

# JAVASCRIPT QUICK REFERENCE

ES6+ · DOM · Fetch · Async

## Basics

### Variables

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

### Data Types

```
string Text: `hello` or `hello`
number Integer or float: 42, 3.14
boolean true / false
null Intentional empty value
undefined Declared but not assigned
object Key-value pairs: {a: 1}
array Ordered list: [1, 2, 3]
```

### Type Checking & Conversion

```
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`
```

### String Methods

```
s.length Number of characters
s.toUpperCase() UPPERCASE copy
s.toLowerCase() lowercase copy
s.trim() Remove leading/trailing whitespace
s.split(",") Split into array
s.includes("x") Contains check → bool
s.indexOf("x") First index (-1 if none)
s.slice(1, 4) Substring by index
s.replace(a, b) Replace first / all matches
s.startsWith(x) Check prefix → bool
```

## Arrays

### Creating & Accessing

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

### Mutating Methods

```
arr.push(x) Add to end
arr.pop() Remove & return last
arr.unshift(x) Add to start
arr.shift() Remove & return first
arr.splice(i, n) Remove n items at index i
arr.sort() Sort in place (lexicographic)
arr.reverse() Reverse in place
```

### Non-Mutating Methods

```
arr.map(fn) Transform each element
arr.filter(fn) Keep elements where fn is true
arr.reduce(fn, init) Accumulate into single value
arr.find(fn) First match or undefined
arr.findIndex(fn) Index of first match (-1)
arr.includes(x) Contains check → bool
arr.slice(a, b) Shallow copy of portion
arr.join(",") Join into string
arr.forEach(fn) Iterate (no return value)
[...a, ...b] Concatenate arrays (spread)
```

## Objects

### Creating & Accessing

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

### Destructuring & Spread

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

### Object Methods

```
Object.keys(o) Array of keys
Object.values(o) Array of [key, value] pairs
Object.entries(o) Copy properties s → t
Object.assign(t, s) Key exists? → bool
"key" in obj Key exists? → bool
delete obj.key Remove property
Object.freeze(o) Make immutable (shallow)
```

## Control Flow

### if / else if / else

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

### Ternary & Falsy Values

```
const status = score >= 60 ? "pass" : "fail";
const name = user.name ?? "Anonymous";
// Falsy: false, 0, "", null, undefined, NaN
```

### switch

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

## Loops

### 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 & continue

```
if (i === 5) break; // exit loop
if (i % 2 === 0) continue; // skip iteration
```

## Functions

### Declaration & Arrow

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

### Default Parameters & 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]
```

## Classes

```
class Dog {
  constructor(name) { this.name = name; }
  bark() { return `this.name says Woof!`; }
}
class Puppy extends Dog {
  constructor(name, toy) {
    super(name);
    this.toy = toy;
  }
}
```

## Error Handling

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

## Throwing Errors

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

## DOM

### Selecting Elements

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

### Modifying Elements

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

### Creating & Removing

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

## Events

### Listening

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

### Common Events

```
click Mouse click on element
input Value changed (input/textarea)
change Value committed (select, checkbox)
submit Form submitted
keydown Key pressed
mouseover Pointer enters element
mouseout Pointer leaves element
DOMContentLoaded HTML parsed, DOM ready
load Page fully loaded
```

## JSON

```
JSON.stringify({ a: 1 }) // '{"a":1}'
JSON.parse('{"a":1}') // {a: 1}
JSON.stringify(obj, null, 2) // pretty print
```

## Fetch API

### GET Request

```
const res = await fetch("/api/users");
const data = await res.json();
```

### POST Request

```
await fetch("/api/users", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ name: "Alice" }),
});
```

## Response Handling

```
res.ok true if status 200-299
res.status HTTP status code
res.json() Parse body as JSON
res.text() Body as plain text
res.headers Response headers
```

## Async / Await

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

## Parallel Requests

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

## Promises

```
const p = new Promise((resolve, reject) => {
  if (ok) resolve(data);
  else reject(new Error("fail"));
});
p.then(val => { }).catch(err => { });
```

## Promise Methods

```
Promise.all([ ]) Wait for all, fail on first error
Promise.allSettled([ ]) Wait for all, never short-circuits
Promise.race([ ]) First to settle (resolve or reject)
Promise.any([ ]) First to resolve
```

## Modern Syntax

### Optional Chaining & Nullish

```
user?.address?.city // undefined if any is null
arr?.[0] // safe array access
fn?.() // safe function call
val ?? "default" // fallback for null/undefined
```

## Destructuring

```
const [a, b, ...rest] = [1, 2, 3, 4];
const { name, age = 0 } = user;
function draw({ x, y }) { }
```

## Modules

### Named & Default Exports

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

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

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

## Map & Set

### Map

```
const m = new Map();
m.set("key", "value");
m.get("key") // "value"
m.has("key") // true
m.delete("key");
m.size // 0
```

### Set

```
const s = new Set([1, 2, 2, 3]);
s.size // 3 (duplicates removed)
s.add(4);
s.has(2) // true
[...s] // [1, 3, 4]
```

## Timers

```
setTimeout(() => { }, 1000); // once after 1s
const id = setInterval(() => { }, 500); // repeat
clearInterval(id); // stop
```

## Console

```
console.log(x) Print to console
console.error(x) Print error (red)
console.warn(x) Print warning (yellow)
console.table(arr) Display as table
console.time(label) Start timer
console.timeEnd(label) Stop timer, print elapsed
```

## Math

```
Math.round(4.7) 5 (nearest integer)
Math.floor(4.7) 4 (round down)
Math.ceil(4.2) 5 (round up)
Math.max(1, 5, 3) 5
Math.min(1, 5, 3) 1
Math.random() 0 to 1 (exclusive)
Math.abs(-5) 5
Math.PI 3.14159...
```

## Date

```
const now = new Date();
now.getFullYear() // 2026
now.getMonth() // 0-11 (Jan = 0)
now.getDate() // 1-31
now.getDay() // 0-6 (Sun = 0)
now.toISOString() // "2026-04-13T..."
now.toLocaleDateString() // locale string
Date.now() // ms since epoch
```

## Storage

```
localStorage.setItem(k, v) Store string (persists)
localStorage.getItem(k) Retrieve value
localStorage.removeItem(k) Delete key
localStorage.clear() Remove all keys
sessionStorage Same API, cleared on tab close
```

## Storing Objects

```
localStorage.setItem("user", JSON.stringify(obj));
const user = JSON.parse(localStorage.getItem("user"));
```