Tutorium zu Kapitel WT:V

- DOM-Manipulation
- `EventListener` & Callbacks
- AJAX & XMLHttpRequest

DOM-Manipulation

Elemente erzeugen, manipulieren und löschen

- Element erzeugen:

```
var newElement = document.createElement("tag-name")
```

- Element ins DOM einhängen:

```
document.body.appendChild(element)
```

- Element löschen:

```
newElement.remove()
```

- HTML-Klasse setzen:

```
newElement.classList.add("class-name")
```

- HTML-Klasse entfernen:

```
newElement.classList.remove("class-name")
```

DOM-Manipulation

querySelector, querySelectorAll

- Bestimmtes Element im DOM finden:

```
var element = document.querySelector(cssSelector)
```

Beispiel:

```
var button = document.querySelector("button")
```

- Liste von Elementen im DOM finden:

```
var elements = document.querySelectorAll(cssSelector)
```

Beispiel:

```
var anchors = document.querySelectorAll("a")
```

`EventListener` & Callbacks

Theorie

- `EventListener` ermöglichen es, auf bestimmte Ereignisse zu reagieren.

```
target.addEventListener(type, callback);
|                       |     |
|                       |     Callback-Funktion, mit der
|                       |     auf das Event reagiert wird
|                       |
|                       Event-Typ, auf den reagiert werden soll
|
```

  Auf diesem Event-Ziel (i.d.R ein DOM-Knoten) wird der EventListener registriert und horcht fortan auf Ereignisse eines bestimmten Typs.

- `callback` ist dabei eine Referenz auf eine Funktion, die immer dann aufgerufen wird, wenn das Ereignis ausgelöst wird

`EventListener` & Callbacks

Beispiel

- Variante 1: Benannte Funktion als Callback

```
document.body.addEventListener("click", removeElement);

function removeElement(event) {
    event.target.remove();
}
```

- Variante 2: Anonyme Funktion als Callback

```
document.body.addEventListener("click", function(event) {
    event.target.remove();
});
```

- Grober Ablauf:
  1. Innerhalb des Event-Ziels `document.body` (das HTML `body`-Element), ...
  2. erwarte das `click`-Ereignis ...
  3. und führe die Callback-Funktion aus.

- Für das Absenden eines Formulars gibt es das submit-Ereignis.

AJAX

Asynchrone HTTP-Requests mit `XMLHttpRequest`

- Standardmäßig wird bei HTTP-Requests eine Ressource im aktuellen Fenster geladen
- AJAX („Asynchronous JavaScript and XML“)
  erlaubt das Verarbeiten von Requests, ohne die aktuelle Seite zu verlassen
- Solche Requests werden zudem asynchron ausgeführt.
  Sie unterbrechen nicht die Ausführung von synchronem Code.
- In JavaScript steht für die Umsetzung von AJAX das `XMLHttpRequest`-Objekt zu Verfügung.

AJAX

`XMLHttpRequest`-Beispiel: Schritt für Schritt

- aus der `feedreader.js` von der Kurswebseite:

```
function postEntry(url, data, encoding, processResponseXML) {
    var req = new XMLHttpRequest();
    if (req) {
        req.onreadystatechange =
            createCallbackFunction(req, processResponseXML);
        req.open("POST", url);
        req.setRequestHeader("Content-Type", encoding);
        req.send(data);
    }
}
```

AJAX

`XMLHttpRequest`-Beispiel: Schritt für Schritt

- `function postEntry(url, data, encoding, processResponseXML)`
  - `processResponseXML`: Callback-Funktion
  - `encoding`: Encoding des Formulars
  - `data`: Zu sendende Formulardaten
  - `url`: Entspricht der URI im HTTP-Request
- `url`: absolute oder relative Pfadangabe
- `data`: Formulardaten in der Form `key1=value(&key2=value)*`
- `encoding`: Setzt `Content-Type` des POST-Requests auf den MIME Type des Inhalts. Dieser ist für Formulare mit der Methode `post` Standardmäßig `application/x-www-form-urlencoded`. Mit diesem Content-Type erwartet der HTTP-Server die POST-Daten in der Form `key1=value(&keyN=value)*`
- `processResponseXML`:
  Funktion, die die HTTP-Response des Servers verarbeiten soll

AJAX

`XMLHttpRequest`-Beispiel: Schritt für Schritt

- Ein neues `XMLHttpRequest`-Objekt wird erzeugt.
  Nur, wenn dieser Schritt erfolgreich war, wird der folgende Code ausgeführt.

```
var req = new XMLHttpRequest();
if (req) {
    // ...
}
```

- Die `onreadystatechange`-Eigenschaft hält eine Methode, die immer dann aufgerufen wird, wenn sich der Wert der `req.readyState`-Eigenschaft ändert

```
req.onreadystatechange =
    createCallbackFunction(req, processResponseXML);
```

- Die Callback-Funktion soll den Status des HTTP-Requests überwachen.
  Ist eine HTTP-Response verfügbar, soll diese von der `processResponseXML`-Funktion weiterverarbeitet werden

AJAX

- Initialisiert ein neues HTTP-Request

```
req.open("POST", url);
```

- Setzt das HTTP-Headerfeld `Content-Type` auf den Wert in `encoding`.

```
req.setRequestHeader("Content-Type", encoding);
```

- Sendet das HTTP-Request an den Server. Das optionale Argument enthält den Inhalt des Request-Body. Bei asynchronen XMLHttpRequests wartet die Methode nicht auf eine Antwort vom Server sondern gibt sofort zurück.

```
req.send(data);
```

- Nach dem Aufruf von `req.send` ist die `postEntry`-Funktion abgearbeitet und gibt an den aufrufenden Kontext zurück. Dort kann nun sofort weiterer Code ausgeführt werden, während im Hintergrund das HTTP-Request vom Server bearbeitet wird.

- Die Callback-Funktion in `req.onreadystatechange` wird weiterhin bei jeder Änderung von `req.readyState` ausgeführt.