimina 1 element a l'índex 2 → [1,2,4,5]
a.splice(2, 0, 3); // Insereix 3 a l'índex 2 → [1,2,3,4,5]

// Cerca
a.indexOf(3); // 2
a.includes(4); // true
a.find(x => x > 3); // 4 (primer element que compleix)
a.findIndex(x => x > 3); // 3 (índex del primer)

// Transformació
a.slice(1, 3); // [2,3] (subarray, no modifica l'original)
a.reverse(); // [5,4,3,2,1] (modifica l'original)
a.sort(); // Ordena (com a text per defecte!)
a.sort((x, y) => x - y); // Ordena numèricament ascendent

// Iteració funcional
let dobles = a.map(x => x * 2); // Nou array transformant cada
↪ element
let parells = a.filter(x => x % 2 === 0); // Nou array filtrant
↪ elements
let suma = a.reduce((acc, x) => acc + x, 0); // Acumula un valor
a.forEach(x => console.log(x)); // Executa una funció per cada
↪ element

// Combinar arrays
let b = [6, 7, 8];
let c = a.concat(b); // [1,2,3,4,5,6,7,8]
let d = [...a, ...b]; // Equivalent amb spread operator
```

13. Funcions

13.1. Declaració clàssica

```
function suma(a, b) {  
    return a + b;  
}  
console.log(suma(3, 4)); // 7
```

13.2. Expressió de funció

```
const multiplica = function(a, b) {  
    return a * b;  
};
```

13.3. Funció fletxa (Arrow function, ES6)

```
const divideix = (a, b) => a / b;  
const quadrat = n => n * n;  
const saluda = () => "Hola!";  
  
// Bloc de codi si necessita més d'una instrucció  
const factorial = n => {  
    if (n <= 1) return 1;  
    return n * factorial(n - 1);  
};
```

13.4. Paràmetres per defecte

```
function saluda(nom = "visitant") {  
    return `Hola, ${nom}!`;  
}  
saluda("Anna"); // "Hola, Anna!"  
saluda();       // "Hola, visitant!"
```

13.5. Rest parameters i Spread

```
// Rest: recull tots els arguments en un array
function suma(...nums) {
  return nums.reduce((acc, n) => acc + n, 0);
}
suma(1, 2, 3, 4); // 10

// Spread: expandeix un array com a arguments
let nums = [1, 2, 3];
console.log(Math.max(...nums)); // 3
```

13.6. Funcions d'ordre superior (Higher-order functions)

```
function aplica(fn, valor) {
  return fn(valor);
}
aplica(x => x * 2, 5); // 10
```

14. Objectes

14.1. Creació i accés

```
let persona = {
  nom: "Anna",
  edat: 30,
  ciutat: "Barcelona",
  saluda: function() {
    return `Hola, em dic ${this.nom}`;
  }
};

console.log(persona.nom);      // "Anna"
console.log(persona["edat"]);  // 30
console.log(persona.saluda()); // "Hola, em dic Anna"
```

14.2. Destructuring

```
const { nom, edat } = persona;
console.log(nom); // "Anna"
console.log(edat); // 30

// Amb nom diferent
const { nom: name, edat: age } = persona;

// En arrays
const [primer, segon] = ["a", "b", "c"];
```

14.3. Spread i Object.assign

```
let persona2 = { ...persona, edat: 25 }; // Còpia amb modificació
let copia = Object.assign({}, persona);
```

14.4. Mètodes d'Object

```
Object.keys(persona); // ["nom", "edat", "ciutat", "saluda"]
Object.values(persona); // ["Anna", 30, "Barcelona", f()]
Object.entries(persona); // [["nom","Anna"], ["edat",30], ...]
```

14.5. Constructors i new

```
function Cotxe(marca, model, any) {
  this.marca = marca;
  this.model = model;
  this.any = any;
  this.descripcio = function() {
    return `${this.marca} ${this.model} (${this.any})`;
  };
}

let cotxe1 = new Cotxe("Seat", "Ibiza", 2020);
console.log(cotxe1.descripcio()); // "Seat Ibiza (2020)"
```

15. Àmbit (Scope) i Hoisting

15.1. Àmbit global i local

```
let global = "Soc global"; // Accessible des de qualsevol lloc

function exemple() {
  let local = "Soc local"; // Només accessible dins la funció
  console.log(global);      // OK
  console.log(local);      // OK
}

console.log(global); // OK
// console.log(local); // ERROR: local is not defined
```

15.2. Àmbit de bloc amb let i const

```
{
  let x = 10;
  const y = 20;
}
// console.log(x); // ERROR: x no és accessible fora del bloc
```

15.3. Hoisting

El **hoisting** és el comportament de JavaScript de “pujar” les declaracions al principi del seu àmbit:

```
// Amb var (declaració pujada, però no la inicialització)
console.log(a); // undefined (no error)
var a = 5;

// Amb let/const (no hi ha hoisting efectiu)
// console.log(b); // ReferenceError
let b = 10;

// Les funcions declarades es pugen completament
saluda(); // "Hola!" (funciona abans de la declaració)
function saluda() { return "Hola!"; }
```

15.4. Mode estricte ("use strict")

```
"use strict";  
x = 10; // ERROR: x no ha estat declarada
```

El mode estricte prevé errors comuns i és activat automàticament als mòduls ES6.

16. El DOM (Document Object Model)

El DOM és la representació en forma d'arbre de tots els elements HTML d'una pàgina. JavaScript pot llegir i modificar el DOM per canviar el contingut i l'estil de la pàgina.

16.1. Selecció d'elements

```
// Per ID (retorna un element)  
let titol = document.getElementById("titol");  
  
// Per classe CSS (retorna HTMLCollection)  
let botons = document.getElementsByClassName("boto");  
  
// Per etiqueta (retorna HTMLCollection)  
let paragrafs = document.getElementsByTagName("p");  
  
// Selectors CSS (querySelector retorna el primer)  
let primer = document.querySelector(".classe");  
let tots = document.querySelectorAll("p.destacat");
```

16.2. Modificar contingut i atributs

```
let el = document.getElementById("resultat");  
  
el.innerHTML = "<strong>Nou contingut</strong>"; // Pot incloure HTML  
el.textContent = "Text sense HTML"; // Només text (més  
↪ segur)  
el.setAttribute("class", "actiu");  
el.getAttribute("href");  
el.removeAttribute("disabled");
```

16.3. Modificar estils

```
let el = document.getElementById("caixa");
el.style.color = "red";
el.style.backgroundColor = "#f0f0f0";
el.style.fontSize = "18px";
el.classList.add("actiu");
el.classList.remove("inactiu");
el.classList.toggle("visible");
el.classList.contains("actiu"); // true/false
```

16.4. Crear i eliminar elements

```
// Crear
let nouP = document.createElement("p");
nouP.textContent = "Nou paràgraf";
document.body.appendChild(nouP);

// Inserir en posició concreta
let container = document.getElementById("container");
container.insertBefore(nouP, container.firstChild);

// Eliminar
let el = document.getElementById("vell");
el.remove();
// o: el.parentNode.removeChild(el);
```

17. Esdeveniments (Events)

17.1. Mètode addEventListener

```
let boto = document.getElementById("boto");

boto.addEventListener("click", function(event) {
  console.log("S'ha clicat el botó!");
  console.log(event.target); // Element que ha generat l'event
});

// Amb arrow function
boto.addEventListener("click", (e) => {
  e.preventDefault(); // Evita el comportament per defecte
  e.stopPropagation(); // Atura la propagació
});
```

17.2. Esdeveniments comuns

Categoria	Esdeveniments
Ratolí	click, dblclick, mouseenter, mouseleave, mousemove
Teclat	keydown, keyup, keypress
Formulari	submit, change, input, focus, blur
Document	DOMContentLoaded, load, resize, scroll

17.3. Exemple pràctic: formulari

```
document.getElementById("formulari").addEventListener("submit",
  ↪ function(e) {
    e.preventDefault(); // Evita l'enviament real
    let nom = document.getElementById("nom").value;
    if (nom.trim() === "") {
      alert("El nom no pot estar buit!");
      return;
    }
    console.log("Formulari enviat per:", nom);
  });
```

18. Classes i POO

ES6 va introduir la sintaxi de **classes**, que és una forma més clara de treballar amb la programació orientada a objectes.

18.1. Definir una classe

```
class Animal {
  constructor(nom, especie) {
    this.nom = nom;
    this.especie = especie;
  }

  parla() {
    return `${this.nom} fa soroll.`;
  }

  toString() {
    return `${this.nom} (${this.especie})`;
  }
}

let gat = new Animal("Miau", "Gat");
```

```
console.log(gat.parla()); // "Miau fa soroll."
console.log(gat.toString()); // "Miau (Gat)"
```

18.2. Herència amb extends

```
class Gos extends Animal {
  constructor(nom, raça) {
    super(nom, "Gos"); // Crida el constructor del pare
    this.raça = raça;
  }

  parla() {
    return `${this.nom} bordes!`; // Sobreescriu el mètode
  }

  info() {
    return `${super.toString()} - Raça: ${this.raça}`;
  }
}

let gosset = new Gos("Rex", "Labrador");
console.log(gosset.parla()); // "Rex bordes!"
console.log(gosset.info()); // "Rex (Gos) - Raça: Labrador"
console.log(gosset instanceof Gos); // true
console.log(gosset instanceof Animal); // true
```

18.3. Getters, setters i mètodes estàtics

```
class Cercle {
  constructor(radi) {
    this._radi = radi;
  }

  get radi() { return this._radi; }
  set radi(valor) {
    if (valor < 0) throw new Error("El radi no pot ser negatiu");
    this._radi = valor;
  }

  get area() {
    return Math.PI * this._radi ** 2;
  }

  static crea(diametre) {
    return new Cercle(diametre / 2);
  }
}
```

```
let c = new Cercle(5);
console.log(c.area.toFixed(2)); // "78.54"
let c2 = Cercle.crea(10);      // Mètode estàtic
```

19. Promeses i programació asíncrona

JavaScript és **single-threaded**, però suporta operacions asíncrones (peticions a servidors, temporitzadors...) a través de callbacks, promeses i async/await.

19.1. Callbacks

```
setTimeout(() => {
  console.log("Executat després de 2 segons");
}, 2000);

setInterval(() => {
  console.log("Executat cada segon");
}, 1000);
```

19.2. Promeses (Promises)

```
function obteDades(id) {
  return new Promise((resolve, reject) => {
    // Simulem una crida asíncrona
    setTimeout(() => {
      if (id > 0) {
        resolve({ id: id, nom: "Usuari " + id });
      } else {
        reject(new Error("ID invàlid"));
      }
    }, 1000);
  });
}

obteDades(1)
  .then(dades => {
    console.log("Dades rebudes:", dades);
    return obteDades(2); // Encadenament de promeses
  })
  .then(dades2 => console.log("Dades 2:", dades2))
  .catch(error => console.error("Error:", error.message))
  .finally(() => console.log("Operació finalitzada"));
```

19.3. async / await (ES2017, recomanat)

```
async function carregaUsuari(id) {
  try {
    const dades = await obteDades(id);
    console.log("Usuari:", dades);

    const dades2 = await obteDades(id + 1);
    console.log("Usuari 2:", dades2);
  } catch (error) {
    console.error("Error:", error.message);
  }
}

carregaUsuari(1);
```

19.4. fetch API (peticions HTTP)

```
async function obteUsuaris() {
  try {
    const resposta = await
      ↪ fetch("https://jsonplaceholder.typicode.com/users");
    if (!resposta.ok) {
      throw new Error(`Error HTTP: ${resposta.status}`);
    }
    const usuaris = await resposta.json();
    console.log(usuaris);
    return usuaris;
  } catch (error) {
    console.error("Error en la petició:", error);
  }
}

obteUsuaris();
```

19.5. Promise.all i Promise.race

```
// Executa totes en paral·lel i espera que acabin totes
const [u1, u2] = await Promise.all([obteDades(1), obteDades(2)]);

// Retorna la primera que resolgui
const primera = await Promise.race([obteDades(1), obteDades(2)]);
```

20. Mòduls ES6

Els mòduls permeten dividir el codi en fitxers separats i importar/exportar funcionalitats.

20.1. Exportar

```
// fitxer: utils.js

// Exportació nomenada
export const PI = 3.14159;
export function suma(a, b) { return a + b; }
export class Calculadora { /* ... */ }

// Exportació per defecte (una per fitxer)
export default function multiplica(a, b) { return a * b; }
```

20.2. Importar

```
// fitxer: main.js

// Importació nomenada
import { PI, suma } from "./utils.js";

// Importació amb àlies
import { suma as addicio } from "./utils.js";

// Importació per defecte
import multiplica from "./utils.js";

// Importar tot
import * as utils from "./utils.js";
utils.suma(2, 3);
```

20.3. Ús en HTML

```
<script type="module" src="main.js"></script>
```

Els mòduls s'executen en **mode estricte** per defecte i les variables no contaminen l'àmbit global.

Exemples: 3 programes senzills

- [Pintura interactiva](#)
- [Pilota que rebot](#)
- [Esquiva obstacles](#)

Resum de bones pràctiques

- Usa `const` per defecte, `let` quan necessitis reassignar. Evita `var`.
- Prefereix `===` i `!==` per a les comparacions.
- Escriu funcions petites amb una sola responsabilitat.
- Usa `async/await` en lloc de callbacks niats (*callback hell*).
- Separa el codi en fitxers (mòduls ES6) per a projectes grans.
- Gestiona sempre els errors amb `try/catch`.
- Comenta el codi quan sigui necessari, però escriu codi prou clar per no necessitar comentaris obvis.
- Usa eines de linting (ESLint) per detectar errors i mantenir estils coherents.

Referències

- [W3Schools JavaScript Tutorial](#)
- [MDN Web Docs - JavaScript](#)
- [JavaScript.info](#)
- [p5.js](#)

Índex de figures

1	JavaScript logo	1
---	---------------------------	---

Versions d'aquest document

- [HTML](#)
- [PDF](#)
- [ODT](#)
- [MD](#)

[Domini Públic \(CC0\)](#)