# Webprog Aufschriebe

# Klausur

- Wieviele Selektoren müssen gelernt werden ?
- Boolean Fragen was zu false evaluiert

```
var y = 2;
console.log(y);//2
//Hier aufpassen. Y ist kein boolscher Datentyp. Deshalb output false
(true == y) ? console.log(true) : console.log(false); //Output: false, bei 1 true
```

- Code muss interpretiert werden
- Code muss selber geschrieben
- Theoretische Fragen

# HTML5

## Attribute

<img src=”mein-bild.jpg” alt=”Alternativer Text”\>

## Semantische Elemente HTML5

Früher: Division-Elemente  (War sehr unübersichtlich)

- <header>: Kopfbereich
- <footer>: Fußbereich
- <main>: Hauptinhalt
- <section>: Themenbezogene Abschnitte
- <article>: Wiederkehrende Blöcke
- <aside>: Zusatzinformationen an der Seitenleiste
- <nav>: Interne Navigationselemente
- <address>: Kontaktinformationen des Autors

## Webseite validieren

https://validator.w3.org/

**Beispiel:**

```
<!doctype html>
<html lang=”de”>
        <head>
                <meta charset=”UTF-8”>
                <title>Titel</title>
        </head>
        <body>
        </body>
</html>
```

# CSS Einführung

## Versionen

CSS 1: 1996

CSS 2: 1998
CSS 3: Living Standard (Beständige Weiterentwicklung), Aufteilung in Module

Viele Browser unterstützen neue Features die noch nicht im Standard sind.

# CleanCode

- Bücher: Clean Code
- Coder lesen Code hauptsächlich

# Aufbau

| HTML | CSS |
|---|---|
| <!-- index.html →<br><head><br>.<br></head><br><body><br><h1> | /* style.css */<br>h1<br>{<br>color:white;<br>text-align: center;<br>font-family: “Arial”;<br>background: blue;<br>} |
| | h1{color:blue;} /* Regel*/<br>**h1**: Selektor<br>**color**: Eigenschaft<br>**blue**: Wert<br>**Alles in geschweiften Klammern**: Deklaration |
| | h1,h2{...} |

# Verlinkung

## Internes Stylesheet

```
<head>
<style type = “text/css”>
        body{margin: 2px}
</style>
```

</head>

### Externes Stylesheet

<link rel = “stylesheet” type = “text/css” href=”style.css”>

### Inlinestyle

<header style=”background: blue; padding: 1px;”>

## Priorität

1. Inline
2. Intern
3. Extern

## Selektoren

### Typ oder Elementselektoren

h1,h2,h3 {...} /* Es werden mehrere Elemente ausgewählt */

### Klassenselektoren

**HTML**

Error .. 
<p class=”fehlertext hinweis”>Zwei Klassen

Klassen werden verwendet wenn man mehrere Elemente gruppieren will

**CSS**

```
.fehlertext{color:red}        //Klassenselektor
p.fehlertext{color:red}       //Klassenselektor
p{color:red}                  //Typ / Elementselektor. Alle p Elemente werden selektiert
```

### ID Selektor

**HTML**

 .. 

**CSS**

#footer{...}
div#footer{...}

### Universalselektor

Alle Elemente ansprechen
*{...}

## Kombinatoren

### Eltern Kind Beziehung

header - h1
main - article
article - p
p - h2,p,p
footer - p

### Geschwister

header - main
main - footer

### Nachfahrenselektor (E F)

Alle Kinder und Kindeskinder
main p{...} /* Alle P sind rot*/

### Kindselektor (E > F)

Alle Kinder aber keine Kindeskinder
main > p{..} /* Keins wird ausgewählt */

### Nachbarselektor (E + F)

Nur Elemente auf derselben Ebene die das selbe Elternelement haben, die direkt nebeneinander lieben.
h2 + p{..} /* Nur p welches nach h2 kommt */

### Geschwisterselektor (E~F)

Nur Elemente auf der selben Ebene bezogen auf das Elternelement. Müssen nicht direkt nebeneinander liegen.
h2~p{..} /* Alle p Geschwister von h2 welche nach h2 folgen*/

## Pseudoklassen

a:link {...} Alle Links werden selektiert. Nur links die noch nicht geklickt wurden
a:visited {...} Alle besuchten Links werden hier ausgewählt
p:hover {...} P Element wird beim drüberfahren formatiert
p:active {...} Hier werden Elemente formatiert die gerade angeklickt werden

# Webarchitekturen

## Serverseite Programmierung

- Render wird von Server durchgeführt und Seite ausgeliefert (HTML, CSS, JS)
- Multi-Page-Application
- Applikationsstatus wird auf Server gespeichert (Session)
- HTML hat keinen Status. Kann sich nichts merken
- Durch Cookies wird die Session ID abgespeichert
- Cookie wird lokal gespeichert und bei gleicher Kontaktierung des Servers wieder mitgeschickt
- PHP 78%, Ruby 2,4%, Java (JSP, JSF) 4% , ASP.Net (Active Server Pages) 12%
- Seite wird geladen -> Klick -> HTTP Request werden Daten zurückgeschickt (page2.html, CSS, JS)

## Clientseitige Web Programmierung

- Anwendung wird geladen und Inhalte werden nachgeladen (Ajax)
- Single Page Application (SPA)
- Seite wird geladen -> Klick -> ggf HTTP Request (Json Daten)
- Daten werden asynchron nach und nach geladen (Ajax)
- Server ist Datenlieferant
- Applikationslogik und -status auf Client
- Flüssige Abbildung komplexer Benutzerinteraktionen
- Gefühl nativer Anwendung

# Einführung JavaScript

## Geschichte

- Standardisiert als ECMA-262, EcmaScript(ES)
- 1995 erste prototyp. Version “Livescript”, “Mocha”
- Wurde dann umbenannt in JavaScript (Marketinggründe)
- 1996: Einbindung in Netscape

- Microsoft: JScript für Internet Explorer
- 1999 “Baseline”
- 2005: Ajax-Hype (Asynchronous JavaScript and XML(heute JSON))
- JQvery entsteht
- 2008: Chrome-Browser
- 2010: ES5
- 2015: ES6

## Frameworks

- Angular
- Vue
- React

## Charakterisierung

- Leichtgewichtig
- Interpretiert
- Dynamisch
- First Class Funktionen (Echte Objekte)
- Objektorientiert und prototypbasiert (vs. klassenbasiert)

## Code Beispiel

```
<head>
	<script>
		alert(“Hallo Welt”);
	</script>
</head>
```

```
<body>
	<script src=”hallo.js”></script>
</body>
```

## Variablen Deklaration

```
var user = “Max Mustermann”; //String (Typ) automatisch ermittelt
let pi = 3.14; //Number
var sitzplatz = 12; //Number
var laenge; //undefined
laenge = 0;
```

```
var x = 3.2, y = 4.0, z = 8;
var Laenge = 0; // Laenge ungleich laenge
laenge = “test”;
laenge = {vorname: “Max”}; //Typ Object
```

# Sichtbarkeitsbereiche (Scopes)

## lokaler Scope innerhalb einer Funktion)

```
//code can not use carName
function myFunc()
{
        var carName = “Volvo;
        //code can use carName
}
//code can not use carName
```

## globaler Scope

```
var carName =”Volvo”;
function myFunc()
{
        carName=”x”;
}
```

## Variable ohne Deklaration: Globale Variable

```
function myFunc()
{
        carName = “Volvo”; //globale Variable (hier deklariert ohne var. Ist im Window Objekt)
        myfunc2(); //globale Funktion
}
```

Gilt auch bei Funktionen !!!

## Problematik

Globale Variable oder Funktionen können unbeabsichtigt **Variablen** und **Funktionen** des Window-Objekts überschreiben

# Hoisting (“Deklaration heben”)

```
<!DOCTYPE html>
   <head>
       <script type="text/javascript">
           function Hoisting()
```

```
            {
                ausgabe = "text";                   //Kann vor Deklaration benutzt werden
                document.write(ausgabe)             //Hier wird "text" ausgegeben

                document.write(test)          //Hier wird 4 zurückgegeben. Außerhalb Funktion deklariert
                var ausgabe;                        //Deklaration wird gehoisted

                document.write(undefiniert)         //Initialisierungen werden nicht gehoisted
                var undefiniert = "undefinedText" //Nur Deklarationen werden gehoisted
            }
            var test = 4                //Hier wird auch Initialiserung für innere Sichtbarkeit gehoisted



        </script>
        <body>
            <form name = "myForm" action="">
                <input type="button" value = "Hoisting" onclick="Hoisting()"/>
            </form>
        </body>
    </head>
</html>
```

# Block Scope mit Let (Seit ES6)

```
<!DOCTYPE html>
    <head>
        <script type="text/javascript">
            function blockScopeLet()
            {
                var i;
                for (i = 0; i<10; i++)
                {
                    let y = i;
                }
                if(i==10)
                {
                    var test = 5;
                    if(test == 5)
                    {
                        var test 2 = 3;
                    }
                }
                console.log(i)  //Es wird 10 ausgegeben
                console.log(y)  //ReferenceError: y is not defined
                console.log(test) //5
                console.log(test2)//3
            }

        </script>
        <body>
            <form name = "myForm" action="">
                <input type="button" value = "blockScopeLet" onclick="blockScopeLet()"/>
```

```
            </form>
        </body>
    </head>
</html>
```

## Konstante

```
<!DOCTYPE html>
    <head>
        <script type="text/javascript">
            function konstante()
            {
                const PI = 3.14;
                PI = 2; //TypeError: invalid assignment to const `PI'

                var PI_2 = 3.14
                PI_2 = 2;
                console.log(PI_2) //Kein Fehler da mit var deklariert
                //BEI LET UND CONST ENTFÄLLT HOISTING
            }
        </script>
        <body>
            <form name = "myForm" action="">
                <input type="button" value = "konstante" onclick="konstante()"/>
            </form>
        </body>
    </head>
</html>
```

## Strikter Modus

```
<!DOCTYPE html>
    <head>
        <script type="text/javascript">
            "use strict";
            function strikterModus()
            {
                var myNum = 10;
                myNum2 = 20;      //ReferenceError: assignment to undeclared variable
myNum2

                strikterModus2();
            }
            function strikterModus2()
            {
                console.log(myNum)//Ohne/mit "use strict"nicht ausführbar da mit var
dekl.

                console.log(myNum2)//Wäre ohne "use strict" ausführbar da global
            }
        </script>
        <body>
```

```
            <form name = "myForm" action="">
                <input type="button" value = "strikterModus" onclick="strikterModus()"/>
            </form>
        </body>
    </head>
</html>
```

# Datentypen

## String

```
function string()
{
    let string1 = 2;
    let string2 = 'test';
    let string3 = string2 + 2
    console.log(typeof string1); //Output: number
    console.log(typeof string2); //Output: string
    console.log(typeof string3); //Output: string
    console.log(string3) //Output: test2
}
```

## Undefined vs. null

Variable der noch kein Wert zugewiesen wurde, ist undefined.
Bei Zuweisung von null wird Wert einer Variable gelöscht

```
function undefinedVar()
{
    if(typeof(notExistingVariable) === "undefined")
    {
        console.log("Test") //Output: Test
        console.log(notExistingVariable)
        //Output: ReferenceError: notExistingVariable is not defined
    }
}
```

## Objekte

- Auflistung von Eigenschaften und Methoden
- Enthalten Properties als Name Wert Paare

```
function objekte()
{
    let otherObj =
```

```
    {
        Nachname: "Gonzales"
    }
    let obj =
    {
        vorname: "Rachel",
        adress: {Strasse: "Bergstr",Plz: "88252"},
        alter: 22,
        ref: otherObj
    };
    console.log(obj.vorname);   //Output: Rachel
    console.log(obj["vorname"]) //Output: Rachel (Alternative)
    console.log(obj["adress"]["Plz"]) //88252
    console.log(obj.adress.Plz) //88252

    obj.vorname = "Maria";      //vorname wird überschrieben
    console.log(obj.vorname)    //Output: Maria

    obj.function = function()
    {
        return "x";
    };
    console.log(obj.function()); //Output: x

    delete obj.function;
    console.log(obj.function); //undefined
}
```

## Number

Entspricht double in Java

```
function number()
{
    let i = 1;
    let y = 1.0;
    let int = parseInt("100.1"); //100
    let float = parseFloat("100.1"); //100,1
    let num1 = new Number("5");
    let num2 = Number("5");
    console.log(num1); //Output: Number { 5 }
    console.log(num2) //Output: 5
    console.log(1/0); //Output: Infinity
    let x = 1/0;
    console.log(x); //Output: Infinity
    console.log(0/"a"); //Output: NaN
    console.log(typeof(0/0)); //Output: number
    console.log(typeof(0/"a")); //Output: number
    console.log(0/0); //Output: NaN
    //Nan und Infinity sind vom Typ number. Man kann damit weiterrechnen
```

```
}
```

## Boolean

```
function boolean()
{
    let boolean = true;
    let alter = 16;
    let volljaehrig = (alter>=15);
    //console.log(volljaehrig); //Output: true
    var y = 2;
    console.log(y);//2
    //Hier aufpassen. Komische Werte
    (true == y) ? console.log(true) : console.log(false); //Output: false,
bei 1 true
    console.log(y);//2
    if(y)
        console.log("Wahr");//Hier wird das wahr
    else
        console.log("falsch");

    var age = 1;
    let accessAllowed = age == true ? true : false;
    //console.log(accessAllowed) //Output: true
}
```

Alles evaluiert zu true, außer:

- 0
- false
- Null
- null
- undefined
- “ “ //leerer String
- Infinity
- **Alle Zahlen außer 1 ?**

# Java Script Grundlagen

## Array

```
    let brands1 = ["vs","Toyota"];
    let brands2 = brands1[1]
    console.log(brands1) //[ "vs", "Toyota" ]
    console.log(brands2) //Toyota
```

```
brands1[1]="new";
console.log(brands1) //[ "vs", "new" ]

brands1.push(20)
console.log(brands1) //[ "vs", "new", 20 ]

brands1.splice(1,2)//Schneidet ab Index 1 zwei Elemente ab
console.log(brands1) //[ "vs" ]
```

## == (Equals) vs. === (StrictEquals)

- Zwei Vergleichsoperatoren in JS
- Nur bei striktem Vergleich erfolgt Vergleich von Datentypen andernfalls erfolgt automatische **Typkonvertierung**

```
console.log(false == 0) //true
console.log(false === 0) //false da Prüfung auf Datentyp

console.log("1234" == 1234) //true
console.log("1234" === 1234) //false da Prüfung auf Datentyp

console.log(false == 1) //false
console.log(false === 1) //false da Prüfung auf Datentyp

console.log(true != 0) //true
console.log(true !== 0) //true


console.log(true == 1) //true
console.log(true === 1) //false da Prüfung auf Datentyp

var bool = (18>5)
console.log(true === bool) //true
```

## Weird Loop

```
function array()
{
    let num = [1,2,3];
    parseInt(num)
    let sum1 = 0;
    let sum2 = 0;

    //Output: 0012
    for(let index in num)
    {
        sum1 = sum1 + index
        console.log(sum1)
    }
```

```
            console.log(sum1)

            //Normal loop
            for(let index in num)
            {
                sum2 = sum2 + num[index]
            }
            console.log(sum2)
        }
```

# Exceptions

```
<!DOCTYPE html>
<html>
<body>
<p>Please input a number between 5 and 10:</p>
<!-- Folgende Zeile stellt das Input Feld dar -->
<input id="inputfield" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="errortext"></p>

<script>
function myFunction()
{
   var message, x;
   message = document.getElementById("errortext");
   message.innerHTML = "";

   //Hier wird das Input Feld ausgelesen
   x = document.getElementById("inputfield").value;
   try
   {
       //Falls Feld leer ist wird "is empty string" an catch gegeben
       if(x == "")  throw "is empty";
       //Falls keine Zahl eingegeben wurde, wird string an catch gegeben
       if(isNaN(x)) throw "is not a number";
       x = Number(x);
       //Prüft einen Zahlenbereich
       if(x > 10)   throw "is too high";
       if(x < 5)  throw "is too low";
   }
   //Parameter wird mit "throw" übergeben
   catch(err)
   {
       message.innerHTML = "Input " + err;
   }
   //Wird immer ausgeführt
   finally
   {
       document.getElementById("inputfield").value = "Finally text";
       x.value = "test"
   }
```

```
}
</script>
</body>
</html>
```

# Funktionen (“Bürger erster Klasse”)

```
function func()
{
    //Annonyme Funktion
    let myFunc1 = function(){console.log("myFunc1 aufgerufen")};
    myFunc1();

    //Optionale Parameter
    function multiply(a,b=2)
    {
        return a*b;
    }
    console.log(multiply(2)) //4

    //Variable Parameteranzahl kann bei Aufruf der Funktion übergeben werden
    function variableParameteranzahl()
    {
        let sum = 0;
        for(index in arguments)
        {
            sum+=arguments[index];
        }
        return sum;
    }
    console.log(variableParameteranzahl(1,2,3));//6

    //Arrow Funktionen - ein Parameter
    let einParameter = val => val*2
    console.log(einParameter(2)) //4

    //Arrow Funktionen - mehrere Parameter
    let mehrereParameter = (param1,param2)=>param1+param2;
    console.log(mehrereParameter(3,2)) //Output 5

    //Arrow Funktion - mehrere Anweisungen
    let mehrereAnweisungen = msg =>
    {
        console.log("Ausgabe");
        console.log(msg);
    }
    mehrereAnweisungen("StringParameter");

    //Funktion ohne Parameter
```

```
                let ohneParameter = () => console.log("String in Funktion");
                ohneParameter()

                //Funktion höherer Ordnung
                //Für jedes Element in arr wird function(e) aufgerufen
                let arr = [1,2,3,4];
                arr.forEach(function(e)
                {
                    console.log(e)
                });

                //Alle ungeraden Zahlen eines Arrays ausgeben
                arr.filter(function(e)
                {
                return e %2 !== 0;
                }).forEach(function(e)
                {
                    console.log(e);
                });//1,3

                //Hier werden alle Wörter die eine Länge von > als 6 Zeichen haben
ausgegeben
                //Die Funktion filter() erstellt ein neues Array
                const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction',
'present'];
                const result = words.filter(word1 => word1.length > 6);
                console.log(result);[ "exuberant", "destruction", "present" ]

                //Neues Array mit Potenzen erzeugen + Array aufsummieren(Summieren fängt
bei 10 an)
                let sum = arr.map(function(e)
                {
                    return e*e;
                }).reduce(function(reduziert,element)
                {
                    return reduziert+element;
                },10);
                console.log(sum)
                //Kurzschreibweise
                //.reduce( (prev, curr) => prev + curr );
            }
```

# DOM-Manipulation

- DOM erlaubt mittels JS Zugriff auf HTML Elemente, Attribute, CSS-Styles
- Auf HTML-Ereignisse kann reagiert werden

## Knotentypen

Wichtigste der 12 Knotentypen

- Dokumenten Knoten (Wurzel DOM-Baum), globales Objekt “document”
- Elementknoten repräsentieren HTML-Elemente
- Attributknoten, z.B. src, href
- Textknoten repräsentieren Text innerhalb von HTML-Elementen

Inhalt

p(Elementknoten)

title = “Absatz”(Attributknoten) Inhalt(Textknoten)

E l e m e n t k n o t e n

## Elemente selektieren

**Id Selektion**

```
<p id = “begruessung>...</p>
<script>
    let element = document.getElementById(“begruessung”);
    if(element)
    {
        element.innerHTML+=”test”;
    }
```

**Andere Selektionsarten**

document.getElementsByClassName(“Klasse”);
document.getElementsByTagName()
document.getElementsByClassName()

Beispiel Code welcher auf Knopfdruck einen den zweiten Listeneintrag (Tea) in das innere HTML des letzten p Elementes schreibt.

```
<!DOCTYPE html>
<html>
<body>

<p>An unordered list:</p>
<ul>
 <li>Coffee</li>
 <li>Tea</li>
 <li>Milk</li>
</ul>

<p>Click the button to display the innerHTML of the second li element (index 1).</p>

<button onclick="myFunction()">Try it</button>
```

```
<p id="demo"></p>

<script>
function myFunction() {
 var x = document.getElementsByTagName("LI");
 document.getElementById("demo").innerHTML = x[1].innerHTML;
}
</script>

</body>
</html>
```

**CSS Selektion**

//Gibt zweites td Element das in table vorkommt zurück
let tableCell = document.querySelector('table td:nth-child(2)')
Leerzeichen im CSS: Nachfahrenselektor

## Verwandschaftsbeziehungen

body parent
h1 p child
sibling ←→ sibling

//Gib mir den Elternknoten des Tables aus. Worin befindet sich mein Table. Z.B um Hintergrund der Tabelle einzufärben
let table = document.querySelector('table');
console.log(table.parentNode);

- Anstatt parentNode kann auch firstChild, lastChild, childNodes oder andere eingesetzt werden

## Textinhalt verändern:

element.textContent = 'neuer Text';
HTML in Element verändern: element.innerHTML = 'Hello';

- **innerHTML** parses content as HTML, so it takes longer.
- **nodeValue** uses straight text, does not parse HTML, and is faster.
- **textContent** uses straight text, does not parse HTML, and is faster.
- **innerText** Takes styles into consideration. It won't get hidden text for instance.

## Textknoten hinzufügen:

let element = document.getElementById(‘container’);
//In container neuen Text einfügen
let textNode = document.createTextNode(‘Beispielsknoten’);
element.appendChild(textNode);

| Beispiel |
| --- |

```
<body>

<p>Click the button to create a Text Node.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
 var t = document.createTextNode("Hello World");
 document.body.appendChild(t);
}
</script>

</body>
```

## Attribute bearbeiten

**Attribute abfragen**
console.log(element.getAttribute(‘href’));
//Kurzschreibweise
console.log(element.href)

**Attribute setzen**
//Link wird in neuer Seite geöffnet
element.setAttribute(‘target’,’_blank’);

## CSS-Klassen bearbeiten

**JavaScript Code:**
element.classList.add(‘active’);
element.classList.remove(‘active’);
**CSS Code:**
.active{color: red}

Hier wird der CSS Code aktiviert.
Änderungen werden sofort gerendert da auf dem DOM gearbeitet wird!

# Event-driven Programming

Event-Emmitter >> Event >> EventQueue >> EventLoop >> Event-handler

### Realisierung als HTML-Event-Handler

<button onclick = ‘changeColor()”>Farbe ändern</button>

<script>
    function changeColor(){...}
</script>

Nachteil:

- Vermischung von JS und HTML
- Event Handler kann nicht deaktiviert werden

### Realisierung als Eigenschaft eines Objekts(DOM-Event-Handler)

let element = document.getElementById(‘submitButton’);
element.onmouseover = function(){..}
element.onmouseover = null;

### Realisierung mit addEventListener(DOM-Event-Listener

**HTML Code**
element.addEventListener(“click”, changeColor); //ist mehrfach möglich
element.removeEventListener(“click”, changecolor); //changecolor ist hier Funktion
**JS Code**
function changeColor(event){ .. }

### Event-Fluss(Event-Propagation)

3 Phasen

1. Capturing-Phase
2. Target-Phase
3. Bubbling-Phase

Beispiel:
<article onclick = “changeColor(this)”>
     Hier Klicken! 
</article>

- Registrieren von Event-Listenern erfolgt standardmäßig immer für die Bubblingphase
- Zeck ist das die Funktion nur einmal implementiert werden muss. Wird für jedes Element in article aktiviert
- In Methode addEventListener können Events für Capture-Phase registriert werden

Beispiel Event Capturing Targeting Bubbling

Hier werden eventHandler für die Capturing wie für die Bubbling Phase registriert. Bei einem Klick auf ein Element, werden erst die Capturing Handler von Oben bis zum Ziel (Hier das geklickte Element) abgearbeitet und dann die Bubbling Handler vom Ziel bis nach oben noch ein zweites Mal.

```
<style>
   body * {
     margin: 10px;
     border: 1px solid blue;
   }
 </style>
  <form>FORM
   <div>DIV
     <p>P</p>
   </div>
 </form>
  <script>
   for(let elem of document.querySelectorAll('*')) {
     elem.addEventListener("click", e => alert(`Capturing: ${elem.tagName}`), true);
     elem.addEventListener("click", e => alert(`Bubbling: ${elem.tagName}`));
   }
 </script>
```

**Wichtige Funktionen:**
stopPropagation(); // Event wird nicht mehr nach oben weitergereicht
preventDefault(); //verhindert Standardaktion des Browsers, z.B. das eine Seite bei einem Link geöffnet wird

# JQuery

Weltweit am meisten verbreitete JS-Bibliothek
Vorteile:

- Vereinfachten Zugriff auf DOM Elemente
- Vereinfachten Umgang mit Events durch Hilfsmethoden
- Einfachere Formulieren von Ajax-Anfragen

## Einbindung

Kann mit Script eingebunden werden
CDN: Content distribution networks (Link von JQuery Webseite holen und in HTML einfügen)
Beispiel: <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

## Verwendung

Funktion jQuery(), Shortcut-FKT $()

3 Formen des Aufrufs oder möglicher Parameter:

- Aufruf mit den Selektoren
  let elements = $('body>div');
- Aufruf mit Knoten des DOM-Baums
  let element = document.createElement("p");
  $(element).text("Inhalt"); //verändert Inhalt von Element
  $(document).ready(()=>{
  console.log("Webseite geladen"); });
- Aufruf mit HTML String
- let new Element = $(' ... '); //neues div Element wird hier erzeugt

# Inhalte und Attribute abfragen und verändern

$("#box").append(knoten);
$("#box").prepend ...
$("#box").after ...
$("#box")remove …

let txt = $("#id01").text();
let txt = $("#id01").html();

<input type = "text"></input>
…
$('input')j.val("Danke für die Eingaben");

let alterlink = $("a").attr("href"); //Link des attr. href wird zurückgegeben
$("a").attr("href","http://…"); //Link wird geändert

# Wiederholung

- Event-Fluss
- Dom Event Handler auf einzelnen Elementen
- Dom Event Listener

# CSS manipulieren

## Jquery

$(".hinweis").addClass("error")
$(".hinweis").removeClass("error")
$(".hinweis").toggleClass("error")

$("div").css("background-color","blue")

### CSS

.error{color:red}

## Effekte und Animationen

$(“div”).hide(1000); //optional[ms]
$(“div”).show(1000); //optional[ms]
$(“div”).toggle(1000); //optional[ms]
$(“div”).fadeIn(1000); //optional[ms]
$(“div”).fadeOut(1000); //optional[ms]

$(“div”).animate({
left:”300px”},2000); //funktioniert nur mit position: relative/absoute/fixed (CSS)

## Operationen verketten

$(“p”).text(“neuer Text”).addClass(“highlight”);

Auf Benutzeraktionen reagieren
$(“a”).click(function{
let addr = $(this).attr.(“href”);});

## NPM (Node Package Manager)

- populärster Paketverwalter für JS
- Node.js serverseitige Platform zum Betrieb von Netzwerkanwendungen (Laufzeitumgebung V8)

### Installieren

Node.js und NPM herunterladen

### Setup

- npm init
- Name der Seite eingeben
- Entry point (index.js)
- Package Json wird erstellt

### Kommandos

- npm search jquery

- npm install jquery (Ordner node modules wird angelegt. Datei lock.json. Unter dependencies wurde jquery hinzugefügt)
- npm update

### Ordner und Dateien

“^3.41” => Version verwenden >= 3.4.1 < 4.0.0
package json Mindestanforderungen
package.lock konkrete Info
Node.modules: jquery stuff

### Allgemein

- Man kann html Datei anlegen und auf Ordner zugreifen
- nur package.lock.json und package json wird eingecheckt für git
- AppData/Roaming/npm (Unterschied lokal global)
- Modulepacker packt alles auf eine Datei zusammen

# Datenformate

## XML

- Extensible Markup language (vom W3C entworfen)
- Dateiendung xml

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- XML Spezifikation enthält wenige formale Regeln für den Aufbau eines Dokumentes
- Wenn Regeln befolgt werden dann wohlgeformt
- Mittels XML Schemas/Document Type Definitions (DTD können Standards für XML Dokumente definiert werden.

### Zeichenketten in XML Objekte umwandeln

```
let xmlString = ‘<?xml version="1.0" encoding="UTF-8"?>’ + ‘<note>’+’</note>’;
let domParse = new DOMParser;
let xmlDOM = domParse.parseFromString(xmlString,’text/xml’);
```

let firstArtist = xmlDOM.querySelector('note');

## XML Objekte in Zeichenketten umwandeln

let xmlSerializer = newXMLSerializer();
let xmlStringSerialized = xmlSerializer.serilizeToString(xmlDom);

# JSON (Java Script Object Notation)

- Nachfolger von XML
- Syntax
    - keine Tags
    - geringerer Overhead
- bis auf wenige Ausnahmen ist JSON valides Java Script
- Dateiendung: .json
- Unterschied zu JavaScript: Es werden Anführungszeichen bei den Key Value Paaren benötigt

Json nested Example

```
<!DOCTYPE html>
<html>
<body>
<p>How to access nested JSON objects.</p>
<p id="car"></p>
<p id="name"></p>
<p id="albumsYear"></p>
<p id="moviesYear"></p>
<script>
var myObj = {
 "name":"John",
 "age":30,
 "cars":
 {
 "car1":"Ford",
 "car2":"BMW",
 "car3":"Fiat"
 }
}
var obj =
{
   "artists":
   [
       {
           "name": "Maria",
           "albums":
           [
```

```
                {
                    "title": "Una vez",
                    "year": 1920
                }
            ],
            "movies":
            [
                {
                    "title": "Once upon a time",
                    "year": 1925
                }
            ]

        }
    ]
}
//document.getElementById("car").innerHTML += myObj.cars.car2 + "<br>";
//or:
document.getElementById("car").innerHTML += myObj.cars["car2"];
document.getElementById("name").innerHTML += obj.artists[0].name; //Maria
document.getElementById("albumsYear").innerHTML += obj.artists[0].albums[0].year; //1920
document.getElementById("moviesYear").innerHTML += obj.artists[0].movies[0].year; //1925
console.log(obj.artists[0].name) //Maria
</script>
</body>
</html>
```

## JS nach JSON Objekte umwandeln

let string = JSON.stringify(obj,(key,value) => {return value},2);

```
//Conversion from JS to Json
var JStoJson = { name: "John", age: 30, city: "New York" };
var JsonObj = JSON.stringify(JStoJson);
console.log(JsonObj)
```

## JSON nach JS Objekt umwandeln

let objectParsed = JSON.parse(string,(key,value) =>{return value};
console.log(objectParsed.artists

```
// Here we convert JSON to object
var jsonobj ='{ "name":"Brendan Eich","designerof":"Javascript","bornin":"1961" }';
var objJS = JSON.parse(jsonobj);
console.log(objJS)
```

# Projekt

- API New York Times verwenden
- Top Stories anzeigen

- Axios - Library zur abfrage
- JSON Format wird von NYT zurückgegeben

# Rest

- Representational State Transfer
- Programmierparadigma für verteilte Systeme
- Nutzung zur M2M (Machine to Machine) - Kommunikation
- Übergang von einem Status in den nächsten
- Neuerung: HTTP-Methoden in Verbindung mit URI verwenden
- curl kann verwendet werden

## Grundprinzipien

Rest besteht aus 5 Kernprinzipien:

1. **Ressourcen mit eindeutiger Identifikation (URL)**
    - URI (ugs. Ressourcen / Endpunkte (wichtig wenn man selbst REST Endpunkte bauen will))
    - Beispiele:
      http://example.com/customers/1234
      http://example.com/orders/ //Liste aller Bestellungen
      http://example.com/orders/2018 //Liste aller Bestellungen aus 2018

2. **Standardmethoden**
    - Alle Ressourcen müssen das Protokoll HTTP implementieren
      Nutzung der HTTP Operationen entsprechend ihrer Bedeutung / Spezifikation
    - HTTP-Operationen(Methoden,Verben):
      GET,POST,PUT,DELETE,HEAD,OPTIONS

3. **Statuslose Kommunikation**
    - Server speichert keine clientspezifischen Status über die Dauer des Requests hinweg

4. **Unterschiedliche Repräsentation**
    - Bereitstellung der Ressourcen in unterschiedlicher Repräsentation
    - Client kann auf Basis von HTTP content negotiation ein bestimmtes Format anfordern
    - GET / customers/1234 HTTP/1.1
      Host: example.com
      Accept: application/xml
    - Accept: text/x-vcard

5. **Hypermedia**
   Über Verknüpfungen (links) können Beziehungen zwischen Ressourcen hergestellt werden und Applikationsfluss kann gesteuert werden
   Beispiel für JSON-Rückgabeobjekt:

```
{       "amount": 19,
        "customer:
        {"href": "http://example.com/customers/1234"},
        "product":
        {"href": "http://example.com/products/12445"},
        "cancel":
        {"href": "./cancellations"}
}
```

# HTTP - Operationen

(Verben / Methoden)

| | safe (Keine Speicherung auf Server) | idempotent (Wenn Operation mehrfach hintereinander ausgeführt wird ist Ergebnis immer gleich) |
|---|---|---|
| GET | 1 | 1 |
| POST | 0 | 0 |
| PUT | 0 | 1 (1 oder 10 mal Kunde wird immer auf gleiche Weise geändert) |
| DELETE | 0 | 1 (1 oder 10 mal ist immer gelöscht) |
| HEAD | 1 | 1 |
| OPTIONS | 1 | 1 |

GET: Rückgabe von Daten

POST: Hinzufügen einer Ressource (z.B neuen Kunden speichern)
Als Ergebnis soll Link der neuen Ressource zurückgegeben
Wird auch verwendet für Operationen, die sonst nicht abgebildet werden

PUT: Anlage einer Ressource. Wenn bereits Ressource existiert wird diese geändert

DELETE: Löscht Ressource

HEAD: Fordert Metadaten von einer Ressource an für zum Beispiel:

- Prüfung ob Ressource existiert
- Ermittlung des Zeitunk der letzten Änderung
-

OPTIONS: Prüft, welche Methoden auf dieser Ressource zur Verfügung stehen

# API Design

**Nicht erwünschte Beispiele:**
/getAlleEmployees
/addNewEmployee
/updateEmployee
/deleteEmployee
/getOneEmployee

**Besser:**
GET/employees => Liste aller MA
GET/employees/246 => MA mit ID 246
DELETE/employee/246 => Lösche MA mit ID 246
POST/employee/ => MA updaten oder anlegen falls noch nicht vorhanden

**Empfohlen:**
- URL sollte nur Ressourcen (Substantive) beinhalten, keine Namen Verben (Aktionen)
- Abfrage von einzelner Ressource über ID
- Geeignete Status Order zurückgeben
    - 200 OK
    - 201 CREATED
    - 204 NO CONTENT z.B. nach DELETE
    - 304 Not modified (Client hat response noch im Cache. Keine Änderung)
    - 400 Bad Request
    - 401 Unauthorized
    - 405 Method not allowed
    - 500 Internal Server Error
    - 503 Server unavailable

## Sortierung, Filterung, Suche, Pagination

GET/employees?sort = name_asc (Aufsteigend u. Name sortieren)
GET/employees?role=manager&location =stuttgart (Filterung)
GET/employees?search=MaxMuster
GET/employees?page=3

## Versionierung

http://api.example.com/v1/employees

### Ressourcen unter anderen Ressourcen

GET/employee/246/books
DELETE/employee/246/books/13479

### Zustände in Webapplikationen

| | | |
|---|---|---|
| Client | | Server |
| | --Request—> | Session mit ID speichern in DB |
| Cookie | <--Response— | |
| Speicherung | --Cookie—> | Cookie Überprüfung in DB |

## Cookies

Textdateien die auf Client gespeichert werden und Wiedererkennung des Nutzers erlauben

### Wichtige Daten

- Name und Wert des Cookies
- Domäne des Servers und Pfad
- Ablaufdatum

### js-cookie

<script src =" ">
cookies.set(“spielstand”,23);
let zähler = cookies.get(“spielstand”);

### Webstorage Schnittstelle

Wenn Zustand von Webanwendung nur Clientseitig benötigt wird.

localStorage.setItem(“spielstand”,23);
localStorage.getItem
localStorage.removeItem

### Nachteile von Cookies

- Privatsphäre
- Datenvolumen
- Einschränkung der Skalierbarkeit der Anwendung

- Falls ein Load balance Server eingesetzt wird welcher wieder auf verschiedene Server verzweigt, muss eine Anfrage immer auf den gleichen Server weitergeleitet werden.

## Zustände bei REST

Client ——-> Server
< ——

- Zustandslos (keine Cookies)
- REST-basierte Server lassen sich gut skalieren

# Ajax

Ajax (Asynchronous JS and XML)

- 2005 A new approach to Web Applications

**Herkömmlicher Aufbau (vor 2005)**
Client Server

betrachtet Webseite Suchanfrage >>

Wartet

<<Ergebnis

betrachtet Ergebnisse

Detailansicht>>

Wartet

**Ajax (Asynchrone Kommunikation)**
Client Server

Suchanfrage 1-10>>
Weitere Anfragen >>

<<Ergebnis kommt

Detailansicht>>

- Es werden nur Teile der Webseiten neu geladen

**Vorteile**

- Fortlaufende Interaktion möglich
- Bessere Nutzerfreundlichkeit, Bedienbarkeit
- kürzere Übertragungszeit

### Funktionsweise

- ursprünglich XML-Datenformat
- kombinierbar mit anderen Formaten (JSON)

```
‘use strict’;
let request = new XMLHttpRequest();
request.onload = (event)=>
{
        console.log(‘Antwort geladen’);
}

request.open(‘GET’,content/data.xml’);
request.setRequestHeader(‘Accept’,’text/xml’);
request.send();
```

## Websockets

Unidirektionale Kommunikation mit dem Server (Standardfall)

- Client stellt Anfrage

### Polling

- Client fragt periodisch Informationen ab

### Long-Polling

- Client fragt zyklisch an aber Server antwortet erst wenn neue Daten da sind.
- HTTP Verbindung bleibt offen bis neue Daten vorliegen

### Bidirektionale Kommunikation

- Client öffnet Socketverbindung
- Server antwortet auf bestehender Verbindung sobald neue Daten vorhanden sind

```
let connection = new WebSocket(‘ws:/example.com/test’);
connection.onopen = function(e){connection.send(‘Nachricht an Server’);}
connection.onerror = function(e){ .. }
connection.onclose = …
connection.onmessage = function(e)
```

```
{
        console.log('Data:' + e.data);
        let data = JSON.parse(e.data);
}
```

## Server - Sent - Events (SSE)

- Immer wenn es neue Daten gibt, dann werden diese vom Server geschickt
- Hier kann nur der Server Daten heraus schicken
- Alles was mit SSE umgesetzt werden kann, kann auch mit Websockets erreicht werden
- News Ticker ist eine typische Applikation für SSE

```
let source = new EventSource('/events');
source.onmessage = (e) =>
{
        console.log(e.data);
        console.log(e.origin);
        console.log(e.lastEventId);
}
source.addEventListener('message',(e) =>{ .. });
```

# Node.js

Node.js ist eine serverseitige Plattform in der Softwareentwicklung zum Betrieb von Netzwerkanwendungen. Insbesondere lassen sich Webserver damit realisieren.

- Ryan Dall (2005)
- Vorteile v J
    - etabliert in Web
    - leistungsstarke Engines
    - große Zahl von Entwickler
- jsConf.ev => Sponsor gefunden
- Meilensteine
- 2011 native Windows Unterstützung
- NPM fester Bestandteil von node.js
- 2013 Stream - API
- 2014 Kritik von Community
  => keine stabile Version
  => Fork io.js
- 2015: Node.js-Foundation (Stiftung zuständig für Weiterentwicklung)
- Transparenter Releaseplan

## Vorteile von node.js

- Kern standardisiert
- Gute Dokumentation
- schnelle performante Engine (V8-Engine v. Google Chrome)
- Wettbewerber ⇔ verbindliche Standards

## Technische Rahmendaten

- Reines JS, kein Sprachdialekt
- Single Threaded Ansatz
- Nebenläufigkeit über nonblocking I/O
    - Auslagerung von Ein und Ausgabefunktionen an das Betriebssystem
    - Im Thread läuft Event-Loop
    - Callback-Funktionen werden sequenziell ausgeführt wenn asynchrone Operation beendet ist.
- V8-Engine
    - Einlesen des Quellcodes
    - Optimierung. Übersetzen des Codes in Maschinencode
    - Ausführen der Applikation auf Basis des Maschinencodes
    - Just in Time Kompilierung (J/T)

## Debugging

- node inspect server.js
- Breakpoints z.B in Code mit “debugger”

## Interne Module

Komponenten, die sich auf Erledigung nur einer Aufgabe konzentrieren

## JS-Modulsysteme

- Ziel. Applikation unterteilen
- AMD (Asynchronous Modul Definition)
    - Fokus: clientseitiges JS
    - Schlüsselwörter: require/define
- Common JS
    - Fokus auf serverseitigem JS
      Datei add.js
      module.exports = function(a,b){return a+b;}
      Datei index.js
      const add = require(“./add”); //Datei befindet sich im gleichen Ordner

```
const result = add(1,2);
```

- EcmaScript-Standard
    - neuer Standard auch von Node.js unterstützt
    - noch im experimentellen Status

### Beispiel serverseitiger Webserver

- Laden des Moduls “http”
- Rückgabe von Header / Body (HTTP)

### Express

- populärste Web-Applikation-Framework in Node.js
- Komponenten

**Router** ->Steuert Zuordnung von URL zu Funktionen

**Middleware** ->Funktionen die zwischen Anfrage und Antwort aufgerufen werden (Daten formatiert oder alles logged)

**http-Modul** -> response request Objekts
->Serverprozess

```
const express = require (“express”);
function log(req,res,next)
{
	console.log(req.url);
	next();
}
app.use(log);
app.get(“/”,function(req,res)
{
	code …
});
```

# Serverseitige Programmierung

Serverseitige Programmierung mit Django
REST-Server mit express

## Webserver mit Node.js

- Laden des HTTP-Moduls mittels require
- HTTP-Body versenden mit response.write oder mit response.end
- Bei PHP wird jede Anfrage separat bedient
    - JS/Node: Antwort verbleibt im RAM

### Express

- populärstes Web-Applikation Framework in Node.JS
- Komponente

| | |
|---|---|
| Router | Steuert, welche Funktionen aufgerufen werden soll, abhängig von URL |
| Middleware | Funktionen zwischen Anfrage und Antwort |
| http-Modul | Erstellt Server process |
| | Bereitstellung request response - Objekts |

### Middleware

- Beliebig viele Funktionen hintereinander aufrufen

```
const express = require(“express”)
const app = express();
function log(req,res,next){
console.log(reyq.url);
next(); //ruft nächste Middlewarefunktion auf
app.use(log);
app.use(express.json()); //Parsen von übertragenen Daten
```

# SPA mit Vue.js (Single Page Application)

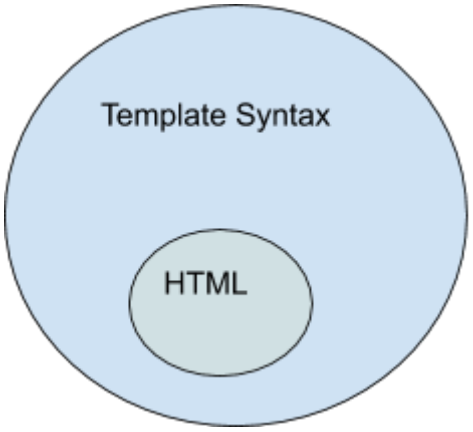
Aktuelle SPA Frameworks:

- Vue (Evan You - Google Developer)
- React (Facebook)
- Angular
    - Wird als Platform gesehen
    - Google

    - Typescript (Typen für Variablen. Wird nach JS transpiliert)

## Wichtige Features

- Mächtige Template Syntax




Superset von HTML

- DOM Manipulation
- Reatkion auf Events
- Reaktivität (Datenänderungen werden automatisch an Templates weitergegeben)

## JQuery vs Vue.js

- Vue.js- Umsetzung wesentlich kürzer
- Logik der APP ist komplett getrennt von der Darstellungslogik

## Installation und Anwendung

- CDN
- für großformatige Anwendungen: NPM
  npm install vue //nicht installiert werden weitere sinnvolle Module wie eslint (es = ecma Script)
  (Linter: Werkzeug zur statischen Codeanalyse)

## Vue-Cli

- CLI: Command-Line-Interface (sollte installiert werden)
- Features:
    - Vorhandene Build-Setups => Prototyping
    - hot-reload
    - Lint-on-save
    - production-ready builds
- npm install -g @vue/cli (@ = NPM Scope, gemeinsame Pakete)

=> Kommandozeile: vue -version

## NPM-Operation

- vue create / vue vi (Grafische Oberfläche)
- npm run serve => Kompilierung+hot-deployment
- npm run build => Kompilieren, minifizieren für die Produktion => dist Ordner
- npm run test
- npm run lint

## Grundlagen

```
var vm = new Vue(
{
        //option-Objekt (Verhalten kann variiert oder durchgeführt werden)
});
```

Templates, Daten, Direktiven

- Darstellung von Daten auf Seite mittels Templates
- Gewöhnliche HTML Code wird angereichert durch Attribute => sogenannten Direktiven

| Vue Code |
| --- |

```
<script src="..."></script> //CDN
<div id = "app">
        <p v-if="isMorning">Good morning</p>//Falls direktive nicht erkannt wird kann Sprachmodus ausgestellt werden.
        <p v-if="isEvening">Good evening</p>
</div>
<script>
//Direktiven
//Kompletter Div Teil: Template

        var hours = new Date().getHours();
new Vue({
        el:"#app"; //el: Element, #app = id, an dieser Stelle soll VueObjekt angehängt
werden
        data: {
                        isMorning: hours < 12;
                        isEvening: hours >= 12
                }
});
</script>
```

## Interpolation

```
<div id = “app”>
        <p v-if = “path ===”/””>Homepage</p>
        <p v-else> You are on{{path}}</p> //Doppelte geschweifte Klammern: Interpolation
</div>
<script>
        new Vue
        ({
                        el: “#app,
                        data: {path: location.pathname} //Options-Obj
        });
</script>
```

# Array in Vue verwenden mittels Interpolation

```
<div id = “app”>
          <p> The 2nd brand is {{brands[1]}}</p>
</div>

<script
          new Vue
          ({
                    el:”#app”,
                    data:
                    {
                              brands.[“VW”,”BMW]
                    }
          });
</scripts
```

# v-if vs v-show

**Beispiel 1:**

```
<div -v-if=”true”> one </div>
<div v-if=”false”>two</div>
```

**Führt zu:**

```
=> <div>one</div>
```

- Iteration über Dom
- Schlechte Performance

**Beispiel 2**

```
<div v-show=”true> one</div>
<div v-show=”false”>two</div>
```

**Führt zu:**

```
<div>one</div>
```

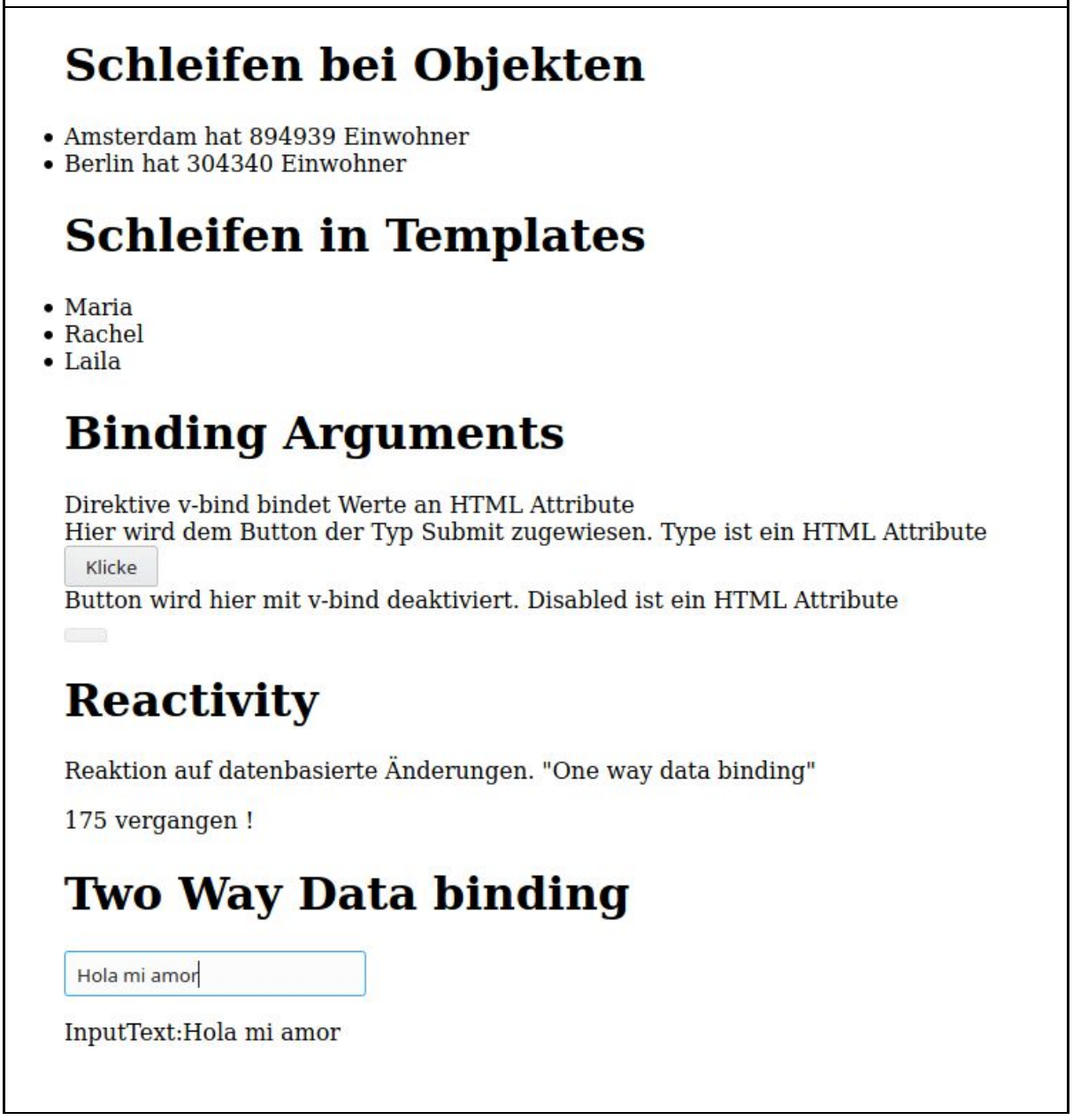
two

- Wenn Elemente sich häufig ändern
- Wenn Element ein Bild enthält (Wird vorgeladen)

# Schleifen, Binding, Reactivity, TwoWayBinding

```
<!DOCTYPE html>
   <head>
       <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
   </head>
   <div id ="app">
       <ul>
           <h1>Schleifen bei Objekten</h1>
           <li v-for= "(residents,city) in residentsOfCities">{{city}} hat {{residents}}
Einwohner</li>

           <h1>Schleifen in Templates</h1>
           <li v-for= "x in models">{{x}}</li>

           <h1>Binding Arguments</h1>
           Direktive v-bind bindet Werte an HTML Attribute<br>
           Hier wird dem Button der Typ Submit zugewiesen. Type ist ein HTML
Attribute<br>
           <button v-bind:type="buttonType">Klicke</button><br>
           Button wird hier mit v-bind deaktiviert. Disabled ist ein HTML Attribute<br>
           <button v-bind:disabled="isButtonDisabled"></button>

           <h1>Reactivity</h1>
           Reaktion auf datenbasierte Änderungen. "One way data binding"<br>
           <p>{{seconds}} vergangen !</p>

           <h1>Two Way Data binding</h1>
           <input type="text" v-model="inputText">
           <p>InputText:{{inputText}}</p>
       </ul>
   </div>
   <script>
       // Erstellung des Vue Objektes
       new Vue(
           {
               el:'#app',// Zugriff auf bestimmtes Div Element
               data:// Erstellung der Daten
               {
                   residentsOfCities:
                   {
                       Amsterdam: 894939,//Amsterdam = key, Zahl = value
                       Berlin: 304340
                   },
                   models:["Maria", "Rachel", "Laila"],
                   buttonType: "submit",
```

```
                    isButtonDisabled:true,
                    seconds:0,
                    inputText:"I love you" //default Text
                },
                created()
                {
                    setInterval(I=>{this.seconds++;},1000)
                }
            });
</script>
</html>
```




## Komponenten

Komponenteneigenschaften:

- eigenständiger Code der Teil einer Seite repräsentiert
- eigenes JS, eigene Daten, eigenes Styling
- können andere Komponenten beinhalten

Vorteile:

- Wiederverwendbarkeit
- Keine Seiteneffekte

## Unterschied Komponente <-> Vue-Instanz