Spickzettel JavaScript · Webseiten mit Leben

Holger Sebastião · www.sebastiao.org
Version 1.0 · 2026-10-25

JavaScript einbinden — extern

```
<head>
  <link rel="stylesheet"
        href="css/style.css">
  <script src="js/script.js" defer></script>
</head>
```

Wie bei CSS: eine eigene Datei im Ordner `js`. `defer` heißt: Das Skript läuft erst, wenn die ganze Seite geladen ist. Sonst findet es die Elemente noch nicht.

```
website/
├── index.html
├── css/style.css
└── js/script.js
```

Ausgabe und Konsole

```
console.log("Hallo Welt");  // Konsole
console.log("Summe:", summe);
alert("Achtung!");          // Meldefenster
```

Die **Konsole** öffnen: **F12** bzw. Cmd+Alt+I → Reiter **Konsole**. Dort stehen auch alle Fehler in Rot, mit Datei und Zeile.

Was ist anders als in Java?

JAVA	JAVASCRIPT
`int zahl = 5;`	`let zahl = 5;` ohne Datentyp
`final`	`const`
`System.out.println`	`console.log`
`public int summe(…)`	`function summe(…)`
`==` bei Zahlen	`===` immer
`a.equals(b)`	`a === b` auch für Text
`int[]` feste Länge	Array wächst mit `push`
`7 / 2` → `3`	`7 / 2` → `3.5`

Gleich wie in Java: `if`, `else`, `for`, `while`, `switch`, `{ }`, `;`, `//`, `&&`, `||`, `!`. Groß/klein zählt.

Rechnen — Math

`Math.round(2.5)`	runden → `3`
`Math.floor(2.9)`	abrunden → `2`
`Math.random()`	Zufall von `0` bis unter `1`
`10 % 3`	Rest → `1`

Variablen — let · const

```
let punkte = 12;        // änderbar
punkte = punkte + 1;
const MAX_PUNKTE = 15;  // fest
const name = "Ali";
```

`let` für Werte, die sich ändern, `const` für feste Werte. Den Typ merkt sich JavaScript selbst. Alte Programme nutzen `var`: nicht mehr verwenden.

Datentypen

TYP	BEISPIEL
`number`	`42`, `3.5` (ganze und Kommazahlen)
`string`	`"Ali"` oder `'Ali'`
`boolean`	`true`, `false`
`undefined`	Variable ohne Wert
`null`	absichtlich „nichts“
Array	`[3, 8, 12]`
Objekt	`{ name: "Ali", alter: 17 }`

`typeof punkte` → `"number"`

Text — string

```
let vorname = "Ali";
let gruss = "Hallo " + vorname + "!";
let gruss2 = `Hallo ${vorname}!`;  // gleich
```

`gruss.length`	Anzahl Zeichen
`gruss.toUpperCase()`	alles groß
`gruss.includes("Ali")`	enthält? → `true`
`gruss.trim()`	Leerzeichen außen weg

In **Backticks** `` `…` `` setzt `${ }` den Wert ein (Platzhalter).

Zahl oder Text?

```
"5" + 3          // "53"  Text!
Number("5") + 3  // 8
Number("abc")    // NaN: keine Zahl
(7.456).toFixed(2)  // "7.46"
```

Eingaben aus Formularen sind **immer Text**. Vor dem Rechnen mit `Number(…)` umwandeln.

Verzweigung — if · else

```
if (punkte >= 13) {
  console.log("sehr gut");
} else if (punkte >= 5) {
  console.log("bestanden");
} else {
  console.log("nicht bestanden");
}
```

`===` `!==`	gleich, ungleich (Wert **und** Typ)
`<` `>` `<=` `>=`	kleiner, größer …
`&&` `||` `!`	und, oder, nicht

`==` wandelt um: `"5" == 5` ist `true`. Darum immer `===`.

Fallauswahl — switch

```
switch (tag) {
  case "Sa":
  case "So":
    console.log("Wochenende");
    break;
  default:
    console.log("Schule");
}
```

Schleifen

ZÄHLSCHLEIFE

```
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
```

KOPFGESTEUERT

```
let summe = 0;
while (summe < 100) {
  summe = summe + 7;
}
```

FUSSGESTEUERT

```
let wurf;
do {
  wurf = Math.floor(Math.random() * 6) + 1;
} while (wurf !== 6);
```

Funktionen

```
function berechneNote(punkte) {
  let note = (17 - punkte) / 3;
  return Math.round(note * 10) / 10;
}

let meineNote = berechneNote(11);  // 2
```

Kein Rückgabetyp, keine Parametertypen. Ohne `return` liefert die Funktion `undefined`.

ZUM LESEN: PFEILFUNKTION

```
const verdoppeln = (zahl) => zahl * 2;
```

Kurzform, kommt oft im Internet vor. Wir schreiben `function`.

Arrays

```
let noten = [12, 8, 15];
noten.push(10);          // hinten anhängen
console.log(noten[0]);   // 12
console.log(noten.length); // 4

for (let i = 0; i < noten.length; i++) {
  console.log(noten[i]);
}
for (let note of noten) {   // jedes Element
  console.log(note);
}
```

`noten.pop()`	letztes entfernen
`noten.includes(8)`	enthalten? → `true`
`noten.indexOf(15)`	Stelle → `2`, sonst `-1`
`noten.join(", ")`	`"12, 8, 15, 10"`

Objekte

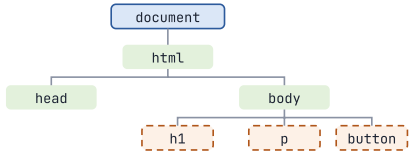
```
let schueler = {
  vorname: "Ali",
  punkte: 12
};
console.log(schueler.vorname);  // Ali
schueler.punkte = 13;
```

Spickzettel JavaScript · Webseiten mit Leben

DOM · Ändern · Ereignisse · Formulare · Zeit · Hilfe

Holger Sebastião · www.sebastiao.org
Version 1.0 · 2026-10-25

Das DOM — die Seite als Baum



Der Browser macht aus dem HTML einen Baum: das **DOM**. JavaScript sucht darin Elemente und ändert sie, die Seite zeigt es sofort.

Elemente finden

```
<h1 id="titel">Notenrechner</h1>
<p class="hinweis">…</p>
```

```
let titel =
  document.getElementById("titel");
let hinweis =
  document.querySelector(".hinweis");
```

`getElementById("titel")`	das Element mit dieser id, **ohne** `#`
`querySelector(".hinweis")`	das **erste** passende, mit CSS-Selektor

Findet es nichts, kommt `null` zurück. Danach Fehler `Cannot read properties of null`: id prüfen und `defer` setzen.

Im Browser speichern

```
localStorage.setItem("name", "Ali");
let name = localStorage.getItem("name");
localStorage.removeItem("name");
```

Bleibt nach dem Neuladen erhalten, nur auf diesem Rechner und nur als Text. Nichts Geheimes speichern.

Neue Elemente

```
let liste =
  document.getElementById("liste");
let eintrag = document.createElement("li");
eintrag.textContent = "Milch";
liste.append(eintrag);    // hinten anfügen
eintrag.remove();         // wieder löschen
```

Inhalt ändern

```
titel.textContent = "Neuer Titel";
hinweis.innerHTML = "Jetzt <b>fett</b>";
```

`textContent`	nur Text, Tags werden angezeigt
`innerHTML`	Text mit HTML-Tags
`value`	Inhalt eines Eingabefelds

Eingaben von Nutzern nie mit `innerHTML` ausgeben, sondern mit `textContent`: Sonst kann jemand fremdes HTML und Skripte einschleusen.

Aussehen ändern

```
/* style.css */
.fehler {
  color: #b3261e;
  font-weight: bold;
}
```

```
hinweis.classList.add("fehler");
hinweis.classList.remove("fehler");
// an, wenn aus; aus, wenn an:
hinweis.classList.toggle("fehler");
```

Besser so: Das Aussehen steht im CSS, JavaScript schaltet nur die Klasse.

```
titel.style.color = "red";   // geht auch
titel.style.backgroundColor = "yellow";
```

Bei `style` wird aus `background-color` → `backgroundColor`.

Viele Elemente bearbeiten

```
let links =
  document.querySelectorAll("nav a");
for (let link of links) {
  link.classList.add("menue");
}
```

`querySelectorAll` liefert eine Liste. Mit `for … of` kommt jedes Element einzeln dran.

Auf Klicks reagieren — Ereignisse

```
<button id="knopf">Klick mich</button>
<p id="ausgabe"></p>
```

```
let knopf =
  document.getElementById("knopf");
let ausgabe =
  document.getElementById("ausgabe");
let anzahl = 0;

function zaehlen() {
  anzahl = anzahl + 1;
  ausgabe.textContent = anzahl + " Klicks";
}

knopf.addEventListener("click", zaehlen);
```

`zaehlen` **ohne** Klammern übergeben. Mit `zaehlen()` läuft die Funktion sofort einmal und nie beim Klick.

EREIGNIS	PASSIERT, WENN …
`"click"`	geklickt wird
`"input"`	im Feld getippt wird
`"change"`	Auswahl geändert ist
`"submit"`	Formular abgeschickt wird
`"keydown"`	eine Taste gedrückt wird
`"mouseover"`	die Maus darüber fährt

ZUM LESEN: IM HTML

```
<button onclick="zaehlen()">Klick</button>
```

Geht auch, mischt aber HTML und JavaScript. Wir trennen: `addEventListener`.

Live mitrechnen — input

```
let feld = document.getElementById("punkte");

function anzeigen() {
  let punkte = Number(feld.value);
  ausgabe.textContent =
    "Note: " + berechneNote(punkte);
}

feld.addEventListener("input", anzeigen);
```

Bei jedem Tastendruck neu berechnet, ganz ohne Knopf.

Formular auswerten — Notenrechner

```
<form id="rechner">
  <label for="punkte">Punkte</label>
  <input type="number" id="punkte"
         min="0" max="15" required>
  <button type="submit">Berechnen</button>
</form>
<p id="ergebnis"></p>
```

```
let rechner =
  document.getElementById("rechner");

function auswerten(ereignis) {
  ereignis.preventDefault();
  let feld =
    document.getElementById("punkte");
  let punkte = Number(feld.value);
  let note = berechneNote(punkte);
  document.getElementById("ergebnis")
    .textContent = "Note: " + note;
}

rechner.addEventListener("submit",
                         auswerten);
```

`preventDefault()` verhindert, dass der Browser das Formular abschickt und die Seite neu lädt. So bleiben Feld und Ergebnis stehen.

Zeitsteuerung

```
// einmal nach 2000 ms = 2 Sekunden
setTimeout(zeigeHinweis, 2000);
// jede Sekunde, bis zum Stoppen
let uhr = setInterval(tick, 1000);
clearInterval(uhr);
```

Auch hier: Funktionsname **ohne** Klammern.

Typische Fehler

- `… is not defined`: Tippfehler im Namen oder Groß/klein.
- `"5" + 3` ergibt `"53"`: `Number()` vergessen.
- `=` statt `===` im `if`: weist zu statt zu vergleichen.
- Skript im head ohne `defer`: Elemente sind `null`.
- Nichts passiert: Konsole (F12) auf rote Meldungen prüfen. Werte mit `console.log` ausgeben.
- Haltepunkt: Reiter **Quellen** → Zeilennummer anklicken. Das Programm stoppt dort.