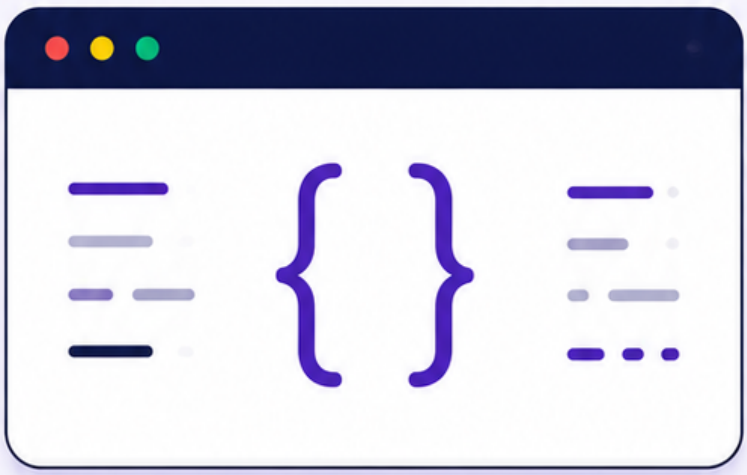
CHEATSHEET SILO

JAVASCRIPT REFERENCE

JavaScript Cheat Sheet

A printable guide to core JavaScript syntax, data types, functions, DOM manipulation, and modern ES6+ features.



CORE JAVASCRIPT • REFERENCE | PRINTABLE A4 | VERSION 1.0 • SEPTEMBER 2026

Free interactive guide: cheatsheetsilo.com/javascript-cheat-sheet/

1. JavaScript Quick Reference

A compact lookup for the syntax and language features used most often in everyday JavaScript.

Syntax	Purpose	Example
console.log()	Print to console	console.log("Hello");
//	Single-line comment	// comment
const	Block-scoped constant	const x = 10;
let	Block-scoped variable	let count = 0;
typeof	Inspect primitive type	typeof value
=== / !==	Strict equality	a === b
?.	Optional chaining	user?.profile?.name
??	Nullish fallback	name ?? "Guest"
...	Spread/rest syntax	const copy = [...items];
`...${}`	Template literal	`Hello ${name}`

Reference area	Includes
Basic syntax	Statements, comments, console output
Variables	var, let, const and scope
Data types	Primitives, objects and typeof
Operators	Arithmetic, comparison, logical, nullish
Strings	Template literals and string methods
Arrays	Iteration, transform, search and mutation
Objects	Properties, spread, destructuring
Functions	Declarations, expressions and arrows
Conditionals	if, else, switch, truthy/falsy
Loops	for, while, for...of
DOM & Events	Select, modify and listen
Async	Promises, async/await and fetch
JSON	parse and stringify
Modules	import and export
Errors	try, catch, finally

2. Variables, Data Types & Operators

JavaScript is dynamically typed. Prefer `const` by default and use `let` only when reassignment is required.

Keyword	Scope	Reassign?	Guidance
const	Block	No	Default choice for bindings
let	Block	Yes	Use when value must change
var	Function	Yes	Legacy; generally avoid in modern code

Type	Example	Notes
string	"hello"	Text
number	42, 3.14	Integers and floating point
bigint	123n	Large integers
boolean	true / false	Logical values
undefined	let x;	Not assigned
null	null	Intentional absence
symbol	Symbol()	Unique primitive
object	{}, [], function	Reference values

Operator group	Common operators	Use
Arithmetic	+ - * / % **	Calculations
Comparison	=== !== > < >= <=	Compare values
Logical	&& || !	Conditional logic
Nullish	??	Fallback only for null/undefined
Optional chaining	?.	Safely access optional properties
Ternary	? :	Compact conditional expression

BEST PRACTICE Prefer === and !== over loose equality. Use ?? when `0`, `false`, or an empty string are valid values.

3. Strings & Arrays

Strings are immutable text values; arrays are ordered mutable collections with powerful iteration and transformation methods.

Common string methods

Method	Purpose	Example
includes()	Contains text	text.includes("JS")
startsWith()	Check prefix	text.startsWith("Hi")
endsWith()	Check suffix	file.endsWith(".js")
slice()	Extract substring	text.slice(0, 5)
replace()	Replace match	text.replace("a","b")
split()	String to array	text.split(",")
trim()	Remove outer whitespace	text.trim()
toUpperCase()	Uppercase	text.toUpperCase()

High-value array methods

Method	Returns / effect	Typical use
map()	New transformed array	Transform every item
filter()	New filtered array	Keep matching items
find()	First matching value	Locate one item
findIndex()	First matching index	Locate position
some()	Boolean	Any item matches?
every()	Boolean	All items match?
reduce()	Accumulated result	Totals/grouping
forEach()	No useful return	Side-effect iteration
includes()	Boolean	Membership check
push()/pop()	Mutates array	Add/remove at end
slice()	New array	Copy/extract
splice()	Mutates array	Insert/remove in place

```
const prices = [10, 20, 30];
const taxed = prices.map(price => price * 1.25);
const total = prices.reduce((sum, price) => sum + price, 0);
```

BEST PRACTICE Use `map`, `filter`, and `reduce` when you want a new result without mutating the original array.

4. Objects & Functions

Objects group related key-value data. Functions are first-class values and can be passed, returned, stored, and composed.

Object task	Syntax
Read property	user.name
Dynamic property	user[key]
Safe nested read	user?.profile?.name
Default value	user.name ?? "Guest"
Own property	Object.hasOwn(user, key)
Keys	Object.keys(user)
Values	Object.values(user)
Entries	Object.entries(user)
Shallow copy	const copy = {...user};
Destructure	const {name, age} = user;

```
function add(a, b) {
return a + b;
}

const multiply = (a, b) => a * b;

function greet(name = "Guest") {
return `Hello ${name}`;
}
```

Function concept	Example / note
Declaration	function run() {}
Expression	const run = function() {};
Arrow function	const run = () => {};
Default parameter	function greet(name = "Guest")
Rest parameters	function sum(...values)
Callback	items.map(item => item.id)
Return value	return result;
Higher-order function	Accepts or returns another function

BEST PRACTICE Use arrow functions for concise callbacks. Use regular functions when their own `this` binding or clearer declarations are useful.

5. Conditionals & Loops

Control flow decides which code runs and how often it runs.

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

Falsy values	Important note
false	Boolean false
0 / -0	Numeric zero
""	Empty string
null	Intentional absence
undefined	Missing value
NaN	Not-a-Number

Loop	Best for	Example
for	Counter/index loop	for (let i = 0; i < 5; i++)
for...of	Iterable values	for (const item of items)
for...in	Enumerable object keys	for (const key in object)
while	Unknown repetition count	while (condition)
break	Exit loop	break;
continue	Skip current iteration	continue;

BEST PRACTICE Order conditional branches from most specific to most general. Named Boolean variables often make complex conditions easier to read.

6. DOM & Events

Browser JavaScript can select elements, change content and attributes, create nodes, and respond to user interaction.

DOM task	Example
Select by ID	document.getElementById("app")
Select first match	document.querySelector(".card")
Select all matches	document.querySelectorAll(".card")
Change text	el.textContent = 'Hello';
Change class	el.classList.add('active');
Toggle class	el.classList.toggle('active');
Set attribute	el.setAttribute('aria-expanded','true');
Create element	document.createElement('div')
Append node	parent.append(child);
Remove node	el.remove();

```
const button = document.querySelector("#btn");

button.addEventListener("click", (event) => {
event.preventDefault();
document.body.classList.toggle("active");
});
```

Event	Common use
click	Buttons and controls
input	Live form input
change	Committed form value
submit	Form submission
keydown	Keyboard interaction
DOMContentLoaded	DOM ready

BEST PRACTICE Prefer `addEventListener()` over inline HTML event attributes. Use event delegation for many dynamic child elements.

7. Async JavaScript, Promises & Fetch

Asynchronous code lets JavaScript wait for network requests, timers, and other operations without blocking the main flow.

```
async function loadUser(id) {
const response = await fetch(`/api/users/${id}`);

if (!response.ok) {
throw new Error(`HTTP ${response.status}`);
}

return response.json();
}
```

Pattern	Purpose
Promise	Represents future completion/failure
.then()	Handle resolved value
.catch()	Handle rejection
.finally()	Run cleanup regardless of outcome
async	Makes a function return a Promise
await	Pause inside async function
Promise.all()	Wait for all; fail fast
Promise.allSettled()	Collect all outcomes
fetch()	Browser HTTP request

BEST PRACTICE Always check `response.ok` when using `fetch()`. A completed HTTP request such as 404 does not automatically reject the fetch Promise.

8. JSON, Modules & Error Handling

JSON is a data-exchange format; modules organize code; structured error handling keeps failures predictable.

JSON task	Syntax
Object -> JSON	JSON.stringify(value)
JSON -> value	JSON.parse(text)
Pretty JSON	JSON.stringify(value, null, 2)

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

// app.js
import { add } from "./math.js";
```

```
try {
const data = JSON.parse(text);
process(data);
} catch (error) {
console.error(error.message);
} finally {
cleanup();
}
```

Error tool	Use
throw	Create/propagate an error
try	Wrap code that may throw
catch	Handle the error
finally	Cleanup that must run
Error	Standard error object
error.message	Human-readable message

BEST PRACTICE Do not silently swallow errors. Handle expected failures and preserve useful context when rethrowing.

9. Modern JavaScript Patterns

A few modern language features remove repetitive checks and make transformations clearer.

Feature	Example	Why useful
Destructuring	const {name} = user;	Extract values clearly
Array destructuring	const [first] = items;	Read positions
Spread	const next = {...user, active: true};	Shallow immutable update
Rest	function sum(...nums)	Collect remaining values
Optional chaining	user?.address?.city	Safe nested access
Nullish coalescing	value ?? "default"	Preserve valid falsy values
Template literals	Hi ${name}	Readable interpolation
Short property syntax	{name, age}	Concise object creation

```
const users = [{ id: 1, active: true }, { id: 2, active: false }];
const activeIds = users
.filter(user => user.active)
.map(user => user.id);
```

BEST PRACTICE Readable code beats clever code. Split long method chains or complex expressions when the intent stops being obvious.

10. Daily Lookup & Best Practices

Keep this final page nearby for the patterns that come up repeatedly.

Task	Recommended syntax
Declare fixed binding	const value = ...
Declare changing binding	let value = ...
Strict compare	a === b
Safe fallback	value ?? fallback
Safe nested property	obj?.child?.value
Transform array	items.map(fn)
Filter array	items.filter(fn)
Find one item	items.find(fn)
Loop values	for (const item of items)
Copy object	{...object}
Copy array	[...items]
Select DOM element	document.querySelector(...)
Listen for event	el.addEventListener(...)
Parse JSON	JSON.parse(text)
Serialize JSON	JSON.stringify(value)
Async request	await fetch(url)
Handle errors	try / catch
Import module	import {x} from './file.js'

Do	Avoid
Prefer `const`, then `let`	Defaulting to `var`
Use strict equality	Unnecessary == coercion
Validate external data	Trusting parsed/API data blindly
Use semantic array methods	Complex manual loops
Use specific error handling	Empty catch blocks
Keep functions focused	Large multi-purpose functions
Use modules	Huge global scripts
Use clear names	Cryptic abbreviations

Full interactive guide: CheatSheetSilo.com/javascript-cheat-sheet/

CheatSheetSilo.com - free, practical cheat sheets for programming, productivity and AI.