DOM Manipulation mit JavaScript

Einführung und Praktische Anwendungen

- **Dozent**: Jacob Menge

- **Datum**: 18.10.2024

Agenda

1. Einführung in DOM und JavaScript
2. Praktische Beispiele:
    - Elemente selektieren und manipulieren
    - Elemente hinzufügen und entfernen
    - Event Listener
3. Pause (15 min.)
4. Übung: Ihr erstellt Interaktive Anwendungen mit JavaScript (2 std.) (dazwischen 20 min. Pause)
5. Besprechung der Aufgaben und Vorstellung der Lösungen

Einführung in DOM und JavaScript

- **DOM**: Document Object Model
- **JavaScript**: Sprache zur Manipulation des DOM
- Seit den 1990er Jahren in Webbrowsern integriert
- Ermöglicht dynamische und interaktive Webanwendungen

Warum ist DOM-Manipulation wichtig?

- Ermöglicht **Interaktivität** auf Webseiten
- Zentrale Rolle in **Single-Page Applications** (SPA) und modernen Frameworks wie React oder Vue.js
- Dynamische Anpassung von Inhalten ohne komplette Seitenneuladen (z. B. AJAX)
- Flexibles Hinzufügen, Entfernen und Modifizieren von HTML-Elementen und -Stilen

Wie funktioniert das DOM?

- Der DOM stellt ein **Abbild** des HTML-Dokuments als **Baumstruktur** dar
- Jeder Knoten im Baum ist ein **HTML-Element** oder ein **Textknoten**
- JavaScript bietet Zugriff auf diese Struktur über das `document` -Objekt

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Beispiel-Überschrift</h1>
    <p>Beispiel-Text</p>
  </body>
</html>
```

Historischer Überblick

- **1990er Jahre**: DOM erstmals eingeführt, um statische HTML-Dokumente zu manipulieren
- **2000er Jahre**: DOM wird zur Standardtechnologie für dynamische Inhalte (DHTML)
- **Moderne Webentwicklung**: DOM-APIs sind Teil jedes Browsers und werden für Frameworks und Bibliotheken genutzt

Wichtige Konzepte der DOM-Manipulation

- **Selektieren**: Zugriff auf spezifische Elemente im DOM
- **Manipulieren**: Ändern von Inhalten, Stilen und Attributen
- **Ereignissteuerung**: Reaktion auf Benutzeraktionen wie Klicks und Tastatureingaben
- **Erweitern**: Neue Elemente hinzufügen oder bestehende entfernen

Methode: `document.getElementById('id')`

- Findet ein Element anhand seiner ID.
- Gibt das erste gefundene Element oder `null` zurück.

Beispiel:

```
<h1 id="header">Hallo Welt</h1>
<script>
  const header = document.getElementById('header');
  console.log(header.textContent); // Ausgabe: Hallo Welt
</script>
```

Methode: `document.getElementsByClassName('class')`

- Findet alle Elemente, die einer bestimmten Klasse angehören.
- Gibt eine HTMLCollection zurück (ähnlich wie ein Array).

Beispiel:

```
<div class="item">Item 1</div>
<div class="item">Item 2</div>
<script>
  const items = document.getElementsByClassName('item');
  console.log(items[0].textContent); // Ausgabe: Item 1
</script>
```

Methode: `document.getElementsByTagName('tag')`

- Gibt eine Sammlung aller Elemente mit einem bestimmten Tag-Namen zurück.
- Gibt eine HTMLCollection zurück.

Beispiel:

```
<p>Absatz 1</p>
<p>Absatz 2</p>
<script>
  const paragraphs = document.getElementsByTagName('p');
  console.log(paragraphs.length); // Ausgabe: 2
</script>
```

Methode: `document.querySelector('selector')`

- Wählt das erste Element, das einem CSS-Selektor entspricht.
- Gibt `null` zurück, wenn kein Element gefunden wird.

Beispiel:

```
<p class="text">Textabschnitt</p>
<script>
  const paragraph = document.querySelector('.text');
  console.log(paragraph.textContent); // Ausgabe: Textabschnitt
</script>
```

Methode: `document.querySelectorAll('selector')`

- Gibt alle Elemente zurück, die einem CSS-Selektor entsprechen.
- Gibt eine NodeList zurück (ähnlich wie ein Array).

Beispiel:

```
<p class="text">Absatz 1</p>
<p class="text">Absatz 2</p>
<script>
  const paragraphs = document.querySelectorAll('.text');
  paragraphs.forEach(p => console.log(p.textContent));
  // Ausgabe: Absatz 1, Absatz 2
</script>
```

Methode: `element.textContent`

- Ändert oder gibt den Textinhalt eines Elements zurück.
- Enthält nur Text, keine HTML-Struktur.

Beispiel:

```
<p id="text">Ursprünglicher Text</p>
<script>
  const text = document.getElementById('text');
  text.textContent = 'Neuer Text';
</script>
```

Methode: `element.innerHTML`

- Ändert oder gibt den HTML-Inhalt eines Elements zurück.
- Kann auch HTML-Tags enthalten.

Beispiel:

```
<div id="container"></div>
<script>
  const container = document.getElementById('container');
  container.innerHTML = '<p>Neuer Absatz</p>';
</script>
```

Methode: `element.style.property`

- Setzt oder liest eine CSS-Eigenschaft eines Elements.
- Beispiel: `element.style.color = 'blue';`

Beispiel:

```
<div id="box">Box</div>
<script>
  const box = document.getElementById('box');
  box.style.backgroundColor = 'yellow';
</script>
```

Methode: `element.setAttribute('attribute', 'value')`

- Setzt ein Attribut auf einen bestimmten Wert.

Beispiel:

```
<a id="link" href="#">Link</a>
<script>
  const link = document.getElementById('link');
  link.setAttribute('href', 'https://www.example.com');
</script>
```

Methode: `element.getAttribute('attribute')`

- Liest den Wert eines bestimmten Attributs.

Beispiel:

```
<a id="link" href="https://www.example.com">Link</a>
<script>
  const link = document.getElementById('link');
  console.log(link.getAttribute('href'));
</script>
```

Methode: `document.createElement('tag')`

- Erstellt ein neues HTML-Element.

Beispiel:

```
<script>
  const newDiv = document.createElement('div');
  newDiv.textContent = 'Neues Element';
  document.body.appendChild(newDiv);
</script>
```

Methode: `element.appendChild(childElement)`

- Fügt ein Kind-Element zu einem übergeordneten Element hinzu.

Beispiel:

```
<ul id="list"></ul>
<script>
  const list = document.getElementById('list');
  const newItem = document.createElement('li');
  newItem.textContent = 'Neues Item';
  list.appendChild(newItem);
</script>
```

Methode: `element.removeChild(childElement)`

- Entfernt ein Kind-Element von einem übergeordneten Element.

Beispiel:

```
<ul id="list">
  <li>Item 1</li>
</ul>
<script>
  const list = document.getElementById('list');
  const item = list.querySelector('li');
  list.removeChild(item);
</script>
```

Methode: `element.addEventListener('event', function)`

- Fügt einem Element einen Event Listener hinzu.

Beispiel:

```
<button id="clickMe">Klick mich</button>
<script>
  const button = document.getElementById('clickMe');
  button.addEventListener('click', () => {
    alert('Button wurde geklickt!');
  });
</script>
```