1 Begriffe

HTML (Hyper Text Markup Language) ist für die Semantic zuständing. CSS für das styling der Elemente und JS für das interaktive Verhalten der Website.
Web-Seite ist ein einzelnes HTML-Dokument, eine **Web-Site** mehrer verlinkte Web-Seiten unter der selben Domain.

2 HTML - Basics

Parsing: Load HTML -> Parse HTML -> Generate DOM -> Display Encoding muss angegeben werden (`<meta charset ="UTF-8>`. **Void-Elemente** haben kein Content und kein End-Tag. Ein End-Slash ist erlaubt (`<tag attribute ="value"/h>`
Im head-Element stehen Meta-Informatinen zur Website (charset, Language, Title, …).

2.1 Content Model

Nicht alle Tags können beliebig in einander verschachtelt werden. Bei jedem Tag sind folgende Eigenschaften angegeben: **Permitted Content**, **Tag omission**, **Permitted parents**, **Permitted ARIA roles** und **DOM interface**.

Tag-Omission gibt an, ob ein Tag (Start / End) vorhanden sein muss, weggelassen werden kann, oder nicht vorhanden sein darf (Void-Elemente). Es sollte dennoch alle erlaubten Start- und End-Tags gesetzt werden.

2.2 Section

body, article, aside, nav und section können einen header, footer, h1 besitzten. Bei neuen Sections ist zwar ein neues h1 erlaubt, gilt aber nicht als best-practice (Screenreaders)

2.3 Semantic

Tags für die Semantic korrekt einsetzten

- main
- header
- footer
- nav
- figure
- figcaption
- article
- section
- aside
- details

3 JavaScript (JS)

Offizieller Name **ECMAScript**. JS ist **dynamisch**, **dynamisch typisiert**, **funktional**, **objektorientiert**. Häufig wird keine Exception geworfen und ist meistens interpretiert.
Typen in JS:

- **Primitive Type**:
- string, number, boolean, null, undefined, symbol (ECMAScript 6)
- compared by value
- immer immutable
- **Objekte**:
- Plain Objects, Arrays, Regular Expresions, Functions
- compared by reference
- mutable by default

Achtung: `typeof(null)` gibt `'object'` zurück

3.1 Falsy / Truthy

Als `false`-Werte werden folgende Werte interpretiert:

- false,
- 0
- "" (empty string)
- null
- undefined
- NaN

Der Rest wird als `true` interpretiert.

3.2 Number

Alle Zahlen sind `float`. Wenn möglich werden `float` als `integers` abgebildet. `NaN` (Not a Number)ist ein Error-Wert vom Typ `number`. Vergleich zwischen zwei `NaNs` ergibt immer `false`. Zur Überprüfung muss `isNaN()` verwendet werden. `Infinity` kann auch positiv und negativ sein. Jeder Wert kann in eine Zahl verwandelt werden, ausser `symbol`.

```
+(true) == 1
Number(true) == 1
isNaN(Number('abc')) == true
```

Die Funktionen `parseInt()` und `parseFloat` parsen bis zum ersten Fehler.

3.3 String

Single und Double Quotes sind möglich. Seit ECMAScript 6 gibt es auch **Template Literals** / **Template Strings**. Ermöglicht Strings mit Plateholder. Werden durch Backticks (‘) eingeleitet. Innerhalb von `${...}` wird JS ausgeführt.

3.4 Operatoren

Es gilt **Punkt vor Strich** und es wird von links nach rechts aufgelöst. **Achtung**: `String + Value = String` und `Value + String = String`. Mit dem `==` muss aufgepasst werden, da

automatische Typumwandlungen statt finden. Um die Typumwandlung zu vermeiden sollte `===` verwendet werden.

3.5 null != undefined

undefined: Variable ist nicht definiert oder wurde nicht initialisiert. **null** wird verwendet, wenn ein Objekt erwartet wird. Funktionen die undefined zurückgeben haben keinen Rückgabewert, Funktionen die null zurückgeben haben einen Rückgabe werde, aber nicht in diesem Fall. **Achtung:** `null =` undefined= ergibt `true`.

3.6 array

Zwei for-each-Loops für Arrays: **For-In** iteriert über die Propery Name und *For-Of*über die Werte:

```
for (const x in arr) {
    console.log('for in', x + ',' + arr[x]);
}

for (const x of arr) {
    console.log('for of', x);
}
```

Unterstützt ganzzahlige Nummern als Indexe, andere werden in String umgewandelt (verhält sich wie ein Objekt). Die Funktion `length` des Arrays beachtet aber nur die echten Array Einträgen. **Wichtig**: Arrays wie Arrays verwenden.

3.7 Object

Object ist eine Sammlung von Properties. Properties und Methoden können hinzugefügt und verändert werden. Auch von Standard-Objekten, sollte aber nicht gemacht werden. Jedes Objekt ist eine HashTable (ausgenommen null, undefined). Es kann mittels `obj[PropertyName]` auf die Properties zugegriffen werden. Alle Index werden in String umgewandelt.

3.8 Functions

Bei Funktionen können mehr oder weniger als geforderte Anzahl Parameter übergeben. Funktionen besitzen auch Properties. `name` der Funktion, `length` die Anzahl Parameter. Es können aber weiter Eigenschaften hinzugefügt werden. Funktionen können Funktionen zurückgeben. Alle Parameter werden in `arguments` abgelegt, sollte in Array umgewandelt werden (`Array.from(arguments);`). Mit `...name` kann der letzte Parameter der Funktion mit allen restlichen Werten gefüllt werden als (Array). JS kennt kein **Function Overloading**, die letzte Funktion mit selben Name überschreibt die Vorherigen. Interne Weiche oder default-Werte.

Arrow Funktionen werden oft auch Lambda genannt. Sinnvoll für kleine Funktionen z.B. als Filter Parameter:

```
[1, 2, 3].map(x => x*x)
```

Bei Arrow-Funktionen ist der Context immer auf das selbe Objekt gebunden.

3.9 Scope

Geltungsbereich, wo etwas gültig oder verfügbar ist. Globale variabel können im Browser über `window.myGlobalVariable` bzw. in Node über `global.myGlobalVariable` zugegriffen werden. Globale Variablen vermeiden, aufgrund von **Global Namespace Pollution** (zu viele Variablen / Funktionen => überschreiben sich gegenseitig). Lösung sind Module. Script-Tags erzeugen kann neuen Scope.

3.10 Context

`this` ist der aktuelle Context. Je nach Aufrufart wird ein anders Objekt referenziert. Bei `object.foo()` wird `object` referenziert. Bei *unbound* Funktionen zeigt `this` auf das globale Objekt. Mittels `apply()` oder `call()` kann der Context gesetzt werden. Die vorherigen Regel werden ignoriert. Z.B. `foo.call({ counter: 123});`

3.11 Strict Mode

Mittels **use strict** werden folgenden

- Eliminiert *fails slently*
- Probleme welche es dem Compiler verermöglicht den Code zu optimieren
- Security wird leicht verbessert

Strict Mode kann nicht mehr entfernt werden, wenn mal aktiviert, muss pro File aktiviert werden und Strict Mode JS sollte nicht mit Non Strict Mode JS zu einem JS file vereint werden. Sollte nach Möglichkeit immer genutzt werden.

3.12 Klassen

Ähnlich wie in Java. Private wird mit dem Prefix `#` ermöglicht. Es gibt getter und setter (`get time() {...} / set time(){...}`. JS unterstützt Vererbung (`class AlarmClock extends Clock {...}`). Mittels

`instanceof` kann überprüft werden ob ein Objekt eine Instanz einer Klasse ist. Mittels `super` kann auf den Parent zugegriffen werden.

Es gibt auch **Prototype - kein Prüfungsstoff**

3.13 Module

Script-Tags erzeugen kein neuen Scope, Reihenfolge wie die Files eingebunden werden ist relevant. Um die Probleme zu umgehen können EcmaScript Module (ESM) eingesetzt werden. Werden im HTML wie folgt eingebunden:

```
<script src="main.js" defer type="module"></script>
```

Mit `import ... from '...'` werden Funktionalität aus einem Modul importiert. Mittels `export` wird Funktionalität exportiert. Es gibt beim Import Named (`bar as b`) und Default. Sind immer im *strict mode*

3.14 Shim / Polyfill

A polyfill is a shim for a browser API. Implementieren API's die noch nicht zur Verfügung stehen, fix JS Bugs in alten Browser, können entfernt werden, wenn nicht mehr notwendig.

3.15 Symbol

A symbol is a unique and immutable data type and may be used as an identifier for object properteis

4 CSS

4.1 Selektor

- `#` ID
- `.` class
- Typen (tags)
- `*` Universal
- `tag[attribute]`
- `a::before` Pseudo-Element
- `a:hover` = Pseudo-Element
- `article.recent` Compound
- `.recent h1` decendant
- `>` child
- `+` next sibling
- `~` subsequent siblings

4.2 Attribut Selektoren

Mit dem Attribut-Selektor kann man vergleiche Anstellen:

- `[a=v]`: ist gleich v
- `[a~=v]`: enthält v
- `[a|=v]`: beginnt mit v-
- `[a^=v]`: beginnt mit v
- `[a$=v]`: endet mit v
- `[a*=v]`: v ist mindestens einmal enthalten

4.3 Pseudoelement

Pseudoelemente erlauben es, bestimmte Teile eines Elements zu formatieren.

- `::first-line`
- `::first-letter`
- `::before` (erstellt neues Element)
- `::after` (erstellt neues Element)
- `::selection`

4.4 Pseudoklassen

Pseudklassen werden verwendet, um auf Basis von verschiedenen Zuständen oder Eigenschaften Elemente zu selektieren.

- dynamische Klassen: `:hover`, `:focus`, `:visited`, …
- strukturelle Klassen: `:root`, `:first-child`, `:nth-child()`, `:empty`, …
- andere: `:lang()`, `:not()`, …

Bei den Pseudoklassen die mit `nth-` beginnen, starte der Index bei 1. Die Pseudoklasse `:not` darf keine Pseudoelemente und keine weiteres `:not` enthalten.

4.5 Kombinatoren

- `e > s`: Kindselektor
- `e s`: Nachfarenselektor
- `e + s`: Angrenzende Geschwister (Nachbar)
- `e ~ s`: Allgemeine Geschwister

4.6 Spezifität

Mit CSS können verschiedene gegensätzliche Anweisungen verwendet werden. Hier entscheidet die Spezifizität. Quellen nach Priorität:

1. Browser (mit Default Werten)
2. Benutzer (Vom Nutzer des Browser)
3. Autor (Stylesheets, welche mit der Webseite mitgeliefert werden.

Um die Spezifität wird aus dem gesamten Selektor berechnet:

1. Drei Zähler (A, B, C) mit Startwert 0
2. A++ für jeden Selektor von IDs (#…)
3. B++ für jeden Selektor von Klassen (. …), sowie Pseudoklassen (: …) und Attributen ([…])
4. C++ für jeden Selektor von Element-Typen sowie Pseudoelementen (:: …)
5. Spzifität = $A \cdot 100 + B \cdot 10 + C$

Der Universalselektor `*` wird ignoriert.

Das Keyword `!important` kann ein Style als wichtig markieren. Sollte nie verwendet werden, wenn das CSS unter eigener Kontrolle ist.

5 DOM

Document Object Model (DOM) repräsentiert HTML-Dokument als Busstruktur. Attribute Nodes sind zugreifbar mit `elementNode.getAttributeNode()` gelten aber nicht als Teil des DOM Trees.

5.1 DOM suchen

- `document.getElementById()`
- `document.getElemtsByName()`
- `<searchRoot>.getElementsByClassName()`
- `<searchRoot>.getElementsByTagName()`
- `<searchRoot>.querySelector()`
- `<searchRoot>.querySelectorAll()`
- `<searchRoot>.closets()`
- `<searchRoot>.matches()`

die `getElementsBy...` Funktionen geben eine `HTMLCollection` zurück (kein Array). Beim Anpassen des DOMs wird die Collection angepasst (Live-Queries, gefährlich). Es sollte daher die modernen Such-Funktionen verwendet werden. Diese erzeugen keine Live-Queries. Auch die neuen Funktionen geben **kein** Array zurück (umwandlung mittels `Array.from()` oder `[...nodeList]`)

5.2 DOM Navigation

Node Interface hat Methoden zur Navigation in DOM (read-only!):

- `<node>.parentElement`
- `<node>.childNodes` (auch Text-Nods, Kommentare)
- `<node>.children` (nur HTMLElemente)
- `<node>.firstChild` (auch text-Nodes, Kommentare)
- `<node>.firstElementChild` (HTMLElemente)
- `<node>.nextSibling` (auch Text, Kommentare)
- `<node>.nextElementSibling` (Element)

5.3 DOM Manipulation

Neues Element erstellen:

- `document.createElement()`
- `document.createTextNode()`
- `document.createDocumentFragment`

Empfohlen über `container.insertAdjacentHTML()`. Um CSS-Classen zu manipulieren sollten `<element>.classList` genutzt werden und nicht mehr `className`. Die Property Style sollte nicht genutzt werden. Besser ist das setzten einer CSS-Klasse.

Bezüglich Performance sollte **nicht** innerHTML in einer Schleife verwendet werden (Einmalig OK).

5.4 Attribute

- `setAttribute()`
- `removeAttribute()`

Wenn boolesche Attribute im HTML gesetzt werden sind sie true (auch `checked ="false"` ist true). Wenn das Attribute nicht vorhanden ist, ist es false.

5.5 DOM Events

Um Events auf einem Element zu registrieren geht man wie folgt vor:

```
const element = document.querySelector('#myID');
element.addEventListner('click', () => alert(1));
```

Möglich auch über `element.on...` oder im HTML inline. Mittels `addEventListener` sind mehrere Registrierungen möglich. Die `on...` Variante überschreibt die vorherige Zuweisung. Die Inline-Version ist geeignet für Prototypen / Prüfung / Generierten Code).
Eine Event durchläuft 3. Phasen:

1. **Capture-Phase**: vom Root zum Leaf (jedes Element kann reagieren)
2. **Target-Phase**: Event wird auf dem *Ziel* ausgelöst
3. **Bubble Phase**: Das Event durchläuft den DOM von Leaf zum Root (nicht jedes Event)

Bei den meisten Events werden Zusatzinformationen mit gegeben. Diese werden der Event-Callback-Funktion als Parmeter mitgegeben. Das Event-Object hat folgende wichtigen Properties / Methoden:

- `target`: Element auf dem der Event ausgelöst wurde (z.B: l
- `preventDefault()`: verhindert Default-Aktionen.
- `currentTarget`
- `stopPropagation()`: Ermöglicht das deaktivieren von *Bubbling* und *Capturing*

5.6 DOM lifecycle

DOM kennt drei unterschiedliche Zustände:

- **loading**: Dokument wird geladen
- **interactive**: Dokument wurde geladen, gewisse Ressourcen fehlen noch (Bilder, Stylsheets, Frames)
- **complete**: Das Dokument und alle Ressourcen sind

geladen (wird durch das Event `load` representiert)

Script tags können mit `defer` oder `async` markiert werden. **defer** Script Files sollen erst name dem Parsen vom Dokument ausgeführt werden. Reihenfolge der Files wird gewährleistet. Bei **async** werden die Script-Files asynchron geladen, die Reihenfolge der Files wird nicht gewährleistet.

5.7 Data-* Attribute

HTML Elemente habe fest definierte Attribute. Eigenen Attribute müssen mit dem Prefix `data` beginnen. Diese Attribute könne benutzt werden um Daten zwischen HTML und JS auszutauschen. Die Werte werden als *Dictionary* im Property `dataset` abgelegt. Im HTML wird kebab-case verwendet. Im Dictionary wird es als camelCase verwendet (`data-factory-id => dataSet.factoryId`)

6 Clean Code

Clean Code meint sauberen, gut wartbaren und gut lesbaren Code. Clean Code sollte immer das Ziel sein. Um dieses Ziel zu erreichen sollten Linter zum Einsatz kommen, welche Style-Regeln aufzwingen sollen. Wichtige Punkte für Clean Code:

- Names: Descripitive Names, Use Standard Nomeclature Where Possible
- Functions: Too Many Arguments, Output Arguments, Flag Arguments => Split, any dead code
- Comments: Obsolet Comment, Commented-Out Code
- General: Use Explanatory Varaibles, Follow Standard Conventions, Replace Magic Number with Named Constats

6.1 Clean DOM - MVC

Model-View-Controller (MVC) ist eine GUI Architektur. Das **Model** speichert den Status der App (User Data, UI State) und die Busniness Logic. Das Model hat keine Referenzen auf den DOM. Die **View** ist für die Darstellung zuständig. Der **Controller** verarbeitet User Input und wechselt die View (Routing).

6.2 async JS

Async JS ist JavaScript, dass in Callbacks ausgeführt wird. (Jake Archibald: In The Loop - JSConf.Asia - YouTube). Event Handler, Timer Callbacks (setTimout), Network Callbacks (fetch) und async Functions, Promise, await sind async. Es sollten keine DOM Updates im Main-Thread gemacht werden (blocking)

7 HTTP

URL: Anchor / Frament identifyer wird nicht an den Server geschickt. Für das URL encoding sollten `encdeURI` und `encodeURIComponent`

7.1 Promise

Wichtige Zustände:

- **pending**: Async-Operation aktiv
- **fullfilled**: Async-Operation erfolgreich beendet, Wert verfügbar
- **rejected**: Async-Operation erfolgreich beendet, Fehler-Detail verfügbar

Funktionen mit Promises `.then()`, `.cancel()`, `await`, `try-catch`.

7.2 Fetch

```
fetch(url)
    .then(response => {
        return response.text();
    })
    .then(text => console.log(text
```

```
const response = fetch(url);
console.log(response);
const responseText = response.text
console.log(responseText);
```

`.catch()` kann am Schluss angehängt werden und es fängt alle vorherigen Erros auf. Alternative kann mit `try-catch` gearbeitet werden.
Um eine POST Request zu erstellen, müssen die Optionen als Objekt an `fetch` übergeben werden.

```
const fetchOptions = {
    "method": "POST",
    "header": {
        "accept": "application/jso
        "content-type": "applicati
    },
    "body": JSON.stringify(song)
}
const response = await fetch(url,
```

8 Node

8.1 Webserver

Um mit Node einen Webserver zu programmieren muss das Modul `http` importiert werden.

```
import http from 'http';
```

```
const PORT = 8080;

function requestHandler(req, res) {
    res.write('<h1>Hello ');
    res.end('World</h1>');
}

const server = http.createServer();
server.on('request', requestHandler);
server.listen(PORT,
        () => console.log(`Node Server running on Port ${PORT}`));

const parsedURL = new URL(req.url, `http://${req.headers.host}`);
```

Do not implement your own file server! Benutzt ein Package, z.B. node-static.

8.2 SOP / CORS

Same Origin Policy (SOP): Nur Seiten dürfen sich verbinden, die auch von diesem Server geservt wurde. **Cross Origin Resource Sharing (CORS)** wenn ein Server auch von anderen Origins kontaktiert wird, muss der Server im Header der Response das Feld `Access-Control-Allow-Origin` auf die Origin des Browser bzw. auf `*` setzten.

9 REST

Representational State Transfer (REST) Levels of REST: Level 0 - The Swamp of POX, Level 1 - Resources, Level 2 - HTTP Verbs, Level 3 - Hypermedia Controls

10 End