SkillNewton

JavaScript Cheat Sheet

Modern JavaScript at a glance: variables, operators, strings, arrays, objects, functions, the DOM, fetch with async/await, modules and errors.

This JavaScript cheat sheet covers the modern syntax you use in browser projects every day, from `const` and template literals to array methods, DOM events and `fetch` with async/await. It is aimed at beginners moving past the basics and at PHP developers adding interactivity to their pages. Each row shows a working snippet and what it does. Try any line in your browser console: press F12 and open the Console tab.

How to use it

1. **Open the console.** Press F12 in Chrome, Edge or Firefox and open the Console tab to run any snippet from this sheet.
2. **Start with variables and operators.** Default to `const`, use `let` when a value changes and always compare with `===`.
3. **Wire up the page.** Use the DOM section to select elements and add event listeners from a script loaded with `defer` or `type="module"`.
4. **Load data.** Follow the fetch pattern with async/await, check `response.ok` and wrap it in try/catch.

Variables and types

`const rate = 18;`	Block-scoped constant binding. Use by default.
`let count = 0;`	Block-scoped variable that can be reassigned.
`var x = 1;`	Old function-scoped declaration. Avoid in new code.
`typeof value`	Returns a string such as "string", "number", "boolean", "object", "undefined" or "function".
`Array.isArray(value)`	True for arrays (`typeof []` is "object").
`null / undefined`	`null` means intentionally empty; `undefined` means not set.
`Number("42") / parseInt("42px", 10)`	Converts strings to numbers. `Number` is strict; `parseInt` reads leading digits.
`String(42) / (42).toFixed(2)`	Number to string; `toFixed` gives "42.00".
`Number.isNaN(x)`	Reliable NaN check.
`10n ** 20n`	BigInt for integers larger than `Number.MAX_SAFE_INTEGER`.

Operators

`a === b / a !== b`	Strict equality: same type and value. Always prefer these.
`a == b`	Loose equality with type conversion: `"1" == 1` is true. Avoid.
`user.name ?? "Guest"`	Nullish coalescing: uses the right side only when the left is null or undefined.

`input || "default"`	Uses the right side for any falsy value (0, "", false, null, undefined, NaN).
`user?.address?.city`	Optional chaining: returns undefined instead of throwing if a link is missing.
`obj.method?.()`	Calls the method only if it exists.
`count ??= 0;`	Assigns only if `count` is null or undefined.
`isAdmin && showPanel()`	Runs the right side only when the left is truthy.
`age >= 18 ? "adult" : "minor"`	Ternary operator.
`2 ** 10`	Exponent: 1024.
`7 % 3`	Remainder: 1.

Strings and template literals

`` `Hello, ${name}!` ``	Template literal with interpolation. Can span several lines.
`str.length`	Number of UTF-16 code units.
`str.includes("seo")`	True if the substring is present.
`str.startsWith("https") / str.endsWith(".pdf")`	Checks the start or end.
`str.slice(0, 5)`	Characters from index 0 up to (not including) 5.
`str.trim()`	Removes surrounding whitespace.
`str.toLowerCase() / str.toUpperCase()`	Changes case.
`str.split(",")`	Splits into an array.
`str.replaceAll("-", " ")`	Replaces every match.
`str.padStart(3, "0")`	"7" becomes "007".
`str.at(-1)`	Last character.

Arrays

`const nums = [3, 1, 2];`	Array literal.
`nums.push(4) / nums.pop()`	Add to or remove from the end (mutates).
`nums.map(n => n * 2)`	New array with each item transformed.
`nums.filter(n => n > 1)`	New array with only matching items.
`nums.reduce((sum, n) => sum + n, 0)`	Reduces to one value; always pass a starting value.
`nums.find(n => n > 1) / nums.findIndex(...)`	First matching item (or undefined) / its index (or -1).
`nums.some(n => n > 2) / nums.every(n => n > 0)`	True if any / all items match.
`nums.includes(2)`	True if the value is in the array.

`nums.toSorted((a, b) => a - b)`	Sorted copy, numerically. `sort()` does the same but mutates the original.
`nums.slice(1, 3)`	Copy of part of the array.
`nums.flat() / nums.flatMap(fn)`	Flatten one level / map then flatten.
`Array.from(document.querySelectorAll("li"))`	Turns an array-like or iterable into a real array.

Objects, destructuring and spread

`const user = { name: "Asha", city: "Pune" };`	Object literal.
`user.name / user["name"]`	Dot and bracket access. Brackets work with variables.
`const { name, city = "Delhi" } = user;`	Destructuring with a default value.
`const { name: fullName } = user;`	Destructure into a differently named variable.
`const [first, ...rest] = nums;`	Array destructuring with rest.
`const copy = { ...user, city: "Mumbai" };`	Shallow copy with one property changed.
`const all = [...a, ...b];`	Join arrays.
`Object.keys(user) / Object.values(user) / Object.entries(user)`	Arrays of keys, values or [key, value] pairs.
`Object.fromEntries(entries)`	Builds an object from [key, value] pairs.
`structuredClone(obj)`	Deep copy of plain data.
`JSON.stringify(obj) / JSON.parse(text)`	Object to JSON text and back.

Functions

`function add(a, b) { return a + b; }`	Function declaration; hoisted, so it can be called before it appears.
`const add = (a, b) => a + b;`	Arrow function with an implicit return.
`const toObj = id => ({ id });`	Wrap an object in brackets to return it from an arrow function.
`function greet(name = "friend") { }`	Default parameter.
`function sum(...nums) { }`	Rest parameter collects arguments into an array.
`function create({ title, price }) { }`	Destructure an options object in the parameter list.
`setTimeout(() => save(), 500);`	Runs a callback once after 500 milliseconds.
`const id = setInterval(tick, 1000); clearInterval(id);`	Repeats every second until cleared.

DOM selection and events

`document.querySelector(".card")`	First element matching a CSS selector, or null.
`document.querySelectorAll(".card")`	Static NodeList of all matches; supports `forEach`.
`el.textContent = "Saved";`	Sets plain text safely.
`el.innerHTML = html;`	Parses HTML. Never use with untrusted input (XSS risk).

`el.classList.add("open") / .remove() / .toggle()`	Change classes.
`el.setAttribute("aria-expanded", "true")`	Sets an attribute.
`el.dataset.id`	Reads `data-id`.
`const li = document.createElement("li"); list.append(li);`	Creates and inserts an element.
`el.remove()`	Removes the element.
`btn.addEventListener("click", handler);`	Runs a function on click.
`form.addEventListener("submit", e => { e.preventDefault(); });`	Stops the normal form submission so JavaScript can handle it.
`list.addEventListener("click", e => { const item = e.target.closest("li"); });`	Event delegation: one listener for many children.

Fetch, promises and async/await

`const res = await fetch("/api/courses");`	Sends a GET request. Rejects only on network failure, not on 404 or 500.
`if (!res.ok) throw new Error(`HTTP ${res.status}`);`	Check the status yourself.
`const data = await res.json();`	Parses a JSON response body.
`fetch(url, { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify(payload) })`	POST a JSON body.
`fetch(url, { method: "POST", body: new FormData(form) })`	POST a form, including files.
`promise.then(fn).catch(handleError)`	Promise chaining without async/await.
`await Promise.all([p1, p2])`	Waits for all; rejects if any one fails.
`await Promise.allSettled([p1, p2])`	Waits for all and reports each result or error.
`const ctrl = new AbortController(); fetch(url, { signal: ctrl.signal }); ctrl.abort();`	Cancels a request.
`fetch(url, { signal: AbortSignal.timeout(5000) })`	Gives up after five seconds.

```
async function loadCourses() {
  try {
    const res = await fetch('/api/courses');
    if (!res.ok) throw new Error(`HTTP ${res.status}`);
    const courses = await res.json();
    console.log(courses);
  } catch (err) {
    console.error('Could not load courses:', err);
  }
}
loadCourses();
```

Modules

`<script type="module" src="/js/app.js"></script>`	Loads a module. Modules are deferred and run in strict mode automatically.
`export function formatPrice(p) { }`	Named export.
`export default class Cart { }`	Default export (one per file).
`import { formatPrice } from "./utils.js";`	Named import. Browsers need the file extension.
`import Cart from "./cart.js";`	Default import.
`import * as utils from "./utils.js";`	Imports everything as one object.
`const { Chart } = await import("./chart.js");`	Dynamic import that loads code only when needed.

Error handling and debugging

`try { risky(); } catch (err) { } finally { }`	`finally` runs whether or not an error was thrown.
`throw new Error("Invalid email");`	Throws an error with a message.
`throw new TypeError("Expected a number");`	Built-in error types: TypeError, RangeError, SyntaxError and more.
`err.message / err.name / err.stack`	Details of a caught error.
`class ValidationError extends Error { }`	Custom error type you can check with `instanceof`.
`console.log(x) / console.table(rows) / console.error(e)`	Print values, arrays of objects as a table, or errors.
`debugger;`	Pauses here when DevTools is open.
`window.addEventListener("error", e => { });`	Catches uncaught errors for logging.

Online version, with the matching free guide: skillnewton.com/resources/cheat-sheets/javascript-cheat-sheet/
Learn this hands-on: PHP Web Development: Foundation & Practical Implementation – skillnewton.com/courses/php-foundation-course/