JavaScript Cheat Sheet

Modern ECMAScript & core browser APIs at a glance — grounded in *JavaScript: The Definitive Guide* (7th ed.) and MDN. See the full reference for details: docs → Web Development → JavaScript Development.

DECLARATIONS & LITERALS

```
// block-scoped, prefer these over var
let x = 1;
const PI = 3.14159;      // binding is constant

// template literals
const name = "Ada";
`Hello, ${name}! ${1 + 2}`

// tagged template
tag`raw ${expr} text`
```

DESTRUCTURING & SPREAD

```
const [a, b, ...rest] = [1,2,3,4];
const {x, y: renamed, z = 9} = obj;
function f({a, b} = {}) {}

const merged = {...obj1, ...obj2};
const copy   = [...arr];
const nums   = [1, ...arr2, 9];
```

OPERATORS

`??`	nullish coalescing (null/undefined only)
`?.`	optional chaining, short-circuits to undefined
`&& / ||`	logical AND/OR short-circuit
`===`	strict equality — always prefer over ==
`...`	spread (expand) / rest (collect)
`typeof / instanceof`	primitive tag / prototype chain check

FUNCTIONS

```
function decl(a, b) { return a+b; }
const expr = function(a) {...};

// arrow: lexical `this`, no own arguments
const add = (a, b) => a + b;
const noop = () => {};

function f(a, b = 10, ...rest) {}
```

CLASSES

```
class Shape {
  #id;                        // private field
  static count = 0;
  constructor(id) { this.#id = id; Shape.count++; }
  get id() { return this.#id; }
  area() { throw new Error("impl?"); }
}
class Circle extends Shape {
  constructor(id, r) { super(id); this.r = r; }
  area() { return Math.PI * this.r ** 2; }
}
```

MODULES (ESM)

```
// math.js
export const add = (a,b) => a+b;
export default class Vector {}

// app.js
import Vector, { add } from "./math.js";
import * as math from "./math.js";
const mod = await import("./math.js");
```

ITERATORS & GENERATORS

```
function* range(n) {
  for (let i = 0; i < n; i++) yield i;
}
for (const v of range(3)) { ... }
[...range(3)]              // [0,1,2]

// obj[Symbol.iterator] = function*() {...}
```

PROMISES

```
new Promise((resolve, reject) => {...})
  .then(v => ...)
  .catch(err => ...)
  .finally(() => ...);

Promise.all([p1, p2]);       // fail fast
Promise.allSettled([p1,p2]); // never rejects
Promise.race([p1, p2]);
Promise.any([p1, p2]);
```

ASYNC / AWAIT

```
async function load(url) {
  try {
    const res = await fetch(url);
    if (!res.ok) throw new Error(res.status);
    return await res.json();
  } catch (err) {
    console.error(err);
  }
}
for await (const x of asyncIter) {}
```

STANDARD LIBRARY QUICK REFS

`Set / Map`	unique values / any-key store, insertion order
`WeakMap/WeakSet`	object-only keys, GC-friendly caches
`JSON.stringify/parse`	(value, replacer?, indent?) / (text, reviver?)
`try/catch/finally`	catch(err) { } — err instanceof TypeError
`Proxy / Reflect`	intercept get/set/has traps on an object

ARRAY METHODS

`map(fn)`	new array of fn(el) results
`filter(fn)`	new array of elements where fn is truthy
`reduce(fn, init)`	fold to a single value, left to right
`find / findIndex`	first match (value / index)
`some / every`	any / all elements satisfy fn
`sort(cmp)`	in place — always pass cmp for numbers
`slice(s,e)`	copy [s,e) — non-mutating
`splice(s,n,...v)`	remove/insert in place
`flat(d) / flatMap`	flatten nested arrays d levels
`includes(v)`	membership test (handles NaN)

STRING & REGEXP

`slice/split/trim`	substring / tokenize / strip whitespace
`padStart/padEnd`	pad to a target length
`replace/replaceAll`	(pattern, replacement) — pattern can be RegExp
`/pat/flags`	g=global i=ignoreCase m=multiline s=dotAll u=unicode
`str.match(re) / matchAll`	one match / all matches (needs /g)
`re.test(str) / exec(str)`	boolean / next match with groups

DOM SELECTION & EDITING

```
document.getElementById("id");
document.querySelector(".a > .b");
document.querySelectorAll("li");     // static NodeList

const el = document.createElement("div");
parent.appendChild(el);
parent.insertBefore(el, ref);
el.remove();

el.textContent = "safe text";
el.classList.add/remove/toggle("x");
```

EVENTS

```
el.addEventListener("click", e => {
  e.preventDefault();
  e.stopPropagation();
}, { once: true, capture: false });

// delegation: one listener, many children
list.addEventListener("click", e => {
  if (e.target.matches("li")) {...}
});
// phases: capture -> target -> bubble
```

FETCH & NETWORKING

```
fetch(url, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify(data),
}).then(r => r.ok ? r.json() : Promise.reject(r));

new WebSocket(url).onmessage = e => {...};
new EventSource(url).onmessage = e => {...};
```

STORAGE & WORKERS

`localStorage`	persistent, string-only, per-origin (~5-10MB)
`sessionStorage`	same, but cleared when the tab closes
`IndexedDB`	async, structured, larger-capacity storage
`new Worker(url)`	postMessage/onmessage off the UI thread
`requestAnimationFrame(fn)`	schedule fn before next repaint

Generated with AI assistance from *JavaScript: The Definitive Guide* (7th ed., David Flanagan) and general/official documentation (MDN, tc39.es/ecma262). Verify against the current ECMAScript specification and MDN before relying on this in production. Full detail on every topic above: see the JavaScript Development section of this documentation site.