Web-Programmieren 2

TSBE Frühlingssemester 2016

http://smlz.github.io/tsbe-2016fs/web/

Marco Schmalz

marco.schmalz@gibb.ch

Kursübersicht

1. HTML und CSS, Bootstrap
2. **Repetition HTML/CSS, JavaScript Einführung**
3. AngularJS Browser-Applikation
4. (Auffahrt)
5. Backend mit Elasticsearch

Heute

1. Repetition HTML/CSS
2. Projektarbeit
3. JavaScript Einführung

Tipps für das Projekt

- Projektbericht vorbereiten
- Journal wärend der Arbeit nachführen
- Kolaboration per GitHub?

JavaScript Geschichte

- 1995 Brendan Eich bei Netscape (10 Tage)
- Standardisierte Version: ECMAScript (aktuell ES5)
- Assembler des Webs

Features

- Dynamisch typisiert
- Keine Klassen. Prototypen!
- Funktionen sind auch Objekte
- Läuft im Browser; wird als Sourcecode ausgeliefert, keine Kompilation
- Keine Threads, Event-Basiert

Grösstes Feature

Die Absenz von Features

Wie sieht JS aus?

```
function add(a, b) {
    return a + b;
}

var y = 5;
var x = 3 + y;
var z = add(x, y);

console.log(z);
```

Objekte in JavaScript

Objekte können mit `new` oder mit dem Objekt-Literal `{}` kreiert werden.

```
var x = new Object();
var y = {};
```

```
// Neues Objekt direkt erstellen
var car = {
    make: "Ford",
    model: "Mustang"
};

// Zugriff auf Felder eines Objektes auch mit eckigen Klammern
car['year'] = 1978;

// Methode zum Objekt hinzufügen
car.printInfo = function() {
    console.log(this.make + " " + this.model + " (" + this.year + ")");
}
car.printInfo();
```


JavaScript Arrays

Arrays werden mit den eckigen Klammern erstellt.

```
var a = [1, 2, 3, 4, 5];

// Arrays beginnen wie üblich bei 0
console.log(a[0]);

var i;

for(i = 0; i < a.length; i += 1) {
    console.log(a[i]);
}
```

Zahlen — Ein Trauerspiel

Alle Zahlen in JavaScript sind Fliesskommazahlen. Es gibt keine Integer

```
console.log(0.1 + 0.2);    // 0.30000000000000004
```

Zahlen sind auch Objekte:

```
var x = 123.456;
x.toString();   // "123.456"
x.toFixed(2);   // "123.46"
```

Zahlen in Strings umwandeln:

```
parseInt("123", 10);        // 123
parseFloat("1.234", 10);    // 1.234
```


Strings

```
var str = "Das ist ein String";
var str2 = "中文 español English हिन्दी বাংলা русский 日本語 ਪੰਜਾਬੀ 한국어"
```

Zugriff auf einzelne Zeichen (Characters)

```
var str = "Das ist ein String";
var x = str[4];   // "i"
```

Funktionen

Funktionen können auf zwei Arten definiert werden.

Mit der Funktions-*Anweisung*:

```
function cube(x) {
    return x * x * x;
}
```

oder als annonymer Funktions-*Ausdruck*, dessen Ergebnis man dann in einer Variable abspeichern kann:

```
var cube = function(x) {
    return x * x * x;
}
```


Funktionen sind Objekte

Man kann sie als Argumente übergeben:

```
// Schreibt hello nach 1000ms
function sayHello() {
    console.log('hello');
}

setTimeout(sayHello, 1000);
```

... oder als von einer Funktion zurückgeben:

```
// Gibt eine Funktion zurück
function makeAdder(initial) {
    // Der Wert von 'initial' geht nicht verloren, wenn makeAdder
    // fertig ist.
    return function(x) {

        return initial + x;
    }
}
var f = makeAdder(5);
console.log(f(2));      // -> 7
```

Neue Objekte erstellen mit Constructor-Funktionen

Normale Funktionen können, wenn mit `new` aufgerufen, als Konstruktoren dienen.

```
function Point(x, y) {
    // 'this' zeigt auf die neue Instanz
    this.x = x;
    this.y = y;
}

// Anwendung mit 'new'
var p = new Point(3, 4);
console.log(p.dist());  // -> 5
```


Vererbung mit Prototypen

Das `prototype` Objekt der Konstruktor-Funktion dient als Vorlage für alle kreierten Objekte.

```
// Methoden definieren über das Prototyp-Objekt
Point.prototype.dist = function () {
    return Math.sqrt(this.x*this.x + this.y*this.y);
};
```

Bad Parts: Globale Variablen

Wenn man das `var` vergisst, kreiert man automatisch eine globale Variable.

```
myname = "Daniel"; // global
window.myname = "Daniel"; // global

// Ausserhalb einer Funktion
var myname = "Daniel"; // global
```


Bad Parts: Man kann alles überschreiben

```
Math.sqrt(9) // 3
Math.sqrt = function (x) {
    return x*x*x;
}
Math.sqrt(9) // 729
```

Nicht empfehlenswert!

Bad Parts: Gleichheit

== und != liefern komische Resultate:

```
'' == 0;           // true
' \t\n' == 0;      // true
null == undefined; // true
true + true;       // 2

"0" == false;      // true
"" == false;       // true
"0" == "";         // false
```

=== und !== brauchen

```
'' === 0;          // false
' \t\n' === 0;     // false
null === undefined; // false
```


JavaScript Zukunft

Alles wird gut mit ES6

Tutorial

http://www.codecademy.com/en/tracks/javascript

Angular template

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>AngularJS Plunker</title>
  <script data-require="angular.js@1.3.x" data-semver="1.3.6"
  src="https://code.angularjs.org/1.3.6/angular.js"></script>
  <script>
  var app = angular.module('itApp', []);
  app.controller('TodoController', TodoController);
  function TodoController() {
    var self = this;
    self.name = "Hans";
  }
  </script>
</head>
<body ng-app="itApp">
  <div ng-controller="TodoController as ctrl">
    <p ng-bind="ctrl.name"></p>

    <p>{{ctrl.name}}</p>
  </div>
</body>
</html>
```