Kurze Repetition:

Delegation, Rendern, Fehlersuche

Javascript Delegation

Elemente aus dynamischer Liste löschen

```
<ul id="parent-list">
      <li id="post-1">Item 1 <button>Löschen</button></li>
      <li id="post-2">Item 2 <button>Löschen</button></li>
      <li id="post-3">Item 3 <button>Löschen</button></li>
</ul>
```

```
<ul id="parent-list">
      <li id="post-1">Item 1 <button>Löschen</button></li>
      <li id="post-2">Item 2 <button>Löschen</button></li>
      <li id="post-3">Item 3 <button>Löschen</button></li>
</ul>
```

Elemente aus dynamischer Liste löschen

Problem:

- Ihr müsst für jedes Element, dass gelöscht werden kann, einen EventListener hinzufügen
- Für nachträglich hinzugefügte Elemente müsst ihr auch einen EventListener hinzufügen

```
<ul id="parent-list">
      <li id="post-1">Item 1 <button>Löschen</button></li>
      <li id="post-2">Item 2 <button>Löschen</button></li>
      <li id="post-3">Item 3 <button>Löschen</button></li>
</ul>
```

Event Delegation

Ansatz Event Delegation:

- Event Listener auf nächst höheres Element setzen
- Auf Event Bubbling warten
- Überprüfen ob event.target gewünschtem CSS-Selektor entspricht
- Falls ja, Code ausführen

Vorteile:

- Nur 1 Event Listener nötig
- Keine Event Listener für Einträge müssen hinzugefügt oder entfernt werden
- Funktioniert mit dynamischer Anzahl von Elementen

Nachteil dieser Lösung:

- Komplizierter zu verstehen

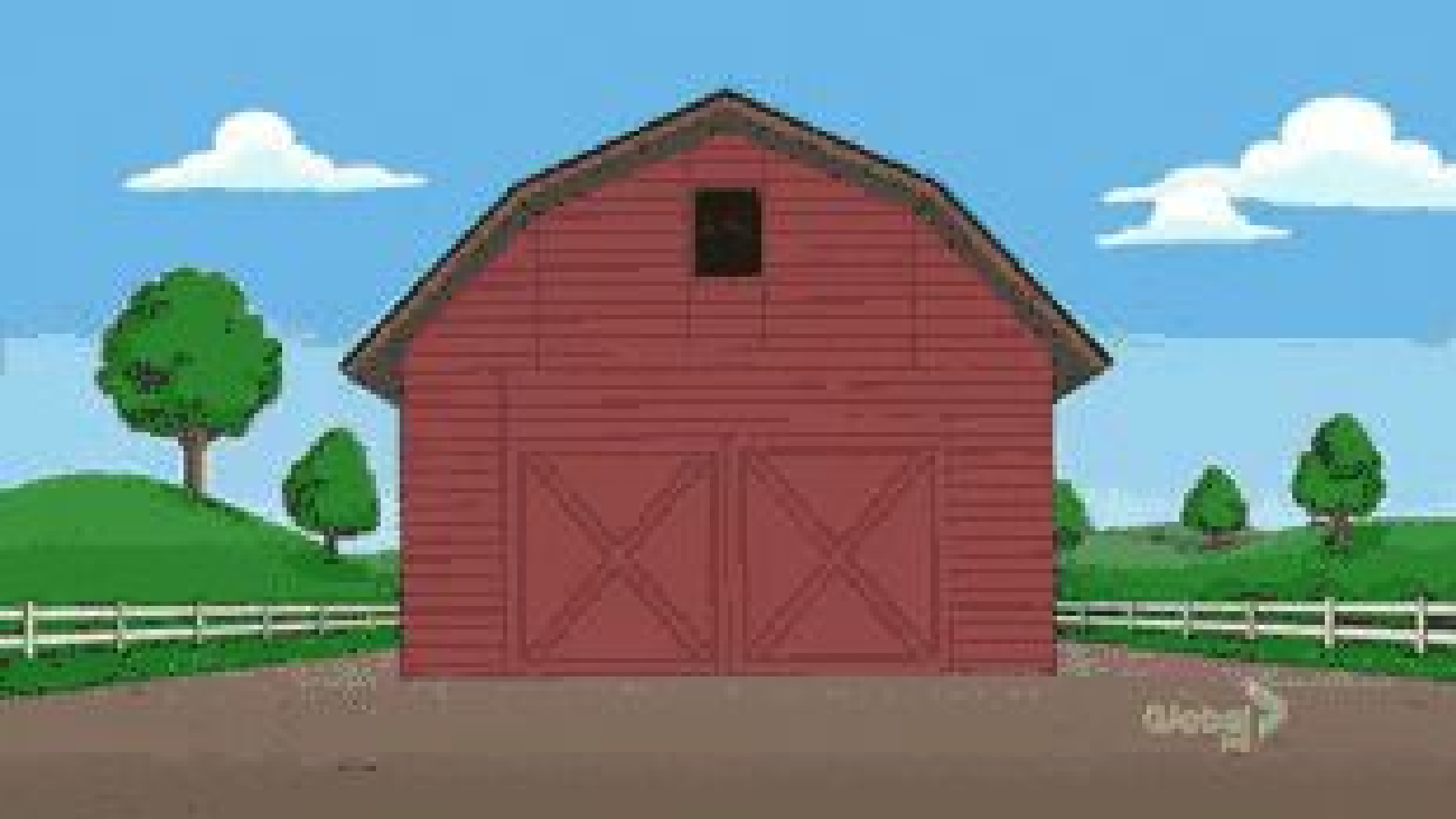
Event Delegation

CSS Selektor der auf Button zutrifft

```
document.querySelector("#parent-list")
.addEventListener("click", function(event) {
  if (event.target && event.target.matches("li button")) {
    removeItem(event);
  }
});
```

```
<ul id="parent-list">
      <li id="post-1">Item 1 <button>Löschen</button></li>
      <li id="post-2">Item 2 <button>Löschen</button></li>
      <li id="post-3">Item 3 <button>Löschen</button></li>
</ul>
```

Listen
Rendern



Listen Render Ansatz

Ansatz Listen Rendern

- Liste der Todos wird bei jeder Änderung neu aufgebaut
- D.h. zuerst werden **ALLE** alten 's gelöscht
- Danach werden neue eingefügt
- Aufbau wird von einer eigenen Funktion gehandhabt

Vorteile:

- Wir müssen nur noch mit dem Array arbeiten und können dann die render() Funktion aufrufen
- Dadurch können wir hinzufügen, entfernen, sortieren einfacher implementieren

Array.forEach()

```
var array1 = ['a', 'b', 'c'];

array1.forEach(function(element) {
  console.log(element);
});
```

```
var array1 = ['a', 'b', 'c'];

for (var i = 0; i < array1.length;
i++) {
  console.log(array1[i]);
}
```

Listen Render Ansatz

```
function renderList() {
  var liste = document.querySelector('#liste');
  liste.innerHTML = ''; // alte Liste löschen
  simpsons.forEach(function (elem, index) {
    var li = document.createElement('li'); // Neues Element erzeugen
    li.innerHTML = elem.firstname + ' ' + elem.name + ' <button>Löschen</button>'; // Daten abfüllen
    li.id = index; // Id setzen (später wichtig für's löschen)
    liste.appendChild(li); // Element an Fragment hängen
  });
}
```

Fehlersuche

Fehler

```
console.log(document.querySelector('gibtsNicht').firstChild);
```

```
TypeError: Cannot read property 'firstChild' of null

TypeError: Cannot read property 'firstElementChild' of null

TypeError: Cannot read property 'lastChild' of null

TypeError: Cannot read property 'addEventListener' of null

TypeError: Cannot read property 'firstChild' of null
```

Fehler

```
var x = [1,2,3,4];

x();
```

```
TypeError: x is not a function

TypeError: undefined is not a function

TypeError: null is not a function
```

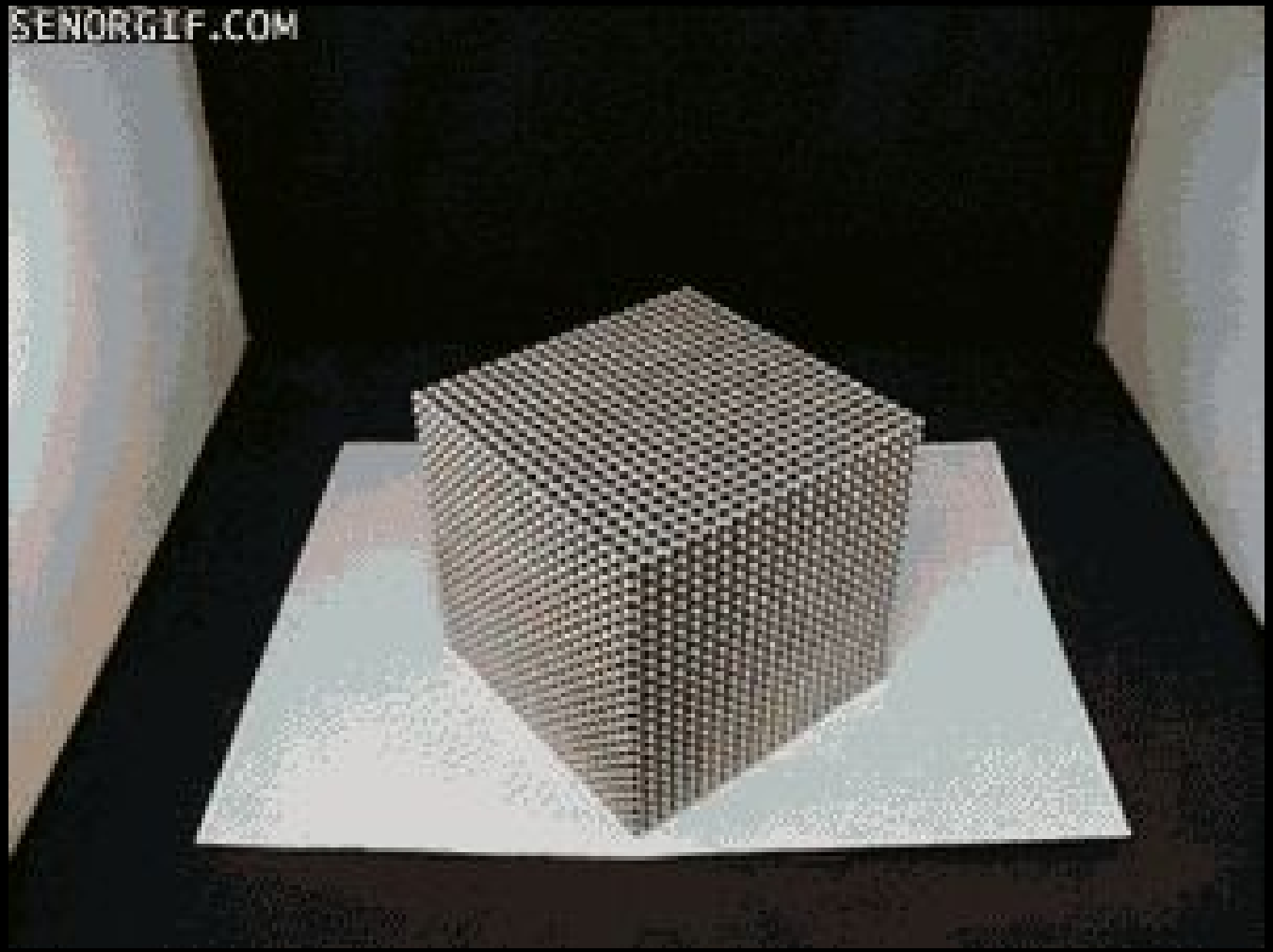
Fehler

```
foo.substring(1); // ReferenceError: "foo" is not defined
```

```
function numbers() {
  var num1 = 2,
      num2 = 3;
  return num1 + num2;
}
```

```
console.log(num1); // ReferenceError num1 is not defined.
```

Divide & Conquer



Divide and Conquer

Problemlösungsstrategie die schneller zum Ziel führt

```
document.querySelector('#liste')
.addEventListener('onclick', function(event) {
  event.target.parentNode.removeChild(event.target);
});
```

Problem: Element wird nicht entfernt!

Divide and Conquer

```
document.querySelector('#liste')
.addEventListener('onclick', function(event) {
  console.log('test');
  event.target.parentNode.removeChild(event.target);
});
```

2 mögliche Resultate

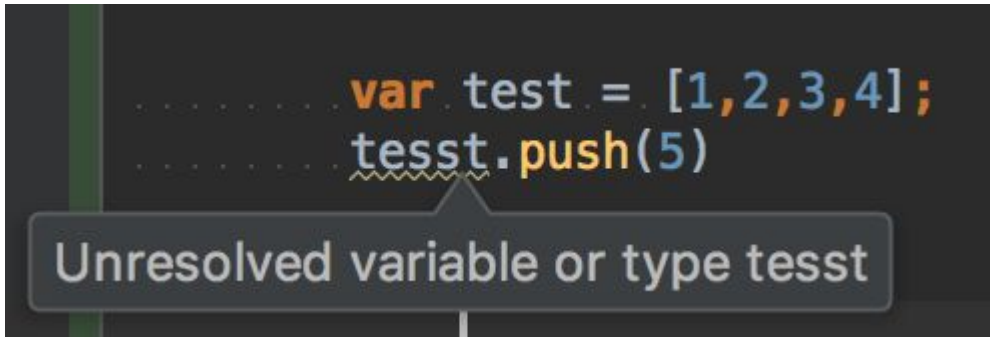
'test' erscheint NICHT auf der Konsole:
=> Methode wird gar nie aufgerufen,
=> Problem liegt am Eventlistener

'test' erscheint auf der Konsole:
=> Methode wird aufgerufen
=> Code zur entfernung des Elements funktioniert nicht richtig

IDE Tricks

```
var test = [1,2,3,4];
tesst.push(5)
```

Gelb unterstrichen = Variable wurde nicht initialisiert



IDE Tricks

```
var teest = [1,2,3,4];
```

Grauer Text = Variable wird nicht verwendet

Fragen zu Übungen?

Module Pattern

Zeit aufzuräumen

WEB PROFESSIONALS

Warum Module?

- Bessere lesbarkeit
- Bessere struktur
- Weniger Konflikte mit Git
- Sinnvolle Aufteilung des Codes nach Aufgaben
- Wiederverwendbarkeit
- Keine Wiederholung von gleichem Code
    - Einfachere Fehlerkorrektur
    - Schnelleres Arbeiten

Module Pattern Beispiel

```
(function () {
  var start = function () {
    console.log('Hallo');
  };
  start();
})()
```

=> "Hallo" auf der Konsole

- Verhindert das unsere Variablen den Globalen Namespace verschmutzen
- Unser ganzes Modul läuft in einer grossen Funktion, welche **beim einbinden direkt ausgeführt wird**
- Innerhalb der Funktion, gibt es weitere Funktionen

Module Pattern Beispiel

- Module können in sich geschlossen sein z.B. unser Hauptmodul
- Module können Funktionen für andere Module anbieten

Tools Modul

```
var Tools = (function () {
  var removeElement = function (element) {  // Code  };
  var delegate = function (target, callback) { // Code };
  return { // Hier werden die "public" Methoden definiert
    delegate: delegate,
    removeElement: removeElement
  };
})()
//Mit Tools.FUNKTIONSNAME kann das Modul aufgerufen werden
```

```
var App = (function () {
  var start = function () {
    console.log('Hallo');
  };
  return { start: start };
})()
App.start()
```

Hauptmodul

tools.js

```
var Tools = (function () {
  var removeElement = function (element) {
    // Remove Element Code hier
  };
  var delegate = function (target, callback) {
    // Delegate Code hier
  };
  return {
    delegate: delegate,
    removeElement: removeElement
  };
})()
```

app.js

```
var App = (function (t) { // über t sind alle Funktionen verfügbar aus tools.js
  var parentList = document.querySelector('#parent-list');
  var textInput = document.querySelector('#text-input'); var addButton = document.querySelector('#add-item');
  var init = function () {
    addButton.addEventListener('click', addItem); // Hinzufügen Button anbinden (addItem ist nicht dargestellt)
    parentList.addEventListener('click', t.delegate('li button', function(event) { //Alle löschen Buttons anbinden
      t.removeElement(event.target.parentNode);
    }));
  };
  return { // Rückgabe der von aussen verfügbaren Methoden
    init: init
  };
})(Tools);
App.init(); // Initialisierungsfunktion aufrufen
```

Einbettung (extern)

In <head> (sehr schlecht)

```
<html>
  <head>
    <title>JS Repetition</title>
    <script src="app.js"></script>
  </head>
  <body>
    <h1>Titel</h1>
  </body>
</html>
```

Ende <body> (am besten)

```
<html>
  <head>
    <title>JS Repetition</title>
  </head>
  <body>
    <h1>Titel</h1>
    <script src="app.js"></script>
  </body>
</html>
```

Einbettung mehrerer JS Files (mehrerer Module)

```
<html>
  <head>
    <title>Module Pattern</title>
  </head>
  <body>
    <h1>Titel</h1>
    <script src="tools.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

Reihenfolge ist wichtig!

Übungen zu Module Pattern

Kopiert aus dem Google Drive die Datei: **Module/Javascript/lektion7.zip** in euer **htdocs** Verzeichnis in einen eigenen Ordner und macht:

1. Uebung_1_0: Findet die 5 Fehler im Code
2. Uebung_1_1: Erstellt ein Modul tools.js. Darin sollt ihr häufig verwendete Codestücke in wiederverwendbare Funktionen abstrahieren.
   Erstellt von aussen verwendbare Funktionen für:
   - Delegate
   - Remove Element

   In *uebung_1_1.html* hat es Code, mit welchem ihr eure Toolbox Funktionen testen könnt.

Baut eure Todo Liste mit allem gelernten aus

Zuerst: Schliesst die Übungen aus Lektion 6 ab, damit ihr alle techniken beherrscht.

Danach, baut eure ToDo-Liste so um, dass sie folgendes kann:

- Deine Liste ist in 2 Module (App und Tools) aufgeteilt
- Elemente können gelöscht werden (mit delegate & removeElement aus Tools.js)
- Die Elemente werden mit dem Render-Ansatz erstellt
- Elemente können mit Enter hinzugefügt werden
- Die ToDo's werden in der LocalStorage gespeichert (reload Funktioniert)
- **Zusatzaufgabe (freiwillig)**: Unten an der Liste kann zwischen Allen, Offenen und Erledigten Todos gefiltert werden