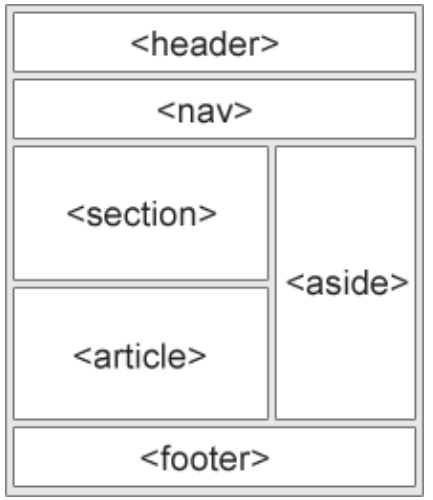
Web Application Development

JavaScript, DOM, AJAX

Martina Freundorfer

Letztes Mal

- Layout von HTML-Seiten
  - HTML 3.x:
    - Einfaches Layout, Paragraphen, Listen, etc.
  - HTML 4.x und XHTML:
    - Nutzung von <table>, <tr>, <td> - Elementen zur Formatierung des Inhalts
    - Nutzung von <frameset>, <frame>-Elementen
  - Besser: , , CSS
  - HTML 5: Neue strukturierende HTML-Elemente:



- Wozu CSS?

  Mit HTML werden Inhalte strukturiert.

  Mit CSS wird der strukturierte Inhalt formatiert.

https://www.w3schools.com/quiztest/quiztest.asp?qtest=CSS

Semesterplan

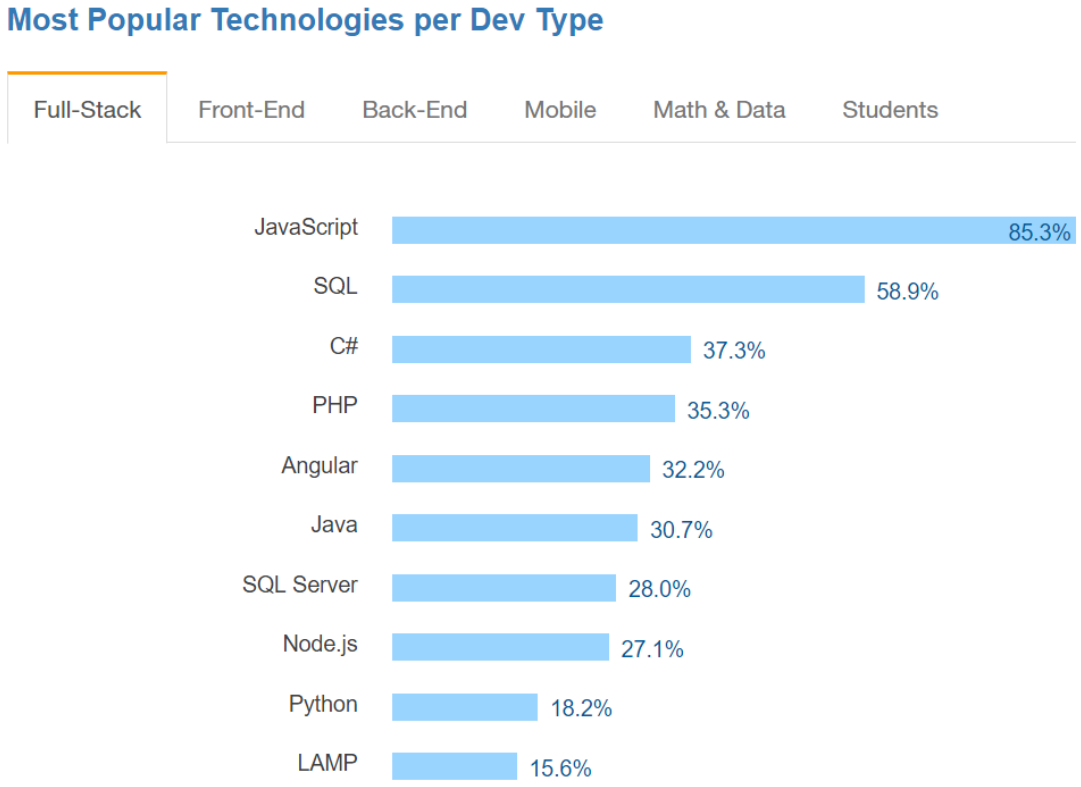
#	KW	Datum	Vorlesung	**Das Labor nach der Vorlesung findet für beide Gruppen wöchentlich statt.**
1	14	3.4.	Einführung, Scheinkriterien	npm im Selbststudium, Anlegen eines Webprojektes, Installation von Live Server
2	16	17.4.	Client-Server, Web Apps, URI, HTTP	Aufgabe von Beleg 1: AdViz (nur HTML und CSS)
3	17	24.4.	HTML, CSS	Prototyp AdViz
4	20	15.5.	CSS, JavaScript allgemein	Abgabe: Prototyp AdViz
5	22	29.5.	JavaScript: DOM, JSON, AJAX	Aufgabe von Beleg 2: Adviz mit JS
6	23	5.6.	React Framework	AdViz mit JS
7	24	12.6.	React Framework	AdViz mit JS
8	25	19.6.	React Framework	Abgabe: AdViz mit JS
9	26	26.6.	React Framework / NodeJS	Aufgabe von Beleg 3: AdViz mit Backend
10	27	3.7.	NodeJS	AdViz mit Backend

Heute

- JavaScript
- DOM
- JSON
- AJAX/XMLHttpRequest
- Lernziele:
  - Sie kennen die Grundlagen von JavaScript und können kleine Skripte in JsFiddle implementieren und ausführen.
  - Sie wissen, wie Sie HTML-Elemente mit Hilfe von JavaScript manipulieren können.
  - Sie verstehen AJAX und können mit Hilfe von XMLHttpRequest eine Anfrage an einen Server schicken.

JavaScript

- Eine der populärsten Programmiersprachen der Welt
- **DIE** Skriptsprache des Webs
- https://www.tiobe.com/tiobe-index/ : Platz 7 (letztes Jahr noch Platz 8)
- Andere sehen JavaScript auf Platz 1



- https://www.quora.com/Why-is-JavaScript-so-popular

JavaScript

- 1995 von Brendan Eich für Netscape entwickelt (er brauchte bloß 10 Tage für den Prototyp)
- Ursprünglich “Mocha”, später “LiveScript”
- 4. Dezember 1995: Kooperation von Netscape und Sun Microsystems mit Ziel einer Interaktion von LiveScript mit Java-Applets:
  - Sun entwickelte die nötigen Java-Klassen,
  - Netscape entwicklte Schnittstelle ”LiveConnect”,
  - Sprache in *JavaScript* umbenannt (Marketing-Gründe)
- April 1996: JavaScript 1.1 (Netscape)
- Mai 1996: *JScript* in Microsofts IE Browserkrieg
- Juni 1997: Netscape und ECMA (European Computer Manufacturers Association) entwickeln einen Standard: *ECMAScript*
- Erst 1.10.1997 MicroSoft’s IE deckt JavaScript 1.1 ab, aber implementiert auch Spracherweiterungen, welche dann zu Kompatibilitätsprobleme zwischen Netscape Navigator und IE führten
- JavaScript ist seit der Übernahme von Sun Microsystems eine Marke des Unternehmens Oracle

- **Aktuelle Version: ECMAScript 2019**
- Zukünftige Version: ECMAScript 2020
- ECMAScript-Unterstützung von Browsern:
  http://kangax.github.io/compat-table/es6/

JavaScript ist …

- Interpretiert
- Single-threaded, Skript wird in einem einzigen Thread ausgeführt (Achtung: manchmal ”unresponsive script”)
- Schwach bzw. dynamisch typisiert: Zuweisung von Variablen unterliegt keinen typbasierten Einschränkungen, Datentyp einer Variable wird zur Laufzeit durch den derzeitigen Wert der Variable festgelegt, mit dem `typeof`-Operator wird der Datentyp ermittelt
- Funktional
- Objekt-orientiert, Vererbung über Prototypen (statt Klassen)
- Läuft in einer Sandbox im Browser:
  - jede Webseite oder Webanwendung wird innerhalb des Browsers isoliert ausgeführt
  - JavaScript hat nur Zugriff auf die Objekte des Browsers und kann nicht auf das Dateisystem zugreifen kann

Ausführung von JS

- JS ist eine interpretierte Sprache
- Brauchen Interpreter, der den JavaScript-Code ausführt
- JavaScript wird am meisten im Browser genutzt
- JS-Interpreter oder auch JS-Engine genannt, Beispiele sind:
  - SpiderMonkey - the first JavaScript engine, powered Netscape Navigator and today Firefox
  - V8 - open source, developed by Google in Denmark, part of Google Chrome
  - JavaScriptCore - open source, marketed as Nitro and developed by Apple for Safari
  - Chakra (JScript9) - Internet Explorer
  - Chakra (JavaScript) - Microsoft Edge
  - Rhino - managed by the Mozilla Foundation, open source, developed entirely in Java

Ausführung von JS

- Browser, d.h., die im Browser integrierte JavaScript-Engine führt JavaScript-Code aus
- Testen von JavaScript-Snippets:
  - Browsers Developer-Tools: z.B., Console in Chrome
  - JsFiddle: https://jsfiddle.net
  - Kommandozeile mit node.js
- Node.js
  - server-side JavaScript
  - nutzt V8
  - node auf PC installieren und in Kommandozeile ausführen:

Interaktiv oder

```
$> node
> function add(a,b) { return
a+b; }
undefined
> add(6,7);
13
```

Skript ausführen:

```
function add(a,b) { return
a+b;}
console.log(add(56, 15));
-----------------------------------------
---
$> node calc.js
71
```

Variablen

```
var eins = 1;        /* global */
zwei = 2;        /* global */
function x() {
     var drei = 3;   /* lokal in Funktion x */
     vier = 4;         /* global */
     fuenf = 5;              /* lokal, wegen “Hoisting” */
     var fuenf;
}
```

- Variablen haben Gültigkeitsbereich:
  - **var** deklariert Variable in aktueller Funktion, bzw. global
  - undeklarierte Variablen sind immer global
  - https://developer.mozilla.org/de/docs/Glossary/Hoisting: In JavaScript kann eine Variable definiert werden, nachdem sie benutzt wurde.

Datentypen

- **Primitive Datentypen:**
  - **number**

```
var x = 5; typeof x;      // liefert "number"
```

  - **string**

```
var x = “5”; typeof x; // liefert "string "
```

  - **boolean**

```
var x = true; typeof x; // liefert "boolean"
```

  - **undefined**

```
var x = undefined; typeof x; // liefert ”undefined "
```

- **Komplexe Datentypen**
  - **function**
  - **object**

```
typeof {name:”Mary”, age:30}      // liefert "object"
typeof [1,2,3,4]                        // liefert "object" (nicht "array”)
typeof null                               // liefert "object"
typeof function myFunc(){}        // liefert "function"
```

Datentypen: Automatische Umwandlung

- + ist Addition für Zahlen, aber auch Concatenation für Strings:

```
1 + 2 + 3;        // 6
“1” + “2” + “3”       // “123”
```

- Automatische Umwandlung:

```
"1" + 2 + 3;                    // "123”
16 + 4 + ”car”;               // “20car”
“car” + 16 + 4;               // ”car164”
var x = 5; var y = true; x + y;          // 6
var x = “5”; var y = true; x + y;     // ”5true”
 5 + null   // returns 5 because null is converted to 0
"5" + null  // returns "5null" because null is converted to "null"
"5" + 2     // returns "52"    because 2 is converted to "2"
"5" - 2     // returns 3  because "5" is converted to 5
"5" * "2"   // returns 10 because "5" and "2" are
          // converted to 5 and 2
```

Vergleichsoperatoren

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type

Vergleichsoperatoren

In JavaScript gibt es “==“, wie zum Beispiel `if (x == 22) { }`
und auch “===“ wie zum Beispiel `if (x === 22){ }`
Was ist der Unterschied?
Given `var x = 5;`

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true

Funktionen

- Syntax für Funktionendefintion:

```
function myFunction (p1, p2) {
    return p1 * p2;
}

function  hallo (name) {
    alert (“Hallo ” + name + “!”); // Popup-Window im Browser
}

hallo(“Du”);
```

Oder:

```
var plus = function (a, b) {

   return a + b;

}

plus(1, 2); // 3
```

Arrays

```
var array_name = [item1, item2, ...];
```

```
var cars = [“car1”, ”car2”, “car3”];
var person = ["John", "Doe", 46];
cars[0];
cars.length;
```

Array-Methoden: https://www.w3schools.com/js/js_array_methods.asp

In JavaScript, **arrays** always use **numbered indexes**.

JavaScript does not support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).
JavaScript **does not support associative arrays**.

WARNING !!
If you use named indexes, JavaScript will redefine the array to a standard object.
After that, some array methods and properties will produce **incorrect results**.

Beispiel: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_2

Objekte

- https://www.w3schools.com/js/js_object_definition.asp:

  In JavaScript, almost "everything" is an object.
  Booleans can be objects (if defined with the **new** keyword)
  Numbers can be objects (if defined with the **new** keyword)
  Strings can be objects (if defined with the **new** keyword)
  Dates are always objects
  Maths are always objects
  Regular expressions are always objects
  Arrays are always objects
  Objects are always objects

  Almost all JavaScript values, except primitives, are objects.
  A **primitive value** is a value that has no properties or methods.

  Was ist damit gemeint?

Vordefinierte Objekttypen und Objekte (Auswahl)

- Es gibt mehrere eingebaute Objekte und Objekttypen, diese können durch namensgleiche Konstruktoren erstellt werden:
  - Das namenlose globale Objekt, das alle Variablen und Objekte enthält.
  - Der Objekttyp `Object`, von dem alle Objekte abgeleitet sind: `var obj = new Object();`
  - Der Objekttyp `Function` für Funktionen : https://javascript.info/new-function
  - Der Objekttyp `Array` für Arrays.
  - Der Objekttyp `String` für Zeichenketten.
  - Der Objekttyp `Boolean` für boolesche Variablen.
  - Der Objekttyp `Number` für Zahlen (64-Bit-Gleitkommazahlen gemäß IEEE 754).
  - Der Objekttyp `Date` für Datumsformate (Daten bzw. Zeitpunkte).
  - Der Objekttyp `RegExp` für reguläre Ausdrücke.
  - Das **Objekt `Math`** stellt Konstanten und Methoden für mathematische Operationen bereit.

new - Operator

- Beispiel “new String()”:

```
var x = "John";             // typeof x is “string”
var y = new String("John"); // typeof y is “object”


// (x == y) is true because x and y have equal values
// (x === y) is false because x and y have different types (string and
object)

var y = new String("John");
var z = new String("John");

// (y == z) is false because y and z are different objects
```

Wozu ‘new String()’?

https://stackoverflow.com/questions/5750656/whats-the-point-of-new-stringx-in-javascript

Ich kann einem String-Objekt Attribute hinzufügen: y.prop = “foo”, alert(y.prop); // ”foo"

Vordefinierte Objekt-Methoden

- String, Number, Array, etc. haben eigene Methoden
- Zum Beispiel, einige String-Methoden:

```
“abcdefg”.charAt(2).toUpperCase(); // 'C’
“abcdefg”.indexOf(“ab”);            // 0
“abcdefg”.length;                   // 7
str.search(“someThing”);
str.subStr(…, …);
str.replace(…,…);
```

https://www.w3schools.com/js/js_string_methods.asp

Wie kann ich Objekte erstellen?

- **1. Methode: Objektliterale**

```
var brot = { // Object
    name: "Matze", // Property
    zutaten: [     // Array
            { "name": "Mehl",  "menge": 2 },
            { "name": "Wasser","menge": 1 }
        ],
    getRezept: function () { // Method
        return this.zutaten[0].name +
            " + " + this.zutaten[1].name +
            " = " + this.name;
    }
};

if (brot.name === brot["name"]) console.log(true); // logs to console
console.log(brot);
```

- Hausaufgabe: Ausprobieren in JsFiddle (https://jsfiddle.net/) oder Chromes Dev Tool

Wie kann ich Objekte erstellen?

- **2. Methode: Mit dem Keyword “new”**

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 30;
person.eyeColor = "blue";
```

Wie kann ich Objekte erstellen?

- **3. Methode: Mit Konstruktoren**

```
function MeinObjekt(x) {                    // Konstruktor
    this.something = x;
}

var object = new MeinObjekt(3);             // Instanz erzeugen

alert(object.something);                  // per Meldefenster ausgeben (3)
```

- Zugriff auf Objekteigenschaften und -methoden:
  - Punktnotation:

```
object.something;
brot.name;
brot.getRezept();
```

  - Klammernotation:

```
object[”something”];
brot[“name”];
brot[“getRezept”]();
```

Wie kann ich Objekte erstellen?

- Welche von diesen Methoden soll man denn jetzt nutzen?
- https://www.w3schools.com/js/js_object_definition.asp empfiehlt:

For simplicity, readability and execution speed, use the the object literal method.

Wie kann ich erstellte Objekte ändern?

- Zu allen Objekten können zur Laufzeit neue Eigenschaften und Methoden hinzugefügt werden:

```
function MeinObjekt(x) {                    // Konstruktor
    this.something = x;
}

var object = new MeinObjekt(3);             // Instanz erzeugen

alert(object.something);                  // per Meldefenster ausgeben (3)

object.newProp = “ah”;

object[“anotherOne”] = “oh”;

object.wow = function () {
    return this.newProp + “und” + this.anotherOne;
}

object.wow();
```

Wie kann ich erstellte Objekte ändern?

- Allen Objekten können zur Laufzeit Eigenschaften und Methoden entfernt werden:

```
function MeinObjekt(x) {                    // Konstruktor
    this.something = x;
}

var object = new MeinObjekt(3);             // Instanz erzeugen

object.newProp = “ah”;

object[“anotherOne”] = “oh”;

object.wow = function () {
    return this.newProp + “und” + this.anotherOne;
}

delete object.newProp
delete object[“anotherOne”]
delete object.something
```

Kontrollstrukturen

- Ähnlich denen in Java:

```
if (bedingung) {
      anweisungen;
} else {
      anweisungen;
}

while (bedingung) { anweisungen; }

do { anweisungen; } while (bedingung);

for (var i = 0; i < 100; i++) { anweisungen; }

for (var eigenschaftsname in objekt) { anweisungen; } /* Eigenschaften eines Objektes
durchlaufen */

for (var wert of objekt) { anweisungen; } /* Eigenschaftswerte eines Objektes durchlaufen */

switch (variable) {
      case wert1 : anweisungen; break;
      default : anweisungen;
}
```

Einbinden von JavaScript im HTML

```
<!– Als externe Ressource-->
<script src="script.js"></script>

<!– Früher so, aber jetzt veraltet: -->
<script type="text/javascript" language="JavaScript"
src="script.js">

<!-- HTML5 -->
<script src="script.js" async defer></script>
<!– defer: erst wenn DOM fertig -->
<!– async (HTML 5): parallel neben DOM-Erstellung -->

<!-- Inline -->
<script>
    confirm('Ok?');
</script>

<noscript> ….Normaler HTML-Code… </noscript>
```

- Browser liest das HTML-Dokument bis zum <script>-Tag und lädt dann das Skript, Parsen des HTMLs pausiert
- Für viele Skripts ist das vollständige Einlesen des HTMLs notwendig, da sie auf DOM zugreifen, (DOM = Document Object Model)
- Was tun?: Attribute “async” oder “defer” nutzen

https://bitsofco.de/async-vs-defer/

Einbinden von JavaScript im HTML

- Als Linkziel:

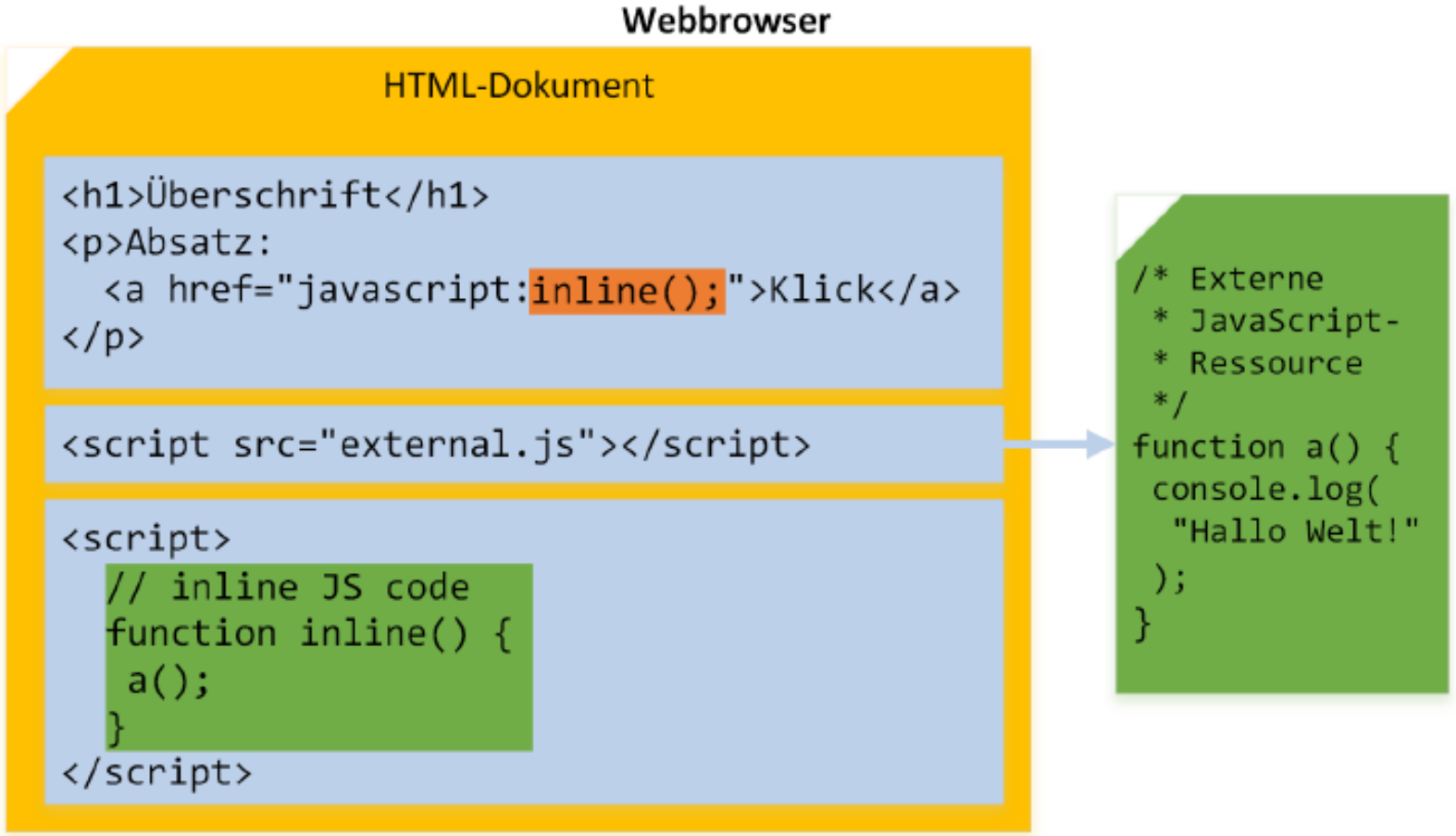
```
<a href="javascript:var b = 2 * 4; alert('Hallo Welt ' +
b)">Klick</a>

<a href="javascript:function a(b){return b*2;}; alert(a(5))">mich</
a>
```

Klick mich

- Verhalten wie normale Links
- Code wird bei Klick ausgeführt

Ausführung von JS im Browser

- Grün: Ausführung beim Laden des Dokuments
- Orange: Ausführung beim Klick



DOM – Document Object Model

- DOM API ist die Schnittstelle zwischen HTML und dynamischem JavaScript
- DOM erlaubt JavaScript den Zugriff auf und die Manipulation von HTML-Elementen
- HTML-Dokument wird als Baumstruktur dargestellt
- Jeder Knoten ist ein Objekt, welches einen Teil des Dokuments repräsentiert
- Die Grundlage für dynamische Webseiten
- Mit Hilfe von DOM und JavaScript kann man:
  - auf alle HTML-Element zugreifen, und
  - im Browser Elemente anzeigen lassen oder verstecken (programmatisch)
  - Elemente verändern, z.B., Text hinzufügen, etc.
  - auch neue Elemente hinzufügen

DOM: window

- Globales Objekt: `window`
- Das `window` Objekt repräsentiert ein Fenster (Window), das ein DOM Dokument enthält.
- In `window`: `window.document`
- Das `document` Attribut zeigt auf das DOM-Document, das im Fenster geladen ist.
- Alle Eigenschaften und Methoden von `window` hier: https://developer.mozilla.org/en-US/docs/Web/API/Window

(Open link, Beispiel: navigator – browser detection)

DOM: document

- Aktuelles Dokument über `window.document`
- Oder direkt über `document`
- Alle Eigenschaften und Methoden von `document` hier: https://developer.mozilla.org/en-US/docs/Web/API/document

```
document
    .head
    .body
    .contentEditable = true
    .links[0].href
    .images
    .cookie
    .createElement()
    .get*
    .getElementById(id) // liefert das Element mit Id “id”
    .[Formularname][Inputname].value
```

Dokument und DOM

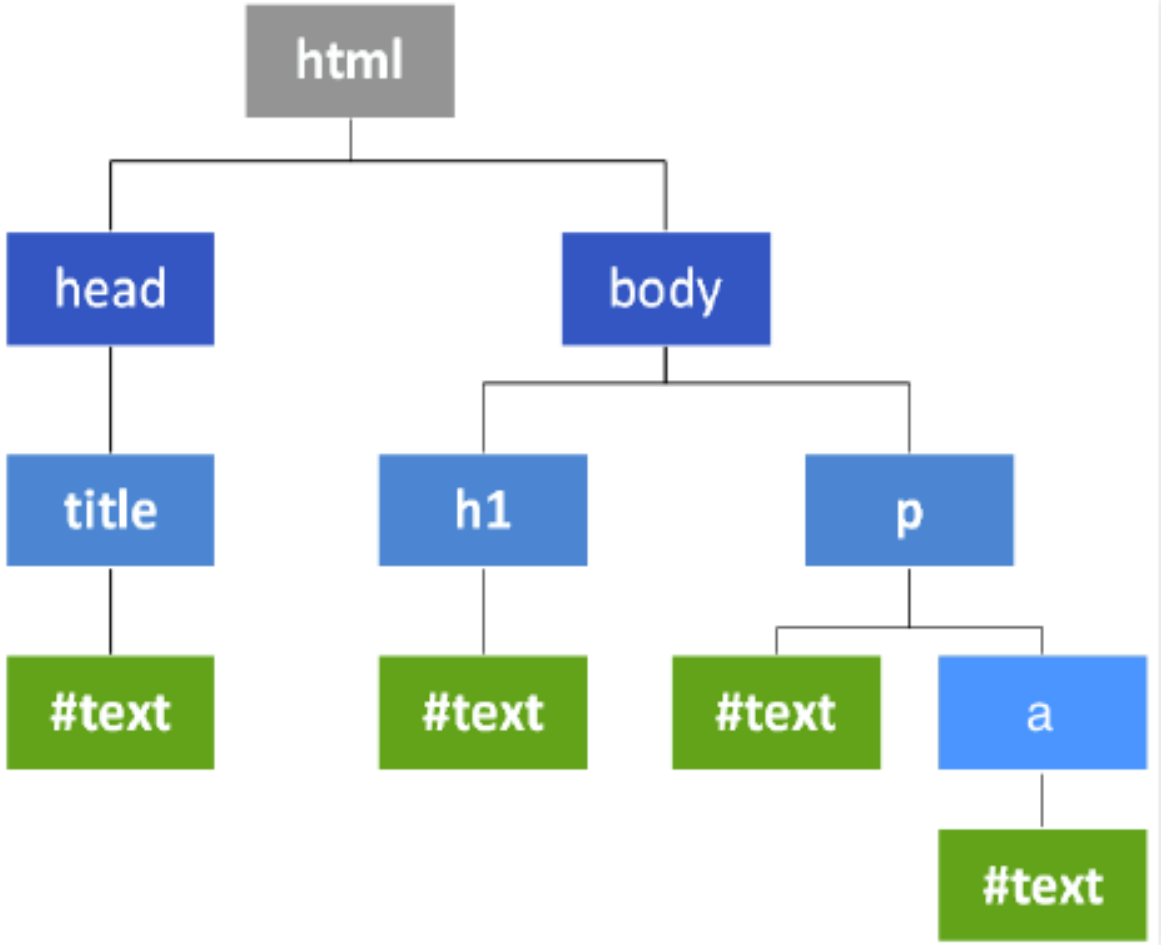
Webbrowser

HTML-Dokument

```
<h1>Überschrift</h1>
<p>Absatz:
  <a href="javascript:inline();">Klick</a>
</p>
```

```
<script src="external.js"></script>
```

```
<script>
  // inline JS code
  function inline() {
    a();
  }
</script>
```



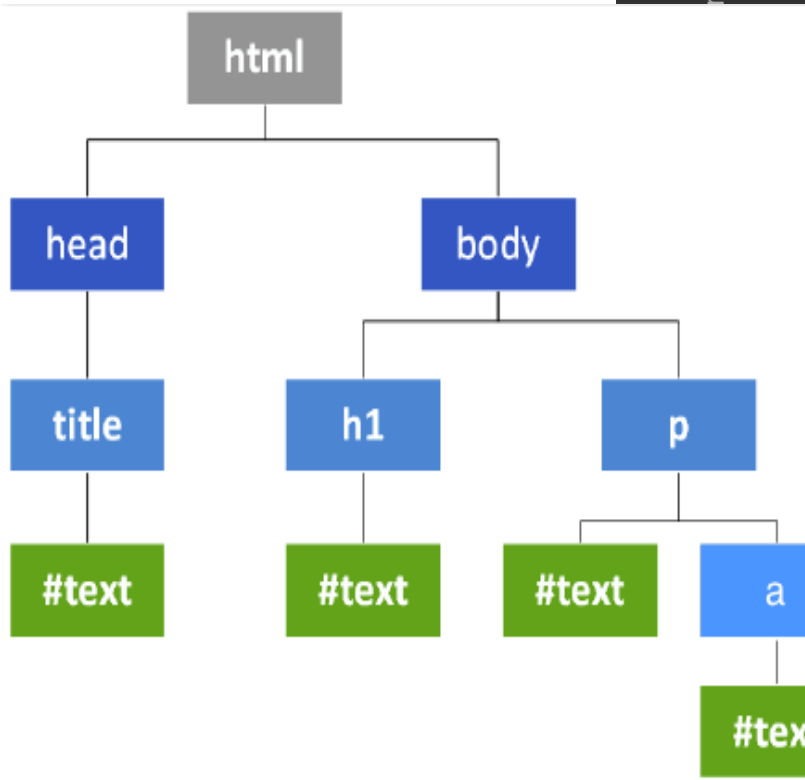
DOM: Traversieren & Manipulieren

- Text von h1 auf “Hallo!” setzen:

```
document.getElementsByTagName(“h1”)[0]
    .textContent="Hallo!";
```

- Inhalt des Knoten unten rechts auf “Hi!” setzen:

```
document.getElementsByTagName(“p”)[0]
    .childNodes[1]
    .textContent="Hi!";
```



- Das p-Element nicht anzeigen:

```
document.getElementsByTagName(“p”)[0].style.display = "none";
```

- Ein HTML-Element mit Id manipulieren, d.h., als Blockelement anzeigen lassen

```
document.getElementById("demo").style.display = “block”
```

- Ausprobieren: https://www.w3schools.com/js/js_htmldom_elements.asp
- Alle Werte von `style.display`: https://www.w3schools.com/jsref/prop_style_display.asp

DOM: Events

- Nutzer-Interaktion mit DOM
- DOM kennt unzählige Events (Ereignisse):
  - Focus Events
  - Mouse Events
  - Clipboard Events
  - Drag & Drop Events
  - ...
  - vollständige Liste hier:
    https://developer.mozilla.org/en-US/docs/Web/Events

DOM: Events

onabort // bei Abbruch
onchange // bei erfolgter Änderung
onclick // beim Anklicken
ondblclick // bei doppeltem Anklicken
onerror // im Fehlerfall
onfocus // beim Aktivieren
onkeydown // bei gedrückter Taste
onkeypress // bei gedrückt gehaltener Taste
onkeyup // bei losgelassener Taste
onload // beim Laden einer Datei
onmousedown // bei gedrückter Maustaste
onmousemove // bei weiterbewegter Maus
onmouseout // beim Verlassen des Elements mit der Maus
onmouseover // beim Überfahren des Elements mit der Maus
onreset // beim Zurücksetzen des Formulars
onresize // bei Größenänderung des Fensters
onselect // beim Selektieren von Text
onsubmit // beim Absenden des Formulars
onunload // beim Verlassen der Datei

DOM: Events

- Ausprobieren mouseover:

https://www.w3schools.com/jsref/event_onmouseover.asp

- Beispiel “onclick” in HTML:

```
<!DOCTYPE html>
<html>
<body>
<p>This example demonstrates how to assign an "onclick" event to a p
element.</p>
<p id="demo" onclick="myFunction()">Click me.</p>
<script>
function myFunction() {
      document.getElementById("demo").innerHTML = "YOU CLICKED ME!";
      alert(“Clicked me!”);
}
</script>
</body>
</html>
```

https://www.w3schools.com/jsref/event_onclick.asp

DOM: Events

- Beispiel “onclick” in JavaScript:

```
// ein Listener
element.onclick = function () { this.innerText = 'Nacht'; }

// mehrere Listener
element.addEventListener('click', function (event) {
      this.innerText = 'Nacht';
});
```

JSON

- JSON = JavaScript Object Notation
- JSON ist ein Datenformat zum Speichern und Austauschen von Daten
- JSON ist Text im Format der JavaScript Object Notation:
  - Daten sind Schlüssel/Werte-Paare
  - Daten werden durch Kommata voneinander getrennt
  - Geschweifte Klammern umschließen Objekte
  - Eckige Klammern umschließen Arrays
- Beispiel:

```
{
    “boolean” : true,
    “number” : 42,
    “string” : “some text”,
    “object” : { “key1” : ”value1”, “key2” : “value2” },
    “array” : [ “element1”, “element2”, “element3”]
}
```

Das JSON-Objekt in JS

```
// typeof myObj liefert object
var myObj = { "name":"John", "age":31, "city":"New York" };

var myJSON = JSON.stringify(myObj)     // typeof myJSON liefert string

// in localStorage mit setItem() speichern (siehe nächste Folie)
localStorage.setItem("textJSON", myJSON);

// mit getItem() holen
text = localStorage.getItem("textJSON");

obj = JSON.parse(text);              // typeof obj liefert object

document.getElementById("demo").innerHTML = obj.name;
```

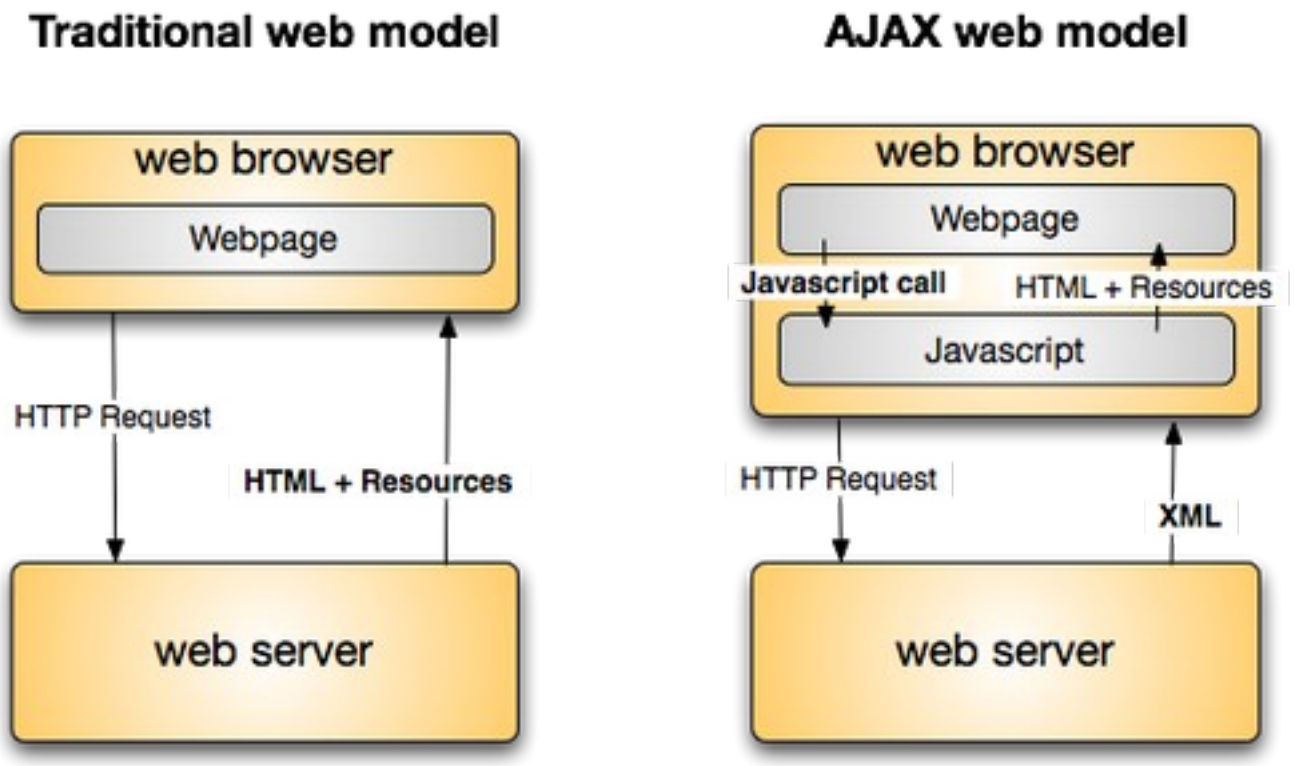
DOM Storage

- Seit HTML 5 zwei Objekte für das client-seitige Speichern von Daten:
  - window.localStorage
  - window.sessionStorage
- Client-seitiger Key-Value-Speicher: String String
- 5 MB
- localStorage - stores data with no expiration date
- sessionStorage - stores data for one session (data is lost when the browser tab is closed)
- https://developer.mozilla.org/de/docs/Web/API/Window/localStorage

AJAX

- Asynchronous JavaScript and XML
- Konzept der asynchronen Datenübertragung zwischen einem Browser und dem Server
- Frontend, meist ein JavaScript-Skript, führt im Hintergrund HTTP-Anfragen durch, während eine HTML-Seite angezeigt wird, ohne die Seite komplett neu zu laden
- Mit AJAX ist es möglich, Daten von einem Server anzufragen und den Inhalt einer Seite zu ändern, ohne das die ganze Seite neugeladen werden muss
- Dabei spielt das JavaScript-Objekt “XMLHttpRequest” eine zentrale Rolle
- Allerdings ist das Format der Daten, die zwischen Server und Client ausgetauscht werden nicht XML, sondern meist JSON

AJAX

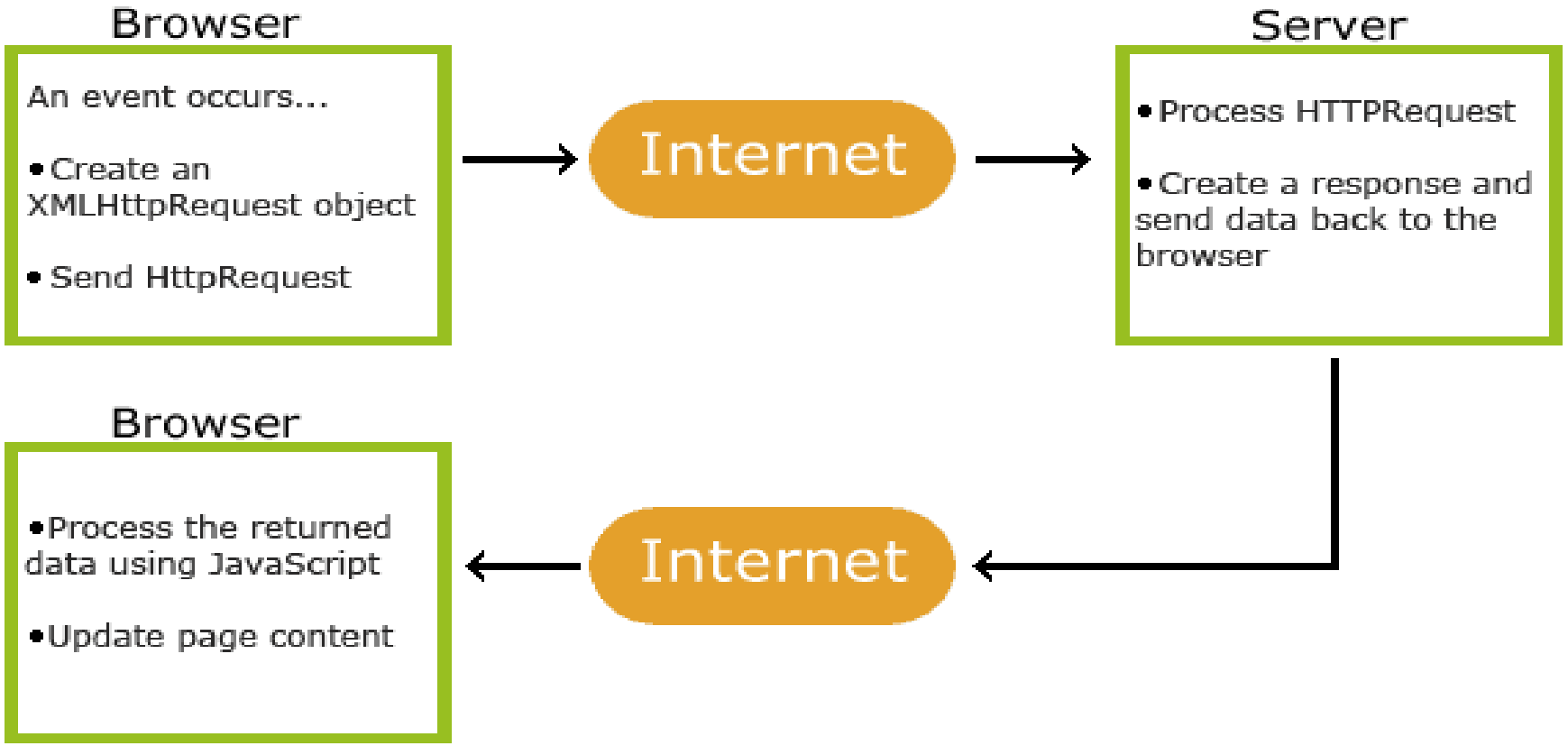


Source: https://derivadow.com/2007/01/05/ajax-what-is-it-its-not-dhtml

- Jesse James Garrett: *"Ajax: A New Approach to Web Applications"*,18. Februar 2005
- XMLHttpRequest oder besser seinen Vorgänger gibt es seit 1999, von MicroSoft entwickelt, w3c-specification für XMLHttpRequest seit 2006

XMLHttpRequest

- https://www.w3schools.com/xml/ajax_intro.asp:



XMLHttpRequest

1. Ein Event auf einer Webseite findet statt (ein Button wird gedrückt)
2. XMLHttpRequest-Objekt wird von/in einem JavaScript-Skript kreiert
3. Das XMLHttpRequest-Objekt schickt eine Anfrage an einen Webserver.
4. Der Server verarbeitet die Anfrage
5. Der Server schickt seine Antwort, meist im JSON-Format, an die Webseite zurück
6. JavaScript erhält und liest die Antwort/Daten.
7. JavaScript erneuert Teile der Webseite, stellt diese Daten dar oder führt weitere Operationen durch (ändert/manipuliert diese Daten)

XMLHttpRequest

- https://www.w3schools.com/xml/ajax_xmlhttprequest_send.asp

Method	Description
open(*method, url, async*)	Specifies the type of request *method*: the type of request: GET or POST *url*: the server (file) location *async*: true (asynchronous) or false (synchronous)
send()	Sends the request to the server (used for GET)
send(*string*)	Sends the request to the server (used for POST)

siehe nächste Folie

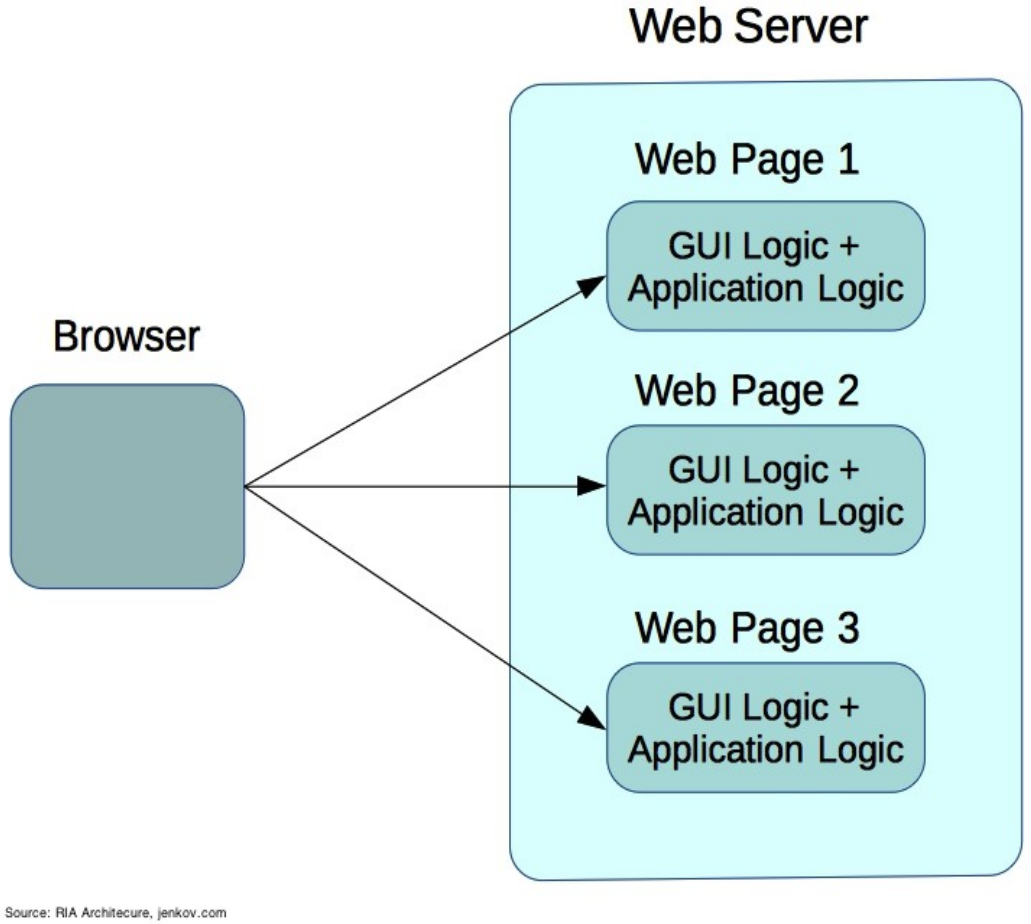
```
var xhr = new XMLHttpRequest();
    var url = "https://maps.googleapis.com/maps/api/geocode/json?"
    url = url + "address=Wilhelminenhofstr 75, Berlin";
    url = url +"&key=....";

    xhr.open("GET", url, true);

    xhr.onerror = function() {// diese Funktion wird ausgefuehrt, wenn ein Fehler auftritt
        alert("Connecting to server with " + url + " failed!\n");
    };
    xhr.onload = function(e) {// diese Funktion wird ausgefuehrt, wenn die Anfrage erfolgreich
war
        var data = this.response;
        var obj = JSON.parse(data);
         console.log(obj);
        if (this.status == 200) {
            if (obj.status != "ZERO_RESULTS")  {
              var lat = obj.results[0].geometry.location.lat; var lng =
obj.results[0].geometry.location.lng;
                console.log (lat +", " + lng);
             } else { alert ("Die Adresse konnte nicht aufgelöst werden!");}
        } else {     //Handhabung von nicht-200er
           alert ("HTTP-status code was: " + obj.status);
        }
    };
```

XMLHttpRequest: Anmerkungen

- In https://www.w3schools.com/xml/ajax_xmlhttprequest_response.asp
  Alternativen zur Handhabung des Response (Antwort)
- XMLHttpRequest ermöglicht das Setzen von Request-Header und Checken der Response-Header
- Mit XMLHttpRequest kann man auch Daten im Request-Body an den Server schicken (POST)
- xhttp.open("GET", url, false) – ist eine synchrone Anfrage, d.h., das Skript pausiert und wartet auf die Antwort des Servers – das wird **nicht** empfohlen, ist aber manchmal nicht vermeidbar.

Thin clients

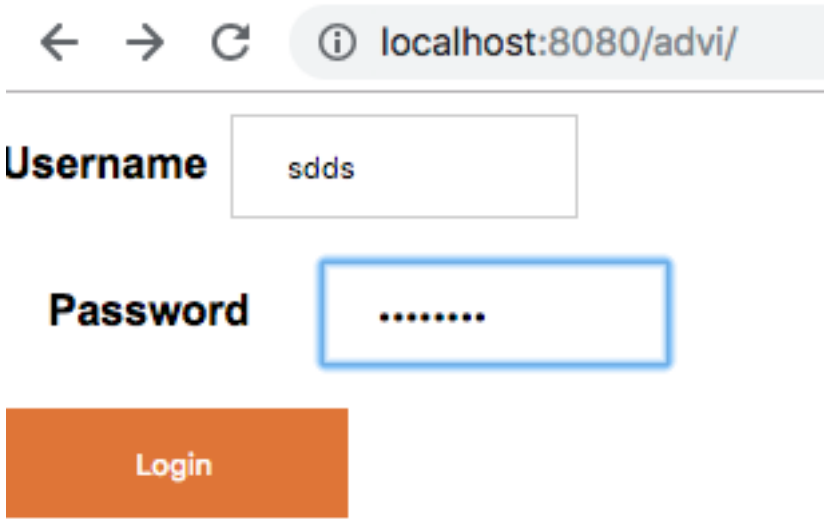
- Traditionelle bzw. ältere Webapplikationen
- Fast alles wird server-seitig verarbeitet
- Client-seitig: Anzeigen der Seite im Browser, evt. Validierungen des Userinputs
- User Eingabe + Button → HTTP-Anfrage an Server → Server “macht alles”, inklusive neue Seite zusammenbauen und schickt komplette, neue HTML-Seite zurück (Repeat)
- Server ist für das Zusammen-setzen der Benutzeroberfläche und Applikationslogik verantworlich



- Beispiele: CGI-Scripts, Java Server Pages (JSP), Java Server Faces (JSF)

Thin client - Beispiel

- Unsere “Login-Demo” Webapp:
  - index.html
  - Login.js (controller)
  - User.js (model)
  - View.js (view)



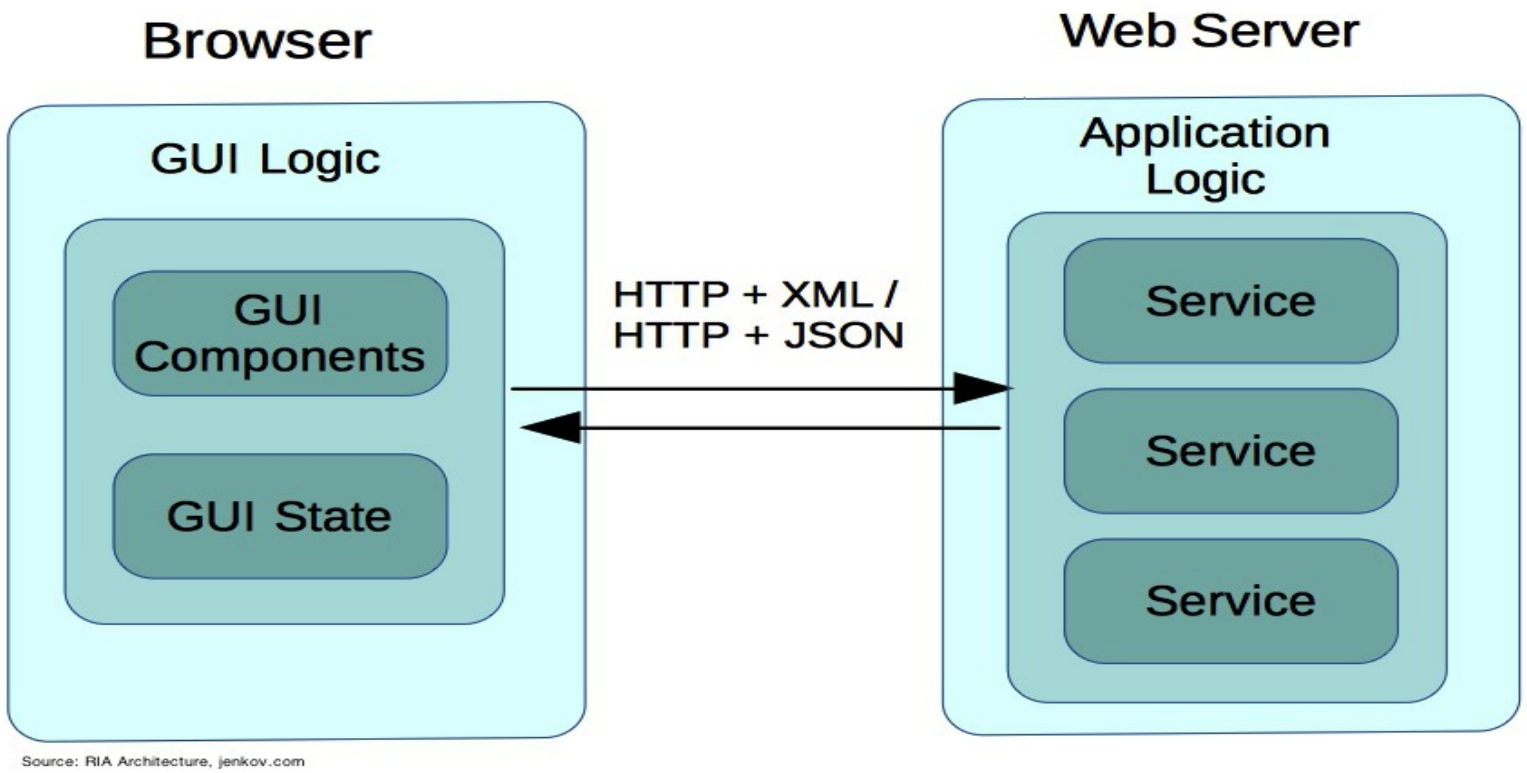
GET: http://localhost:8080/advi/login?username=sdds&password=saadsdas

Login.js:
- Erhält Anfrage
- Business-Logik:
  - User-Objekt belegt mit Daten aus DB (not really, aber bald)
  - User-Objekt in der HttpSession speichern
- Request an View.js weiterleiten

View.js:
- Ist HTML mit JSX-Code
- Wird von Node komplett in HTML umgewandelt
- Dieser JavaScript-Code generiert die HTML-Seite, welche an den Browser zurückgeschickt wird

ANMERKUNG: **Passwörter niemals per http und GET schicken, sondern https & POST**

Modern (rich) web clients

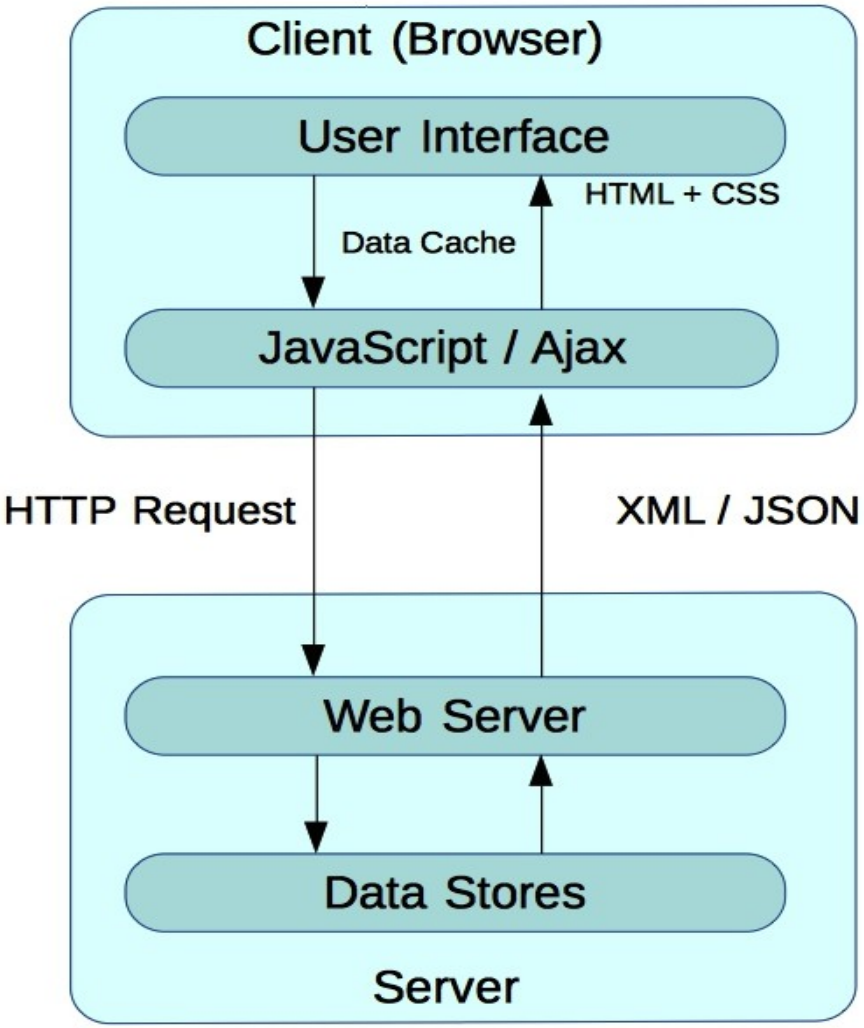


- Dar- und Erstellung von UI-Komponenten
- Datenanfragen an Server
- Anzeigen von Daten und kleinere Datenmanipulationen
- Lokaler Datenspeicher
- Application-Logic in direktem Zusammenhang mit UI-GUI State

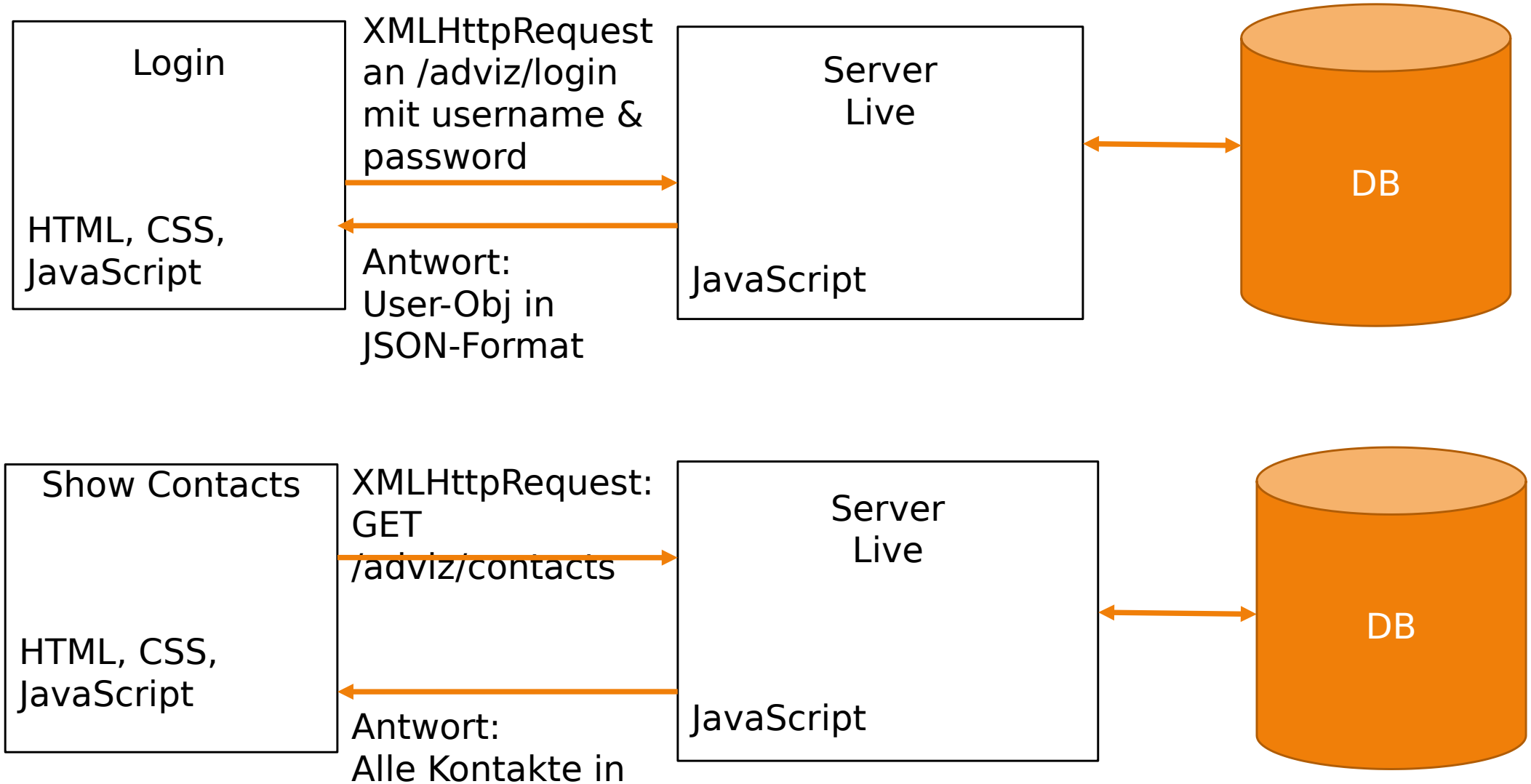
- Service
- Bedient Datenanfragen des Clients
- Application/Business Logic
- Datenmanagement
- Komplexere Berechnungen/Daten-manipulationen

Modern (rich) web clients

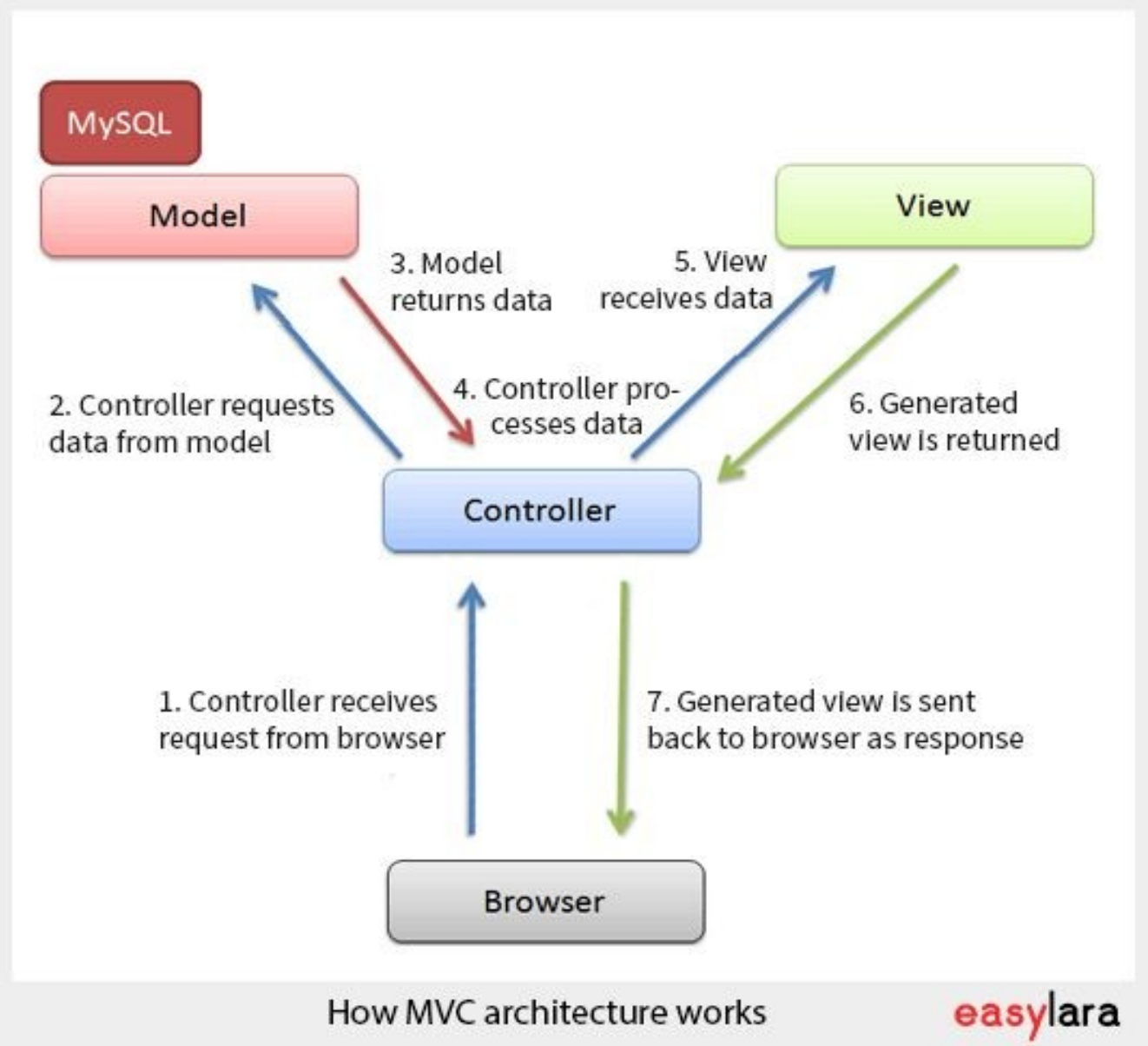
- Datenaustausch zwischen GUI und Server mittels AJAX:
  - Asynchrone Datenübertragung zwischen Browser und Server
  - HTTP-Anfragen im Background durchführen, während eine HTML-Seite angezeigt wird
  - Seite wird verändert, ohne sie komplett neu zu laden
- Server-seitig: meist ein Webservice, nutzt Frameworks wie Node, Spring Web MVC, Jersey, RESTEasy (alle Java), PHP-Frameworks (Symfony), etc.
- Client-seitig: meist ein JS-Framework wie Angular, React, etc. oder JQuery, oder reiner JavaScript-Code (adviz)



Modern client – Beispiel: adViz



MVC für Webapplikationen: server-seitig



- Thin Clients:
  - View ist HTML-Format
- Rich Clients:
  - View ist JSON od. XML - Format

MVC für Webapplikationen

- Rich Clients: Clients, die mittels AJAX Daten vom Server anfordern
- Sie erhalten diese Daten meist im JSON-Format und müssen sie weiterverarbeiten
- Im Frontend wird meist komplexer JavaScript-Code ausgeführt:
  - Interaction mit dem User
  - Validierung der Usereingaben
  - Beschaffung der Daten (Server oder evtl. ein anderer Webservice)
  - Evtl. Business Logic
  - Bauen der GUI (HTML-Seite)
- Deshalb im Frontend auch komplexe JS-Frameworks

JavaScript-Test

- https://www.w3schools.com/quiztest/quiztest.asp?qtest=JavaScript\

The End

- Questions?
- References:
  - https://www.quora.com/Why-is-JavaScript-so-popular
  - JavaScript Tutorial: https://www.w3schools.com/js/default.asp
  - AJAX Tutorial: https://www.w3schools.com/xml/ajax_intro.asp
  - Dane Cameron: “A Software Engineer Learns HTML5, JavaScript & jQuery”, Cisdal Publishings, 2015
  - Frank Zieris, Webentwicklung, WiSe2017/18, HTW: https://www.zieris.net/teaching/htw-berlin/webdev-slides/
  - Max Beier/Thomas Ziemer “Webentwicklung WiSe 2017/18 HTW Berlin” - Folien: https://beier.f4.htw-berlin.de
  - JsFiddle: https://jsfiddle.net/
  - https://docs.microsoft.com/de-de/dotnet/standard/modern-web-apps-azure-architecture/choose-between-traditional-web-and-single-page-apps