Web Engineering + Design 1

JAVASCRIPT

Michael Gfeller

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Intro

- **JavaScript ist mächtig**
- **JavaScript ist super**
- **JavaScript wird immer besser**
- **JavaScript ist gefährlich**

Bild-Quelle: http://engineering.wix.com/wp-content/uploads/2015/04/mIuuwgx-1024x576.jpg

Lernziele

Die Teilnehmer…

- **… können ihre Kenntnisse von Java auch auf JavaScript anwenden**
- **… kennen die Grundlagen von JavaScript und können diese anwenden**
- **… können die Spezialitäten von JavaScript erklären und nachvollziehen**
- **… kennen den Unterschied zwischen ECMAScript und JavaScript**
- **… kennen die Natur von JavaScript und können die Punkte an Beispiele erklären**
- **… können Array-Funktionen anwenden und kennen die Unterschiede zwischen den verschiedenen Iterationsvarianten**
- **… können Arrow / Lambda-Funktionen anwenden**
- **… können die neuen Features von ECMAScript 6 anwenden**
- **… können JavaScript interpretieren**

Lernziele

Die Teilnehmer…

- **… können (automatische) Typenumwandlungen nachvollziehen und erklären**
- **… können JavaScript Programmieren ☺**

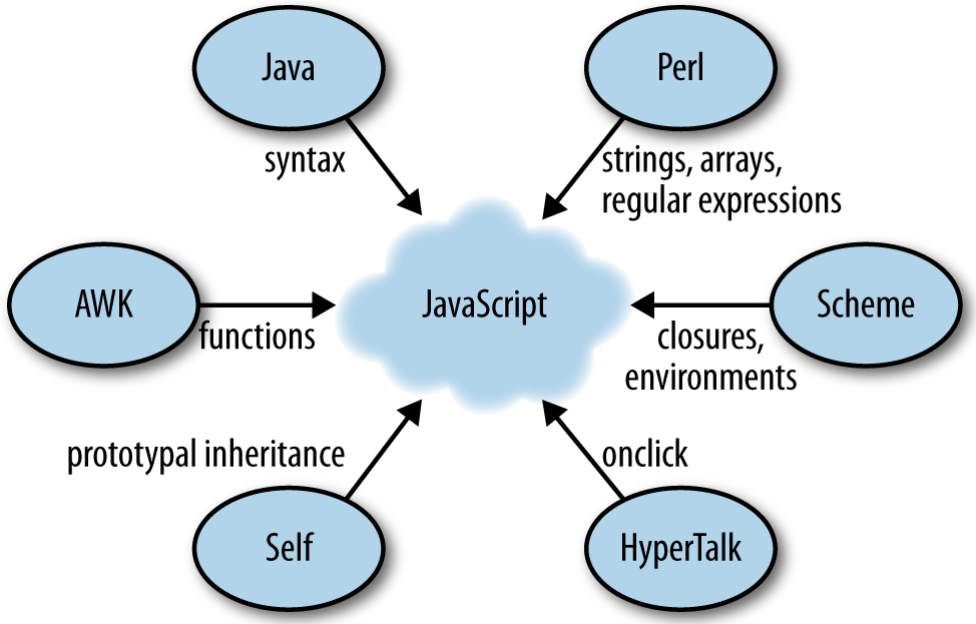
Inhaltsverzeichnis Woche 1

Inhaltsverzeichnis

EINSTIEG

JavaScript Erfolgsgeschichte

1995	JavaScript in 10 Tagen entwickelt
1997	Dynamic HTML: HTML kann geändert werden
1999	XMLHttpRequest: Nachträgliches laden von Daten vom Server
2001	JSON
2005	Google Maps erscheint: Perfektionierte Dynamic HTML und XMLHttpRequest. => AJAX
2006	JQuery
2007	WebKit
2008	V8 – Engine (Chrome)
2009	NodeJs
2009	PhoneGap / ChromeOS



ECMAScript vs. JavaScript

- **ECMAScript ist der offizielle Name für JavaScript**
- **Sun / Oracle haben die Rechte an dem Namen «JavaScript»**

Generell:

- **JavaScript wird für die Programmiersprache verwendet**
- **ECMAScript ist der Name für die Sprachdefinition**
  - z.B. Die aktuelle Version von JavaScript ist ECMAScript 6 und ECMAScript 7 wird entwickelt.

ECMAScript Versionen

Version	publiziert am	Unterschiede zur Vorgängerversion
1	Juni 1997	erste Version
2	Juni 1998	Änderungen zwecks Kompatibilität zum internationalen Standard ISO/IEC 16262
3	Dezember 1999	Neu sind reguläre Ausdrücke, bessere Verarbeitung von Zeichenketten, Kontrollfluss, **Fehlerbehandlung mit try/catch**, bessere Fehlerbehandlung, bessere Formatierung bei der Ausgabe von Zahlen usw.
4	abgebrochen	Wegen Uneinigkeit in Bezug auf die Zukunft der Sprache wurde die weitere Entwicklung des komplexen Entwurfes zu ECMAScript 4 eingestellt. Einige Ideen werden in ES6 wieder aufleben.
5	Dezember 2009	Im „strict mode“ wird eine erweiterte Fehlerprüfung eingeschaltet. Unklare Sprachkonstrukte von ECMAScript 3 werden entschärft und neue Features wie getter- und setter-Methoden, Unterstützung von JSON usw. hinzugefügt.
5.1	Juni 2011	Entspricht dem internationalen Standard ISO/IEC 16262:2011, Version 3
6	Juni 2015	**Neue Syntax für komplexe Applikationen wie Klassen und Module**, die aber mit ähnlicher Terminologie wie in ECMAScript 5 (strict mode) definiert werden können. Neue Sprachbestandteile wie for/of-Schleifen, teilweise an Python angelehnte Syntax usw. Der Codename lautet “Harmony” und wurde bis kurz vor Verabschiedung als „ECMAscript 6“ bezeichnet
7	in Arbeit	Weiterführung der mit ES6 begonnenen Änderungen

Quelle: https://en.wikipedia.org/wiki/ECMAScript#Versions

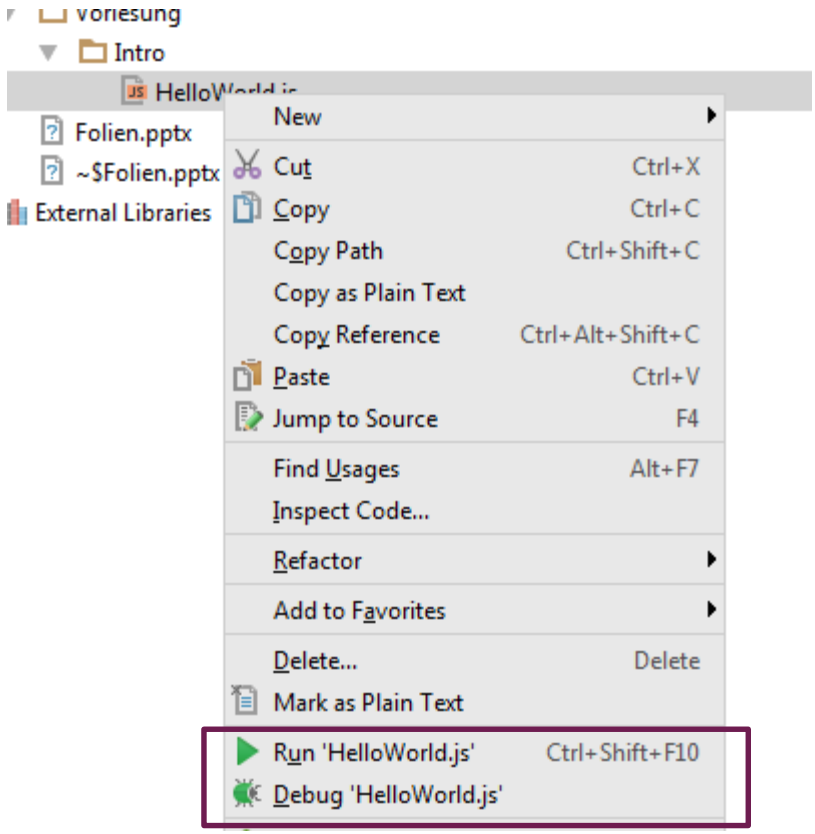
ECMAScript 6

- **Grösster Schritt von JavaScript**
- **Teil der Vorlesung**
- **Browser Support wird besser und besser.**
  - https://kangax.github.io/compat-table/es6/
- **ECMAScript 6 kann schon jetzt verwendet werden mit sogenannten «JavaScript Compiler»**
  - https://babeljs.io/
  - https://github.com/google/traceur-compiler
- **Zusammenfassung der ECMAScript 6 Features mit Beispiele**
  - http://es6-features.org/

JAVASCRIPT AUSFÜHREN

Node.js

- **Nutzt die V8 Engine von Chrome**
- **Kann JavaScript ausführen**
- **Kann JavaScript debuggen**
- **Installation https://nodejs.org/en/**
- **Wird von WebStorm genutzt um JavaScript auszuführen**
- **Kann über die Command-Line angesteuert werden**
- **Folgende Parameter aktivieren ES6 Features**
  - --harmony
  - --harmony_destructuring
  - Aktuelle liste der Features: https://nodejs.org/en/docs/es6/



- **Node.js kann als Web Server fungieren => später in der Vorlesung**

Hello World

helloWorld.js

```
console.log("Hello World");
```

Hello World 2

- **JavaScript ist (in den meisten Fällen) gut lesbar**
- **Was gibt folgender Code aus?**

```
var y = 3;
var x = 3 + y;
var value = add(x, y);

(value > 10) ? console.log("Big") : console.log("Small");

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

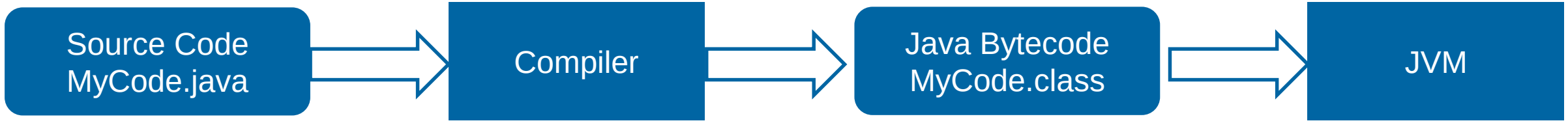
JAVASCRIPT

Die Natur von JavaScript

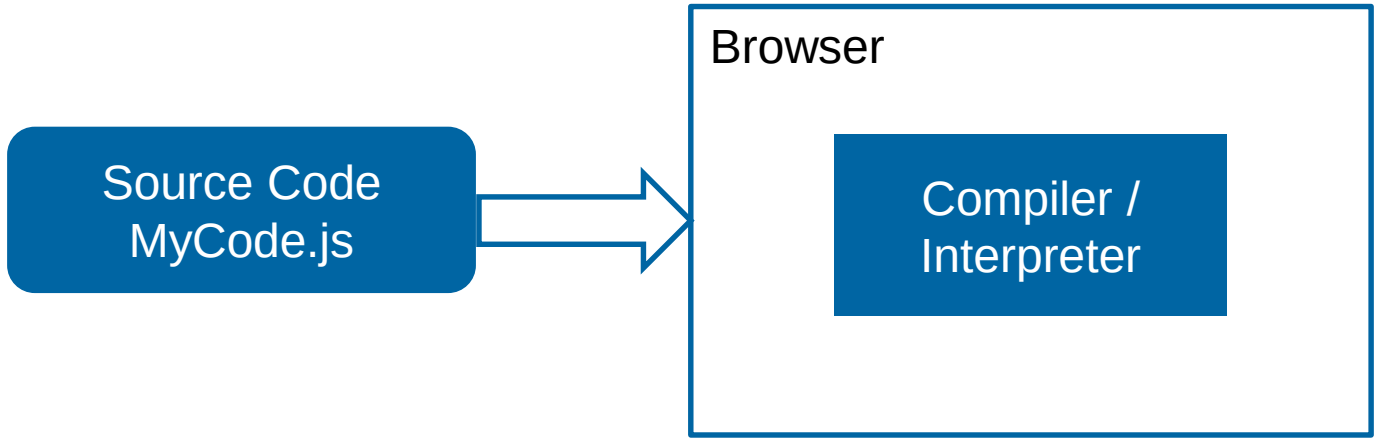
- **It’s dynamic**
  - Objekte können verändert werden z.B. Methoden überschrieben werden
- **It’s dynamically typed**
  - Variablen können den Type ändern – je nach Inhalt
- **It’s functional and object-oriented**
- **It fails silently**
  - Bei Fehler wird oft keine Exception geworfen sondern läuft weiter. z.B. 0/0 = NaN (Not a Number)
- **It’s deployed as source code**
  - JavaScript wird erst beim Ziel (z.B. Browser) interpretiert bzw. kompiliert
- **It’s part of the web platform**
  - Auch ohne Browser lauffähig

It’s deployed as source code

Java Code



JavaScript Code



JAVASCRIPT-TYPEN

It’s dynamically typed

JavaScript is a loosely typed or a dynamic language.

- **Variablen benötigen keine Typendeklarierung**
- **Die gleiche Variable kann über die Zeit unterschiedliche Typen beinhalten**
- **typeof() kann genutzt werden um den Type der Variable abzufragen**
- **Wichtig: Für Nachvollziehbarkeit sollte einer Variable immer nur Werte vom gleichem Typen zu gewissen werden.**

```
var foo;
foo = "Michael";
console.log(foo + " is a " + typeof(foo));
foo = 42;
console.log(foo + " is a " + typeof(foo));
foo = true;
console.log(foo + " is a " + typeof(foo));
```

```
Output:

Michael is a string
42 is a number
true is a boolean
```

Typen-System

Wie Java unterscheidet auch JavaScript zwischen Primitives und Objekten

Primitive Typen:

- **string; number; boolean; null; undefined; symbol (ECMAScript 6)**
- **Compared by value**
- **Always immutable**

Objekte:

- **Alles andere: Plain Objekts, Arrays, Regular Expressions, Functions**
- **Compared by reference**
- **Mutable by default**

Typeof

typeof(type)	Result
Undefined	'undefined'
Null	'object'
Boolean	'boolean'
Number	'number'
String value	'string'
Function	'function'
Symbol (ECMAScript 6)	‘symbol'
All other	'object'

PRIMITIVES

Booleans

- **`true` und `false`**
- **Jeder Wert kann in ein boolean gewandelt werden**
  - !!(null) => false
  - Boolean(null) => false
- **Logische Operatoren**
  - And: &&
  - Or: ||
- **Prefix Opertoren**
  - Not: !
- **Comparison operators**
  - ===, !==, ==, !=
  - >, >=, <, <=

Falsy / Truthy

`false` Werte:

- **false**
- **0 (zero)**
- **"" (empty string)**
- **null**
- **undefined**
- **NaN**

`true` Werte:

- **Alles andere**
  - Z.B. "0" "false" [] {}

```
console.log(Boolean(undefined));   //false
console.log(Boolean(0));           //false
console.log(Boolean(3));           //true
console.log(Boolean({}));          //true
console.log(Boolean([]));          //true
```

Number

- **Nach Definition sind alle Zahlen «floats» (Gleitkommazahl)…**
- **Die Engines versuchen die floats auf integers abzubilden – falls möglich**

```
var x = 1/3;
console.log(0.3333333333333333 * 3 == 1);     //true
console.log(x + x + x);                       //1
console.log(0.3 + 0.7);                       //1
console.log(Number.isInteger(x + x + x));     //true
console.log(Number.isInteger(0.3));           //false
```

  - Funktioniert nicht immer. Grund: Ist ausserhalb vom gültigen Bereich

```
console.log(99999999999999999); //100000000000000000
Number.isSafeInteger(99999999999999999); //false
```

- Definition: http://de.wikipedia.org/wiki/IEEE_754
- Beschreibung: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Number

Number

- **NaN («not a number»)**
  - Ist ein Error-Wert z.B. 0/0 => NaN
  - Hat auch den Type «number»
  - NaN == NaN ist immer false
    - isNaN( ) zum Überprüfen
- **Infinity**
  - Unendlich
  - Kann auch negativ sein

```
var div0 = 0 / 0;
console.log( div0);                //NaN
console.log( typeof(div0));        //number
console.log( parseInt("abc"));     //NaN
console.log( div0 == NaN);         //false
console.log( isNaN(div0));         //true


console.log( 3 / 0);                 //Infinity
console.log( Math.pow(2,10000));     //Infinity
console.log( -Math.pow(2,10000));    //-Infinity
```

Number

- **Jeder Wert kann in eine Zahl verwandelt werden**
  - +(true) => 1 bzw. Number(true) => 1
  - Number(null) => 0
  - Number(«abc») => NaN
  - *Ausnahme: Symbol*
- **parseInt(«string») / parseFloat(«string»)**
  - Parst bis zum ersten Fehler

```
console.log( +(true));                  //1
console.log( +(false));                 //0
console.log( +("1ab"));                 //NaN
console.log( +("123"));                 //123
console.log( +([]));                    //NaN
console.log( parseInt("1.2ab"));        //1
console.log( parseFloat("1.2ab"));      //1.2
console.log( parseInt("abc"));          //NaN
```

String

- **Mit "Text" oder ‘Text’**
- **Escape mit «\»**
- **Typische Properties / Methoden vorhanden**
  - length
  - slice()
  - trim()
  - indexOf()

```
'abc'
"abc"

'Did she say "Hello"?'
"Did she say \"Hello\"?"

'That\'s nice!'
"That's nice!"

'Line 1\nLine 2'  // newline
'Backlash: \\'
```

String – Template (ECMAScript 6)

- **ECMAScript 6 bringt ‘Template strings’**
  - Ermöglicht Strings mit Placeholders (und mehr)
  - https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/template_strings
- **Es gibt einen neuen String-Typ welcher mit `umschlossen` wird**
  - Inhalt innerhalb von ${ … } wird interpretiert und durch das Resultat ersetzt
  - Linebreaks und Leerzeichen werden beibehalten.

```
var name = "Michael";
var hobby = "Hike";

console.log("Mein Name ist: "+name +
            "\nHobby: " + hobby);
```

```
console.log(`Mein Name ist: ${name}
Hobby: ${hobby}`);

//kann auch rechnen
var a = 4;
var b = 5;
console.log(`${a} + ${b} = ${a + b}`);
```

- **Was geben folgende Ausdrücke aus?**

```
"4" / "2"

"4" - "2"

"4" * "2"

"4" + "2"

10 * 3 + "px"

"px" + 1 - 2

1 / 0

"3px" + 3 * 2 + "3px"

"foo"+ +"abc"

"2"- -1
```

+ - * / Regeln

- **Punkt vor Strich**
- **Von Links nach Rechts aufgelöst**

Spezielle:

- **String + Value = String**
- **Value + String = String**

Ansonsten:

- **Value [Nummerische Operator] Value = Number**
  - d.h. + - * / %

The Abstract Equality Comparison Algorithm

```
console.log([] == false);          //true

console.log("" == false);          //true

console.log(null == false);        //false

console.log(0 == "0");             //true

console.log(null == undefined);    //true

console.log([1,2] == "1,2");       //true

console.log(NaN == NaN);           //false

console.log([] == ![]);            //true
```

The Abstract Equality Comparison Algorithm

- **The Abstract Equality Comparison Algorithm (==)**

  Definition:
  http://www.ecma-international.org/ecma-262/5.1/#sec-11.9.3

  Beschreibung:
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Using_the_Equality_Operators

- **The Abstract Relational Comparison Algorithm (<)**

  Definition:
  http://www.ecma-international.org/ecma-262/5.1/#sec-11.8.5

The Abstract Equality Comparison Algorithm

- **===**
  - Verhindert die typen Umwandlung von Primitives
  - Für Objekte nicht notwendig
  - Im Zweifelsfall immer verwenden

```
console.log(false === false);        //true
console.log(4 === 4);                //true
console.log(false === false);        //true
console.log([] === false);           //false
console.log("" === false);           //false
console.log(null === false);         //false
console.log(0 === "0");              //false
console.log(null === undefined);     //false
console.log([1,2] === "1,2");        //false
console.log(NaN === NaN);            //false
console.log([] === ![]);             //false
```

null != undefined

undefined:

- Variable ist nicht definiert. Z.B. `var a` wurde vergessen
- Variable ist nicht initialisiert. Z.B. `a = 1234` wurde vergessen

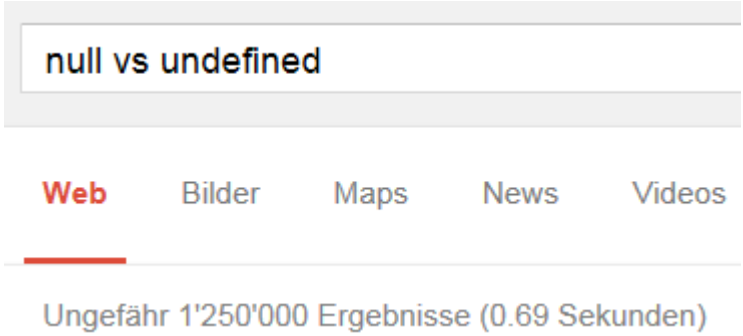
null:

- Ist ein Wert von einer Variable.

=> undefined ist ein Zustand und null ist ein Wert

Achtung: `null == undefined` resultiert in `true`

Hinweis: In alten Browsern ist undefined überschreibar.



null != undefined

Wie würden Sie überprüfen ob eine Variable undefined ist?
Z.B. myVariable

Wie würden Sie überprüfen ob ein Wert auf einem Objekt undefined ist?

z.B. myVariable.a

null != undefined

Wie würden Sie überprüfen ob eine Variable undefined ist?
Z.B. myVariable

Wie würden Sie überprüfen ob ein Wert auf einem Objekt undefined ist?

z.B. myVariable.a

```
A:
typeof myVariable == 'undefined';

B:
myVariable.a == undefined;

Combined:
typeof (myVariable) != 'undefined' && myVariable.a == undefined;

Performance:  http://jsperf.com/undefined-null-typeof
```

ARRAY

Array

```
var arr = [ 'a', 'b', 'c' ];
arr[0] = 'x';
arr.push("d");
console.log(arr);
console.log(arr.length);
```

- **Wichtige Methoden:**
  https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Array#Methods_of_array_instances

Array Methoden & Iterieren

- **Liste aller Varianten:**
  https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Array#Iteration_methods

```
var arr = [ 'a', 'b', 'c' ];
arr.forEach(function(elem, index){
    console.log(index +":"+elem);
});

var array = [1,2,3,4].map(function (x) { return x*x });
console.log(array);

var filteredArray = array.filter(function(elem){
    return elem > 5;
});
console.log(filteredArray);

console.log(array.every( function(elem){ return elem > 5}));
```

Array Iterieren for of/in

- **Normale**

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

- **For-In iteriert über die Property Namen**

```
for(var x in arr)
{
    console.log("for in", x + ":" + arr[x]);
}
```

```
var dummy = {name: "Hallo", date : Date.now()};

for(var x in dummy)
{
    console.log("dummy has property", x);
}
```

- **For-Of iteriert über die Werte**

```
for(let y of arr)
{
    console.log("for of", y);
}
```

OBJECT

Simple Object

- Ein Object ist eine Sammlung von Properties.
- Die Properties werden mit einem Set (HashSet) verwaltet
  - Key = String
  - Value= Value
    - boolean / function / string / …
- Objekt können als «object literals» erstellt werden und/oder nachträglich mit Properties ergänzt werden:

```
var person = {
    name : "Michael",
    hallo : function(){
        return "Hallo "+this.name;
    }
};

console.log(person.hallo());

person.name = "Bob";
console.log(person.hallo());
```

```
var mySimpleObject = {};
mySimpleObject.name = "Michael";
mySimpleObject.hallo =  function(){
    return "Hallo "+this.name;
};

console.log(mySimpleObject.hallo());
```

Simple Object: It’s dynamic

- Properties und Methoden können hinzugefügt / verändert werden:

```
var person = {
    name : "Michael",
    hallo : function(){
        return "Hallo "+this.name;
    }
};
person.hobby = "Hike";
person.hallo = function(){
    return "Hallo " + this.name + " Hobby: "+ this.hobby;
};
console.log(person.hallo());
```

- Auch von «Standard»-Objekten. *Wichtig: Standardfunktionalität sollte nie verändert werden.*

```
console.log("X");
console.log = function(value) {

};
console.log("X");
```

FUNCTIONS

JavaScript Funktionen

- Funktionen können in Variablen abgespeichert werden.
- Funktionen können als Parameter übergeben werden.
- Funktionen besitzen eine offene Parameter-Liste
  - Es können mehr oder weniger als die deklarierte Anzahl an Parameter übergeben werden.
  - Alle Parameter werden in `arguments` abgelegt.
- Funktionen besitzen Properties
- Eine Funktion erzeugt einen eigenen Scope.

```
function helloWorld(a){
    console.log(a || "No Data");
}

function helloWorld2(){
    console.log(arguments[0]);
}

var sayHello = function(fnOutput)
{
    fnOutput("Hallo")
}

sayHello(helloWorld);
sayHello(helloWorld2);
```

Funktionen definieren

```
//Funktionen können definiert werden
function hallo(){
    console.log("Hallo");
}
hallo();

//Funktionen können einer Variable zugewiesen werden
var hallo2 = function(){
    console.log("Hallo2");
};
hallo2();

//Funktionen können einer Variable zugewiesen werden
var foo = hallo;
foo();
```