Web Engineering 1

[1 Hyper Text Markup Language (HTML) 1](#_Toc157977518)

[2 Cascading Style Sheet (CSS) 1](#_Toc157977519)

[3 Usability & Barrierefreiheit 2](#_Toc157977520)

[4 JavaScript 3](#_Toc157977521)

[5 Asynchronous JavaScript and XML (AJAX) 5](#_Toc157977522)

[6 Anwendungsentwicklung 5](#_Toc157977523)

# Hyper Text Markup Language (HTML)

Beschreibung des Inhalts und der Struktur, HT=Verlinkung von Docs möglich, M=Semantik, L=Syntax und Bedeutung

## Bestandteile eines HTML-Elements

**Start-Tag**: <name> & **End-Tag**: </name>, **Inhalt** zwischen Start- und End-Tag, **Attribute** im Start-Tag <name attname="value">, **Boolsche Attribute** im Start-Tag <name attname>, **Leere Elemente** haben nur Start-Tag und Attribute, **Non-Void Elemente** können in ihrem Inhalt weitere Elemente haben, **Void Elemente** nicht

## Globale Attribute

id="": Unique ID für ein Element, onclick="": Aufzurufende Funktion beim Anklicken, title="": liefert zusätzliche Informationen zum Element, lang="": Sprache des Elements

## Grundgerüst (Pflichtteile fett)

<!**DOCTYPE** html><**html** lang="de">

<**head**>

<meta charset="utf-8"><meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Titel</title>

<script src="scripts/main.js"></script>

<link rel="stylesheet" href="styles/main.css">

<link rel="icon" href="icon.png">

<meta name="author" content="bla">

</head>

<**body**></body></html>

## Body (Sichtbarer Teil)

Element-Spezifikation besteht immer aus folgenden Punkten:

* **Content Categories** (Flow, Interactive, Metadata, Heading, Sectioning, Phrasing, Embedded)
* In welchem **Context** es verwendet werden kann
* **Content Model**: in welchen Elementen es vorkommen darf und welche Elemente vorkommen dürfen in diesem
* Wann End-Tag weggelassen werden darf (**Tag Omission**)
* **Content Attributes** (erlaubte Attribute im Start-Tag)

**Syntax**: Regeln und Struktur der Sprache; **Semantik**: Bedeutung der Elemente und Attribute; <div class="li">Milk</div> ist zwar syntaktisch korrekt, semantisch besser ist aber <li>Milk</li> (verständlicher, leichtere Entwicklung, mobile-freundlich, responsiver, kleinere Filegrösse, SEO, zugänglicher für Screenreader/Crawler)

**Für Überschriften**: <h1>-<h6> benutzen, keine Ebenen überspringen und nur einmal <h1> pro Seite

A screenshot of a website

Description automatically generated**article**: in sich geschlossener Abschnitt, kann unabhängig verteilt werden

**section**: article besteht aus mehreren sections, macht allein keinen Sinn

A diagram of a website

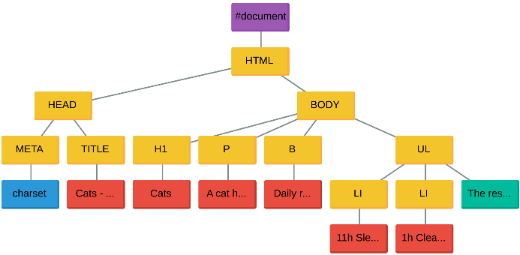
Description automatically generated

## Document Object Model (DOM)

API, welche die Manipulation von Struktur, Inhalt und Aussehen von Webdokumenten erlaubt, sprachunabhängig

**Document**: semantisch ausgeschriebene Webdokumente

**Object Model**: Repräsentation Webdocs als Datenstruktur



* **Dokument-Knoten**: Wurzel
* **Element-Knoten**: Elemente (p, h1, ...)
* **Text-Knoten**: Texte (A cat has...)
* **Kommentar-Knoten**: Kommentare (<!-- ... -->)
* **Attribut-Knoten**: Attribute des Dokuments. (charset)

# Cascading Style Sheet (CSS)

Styling, Layout, Animationen

## Anwendung

* **Inline:** <span style="background: green; color: pink">Hello World</span>
* **Style-Element:** <head><style> span { background: green; color: pink; } </style></head>
* **CSS-File:** siehe [2.3 Grundgerüst](#_Grundgerüst_(Pflichtteile_fett)) im head, Syntax:  
  A screenshot of a computer

  Description automatically generated

## Selektoren

\* (Universalselektor, betrifft alle Elemente), tagname (Typ-Selektoren, z.B. p), .classname (bestimmte Klasse selektieren), #id (bestimmte ID selektieren), [attribut] (Elemente mit bestimmtem Attribut)

### Listen vs. Verbindungen

* p, .main (betrifft Texte **und** Elemente mit Klasse main)
* a, [href="…"] (betrifft Links und Elemente mit href="…")
* p.main (betrifft Texte mit Klasse main)
* a[href="…"] (betrifft Links mit href="…")

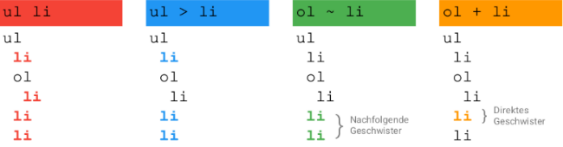
### Pseudoelement-Selektor

::first-letter (erster Buchstaben eines Elements), ::marker (betrifft Aufzählungszeichen eines Elements), ::selection (betrifft Maus-Auswahl eines Elements), ::before (Inhalte wie Icons am Anfang eines El. anfügen), ::after (Inhalte wie Icons am Ende eines El. anfügen), Beispiel: a::before { content: "-"; }

### Pseudoklassen-Selektor

:visited (Elemente mit bereits besuchten Links), :invalid (Elemente mit invalidem Inhalt), :hover (Elemente, auf die die Maus aktuell zeigt), :nth-child(1|odd|even) (Childs basierend auf Index), :first-child, :last-child, Beispiel: article p:first-child::first-line {}

### Kombinatoren



### Kaskade

**Priorität** absteigend: 1. Browser-Einstellungen Benutzer, 2. Styles Website-Autor, 3. Browser Default-Werte

1. Innerh. der 3 Punkte: nach Spezifizität, 2. Gleiche Spezifizität: später deklarierte Eigensch. gewinnt, 3. !important gewinnt immer

**Spezifizität** (Auflösung von Konflikten zwischen Regeln, die selbes Element betreffen) absteigend: 1. Inline (a++), 2. ID-Selektoren (b++), 3. Klassen-Selektoren/Pseudoklassen/Attribute (c++), 4. Typ-Selektoren/Pseudoelemente (d++)

A white rectangular box with black text

Description automatically generated

## Box-Model

A screenshot of a computer

Description automatically generated

## Display-Property (Default display: block)

A green and yellow bar graph

Description automatically generated#text1 { background-color: red;

display: inline; }

#text2 { background-color: green;

display: inline-block;

height: 140px; }

#text3 { background-color: yellow;

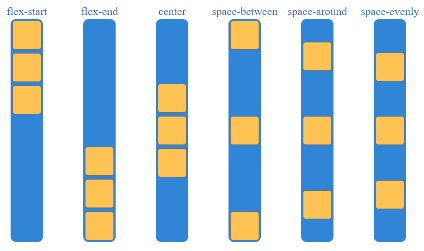
border: 1px black solid; }

## Einheiten und Werte

* **px** (Basis-Einheit Browser, entspricht virtuellem Pixel)
* **em** (1em = gleiche Grösse wie Parentelement)
* **rem** (1rem = gleiche Grösse wie html-Element)

## Flexbox (display: flex)

flex-direction: row &  
justify-content: ⬅️

flex-direction: column & justify-content: ⬅️

Reihenfolge gleich wie bei row, am Schluss ganz rechts noch space-evenly

# Usability & Barrierefreiheit

Barrierefreiheit Anforderungen: **Wahrnehmbar**. Informationen und Komponenten des UI müssen für alle Benutzer wahrnehmbar sein. **Bedienbar**. Komponenten des User Interface und die Navigation müssen für alle Benutzer bedienbar sein. **Verständlich**. Informationen und die Bedienung des User Interface müssen verständlich sein. **Robust**. Inhalte müssen so robust sein, dass sie von einer Vielzahl von User Agents, einschliesslich assistiver Technologien, interpretiert werden können.

**alt-Attribut** bei img-Tags verpflichtend, Aussage der Website soll nicht verändert werden, wenn Bilder durch Alt-Texte ersetzt werden, wenn dekorativ alt=""

Jakob Nielsen 10 Prinzipien für User Interaktion:

1. **Sichtbarkeit des Systemstatus** (Interaktionsmöglichkeiten kommunizieren, Mehrfachausführung vermeiden, zeigen, dass Anfrage noch dauert)  
   A purple rectangle with white text

   Description automatically generated

let suspense = false; let waitString = "";  
function updateView () {  
 if (suspense) { nextBtnElement.disabled = true;  
 waitString += "."  
 counterOutpEl.value='Running'+waitString;  
 setTimeout(updateView, 200);  
 } else { nextBtnElement.disabled = false;  
 counterOutpEl.value= localCnter.counter; }}

1. **Übereinstimmung zwischen System und realer Welt** (Bekannte Begriffe verwenden, Konventionen respektieren, Affordances, Constraints (Zwang), logisches und natürliches Mapping, Metaphern)
2. **Freiheit und Kontrolle der Nutzer:in**
3. **Konsistenz und Standards** (Sichern=speichern?, schliessen=weiter?, Links, Buttons klickbar)
4. **Fehlervermeidung** (einfaches Undo, Touchzonen mind. 1cm2, Abreisedatum kann nicht vor Anreise liegen)
5. **Wiedererkennen statt erinnern**
6. **Flexibilität und effiziente Nutzung**
7. **Ästhetik und minimalistische Gestaltung**
8. **Hilfe beim Erkennen und Beheben von Fehlern**
9. **Hilfe und Dokumentation**

**Change Blindness**: Veränderungen abseits des Aufmerksamkeitsfokus werden eher übersehen, es braucht ein visuelles Signal, um Aufmerksamkeit darauf zu lenken

**Sichtfelder**: **Scharfer Fleck** (Fovea): 1-2°, **Zentraler Bereich** (Para-Fovea) mit Farben & Details: 2-5°, **Peripherer Bereich**, wo nur Bewegungen und Veränderungen wahrgenommen werden: 6-220°

Web-User lesen nicht, sie scannen, z.B. im F-Muster, Augen müssen durch Grafiken, Textgrössen, Farben geführt werden

* **Scannen**: Headings, Schlüsselwörter/-bilder, Links/Suchhilfen
* **Skimmen**: Kurze Absätze, Frontloading, Listen/Tabellen
* **Lesen**: Längere Seiten mit Fliesstext, Ausdruck-Format

**Affordance** stärken durch: Konventionen befolgen, Konsistenz, Beschriftung, Vorsicht bei Metaphern

**Gibson**: natürlich wahrgenommene Nutzungsmöglichkeit eines Geräts, **Norman**: Wahrgenommene Affordance (Touchscreen=Tap/ Swipe, Maus=Clicken/Draggen, Schalter=On/Off, Schieber)

A screenshot of a computer screen

Description automatically generated

**Visuelles Design**: bessere Lesbarkeit, lenkt Aufmerksamkeit, schafft Vertrauen/Glaubwürdigkeit, stärkt Marke

**wenige Farben** (<6, besser 2+1), **Standards/Normen** (Styleguide), **kulturelle Aspekte** (grün nicht immer gut), ausreichender **Kontrast**, **Farbfehlsichtigkeit** (z.B. Farbe und Symbol zur Unterscheidung) und **Seheinschränkungen** berücksichtigen (Bild-Kontrast mind. 4.5:1, grosser Text mind. 24px oder 18.66px fett, Kontrast für Text mind. 3:1)

**Gestaltgesetze**: Grösse, visuelle Hierarchie, Ausgewogenheit, Kontrast

A diagram of a diagram of a number of dots

Description automatically generated with medium confidence

# JavaScript

Interactives Verhalten, Event-Handling, Eigenschaften:

Implementierung von **ECMAScript 2023,** Üblicherweise im HTML eingebunden und vom **Browser** ausgeführt, **Node.js** kann auch ausführen/debuggen, Node.js kann auch **Webserver**, nutzt Chrome V8 Engine, **Polymorphie, Dynamic** (Objekte verändern & Methoden überschreiben), **Dynamically typed** (Variablen können Type ändern, keine Typendeklarierung notwendig, typeof() möglich), **Funktional & Objektorientiert, Fails silently** (oft keine Exception und läuft weiter), Wird als **Code** deployed (compile erst im Browser)

## Variablen

**Primitive Typen** (compare by value, immutable): string, number, boolean, undefined, symbol, bigint, NaN, null (Bug: typeof null = object)

**Objekte** (compare by reference, mutable): alles andere, z.B. Plain objects, Arrays, Regexes, Funktionen

Jeder Wert kann in ein Boolean umgewandelt werden

**False**: false, 0, "" (leerer String), null, undefined, NaN

**True**: alles andere, "0" oder "false" (nicht leerer String), [], {}

## Zahlen

Alle Zahlen sind **floats**, Engines probieren Abbildung auf int:

console.log(0.3333333333333333 \* 3 == 1); //true

let x = 1/3; console.log(x + x + x); //1

console.log(Number.isInteger(x + x + x)); //true

console.log(Number.isInteger(0.3)); //false

console.log(0.1 + 0.2); //0.3000000000000004

console.log(99999999999999999); //100000000000000000, ausserh. gültigem Bereich

Number.isSafeInteger(99999999999999999); //false

**Not a Number (NaN)**: Error-Wert, Type number, NaN == NaN immer false, isNaN() zum Prüfen

let div0 = 0 / 0; console.log(div0); //NaN

console.log(typeof(div0)); //number

console.log(parseInt("abc")); //NaN

console.log(div0 == NaN); //false

console.log(isNaN(div0)); //true

console.log(3 / 0); //Infinity

console.log(-Math.pow(2,10000)); //-Infinity

Jeder Wert kann in Zahl umgewandelt werden:

console.log(+(true)); //1

console.log(+(false)); //0

console.log(+("1ab")); //NaN

console.log(+("123")); //123

console.log(+([])); //0

console.log(parseInt("1.2ab")); //1

console.log(parseFloat("1.2ab")); //1.2

console.log(parseInt("abc")); //NaN

## Strings

Mit "Text" oder 'Text', Escape mit "\"

String+Value oder Value+String ergibt String

**Properties & Methoden**: length, slice(), trim(), includes(), indexOf()

const name = "Joe", hobby = "Hike", a = 4, b = 5;

console.log(`Name: ${name} //Name: Joe

Hobby: ${hobby}`); //Hobby: Hike

console.log(`${a}+${b}=${a + b}`); //4+5=9

## Comparison

== Abstract Equality Comparison Algorithm

console.log([] ==(=) false); //true(false)

console.log(null ==(=) false); //false(false)

console.log(0 ==(=) "0"); //true(false)

console.log(null ==(=) undefined); //true(false)

console.log([1,2] ==(=) "1,2"); //true(false)

console.log([] ==(=) ]); //true(false)

## Null/undefined

console.log(typeof x); //undefined

x = null; console.log(x, typeof x); //null object

console.log(users.get(0));

//{name: 'Lea', bday: '19.05.1985'}

c.log(users.update(0, {bday: "19.05.1986"})); //{name: 'Lea', bday: '19.05.1986'}

c.log(users.update(0, {name: undefined, bday: null})); //{name: 'Lea', bday: null}

## Arrays

const arr = [ 'a', 'b', 'c' ];

arr[0] = 'x'; arr.push("d"); //['x','b','c','d']

arr.forEach((e, i) => console.log(i +":"+e));

console.log(numberArr.filter(elem => elem > 5));

arr.sort((a,b)=>a-b).map(e=>`<li>e</li>`).join('');

for(let i=0; i<arr.length; ++i) {

console.log("for",arr[i]); }

for(const x in arr) {

console.log("for in", x + ":" + arr[x]); }

for(const y of arr) {

console.log("for of", y); }

**Weitere Methoden**: concat, find, findIndex, forEach, reduce, map, some, every, toSorted, toReversed, toSpliced

## Simple Object

const person = {

name: "Michael",

hallo: function() {

return "Hallo " + this.name; }};

person.name = "Bob"; person.hallo();

person[name] = "Bob"; //gleich wie 1 oben

Jedes Objekt ist eine **HashTable**, Index Wert in [] wird immer zu String umgewandelt, **Methoden**: entries, keys, values

## Funktionen

**Möglich**: Speichern in Variable, als Parameter übergeben, von Funktionen returnen, kein Overloading, new-callen

Alle **Parameter** werden in arguments[] abgelegt, Übergabe von mehr/weniger Params als definiert möglich, mit *…name* den letzten Parameter als "Rest-Parameter" definieren

haben **Properties**: .name beinhaltet Namen der Funktion (anonyme Methoden haben keinen, wird für Stacktrace genutzt), .length beinhaltet Anzahl Parameter

function hallo() { //Funkt. definieren

console.log("Hallo"); }

var hallo2 = function() { //Variablenzuweisung

console.log("Hallo2"); };

const foo = hallo; //Variablenzuweisung, foo();

const foo2 = (value) => console.log(value) //arrow function, sinnvoll für kleine Fns, foo2('x');

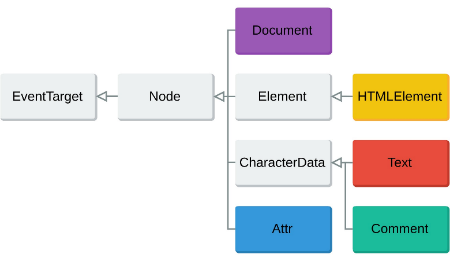
function minus(a, b) {

return a - b; }

function calc(fn, a ,b) {

console.log(fn(a,b)); } //z.B. calc(minus,3,4);

## Document Object Model

**EventTarget** Interface hat 3 **Methoden**: addEventListener(type, fn[, options]), removeEventListener(type, fn), dispatchEvent(event)

**Options**: capture (in Capture-Phase reagieren), once (nach erster Aktivierung entfernen), passive (für Performance-Optimierung)

**Node** Interface hat u.a. diese **Methoden**: cloneNode, appendChild, removeChild und **Eigenschaften**: childNodes, firstChild, lastChild, nextSibling, parentNode, nodeType

**Element** Interface hat u.a. diese **Methoden**: querySelector, querySelectorAll, insertAdjacentHTML, append, prepend, remove   
und **Eigenschaften**: id, classList, attributes, innerHTML, children

**HTMLElement** Eigenschaften: dataset, style, hidden, innerText

Dokument hat 3 **Ladezustände** (document.readyState):

**Loading**: Doc lädt noch, **Interactive**: Doc geladen, Ressourcen (Scripts, Bilder, Videos, CSS) werden noch geladen, **Complete**: Doc und Ressourcen geladen

### Selectors

document.querySelector('#list-container'); //erstes passendes Element

const navItems = document.querySelectorAll('.nav-item');for (let i = 0; i < navItems.length; i++) {

console.log("for", navItems[i]); }

for (const item of navItems) {

console.log("for of", item); }

[…navItems].filter((el, i) => i % 2 === 0)

.forEach(el => el.style.background = "green");

getElementBy…: Id, ClassName, Name, TagName, TagNameNS

### Manipulation

const newEl = document.createElement('div');

newEl.innerText = 'Hello';

document.querySelector("#id").appendChild(newEl);

document.querySelector("#id").innerHTML = '<div>Changed</div>';

**CreateElement:** Schneller bei kleinen Änderungen, DOM-Referenzen bleiben erhalten, Eventhandler bleiben erhalten, **innerHTML:** wahrscheinlich schneller, lesbarer

**innerText**: liefert gerenderten Text, berücksichtigt CSS

**textContent**: kompletter Text mit Tags, berücksichtigt CSS nicht

Beim Schreiben sind innerText und textContent gleichwertig

Viele HTML-Attribute haben 1:1 Mappings zu Properties, einige Attribute haben aber keine Properties (z.B. aria-\*), einige Properties haben keine Attribute, z.B. textContent, setAttribute setzt beliebige Werte beliebiger Attribute und entsprechende Property, removeAttribute löscht Attribut

btn1.setAttribute("value", "BtnPropValue");

btn1.removeAttribute(value);

CSS-Klassen können auch mit JavaScript manipuliert werden:

<element>.className //Klassen als str, space-separated

<element>.classList //Klassen als DOMTokenList

classList Hilfsfunktionen: .add, .remove, .toggle, .contains, .replace

### Events

**Properties**: **target**: Element von dem Event ausging, **currentTarget**: Element, das Event-Listener registriert hat

**Methoden**: **preventDefault()**: Standardaktion nach Event nicht ausführen, **stopPropagation()**: weitere Propagation des Events deaktivieren in Phase 1 und 3

**Events haben 3 Phasen**: **Capture-Phase**: Event geht von Wurzel zu Blatt, jedes Element kann reagieren, muss aber nicht, **Target-Phase**: Event wird auf Target ausgelöst, **Bubble-Phase**: Event geht von Blatt zu Wurzel, jedes Element kann reagieren, muss aber nicht

## Scope (B = Block Scope, ESM = ECMAScript Module)

![A screenshot of a computer program

Description automatically generated

Jede Funktion und jedes Objekt generiert einen neuen **Scope**, innerhalb eines Scopes kann man auf dessen und globale Variablen und Parent-Scope Vars (Closure) zugreifen, <script> erzeugt keinen Scope & Fns werden im globalen Scope platziert, ESMs schon

**Global**: ohne var/let/const, über window.myGlobalVar im Browser oder global.myGlobalVar in Node.js aufrufbar, **Global Namespace Pollution** heisst zu viele Vars und Funcs im globalen Scope, überschreiben sich bei gleichem Namen

**Modul**: Node.js & ES6 Modul erzeugt pro File Scope, nicht explizit als global markierte Vars sind auf diesem Scop

**Lokal/Funktion**: mit var/let/const, nur im Scope oder darunter aufrufbar, **Block**: mit let und const

**this** ist aktueller Kontext und zeigt je nachdem auf:

* object.foo(); 🡪 this=object
* new Foo() 🡪 this = neu erstelltes Object
* unbound Funktion 🡪 this = globales Object

Kontext einer Funktion setzen (call kann bind nicht überschreiben):

foo.call({counter: 123}); //oder

foo.apply({counter: 123});

Kontext einer Funktion vorgeben (neue Fn erzeugen):

const boundFoo = foo.bind({counter: 11});

boundFoo();

## Strict Mode

Aktivieren mit 'use strict'; am Anfang eines Files für das ganze File oder am Anfang einer Funktion für die Funktion, nested Scopes auch betroffen, von strict Funktion aufgerufene Fns werden nicht umgeschaltet, sollte immer verwendet werden, ausser bei Legacy Code, Klassen und ESMs sind automatisch im Mode, Ziele:

* **Eliminiert "fails silently"** z.B. wenn Variable ohne var definiert wird oder Read-only Werte gesetzt werden
* **Keine "this" Substitution**
* **Eliminiert Probleme, welche es Compiler verunmöglichen, Code zu optimieren**
* **Security wird leicht verbesset**

## Klassen

Seit ES6 vorhanden, ähnlich wie in Java mit Methoden, Properties, Vererbung, static Methods/Properties

class Clock { //define class

#timer //private field

currentTime //field (optional)

constructor(currenTime = new Date()) {

this.currentTime = currenTime; //create a public property

this.start(); }

start() { //method

this.#timer = setTimeout(() => {

this.currentTime = new Date(); }, 1000); }

get time() { //getter

return this.currentTime; }

set time(newTime) { //setter

return this.currentTime = newTime; } }

const clock = new Clock(); //create instance

class AlarmClock extends Clock { … }

**Vererbung**:

* instanceof: prüft, ob Object Instanz von Klasse ist  
  console.log(alrmClock instanceof AlarmClock); //true
* super: Zugriff auf Parent; constructor() { super(); }  
  super.time = value;

## Module

Lade-Reihefolge kann sichergestellt werden, Apps werden immer grösser, ESMs sind die Lösung

<script src="index.js" type="module"></script>

Named Import/Export:

export function f() {}

export const one = 1

export {foo, b as bar};

import {foo, bar as b} from './module.mjs';

import \* as someModule from './module.mjs';

Default Import/Export:

export default function f() {}

export default value;

import value from './module.mjs';

## Sprachfeatures

Destructuring assignment: [a,b, ..rest]

Nullish Coalescing Operator (rechter Wert wenn links null/undefined):

console.log(null ?? 'defstr'); //defstr

console.log(0 ?? 42); //0 console.log(0 || 42); //42

Optional Chaining (undefined statt error bei nicht-existenter Object-Property):

const obj = { name: 'Alice' };

console.log(obj?.name); //undefined

console.log(obj.method?.()); //undefined

# Asynchronous JavaScript and XML (AJAX)

**JavaScript Laufzeitmodell**: **Callstack**: Funktionsaufrufe, **Message-Queue**: z.B. Events, **Heap**: Hier liegen die Objekte

Bearbeitung neuer Messages, wenn der Callstack leer ist, in JS muss man oft auf eine Antwort warten und dann einen Callback machen auf eine andere Funktion, dies kann schnell zu tief verschachteltem Code führen (Callback Hell)

## Unzuverlässiges Backend

Zufällige Fehler(404, 401, 50x), keine/leere Antworten, Lösungen:

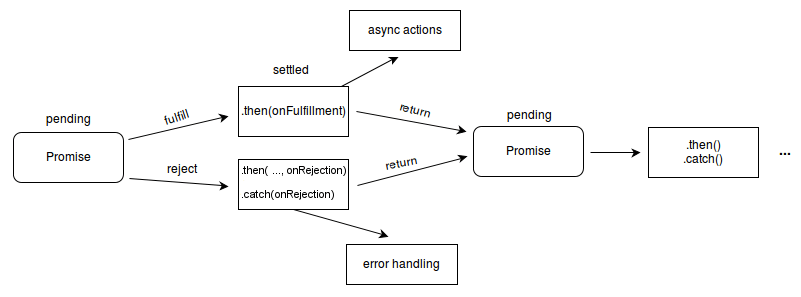
* **Timeout z.B. mit AbortController**  
  const controller = new AbortController();   
  setTimeout(() => controller.abort(), 5000);   
  fetch(url, {controller.signal}).then(response => { return response.text(); });
* **Retry**  
  function resilientGetJSON(url, refetchTries) {  
   fetch(url, {controller.signal}).then(res => {  
   if(res.ok) { return res.json(); }  
   else { return Promise.reject(); } })  
   .catch(() => { if (refetchTries > 0) {  
   resilientGetJSON(url, refetchTries-1); }};}
* **Fehlermeldung ausgeben**

## Debouncing/Throttling

In einem Suchfeld ist ein Request nach jedem Tastendruck nicht sinnvoll (viele Backend-Requests, veraltete Resultate), Lösungen:

* Kurze Zeit nach jedem Tastendruck warten und Request nur senden, wenn keine weiteres Zeichen gefolgt ist
* Bestehende Requests abbrechen und neuen senden

## Promises



const p1 = new Promise((resolve, reject) => {

resolve("Success!"); });

p1.then(

(value) => { console.log(value); },

(reason) => { console.error(reason); } );

let myResolveFn1; let myRejectFn1;

const p = new Promise(

(resolve, reject) => {

myResolveFn1 = resolve;

myRejectFn1 = reject; }); //p pending

myRejectFn1(3)

//p = Promise {<state>: "rejected", <reason>: 3}

const p2 = new Promise ((resolve) => {   
setTimeout(() => resolve(5), 1000)});

Promise.all([prom1, prom2, prom3]) //any statt all mögl.

.then((values) => { console.log(values); }, err => {});

function fetchMoviesJSON() { fetch(url)  
 .then(r => r.json()).then(r=> return r); };

Then() kann: ein weiteres Promise returnen, einen Wert returnen oder einen Fehler werfen, sollte nicht innerhalb von then verwendet werden

Catch() ist kurz für then(undefined, onRejected), sollte verwendet werden, da Fehler in einigen Browsern nicht in der Konsole erscheinen, gibt Promise zurück

## Async/Await

A message with purple text

Description automatically generated with medium confidence

async function fetchMoviesJSON() {

const response = await fetch(url);

const ret = await response.json();

return ret; }

# Anwendungsentwicklung

## Clean Code

### Namensgebung

kurz, intuitiv, aussagekräftig, aussprechbar, bekannt & spezifisch, konsistent, Kontext nicht wiederholen

**Variablen**: camelCase ,**Konstanten**: ALL\_CAPS, **Booleans**: is…, **Arrays**: Plural, **Klassen**: Substantive

### Funktionen

kurz, eine Aufgabe erledigen, möglichst wenig Argumente, beschreibender Name, nur das im Namen tun, Namensgebung: camelCase, mit Verb beginnen

A screenshot of a computer code

Description automatically generated

Beispiel:

function getSumShoppingCart(s) { return s.reduce(  
(cur, next)=>cur + (next.count ?? 1) \* next.price, 0); }

### Weitere Regeln

* Schlecht (Werte mit ungleichem Typ verglichen):  
  inputElement.value == 0;  
  Besser: parseInt(inputElement.value, 10) === 0;
* Schlecht (mit Truthyness getrickst):  
  if(array.length) { /\* … \*/ }Besser: if(array.length > 0) { /\* … \*/ }
* Keine Magic Numbers, stattdessen Konstanten
* Built-In Funktionen nutzen
* Schlecht: const name = "Capt. Janeway";  
  Besser: const name = 'Capt. Janeway';
* Schlecht: console.log('Hi ' + name + '!');  
  Besser: console.log(`Hi ${name}!`);
* Schlecht: s = new SuperPower();  
  Besser: const s = new SuperPower();
* Schlecht: if(isJedi) { … }  
  Besser: if (isJedi) { … }
* AirBnB Styleguide: no-var, prefer-const, quotes, operator-linebreak, eqeqeq, space-infix-ops, eol-last, no-multiple-empty-lines, semi
* Linter: Tool zur Einhaltung von Clean Code-Regeln, ESLint für JS, Stylelint für CSS

## Model View Controller (MVC)

**Model**: Zustand der App verwalten und verändern, Datenmodell, User-Daten, Anwendungslogik, no View Refs

**View/Darstellung**: DOM aktualisieren, Event-Listener registrieren

**Controller/Interaktion**: Event-Handling, User-Input, View-Wechsel

A screen shot of a computer program

Description automatically generated

A diagram of a controller

Description automatically generated

## Hypertext Transfer Protocol (HTTP)

Aufbau **Request**: HTTP-Methode, URL Path, Version, Headers, Body

Aufbau **Response**: Protocol Version (1.0/1.1/2.0/3.0), Status Code, Response Headers, Response Body

HTTP-Methoden:

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name*** | ***Effekt*** | ***Idemp.*** | ***Sicher*** |
| GET | R in bestimmter Repräsentation zurückgeben | Ja | Ja |
| POST | Daten an Server senden und ggf. R erstellen | Nein | Nein |
| PUT | R erstellen/ersetzen | Ja | Nein |
| DELETE | Bestimmte R löschen | Ja | Nein |
| PATCH | Teilweises Update | Nein | Nein |

**R**=Ressource, **Idempotent**=kann mehrmals ausgeführt werden mit selbem Ergebnis, **Sicher**=Keine Seiteneffekte, Server-Zustand bleibt

**Statuscodes**: 1xx Informational, 2xx Successful, 3xx Redirection, 4xx Client Error, 5xx Server Error

Beim **Caching** werden Daten auf näheren HTTP-Servern zwischengespeichert und von dort ausgeliefert, kann mit dem Response-Header Cache-Control: "max-age=0" deaktiviert werden

## Cross-Origin Resource Sharing

Site kommt von domain-a.com

Daten auf dieser Site kommen teilweise von domain-b.com, der Zugriff wird von CORS auf dem Webserver von domain-b.com kontrolliert, standardmässig funktioniert es nicht aufgrund der same-origin Policy (SOP)

Die Lösung dafür ist, dass bei der Response von domain-b.com der Header Access-Control-Allow-Origin: <http://domain-a.com> gesetzt sein muss

## Node.js

Laufzeitumgebung für server-seitiges JavaScript, asynchron, event-basiert, geeignet für Webserver, NPM Paketmanagement

Paket erstellen: "npm init"

Jedes Paket benötigt: **package.json**, **.js-Files**, **node\_modules** Ordner mit Dependencies aus package.json nach "npm install"

Simple Server:

import http from 'http';

const server = http.createServer((req, res) => {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('Hello, World!'); });

server.listen(3000, () => {

console.log(`Server is running on [http://localhost:${3000}`](http://localhost:$%7b3000%7d%60)); });

Static Server:

import http from 'http';

import nstatic from 'node-static';

const fileServer = new nstatic.Server('./pub');

function requestHandler(req, res) {

fileServer.serve(req, res); }

const server = http.createServer(requestHandler);

server.listen(8080, () => console.log('p',8080));

## Representational State Transfer (REST)

Software-Architektur Style, bestehend aus Guidelines und Best Practices, Resource-oriented Architecture:

**Ressource**: was wichtig genug ist, um eigenständig referenziert zu werden. **Name**: Eindeutige ID der Ressource, **Repräsentation**: Infos über Zustand einer Ressource, **Links**: Hyperlinks verknüpfen Ressourcen, **Schnittstelle**: Einheitliche Schnittstelle für Ressourcen-Manipulation, **Statuslose Kommunikation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| http:// | [www.a.ch](http://www.a.ch) | :80 | /path/a | ?k1=v1&k2=v2 | #loc |
| Scheme | FQDN | Port | Path | Parameters | Anchor |

Ressource **anfordern**: GET <https://.../orders/1>

Ressource **versenden**: POST <https://.../orders>  
Body: { productId: 33, amount: 10 }

### Richardson's Maturity Model

**Level 0 (The Swamp of Plain old XML (POX))**: einen einzigen URI und eine einzige HTTP-Methode (meist POST)

**Level 1 (Ressourcen)**: verschiedene URIs für verschiedene Ressourcen, eine einzige HTTP-Methode (meist POST)

**Level 2 (HTTP-Verben)**: verschiedene HTTP-Methoden auf verschiedene URIs haben andere Wirkungen

**Level 3 (Hypermedia Controls)**: Die Responses beschreiben sich selbst dank Hypermedia as the Engine of Application State (HATEOAS), Clients können ohne Doku durch die API navigieren und und verstehen, Client-Server-Entkoppelung

**Ressourcenorientierte Architektur**: Ressourcen werden in unterschiedlichen Repräsentationen zur Verfügung gestellt, Gegensatz zur funktionalen Sicht, z.B. bei RPCs

**Merkmale REST-Schnittstelle**: Korrekte Verwendung der HTTP-Verben, Zustandslosigkeit

**Polyfill**: rüstet in älteren Browsern eine neuere, nicht unterstütze Funktion nach