Javascript in Webseiten

Übersicht

- Javascript in Webseite einbinden
- Document Object Model (DOM)
- DOM Elemente selektieren
- Elemente bearbeiten
- Events
- Fehlerbehandlung

Javascript in Webseite einbinden

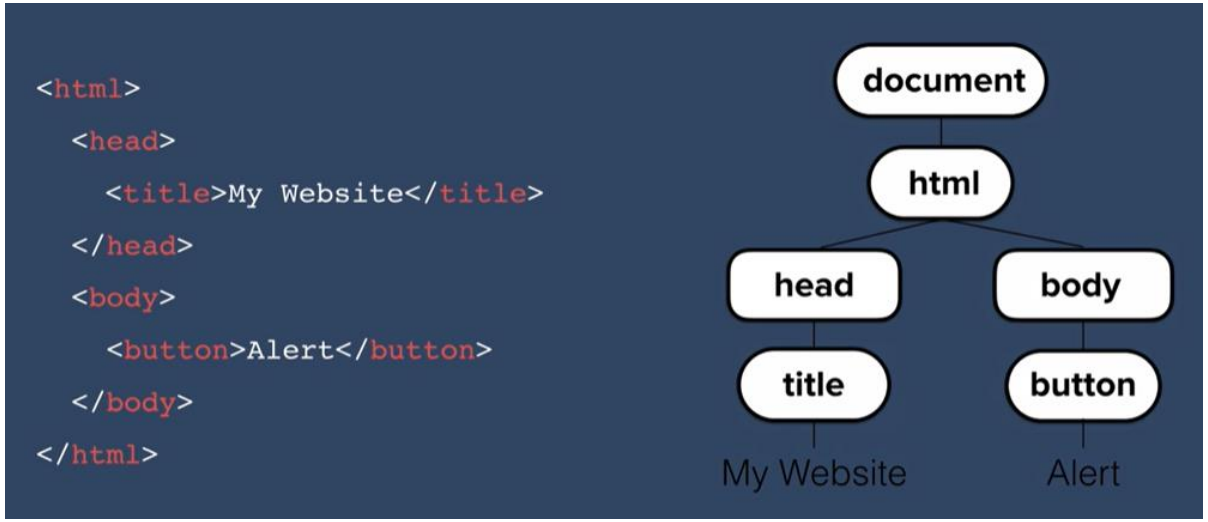
- Über den <script> Tag wird JS in HTML Eingebunden
- Es gibt 3 unterschiedliche Möglichkeiten Javascript im HTML einzubinden
  - Inline
  - Internal
  - Externe js Files
- Die Stelle an der Javascript im HTML eingebunden wird ist von großer Bedeutung
- In den meisten Fällen am Ende des Body Tags empfehlenswert

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width
    <title>Document</title>
</head>
<body>
    <!-- Der Rest des HTML-Dokuments -->

    <script src="script.js"></script>
</body>
</html>
```

Document Object Model (DOM)

- Browser wandelt HTML Datei in eine hierarchische Baum-Struktur um
- Jedes HTML-Element repräsentiert ein Objekt des Baumes
- Objekte besitzen Eigenschaften (Properties)
  - innerHTML, style, firstChild,…
- Objekte besitzen Methoden
  - Click(), setAttribute(), appendChild()
- Javascript kann sowohl Methoden als auch Eigenschaften abrufen und verändern



DOM Elemente selektieren

- Um einzelne Elemente im DOM zu selektieren können folgende Methoden verwendet werden
  - getElementsByTagName
  - getElementsByClassName
  - getElementById
  - querySelector
  - querySelectorAll
  - getElementsByName (wird hauptsächlich für Formularelemente verwendet)
- Elemente werden auch häufig als Nodes bezeichnet

Get Elements By Tag Name

- Mit document.getElementsByTagName(tagname) erhalten wir eine Liste an Elementen.
- **Achtung, diese Liste (HTMLCollection) ist kein echtes Array, nur Array-ähnlich.**
- Man kann mit .length die Länge abprüfen, und mit [i] auf bestimmte Stellen zugreifen, aber keine Array-Methoden wie z.b. .forEach() benutzen.
- for-Schleifen können aber verwendet werden.

```js
const nodes = document.getElementsByTagName("article");

for (let node of nodes) {
    node.style.backgroundColor = '#ffff00';
}
```

```html
<article id="el1">
    Artikel A
</article>
<article id="el2" class="demo">
    Artikel B
</article>
<footer id="el3" class="demo test">
    Footer
</footer>
```

Get Elements By Class Name

- Mit document.getElementsByClassName(classname) erhalten wir eine Liste an Elementen.
- Ergebnis enthält erneut eine HTML-Collection und verhält sich genauso wie getElementsByTagName(tag)

```
const nodes = document.getElementsByClassName("demo");

for (let node of nodes) {
    node.style.backgroundColor = '#ffff00';
}
```

```
<article id="el1">
    Artikel A
</article>
<article id="el2" class="demo">
    Artikel B
</article>
<footer id="el3" class="demo test">
    Footer
</footer>

```

Get Element By ID

- Mit document.getElementById(ID) erhalten wir genau ein Element oder null, je nachdem ob es ein Element mit der angegebenen ID gibt.
- Wir können direkt auf das Ergebnis zugreifen und Eigenschaften oder Methoden abrufen

```
1 const node = document.getElementById("el1");
2 if (node) {
3     node.style.backgroundColor = '#ffff00';
4 }
```

```
1 <article id="el1">
2     Artikel A
3 </article>
4 <article id="el2" class="demo">
5     Artikel B
6 </article>
7 <footer id="el3" class="demo test">
8     Footer
9 </footer>
10
```

Query Selector (All)

- Mit **document.querySelector(query)** können **CSS-Selektoren** als query verwendet werden.
- Man erhält **ein Element** oder null.
- Gibt es mehrere Elemente die auf die query zutreffen, bekommt man nur das erste davon.

```
const node = document.querySelector("article.demo");
if (node) {
    node.style.backgroundColor = '#ffff00';
}

```

- Mit **document.querySelectorAll(query)** können ebenfalls CSS-Selektoren als query verwendet werden.
- Man erhält hier eine **Liste an Elementen** (wiederrum kein echtes Array).

```
const otherNodes = document.querySelectorAll("#el1, #el3");
for (let otherNode of otherNodes) {
    otherNode.style.fontWeight = 'bold';
}
```

Elemente bearbeiten

- Elemente können auf verschiedene Weisen bearbeitet werden:
- **style** lässt uns die Stile auslesen und verändern (Achtung, nur inline Styles auslesbar!)
  - die Schreibweise einzelner Stile kann leicht abweiche: Stile, die man in CSS mit – im Namen schreibt, schreibt man in diesem Objekt in camelCase (z.B. background-color in CSS entspricht backgroundColor in JavaScript).
- className bzw. classList lässt uns die Klassen eines Elements lesen und verändern
- innerText bzw. innerHTML lässt uns den Inhalt des Elements verändern
- value lässt uns den Wert eines Input-Elements auslesen

```
const el1 = document.querySelector("#el1")
el1.style.backgroundColor = "#ffff00";

console.log(document.querySelector("#el3").className);
document.querySelector("#el3").classList.add("a");
document.querySelector("#el3").classList.remove("test");
console.log(document.querySelector("#el3").className);

document.querySelector("#el1").innerHTML = "<b>Hallo</b>";
document.querySelector("#el2").innerText = "<b>Hallo</b>";

console.log(document.querySelector("#inputEl").value);
```

```
<article id="el1">
    Artikel A
</article>
<article id="el2" class="demo">
    Artikel B
</article>
<footer id="el3" class="demo test">
    Footer
</footer>
<input id="inputEl" value="Eingabe"></input>
```

Übung zu DOM Manipulation Teil 1

- Schreibe eine JavaScript-Datei (index.js) und manipuliere den DOM, um Folgendes zu erreichen:
- Wähle das <p>-Element mit der ID "output" aus und ändere seinen Textinhalt auf "Text geändert!".
- Wähle das <h1>-Element innerhalb des mit der ID "main-container" aus und ändere seinen Textinhalt auf "DOM-Manipulationsübung abgeschlossen!".
- Wähle das <button>-Element mit der ID "changeTextButton" aus und ändere seinen Textinhalt auf "Klick mich!„ und seine Hintergrundfarbe auf blau.

```html
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, ini
  <title>DOM-Manipulationsübung</title>
</head>
<body>
<h1>Willkommen zur DOM-Manipulationsübung</h1>
  <div id="main-container">
    <h1>Test</h1>
    <p id="output">Dies ist ein Absatz.</p>
    <button id="changeTextButton">Text ändern</button>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

Elemente erstellen

- Mit document.createElement(tagname) kann man Elemente erstellen.
- Mit .appendChild(createdElement) kann man sie an gewünschter Stelle einfügen, es hängt das element immer als letztes Children an.
- Mit .insertBefore(createdElement, otherChild) kann man das Element auch an gewünschter Stelle in den Children platzieren.

```
1 const listItem = document.createElement("li");
2 listItem.innerText = "...oder 4";
3
4 const otherItem = document.createElement("li");
5 otherItem.innerText = "Elementen";
6
7 const list = document.querySelector("ul");
8 list.appendChild(listItem);
9 list.insertBefore(otherItem, listItem);
```

```
1 <ul>
2   <li>Eine Liste</li>
3   <li>mit 3</li>
4 </ul>
5
```

Elemente löschen

- Mit .remove() kann ein Element aus dem DOM gelöscht werden.
- Das gleiche kann auch mit Parent.removeChild(node) erzielt werden. Letztere gibt das entfernte Element zurück, so dass man es an anderer Stelle wieder einfügen kann.

```
const lastItem = document.querySelector("li:last-child");
lastItem.remove();

const firstItem = document.querySelector("li:first-child");
const list = document.querySelector("ul");
const removedItem = list.removeChild(firstItem);
list.appendChild(removedItem);
```

```
<ul>
  <li>1</li>
  <li>2</li>
  <li>3</li>
  <li>4</li>
</ul>
```

Events

- Ein Event ist ein Ereignis, das meist durch eine Interaktion des Benutzers ausgelöst wird, z.B der Nutzer drückt einen Button.
- JavaScript erlaubt es uns, auf solche Ereignisse zu reagieren, und in Folge dessen Code auszuführen.
- Dabei gibt es verschiedene Wege, einen sogenannten
- Event-Listener zu definieren:
  - Direkt im HTML als Attribut (nicht empfohlen!)
  - Setzen des Attributs mit JavaScript
  - Explizite setzen eines Event-Listeners (der bevorzugte Weg):

```
function inHtml() {
    console.log('In HTML, function name must match');
}

function inJs() {
    console.log('In JS! This is better readable!');
}

document.querySelector('#B').onclick = inJs;

document.querySelector('#C')
        .addEventListener('click', inJs);
```

```
<input value="A" onclick="inHtml()" type="button">
</input>
<input value="B" id="B" type="button"></input>
<input value="C" id="C" type="button"></input>
```

Eventübersicht

- Es gibt in JavaScript eine Vielzahl an Events, hier eine kurze Übersicht über die wichtigsten:

  - click: Klick auf ein Element
  - contextmenu: Klick mit der rechten Maustaste auf ein Element
  - mouseover: Cursor wird über ein Element bewegt
  - mouseout: Cursor verlässt ein Element
  - mousedown: Maustaste wird gedrückt
  - mouseup: Maustaste wird losgelassen
  - mousemove: Maus wird bewegt
  - dblclick: Doppelklick auf ein Element
  - submit: Formular wird abgeschickt
  - focus: Anwender setzt den Fokus auf ein Element
  - blur: Ein Element verliert den Fokus
  - keydown: Drücken einer Taste auf der Tastatur
  - keyup: Loslassen einer Taste auf der Tastatur
  - load: Wird ausgelöst, wenn der Inhalt geladen ist

- Wird der Event-Listener als Attribut auf das Element gesetzt, so wird "on" vor den Eventnamen gesetzt (siehe vorige Folie für ein Beispiel).

Übung zu DOM Manipulation Teil 2

- Aktualisiere deine index.js-Datei, um die Ereignisbehandlung einzubeziehen:
- Füge dem main-container einen weiteren Button mit dem Text „Dark Mode“ nur unter Verwendung von Javascript hinzu
- Füge dem neuen Button einen Ereignis Listener hinzu. Wenn darauf geklickt wird, ändert sich Hintergrundfarbe des mit der ID "main-container" zu schwarz und die Schrift darin wird weiß. Der Text innerhalb des Buttons ändert sich zu „Light Mode“
- Wird der Button erneut geklickt dann ändert sich die Hintergrundfarbe wieder auf weiß und der Text des neuen Button zu „Dark Mode“

```html
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, ini
  <title>DOM-Manipulationsübung</title>
</head>
<body>
<h1>Willkommen zur DOM-Manipulationsübung</h1>
  <div id="main-container">
    <h1>Test</h1>
    <p id="output">Dies ist ein Absatz.</p>
    <button id="changeTextButton">Text ändern</button>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

Abschlussübung „The Dice Game“

- Schreibe JS Code um ein Spiel zu implementieren bei dem 2 Würfel zufällig geworfen werden, einer für Spieler 1 und einer für Spieler 2.
- Das Programm soll je nachdem wer die höhere Zahl gewürfelt hat den Sieger ernennen.
- Durch Klick auf einen Button wird das Spiel erneut gestartet
- Das HTML Grundgerüst findet ihr in euren Codebeispielen

Abschlussübung Expert Level „Black Jack“

- Schreibe eine Webanwendung um ein Black Jack Spiel zu simulieren, bei der ein Spieler Black Jack gegen die Bank spielt
  - Der Spieler versucht so nahe wie möglich auf 21 zu kommen
  - Die Bank zieht Karten, so lange bis sie 16 oder mehr auf der Hand hat
  - Wenn Spieler oder Bank über 21 kommen, verlieren sie das Spiel
  - Es gewinnt derjenige der näher an 21 dran ist
- Der Spieler startet mit einem vorgegebenen Chipstack
- In jeder Runde wird ein fixer Einsatz seiner verfügbaren Chips eingesetzt
- Der Chipstack des Spielers verändert sich nach jeder Runde
  - Gewinnt der Spieler bekommt er seinen doppelten Einsatz zurück
  - Verliert er die Runde verliert er auch seinen Einsatz

Event Object

- Man bekommt auch weitere Informationen über das Event, wie zum Beispiel die gedrückte Taste bei einem keydown.
- Diese Informationen findet man im Event Objekt, das dem Listener immer als erster Parameter übergeben wird.
- Der Inhalt des Objekts unterscheidet sich je nach Event-Typ (mouse, keyboard, etc.)
- Mit dem Keyword **this** kann auf das auslösende HTML Element referenziert werden

```
1 function myListener(event) {
2     console.log('keyup event:', event);
3     console.log('keyup event.code:', event.code);
4     console.log('keyup event.key:', event.key);
5     console.log('keyup event.keyCode:', event.keyCode);
6 }
7
8 document.addEventListener('keyup', myListener);
```

Event Bubbling

- Ein Event wie zum Beispiel ein click wird nicht nur für das direkt angeklickte Element ausgelöst, sondern auch nacheinander auf alle Parent-Elemente.
- Beobachte das Verhalten, wenn du auf die verschiedenen verschachtelten Boxen klickst:
- Mit der Funktion event.stopPropagation() kann man verhindern, dass das Event an die Elternelemente weiter gegeben wird.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Event Bubbling Example</title>
    <style>
        div {
            padding: 20px;
            border: 1px solid #ccc;
            margin: 10px;
        }
    </style>
</head>
<body>

    <div id="parent">
        <p style="border: 1px solid">Click me!</p>
    </div>

    <script>
        // Get references to the parent and child elements
        var parentElement = document.getElementById('parent');
        var childElement = document.querySelector('p');

        // Add click event listeners to both elements
        parentElement.addEventListener('click', function(event) {
            alert('Parent Element Clicked!');
        });

        childElement.addEventListener('click', function(event) {
            alert('Child Element Clicked!');
        });
    </script>

</body>
</html>
```

Audio durch Event

- Man kann auch Audio durch Events abspielen lassen
- Dafür wird innerhalb der Funktion die der Event Listener aufruft folgender Code ausgeführt:

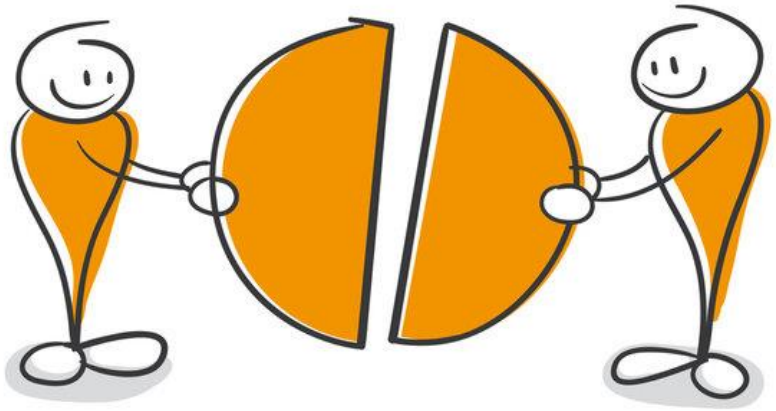
```
let sound = new Audio("sounds/tom-4.mp3")
sound.play();
```

Übung Drumkit gemeinsam

- Programmiere ein digitales Schlagzeug als Webapplikation
- Klänge des Schlagzeugs werden abgespielt durch:
  - Klicken auf Bilder der unterschiedlichen Drums
  - Betätigen einer der zugewiesenen Tasten des Keyboards

Tipp: Teile und Herrsche

- Zerlege ein komplexes Problem in kleinere, überschaubare Teilprobleme.
  Dies erleichtert das Verständnis und den Lösungsprozess.
- Gehe dabei iterativ auf mehreren Ebenen vor
- Beispiel im Kontext Drumkit:
  - 1. Frontend – Aufbau der Seite mit allen Drums
  - 2. Backend – Javascript um Sounds abzuspielen
    - 2.1 Sounds beim Klicken auf ein Bild
    - 2.2 Sound beim Drücken der Tastatur
      - 2.2.1 Aktion beim Drücken einer beliebigen Taste
      - 2.2.2 Sound beim Drücken einer beliebigen Taste
      - 2.2.3 Unterschiedliche Sounds beim Drücken von unterschiedlichen Tasten
      - …



Fehlerbehandlung durch Try-catch

- Tritt im Programmcode ein Fehler auf, führt das für gewöhnlich zum Absturz des Programmes, bzw. zum Programmende.
- Die meisten Fehlerquellen sollten bereits im Code ausgemärzt werden (Fehler des Programmierers), jedoch gibt es unvorhersehbare Fehler, beispielsweise dass ein Server nicht erreichbar ist oder eine unerwartete Antwort schickt.
- Für solche Fälle gibt es das sogenannte try-catch, welches erlaubt, beliebige Fehler abzufangen und zu behandeln.
- Hierbei wird der Fehleranfällige Code in den try Code-Block gesetzt, tritt ein Fehler auf, wird dieser abgebrochen und man gelangt in den catch Code-Block.
- Dieser catch Code-Block erhält zudem einen Parameter (Variable), in dem man weitere Informationen zum aufgetretenen Fehler findet

Try-catch in Aktion

- Das Programm versucht den Code im try-Block auszuführen
- Kommt es zum Fehler, bricht es den Block ab und geht in den catch-Block
- Ein optionaler finally-Block wird immer ausgeführt (Code nach dem try-catch auch)

```
1 const a = 1;
2
3 try {
4     a = 2; // leads to error
5 } catch (error) {
6     console.log(error.message);
7 }
8
9 console.log('Kommt immer');
```

Fehler auslösen - Throw

- Möchte man selbst einen Fehler auslösen, kann man dies mit dem Schlüsselwort **throw** machen.
- Diesem **throw** muss ein Objekt der Klasse Error folgen,
- Dieses erzeugt man mit dem new Schlüsselwort, und den runden Klammern hinter dem Klassennamen.
- Innerhalb dieser Klammern kann man dem Error eine Nachricht geben, warum der Code fehlgeschlagen ist.

```
let num = parseFloat(await prompt('Please enter a positive number: '));
try {
    if (num < 0) {
        throw new Error('Number must be positive!');
    }
    console.log('The root is', Math.sqrt(num));
} catch (error) {
    console.log(error.message);
}
```

Übung zu Fehlerbehandlung

- Schreibe eine Funktion die 2 Zahlen als Parameter übergibt und beide Zahlen miteinander dividiert (erste / zweite)
- Füge eine Fehlerbehandlung ein für den Fall dass:
  - Anstelle von Zahlen, Strings übergeben wurden
  - Die 2. Zahl die übergeben wurde = 0 (Division durch 0)

Viel Erfolg beim Entwickeln!