

# Referencia Rápida de JavaScript

ES6+, DOM, eventos, Fetch API, async/await

## Básico

### Variables

```
let name = "Alice"; // reassignable
const PI = 3.14; // constant
var old = "avoid"; // function-scoped (legacy)
```

### Tipos de datos

<b>string</b>	Texto: "hello" o 'hello'
<b>number</b>	Entero o decimal: 42, 3.14
<b>boolean</b>	true/false
<b>null</b>	Valor vacío intencional
<b>undefined</b>	Declarado pero sin asignar
<b>object</b>	Pares clave-valor: { a: 1 }
<b>array</b>	Lista ordenada: [1, 2, 3]
<b>symbol</b>	Identificador único

### Verificación y conversión de tipos

```
typeof "hello" // "string"
typeof 42 // "number"
Number("42") // 42
String(100) // "100"
parseInt("3.9") // 3
parseFloat("3.14") // 3.14
```

## Strings

### Template Literals

```
const name = "Alice";
`Hello, ${name}!` // Hello, Alice!
`Total: ${2 + 3}` // Total: 5
`Multi
line string`
```

### Métodos de string

<b>s.length</b>	Número de caracteres
<b>s.toUpperCase()</b>	Copia en MAYÚSCULAS
<b>s.toLowerCase()</b>	Copia en minúsculas
<b>s.trim()</b>	Eliminar espacios iniciales y finales
<b>s.split(",")</b>	Dividir en array
<b>s.includes("x")</b>	Verificar si contiene → bool
<b>s.indexOf("x")</b>	Primer índice (-1 si no existe)
<b>s.slice(1, 4)</b>	Subcadena por índice
<b>s.replace(a, b)</b>	Reemplazar primera coincidencia
<b>s.replaceAll(a, b)</b>	Reemplazar todas las coincidencias
<b>s.startsWith(x)</b>	Verificar prefijo → bool
<b>s.endsWith(x)</b>	Verificar sufijo → bool
<b>s.padStart(n, c)</b>	Rellenar al inicio hasta longitud n

## Arrays

### Crear y acceder

```
const fruits = ["apple", "banana", "cherry"];
fruits[0] // "apple"
fruits.length // 3
fruits.at(-1) // "cherry"
```

### Métodos que mutan

<b>arr.push(x)</b>	Agregar al final
<b>arr.pop()</b>	Eliminar y devolver el último
<b>arr.unshift(x)</b>	Agregar al inicio
<b>arr.shift()</b>	Eliminar y devolver el primero
<b>arr.splice(i, n)</b>	Eliminar n elementos en el índice i
<b>arr.sort()</b>	Ordenar en el lugar (lexicográfico)
<b>arr.reverse()</b>	Invertir en el lugar

### Métodos que no mutan

<b>arr.map(fn)</b>	Transformar cada elemento
<b>arr.filter(fn)</b>	Mantener elementos donde fn sea verdadero
<b>arr.reduce(fn, init)</b>	Acumular en un único valor
<b>arr.find(fn)</b>	Primera coincidencia o undefined
<b>arr.findIndex(fn)</b>	Índice de la primera coincidencia (-1)
<b>arr.includes(x)</b>	Verificar si contiene → bool
<b>arr.slice(a, b)</b>	Copia superficial de una porción
<b>arr.join(",")</b>	Unir en string
<b>arr.forEach(fn)</b>	Iterar (sin valor de retorno)
<b>[...a, ...b]</b>	Concatenar arrays (spread)

## Objetos

### Crear y acceder

```
const user = { name: "Alice", age: 20 };
user.name // "Alice"
user["age"] // 20
user.gpa = 3.85; // add/update
```

### Destructuring y spread

```
const { name, age } = user;
const copy = { ...user, age: 21 };
```

### Métodos de objeto

<b>Object.keys(o)</b>	Array de claves
<b>Object.values(o)</b>	Array de valores
<b>Object.entries(o)</b>	Array de pares [clave, valor]
<b>Object.assign(t, s)</b>	Copiar propiedades de s a t
<b>"k" in obj</b>	¿Existe la clave? → bool
<b>delete obj.k</b>	Eliminar propiedad
<b>Object.freeze(o)</b>	Hacer inmutable (superficial)

## Flujo de control

### if / else if / else

```
if (score >= 90) {
  grade = "A";
} else if (score >= 80) {
  grade = "B";
} else {
  grade = "C";
}
```

### Ternario y coalescencia nula

```
const status = score >= 60 ? "pass" : "fail";
const name = user.name ?? "Anonymous";
```

### switch

```
switch (color) {
  case "red": stop(); break;
  case "green": go(); break;
  default: wait();
}
```

## Bucles

### for / for...of / for...in

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

for (const item of ["a", "b"]) { } // arrays

for (const key in obj) { } // object keys
```

### while / do...while

```
while (count < 10) { count++; }

do { count++; } while (count < 10);
```

### break y continue

```
for (let i = 0; i < 10; i++) {
  if (i === 5) break; // stop loop
  if (i % 2 === 0) continue; // skip
}
```

## Funciones

### Declaración y arrow function

```
function greet(name) {
  return `Hello, ${name}!`;
}
const greet = (name) => `Hello, ${name}!`;
const square = x => x * x; // single param
```

### Parámetros por defecto y rest

```
function greet(name = "World") { }

function sum(...nums) {
  return nums.reduce((a, b) => a + b, 0);
}
```

### Callbacks

```
[1, 2, 3].map(x => x * 2); // [2, 4, 6]
[1, 2, 3].filter(x => x > 1); // [2, 3]
setTimeout(() => console.log("done"), 1000);
```

## Clases

```
class Dog {
  constructor(name, breed) {
    this.name = name;
    this.breed = breed;
  }
  bark() { return `${this.name} says Woof!`; }
}

class Puppy extends Dog {
  constructor(name, breed, toy) {
    super(name, breed);
    this.toy = toy;
  }
}
```

## Manejo de errores

```
try {
  JSON.parse("bad json");
} catch (err) {
  console.error(err.message);
} finally {
  console.log("always runs");
}
```

### Lanzar errores

```
throw new Error("Something went wrong");
```

## DOM

### Seleccionar elementos

```
document.querySelector(".cls") // first match
document.querySelectorAll("li") // all matches
document.getElementById("main")
```

# Referencia Rápida de JavaScript

## Modificar elementos

```
el.textContent = "new text";
el.innerHTML = "<b>bold</b>";
el.style.color = "red";
el.classList.add("active");
el.classList.toggle("hidden");
el.setAttribute("data-id", "42");
```

## Eventos

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

## Crear elementos

```
const li = document.createElement("li");
li.textContent = "New item";
ul.appendChild(li);
el.remove(); // remove element
```

## Fetch API

### Solicitud GET

```
fetch("https://api.example.com/data")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

### Solicitud POST

```
fetch(url, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ key: "value" }),
});
```

## Async / Await

```
async function loadData() {
  try {
    const res = await fetch(url);
    const data = await res.json();
    return data;
  } catch (err) {
    console.error(err);
  }
}
```

## Solicitudes en paralelo

```
const [users, posts] = await Promise.all([
  fetch("/users").then(r => r.json()),
  fetch("/posts").then(r => r.json()),
]);
```

## Módulos

### Exportaciones nombradas

```
// math.js
export const PI = 3.14;
export function add(a, b) { return a + b; }

// main.js
import { PI, add } from "./math.js";
```

### Exportación por defecto

```
// logger.js
export default function log(msg) { }

// main.js
import log from "./logger.js";
```