

# Web-Entwicklung 3

TSBE Frühlingssemester 2015

<http://smlz.github.io/tsbe-2015fs/web-t/>

Marco Schmalz

[marco.schmalz@gibb.ch](mailto:marco.schmalz@gibb.ch)



1 / 31

## Kursübersicht

1. HTML und CSS
2. Bootstrap & UI Design, JavaScript Einführung
3. **Prüfung, JavaScript Einführung, AngularJS Browser-Applikation**
4. Python Server Backend, Datenbank-Zugriff
5. Vollständige App (Client-Server Kommunikation)

2 / 31

# Heute

1. Test HTML/CSS
2. Projekt-Organisation
3. JavaScript Einführung
4. Angular

3 / 31

## JavaScript Geschichte

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

4 / 31

# Features

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

5 / 31

## Grösstes Feature

6 / 31

# Grösstes Feature

## Die Absenz von Features

7/31

## 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);
```

8/31

# 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();
```

9 / 31

# 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]);  
}
```

10 / 31

# 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
```

11 / 31

# 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"
```

12 / 31

# 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;  
}
```

13 / 31

## 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
```

14 / 31

# 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
```

15 / 31

# Vererbung mit Prototypen

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

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

16 / 31



# Bad Parts: Globale Variablen

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

```
myname = "Hans"; // global  
window.myname = "Kurt"; // global  
  
// Ausserhalb einer Funktion  
var myname = "Gabi"; // global
```

17 / 31

# Bad Parts: Man kann alles überschreiben

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

Nicht empfehlenswert!

18 / 31

# 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
```

19 / 31

# JavaScript Zukunft

Alles wird gut mit ES6

20 / 31

# Tutorial

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

21 / 31

## Was ist AngularJS

- Ein JavaScript Programmier-Framework, um im Browser dynamische Webseiten zu erstellen.
- Open Source (Von Google unerstützt)
- <https://angularjs.org/>

22 / 31

# Wieso AngularJS

- Standard JavaScript
- DataBinding (Daten werden automatisch aktualisiert)
- Routing (Mehrere Seiten simulieren)
- Leichtgewichtig
- Grosse Community (60'000 Fragen auf StackOverflow)

23 / 31

## Angular: Los gehts!

angular.js in Webseite einbinden:

```
<script
  src="https://code.angularjs.org/1.3.6/angular.js"
  data-require="angular.js@1.3.x"
  data-semver="1.3.6">
</script>
```

Erstes Beispiel (ng-app Attribut zum body-Tag hinzufügen):

```
<body ng-app>
  <h1>{{2+2}}</h1>
</body>
```

24 / 31

# Eigenes Modul schreiben

Jede Angular-App besteht mindestens aus einem Haupt-Modul:

```
// Neues Modul erstellen
var app = angular.module('meineApp', []);

// Neues Modul brauchen um einen Controller zu erstellen
app.controller("MeinController", function() {
  ...
});
```

Neues Modul im HTML angeben:

```
<body ng-app="meineApp">
  ...
</body>
```

25 / 31

# Controller

Der Controller ist das "Bindeglied" von JavaScript zum HTML.

```
// Neues Modul erstellen
var app = angular.module('meineApp', []);

// Neues Modul brauchen um einen Controller zu erstellen
app.controller("MeinController", MeinController);

function MeinController() {}
```

Controller im HTML mit `ng-controller` verwenden:

```
<body ng-app="meineApp">
  <div ng-controller="MeinController as ctrl">
    ...
  </div>
</body>
```

26 / 31

# Datenaustausch

Variablen welche im Controller definiert sind ...

```
function MeinController() {  
  var self = this;  
  self.name = "Marco";  
}
```

... können in der View (HTML) verwendet werden:

```
<div ng-controller="MeinController as ctrl">  
  <p ng-bind="ctrl.name"></p>  
  <p>{{ctrl.name}}</p>  
</div>
```

27 / 31

# Funktionen aufrufen

Funktionen welche im Controller definiert werden ...

```
function MeinController() {  
  var self = this;  
  self.name = "Marco";  
  self.sagHallo = function() {  
    alert('Hallo ' + self.name);  
  };  
}
```

... können vom HTML aus mit `ng-click` aufgerufen werden:

```
<div ng-controller="MeinController as ctrl">  
  <button ng-click="ctrl.sagHallo()">Sag Hallo</button>  
</div>
```

28 / 31

# Datenaustausch 2

Daten können in beide Richtungen ausgetauscht werden.

```
function MeinController() {  
  var self = this;  
  self.name = "Marco";  
  self.sagHallo = function() {  
    alert('Hallo ' + self.name);  
  };  
}
```

In der View (HTML) das Attribut `ng-model` verwenden:

```
<div ng-controller="MeinController as ctrl">  
  <input ng-model="ctrl.name"/>  
  <p>{{ctrl.name}}</p>  
</div>
```

29 / 31

## Beispiel

<http://plnkr.co/edit/S99nTa2iTM0oKEkLerE9?p=info>

30 / 31

# 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('app', []);
    app.controller('MeinController', MeinController);
    function MeinController() {
      var self = this;
      self.name = "Hans";
    }
  </script>
</head>
<body ng-app="app">
  <div ng-controller="MeinController as ctrl">
    <p ng-bind="ctrl.name"></p>
    <p>{{ctrl.name}}</p>
  </div>
</body>
</html>
```