Grundlagen HTML, CSS, JS

Frontend (Mobile/Desktop-App): HTML, CSS, JS, AJAX, DOM
↕
Netzwerk (Internet): Router, Firewalls, VPN
↕
Backend (Rendering, Server-APP, REST-API): Webserver, DB

HTML: Inhalt, CSS: Styling & Layout, JS: Verhalten
- Zusammen: Dynamic HTML
- Sollte getrennt in einzelnen Files sein (.html, .css, .js)

JavaScript

ECMASkript -> Standard für Skript-Sprachen
JavaScript -> Eine Implementation von ECMA
Verarbeitung: Source (.js) -> Compiler/Intepreter (Browser)
➔ Dynamic Typed Language (Variable kann div. Typen haben)
- typeof() gibt Type der Variable zurück

Primitives: string, number, boolean, undefined, symbols, BigInt
- Compared by values, always immutable
- ***boolean***
  - false: false, 0, "", null, undefined, NaN
  - true: "0", "false", "stringd", [], {}, ...

```
3>2>1 //=> false: 1. 3>2 = true -> true=1 2. 1>1 = false
[]==![] //=> true
null == undefined //=> true
```

- ***number***
  - Alle Zahlen sind floats
  - NaN (Not A Number)
    - Error-Wert, hat aber Type «number»
    - NaN == NaN immer false (*isNaN() verwenden*)

```
"px" +1-2 //=> NaN
0/0 //=> NaN
```

  - Infinity (Unendlich, kann auch Negativ sein)

```
1/0 //=> Infinity
```

- ***string :*** Mit "text" oder 'text'
  - Template Literals:
    - String mit backticks ``
    - Inhalt innerhalb ${...} wird mit JS intepretiert
- Vergleiche
  - == (Abstract Equality Comparison Algorithm)
    - Macht Typenumwandlung nach Algorithmus
  - ===
    - Verhindert Typenumwandlung, Präferiert
- ***null/undefined***
  - null: Muss explizit zugewiesen werden, Platzhalter
  - undefined: Variable ohne Zuweisung, ! nach *let*

Objects: Alles andere : Plain Objekts, Arrays, Regular Expression, Functions
- Compared by reference, Mutable by default
- ***Arrays***

```
let arr = ['a', 99, 'b']
arr.forEach((elm, index) => {
    console.log(index +": " + elm);
})
```

  - keine Fixe Länge
- ***simple object***

```
const person ={
    name: "Roman",
    hallo: function(){return "Hallo "+this.name;} };
person.name = "Bob"; //Name ändern
person.hobby = "Tennis"; //Neues Element erstellen
console.log(person.hallo());//Hallo Bob
```

- ***Functions***
  - Können in Variablen gespeichert werden
  - Können als Parameter übergeben werden

```
function addto(a){ return function(b) {return a+b;}}
```

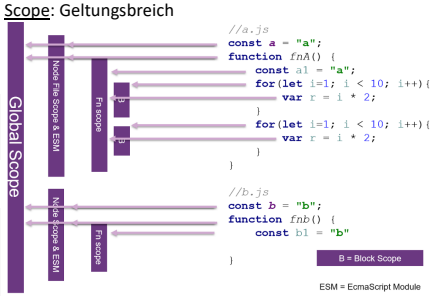
«Fucking»-Javascript (Fails sliient!):
Regeln:
- Punkt vor Strich
- Von Links nach Rechts
- Spezialfall: String + Value = String; Value + String = String
- Ansonsten: Value [*Num Operator*] Value = Number
- Var nicht verwenden, da im Global Scope, nicht Block-Scope, zugriff schon vor Definition

```
"4" / "2"                 //=> 2: Number
"4" - "2"                 //=> 2: Number
 "4" * "2"                //=> 8: Number
"4" + "2"                 //=> '42': String
10 * 3 + "px"             //=> '30px': String
8 * "1px"                 //=> NaN: Number
"px" + 1 -2               //=> NaN: Number
"3px" + 3 * 2 + "3px"     //=> '3px63px': String
"foo" ++ "abc"            //=> 'fooNaN': String
"2" - -1                  //=> 3 :Number

[] + []                   //=> '': String
[]+ {}                    //=> '[object Object]'
[] == false               //=> true
[] === false              //=> false
"" == false               //=> true
"" === false              //=> false
0 == "0"                  //=> true
0 === "0"                 //=> false
null == undefined         //=> true
null === undefined        //=> false
[1,2] == "1,2"            //=> true
[1,2] === "1,2"           //=> false
NaN == NaN                //=> false
NaN === NaN               //=> false
(1-"i") == (1-"i")        //=> false: (NaN == NaN)
[] == ![]                 //=> true
[] === ![]                //=> false
false === false           //=> true
4 === 4                   //=> true
false === true            //=> false
true + true + true == 3 //=>true
true - true               //=> 0
true == 1                 //=> true
true === 1                //=> false
[] == 0                   //=> true
a={0,1,2}
a[3] === undefined        //=> true
a[3] == null              //=> true
a[3] === null             //=> false

parseInt("4k", 10)        //=> 4
parseInt("4k", 2)         //=> NaN
parseInt("4k", 30)        //=> 140: = 4*30^1 + k*30^0
Number("4k")              //=> NaN
Number("4")               //=> 4
typeof(<Undefined>)                  //=> 'undefined'
typeof(<Null>)                       //=> 'object'
typeof(<Boolescher Wert>)            //=> 'boolean'
typeof(<Zahl>)                       //=> 'number'
typeof(<Zeichenkette>)               //=> 'string'
typeof(<Symbol>)                     //=> 'symbol'
typeof(<Funktionsobjekt>)            //=> 'function'
typeof(<Alles andere>)               //=> 'Object'
(typeof( [])[0]                      //=> 0
[1,2,3].reduce((r,v)=> (r+v*v),10)
    //=>10+1*1=11, 11+2*2=15, 15+3*3=24
```

Scope: Geltungsbreich



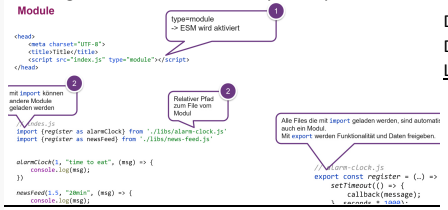
Node.js : Pro file, global.myGlVar, Browser: window.myGlVar

«this» Context: aktueller Context
- referenziert je nach dem auf anderes Objekt.
  - object.foo() ➔ this = object
  - new Foo() ➔ neu erstelltes Objekt
  - unbound ➔ globale Objekt
- Mit apply() oder call kann Context gesetzt werden

Strict Mode (use strict):
- Eliminiert « fails sliently », verbessert Security, hilft codeoptimierungen möglich zu machen

Klassen: Seit ES6, ähnlich wie in Java
- Private # (wird selten verwendet)
- keine Variablendefinition zu Beginn
- instanceof() Ob objekt instanz klasse ist
- super() Zugriff auf parent.
- Klasse ist strict: Context problme, methode als arrow

Modules: Verhindert global namespace pollution
- Hilft bei Verwendung von Libs und verhindert:
  - Reihenfolge Probleme, überschriebene Klassen
- Es gibt named und default import/export. Default 1x



HTML (Hyper Text Markup Language)

Hypertext: Verlinkung möglich
Markup: Inhalte werden semantisch annotiert
Language: Definierter Syntax

Parsing:
- *Load HTML:* File mit HTML-Markup laden, Encoding!!!
- *Parse HTML*: Tags/Elemente Verarbeiten
- *DOM:* Elemente, Attr. werden zu Nodes im Dom-Baum
- *Display:* Browser stellt Elemente mit CSS dar

Empty-Element: Kein Inhalt und End-Tag, nur Slash (img

Struktur eines HTML-Dokuments

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  Hello World
</body>
</html>
```

doctype
html: Root-Element
- z.B. Sprache (für Übersetzer/Screen-Reader)

head: Metadaten
- Nicht daraestellt im Browser
- Nur ein Element erlaubt

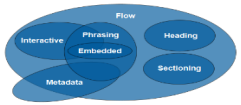
body: Sichtbarer Inhalt der Webseite
- Nur ein body-Element erlaubt

Content-Model:
- ... definiert Gruppen, welche auf HTML-Elemente angewendet werden. Diese definieren das Verhalten und die Regeln der Elemente.

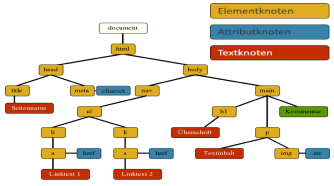
Tag-Omission: Minimales HTML, nicht empfohlen
Semantik: Wichtig unter anderem für Screenreader

Kategorie	Element	Semantik
Heading	<h1> - <h6>	Überschriften; h1 höchste Stufe
Flow	<p>	Absatz
	<blockquote>	Zitat
	 	Semantischer Umbruch
		Gruppiert andere Elemente. Keine semantische Bedeutung
Phrasing	<b>	"Bring Attention To": Wichtig für Benutzer:in
	<code>	Programm-Code
	<img>	Bild
	, , <mark>, <cite>, <dfn>	: Hervorhebung anzeigen : Starke Hervorhebung anzeigen <mark>: Relevanz indizieren <cite> Name eines Werks, oder Buch markieren <dfn>: Definition eines Begriffs markieren
		Keine semantische Bedeutung

Section-Elemente: body, article, aside, nav, section
- Können header, main, footer, h1 besitzen
- Aus Headings wird Outline generiert



DOM (Document Object Model)



DOM Representiert HTML-Dokument als Baumstruktur
DOM hat Methoden fürs Traversieren und Manipulieren

Lifecycle
- **Interactive**: Dokument geladen/geparst.
  - Ressourcen fehlen noch (Skripte, Bilder, Style)
  - Event: DOMContentLoaded
  - Warten?
    - Libaries oder Skripte, die nicht interagieren
    - Eigene Skripte eher am Ende von body
- **Complete**: Alles Geladen
  - Event: load
  - Warten?
    - Wenn wegen Grösse von Elemente Aktionen ausgelöst werden müssen. (Overlay bei Bilder)
- *async*: sofortiges asynchrones laden und ausführen.
- *defer*: Ausführen erst nach parsen des Dokument

globale Browser Objekte:
- *window*
  - Globale Variablen liegen auf dem Window
  - Stellt alle anderen globalen Objekte zu Verfügung (console, history, document,...)
- *document*
  - Einstiegspunkt für den DOM-Tree
  - Bietet DOM-Such-Methoden und DOM-Manipulations-Methoden an

Selektion/(Suche):
- Klassisch: document.getElements...
  - gibt kein Array zurück sondern eine HTMLCollection
  - kein forEach, map, filter
  - Erzeugt Live-Query (dym. Update)
  - Loop mit for(elmt of htmlcollection){...}

```
document.getElementById('mainID');
document.getElementsByClassName('mainClass');
document.getElementsByName();
document.getElementsByTagName('li');
```

- Neu: document.querySelector
  - Mit CSS Selektoren

```
document.querySelector('#mainID');
//erste gefundene Element im Sub-Baum
document.querySelectorAll('.mainClass');// alle gefundenen Elemente im Sub-Baum als NodeList
document.closest('li'); //First parent
document.matches('next'); //boolean
```

  - NodeList definiert forEach
    - kein map, filter
    - Loop mit for(elmt of nodelist){...}

DOM Navigation

```
<node>.parentElement
<node>.childNodes //Auch text-Nodes und Kommentare
<node>.children // Nur Nodes vom Typ HTMLElement
<node>.firstChild //erste Node,auch Txt und Kommentare
<node>.firstElementChild // erstes HTMLElement
<node>.nextSibling
<node>.nextElementSibling
```

NodeTypen:
- EventTarget:
  - addEventListener(); removeEventListener();
- Node Interface:
  - childNodes
  - firstChild
  - nextSibling
  - appendChild()
  - removeChild()
- Element Interface
  - id / className / classList / innerHTML /
  - getAttributes() / setAttribute() / toggleAttribute()
  - querySelector() / querySelectorAll() / ...
  - scrollTo()
  - Parent Node:
    - children / firstElementChild /lastElementChild
    - append() / remove()

```
const newEl = document.creatElement('div');
newEl.appendChild(document.createTextNode('Hello'));
document.getElementById('mainId').appendChild(newEl);
```

innerHTML:
- Liest und schreibt den Inhalt vom Element als HTML-Code

```
const c = document.getElementById('mainId');
console.log(c.innerHTML);
c.innerHTML = '<div>Neuer HTML Inhalt</div>';
```

innerText:
- Beinhaltet nur sichtbarer Text

style:
- Besser als direktes Setzen des style-Attr als String

```
const c = document.getElementById('mainId');
c.style.background = "yellow";
```

Event:
- Mehrere registrierte listener für ein Event Möglich
- Bubbling: Event durchläuft DOM-Tree. So kann jedes Element reagieren. So Listener auf übergeordnetes Event möglich.

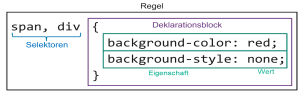
```
const eventFunction = () => {
document.querySelector('container')
.appendChild(document.creatTextNode("Hello World"));}
const btn = document.querySelector('#meinBtn');
btn.addEventListener('click', eventFunction);
```

Data-*:
- HTML Elemente haben fest definierte Attr
- Benutzerdefinierte Attr -> data-...
- Daten Austausch zwischen HTML und Javascript
  - .getAttribute() / setAttribute()

CSS (Cascading Style Sheet)

Wiederverwendbares/austauschbares Styling
Cascading: Regel wirkt von oben nach unten

Inline: HTML-Property style
Style-Tag im Header (nur testen): <style>...</style>
Separates File: <link rel="stylesheet" href="file.css">



Kommentare: /** **/

Selektor	Alle Elemente	Universal
div	Elemente von Typ	Typ
.class	Elemente mit Klasse	Klassen
#id	Elemente mit ID	ID
div a	Alle a-Elemente in . div	
div > a	Nur a-Elemente mit direkten div-Eltern	

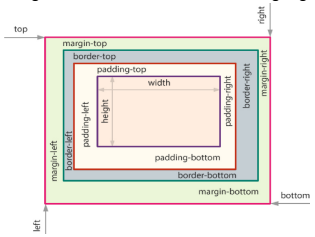
div ~ a	Alle nachfolgenden geschwister-Elemente a, des div-Elements	
div, a	Alle Elemente welche a oder div sind	
div[href="..."]	Alle Elemente mit diesem Attribut	Attribut
div::before	Empty-Element direkt vor den Kindern des div	Pseudo Element
div::after	Empty-Element direkt nach den Kindern des div	Pseudo Element
a:hover	a-Elemente, welche «gehovert» wurden	Pseudo Klasse
input:required	Inputs, welche required sind	Pseudo Klasse
div:first-child div:nth-child(2n) div:nth-child(even)	Kinder des div, welche die Bedingung erfüllen	Pseudo Klasse

Spezifität-Algorithmus

- Vier Zähler (A, B, C, D), Startwert 0
  - A++ inline-Styles
  - B++ ID-selektoren
  - C++ Klassen-selektoren, Pseudo-Klasse, Attribute
  - D++ Typ-Selektoren und Pseudoelemente
- Zuletzt definierte Regel gilt falls gleiche Spezifität
- !important überschreibt andere Deklarationen

Box-Model

- Background wird auf Contend und Padding angewendet.

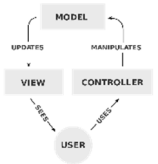


- box-sizing:default (height&width = content)
- box-sizing: border-box (height &width = content+padding+border)

Layouting: ~~Tabellen/Floats~~, **FlexBox**, CSS-Grid

- display: flex oder display: inline-flex
- flex-direction: row (oder column) [X bzw. Y-Achse]
- justify-content: center (flex-start, flex-end, ...) Anordnung auf Main-Axis
- align-items: center (etc.) Anordung auf Cross-Axis
- align-self (Einzelne elemente anpassen)
- flex-grow, flex-shrink, flex-basis, flex-wrap

MVC (GUI Archidektur)



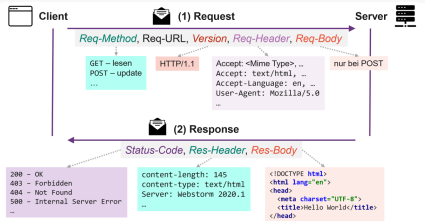
- *Model*: Data Model, Business Logic, keine View Refs
- *Views*: Anzeige
- *Controller*: User Input, View Wechsel

Clean Code

- *Name*: Kurz, Intuitiv, Beschreibend, Aussprechbar
  - Funktionen: Verb
  - JS: camelCase
  - CSS: kebab-case
- Funktionen sollen kurz sein
- Linter hilft CleanCode einzuhalten (ESLint, Stylelint)

Server / Client

```
import http from 'http';
const PORT = 8080;
function requestHandler(req, res) {
    if (req.url === '/favicon.ico') {res.end();
return;}
    console.log('url: ', req.url);
    console.log(req.headers);
    res.write('<h1>Hello ');
    res.end('World</h1>');
}
const server = http.createServer();
server.on('request', requestHandler);
server.listen(PORT,() => console.log('Node listening
on Port ', PORT));
```



Method	Description
GET	The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
HEAD	Same as GET but transfers the status line and header section only.
POST	A POST request is used to send data to the server, for example, new customer information, file upload, etc. using HTML forms.
PUT	Replaces all current representations of the target resource with the uploaded content.
DELETE	Removes all current representations of the target resource given by a URI.
CONNECT	Establishes a tunnel to the server identified by a given URI.
OPTION	Describes the communication options for the target resource.
TRACE	Performs a message loop-back test along the path to the target resource.

Code	
1xx	Informational
2xx	Successful 200 OK 201 Created 204 No Content
3xx	Redirection 301 Moved Permanently
4xx	Client Error 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found
5xx	Server Error 500 Internal Server Error 505 HTTP Version Not Supported
9xx	Non-Standard Codes

Ajax (Asynchronous JavaScript and XML)

Vorteile:

- Interaktive Kommunikation mit Server
- Dynamische Seiten möglich
- Reduzierter Traffic-Verbrauch (Nur notwendiges)

Nachteile:

- Nutzer ohne JS sehen Daten nicht (10% der Nutzer)
- Suchmaschinen sehen Daten nicht
- Zurück-Funktion, Loading-Indicators aufwendiger

Callback-Funktionen: asyncrone Antworten

- ***Callback***: Einfacher

```
async function doHomework(subject, callback){
  alert('Starting my ${subject} homework.');
  await callback();
}
function alertFinished(){
  alert('Finished my homework');
}
doHomework('math', alertFinished);
```

- ***Promise***: (Liefert Promise-Wert)
  - Komplexer, chaining möglich
  - Stellt Abschluss oder Misserfolg einer asynchronen Operationen und deren Wert dar

```
var hans = new Promise(function(resolve, reject) {
  let x = "Hello";
  let y = "World";
   if(x === y) {
       resolve("x=y");
    } else {
        reject("x!=y");
    }
});
hans.then(function (successMsg) {
    console.log(successMsg); //Output: x=y
    }).catch(function (errorMsg) {
        console.log(errorMsg); //Output: x!=y
    });
```

```
Argument: Executor-Funktion: Fn(resolveFn, rejectFn) => void
new Promise((resolve, reject) =>
    // Synchronous setup possible here
    >>>async fn<<<(..., (...callbackArgs) => {   } Zu "wrappende" async-Funktion aufrufen
    if (error) {
        reject(error);                           } Aufruf von reject (wo sinnvoll)
    } else {
        resolve(value);                          } Aufruf von resolve mit Ergebnis
    }
})
```

Status: *pending*: aktiv, *fulfilled*: erfolgreich, *rejected*: gescheitert

Callbacks: .then, .catch oder await.

Fetch

API für HTTP-Requests (Ersetzt XLMHttpRequest)

Fehlerbehandlung (then): .catch

```
fetch(fetchURL)
  .then((response) => {
    if (response.ok) {
      return response.json();
    } else {
      return Promise.reject();
    }
  })
  .then((data) => {
    console.log('thenData', data);
  }).catch((error) => {
    console.log('thenError')
  });
```

Fehlerbehandlung (await) try/catch-Block

```
async function test(fetchURL) {
  try {
    const response = await fetch(fetchURL);
    if (!response.ok) {
      throw "awaitError not ok";
    }
    const data = await response.json();
    console.log('awaitData', data);
  } catch {
    console.log('awaitError');
  }
}
```

```
// Umgang mit URLS
let url = new URL('/ressource', 'https://myapi.ch');
url.search = new URLSearchParams({some:1,params: 2});
fetch(url);
```

Ajax Probleme und deren Lösung

Langsamer API-Service

- View in suspense-Status setzen
- UI-Elemente deaktivieren, Feedback geben (Ladeanimation, aktualisierende Werte entfernen)

Unzuverlässiger API-Service

- Request wiederholen (Max Anzahl!)
- Zeitlimit (Timeout):

```
const controller = new AbortController();
const signal = controller.signal;
setTimeout(() => controller.abort(), 5000);
fetch(url, { signal }).then(response => …
```

- Fehlermeldung bei leerer Antwort

Langsamer API-Service bei kont. Nutzerinteraktivität

- Debouncing / Trhotelling (Warten und erst nach bestimmter Zeit, wenn User gerade nichts eingibt Request senden)
- Ersetzen von Requests (preempt): abbrechen!

Polling (Server Push)

- Updates vom Server holen

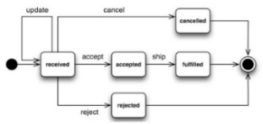
```
setInterval(updateChatFromServer, 10000);
```

- Sockets verwenden (3rd Pary Libary)

REST (Repersentational State Transfer)

Richardson's Maturity Model:

- *Level 0 – The Swamp of POX:*
  - Datenstruktur
- *Level 1 – Ressourcen:*
  - Siehe ROA
  - Stellt die Ressourcen in den Mittelpunkt
  - Ressource immer in Mehrzahl
  - Ressourcendarstellung, was kann die Ressource
    - Welche Verbs werden unterstützt?
    - /customers/12/orders/1
    - orders/1
- *Level 2 – HTTP Verbs:*
  - saubere Umsetzung der Verbs
  - Verbs:
    - GET -> Ressource wird angefordert
    - POST -> erzeugt eine neue Ressource
    - PUT -> aktualisiert / erzeugt eine Ressource
    - DELETE -> löscht eine Ressource
    - HEAD -> gleich wie GET jedoch ohne die Ressource zu erhalten
      ➔ Nur Liste der Ressourcen, bzw. Header
    - OPTIONS -> wie darf eine Ressource verwendet werden
    - PATCH -> partielles Updaten einer Ressource
- *Level 3 – Hypermedia Controls:*
  - Wie sind die Ressourcen zusammen gelinkt
  - Hateoas (Hypermedia as The Engine of Application State):
    - REST-Eigenschaft
    - REST-Client nur durch das Folgen von URI im Hypermedia-Format durch die Webanwendung bewegen kann dadurch lässt sich die Schnittstelle eines REST-Services jederzeit anpassen
  - Beschreibung des Ablaufs:



ROA (Resource Oriented Architecture): Grundprinzip

- Ressource
  - Alles was genug wichtig ist, um referenziert zu werden
- Ressource Name
  - Eindeutige ID der Ressource
  - in Mehrzahl
  - sind Nomen
  - REST benutzt URI
    - Beispiel: orders/1 oder books/0-330-25864-8
- Ressource Repräsentation (mehrere)
- Ressource Links
  - Benutze Hyperlinks als Verknüpfung der "Dinge"
- Ressource Interface
  - Uniform Interface / Benutze Standard-methoden
- Statuslose Kommunikation

Grundlagen Web-Design, Usability

Heuristiken (Daumenregeln): Es gibt nicht DIE Lösung

Augenführung:

- nur einen kleinen "scharfen Fleck" (Fovea)
- durch Augenbewegung verschoben (Fixation)
- leicht Informationen im GUI zu übersehen
- Aufmerksamkeit kann gelenkt werden
  - Position, Eye Catcher, Bewegung
- Standardreihenfolge: Oben -> Unten, Links -> Rechts
- Ablauf:
  - Scannen
    - Akzente setzen (Headlines, Schlüsselwörter/-Bilder, Links/Suchhilfen)
  - Skimmen
    - Zusammenfassen (kurze Absätze, Listen/Tabellen/Grafiken,...)
  - Lesen
    - In die Details gehen (längerer Fliesstext, Druckformat)
- Eye Catcher können Reihenfolge brechen
  - hell /dunkel, einzeln/gruppiert, Grafik/Text, Farbe/SW, gross/klein, Bewegungen, Augen & Blickrichtung, ...
- Change Blindness: Tendenz Veränderung übersehen
- Meldungen immer beim Feld

Affordances: Greifbarkeit (von UI Elementen)

- Gibson: Natürliche Wahrnehmung
- Norman: Wargenohmene affordance (Tochscreen)
- Visuelles Design & Interaktionsdesign bei Controls
- gutes Interaktionsdesign
  - Zeigt den aktuellen Zustand
  - Zeigt dem Nutzer Möglichkeiten an
  - Die Sprache des Nutzers
  - Begleitet den Nutzer Schritt für Schritt
- Interaktionselemente sollten...
  - ...grundsätzlich sichtbar sein (Pixel on Screen)
  - ... vom Nutzer wahrgenommen werden
  - ... vom Nutzer zielführend interpretiert werden
  - .. vom Nutzer genutzt werden können

Constraints: Einschränkungen setzen, Konventionen einhalten (Maus kann Bildschirm nicht verlassen)

Visuelles Design:

- Farbe:
  - Farbblindheit beachten
  - Kontrast beachten
  - Galitz (2002)
    - GUI Design in Grau, Farbe zu Akzentuierung
    - Nicht mehr als 4 Farben
  - Psychologische & Kulturelle Aspekte beachten
    - Rot ist nicht immer gleich "Stop" und "Achtung"
    - Grün ist nicht immer gleich "Gut" und "Weiter"

Nielsen-Heuristiken:

1. **Sichtbarkeit des Systemstatus**
   - Zustand und Interaktionsmöglichkeit anzeigen
2. **Übereinstimmung zwischen System und realer Welt**
   - Begleitet Nutzer durch den Prozess, Keine Ablenkung
3. **Freiheit und Kontrolle der Nutzer:in**
   - Undo statt Sicherheitsabfragen
4. **Konsistenz und Standards**
5. **Fehlervermeidung**
6. **Wiedererkennen statt erinnern**
7. **Flexibilität und effiziente Nutzung**
   - Touch-Target 1cm2: *0.9cm + 0.2cm padding*
8. **Ästhetik und minimalistische Gestaltung**
9. **Hilfe beim Erkennen und Beheben von Fehlern**
10. **Hilfe und Dokumentation**

Beispiele JS

Counterfunction:

```
function makeCounterFn(start){
  return _ => {return start++;};
}
const fn = makeCounterFn(10);
console.log(fn()); //10
console.log(fn()); //11
console.log(fn()); //12
```

Counterobjekt:

```
function makeCounterObject(start){
  return {getNext: _ => start++};
}
const obj = makeCounterObject(20);
console.log(obj.getNext()); //20
console.log(obj.getNext()); //21
console.log(obj.getNext()); //22
```

Counterklasse:

```
class Counter {
  constructor(start){this.counter=start;}
  getNext(){return this.counter++;}
}
const obj = new Counter(30);
console.log(obj.getNext()); //30
console.log(obj.getNext()); //31
console.log(obj.getNext()); //32
```

StepCounterklasse:

```
class StepCounter extends Counter{
  constructor(start, step=1){
    super(start);
    this.step = step;
  }
  getNext() {
    const before = this.counter;
    this.counter += this.step;
    return before;
  }
}
```