Vorwissen

CSS

```
p:nth-child(n) /* select every child*/
p:nth-child(7) /* selects the 7th element */
p:nth-child(n+8):nth-child(-n+15) // eighth through the fifteenth
```

CSS Variablen

```
:root {   --me-color: hsl(350,50%,50%); }
.top-menu h1 {   color: var(--me-color); }
```

Kann nicht auf media-queries angewendet werden

Javascript

Arrow vs Functions: Funktionen referenzieren den Kontext, indem sie sich befinden. Arrows besitzen keine Referenz auf Kontext (this -> fail)(Closure = Funktion + Referenz auf Kontext)

Prototype Functions

```
Object.keys(object) // alle Attribut-Namen des Objekts,
Object.values(object) // alle Attribut-Werte des Objekts
Arrays.every(fn(e): boolean) // check if all elements true
Array.includes(obj) // equal check each elem in array
```

Nodejs

Module Suchreihenfolge 1. Core-Module, 2. falls mit Slash/Punkt am Anfang: (a. File, b. Directory), 3. falls Filename (require('mymodule')): in node_modules, danach bis zum System Root. node_modules nicht ausliefern! Kann Binary-Teile beinhalten, besser in package.json.

Express

Middleware Die Reihenfolge der Registrierung bestimmt die Ausführungsreihenfolge. **Static Middleware** es sind auch mehrere static-Routes möglich. **Error-Middleware** muss als letztes registriert werden. Es können mehrere sein, die Letzte muss die Anfrage beenden. Wird aufgerufen, wenn Error-Objekt dem Callback übergeben wird. Signatur wie normale Middleware, aber zusätzlicher erster Parameter error

Ordnerstruktur vs. Architektur

/routes Entry point Front Controller (does routing). /controller **Controller**. /services **Model**.

Token Idee: bei jeder Anfrage ein Token mitgeben. Bildet Berechtigungen ab, hat Ablaufdatum. Bsp: JSON Web Token (JWT).

Callback vs Promise

Callback `fs.readFile(path, [, options], callback) // return void`

Promise `fs.Promises.readFile(path[, options]) // return <Promise>`

```
async function promisedFileData(path) { // callback to promises
    return new Promise((resolve, reject) => {
        fs.readFile(path, 'utf8', function (err, data) {
            if (err) { reject(err); }
            resolve(data);
        });
    });
}
```

App.js

```
const express = require('express'), bodyParser = require('body-parser'), session = require('express-session'), router = require()
const app = express()
app.set('views', path.join(__dirname, 'views'))
app.set('view engine', 'hbs')
app.use(bodyparser.urlencoded())
app.use(cookieParser())
app.use(session({secret: 'abc', resave: false, cookie {secure: true}})
app.use(express.static(path.join(__dirname, 'public')))
app.use("/", routes)
app.use((req, res, next) => { // catch 404
    const err = new Error('Not found'); err.status = 404;
    next(err) // will skip to error middleware
});
app.use((err, req, res, next) => { // error middleware
    res.status(err.status || 500)
    res.render('error') });
module.exports = app
```

routes/search.js

```
const express = require("express")
const controller = require("../controllers/searchController")
const router = express.Router()
router.post("/:id/edit", noteController.edit); router.get("*", noteController.index);
module.export = router
```

controller/noteController.js

```
const noteService = require('../services/noteService.js');
function edit(req, res) {
    const { theme } = req.session.userSettings
    if (!req.params.id) { res.redirect(302, "/");}
    if (req.method === "POST") {
        notesService.update(req.body,
            (err) => if(err) throw err; render("index", {theme}));
    } else { const note = notesService.get(req.params.id);
        res.render("form", { note, theme });
    }
}
function add(req, res) {
    req.session.warenkorb[req.body.id] =
        (req.session.warenkorb[req.body.id] || 0) + 1;
    res.redirect("/");
}
module.export = {edit}
```

service/noteService.js

```
const Datastore = require('nedb'); const db = new Datastore({filename: './notes.db', autoload: true});
class Note { constructor(title, importance) {
this.title = title; this.importance} }
function update (note, cb) {
    if(!note._id) throw new Error("error msg");
    db.update({ _id: note._id }, note, {}, cb); // synchron
}
async function get(id) {
    return await db.findOne({ _id: id }); // async
}
module.export = { update, get }
```

Views/index.hbs

```
<ul>
{{#each items}}
    <li>
        <form action="/add" method="post">
           {{name}} a {{preis}} CHF (<strong>{{count}}</strong>)
           <input name="id" value="{{_id}}" hidden> // this._id
           <button type="submit">+</button>
           {{#if count}}
             <button type="submit" formaction="/remove">-</button>
           {{/if}}
        </form>
    </li>
{{/each}}
</ul>
```

middleware/authentication.js

```
function authenticationMiddleware(req, res, next) {
    if(!req.session.user) {res.redirect('/login') }
    next()
}
```

REST

URL Regeln Ressourcen werden immer mit einer URL identifiziert (Resource Oriented Architecture **ROA**). Ressourcen in Mehrzahl, Query-Parameter nur für Algorithmen oder Filter (/orders?state=delay). Hat man verlinkte Ressourcen, so kann man sie entweder direkt in der Antwort mitgeben, oder die URL zum Abruf dieser Ressource zurückgeben.

HATEOEAS Hypermedia as the engine of application state. "Browsable API", selbstbeschreibend, technisch sehr schwer umsetzbar.

Versionierung Oftmals ein Knackpunkt. In der Realität oft über URL, auch wenn das ROA widerspricht. Andere Varianten wären der Media-Type im Accept. Am besten: keine Versionierung

Best Practices Use nouns but no verbs. Use plural nouns. Use HTTP status codes. Respect the meaning of the HTTP methods: GET method should not alter the state. PUT is idempotent, POST is not idempotent

HTTP-Methoden

GET: abrufen, Accept-Header definiert Repräsentation. Status-Codes beachten. **POST**: neue Ressource erzeugen. Content-Type Header angeben. Wenn die Ressource erzeugt wird, im Response-Header. Location die URI angeben. Kein Cache. **PUT** aktualisiert eine Ressource, oder erzeugt sie falls noch nicht vorhanden. Idempotent! Kein Cache. **DELETE** löscht eine Ressource. Kein Cache. **OPTIONS** gibt an, wie die Ressource verwendet werden darf, z.B. welche Methoden erlaubt sind. **HEAD** genau gleich wie GET, aber ohne die eigentliche Ressource. Mit Cache. Header-Informationen und Statuscodes sind relevant. **PATCH** wird für partielles Updaten genutzt. **Repräsentation** Wie die Ressource schlussendlich repräsentiert werden soll (JSON, XML, Excel-Tabelle, ...) entscheidet der Accept-Request- Header

Typescript, OO

Achtung: Schreibt man eine Klasse, die von einer (in der Datei) später definierten Klasse erbt, gibt es einen **Compiler-Fehler**. Dasselbe mit Funktionen, da der Compiler sie zu diesem Zeitpunkt noch nicht kennt.

--strict enables noImplicitAny, noImplicitThis, strictNullChecks (null and undefined values are not in the domain of every type and are only assignable to themselves) and strictFunctionTypes (contravariantly instead of bivariantly).

Declare: This thing already exists in plain JS code but we need to tell the compiler, which type the plain JS code thingy has

Variablen

Variablen ohne Typendekleration wird «Type-Inferenz» benutzt

```
let myInferedNumVar = 1 /* Int */ myInferedNumVar = 'hi' /* NOK */
let myNumVar: number = 1; let myAnyVar: any = 'hi'
myNumVar = myAnyVar /* OK */
let myNotInferredTupel = [1, 'abcd'];
myNotInferredTupel[2] = 2 /* OK Tupels are not infered*/
declare let foo: string /* variable nur definiert */
```

ES6 Syntax

CommonJS

```
module.export = { add: function(a, b) {a + b } } // export module
const module = require('module') // import module
```

ES6

```
export default = (a,b) => a + b // Export Module
import module from 'module' // Import Module
```

Klassen

Erweiterte ES6 Syntax inkl. Properties und private/readonly. Private kann überschrieben werden.

```
interface IPoint { readonly x: number; readonly y: number }
interface ILikableItem { likes? : number }
class DescribableItem { constructor(public description: string) {} }
class POI extends DescribableItem implements IPoint, ILikableItem {
constructor(public x: number, public y: number,
    description: string, public likes?: number {
        super(description)
    }
}
```

Responsive Layout

Device Pixel: CSS-Pixel sind nicht Device-Pixel. Smartphones haben unterschiedliche Pixel Ratios.

Em sollten für die höchst mögliche Accessibility verwenden. Da es sich an die relative Font size orientiert. **Rem** orientiert sich an root font size

Browser nicht auf dem selben Stand: Zwei Vorgehensweisen. **Graceful Degradation** Alle modernen Features nutzen. Bei älteren Browser Polyfills nutzen, sinnvolle Alternative geben oder auf Problem hinweisen.

Progressive Enhancement für alle Browser zugänglich machen. Mit Grundfunktionalität starten (kein JS, keine Media Queries), dann mit zusätzlichem CSS und JS Zusatzfunktionalität bieten.

Media Queries

Typen: Spezifische CSS für unterschiedliche «Medien» `@media screen {…}, @media print {…}`.

Dimensionen: `@media ( [ width | min-width | max-width ] : 375px) {…}`

Zustände: `@media (orientation: landscape) { ... }`

Support abfragen: Mit `@supports not (flex-wrap: wrap) {…}` kann abgefragt werden, ob ein Browser ein Feature unterstützt. Es gibt abr auch den Modernizr. Operatoren: @media (…) and (…) { … }, not screen, only screen {…}

Breakpoints: 480px/30em: Smartphones, 768px/48em: Tablets, 992px/62em: Desktops

Style Tag: <link rel="stylesheet" href="style.css" media="(min-width: 30em)">

ViewPort

Bei einer responsiven Seite setzt man den Viewport, damit der Browser nicht "intelligent" versucht zu zoomen. Ohne, greifen die Media Queries nicht. `<meta name="viewport" content="width=device-width, initial-scale=1">`. width=device-width: die Seite soll so breit sein, wie das Gerät. initial-scale: 1 zoome zu Beginn so, dass es 100% entspricht.

CSS Box Layout

Box-sizing: content-box Wenn man width von 200px hat und, border von 20px und padding 20px, dann ist das Object 240px gross. Padding und Border werden also hinzugefügt.

Box-sizing: border-box Die Paddings und Borders werden brauchen den Platz gegen innen auf -> ist Empfohlen

Vh/vw steht für view height/width. 100vh ist die ganze Höhe des Screens. Kann aber auch grösser sein, wenn z.B Browserheader erscheint.

Percent: Bezieht sich immer auf Parent element ausser bei translate-top/bottom

Funktionen für die Berechnung von flexiblen Layouts

Calc() Führt eine Berechnung aus. calc(100vh – 5em)

Min()/Max(): Nimmt eine von zwei werten min(200px, 100vh – 5em)

Clamp(): Definiert Verhalten im Bereich clamp(400px, 100vw-5em, 500px)

Position: absolute | fixed | sticky | relative | static

Absolute relativ zum ersten Parent mit position: relative or absolute, Element wird aus dem Element-Fluss entfernt. Erlaubt Überlappung von Elementen

Fixed: Element wird fix in einem Browserfenster platziert (z.B Navigation)

Sticky: Bleibt haften am oberen oder unteren rand

Relative: Referenz für Kind Elemente mit position: absolute

Static: Default -> Element ist im Fluss

Display: block | inline | inline-block | contents

Inline: wie span a-tags, erlaubt left/right margin/padding, aber nicht top/bottom. Ignoriert width/height. Erlaubt andere Elemente auf der gleichen Zeile. White-Space zwischen Inline-Elementen wird dargestellt

Block wie h1 div form p , erlaubt Margin/Padding, jedes Element auf einer neuen "Zeile" (Umbruch), erlaubt Text-Inhalte zu scrollen, clippen (overflow: scroll | hidden | invisible | eclipse. Eclipse -> macht: «balbal…»

Inline Block wie Inline-Flex, Inline-Table erlaubt Margin/Pading, erlaubt andere Elemente auf der gleichen Zeile mit Alignment. (vertical-align: top), erlaubt width/height, White-Space zwischen Inline-Block-Elementen wird dargestellt.

Contents applied die eigenen flex attribute seine Kinder und macht sich somit unsichtbar.

Flexbox

Container:

`display: flex;` gibt flex verhalten an. `display: flex-inline` gegen aussen inline und gegen innen ein flex.

flex-wrap: nowrap | wrap | wrap-reverse

flex-direction: column(top to down) | row(left to right) | row-reverse | column-reverse

justify-content: flex-start | flex-end | center | space-between| space-around | space-evenly (Main Axis)

align-items: flex-start | flex-end | center | baseline | stretch (Cross Axis)

align-content: flex-start | flex-end | center | space-between | space-around | stretch (Cross Axis)

Children Properties

```
flex-grow: <number>; /* default 0 -> size depends on content */ }
```

Flex-shrink: Die Boxen werden beim Verkleinern des Browserfenster kleiner gemacht. Desto grösser Ratio, desto kleiner Box. Ignoriert flex-shrink

Flex-Basis: Anstelle von min-width verwenden für Grösse welche angestrebt werden soll.

Flex: Komb. der oberen 3. 2+3 sind optional.

```
flex: 0 1 auto; // Default properties
```

0 0 50px (werde grösser, noch kleiner werden, Grösse 50 px)

0 1 50px (werde nicht grösser, kann kleiner werden, wenn nicht genug Space)

0 5 50px (auf Box 1: Darf nicht grösser werden, aber 1 muss 5 mal kleiner werden als die restlichen Boxes)

1 0 50px (nimmt voller platz ein, erlaubt es aber nicht die children kleiner zu machen)

align-self: auto | flex-start | flex-end |center | baseline | stretch

Order: <number> Kann Minuswert annehmen

Grid

Container

```
grid-template-columns: 1fr auto 20%;
grid-template-rows: repeat(4, 1fr);
```

fr: Fraction of available Space, auch dezimal. Können nicht schmaler als das längste Wort werden. Overflow wird vermieden.

max-content/min-content grösst-/kleinstmöglich.

auto maximale gewünschte Breite der Grid-Elemente in der Spalte.

minmax(min, max): Wert zwischen min und max wird sichergestellt.

minmax(auto, 1fr): Soviel wie vom Inhalt vorgegeben wachsen mit fr.

repeat(times: int, measure) Wiederholung der measure,

`grid-template-columns: 10em repeat(3, 1fr 2fr)` 10em => 8 Spalten: 10em, 3x (1fr/2fr) abwechselnd, 10em.

`place-items: start | end | center | stretch//align-items+justify-items`

If all element fixed size -> grid is smaller as grid container

justify-content: flex-start | flex-end | center | space-between| space-around | space-evenly (row Axis)

align-content: flex-start | flex-end | center | space-between| space-around | space-evenly (column Axis)

Children Properties

```
grid-column-start: 1; /* mit 1 indexiert */
grid-column-end: span 2; /* 2 ab -start */
grid-row-start: 2;
gird-row-end: -2; /* vis a vis gezählt */
grid-area: 2 / 1 / -2 / span 2; /* Kurzschreibweise */
/* [row-start]/[column-start]/[row-end]/[column-end] */
order: -2; /* default 0 */
place-self: start | end | center | stretch // align-self+justify-self
```

template-areas

```
grid-template-areas: "aa bb bb" "cc dd ." ;  /* Container */
grid-area: aa /* Item */
```

Testing

Phasen: Ein Unit Test hat vier Phasen: (1) Setup, (2) Exercise, (3) Verify, (4) Teardown

Monkey patching is a technique to add, modify, or suppress the default behavior of a piece of code at runtime without changing its original source code.

Types: Integration, Regression, Load (Stress), Performance, Endurance, Chaos, Security, Usability

Mocha

```
describe('Array', function() {
    describe('#indexOf()', function() {
        beforeEach(function() { this.testArray = [1, 2, 3]});
        it('should return ...', function() {
            expect(this.testArray.indexOf(4)).toBe(-1)
        }); // weitere it()...
    }); // weitere describe()...
});
```

Expect API

```
expect(x). toBe(val). toEqual(val). toThrow(err).
toExist(). toBeTruthy(). toNotExist(). toBeFalsy()
```

Spies API

```
const video = { play: () => {...} } // intervening object
```

```
spy = expect.spyOn(video, 'play')
expect(spy.calls.length).toEqual(1)
expect(spy.calls[0].context).toBe(video)
```

Test Patterns

Test Double Pattern DOC (depended-on Component) mit einem Interface ersetzen und einmal real und einmal als Double implementieren.
Test Stub Pattern Immer dasselbe zurückgeben.
Test Spy Pattern

```
class BankAccount { withdraw() {}, deposit() {} }
describe('A new transaction executed', function() {
    beforeEach(function() {
        this.accountA = new BankAccount()
        this.accountB = new BankAccount()
        spyOn(this.AccountA, 'withdraw')
        this.transaction = new Transaction(
        this.accountA, this.accountB, 25)
    });
    it('withdraws 25 from account A', function() {
        this.transaction.execute()
        expect(this.accountA.withdraw).
        toHaveBeenCalledWith(25)
    });
});
```

Fake Object Pattern Z.B. einen Fake Hash Service bauen, der immer auf dem gleichen Seed operiert. Unterschied zum Stub ist sehr fein.
Mock Object Pattern Objekt, das sich selber verifizieren kann, das wir dann fragen können ob alles korrekt lief. Beispiel Libraries: Sinon, Proxyquire

Empfehlungen

Pure Functions: Testbarkeit optimal, wo nötig Test Doubles als Argumente übergeben. **Non-pure Functions/JS Objekte:** Context für Test fixieren, globale Variablen setzen, Veränderung globaler Variablen überprüfen, Veränderung der Input-Argumente überprüfen, wo nötig Test Doubles als Argumente übergeben.

Web Security Grundlagen

Bekannt aus InfSi 1

CIA Dreieck: Confidentiality, Availability, Integrity; **Stride Threat Model**: Spoofing (falsified Identity), Tampering (unauthorized changes), Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege

CSRF: Cross Site Request Forgery

Beschreibung: Angreifer von malicious Seite liest client session und führt Request in Opfers Namen aus.
Angriff: Bei einer Seite angemeldet sein, und eine fremde Seite sendet in meinem Namen Daten an die erste Seite.
Lösung: csurf, http-only cookie, JWT

XSS: Cross Site Scripting oder Stored-XSS

Beschreibung: In einem Formular angegebene Daten werden an andere Nutzer ohne Escaping vom Server ausgeliefert.
Angriff: Gibt im Input Feld JavaScript-Code statt regulärem Text HTML mit JS ein.
Lösung: Encoding bei der Darstellung (Sanitizin), Frameworks wie Handlebar haben das kommt aber auf context drauf an, csp,

JS Remote Code Execution / Code Injection

Beschreibung: wenn ein Angreifer den Server dazu bringen kann vom Angreifer eingeschleusten Code zum Ausführen zu bringen.
Angriff: Eingabe von JS im Feld
Gegenmassnahmen: NICHT `eval()` verwenden, Global scopes reduzieren

Broken Authentication

Beschreibung: Bei Problemen bei der Authentisierung und dem Session-Management können externe Angreifer mit einem validen Login auf Informationen zugreifen, welche nicht für sie bestimmt sind.
Angriff: Network Sniffing, Session Timeout zu lang, Cross-Site Requerst Forgery (CSRF)
Gegenmassnahmen: HTTPS, PW based Auth auslagern, session timout sinvoll, CSRF-Token, Express csrf-middleware

Insecure Direct Object Reference

Beschreibung: Über eine manipulierte URL auf Daten zugreifen, für die man keine Berechtigung hat.
Gegenmassnahmen: sicherstellen, dass der Benutzer eingeloggt ist (Session-Token).

Replay Attacks

Beschreibung: Die gleiche Aufgabe und Lösung (das gleiche Formular) kann beliebig häufig eingereicht werden. Jede Einreichung wird als Erfolg gerechnet.
Gegenmassnahmen: CSRF-Token, in Session Status speichern

Security Node.js Application

Preventing: Enabling CSRF Protection & Using correct http methods (GET-> does not mutate). Remove `X-Powered-BY` Benutzung generischer cookie namen.
HTTP Parameter Pollution (HPP): Man gibt mehrere Request parameter mit gleichen Namen. Triggert Typererror -> Denial of service & bypass validation.
Lösung: Type Checking, Good error handling mechanism

Regular Expression Denial Of Service Attack (ReDos): Regex ist O(n^2). **Lösung**: Review Reegex, Not use user supplied inputs as regex.
Nur wenn Team gross genug, dann eigenes Passwort Management. Alternative IAM provider. Passwordless login mit email und code.

Web Accessibility

Farbenblindheit: Informationen nicht nur mit Farbe codieren (Gray-Scale Test), Doppel-Codierung / Mehrfach-Codierung -> nicht Info nur mit Farbe darstellen
Kontrast: Wichtig für Personen mit Sehschwäche und ab 50+, 15.9:1 -> 80+ (AAA), 5,7:1 -> 50+(AA), unter 5,7 Kritisch, Tools dafür sind limitiert.
Zoombarkeit: Zoom sollte nicht unterbunden werden.
Animationen: Sollte man abstellen können. Verringerung der Ablenkung, Epilepsie und Migräne
Auszeichnung von Medien: Bilder immer mit Alt-Text
Keyboard focusable: Personen die Screenreadre benutzen oder keine Maus bedienen können
Screen-Reader Unterstützung: Keine Headings-Level auslassen, sematic elements richtig nutzen, skip-Links to main content am Anfang der Seite
WAI-ARIA: Web Accessibility Initiative Accessible Rich Internet Applications–Accessibile RIA -> haben Guidlines, -> CH version Zugang für alle
Tables: Mit heading für Rows und Columns (th), mit captions
Custom Controls: Auch mit form und input machen, keine content über css
Accessibility Tree: Ist eine vereinfachte und mit Zusatzinformationen ausgestatte Repräsentation von dem HTML. (Beispiel mit btn Switch)

Accessibility Regeln - Technologieunabhängig

Bilder etc. mit angemessener **textueller Beschreibung** (alt-Text); **Keine Information** ausschliesslich durch Farbe dargestellt (Test: Graustufenbild)
Vorder- und Hintergrund bei reduzierter Farb- und Kontrastwahrnehmung in Standardansicht **deutlich unterscheidbar**
Skalierung der Schrift über Funktionen des Browsers möglich (arbeiten mit em/rem und % statt px)
Jede Funktion der Seite ist auch über die alleinige **Verwendung der Tastatur** in einer schlüssigen Reihenfolge zu erreichen (z.B. sichtbarer Fokus)

Wichtigste Regeln um HTML Accessibility zu verbessern

Wichtige Seitenelemente am Start der Seite: Navigation zuerst, Bereiche gruppieren, alt=Beschreibungen
Semantic Markup nutzen: nav, header, main, h1 …**Nicht div, span, Tabellen für Layout,** Tabellen mit Daten mit Colum Heading (thead, th scope=col), Caption, Form mit <label>- Elementen für <input> Elemente. **Forms:** <label> für <input> HTML5 Validation nutzen
Aria-Labels: Fügt zusätzliche accessibility hinzu. z.B `aria-label="Close"`

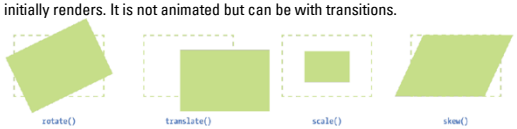
Animation mit CSS

Transitions

CSS transitions create a smooth change from one state to another. They fill in the frames in between (tweening).
`transition-property:` Which property to change
`transition-duration`: Wie lange dauert die transition in s. (or ms.)
`transition-timing-function`: Wie sollte sich die Transition beschleunigen (ease, linear, ease-in, ease-out, ease-in-out, step-start, step-end, steps(), cubic-bezier(#,#,#,#)
`transition-delay`: Wie lange sollte die Pause sein, bevor die Transition startet
`transition:` Shorthand Property `transition: background-color .3s ease-in-out 0.2s, all 2s ease-in;`
Wenn man die opacity animieren will muss man auch die visibility animieren `transition: visibility 0s linear 2s, opacity 2s;`

Transform

The transform property changes the shape and location of an element when it initially renders. It is not animated but can be with transitions.




```
img {transform: rotate(-10deg)}
img {transform: translate(90px, 60px)} // translate(x, y)
img {transform: scale(1.5)} // scaleX(), scaleY(), scale(1.2, 0.5)
img {transform: skew(15deg)} // skewX(), skewY(), skew(10deg, -12deg)
img {transform: scale(1.5) rotate(-5deg) translate(50px, 30px)}
img {transition: transform 1s} img:hover { transform: rotate(10deg)}
```

`transform-origin: center top;` Um was rotiert werden soll
Mit `:hover` werden die vorherigen transform Attribute überschrieben

Keyframe Animation

Keyframe animation enables you to create transitions between a series of states (keyframes). 1. Erstelle Keyframe, 2. Apply animation properties

```
@keyframes rainbow { 0% { background-color: red; } 50% { background-color: orange; } 100% { background-color: yellow; } }
#magic {
  animation: rainbow 5s linear infinite;
  // animation-name: rainbow;
  // animation-duration: 5s;
  // animation-timing-function: linear;
  // animation-iteration-count:
  // infinite; animation-direction: alternate;}
```

Es gibt non-animateable properties: border-style, display, … ; die meisten Properties mit quantitativen Wert sind animierbar.

Other Animation

JS-based Animation: Powerful but also not performant
SVG-based Animation (CSS/JS):Might be the best if browser support is given

User Centered Design (UCD)

UC Requirements Engineering / UC Product Management

User, Task, Tool, Context need to be considered in user-centered design. Asking user is not user centered design. Customer and user can be different people.
The user is the **problem expert**, the designer is the **solution expert**.
Good User Research: Representative set of users; do a set of realistic tasks; using current tooks&strategies; representative context
Scenarios (and Personas): Story of user solving the problem; Problem-Scenario show current (problematic) situation; Future-Scenario show how new tool lead better outcomes -> need to be validated by presenting them to user and customers.

Raster Design, Struktur Design

Ausschilderung: Wo bin ich im Moment; Wo kann ich hin; was ist passiert; Breadcrums -> Wo bin ich?
Kofferraum Test bestimmt die Qualität der "Ausschilderung": Site-Kennung, Seitname, Sektionen, Lokale Navigation, Anzeiger für «Sie befinden sich hier», Wie kann ich die Suche Starten?
Navigationsdesign ist wie Haus-Architektur, welche tools? welche Ansichten, Navigation zwischen Views
Concept-Model: Wichtige Konzepte und deren Relationen (z.B. UML-Modell mit Student, Studiengang, …)
Design Diagramm: Site map (Baum/Netz der Seiten)
Card Sorting: Jede Benutzergrupper ordnen die Navigation Links in Gruppen. Erst Clustering erstellen und danach Clustering überprüfen«Information Scent». Ist nur für eine Hirarchiestufe
Tree Testing: Aktuelle Baumstruktur (Site Map) aufnehmen -> Aufgabe zur Erreichung von Zielen erstellen -> testet ganze Baumstruktur
Usability Testing: Herausforderung Erstellung guter Aufgaben. Gute Aufgaben: keine StepByStep Anleitung, Kontext geben, keine Hinweise auf Label/Buttons, Formulierung als Instruktion (Sie …), Einzutragende Informationen vorgeben (aber nicht sagen wo einzutragen), keine Keywords Situation X ist eingetroffen, nutzen sie App Y um Ziel Z zu erreichen, mit folgenden Angaben: A, B und C. (**Zielorientiert**)

UC Wireframing, Prototyping, Testing

Problem Space (Analysiere User, Describe (Model), Needs/Problems)
Solution Space (Vision / Storyboard / Prototype)
Wireframes: A rough sketch of single screen
Screen-Flow: Ist gleiche wie Storyboard, Mockup, Prototype; Zeigt screen flow
Usability Test sind schwierig gut zu machen und aufwändig
Sensorisches Design: systematischen Abstimmung aller sinnlich wahrnehmbaren Designs

Standards, Prozesse, Techniken

Standard: ISO 9241-11 und Quesenbery (Effektivität, Effizienz, Zufriedenheit)
ISO 9241 – 110: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlertoleranz, Individualisierbarkeit, Lernförderlichkeit
ISO 9241 – 210 User Centered Design Process, **1.** Verstehen des Kontext der Nutzer, **2.** Anforderung Spezifizieren, **3.** Lösung machen, **4.** Evaluieren gegen Requirements (schwierigster Schritt)
Usability nach Nielsen: (1) Sichtbarkeit des System-Status **(2)** Enger Bezug zwischen System und realer Welt **(3)** Nutzerkontrolle und Freiheit **(4)** Konsistenz & Konformität mit Standards **(5)** Fehler-Vorbeugung **(6)** Besser Sichtbarkeit als Sich-erinnern-müssen **(7)** Flexibilität und Nutzungseffizienz **(8)** Ästhetik und minimalistischer Aufbau **(9)** Nutzern helfen, Fehler zu bemerken, zu diagnostizieren und zu beheben **(10)** Hilfe und Dokumentation

Internationalization

I18N: Internationalization; das Programm so zu coden, dass alle Benutzerausgaben per Spracheinstellung ausgetauscht werden können, un damit das Lokalisierung (L10N) möglich ist.
L10N: Lokalisierung; Inhalt für eine bestimmte Zielkultur angepasst werden
G11N: Globalisierung; Inhalt auf internationale Sprache vereinheitlichen, sodass alle diese verstehen und sich nicht beleidigt fühlen
T9N: Translation, **Locale**: Sprachregion
Locales String: z-.B de, de-CH, de-AT, de-DE-u-co-phonebk

ECMA2020: Hat internationalisierung
- `Intl` (Int globales Objekt)
- `Intl.Collator` (Konstruktor) -> Sprachsensitiver Stringvergleich
- `Intl.DateTimeFormat` (Konstruktor) -> Datum und Zeiten sprachsensitiv formatieren
- `Intl.ListFormat` (Konstruktor) -> Aufzählungen (und/oder verknüpft) sprachsensitiv formatieren
- `Intl.NumberFormat` (Konstruktor) -> Zahlen sprachsensitiv formatieren
- `Intl.PluralRules` (Konstruktor) -> Mit Pluralsprachregeln pluralsensitiv interpolieren
- `Intl.RelativeTimeFormat` (Konstruktor) -> Relative Zeitangaben sprachsensitiv formatieren

```
const date = new Date(Date.UTC(2019, 11, 16, 15, 30, 0));
// US Englisch: Monat-Tag-Jahr
console.log(new Intl.DateTimeFormat('en-US').format(date));
```

CSS Präprozessoren (SASS und PostCSS)

Müssen durch Präprozessor wie webpack, rollup oder parcel
CSS Präprozessoren: Sind nicht an Limitationen von CSS gebunden -> besser wartbaren Code
SASS vs SCSS: SCSS hat ; und {} ist CSS kompatibel, einafcher zu lernen, SASS Einrück Syntax

SASS / SCSS Features

Variablen($variable): Erhöht Wiederverwendbarkeit von konstanten Werten, erhöht lesbarkeit
Nesting: SCSS `nav { ul { margin:0; }}` -> CSS `nav ul {margin:0;}`
Parent Selector: `.headline { &:hover { color: blue; }}`
Import: Auslagern von Informationen, partials müssen mit Underline «_» beginnen. Partials werden von Sass nicht von CSS übersetzt `@import 'csontants'` Die Dateiendung und «_» muss beim import nicht angegeben werden.
Use: Ähnlich wie import definiert aber Namespace. Besser @use als @import benutzen
Extend/Inheritance: Mit `@extend` kann man properties von anderen Klassen erben. Wenn z.B `%icon` -> macht Scss die Styledefinition abstrakt

```
%icon { transition: background-color ease .2s;  }
.error-icon { @extend %icon; /* error specific styles... */ }
```

Mixins

Eine Art funktionale helper.

```
@mixin border-radius($radius: 1em) {
  -webkit-border-radius: $radius;
  -moz-border-radius: $radius;
  -ms-border-radius: $radius;
  border-radius: $radius;
}
.box { @include border-radius(1rem); }
```

@content: Aufruf von mixin kann Content-Block mit SCSS/CSS umfassen {…} -> kann für media queries braucbar sein.

```
@mixin breakpoint($size) {
    @media screen and (min-width: $size) {
        @content;
    }
}
@include breakpoint(30em) {
    .box { @include border-radius(0px); }
}
```

Der & Parameter: Standartisiertes Verhalten definieren: `@mixin hover() { &:hover { background: red; } }`
Mixin vs. Extends/Inheritance: Extends generiert weniger CSS code, Mixin ist flexibler: Im Zweifelsfall Mixin verwenden
Interpolation: Den Value einer Variable einsetzten z.B. p.#{$name}{}

Programmieren

Collections: $list: 1.5em 1em 0 2em, Maps: $map: (key1: value1, key2: value2)
Rechnen (wie Physik): 10px * 10px = 100px*px => «Error», 10 + px = «10px»
Loops & Coditional: Sass erlaubt Verzweigungen und Loops

```
$breakpoints: 30em 46em ; /*list*/
@each $point in $breakpoints {
  @media all and (max-width : $point) {
    body{
      @if $point > 40em { width: $width;}
      @else { width: $width * 2; }
}}}
```

Funktionen: Sass erlaubt eigene Funktionen

```
@function properZero($para){
  @if($para / (($para * 0) + 1) == 0) {
    @return $para / (($para * 0) + 1);
  } @else { @return $para; }
}}
body { width: properZero(0px) }
```

Loop Through Maps: `@each $key, $value in $map {…}`

```
@mixin color-modifiers {
  @each $name, $hex in $colors {
    &-#{$name} { color: $hex; }
}}
```