Frontend Cheat-Sheet

Inhalt

HTML

Grundstruktur:

```
<!DOCTYPE html>
```

```
<html>
 <head>
  <title>Seite</title>
 </head>
 <body>
 </body>
</html>
```

Semantische HTML-Tags

Struktur & Dokument

- <html> → Wurzelelement
- <head> → Metadaten
- <body> → Dokumenteninhalt

Dokumentgliederung

- <header> → Kopfbereich (z. B. Logo, Navigation)
- <nav> → Navigation
- <main> → Hauptinhalt
- <section> → Abschnitt eines Dokuments
- <article> → unabhängiger Artikel / Beitrag
- <aside> → Randinfo, Seitenleiste
- <footer> → Fußbereich

Text & Inhalt

- <h1> … <h6> → Überschriften
- <p> → Absatz
- <blockquote> → Zitatblock
- <q> → kurzes Zitat inline
- <abbr> → Abkürzung
- <cite> → Quelle / Werkangabe
- <code> → Quelltext

- <pre> → vorformatierter Text
- <mark> → Hervorhebung (markiert)
- → Betonung (kursiv)
- → starke Betonung (fett)
- <time> → Uhrzeit / Datum

Listen

- → ungeordnete Liste
- → geordnete Liste
- → Listenelement
- <dl> → Definitionsliste
- <dt> → Begriff
- <dd> → Definition

Medien

- <figure> → Abbildung inkl. Caption
- <figcaption> → Bildunterschrift
- <audio> → Audioinhalt
- <video> → Video
- <source> → Quelle für <audio>/<video>
- <track> → Untertitel/Spuren
- <picture> → responsives Bild
- <img> → Bild

Tabellen

- <table> → Tabelle
- <caption> → Tabellenüberschrift
- <thead> → Tabellenkopf
- <tbody> → Tabellenkörper
- <tfoot> → Tabellenfuß
- <tr> → Tabellenzeile

- <th> → Tabellenüberschrift
- <td> → Tabellenzelle

Formulare

- <form> → Formular
- <label> → Beschriftung
- <input> → Eingabe (Text, Checkbox, etc.)
- <textarea> → mehrzeiliger Text
- <button> → Button
- <select> → Auswahlfeld
- <option> → Option im Select
- <fieldset> → Gruppierung
- <legend> → Beschriftung für Fieldset

Selbstschliessende HTML-Tags

Selbstschliessende Tags sehen so aus**: < XYZ/>**

<area> → Bereich in einer Image-Map

<base> → Basis-URL für relative Links

 → Zeilenumbruch

<col> → Spalteneigenschaften in Tabellen

<embed> → Einbetten von externen Inhalten (z. B. Plugins)

<hr> → Horizontale Linie (Trenner)

<img> → Bild

<input> → Eingabefeld

<link> → Verknüpfung (z. B. CSS-Datei)

<meta> → Metainformationen im Head

<source> → Medienquelle für <audio>/<video>

<track> → Untertitel-/Spurdatei für Medien

<wbr> → Weicher Zeilenumbruch (optional break)

HTML-Attribute

Globale HTML-Attribute

- id → eindeutige Kennung
- class → Klassenzuordnung
- style → Inline-CSS
- title → Tooltip / Zusatzinfo
- lang → Sprache (<html lang="de">)
- dir → Schreibrichtung (ltr, rtl)
- hidden → Element verstecken
- tabindex → Tab-Reihenfolge
- contenteditable → Inhalt editierbar
- draggable → Drag & Drop möglich (true/false)
- accesskey → Tastaturkürzel

Spezifische Attribute

- **Formulare:**
  - type, name, value, placeholder, required, disabled, readonly, checked, min, max, step, pattern
- **Bilder/Medien:**
  - src, alt, width, height, controls, autoplay, loop, muted, poster
- **Links:**
  - href, target, rel, download

CSS

CSS Einbindungen

- **Inline**:

  Text

- **Intern (im Head)**:

  <style>

  p { color: red; }

  </style>

- **Extern (beste Variante)**:

  <link rel="stylesheet" href="style.css" />

Selektoren

Element: p {}

ID: #header {}

Klasse: .nav {}

Attribut: input[type="text"] {}

Kombinatoren:

- Nachfahre (tief): div p {}
- Kind (nur direkte ebene): div > p {}
- Direktes Geschwister (direkt nach h1): h1 + p {}
- Alle Geschwister (jeder auf der Ebene von h1): h1 ~ p {}

JavaScript

Ausführung & Einbindung

Wird im Browser ausgeführt (JS-Engine).

Einbinden

<!-- Beste Variante: DOM sicher bereit -->

<script src="app.js" defer></script>

<!-- Alternativen -->

<script src="app.js"></script> <!-- am Ende von <body> -->

<script>document.addEventListener("DOMContentLoaded", () => {}) </script>

- <script> ist **nicht** selbstschliessend.

Sprache Basics

Variablen: let (veränderbar), const (konstant), **kein** var im modernen Code.

Datentypen: string, number, boolean, null, undefined, bigint, symbol, object.

Operatoren: ===/!== (streng), &&, ||, ?? (nullish), ?. (optional chaining), ... (spread/rest).

Template-Strings: `Hallo ${name}` .

Truthy/Falsy: false, 0, "", null, undefined, NaN sind falsy.

DOM – Selektion

Grundprinzipien

- **DOM = Document Object Model** → gesamte HTML-Struktur als Baum.
- Selektion = Zugriff auf Knoten (Elemente) über verschiedene Methoden.

Klassische Selektoren

By ID

document.getElementById('header')

- Gibt **ein einzelnes Element** zurück.
- IDs sollten eindeutig sein.

By Class

document.getElementsByClassName('rot')

- Gibt **HTMLCollection** zurück (live, aktualisiert sich).

By Tag

document.getElementsByTagName('li')

- Ebenfalls **HTMLCollection** (live).

Moderne CSS-Selektoren

querySelector

document.querySelector('ul li.rot')

- Gibt **erstes passendes Element** zurück.

querySelectorAll

document.querySelectorAll('ul li.rot')

- Gibt **NodeList** zurück (statisch, nicht live).
- Kann mit .forEach() durchlaufen werden.

Kombinatoren (CSS-Logik in JS nutzbar)

Nachfahre (A B):

document.querySelectorAll('div p')

→ alle <p> in (egal wie tief).

Kind (A > B):

document.querySelectorAll('div > p')

→ nur direkte Kinder.

Direktes Geschwister (A + B):

document.querySelectorAll('h1 + p')

→ erstes <p> direkt nach <h1>.

Alle Geschwister (A ~ B):

document.querySelectorAll('h1 ~ p')

→ alle <p> nach <h1> auf derselben Ebene.

Pseudoklassen

Erstes Kind:

- document.querySelectorAll('ul li:first-child')

→ gibt **alle ersten ** jedes <ul> zurück

N-tes Kind:

- document.querySelectorAll('ul li:nth-child(2)')

→ gibt **alle zweiten ** jedes <ul> zurück

Letztes Kind:

- document.querySelectorAll('ul li:last-child')

→ gibt **alle letzten ** jedes <ul> zurück

Negation:

- document.querySelectorAll('li:not([class])')

→ gibt **alle <li> zurück, die kein class-Attribut** besitzen

Letztes Element des Typs:

- document.querySelectorAll('ul li:last-of-type')

→ gibt das **letzte <li> jedes ** zurück

DOM – Manipulation

Inhalte ändern

Text ändern:

- el.textContent = "Neuer Text"; // nur reiner Text
- el.innerText = "Neuer Text"; // berücksichtigt CSS (z.B. hidden)

HTML ändern:

el.innerHTML = "Fett"; // HTML wird interpretiert

Attribute ändern

Setzen & Lesen:

- el.setAttribute("id", "neu");
- el.getAttribute("id"); // "neu"
- el.removeAttribute("id");

Direkt über Properties:

- img.src = "bild.png";
- input.value = "Hallo";

Klassen manipulieren

- el.classList.add("rot");
- el.classList.remove("rot");
- el.classList.toggle("active");
- el.classList.contains("rot"); // true/false

Elemente erzeugen & einfügen

Neu erstellen:

- const li = document.createElement("li");
- li.textContent = "Spargel";

Einfügen:

- ul.appendChild(li); // am Ende einfügen
- ul.prepend(li); // am Anfang
- el.before(li); // direkt davor
- el.after(li); // direkt danach

Mehrere gleichzeitig:

- ul.append(li1, li2, "Text");

Elemente entfernen & verstecken

Komplett entfernen:

- el.remove();

Verstecken:

- el.style.display = "none"; // unsichtbar, Platz frei
- el.style.visibility = "hidden"; // unsichtbar, Platz bleibt
- el.hidden = true; // entspricht hidden-Attribut

Styles ändern

el.style.color = "red";

el.style.backgroundColor = "yellow";

- Für mehrere Styles besser CSS-Klasse verwenden:

el.classList.add("highlight");

Traversieren + Kombination

Beispiel: Alle roten Gemüse entfernen

```
document.querySelectorAll(".rot").forEach(el => el.remove());
```

Beispiel: Kürbis orange färben

```
document.querySelectorAll("li").forEach(li=>{
 if (li.textContent === "Kürbis") li.classList.add("orange");
});
```

Beispiel: Spargel zu Frühlingsgemüse hinzufügen

```
const neu = document.createElement("li");
neu.textContent = "Spargel";
document.querySelector("#fruehling + ul").appendChild(neu);
```

DOM – Events

Grundidee

- Events = **Aktionen im Browser** (z. B. Klick, Tastendruck, Laden).
- Mit JS können wir auf Events reagieren.

Events registrieren

Direkt im HTML (nicht empfohlen)

```
<button onclick="alert('Hallo')">Klick mich</button>
```

Per JS (empfohlen)

```
btn.addEventListener("click", () => {
  alert("geklickt!");
});
```

- addEventListener(event, handler, options)
- removeEventListener("click", handler)

Wichtige Events

- **Maus:** click, dblclick, mouseover, mouseout, mousemove
- **Tastatur:** keydown, keyup, keypress
- **Formulare:** submit, input, change, focus, blur
- **Fenster/Seite:** load, DOMContentLoaded, resize, scroll

Event-Objekt

Jeder Event-Handler bekommt ein Event-Objekt:

```
btn.addEventListener("click", (e) => {
  console.log("Geklicktes Element:", e.target);
```

```
  console.log("Position im Viewport:", e.clientX, e.clientY);
  console.log("Position im gesamten Dokument:", e.pageX, e.pageY);
});
```

UI – Komponente