



# ***Web-basierte Anwendungen***

*Studiengänge AI (4140) & WI (4120)*



# JavaScript

## Eine kurze Einführung

(mit Material von Prof. G. Behrens und M. von Rohden)

## **1. Skriptsprachen**

- 2. Einführung in JavaScript
- 3. Technisches Umfeld
- 4. Grundlagen (Auswahl)
- 5. Objektmodell
- 6. Zugriff auf HTML-Dokumente
- 7. Event-Handler



# Wozu Skriptsprachen?

---

- Häufig wiederkehrende Aufgaben
- Abstraktionen dieser Aufgaben in Skriptsprachen
  - Sprachmittel auf höherer Ebene (z.B. Schleifen = für alle)
  - z.T. Erreichen der Abstraktion natürlicher menschlicher Sprache
  - mächtige Datenstrukturen als Teil der Sprache (z.B. Listen)
  - umfangreiche integrierte Bibliotheken
- Ziele
  - Geringer Einarbeitungsaufwand
  - Schnelle Implementierung
  - Leicht lesbarer und verständlicher Code
  - Fehlerfreiheit
  - Einfache Wartbarkeit

# Eigenschaften von Skriptsprachen I

---

- Automatische Speicherverwaltung
- Mächtige Datenstrukturen
  - String-/ bzw. Zeichenkettenverarbeitung
  - statische und dynamische Listen
  - Tupel
  - Wörterbücher, assoziative Arrays
- z.T. Objektorientierung (Ruby, Python, JavaScript; PHP, Perl)
- z.T. Typsysteme und Modulkonzepte

# Eigenschaften von Scriptsprachen II

---

- Umfangreiche Bibliotheken und spezifische Features
  - Web-Anwendungen  
(HTML-Integration, HTTP, Session-Verwaltung, Template-Bearbeitung, ...)
  - Benutzeroberflächen (grafische Elemente und Interaktivität)
  - Allgemein  
(reguläre Ausdrücke, mächtige eingebaute Klassen bzw. Datenstrukturen, ...)
  - Kombination bekannter Bausteine  
(Kontrolle von Ein- und Ausgabe, Prozessverwaltung)

# Beispiele für Scriptsprachen

---

- Kommandozeileninterpreter
  - /bin/sh Bourne Shell
- Sprachen/Tools zur String-Manipulation
  - sed, awk
- Vollwertige Programmiersprachen
  - **Perl** (aus Awk mit Focus auf String-Manipulation)
  - **Python** (Übernahme von allen allgemeinen Eigenschaften aus Vorgängern Sh, Sed, Awk, Perl)
  - **Ruby** (Anleihen bei Python, Perl, Smalltalk, CLU, Lisp, ...)
  - Visual Basic, VBScript, JScript, ...
- Spezialisierung auf bestimmte Anwendungen
  - PHP für serverseitige Web-Applikationen
  - JavaScript für clientseitige Web-Applikationen



# Skriptsprachen im Vergleich mit anderen Programmiersprachen

---

	Skriptsprachen	andere Sprachen
Einsatzziel		
Einsatzbereich		
Ausführung		
Typisierung		
Stärke		





# Skriptsprachen im Vergleich mit anderen Programmiersprachen

---

	Skriptsprachen	andere Sprachen
<b>Einsatzziel</b>	Kombination bekannter Bausteine und Bibliotheken	Neuentwicklungen (z.B. C)
<b>Einsatzbereich</b>	meist speziell (z.B. GUI mit JavaScript, Reporting mit Perl)	meist allgemein (Java, C++, C)
<b>Ausführung</b>	Interpreter oder Binärer Code (z.B. Python)	meist in Maschinencode compiliert (Java, C++, C)
<b>Typisierung</b>	flexibles Typkonzept (z.B. Python, Ruby)	streng typisiert (Java, C++, C)
<b>Stärke</b>	Schnelle, unkomplizierte Entwicklung (Python, PHP, Ruby, JavaScript)	Effiziente Ausnutzung der System-Ressourcen

1. Skriptsprachen
- 2. Einführung in JavaScript**
3. Technisches Umfeld
4. Grundlagen (Auswahl)
5. Objektmodell
6. Zugriff auf HTML-Dokumente
7. Event-Handler

# Einführung in JavaScript

---

## Entstehung:

- 1995 bei Firma Netscape (damaliger Name: *Mocha*, *LiveScript*)
- Bezeichnung **JavaScript** seit Impl. in Netscape Navigator 2

## Entwicklungsziele:

- Erweiterung statischer Webseiten mit dynamischen Inhalten
- Attraktivere Gestaltungsmöglichkeiten
- Verbesserte Funktionalität

## Beispielhafte Anwendungsfälle:

- Überprüfen von Dateneingaben auf Korrektheit beim Client
- Kleine eigenständige Clientanwendungen (Taschenrechner, Währungsrechner, ...)
- Clientseitige Scripte für dynamisch eingeblendete Schaltflächen, Grafiken, Spiele, ...



# Einführung in JavaScript

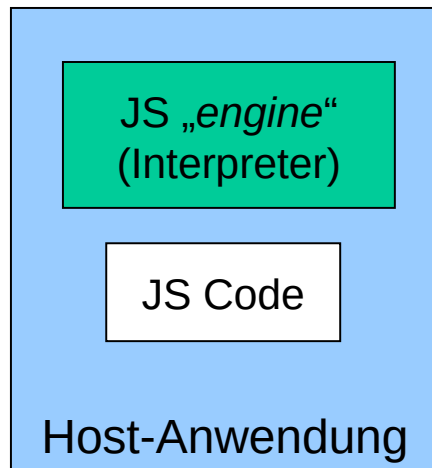
---

## Eigenschaften:

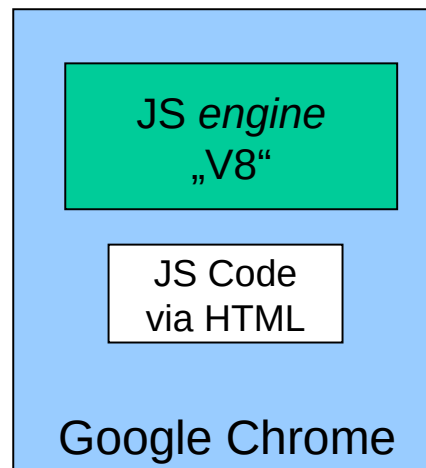
- Vollwertige Programmiersprache, „eingebettet“
- Objektorientierte Basis
- Interpreter-Sprache
- (Angeblich) Einfach zu erlernen
- Überwiegend clientseitige Nutzung im Browser
- Serverseitige Nutzung möglich mit *JScript* ; aber sehr selten (eher verbreitet: PHP, Perl, ASP - Active Server Pages, JSP – Java Server Pages).  
Rel. neu: Node.js
- Plattformübergreifend (versch. Browser, versch. Betriebssysteme)
- Standardisierungsprobleme bei Browsern: z.T. laufen JavaScript-Programme nicht oder nicht richtig
- Greift auf HTML-Code zu und erzeugt HTML-Elemente
- Wird in HTML-Code eingebunden über Element **<script>... </script>**:

```
<script type = "text/JavaScript">  
    <!-- JavaScript - Anweisungen -->  
</script>
```

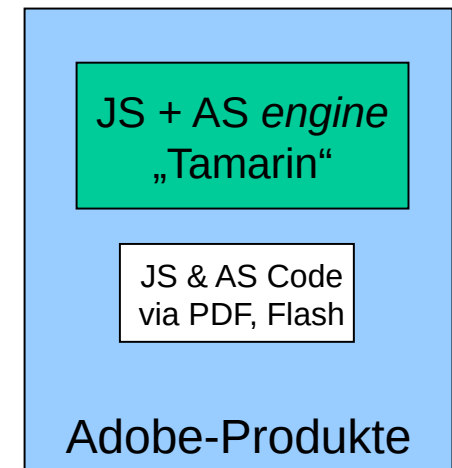
# \* Einführung in JavaScript



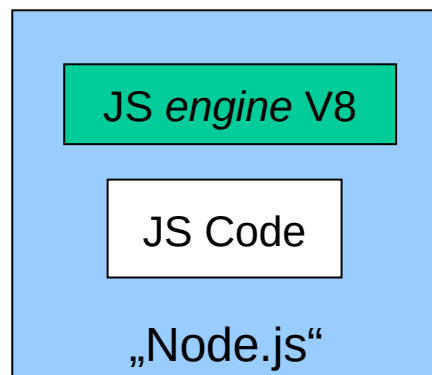
Einbettung: Prinzip



Beispiel Browser



Flash-Plugin, Acrobat, ...



(Auch) Für ereignisgesteuerte „standalone-Anwendungen“

## Aktuelle JS engines, mit JIT-Techniken

- Caracan
  - Chakra
  - SpiderMonkey
  - JavaScriptCore
  - Tamarin
  - **V8**
  - ...
- Opera  
MS IE, Edge V.1  
Mozilla  
WebKit (Apple)  
Adobe  
**Google**

# Einführung in JavaScript

---

## Konsequenz aus der Einbettung:

- Je nach einbettender Umgebung stehen unterschiedliche Objekte & Methoden zur Verfügung.

Die Kommunikation mit der Umgebung unterscheidet sich bereits bei der Ausgabe einer Textzeile!

```
document.alert("Hallo, Welt!");
```

Browser-Kontext  
Erzeugt ein Popup-Fenster

```
document.write("Hallo, Welt!<br/>");
```

Browser-Kontext  
Ergänzt d. HTML-Dokument

```
console.log("Hallo, Welt!");
```

Node.js-Kontext  
Gibt Text auf Kommandozeile aus

```
$ node hello.js  
Hallo, Welt!
```

# \* Einführung in JavaScript

---

## Einbettung in (X)HTML: Das Element „script“

- JS-Quellcode wird immer als Nutzdaten-Inhalt eines Elements **script** „verpackt“ oder über dieses Element aus externen Quellen importiert
- **script** lässt sich an vielen Stellen in (X)HTML einbetten, insb. als Unter-Element von **head** und **body**. Es darf mehrfach vorkommen.
- Eine Kombination von JS- und HTML-Kommentaren verhindert Missverständnisse

```
<!-- Externe JS-Bibliotheksdatei einbinden -->
<script type="text/javascript" src="/assets/jquery_ujs.js"/>
<!-- JS-Code direkt in HTML-Seite einbetten -->
<script type="text/javascript">
  <!-- Beginn eines HTML-Kommentars: JS-Code wird vom HTML-Parser ignoriert
    document.write("Hallo, Welt!<br/>");
  // -->
</script>
  JS-Kommentar, damit der JS-Interpreter "-->" ignoriert
```

# Einführung in JavaScript

---

## Alternativen zu JavaScript?

- MS Silverlight? – Proprietär, nutzt intern JS (neben anderen Skriptsprachen), Plattformunabhängigkeit fraglich (→ „Moonlight“-Projekt)
- Adobe Flash/ActionScript? – Proprietär, z.B. von Apple nicht unterstützt

## CoffeeScript – das „bessere“ JavaScript?

- Die JS-Infrastruktur wird beibehalten
- Das Objektmodell von JS wird als bewährt betrachtet und übernommen
- Die Erfahrungen der letzten 10..15 Jahre Skriptsprachen-Entwicklung (Inspirationen von Ruby, Python u.a.) finden Ausdruck in einer neuen Sprache, die aber direkt in JS-Code übersetzt wird
- Ziel: Höhere Produktivität bei der JS-Entwicklung
- Rails integrierte CoffeeScript von Version 3 bis 5, nun optional installierbar
- Alternativen? Siehe z.B. <http://altjs.org/>
- Unser Weg: Annäherung an CS durch Beispiele, im Vergleich zu JS





# Einführung in JavaScript

---

- **Aktuelle JavaScript-Alternativen**

- Microsoft: **TypeScript**
  - Voll abwärtskompatibel zu JavaScript
  - Mit optionaler statischer Typisierung
  - Sehr gute Integration in MS-Tools wie VisualStudio
- Google: **Dart**
  - Java-artige Syntax, ohne JS-“Altlasten“, Fokus auf Geschwindigkeit
  - Vom inzwischen dominierenden Browser Chrome unterstützt
- OSS-Lager: **CoffeeScript**
  - Fördert die guten Aspekte und versteckt die schlechten von JavaScript
  - OSS: Ohne Firmen-“Bias“
  - Fokus auf Code-Kompaktheit und bewährte Skriptsprachen-Konstrukte
- W3C: **WebAssembly (WASM)**
  - Effizientes, schnelles Binärformat für Client- und Server-Entwicklung im Web
  - Kann von diversen (statisch typisierten) Sprachen aus kompiliert werden
  - Wird von den wichtigsten 4 Browsern unterstützt

- Einige vergleichende Artikel zu diesen Alternativen:

- <http://codeforhire.com/2013/06/18/coffeescript-vs-typescript-vs-dart/>
- <http://www.drdobbs.com/jvm/the-javascript-alternatives/240166433>
- <http://www.hanselman.com/blog/WhyDoesTypeScriptHaveToBeTheAnswerToAnything.aspx>

# Aktuell: 20 Jahre JavaScript; ECMAScript 6

---

- Ein Artikel zur Entstehung von JavaScript mit einem Blick in die zukünftige Entwicklung:
  - <http://www.heise.de/ix/heft/Schubladendenken-2822765.html>
- Weitere Entwicklung:
  - JavaScript 2 / ECMAScript 2015 (**ES6**, 17.6.2015; ISO/IEC 16262:2016)
    - Viele Konzepte von CoffeeScript wurden übernommen, z.B. Klassen, Iteratoren, Scoping von Variablen, Arrow-Funktionen, Maps,
    - Von Ruby stammende Konzepte: Symbole, Module, Generatoren
    - Weitere, eigene Konzepte: z.B. Promises (für asynchrone Verarbeitung)
    - Browser-Unterstützung von ES6: Schon weit fortgeschritten
    - CoffeeScript wandelt seit V 2.0 in ES6-Code um!
  - Aktuell: ES11 (ECMAScript 2020, Juni 2020)
    - Viele Fortschritte seit dem Durchbrechen des Stillstands durch ES6, u.a.:
    - **\*\***-Op., await, async; Object.values etc., **...**-Op. (spread-Op.), besseres Array.sort, BigInt, **??**-Op., chaining...

**Unser Fokus** hier: **JavaScript 1.x**, also meistens ohne die neuen ES-Features



# JavaScript: Die Bestandteile

---

- In der **Sprache** selbst eingebaute Objekte
  - Array, Boolean, Date, Math, Number, Object, RegExp, String
- In der **Umgebung** eingebaute Objekte
  - Browser: Window, Navigator, Screen, History, Location
  - Node: console, process, sys, util, ...
- **Core DOM-Objekte** (für alle XML- und HTML-Daten)
  - Node, NodeList, NamedNodeList, Document, Element, Attr
- **HTML DOM-Objekte**
  - (eine lange Liste...)



1. Skriptsprachen
2. Einführung in JavaScript
- 3. Technisches Umfeld**
4. Grundlagen (Auswahl)
5. Objektmodell
6. Zugriff auf HTML-Dokumente
7. Event-Handler

# Technisches Umfeld zu JavaScript

---

1. Sicherheit
2. Im Browser aktivieren
3. Debuggen von Funktionen

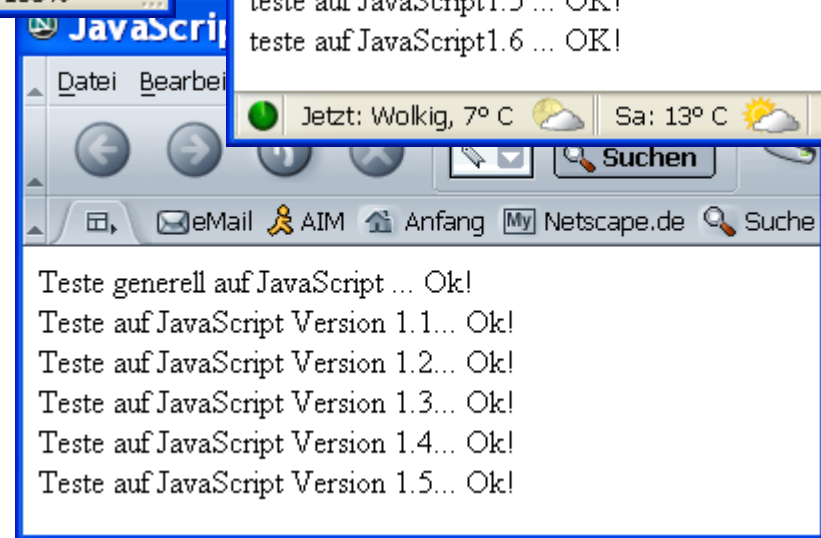
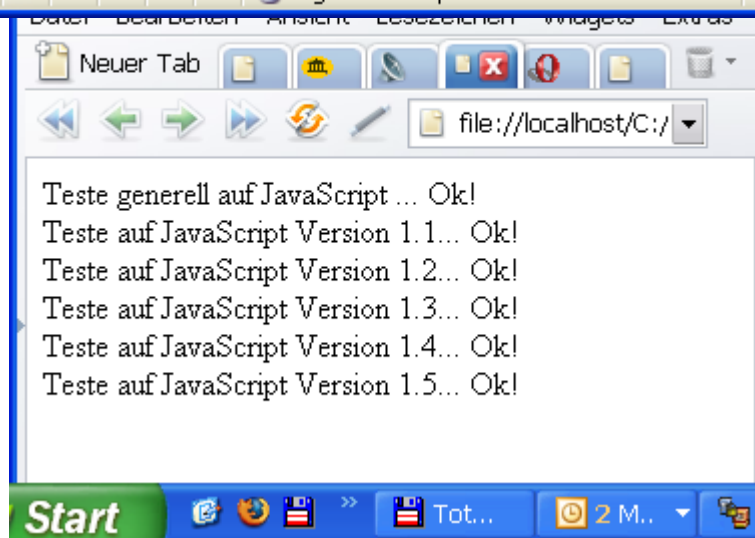
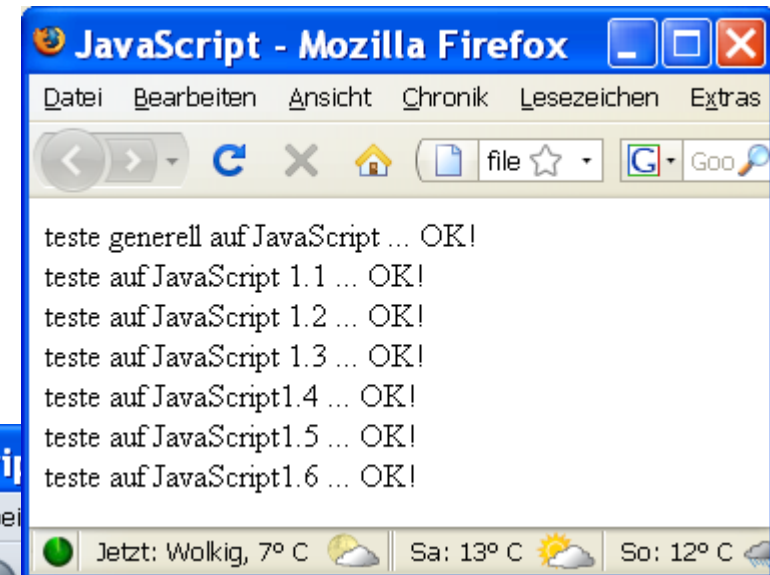
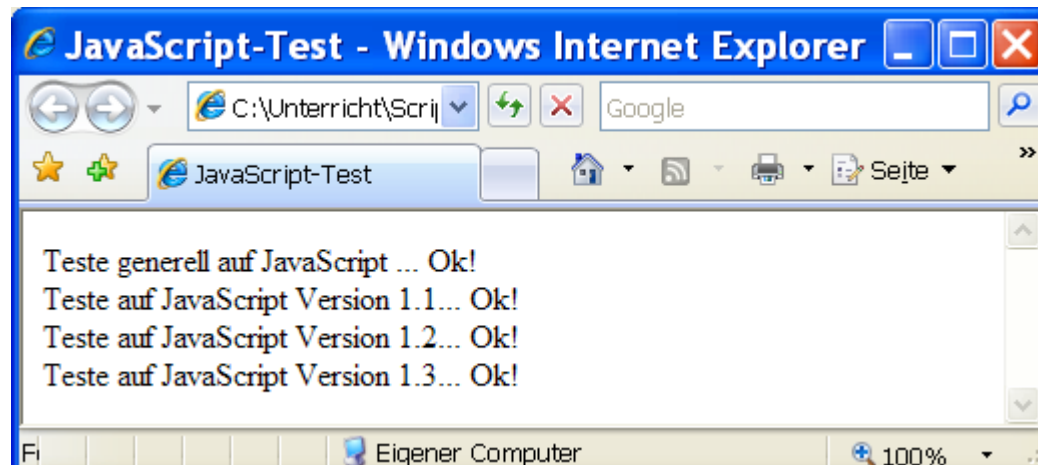
# JavaScript und Sicherheit

---

- Sicherheitslücken in Browserimplementierungen können durch JavaScript-Programme ausgenutzt werden z.B.:
  - unbemerktes Versenden von E-Mails
  - Auslesen des Browserverlaufs
  - Live-Verfolgungen von Internetsitzungen
  - Erraten von eBay-Passwörtern
- Anwender deaktivieren daher manchmal das „Ausführen von JavaScript-Code“ im Browser
- JavaScript-Anwendungen laufen im Browser: *Sandbox* (abgeriegelte Umgebung ohne Zugriff auf Dateien, Benutzerdaten, BS, ...)

# \* JavaScript: Versionen in verschiedenen Browsern

- MSIE 7.0 JavaScript Version 1.3
- Mozilla(5.0) FireFox 3.0.3 JavaScript Version 1.6
- Netscape 7.1 JavaScript Version 1.5
- Opera 9.2.4 JavaScript Version 1.5



# \* Versionen von JavaScript

---

```
<html>
<head>
  <title>JavaScript-Test</title>
</head>
<body>
  <script type="text/javascript">
    <!--
      document.write("Teste generell
auf JavaScript ... Ok!<br>")
    //-->
  </script>
  <noscript>
    Ihr Browser versteht kein
    JavaScript. Es kann nicht
    ausgeführt werden.
  </noscript>
  <script language="JavaScript1.1">
    <!--
      document.write("Teste auf
JavaScript Version 1.1... Ok!<br>")
    //-->
  </script>
```

```
    <script language="JavaScript1.2">
      <!--
        document.write("Teste auf
JavaScript Version 1.2... Ok!
<br>")
      //-->
    </script>
    <script language="JavaScript1.3">
      <!--
        document.write("Teste auf
JavaScript Version 1.3... Ok!
<br>")
      //-->
    </script>
    <script
language="JavaScript1.4"><!--
      document.write("Teste auf
JavaScript Version 1.4... Ok!
<br>")
    //--></script>
    <script language="JavaScript1.5">
      <!--
        document.write("Teste auf
JavaScript Version 1.5... Ok!
<br>")
      //-->
    </script>
  </body>
</html>
```



# JavaScript im Firefox-Browser aktivieren

---

- Einstellung mit Bordmitteln
  - URL: „about:config“
  - Sicherheitsabfrage bestätigen
  - Stichwort „javascript“ suchen
  - Eigenschaft „javascript.enabled“ auf true oder false stellen
- Hilfsmittel
  - FF-Addon „noscript“
  - FF-Addon „ghostery“



# Beispiel eines Webshops mit Überprüfung von JavaScript

ESPRIT Online-Shop Deutschland - Kleidung versandkostenfrei bestellen! - Mozilla Firefox


Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

http://www.esprit.de/

Google

## ESPRIT

esprit.de

**Bitte aktivieren Sie JavaScript**  
Die Nutzung der Website [www.esprit.de](http://www.esprit.de) ist ohne JavaScript nicht in vollem Umfang möglich.

**Bitte ändern Sie die Einstellungen in Ihrem Browser wie folgt:**

**Internet Explorer**  
Wählen Sie: Extras > Internetoptionen > Sicherheit > Internet > Standardstufe.


**Firefox**  
Wählen Sie: Extras > Einstellungen > Web-Features > "Java aktivieren" und "Java Script aktivieren" jeweils ein Häkchen setzen.

**Mozilla**  
Wählen Sie unter Bearbeiten > Einstellungen > Erweitert > Scripts und Plugins > JavaScript aktivieren.

**Netscape**  
Wählen Sie: Bearbeiten > Einstellungen > Erweitert > JavaScript aktivieren.

**Opera**  
Wählen Sie: Datei > Einstellungen > Multimedia > JavaScript aktivieren.

**Safari**  
Wählen Sie: Safari > Einstellungen > Sicherheit > JavaScript aktivieren.



**Sie haben Ihre Einstellungen geändert?**  
Hier geht's zum e-shop: [women](#) [men](#) [kids](#)  
Hier geht's zum Esprit Katalog: [Esprit Katalog](#)

Fertig

Jetzt: Wolkig, 4° C Mi: 7° C Do: 3° C Fr: 5° C

Quelle:  
<http://www.esprit.de>

# \* Debuggen von JavaScript - Funktionen (MF)

**Demo: Firebug (F12)**

**Extras-> Fehlerkonsole**

The screenshot shows the Mozilla Firefox browser window with the Firebug extension installed. The 'Extras' menu is open, and a red arrow points to the 'Fehlerkonsole' (Error Console) option. Below the browser window, the 'Fehlerkonsole' (Error Console) window is open, displaying a list of errors. The first error is a warning icon followed by the text: 'Deklaration erwartet, aber '\*' gefunden. Übersprungen bis zur nächsten Deklaration' (Declaration expected, but '\*' found. Skipped to the next declaration). The error is linked to the file [http://www.hrs.de/css/styles\\_homepage.css](http://www.hrs.de/css/styles_homepage.css) at line 144. The second error is partially visible and follows the same pattern.

**Fehlerkonsole**

Alle Fehler Warnungen Mitteilungen Löschen

Code:  Evaluieren

Deklaration erwartet, aber '\*' gefunden. Übersprungen bis zur nächsten Deklaration  
[http://www.hrs.de/css/styles\\_homepage.css](http://www.hrs.de/css/styles_homepage.css) Zeile: 144

Deklaration erwartet, aber '\*' gefunden. Übersprungen bis zur nächsten Deklaration

1. Skriptsprachen
2. Einführung in JavaScript
3. Technisches Umfeld
- 4. Grundlagen (Auswahl)**
5. Objektmodell
6. Zugriff auf HTML-Dokumente
7. Event-Handler

# Ausgewählte Grundlagen von JavaScript

---

- Notationen
- Variablennamen
- Datentypen
- Operatoren
- Programmsteuerung
- Funktionen
- Objekte

## CoffeeScript

Eingeschobene Kästen  
dieser Art zeigen das  
aktuelle JS-Feature in  
seiner CoffeeScript-  
Umsetzung

Lese-Aufgabe (JS Tutorial):

<https://w3schools/js/default.asp>JavaScript

Ergänzend: Komplettes Tutorial unter

<https://wiki.selfhtml.org/wiki/JavaScript>



# Notationen

---

Alle Anweisungen in JavaScript müssen mit einem Semikolon “;” enden.

## CoffeeScript

Abschließendes ; ist optional

## Kommentare:

// bezeichnet einen einzeiligen Kommentar

/\* \*/ bezeichnet einen mehrzeiligen Kommentar

## CoffeeScript

# bezeichnet einen einzeiligen Kommentar

###

bezeichnet einen  
mehrzeiligen Kommentar

###

# \* Variablennamen

---

## Regeln für Variablennamen in JavaScript:

- Beginn mit Buchstabe oder Unterstrich (Achtung: *case sensitive*)
- Rest kann Buchstabe, Ziffer oder das Sonderzeichen „\_“ (*underscore*) sein (keine Leerzeichen, keine anderen Sonderzeichen)
- Maximallänge 32 Zeichen
- Es gibt reservierte Wörter wie : **var**, **case**, **for**, ...

## Zum Beispiel:

_2_Test_Wert	<i>gültig</i>	
2_Test_Wert	<i>ungültig</i>	Zahl
Email_Adresse	<i>gültig</i>	
@_Adresse	<i>ungültig</i>	Sonderzeichen
Langer_Variablenname	<i>gültig</i>	
Das_ist_noch_laengerer_Variablenname	<i>ungültig</i>	zu lang
Test 1	<i>ungültig</i>	Leerzeichen
Test_1	<i>gültig</i>	

# \* Wertzuweisungen an Variablen

---

Variablen können durch **var** deklariert werden und einen Anfangswert zugewiesen bekommen.

Der Anfangswert legt den Datentyp fest.

## Beispiel:

```
var beispiel = 1;  
var beispiel2 = 8;  
var zusammen = beispiel+beispiel2;
```

## CoffeeScript

```
beispiel = 1  
beispiel2 = 8  
zusammen = beispiel+beispiel2
```

JavaScript lässt es zu, dass Programmierer **eine Variable überhaupt nicht deklarieren**. Achtung: Fehler nach **use strict**;

Durch **dynamische Typisierung** erhält die Variable während der Programmausführung vom Interpreter einen Datentyp.

Achtung: Unterschiedliche Realisierung in Browsern möglich!

Der **typeof**-Operator liefert den aktuellen Typ: string, number oder boolean.

```
var Zahl = 5;  
Typ = typeof Zahl; //number
```

```
var Zeichen = "Hallo";  
Typ = typeof Zeichen; //string
```

```
var istWahr = true;  
Typ = typeof istWahr;  
//boolean
```





# Datentypen

---

## Ganzzahlen

- *besitzen keine Nachkommastellen*
  - *Dezimaldarstellung z.B.: 42*
  - *Oktaldarstellung mit vorangestellter Null „0“ z.B.: 052*
  - *Hexadezimaldarstellung mit vorangestelltem „0x“ z.B.: 0x2A*

## Gleitkommazahlen

- *es wird ein Punkt und kein Komma verwendet*
  - *z.B. 1E3 (1000.0) , 1.0E3 (1000.0) , 10.0e2 (1000.0), 1004.0E-1 (100.4)*

## Strings

- *Kette von Zeichen, die in Anführungszeichen eingeschlossen ist " " oder ' '*
  - *z.B. "Zeichenkette", "Er sagte:'Das kann doch nicht wahr sein!', und ging."*

## Boolesche Werte

- ***true** und **false** (meist als Ergebnis von Vergleichen)*



# Datentypen: Beispiel

```
<html>
<head>
  <Title>Datentypen</Title>
</head>

<body>
  <script language="JavaScript">
    <!--
      var a = 4.2e1
      document.write(a, "<br>");

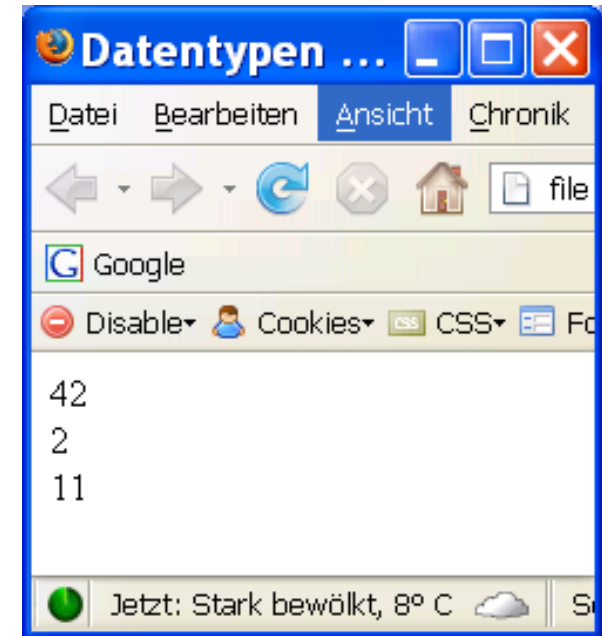
      var nummer = 1;
      var zeichen = '1';

      var ergebnis = nummer+nummer;
      document.write(ergebnis, "<br>");

      var ergebnis = zeichen+zeichen;
      document.write(ergebnis);

    // -->
  </script>
</body>
</html>
```

*Welche Ausgabe generiert dieses Beispielprogramm?*



# \* Typen

---

- JavaScript kennt nur wenige eingebaute Typen:
  - Zahlen
  - Strings
  - Boolesche Werte
  - Funktionen
  - Objekte
  - Nicht definierte Dinge
- Die eingebaute Funktion „typeof“ nennt den Typ als String:

<code>typeof(1);</code>	→ "number"
<code>typeof(1.5e-3);</code>	→ "number"
<code>typeof("abc");</code>	→ "string"
<code>typeof(false);</code>	→ "boolean"
<code>typeof(Math.sin);</code>	→ "function"
<code>typeof({a: 'b'});</code>	→ "object"
<code>typeof([1,2]);</code>	→ "object"
<code>typeof(/d/);</code>	→ "object"
<code>typeof(nosuchvar);</code>	→ "undefined"

# \* Typen

---

- Beispiel-Code (für die Umgebung von node.js)

```
var null_fun = function() {return 0};
var type_samples =
  [1, 1.5e-3, "abc", false, null_fun, Math.sin,
   {prop: 'value'}, [1,2], /\d/]

for (var i=0; i < type_samples.length; i++) {
  console.log("Type of " + type_samples[i] + " is: '" +
    typeof(type_samples[i])+"'");
}
console.log("Type of a not yet defined variable is: '" +
  typeof(nosuchvar)+"'");
```

- Erste Beobachtungen
  - C-artige Syntax, einschließlich for-Schleife
  - Literale Notationen für Zahlen, Strings, Hashes (?), Arrays, Reguläre Ausdrücke
  - Nutzung von Arrays wirkt vertraut
  - String-Konkatenation per Infix-Operator „+“; „to\_s“ implizit wirksam

# \* Typen

---

- Erste Überraschungen
  - Anders als in Ruby gibt es „Typen“
  - Der Begriff „Datentyp“ passt nicht, denn er erfasst auch „function“
  - Funktionen spielen offenbar eine herausragende Rolle
  - Anders als in Java wird nicht zwischen verschiedenen Zahlentypen unterschieden (int, long, float, double ...) – alles ist „number“
  - Es scheint die gewohnten literalen Notationen für Hashes, Arrays und Reguläre Ausdrücke zu geben, aber die so erzeugten Dinge bezeichnet JavaScript alle pauschal als „object“
- Was also versteht JavaScript unter einem „Objekt“?
  - Wir kommen darauf ein paar Folien später zurück ...

# Operatoren

---

Operatoren	Beispiele	Datentyp
Vergleichsoperatoren	==, !=, <>, <, >, <=, >=, ===, !== Ergebnis: boolescher Wert	Zahlen, Strings, Objekte (== ...)
Berechnungsoperatoren	+, -, *, /, % (Modulo), ++ (Inkrement), -- (Dekrement)	Zahlen
Konkatenationsoperator	'1'+'1' → '11'	Strings
Logische Operatoren	&& - UND,    - ODER, ! - NICHT	Boolesche Werte
Bit-Operatoren	& - (1010&0110) → 0010  - ODER, ~ - NICHT, << Linksverschiebung	Zahlen, boolesche Werte
Zuweisungsoperatoren	a=a+5; a+=5; // beide gleich	alle



# Programmsteuerung

---

*Bedingte Ausführungen und Schleifen dienen dazu, die lineare Abarbeitung eines Programms aufzubrechen.*

Anweisungsblöcke werden in geschweifte Klammern eingeschlossen.

Beispiel:

```
function AddEm(Zahl1,Zahl2)
{
    var Ergebnis = Zahl1 + Zahl2;
    return Ergebnis;
}
```

```
// Alternative, in JS übliche Schreibweise:
var AddEm = function(Zahl1,Zahl2)
{
    var Ergebnis = Zahl1 + Zahl2;
    return Ergebnis;
}
```

## CoffeeScript

```
# Anweisungsblöcke entstehen
# durch Einrückung
AddEm = (zahl1, zahl2) ->
    zahl1 + zahl2
```



# Programmsteuerung

---

Wenn-Dann-Anweisungen werden durch den **if-Befehl** abgebildet. Bedingungen können auch ineinander verschachtelt sein. Bsp.:

```
if (Bedingung)
{
    Anweisung 1;
    Anweisung 2;
    ...
}
```

## CoffeeScript

```
if Bedingung
  Anweisung 1
  Anweisung 2
  ...
```

Die alternative Bedingung wird durch den **else-Befehl** abgebildet. Bsp.:

```
if (Bedingung)
{
    Anweisung A1;
    Anweisung A2;
}
else
{
    Anweisung B1;
    Anweisung B2;
}
```

## CoffeeScript

```
if Bedingung
  Anweisung A1
  Anweisung A2
else
  Anweisung B1
  Anweisung B2
```





# Programmsteuerung

---

Mehrstufige Bedingungen werden durch den **else-if-Befehl** dargestellt. Bsp.:

```
if (Bedingung 1)
{
    Anweisungsblock A;
}
else if (Bedingung 2)
{
    Anweisungsblock B;
}
else
{
    Anweisungsblock C;
}
```

## CoffeeScript

```
if Bedingung 1
    Anweisungsblock A
else if Bedingung 2
    Anweisungsblock B
else
    Anweisungsblock C
```

Einfache Entweder-Oder-Abfragen für zwei einzelne Anweisungen können mit dem Entweder-Oder-Operator " ? : " realisiert werden.

(Bedingung) ? Erfüllt\_Anweisung : NichtErfüllt\_Anweisung;

## CoffeeScript

```
if Bedingung then Erfüllt_Anweisung else NichtErfüllt_Anweisung
```



# Programmsteuerung

---

Die Mehrseitige Auswahl wird mit der **switch-case**-Anweisung geschrieben.

```
switch (Variable)
{
    case "Wert 1": Anweisungsblock A;
        break;
    case "Wert 2": Anweisungsblock B;
        break;
    ...
    default: Anweisungsblock X;
}
```

## CoffeeScript

```
switch Variable
  when "Wert 1" then Block A
  when "Wert 2" then Block B
  else Block X
```

## Schleifen

**while**

**do-while**

**for**

**break**

**continue**

Schleife wird nur ausgeführt, wenn die Bedingung erfüllt ist,  
Schleife wird erst einmal ausgeführt, bevor Bedingung am Ende  
eines jeden Durchlaufs getestet wird oder  
gezählte Schleifen mit Anfangswert, Endwert und Schrittweite des Zählers  
stoppt Schleifenabarbeitung,  
stoppt nur den aktuellen Schleifendurchlauf.

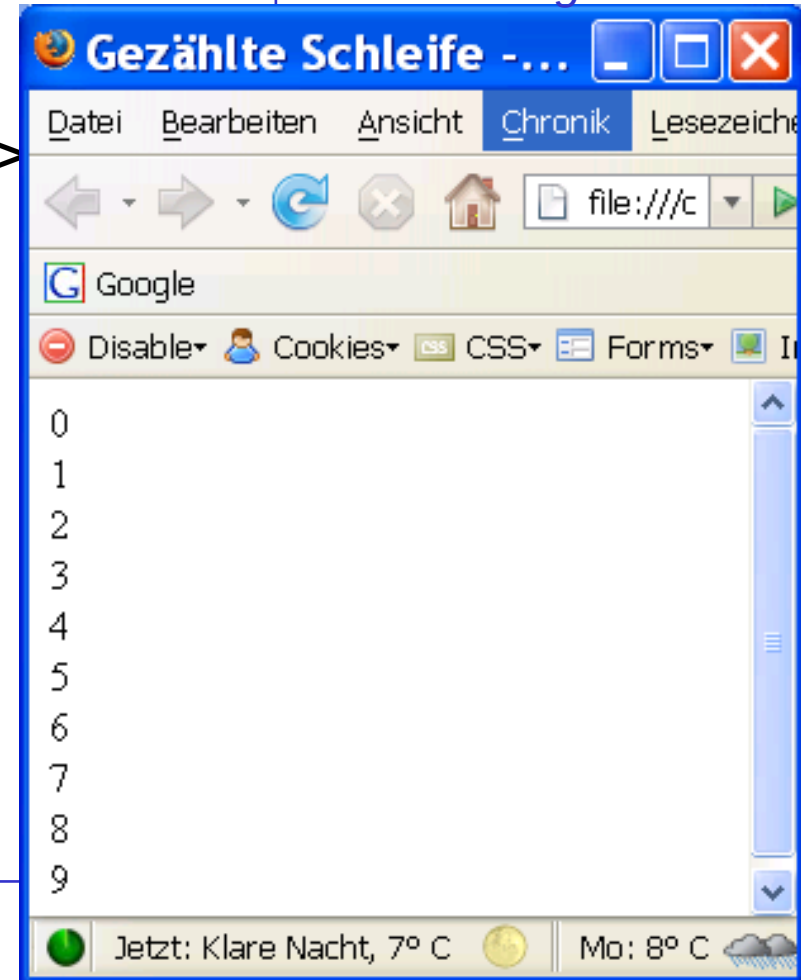


# Programmsteuerung

## Beispiel 1 - Gezählte Schleife:

```
<html>
<head>
  <Title>Gezählte Schleife</Title>
</head>
<body>
  <script language="JavaScript">
    <!--
    for (i=0; i<10; i++)
    {
      document.write(i,"<br>");
    }
    // -->
  </script>
</body>
</html>
```

*Welche Ausgabe wird im Browserfenster erzeugt?*

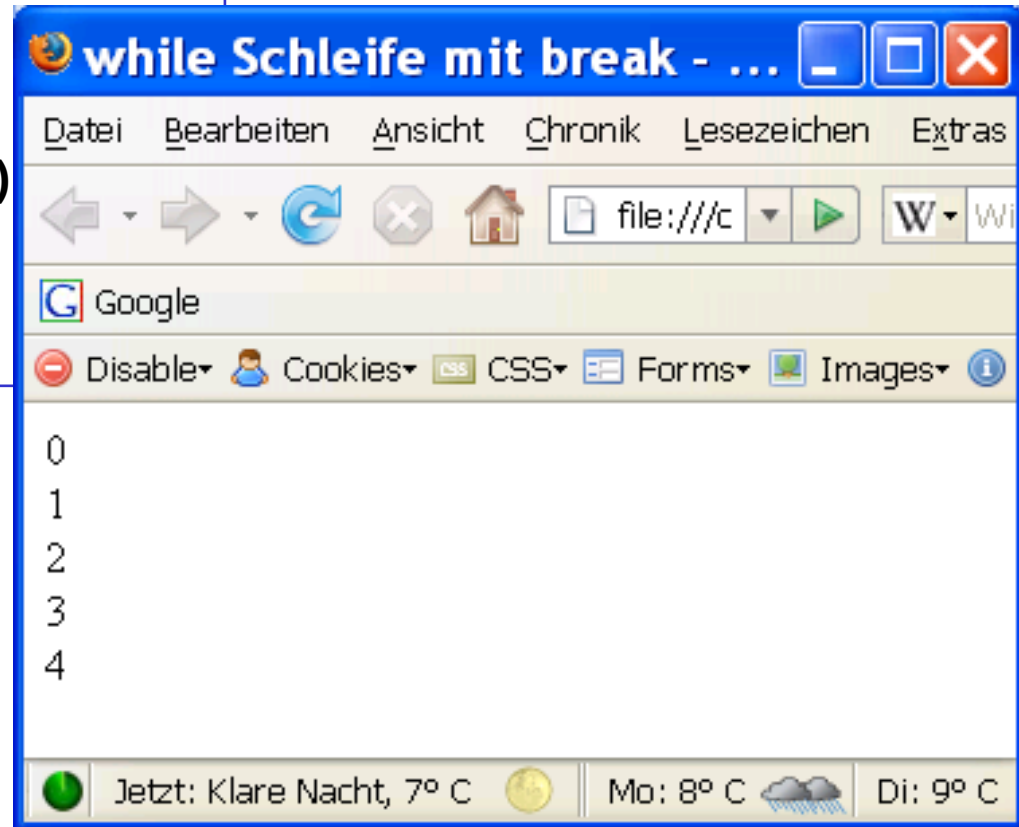


# \* Programmsteuerung

## Beispiel 2 - While-Schleife mit break:

```
<!--  
var i = 0;  
while (i < 10)  
{  
    if (i == 5)  
        break;  
    document.write(i, "<br>")  
    i++;  
}  
// -->
```

*Welche Ausgabe wird im Browserfenster erzeugt?*

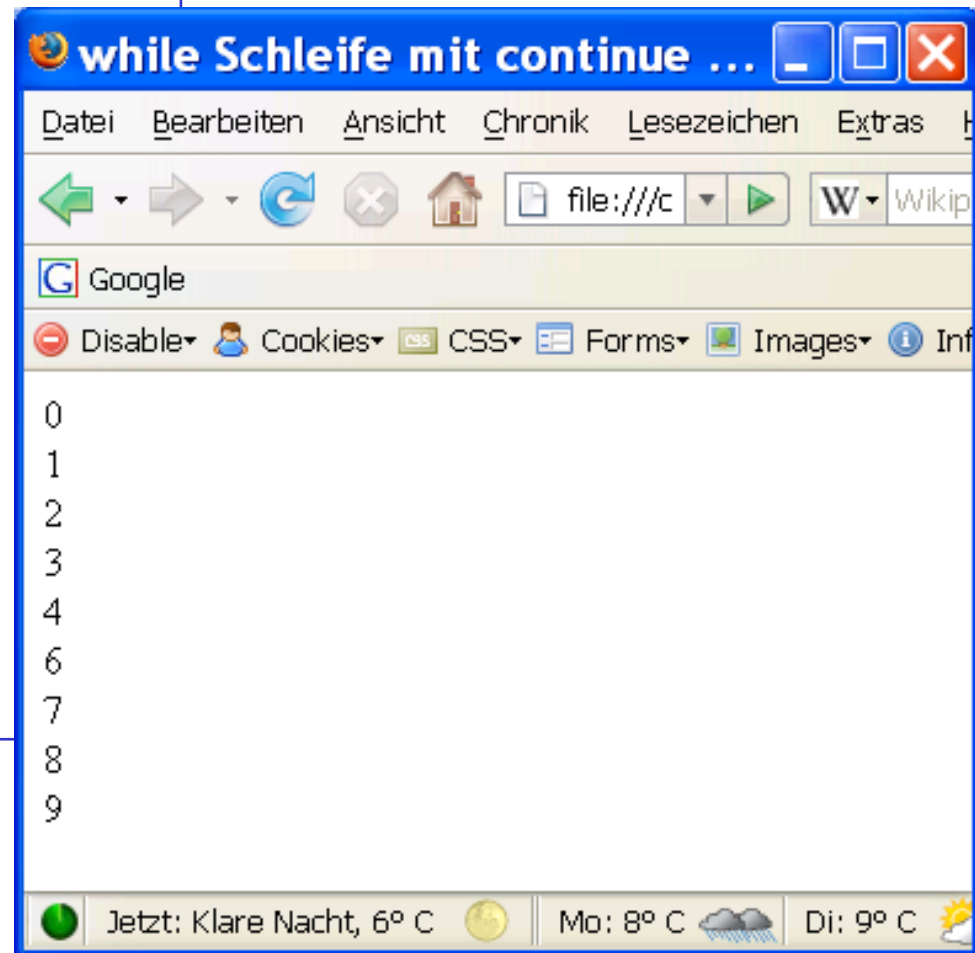


# \* Programmsteuerung

## Beispiel 3 - While-Schleife mit continue:

```
<!--  
var i = 0;  
while (i < 10)  
{  
    if (i == 5)  
    {  
        i++;  
        continue;  
    }  
    document.write(i, "<br>");  
    i++;  
}  
// -->
```

*Welche Ausgabe wird im Browserfenster erzeugt?*





# Programmsteuerung

---

## Beispiel 4 – Iteration, hier über Eigenschaften eines Objekts:

```
<!--  
for (prop in window) {  
    document.write(prop + ': ' + window[prop] + "<br/>");  
}  
// -->
```

*Ausgabe im Browserfenster:*

```
window: [object Window]  
document: [object HTMLDocument]  
InstallTrigger: [object Object]  
console: [object Object]  
...  
dispatchEvent: function dispatchEvent() { [native code] }  
dump: function dump() { [native code] }  
name:  
parent: [object Window]  
top: [object Window]
```

# \* Objekte

---

- **Objekte in JavaScript sind Sammlungen von Eigenschaften und zugeordneten Werten**

- Sie erinnern – nur anfangs! – an Hashes von Ruby oder HashMaps von Python:

```
my_obj = { prop1: 1, prop2: "2" };
```

- Per **Klammer-Notation** können auch beliebige Eigenschaftsnamen verwendet werden:

```
my_obj["yet another property"] = 3;
```

- Mehrfachzugriff auf Eigenschaften eines Objekts: „with“

```
with (my_obj) {  
    prop1 = 2;  
    prop2 = "23";  
}
```

# \* Objekte

---

- Iterieren über Objekt-Inhalte
  - Wiedergabe der Eigenschaften mit for (... **in** ...)

```
for (var property in my_obj) {  
    console.log('Property: ' + property);  
    console.log('Value: ' + my_obj[property]);  
}
```

- Wiedergabe der Eigenschafts-Werte mit for (... **of** ...)

(Non-Standard-Erweiterung, nicht in Node.js vorh.!)

```
for (var value of my_obj)  
    console.log('Value: ' + value);
```



# \* Objekte

---

- **Methoden** sind Funktionen als Werte von Eigenschaften, mit **this** wird auf das aktuelle Objekt verwiesen:

```
area = function() { return this.a * this.b };  
rect1 = { a: 2.0, b: 3.5, area: area };
```

```
rect1.a      → 2.0  
rect1.b      → 3.5  
rect1["b"]   → 3.5  
rect1.area() → 7.0  
rect1.area   → Function { ... }
```

- Beachte die Notwendigkeit der Klammern „()“: **area()** ruft die Funktion auf, während **area** nur den Wert der Eigenschaft „area“ liefert – der hier kein String, Array etc. ist, sondern eine Funktion
- „**this**“ in JS entspricht „**self**“ in Ruby oder Python

# \* Funktionen

- „**Functions are first-class citizens**“

Funktionen werden in JS ähnlich wie andere Variablenwerte zugewiesen, als Parameter übergeben und auch als Ergebnisse zurückgegeben. Sie spielen in JS eine fundamentale Rolle.

- Gewöhnliche, benannte Funktion:

```
function abs(x) {return (x < 0 ? -x : x); };  
console.log(abs(2), abs(-2)); // 2 2
```

- Anonyme JS-Funktion, auf Variable gelegt:

```
var abs = function(x) {return (x < 0 ? -x : x); };  
console.log(abs(2), abs(-2)); // 2 2
```

- Neu in ECMAScript 6: *arrow functions*

```
let abs = (x) => x < 0 ? -x : x;  
console.log(abs(2), abs(-2)); // 2 2
```

CoffeeScript-Version:

```
abs = (x) -> if (x < 0) then -x else x  
console.log abs(2), abs(-2)
```

# \* Funktionen

- „*Higher order functions*“

Funktionen, die andere Funktionen als Argumente erhalten. JS erreicht damit eine funktionale Programmierung und ähnliche Effekte wie Ruby mit Code-Blöcken von Methoden:

- Bibliothek „underscore.js“:

```
var result = _.map([1,2,3,4], function(x) {  
    return x * x;  
});
```

```
console.log(result);  
[1, 4, 9, 16]
```

# Ruby:

```
result = [1,2,3,4].map {|x| x*x}
```

# CoffeeScript:

```
result = _.map [1,2,3,4], (x)-> x*x
```

```
_.each([1,2,3,4], function(x) {  
    console.log(x);  
});
```

1 2 3 4

# Ruby:

```
[1,2,3,4].each {|x| print x, " "}
```

# CoffeeScript:

```
_.each [1,2,3,4], (x)-> console.log x
```

# Funktionen

---

*Funktionen können den Code übersichtlicher gestalten. Sie werden oft von Ereignissen ausgelöst und bringen so Dynamik auf die Webseite.*

## Syntax für Funktionsdefinition:

Eine Funktion ist ein Anweisungsblock, der mit dem Schlüsselwort **function**, optional dem Namen der Funktion und **runden Klammern** eingeleitet wird.

```
function Funktionsname (Parameter 1, Parameter 2, ... )  
{  
    ... // JavaScript - Anweisungen  
}
```

## Syntax für Aufruf der Funktion in einem Anweisungsblock:

```
Anweisung 1;  
Anweisung 2;  
X = Funktionsname( Parameter 1, Parameter 2, ... );  
Anweisung 3;
```

# \* Funktionen

*Welche Ausgabe wird im Browserfenster erzeugt?*

## Beispiel Funktionen aufrufen:

```
<html>
<head>
  <Title>Funktionen</Title>
</head>
<body>
  <script language="JavaScript">
    <!--
    function Produkt (x, y) //Funktion
    {
      var Ergebnis = x * y;
      return Ergebnis;
    }
    var MeinProdukt = Produkt (33, 11); //Aufruf der Funktion
      //mit 2 Parametern
    document.write("Das Produkt aus 33
      und 11 ergibt: ", MeinProdukt);

    // -->
  </script>
</body>
</html>
```



# \* Objekte

---

- Arrays und andere eingebaute Objekte besitzen besondere Eigenschaften und Methoden, sind aber auch erweiterbar wie einfache Objekte:

```
var ary = [1, "2"];  
var obj = {0: 1, 1: "2"};
```

```
ary[0]           → 1  
obj[0]           → 1  
ary[1]===obj[1]  → true  
ary.length       → 2  
obj.length       → undefined  
ary['extra prop'] = "foo"  
ary.length       → 2
```

# \* Objekte

---

- Array-Methoden:

```
concat(), indexOf(), join(), lastIndexOf(), pop(), push(),  
reverse(), shift(), slice(), sort(), splice(), toString(),  
unshift(), valueOf()
```

- Anwendungsbeispiele:

```
var ary = [1, "2"];  
  
ary.reverse(); → ary is ["2", 1]  
ary.push(3); → ary is ["2", 1, 3]  
ary.pop(); → 3  
           // ary is ["2", 1]  
ary.unshift(0); → ary is [0, "2", 1]  
ary.shift(); → 0  
            // ary is ["2", 1]
```

# \* Objekte

- Arrays iterieren?

```
squares = [1, 4, 9, 16];
```

```
/* Version 1: Wie in C */
```

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

```
/* Echtes Iterieren? */
```

```
for (var x in squares)  
    console.log(x); /* Ergibt 0, 1, 2, 3 ! */
```

```
// Non-Standard-Erweiterung:
```

```
for (var x of squares)  
    console.log(x); /* Ergibt 1, 4, 9, 16 ! */
```

- Grund:
  - Schlüsselwort „in“ ermittelt die Eigenschafts-Namen (nicht: -Werte!) des Objekts. Beim Array sind das die implizit vorhandenen Indices!
  - Die for-in-Variante akzeptiert alle Objekte – nicht nur Arrays





# Objekte vergleichen

---

- Wann sind zwei Objekte gleich?
  - Operatoren **==**, **!=** :  
Wenn die zu vergleichenden Objekte nicht vom gleichen Typ sind, passt JS sie erst an und vergleicht dann auf (Un-)Gleichheit

```
2 + 3 == 5           // true
2 + 3 == '5'         // true
1 == true            // true
0 == false           // true
[1, 2] == [1, 2]     // false (different references)
```

- Operatoren **===**, **!==** : Keine Umwandlung, false bei Ungleichheit

```
2 + 3 === 5          // true
2 + 3 === '5'        // false
1 === true           // false
0 === false          // false
```



# Klassen vs. Prototypen

---

- **JS: Keine Klassen!**
  - JS ist eine objekt-orientierte Sprache, kennt aber keine Klassen.
  - Sie verfolgt vielmehr eine Ableitung von Objekten von Vorlagen (also bereits bestehenden Objekten), den sogenannten **Prototypen**
  - Sie kennt ferner **Konstruktoren** und das Schlüsselwort „**new**“
  - Wir werden hier auf Prototyp-Verwendung verzichten und Konstruktor-Funktionen an einem Beispiel kurz einführen. Sie sollten diese JS-Grundlagen kennen, weil es noch viel alten JS-Code gibt, der sie verwendet.
  - Für neue Entwicklungen können Sie stattdessen die seit ECMAScript 6 verfügbare Klassen-Syntax verwenden.
  - Wichtiger ist der Umgang mit Funktionen als Parametern (Callback-Technik)
- **CS: Klassen!**
  - Klassen werden simuliert mit JS-Prototypen & Konstruktoren
- **ECMAScript 6 / JS 2.0:** Einführung von Klassen, ähnlich wie bei CS

# \* Objekte

---

Die **Objekte** bestehen aus **Eigenschaften und Methoden**.

**Eigenschaften** sind eine Ansammlung von Werten, die das Objekt beschreiben.

**Methoden** sind objekteigene Funktionen, die die Eigenschaften oder andere Werte manipulieren.

Eine **Konstruktor-Funktion** definiert ein Objekt.

**Instanzen** des Objektes werden mit „**new**“ und mit Hilfe der Konstruktor-Funktion erzeugt.

## Beispiel: Definition des Objekts Halloweengast durch gleichnamige Konstruktor-Funktion:

```
function Halloweengast(Name, Haarfarbe, Augenfarbe)
{
    this.Name = Name;
    this.Haarfarbe = Haarfarbe;
    this.Augenfarbe = Augenfarbe;
    this.KleidetSich = KleidetSich; //Methode, ohne Klammern!
}
```



# Objekte: Beispiel

---

## Beispiel: Definition der Methode KleidetSich als separate Funktion:

```
function KleidetSich(Tag)
{
    document.write("Am " + Tag + " hatte " + this.Name + " " +
        this.Haarfarbe + "es Haar und " + this.Augenfarbe + "e Augen.");
}
```

- Funktion greift mit „**this**“ auf die aktuelle Instanz zu.

## Beispiel: Instanz von Halloweengast mit Konstruktor-Funktion Halloween:

```
function Halloween()
{
    Tag = "31.Oktober 2014";
    Hans = new Halloweengast("Hans", "gr&uuml;n", "rot"); // Instanz
    Hans.KleidetSich(Tag); // Funktionsaufruf, mit Klammern!
}
```

- mit der „**new**“-Funktion wurde eine Instanz von Halloweengast erzeugt. Es werden die Eigenschaften Hans, grün und rot als Parameter übergeben.

# Der JS-Code mitsamt seiner HTML-Seite

```
<html><head><title>Objekte zu Halloween</title>
<script type="text/javascript">
<!--
    function Halloweengast(Name, Haarfarbe, Augenfarbe)
    {
    this.Name = Name;
    this.Haarfarbe = Haarfarbe;
    this.Augenfarbe = Augenfarbe;
    this.KleidetSich = KleidetSich; // Funktion zuweisen
    }
    function KleidetSich(Tag) // Funktion definieren
    {
    document.write("Am " + Tag + " hatte " + this.Name + " " +
        this.Haarfarbe + "es Haar und " +
        this.Augenfarbe + "e Augen.");
    }
    function Halloween()
    {
    Tag = "31.Oktober 2014";
    Hans = new Halloweengast("Hans", "gr\u00fcn", "rot"); // Instanz erstellen
    Hans.KleidetSich(Tag);
    } // -->
</script>
</head><body>
<script type="text/javascript">
<!--
    Halloween();
// -->
</script>
</body></html>
```

*Welche Ausgabe wird im Browserfenster erzeugt?*

*„Am 31. Oktober 2014 hatte Hans grünes Haar und rote Augen.“*



# CoffeeScript-Variante:

---

```
# Funktionsdefinition
kleidet_sich = (tag) ->
  "Am #{tag} hatte #{@name} #{@haarfارbe}es Haar und #{@augenfarbe}e Augen."

# Prototypen-Definition („Konstruktor“-Funktion)
Halloweengast = (name, haarfارbe, augenfarbe) ->
  @name = name
  @haarfارbe = haarfارbe
  @augenfarbe = augenfarbe
  @kleidetSich = kleidet_sich
  return # Wichtig - hier nicht auslassen!

# Funktionsdefinition
halloween = ->
  tag = "31. Oktober 2014"
  hans = new Halloweengast("Hans", "grün", "rot")
  hans.kleidetSich(tag)

console.log halloween() # Funktionsaufruf, daher mit Klammern „()“!
```

Hier: Standalone-Version mit Konsolen-Ausgabe statt im Browser

# CoffeeScript-Variante mit CS-Klasse:

---

```
# Klassendefinition
```

```
class HalloweenGast
```

```
  constructor: (@name, @haarfارbe, @augenfarbe) ->
```

```
    kleidetSich: (tag) ->
```

```
      "Am #{tag} hatte #{@name} #{@haarfارbe}es Haar und #{@augenfarbe}e Augen."
```

```
# Anwendung: Neues Objekt erzeugen, Methode aufrufen:
```

```
hans = new HalloweenGast("Hans", "grün", "rot")
```

```
console.log hans.kleidetSich("31. Oktober 2014")
```

Hier: Standalone-Version mit Konsolen-Ausgabe statt im Browser. Die Ausgabe lautet:

**Am 31. Oktober 2014 hatte Hans grünes Haar und rote Augen.**

# ECMAScript-Variante mit neuer Klasse:

---

```
// Klassendefinition

class HalloweenGast {
  constructor(name, haarfarbe, augenfarbe) {
    this.name = name
    this.haarfarbe = haarfarbe
    this.augenfarbe = augenfarbe
  }
  kleidetSich(tag) {
    return `Am ${tag} hatte ${@name} ${this.haarfarbe}es Haar und ${this.augenfarbe}e Augen.`
  }
}

// Anwendung: Neues Objekt erzeugen, Methode aufrufen:

hans = new HalloweenGast("Hans", "grün", "rot")
console.log(hans.kleidetSich("31. Oktober 2014"))
```

Hier: Standalone-Version mit Konsolen-Ausgabe statt im Browser. Die Ausgabe lautet:

**Am 31. Oktober 2014 hatte Hans grünes Haar und rote Augen.**



# Trend: *Unobtrusive JavaScript* (UJS)

---

JavaScript-Code wird restlos aus HTML entfernt, analog zu CSS, und aus ähnlichen Gründen.

*Das schließt auch den Aufruf von Funktionen über Event-Attribute wie „onLoad“ oder „onClick“ ein!*

Interaktion zwischen HTML und JS findet statt über *Event Handler* und Selektoren im JS-Code. Dabei kommen meist leistungsfähige Bibliotheken wie jQuery zum Einsatz, die u.a. CSS-artige Selektoren bieten.

**HTML5** unterstützt diese Entwicklung mit einer neuen Attribut-Konvention: **Attribute** mit dem **Präfix „data-“** dürfen in fast allen Elementen vorkommen. Über sie sind flexible Selektionen und Interaktionen möglich.

# \* Trend: *Unobtrusive JavaScript* (UJS)

---

- Beispiel aus Rails 2.x: Link zum Löschen eines Eintrags (Tabellenzeile). Folgender Helper:

```
link_to('Löschen', path, :method => :delete, confirm: 'Sind Sie sicher?')
```

- erzeugt diesen Code in der HTML-Datei:

```
<a href="/project_proposals/2" onclick="if (confirm('Sind Sie sicher?')) {  
  // Zusammenbau eines unsichtbaren Formulars - Ändern nur mit POST  
  var f = document.createElement('form');  
  f.style.display = 'none';  
  this.parentNode.appendChild(f);  
  f.method = 'POST';  
  f.action = this.href;  
  var m = document.createElement('input');  
  m.setAttribute('type', 'hidden');  
  m.setAttribute('name', '_method');  
  m.setAttribute('value', 'delete');  
  f.appendChild(m); // ... (weitere 5 Zeilen Code)  
  f.submit(); };  
return false;">Löschen</a>
```

- Dieser ganze Code wurde für jede Tabellenzeile in die HTML-Seite eingebettet!

# *Unobtrusive JavaScript (UJS)* – verwenden!

---

Rails ab V3.x: Analoge Konstruktion, mit UJS:

```
link_to('Löschen', path, :method => :delete,  
        data-confirm: 'Sind Sie sicher?')
```

ergibt nun:

```
<a href="/project_proposals/2" data-confirm='Sind Sie sicher?'  
  data-method='delete'>Löschen</a>
```

Zur Auswertung von „data-confirm“ liefert Rails ab V3 Standard-JS-Code mit, der am Ende das Gleiche bewirkt.

Eigener JS/CS-Code zu selbst definierten data-\*-Attributen ermöglicht nahezu beliebige Aktivitäten. Hinweis: Praktikum 05, Attribut „data-cmd“

- DOM (Document Object Model)
  - DOM Core: Das API von JavaScript für Zugriffe auf HTML- und XML-Seiteninhalte
  - Grundlage: Abstraktes Knotenmodell von HTML/XML
    - Verschiedene Knotentypen
  - Level 1: 1.10.1998      <http://www.w3.org/TR/REC-DOM-Level-1>
  - Level 2: 13.11.2000      <http://www.w3.org/TR/DOM-Level-2-Core>
  - Level 3: 7.4.2004      <http://www.w3.org/TR/DOM-Level-3-Core>
  - Aktuell: Gemeinsame Weiterentwicklung mit HTML5

- DOM (Document Object Model)
  - In diesem Kurs geben wir jQuery den Vorrang vor dem Arbeiten mit dem elementarerem DOM
  - Wir besuchen stattdessen das **DOM-Tutorial von W3Schools.com** unter **[https://www.w3schools.com/js/js\\_htmlDOM.asp](https://www.w3schools.com/js/js_htmlDOM.asp)**



# ***Frameworks***

# Frameworks und Bibliotheken

---

- **Vorteile**

- Bündelung der bekannten Technologien
- Code läuft auf möglichst vielen Clients
- Verwendung von bereits vorhandenem Code
- Häufig als Open-Source-Projekt verfügbar

- **Nachteile**

- Einarbeitungszeit ggf. hoch
- Häufig sehr umfangreich, Problemlösung nicht enthalten
- Abhängigkeiten zu Hersteller/Projekt, Philosophie

- **Generell**

- Verwendung wird immer populärer

# Frameworks und Bibliotheken

---

- **Wobei soll ein Framework unterstützen?**
  - Funktionalität für verschiedene Browser zusichern
  - Überprüfung von Formulareingaben mittels Schablonen
  - Dynamische Strukturen (Schaltflächen, Menüs, Animationen, Slideshows,...)
  - Strukturierte Webseiten (Browser, Betriebssystem, -version, Bildschirmauflösung,...)



# Frameworks und Bibliotheken

---

- Wann sollte ein Framework verwendet werden?

## Faustregel:

Je größer, je mehr multimedial und je interaktiver die Webpräsenz, desto sinnvoller ist der Einsatz eines Frameworks, insbesondere, wenn man Probleme hat, deren manuelle Umgehung aufwändig ist.

# Frameworks und Bibliotheken

---

- **JavaScript-Frameworks**

- Sind rein für (X)HTML, JavaScript und/oder CSS
- Open-Source-Projekte
- Betrachtet werden soll: **jQuery**
  
- Weitere sind:
  - Prototype, Dojo Toolkit, MooTools, MochiKit, Ext JS oder qooxdoo
  - Bootstrap, AngularJS, ...
  
- TypeScript-Frameworks
  - Inzwischen ist TypeScript so populär, dass einige bekannte JS-Frameworks auf TypeScript umgestellt worden sind. Bekanntestes Beispiel: Angular

# Frameworks und Bibliotheken

---

- jQuery

- populärstes Framework für Webapplikationen
- seit 2006 verfügbar
- Stellt Funktionen zur DOM-Manipulation und –Navigation sowie AJAX-Support zur Verfügung
- CSS, erweitertes Event-System, Effekte, Animationen, Hilfsfunktionen, Plug-ins, ... stehen zur Verfügung
- beliebig erweiterbar !!!
- Einbindung in Webplattformen möglich (z.B. Microsoft Visual Studio, Nokia Web-  
Runtime-Plattform, ...)
- In Rails fertig integriert bis Rails 5.0
  - ersetzte damals prototype.js & scriptaculous.js
- **<http://jquery.com>** - aktuelle Version 3.6.0
- Lernbar hier: **<https://www.w3schools.com/jquery/default.asp>**

- Kurze Einführung (I)

Zentrale jQuery-Funktion:

`$()`

Element-Selektion mit Element-Namen:

`$("p")`

Element-Selektion mit ID:

`$("#Absatz")`

Element-Selektion mit einer bestimmten Klasse:

`$("p.meineKlasse")`

Wie bei CSS-  
Selektoren!

- Kurze Einführung (II)

Beispielzugriffe:

HTML-Markup:

```
<ul id="Liste"></ul>
```

Zugriff auf Liste und Anhängen eines neuen Elements:

```
$("#Liste").append($("<li>"))
```

und weiter...

```
$("#Liste").append($("<li>").append($("<a>"))
```

und nochmal weiter...

```
$("#Liste").append($("<li>").append($("<a>").attr("href",  
"http://jquery.com"))).text("jQuery");
```

- Ein Beispiel: Inhalte dynamisch ändern

```
<html>
  <head>
    <title>jQuery</title>
    <meta charset="UTF-8"></meta>
    <script type="text/javascript" src="jquery-2.1.4.js"></script>
    <script type="text/javascript">

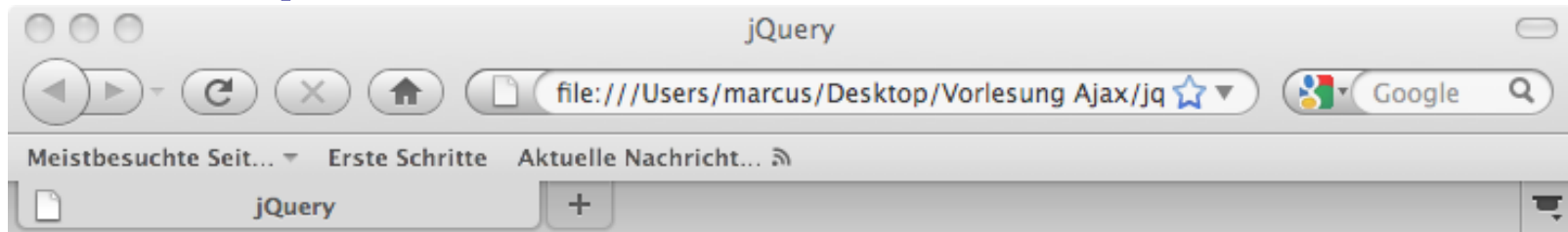
      $(document).ready(function() {

        $("#Liste").append($("<li>").append($("<a>").attr("href",
          "http://www.jquery.com/").text("jQuery")));

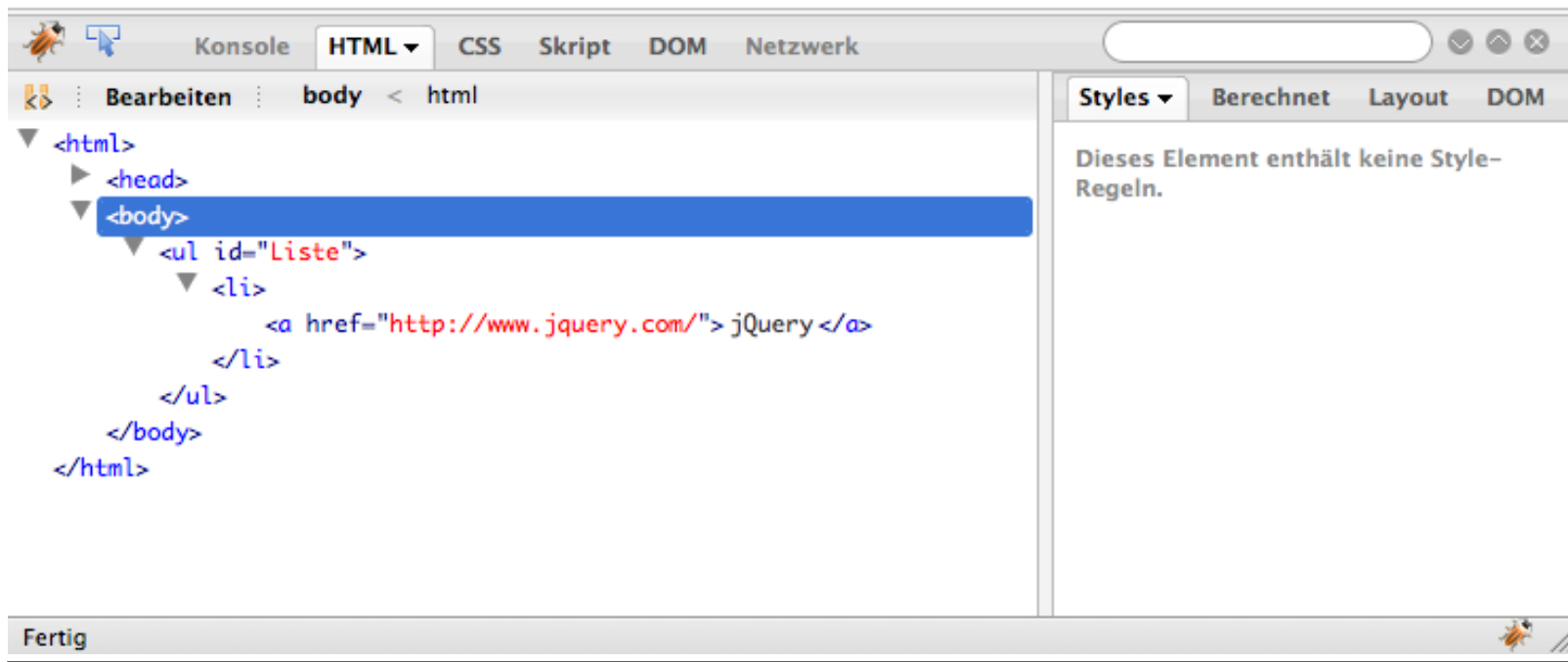
      });
    </script>
  </head>
  <body>
    <ul id="Liste"></ul>
  </body>
</html>
```

# \* Frameworks und Bibliotheken

- Ein Beispiel



- [jQuery](#)





# Zwischenstand

---

- Dank jQuery können wir also schon recht einfach
  - **Elemente einer Seite gezielt selektieren**
    - und sie anschließend manipulieren, z.B. mit `hide()` und `show()`, durch Änderung ihrer CSS-Eigenschaften und Inhalte
  - **Teile einer Seite neu hinzufügen**
    - wie im o.g. Beispiel das Listenelement mit dem jQuery-Link
  - **auf Ereignisse reagieren**
    - wie im Praktikum auf Klicks auf Taschenrechner-Tasten
- Nun werden wir noch
  - **Daten** asynchron vom Server **anfordern**
    - mit der **AJAX**-Unterstützung von jQuery
    - im **JSON**-Format
  - und in unserer Webseite **zur Darstellung bringen**
    - als Tabelle und sogar als Grafik



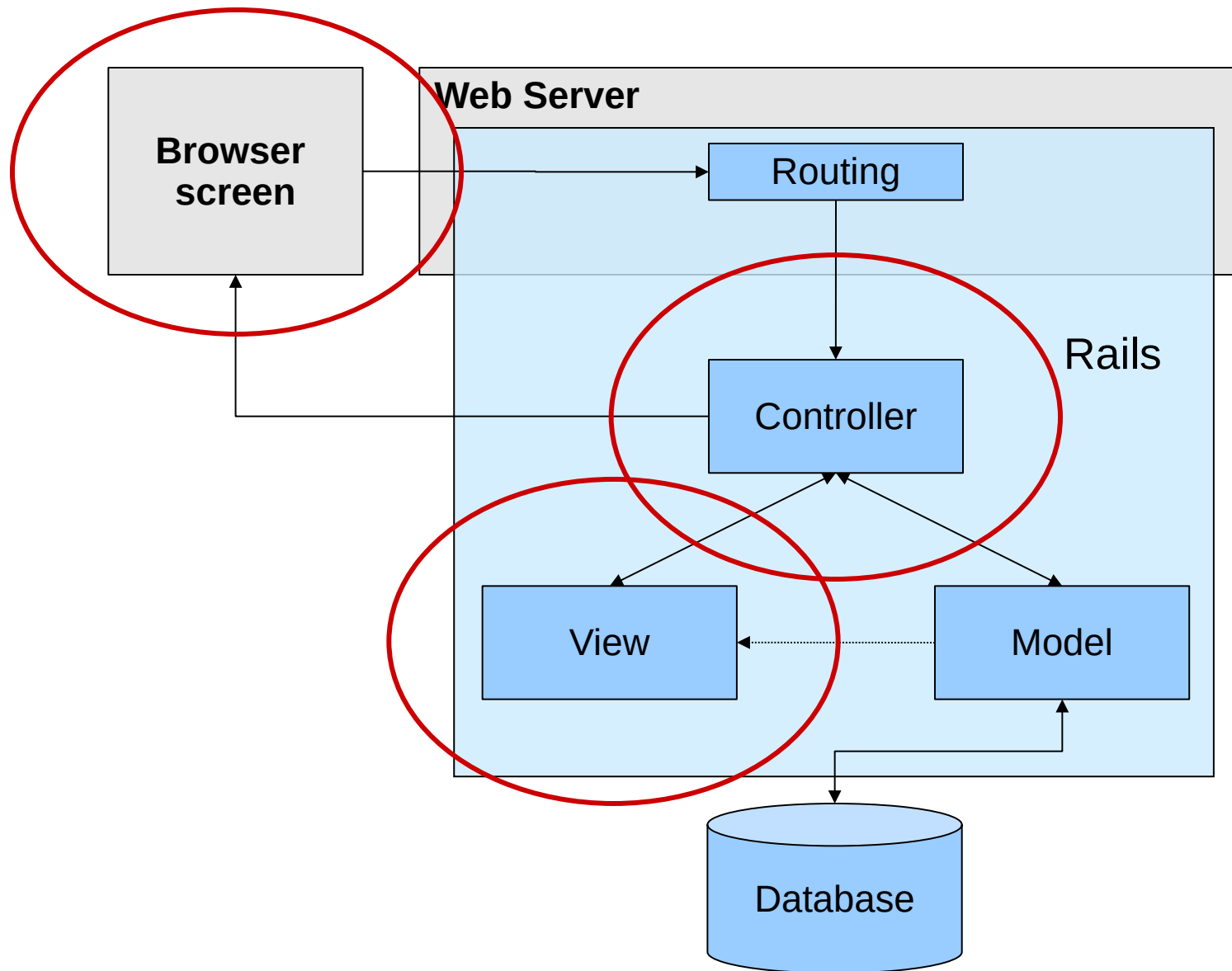


***AJAX***      ***!=***



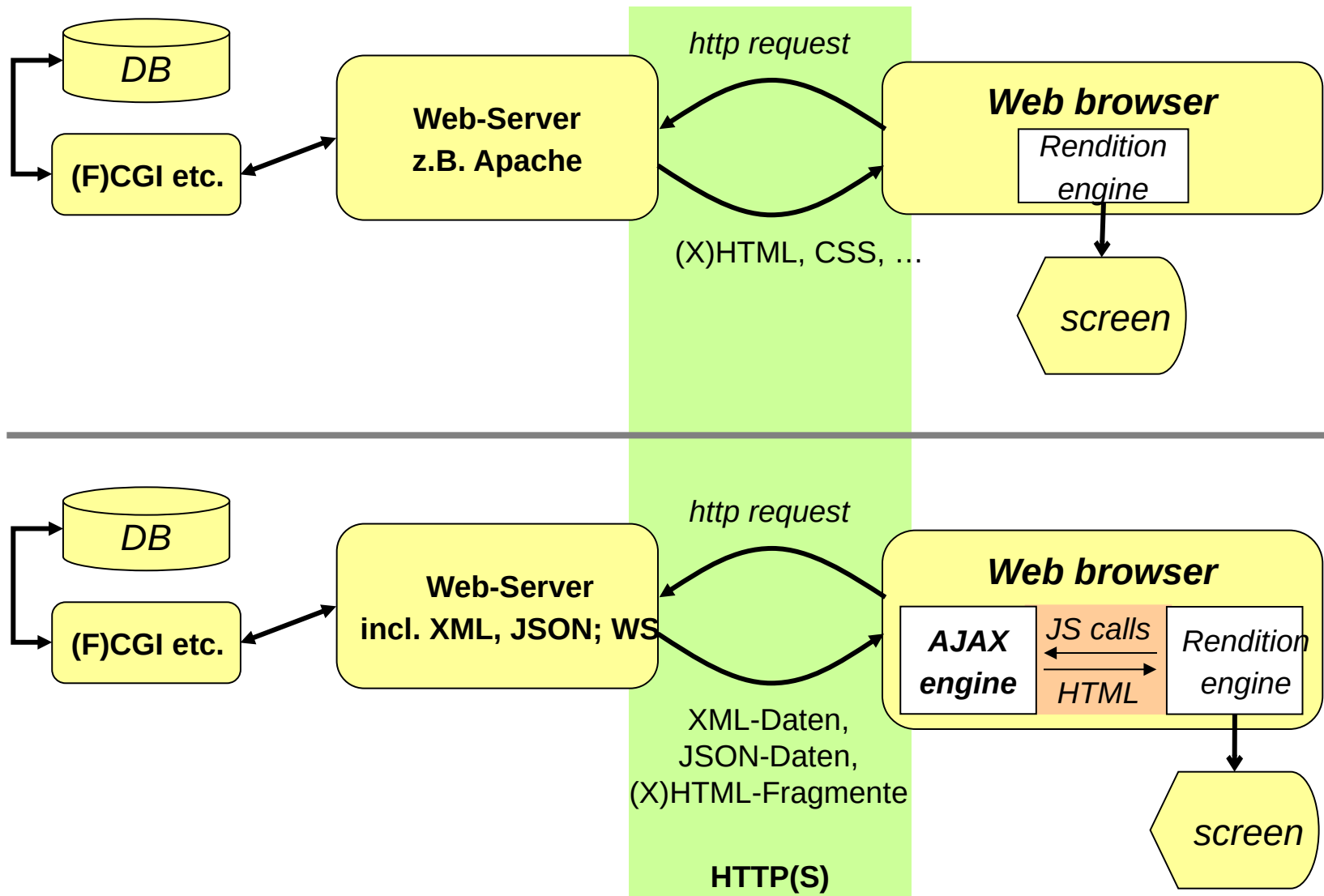
***Asynchronous JavaScript and XML***  
Interaktivere Benutzerschnittstellen

# \* Views



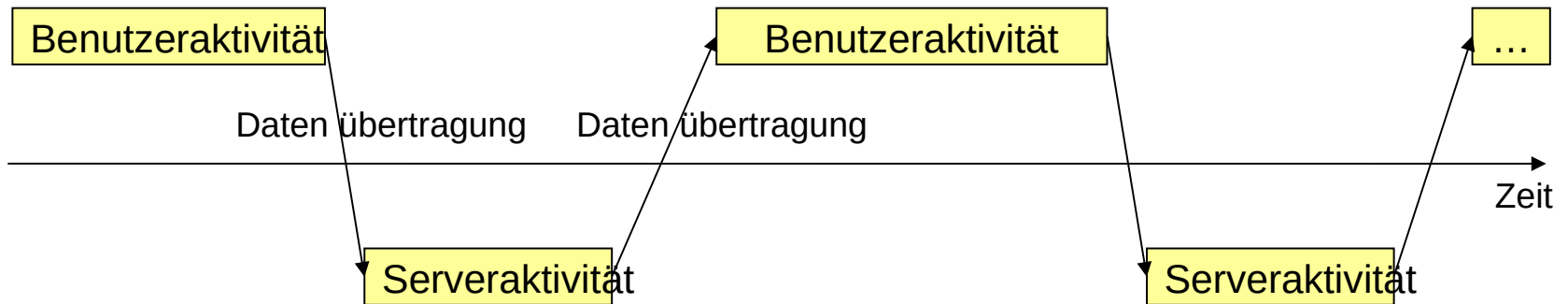
- Ajax – Eine Mischung bekannter Techniken:
  - Präsentation von Information auf der Basis von Standards, insbesondere von **XHTML** bzw. **HTML5** und **CSS**
  - Dynamische Anzeigen und Interaktion mit den Inhalten mittels **DOM** (*Document Object Model*) oder Bibliotheken wie **jQuery**
  - Datenrepräsentation und –transformation mit **XML** und **XSLT**
  - Asynchroner Datenaustausch mit **XMLHttpRequest**
  - und **JavaScript**, um all dies zu verbinden

# \* WBA: Ajax vs. traditionelle Interaktionen



# \* Ajax: Synchrone Datenübertragung

---



**Traditionelle WBA wechseln zwischen Benutzer- und Server-Aktivitäten**

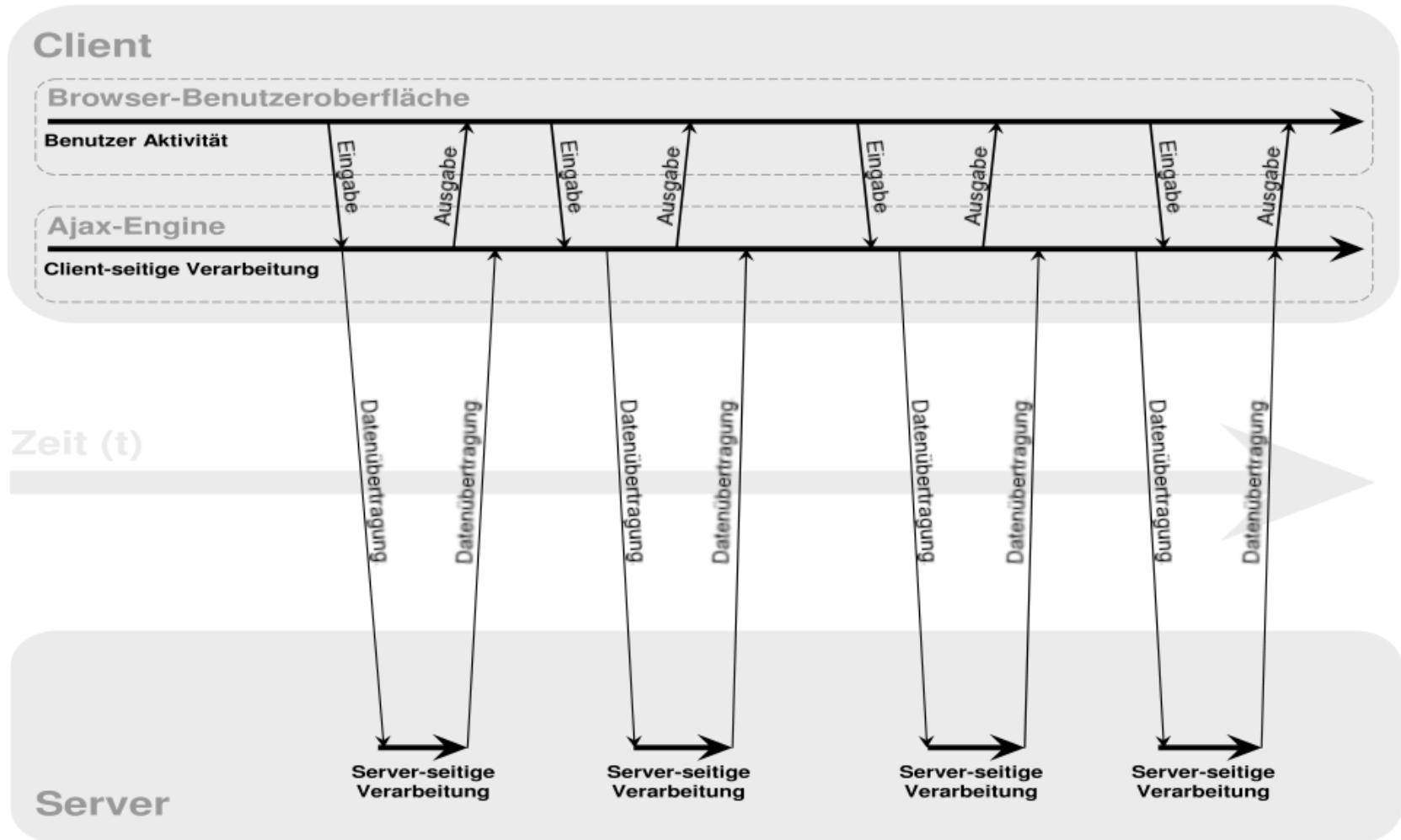
**Anwender empfinden die entstehenden Wartezeiten als störende Unterbrechungen ihres Arbeitsflusses.**

# \* Ajax: Alternative Implementierungen

---

- Direkte Ajax-Implementierung
  - *Client* besitzt API zur XML- bzw. JSON-basierten Kommunikation mit dem Server (XMLHttpRequest)
  - Datenaustausch effizient und flexibel,
  - Transformation erforderlich, komplexer *client*
- Indirekte Ajax-Implementierung
  - Client tauscht HTML-Fragmente mit Server aus
  - Client aktualisiert Darstellung mittels DOM
  - Beispiele:
    - Einfügen / Aktualisieren von Listeneinträgen,
    - Positionen eines Warenkorbs, einer Bestellung etc.
  - Ein solcher Mechanismus ist bereits in Rails implementiert.

## Ajax-Modell einer Web-Anwendung (asynchrone Datenübertragung)



Quelle: de.wikipedia.org

- Beispielanwendungen

- *Frühes Beispiel: Google Maps:* <http://maps.google.de>
  - „Gleitende“ Verschiebung des Sichtbarkeitsfensters auf der Karte,
  - vorausschauendes Nachladen der nächsten Kacheln asynchron im Hintergrund
  - Ähnliche Wirkung mit Java Applets: Stadtplan Wiesbaden.
- *Rails' [Turbolinks](#):*
  - Unser Rails-Framework nutzt AJAX-Technik zur Beschleunigung beim Seitenaufbau:
  - An die Stelle eines kompletten Seitenaufbaus mitsamt Laden aller CSS- und JS-Dateien aus dem Header wird bei Site-internen Klicks nur der Inhalt von „body“ ausgetauscht – per AJAX call!



# \* Frameworks und Bibliotheken

- Ajax-Anfrage mit jQuery

```
$.ajax({  
  url: "/api/getWeather",  
  data: {  
    zipcode: 65195  
  },  
  success: function( result ) {  
    $("#weather-temp").html("<strong>" + result + "</strong> °C");  
  }  
});
```

Eingabedaten

(JS-Version)

Code, der asynchron (!) nach fehlerfreiem Eintreffen der Antwort starten soll (callback)

```
$.ajax  
  url: "/api/getWeather"  
  data:  
    zipcode: 65195  
  success: ( result ) ->  
    $("#weather-temp").html("<strong>" + result + "</strong> °C")
```

(CS-Version)

(Beispiel von der jQuery-Homepage, leicht variiert)

# \* Ajax: Stimmt das Akronym?

---

- **A** wie asynchron:  
Ja (Regelfall), aber auch synchrone Aufrufe sind möglich
- **J** wie Javascript:  
Ja (Regelfall), aber auch Präcompiler (CoffeeScript, TypeScript sind heute verbreitet)
- **a** wie and
- **X** wie XML:  
Inzwischen werden XML-basierte Austauschformate eher selten bzw. nur für komplexere Datenstrukturen eingesetzt
- Sehr beliebter Nachfolger: **JSON** - unser nächstes Thema!



# JSON

JavaScript Object Notation

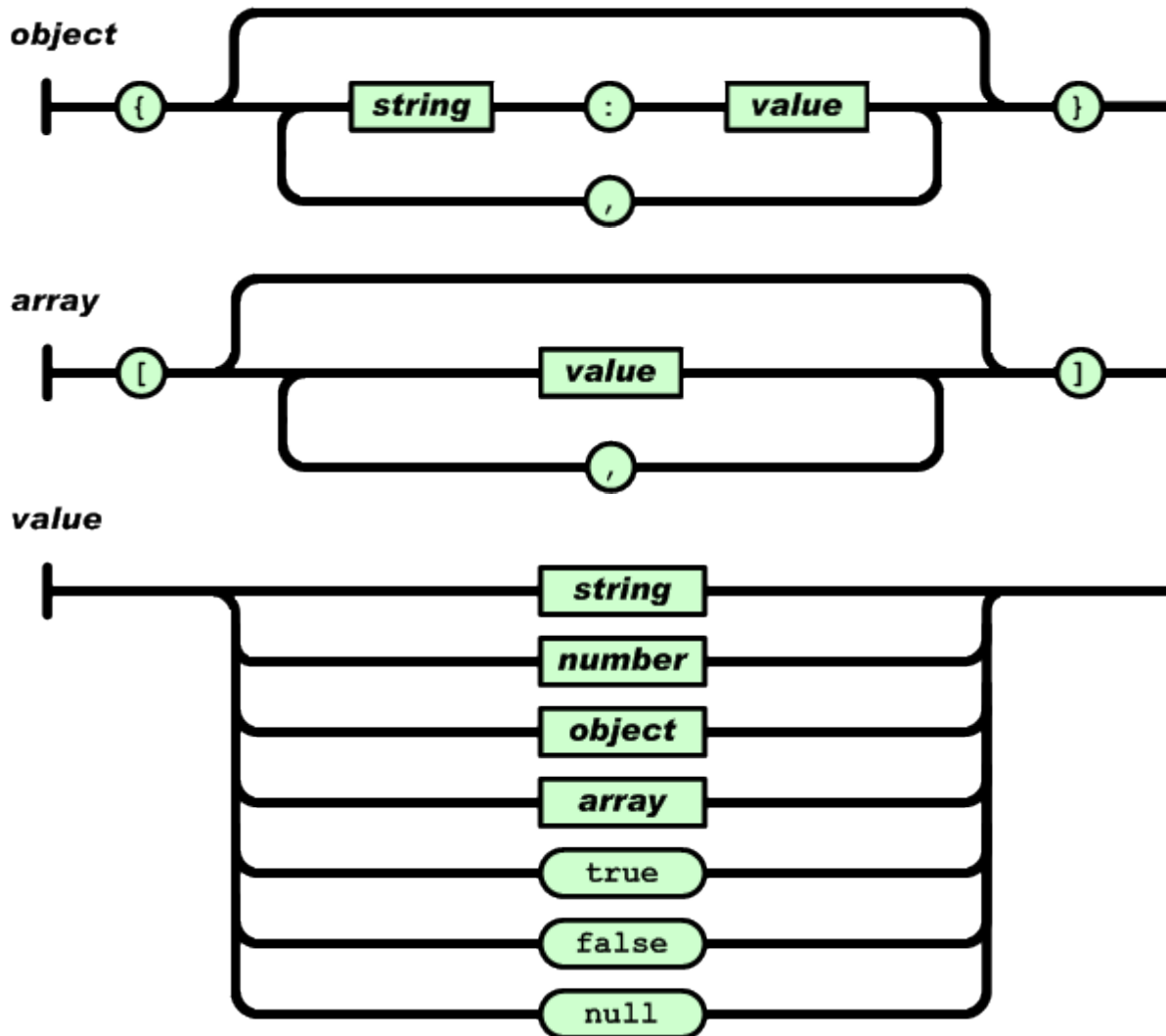
- Situation
  - Viele aktuelle Web-Anwendungen, sog. *single page applications*, laden nur noch ein HTML-Grundgerüst vom Server. Umfangreicher JS-Code baut die HTML-Seiten dann clientseitig auf. Mit dem Server werden nur noch die Nutzdaten ausgetauscht, nicht mehr ihre HTML-Darstellungen.
  - Dieses Vorgehen ist technisch verwandt mit mobilen Anwendungen in Form von „Apps“.
  - Mit AJAX haben wir gerade eine Technik kennengelernt, um Inhalte vom Server unter JS-Kontrolle nachzuladen und in die Browserseite einzufügen. Dabei handelte es sich aber zunächst nur um unstrukturierten Text oder HTML-Fragmente.
- Anschlussfrage
  - Wie tauscht man strukturierte Daten aus?
- Kandidaten:
  - CSV; HTML-Fragmente; XML (verschiedene DTDs/Schemata), YAML
  - **Der JavaScript-Weg: JSON**

## JSON: Vorstellung auf [www.json.org](http://www.json.org):

- **JSON** (*JavaScript Object Notation*) is a *lightweight data-interchange format*.
  - *It is easy for humans to read and write.*
  - *It is easy for machines to parse and generate.*
  - *It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999.*
  - *JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.*
  - *These properties make JSON an ideal data-interchange language.*
- **Bemerkungen**
  - Ursprünglich ein Mitbewerber von YAML
  - Seit YAML 1.2 ist JSON eine echte Untermenge von YAML!
  - Inzwischen weit über JavaScript hinaus verbreitet

# \* JSON

## JSON: Struktur-Diagramme von [www.json.org](http://www.json.org):



Bemerkung:  
Diagramme für  
**string**  
und  
**number**  
hier ausgelassen

## JSON: Das Praktikums-Beispiel

```
{
  "Titel": "EU-27, Zahlen von 2010",
  "Spaltentitel": [
    "Ländername", "Fläche", "Einwohner", "Währung", "BIP", "BIP",
    "Staatsverschuldung", "Neuverschuldung", "Wachstum", "Arbeitslosigkeit" ],
  "Einheit": ["-", "km²", "Mio.", "-", "Mrd. Einh.", "Mrd. EUR", "% BIP", "%
    BIP", "%", "%" ],
  "Belgien": [32545, 10.8, "EUR", 352.3, 352.3, 96.8, 4.1, 2.1, 8.3],
  "Bulgarien": [110994, 7.6, "BGN", 70.5, 36, 16.2, 3.2, 0.2, 10.2],
  "Tschech. Rep.": [78866, 10.4, "CZK", 3670, 145.9, 38.5, 4.7, 2.4, 7.3],
  "Dänemark": [43098, 5.5, "DKK", 1746, 234.7, 43.6, 2.7, 2.1, 7.4],
  "Deutschland": [357093, 82.2, "EUR", 2499, 2499, 83.2, 3.3, 3.6, 7.1],
  "Estland": [45227, 1.3, "EUR", 14.5, 14.5, 6.6, 0.1, 3.1, 16.9],
  "Irland": [70273, 4.6, "EUR", 153.9, 153.9, 96.2, 32.4, -1, 13.7],
  (usw.)
  "Ver. Königreich": [242910, 62.0, "UKP", 1454, 1694, 80.0, 10.4, 1.3, 7.8]
}
```

**Ziel:** Diese Daten vom Web-Server laden und als HTML-Tabelle anzeigen

- jQuery-Unterstützung

- jQuery bietet nicht nur reine AJAX-Aufrufe an, sondern auch gleich vorbereitete Versionen davon speziell zum Laden von JSON-Daten
- Code-Beispiel (CS) aus dem Praktikum (Fragment):

```
$.getJSON '/eu-data.json'  
.done (data) ->  
    $.each data, (key, value) ->  
        # Ihr Code zum Verarbeiten der EU-Daten ...  
.fail (d, textStatus, error) ->  
    alert "getJSON gescheitert, Status: " +  
        textStatus + ", Fehler: " + error
```

- data: Hier erwarten wir ein JS-Objekt (key/value-Paare)
- \$.each: Wir besuchen jedes dieser Paare.
- „value“ ist entweder ein String oder ein Array von Strings bzw. Zahlen
- .done: Wird im Erfolgsfall aufgerufen, .fail: dito, im Fehlerfall





# Zusammenfassung

---

- Ajax ermöglicht das Ergänzen von Informationen vom Webserver ohne Nachladen der kompletten Seite
- Nachgeforderte Daten werden als
  - reiner Text, weitgehend unverändert eingefügt
    - ggf. enthaltener HTML-Code wird vom Browser interpretiert
  - strukturierte Informationen an Client zur dortigen Bearbeitung gesendet und dann entsprechend eingefügt
    - möglich z.B. mit XML- oder JSON-Dateien
- Gezieltes Nachladen zum Bedarfszeitpunkt und die dynamischen Elemente ermöglichen Desktop-ähnliches Verhalten
- Frameworks kennengelernt und Vor- und Nachteile betrachtet



# Literaturangabe

1. Mintert, Leisegang:  
„Ajax: Grundlagen, Frameworks und Praxislösungen“
2. Sowa, Radinger, Marinschek:  
„Google Web Toolkit“, dpunkt.verlag, 2008
3. Wenz:  
„JavaScript – Das umfassende Handbuch“,  
Galileo Computing, Galileo Press, 2010
4. Ackermann:  
„JavaScript – Das umfassende Handbuch“,  
Rheinwerk Verlag, Bonn, 2016
5. <https://www.json.org>
6. <https://jquery.com>
7. <https://w3schools.com> (Tutorials)

