Webentwicklung – Teil 5

DHBW SS2018 – Elias Henrich

Überblick Heute

- Wiederholung JS-Grundlagen
- Funktionen & Events
- Variablengültigkeit
- AJAX
- Abschluss Komponente „Flugliste“
- Verteilung Vortragsthemen

JS einbinden

```
<!DOCTYPE html>
<html lang="de">
    <head>
        <meta charset="UTF-8">

        <script>
            /* Platz für JavaScript */
        </script>

    </head>
    <body>
        <p>
            Ich bin der Body
        </p>

        <script>
            /* Platz für JavaScript */
        </script>

    </body>
</html>
```

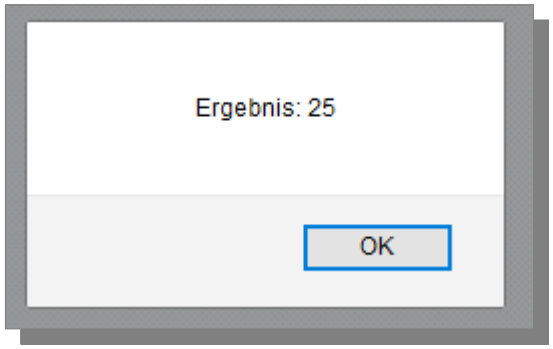
Interne Anweisungen:

- in `<head>` oder `<body>`
- innerhalb von `<script>`
- `type`-Attribut nicht notwendig!
- beliebig viele `<script>`-Abschnitte erlaubt

Ausgabe

- Message-Box

```
var a = 10;
var b = 15;
var c = a + b;

alert("Ergebnis: " + c);
```



Ausgabe

- Message-Box
- HTML-Body anhängen

```
var a = 10;
var b = 15;
var c = a + b;

document.write("Ergebnis: " + c);
```

Hier ist der Body...

Ergebnis: 25

Ausgabe

- Message-Box
- HTML-Body anhängen
- Inhalt eines HTML-Elements ändern (Id)

```
<p>
    Ergebnis: <span id="result">-</span>
</p>

<script>
    var a = 10; var b = 15;
    var c = a + b;

    document.getElementById('result').innerHTML = c;
</script>
```

Ausgabe

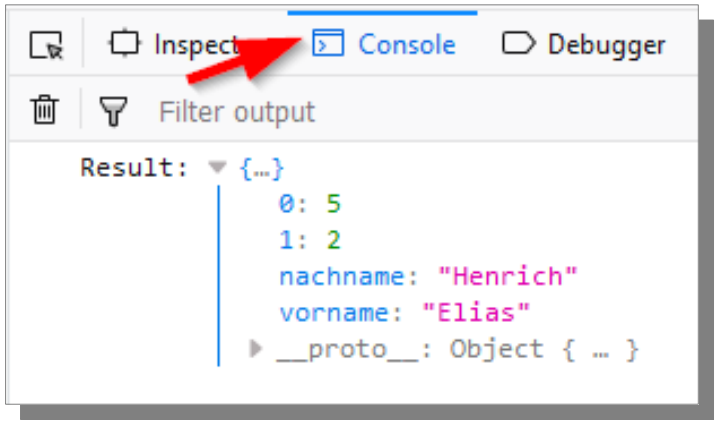
- Message-Box
- HTML-Body anhängen
- Inhalt eines HTML-Elements ändern (Class)

```
<p class="absatz">Erster Absatz</p>
<p class="absatz">Zweiter Absatz</p>

<script>
    var a = 10; var b = 15;
    var c = a + b;

    var elemente = document.getElementsByClassName('absatz');
    for (i = 0; i < elemente.length; i++) {
        elemente[i].innerHTML = "Ergebnis: " + c;
    }
</script>
```

Ausgabe

- Message-Box
- HTML-Body anhängen
- Inhalt eines HTML-Elements ändern
- Browser-Konsole (Developer Tools öffnen)

```
var a = {
    0: 5,
    1: 2,
    'vorname': 'Elias',
    'nachname': 'Henrich'
};

console.log("Result:", a);
```



Funktionen

```
<script>
    function vergleiche(a, b) {
        return (a === b);
    }

    console.log("10, 10", vergleiche(10,10));
    console.log('10, "10"', vergleiche(10,"10"));
</script>
```

Objekt-/Klassenfunktionen

```
<script>
    function VergleichsObjekt(a, b) {
        this.a = a;
        this.b = b;

        this.vergleichWert = function() {
            return (this.a == this.b);
        };

        this.vergleichTyp = function() {
            return (this.a === this.b);
        };
    }

    var meinVergleich = new VergleichsObjekt(10, "10");

    console.log("Wertvergleich", meinVergleich.vergleichWert());
    console.log("Typvergleich", meinVergleich.vergleichTyp());
</script>
```

Events

= zusätzliches HTML-Tag-Attribut
mit JS-Code (meist Funktions-Call)

```
<input type="button"
       onClick="alert(vergleiche(10, 100));"
       value="Vergleichen!"
>

<script>
    function vergleiche(a, b) {
        return (a === b);
    }
</script>
```

Events

Häufig eingesetzte Events

Event	Bedeutung
`onclick`	Mausklick (z.B. Button)
`onmouseover`	Maus über dem Element
`onmouseout`	Maus verlässt das Element
`onchange`	Wertänderung (z.B. input)
`onload`	Seite wurde vollständig geladen

es gibt zahlreiche weitere Events, siehe auch:
https://www.w3schools.com/jsref/dom_obj_event.asp

Übung: Overlay einblenden

1. Fügen Sie dem Formular (`form`-Tag) folgendes Attribut hinzu:

```
onsubmit="return showOverlay()"
```

2. Erstellen Sie eine JS-Funktion `showOverlay()`, die:
   - das Overlay selektiert und mit Hilfe einer Variablen angesprochen werden kann
   - das Overlay einblendet → `element.style.display`
   - den Wert `false` zurückliefert

Nutzen Sie während der Entwicklung `console.log("Test")` um zu prüfen, ob die Funktion tatsächlich ausgeführt wird.

Übung: Overlay einblenden

3. Erstellen Sie eine Funktion `hideOverlay()`, die den Modal-Dialog wieder ausblendet
4. Rufen Sie die Funktion `hideOverlay()` nach 3s auf, sobald das Overlay eingeblendet wurde

```
window.setTimeout()
```

Führt eine Funktion nach einer angegebenen Zeit [ms] aus

Variablengültigkeit

- **Lokale Variablen**
  sind nur innerhalb ihrer Funktion verfügbar

```
<script>
    var a = 6;

    function rechne() {
        var a = 3;
        var b = 5;

        return (a + b);
    }

    console.log(rechne());     // Ausgabe: 8
    console.log(a);            // Ausgabe: 6
</script>
```

Variablengültigkeit

- Lokale Variablen
- **Globale Variablen**

```
<script>
    var a = 6;

    function rechne() {
        var b = 5;

        return (a + b);
    }

    console.log(rechne());     // Ausgabe: 11
    console.log(a);            // Ausgabe: 6
</script>
```

Variablengültigkeit

- Lokale Variablen
- Globale Variablen

→ **Automatisch globale Variablen**

```
<script>
    function rechne() {
        a = 7;
        var b = 5;

        return (a + b);
    }

    console.log(rechne());      // Ausgabe: 12
    console.log(a);             // Ausgabe: 7
</script>
```

HTML-Attribute

Attribut lesen:

```
var klasse = document.getElementById('feld').getAttribute('class');
```

Attribut schreiben:

```
document.getElementById('feld').setAttribute('class', 'active');
```

Zugriff auf CSS-Klassen

Auf Klasse prüfen:

```
<div id="feld" class="box active">[...]</div>

<script>

    var active =
        document.getElementById('feld').classList.contains('active');
    // active = true

</script>
```

Klasse hinzufügen:

```
<div id="feld" class="box active">[...]</div>

<script>
    document.getElementById('feld').classList.add('selected');
</script>
```

Zugriff auf CSS-Klassen

Klasse entfernen:

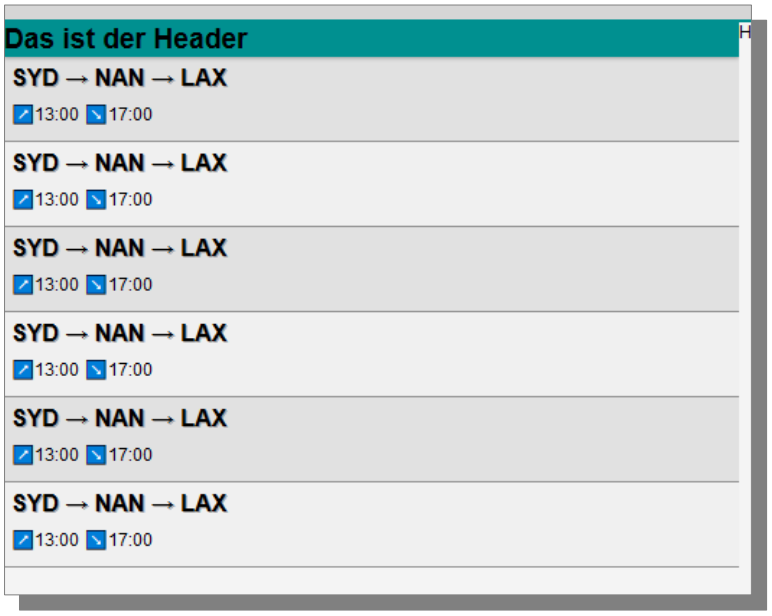
```
<div id="feld" class="box active">[...]</div>

<script>
    document.getElementById('feld').classList.remove('selected');
</script>
```

Klasse toggle / wechseln:

```
<div id="feld" class="box active">[...]</div>

<script>
document.getElementById('feld').classList.toggle('selected');
</script>
```

Übung: Liste dynamisch füllen

1. Erstellen Sie eine CSS-Klasse, mit `display: none;`
2. Entfernen Sie, außer das erste, alle Listenelemente
3. Weisen Sie diesem die obige CSS-Klasse und ID=`listDummy` zu



Erstellen & testen Sie eine Funktion, die folgendes ausführt:

1. `<ul>`-Liste einer Variablen zuweist
2. HTML-Code des Listenelements `<li>` (`element.outerHtml`) einer Variablen zuweist
3. der `<ul>`-Liste den HTML-Code des Elements 5-mal hinzufügt

Nutzen Sie min. eine Schleife!

Ziel: Vorbereitung für das Einlesen der Flugdaten per AJAX

AJAX

Asynchronous **J**avaScript **a**nd **X**ML

- Daten holen, nachdem die Seite geladen ist
- Daten senden, ohne Seite neu laden zu müssen
- Daten darstellen, ohne die Seite neu laden zu müssen
- Nicht auf XML beschränkt
  u.a. auch Text-, HTML- oder JSON-Daten

AJAX

Grundlage für AJAX:

XMLHttpRequest

- ursprünglich von Microsoft, mittlerweile Standard in allen Webbrowsern
- API für JavaScript zur Datenübertragung mit HTTP
- strenge Sicherheitsauflagen, daher aktuell neue Standardisierung im Gange

API = Application Programming Interface - Programmierschnittstelle

XMLHttpRequest

1. `XMLHttpRequest`-Objekt erzeugen
2. Verbindung herstellen

   `open(methode, url, async);`
3. Anfrage abschicken

```
var request = new XMLHttpRequest();

request.open('GET', 'http://example.com', true);
request.send();
```

XMLHttpRequest

Event beim Wechsel des Zustands:
onreadystatechange()

```
request.onreadystatechange = function() {
    if (this.readyState == 0) {
        // open() wurde noch nicht aufgerufen
    } else if (this.readyState == 1) {
        // Verbindung geöffnet, open() aufgerufen
    } else if (this.readyState == 2) {
        // send() aufgerufen, Response-Header empfangen
    } else if (this.readyState == 3) {
        // Daten werden heruntergeladen
    } else if (this.readyState == 4) {
        // Request abgeschlossen
    }
}
```

→ i.d.R. ist readyState = 4 relevant

XMLHttpRequest

ReadyState gibt nur Infos über Zustand der Anfrage aus, nichts über den Erfolg!

daher: HTTP-Statuscode prüfen

```
request.onreadystatechange = function() {
    if (this.readyState == 4) {
        if (this.status == 200) {
            // Alles ok
        } else {
            // Anfrage abgeschlossen, jedoch fehlerhaft
        }
    }
}
```

XMLHttpRequest

Empfangene Daten liegen in `this.responseText`:

```
function getData() {
    var request = new XMLHttpRequest();

    request.onreadystatechange = function() {
        if (this.readyState == 4) {
            if (this.status == 200) {
                var result = this.responseText;

                document.getElementById('divElement').innerHTML = result;
            }
        }
    }

    request.open('GET', 'http://eliashenrich.de/dhbw/ajax.php', true);
    request.send();
}
```

Same-Origin-Policy

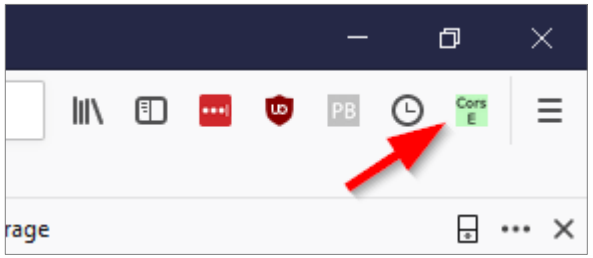
ist ein Sicherheitskonzept, das [...] JavaScript [...] untersagt, auf Objekte zuzugreifen, [...] deren Speicherort nicht der Origin entspricht. Sie stellt ein wesentliches Sicherheitselement [...] zum Schutz vor Angriffen dar.

https://de.wikipedia.org/wiki/Same-Origin-Policy

Same-Origin-Policy

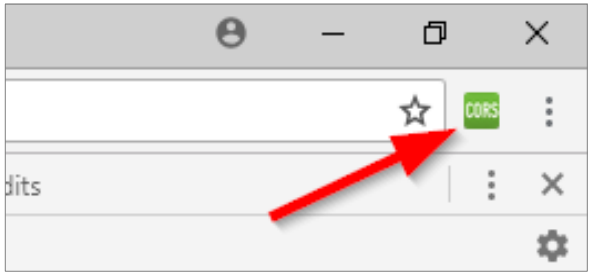
Durchsetzen der SOP unterdrücken:

Addon für Firefox:



https://addons.mozilla.org/de/firefox/addon/cors-everywhere/

Erweiterung für Chrome:



https://chrome.google.com/webstore/detail/cors-toggle/jioikioepegflmdnbocfhgmpmopmjkim

Schnittstellendoku

Flightsearch Web-API

Get all airports

Endpoint: http://flights.eliashenrich.de/api.php
Method: GET
Params: action /airports/all

Example:

```
http://flights.eliashenrich.de/api.php?
        action=/airports/all
```

Schnittstellendoku

Flightsearch Web-API

Get routes between airports

Endpoint: http://flights.eliashenrich.de/api.php
Method: GET
Params: action /route/find
from <airport-id> (e.g. 25)
to <airport-id> (e.g. 23)

Example:

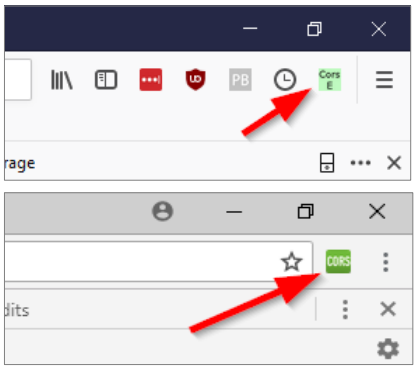
```
http://flights.eliashenrich.de/api.php?
  action=/route/find&from=25&to=23
```

Übung: XMLHttpRequest 1/2

Erstellen Sie eine Funktion, mit dieser Definition:
`requestAPI(action, data, callback)`

Funktion ruft den Endpunkt der Flug-Web-API mit der angegebenen `action` und allen Werten des `data`-Arrays auf. Anschließend wird die in `callback` übergebene Methode aufgerufen und an diese das Ergebnis der API-Abfrage übergeben.

```
var params = {
    from: 25,
    to: 23
};

requestAPI('/airports/all', params, function(result) {
    alert("Anfrage beendet");
    console.log("Ergebnis", result);
});
```



Übung: XMLHttpRequest 2/2

Fügen Sie alle Komponenten zusammen:

- Event `onsubmit` löst Abfrage der Route `25` nach `23` aus
- Während die Abfrage läuft, wird automatisch das Overlay ein- und ausgeblendet
- Die Ergebnisliste wird geleert
- Die empfangenen JSON-Daten werden in der Liste dargestellt

Abschlussvorträge

- siehe Handout -