JavaScript — Detailed Complete Reference

Generated: 2025-08-19 14:32 UTC

Title

JavaScript — Detailed Complete Reference
Author: Generated by ChatGPT
Created: 2025-08-19 14:32 UTC

An extensive JavaScript reference covering language fundamentals, detailed examples, advanced topics, browser APIs, Node.js, tooling, patterns, performance, security, and practical recipes.

Table of Contents

1. Language Essentials & Types
2. Execution Context, Hoisting & Scope
3. Operators, Expressions & Coercion
4. Functions, Closures & Higher-Order Functions
5. Objects, Prototypes & Inheritance
6. Classes, Mixins & Composition
7. Arrays & Typed Arrays
8. Strings, RegExp & Internationalization
9. Modules (ESM) & Packaging (CJS)
10. Asynchronous JavaScript & Event Loop
11. Promises, async/await, and Control Flow
12. Web APIs: DOM, Fetch, Storage, Events
13. Advanced Browser APIs: Service Workers, WebSockets, WebRTC
14. Security: XSS, CSP, CORS, Secure Coding
15. Performance & Memory (GC, Profiling)
16. Tooling: npm, bundlers, Babel, TypeScript intro
17. Testing & Debugging (Jest, Playwright, DevTools)
18. Node.js Deep Dive: Streams, Cluster, Child Process
19. Networking: HTTP, REST, GraphQL, WebSockets
20. Databases & ORMs with JS (Mongo, SQL)
21. Design Patterns & Architecture
22. Build, CI/CD, Deployment & Observability
23. Appendix: Cheatsheets, Common Gotchas, Useful Snippets
24. Further Resources and Reading

1. Language Essentials & Types

Primitive types:
- undefined, null (distinct values), boolean, number (IEEE 754 double), bigint, string, symbol.

Reference types:
- Object (plain objects), Function, Array, Date, RegExp, Map, Set, WeakMap, WeakSet, Typed Arrays.

Numeric gotchas: NaN, +0 and -0, Number.isNaN vs isNaN, Number.MAX_SAFE_INTEGER (2**53 - 1) and BigInt for larger integers.
Type checks: typeof, instanceof, Array.isArray, Object.prototype.toString.call(value).
Literal forms and constructors: {}, [], /regex/, new Date(), new Map().

Examples:
```
let n = 42; // number
let b = BigInt('9007199254740993');
let s = `template ${n}`;
let sym = Symbol('id');
```

2. Execution Context, Hoisting & Scope

JavaScript executes code inside execution contexts with lexical environments. There are global, function, and block scopes (let/const).

Hoisting: function declarations and var declarations are hoisted; let/const are hoisted but in temporal dead zone (TDZ) until initialization.
Example:

```
console.log(a); // undefined (var hoisted)
var a = 10;

function foo(){
  console.log(b); // ReferenceError if b is let
  let b = 2;
}
```

Closures: inner functions capture variables from outer lexical scope, enabling private state.

3. Operators, Expressions & Coercion

Operators: unary, binary, ternary, logical, bitwise, optional chaining (?.), nullish coalescing (??), spread, rest.
Coercion rules: ToPrimitive, ToNumber, ToString. Prefer strict equality (===) to avoid surprises from type coercion.
Example pitfalls:

```
[] + [] // '' (string concatenation)
[] + {} // '[object Object]'
{} + [] // 0 (when parsed as block vs expression)
```

Bitwise operations coerce to 32-bit signed integers.

4. Functions, Closures & Higher-Order Functions

Function declaration, expression, arrow functions. Arrow functions do not have their own this, arguments object, or new.target.
Default params, rest parameters, spread syntax.
Closures pattern examples: memoization, private variables, factory functions.
Higher-order functions: functions that accept or return functions (map, filter, reduce are examples).
Example — debounce:

```
function debounce(fn, wait){
  let t;
  return function(...args){
    clearTimeout(t);
    t = setTimeout(()=>fn.apply(this,args), wait);
  }
}
```

5. Objects, Prototypes & Inheritance

Objects are key-value collections; property descriptors (writable, enumerable, configurable) via Object.getOwnPropertyDescriptor and defineProperty.
Prototype chain: objects link to a prototype object; lookup proceeds up the chain.
Object.create(null) creates a truly prototypeless object.
Property lookup and shadowing, hasOwnProperty.
Example: function-based inheritance vs ES6 class syntax.

6. Classes, Mixins & Composition

class syntax provides constructor, methods, static methods, inheritance via extends, super().
Mixins: copy methods into prototypes; composition preferred over deep inheritance.
Private fields (#name) and private methods are supported in modern JS. Example:

```
class Counter{ #count = 0; inc(){ this.#count++ } get count(){ return this.#count }}
```

7. Arrays & Typed Arrays

Array creation, length property, sparse arrays. Iteration methods: forEach, map, filter, reduce, some, every, find, entries, keys, values.
Mutable methods: push, pop, shift, unshift, splice, sort (in-place).
Typed Arrays and ArrayBuffer for binary data (Uint8Array, Float32Array) used in WebGL, audio processing.
Performance tips: avoid repeatedly growing arrays one-by-one in hot loops; pre-allocate if needed.

8. Strings, RegExp & Internationalization

String API: slice, substring, substr (deprecated), includes, startsWith, endsWith, padStart, padEnd, repeat.
Template literals with expressions and tagged templates.
RegExp: flags (g, i, m, u, y, s), lookbehind support in modern engines, capture groups, named groups.
Intl API: Intl.NumberFormat, Intl.DateTimeFormat, Intl.Collator for localization and proper sorting.

9. Modules (ESM) & Packaging (CJS)

ES Modules: export default, named exports, live bindings (import reflects runtime updates to exported bindings).
Top-level await is supported in modern environments.
CommonJS used in older Node: module.exports and require() — important differences: ESM is async and has static analysis benefits.
Package.json fields: main, module, exports, types, scripts, peerDependencies.

10. Asynchronous JavaScript & Event Loop

Event loop phases: macrotasks (task queue), microtasks (Promises job queue), rendering.
Understanding order: current call stack -> microtasks -> next macrotask -> rendering.
setTimeout, setInterval, requestAnimationFrame differences.
Example: Promise.resolve().then(()=>console.log('micro')) ; setTimeout(()=>console.log('macro'),0)
// micro runs first

11. Promises, async/await, and Control Flow

Promises are state machines (pending, fulfilled, rejected). Chain with then/catch/finally.
async functions return promises; await pauses async function execution, yielding to event loop.
Error handling: try/catch inside async, or .catch on returned promise.
Utility functions: Promise.all, allSettled, race, any (ES2021).
Handling cancellation: AbortController pattern, or external token/cancellation library.

12. Web APIs: DOM, Fetch, Storage, Events

DOM selection: querySelector, closest, matches. Event delegation to reduce listeners.
Creating nodes: createElement, templates with <template> tag.
Fetch API: streams, response.clone(), handling JSON/text/blobs. Use AbortController to cancel fetch.
Storage: localStorage (synchronous), IndexedDB (asynchronous, complex but powerful), Cache API for service workers.

13. Advanced Browser APIs: Service Workers, WebSockets, WebRTC

Service Workers enable offline-first apps, intercept network requests, and use Cache API. Lifecycle: install, activate, fetch events.
WebSockets for bidirectional persistent connection (ws protocol). Use in chat apps/realtime features.
WebRTC for peer-to-peer real-time media and data channels—useful for video, audio, and low-latency data transfers.

14. Security: XSS, CSP, CORS, Secure Coding

XSS prevention: escape user data before inserting into innerHTML; use textContent; sanitize HTML on server or use safe libraries.
CSP (Content Security Policy) headers reduce injection risk by restricting script sources.
CORS controls cross-origin access; server must send appropriate Access-Control-Allow-Origin headers for browser to allow requests.
Authentication: avoid storing tokens in localStorage when possible; prefer httpOnly secure cookies for session tokens; mitigate CSRF with same-site cookies or CSRF tokens.

15. Performance & Memory (GC, Profiling)

Garbage collection: generational collectors—short-lived objects collected cheaply. Avoid retaining memory via closures or global caches.
Minimize layout thrashing: batch DOM reads and writes, use transform/opacity for animations when possible.
Use performance.now(), Lighthouse, DevTools profiler to find bottlenecks. Use web workers for CPU-heavy tasks.

16. Tooling: npm, bundlers, Babel, TypeScript intro

npm and package.json basics, semantic versioning, lockfiles (package-lock.json / yarn.lock / pnpm-lock.yaml).
Bundlers: Vite (fast dev), webpack (flexible), Rollup (libraries). Tree-shaking to remove unused exports.
Babel transpilation for older browsers. TypeScript: superset adding static types; basic migration tips and declaration files (.d.ts).

17. Testing & Debugging (Jest, Playwright, DevTools)

Unit testing with Jest (mocks, snapshot testing), integration with testing-library for React. End-to-end with Playwright or Cypress.
Debugging: breakpoints, step-over/into, console, conditional breakpoints, heap snapshots to detect leaks.
Linters and formatters: ESLint + Prettier for consistent code style and catching issues early.

18. Node.js Deep Dive: Streams, Cluster, Child Process

Stream types: readable, writable, duplex, transform. Backpressure handling via pause/resume and stream.pipeline.
Cluster module to take advantage of multiple CPU cores; worker threads for CPU-bound tasks.
Child processes: spawn, exec, execFile for running sub-processes and shell commands.

19. Networking: HTTP, REST, GraphQL, WebSockets

HTTP basics: methods (GET, POST, PUT, DELETE), status codes, headers, cookies, content negotiation.
REST principles: resource-oriented design, statelessness, proper use of status codes.
GraphQL: single endpoint, flexible queries, schema-first approach. Use persisted queries and batching to optimize.
WebSockets for realtime push. Consider reconnection strategies and heartbeat/ping-pong.

20. Databases & ORMs with JS (Mongo, SQL)

MongoDB with Mongoose: schemas, validation, population, transactions (on replica sets).
SQL: use parameterized queries to avoid injection; ORMs: Sequelize, TypeORM, Prisma (popular modern ORM with TypeScript support).
Connection pooling, caching strategies (Redis), and indexing for query performance.

21. Design Patterns & Architecture

Common patterns: Module, Singleton, Factory, Observer (EventEmitter), Strategy, Decorator (via function wrappers), Middleware (Express), Repository pattern.
Architectural styles: Monolith vs Microservices, Serverless functions (FaaS), Client-Server, CQRS for complex systems.

Examples: Express middleware chain, event-driven microservices using message brokers (RabbitMQ, Kafka).

22. Build, CI/CD, Deployment & Observability

Build steps: lint -> test -> build -> bundle -> deploy. CI services: GitHub Actions, GitLab CI, CircleCI.
Deployment options: static hosts (Netlify, Vercel), containers (Docker -> Kubernetes), serverless platforms (AWS Lambda, Cloud Run).
Observability: logging, metrics, tracing (OpenTelemetry), Sentry for error reporting.

23. Appendix: Cheatsheets, Common Gotchas, Useful Snippets

Cheatsheet highlights:
- Equality: use === and !==
- Avoid floating-point equality; use Number.EPSILON for comparisons when needed.
- Use const for values that won't reassign; prefer immutability.

Common gotchas:
- this depends on call-site; arrow functions capture lexical this.
- Floating math precision (0.1 + 0.2 !== 0.3)
- for...in iterates enumerable keys (not recommended for arrays)

Useful snippets: deep clone, debounce, throttle, memoize, retry logic with exponential backoff, safeFetch wrapper using AbortController.

Example — safeFetch:
```
async function safeFetch(url, options={}, timeout=5000){
  const controller = new AbortController();
  const id = setTimeout(()=>controller.abort(), timeout);
  try{
    const res = await fetch(url,{...options, signal: controller.signal});
    return res;
  } finally { clearTimeout(id) }
}
```

24. Further Resources and Reading

Official & high-quality references:
- MDN Web Docs (comprehensive web API documentation)
- ECMAScript Language Specification
- You Don't Know JS (book series by Kyle Simpson)
- JavaScript: The Good Parts (classic but dated)
- Node.js official docs
- Frontend Masters, Pluralsight, and Udemy for focused courses
- Blogs: v8.dev, 2ality, Jake Archibald, MDN blog

Closing Notes

This document provides an in-depth overview but cannot literally contain 'all knowledge' due to the continuously evolving JS ecosystem. Use this as a master reference and jump-off point. If you want, I can:
- Expand any section into a multi-page tutorial with code examples and exercises,
- Generate a version with more code samples for each topic,
- Create a linked HTML version for easier navigation,
- Or split into smaller focused PDFs (Language, Browser APIs, Node.js, Tooling).

Tell me which you prefer and I'll produce it.