# JavaScript — Complete Guide

## Title

JavaScript — Complete Guide
Author: Generated by ChatGPT
Created: 2025-08-19 14:31 UTC

A concise but comprehensive JavaScript reference and guide covering core language features, modern patterns, DOM, async, Node.js, and best practices.

## Table of Contents



## 1. Introduction & History

JavaScript is a high-level, interpreted programming language originally created by Brendan Eich in 1995. It is the lingua franca of the web, used on both client and server (Node.js). ECMAScript (ES) is the specification that standardizes JavaScript; modern development follows ES6+ (ES2015 and later).

## 2. Basics (syntax, types, variables)

Primitive types: undefined, null, boolean, number, bigint, string, symbol.
Objects & functions are reference types.
Variable declarations:
- var: function-scoped, hoisted
- let: block-scoped
- const: block-scoped, immutable binding
Template literals: `Hello ${name}`
Type coercion: == vs === (use === for strict equality)
Type checking: typeof, instanceof, Array.isArray()

Examples:
let x = 10;
const name = 'Aarav';
let arr = [1,2,3];

## 3. Operators, Control Flow, Loops

Operators: arithmetic (+, -, *, /, %), assignment (=, +=), comparison (<, >, <=, >=),

logical (&&, ||, !), nullish coalescing (??), optional chaining (?.).
Control flow: if/else, switch
Loops: for, while, do...while, for...of, for...in
Example:
for (const item of arr) { console.log(item); }

## 4. Functions & Scope

Function declaration vs expression:
function add(a,b) { return a+b }
const add = (a,b) => a+b; // arrow function
Arrow functions do not bind their own 'this'.
Default parameters, rest (...args), spread (...) usage.
Closures: functions that remember lexical scope.
Examples:
function outer(x) { return function(y) { return x+y; } }

## 5. Objects, Prototypes & Classes

Object literal: const o = {a:1, b:2}
Property access: dot and bracket notation.
Prototype chain: inheritance via prototypes. ES6 classes are syntactic sugar over prototypes.
class Person { constructor(name){ this.name = name } greet(){ return `Hi ${this.name}` } }
Object methods: Object.assign, Object.keys, Object.values, Object.entries, Object.create

## 6. Arrays & Built-in Methods

Common methods: push, pop, shift, unshift, slice, splice, map, filter, reduce, find, findIndex, some, every, sort, flat, flatMap.
Example: const doubled = arr.map(x => x*2)
Immutable patterns: avoid mutating arrays in functional code (use map/filter/reduce).

## 7. Asynchronous JS (Promises, async/await, Event Loop)

Event loop: call stack, task/microtask queues. Promises represent future values.
Creating a Promise:
new Promise((resolve,reject) => { /* async work */ })
async/await simplifies Promise handling:
async function fetchJSON(url){ const res = await fetch(url); return res.json(); }
Error handling with try/catch and .catch()
Promise utilities: Promise.all, Promise.race, Promise.allSettled, Promise.any

## 8. DOM & Browser APIs

Selecting elements: document.querySelector, querySelectorAll, getElementById.
Event handling: addEventListener('click', handler)
Manipulating DOM: createElement, appendChild, innerHTML (careful with XSS), textContent.
Storage: localStorage, sessionStorage, cookies.
Fetch API for network requests, AbortController to cancel requests.

## 9. Modules, Tooling & Bundlers

ES Modules (ESM): import/export syntax. Example: export function f(){}; import {f} from './mod.js'
CommonJS (Node): module.exports, require()
Tooling: npm, yarn, pnpm. Bundlers: webpack, rollup, parcel, Vite (dev server + build).
Transpilation: Babel for older browser support.

## 10. Node.js Basics

Node.js runtime for server-side JS. Core modules: fs, http, path, events, stream.
Create a simple server:
const http = require('http');
http.createServer((req,res)=>{ res.end('Hello'); }).listen(3000)
Package.json, npm scripts, environment variables, express framework for routing.

## 11. Testing, Debugging, Performance

Testing frameworks: Jest, Mocha, Chai, Playwright (end-to-end), Cypress.
Debugging: browser devtools, node --inspect, breakpoints.
Performance tips: avoid unnecessary reflows, debounce/throttle events, memoize expensive work, use web workers for heavy computations.

## 12. Modern Patterns & Best Practices

Use === and !==, prefer let/const, keep functions small and pure when possible, handle errors explicitly, sanitize user input, avoid global scope pollution, use linters (ESLint), formatters (Prettier).
Accessibility: ARIA roles, semantic HTML.
Security: avoid eval, sanitize HTML, use HTTPS, set proper CORS and CSP headers.

## 13. Cheatsheet & Useful Snippets

Debounce function:
function debounce(fn, ms){ let t; return (...args)=>{ clearTimeout(t); t = setTimeout(()=>fn(...args), ms) } }
Deep clone (structuredClone when available):
const clone = structuredClone(obj)
Shallow copy of array/object: [...arr], {...obj}
Clone for JSON-safe objects: JSON.parse(JSON.stringify(obj))

## 14. Resources

- MDN Web Docs
- ECMAScript specification
- You Don't Know JS (book series)
- JavaScript.info
- Official Node.js docs
- StackOverflow

## Appendix — Example Projects

1) Todo App (vanilla JS) — add, remove, toggle tasks, persist to localStorage.
2) Fetch + Render — call REST API, render list, pagination, error handling.
3) Express REST API — CRUD for resources, input validation (Joi), JWT auth.

End of guide.