Kurze Repetition:

Module Pattern

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 Teil-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)

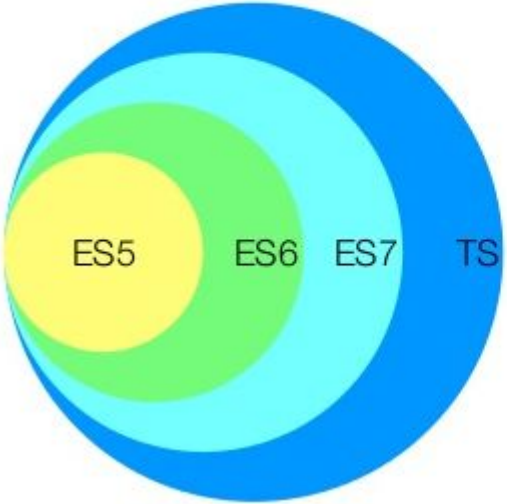
```
<html>
  <head>
    <title>Module Pattern</title>
  </head>
  <body>
    <h1>Titel</h1>
    <script src="tools.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

Reihenfolge ist wichtig!

Fragen zu Übungen?

ES6 vs ES5

ES6, ES7 and TypeScript



ES5: es5.github.io

ES6: git.io/es6features

ES7: bit.ly/es7features

TS: www.typescriptlang.org


ES6
ECMAScript 2015

ES6 vs ES5 - Constants

```
const PI = 3.141593;

PI = 5; // Fehler
```

- Wert der Variable kann später nicht wieder verändert werden

ES6 vs ES5 - Block Scoped Variables

```
for (let i = 0; i < a.length; i++) {
    let x = a[i]
    …
}
for (let i = 0; i < b.length; i++) {
    let y = b[i]
    …
}
console.log(i) // undefined
```

```
var i, x, y;
for (i = 0; i < a.length; i++) {
    x = a[i];
    …
}
for (i = 0; i < b.length; i++) {
    y = b[i];
    …
}
console.log(i) // b.length -1
```

ES6 vs ES5 - Arrow Functions

macht das Selbe wie

```
odds = evens.map(v => v + 1)
pairs = evens.map(v => ({ even: v, odd: v + 1 }))
nums = evens.map((v, i) => v + i)
```

```
odds = evens.map(function (v) { return v + 1; });
pairs = evens.map(function (v) { return { even: v, odd: v + 1 }; });
nums = evens.map(function (v, i) { return v + i; });
```

ES6 vs ES5 - Arrow Functions II

```
nums.forEach(v => {
    if (v % 5 === 0)
        fives.push(v)
});
```

```
nums.forEach(function (v) {
    if (v % 5 === 0)
        fives.push(v);
});
```

ES6 vs ES5 - Default Parameter Handling

```
function f (x, y, ...a) {
    return (x + y) * a.length
}
f(1, 2, "hello", true, 7) === 9
```

```
function f (x, y) {
    var a = Array.prototype.slice.call(arguments, 2);
    return (x + y) * a.length;
};
f(1, 2, "hello", true, 7) === 9;
```

ES6 vs ES5 - Default Parameter Handling

```
function f (x, y = 7, z = 42) {
    return x + y + z
}
f(1) === 50
```

```
function f (x, y, z) {
   y =  y || 7;
   z = z || 42;
   return x + y + z;
};
f(1) === 50;
```

Um so weniger wichtig
der Parameter, desto weiter rechts

ES6 vs ES5 - Template Literals

```
var a = 5;
var b = 10;
console.log(`Fifteen is ${a + b} and not ${2 * a + b}.`);
// "Fifteen is 15 and not 20."
```

```
var a = 5;
var b = 10;
console.log("Fifteen is " + (a + b) + " and not " + (2 * a + b)
+ ".");
// "Fifteen is 15 and not 20."
```

ES6 vs ES5 - Property Shorthand

```
obj = { x, y };
```

```
obj = { x: x, y: y };
```

ES6 vs ES5 - Array.find()

```
[ 1, 3, 4, 2 ].find(x => x > 3) // 4
```

```
[ 1, 3, 4, 2 ].filter(function (x) { return x > 3; })[0];
// 4
```

canluse.com

Javascript.info

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