Vorlesung

Cloud- und Web-Technologien

(CS3140)

Javascript und PHP

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Chronologische Übersicht über die Themen

Nr	Thema	
1	Einleitung	
2	Einführung in das Semantic Web, RDF und SPARQL	 Datenmodell
3	Die Semantic Web-Ontologiesprachen RDFS und OWL	
4	Multiplattform-Entwicklung mit Kotlin	 Multiplattform
5	Fortgeschrittene Themen mit Kotlin	
6	Einstieg in Cloud Computing, Hadoop	 Backend
7	Operatoren der relationalen Algebra in Hadoop	
8	Datenverarbeitung mit Pig	
9	Einführung in Spark und Flink	
10	Stromverarbeitung mit Flink	
11	Knotenzentrische Algorithmen mit Flink	
12	HTML und CSS	Web
13	**Browserprogrammierung mit JS/JQuery und Serverprogrammierung mit PHP Hypertext Preprocessor**	
14	Zusammenfassung und Ausblick	

JavaScript - Anwendungen

- Realisierung von client-seitig im Browser ablaufenden Web-Anwendungen
- Bedienoberflächen in dynamischen Web-Seiten, Animationseffekte
- Reaktion auf Ereignisse bei der Interaktion mit Web-Seiten
- Dynamische Erzeugung von Formularelementen, Eingabeüberprüfung

jQuery

- Eine der schnellsten JavaScript Bibliotheken (Open Source)
- Vereinfachung von
  - HTML-Navigation
  - Event-Handling
  - AJAX
  - Animationen
- Kompabilität
  - Cross-Browser
  - CSS3
- Achtung: teilweise nicht rückwärtskompatibel
- Umfangreicher Plugin-Support
- Lightweight footprint
  - 84,8 KB für Version 3.3.1 (minified, aber unkomprimiert)
- Meist verwendete JS-Bibliothek in über 70% der Webseiten*

* Usage of JavaScript libraries for websites

Interpretation von Sprachen

- Interpreter
  - Lesen und Ausführen eines Satzes (Programm) einer Sprache

	strikt interpretiert[1]	neuere Interpreter[2]
statische Semantik (& Prüfung dieser vor Ausführung)	Nein	Ja
Prüfung von nicht ausgeführten Programmteilen	Nein	Ja
Prüfung von Bindungs- und Typregeln vor Ausführung	Nein	Je nach Sprache
Generation einer expliziten internen Repräsentation des Satzes/Skripts	Nein	Ja

[1] z.B. Prolog und interpretiertes Lisp [2] z.B. JavaScript, PHP und Perl

  - Einfache Integration in andere Software (wie z.B. Web-Browser)
  - weitere Kapselung der Programmausführung
  - gegenüber der Ausführung von übersetztem Maschinencode zumeist 10-100 mal zeitaufwändiger

JavaScript - Eigenschaften

- Einfluss anderer Programmiersprachen
  - abgeleitet von Perl
  - C/C++/Java ähnliche Notation
  - aber wenig Bezug zu Java
- interpretiert
- **dynamisch** typisiert
- spezielle objektorientierte Konzepte
- Interpreter in Web-Browsern integriert
- eng verknüpft mit HTML
  - Zugriff auf Elemente des dargestellten Dokuments via DOM-API

PHP Hypertext Preprocessor

- Open Source
- Server-basierte Skriptsprache
- dokumentenzentrierte HTML Programmierung
  - Einbettung von PHP Code in HTML mit HTML als Ergebnis
  - aber auch General-Purpose Programmiersprache
- prozedurale Sprache mit objektorientierten Erweiterungen
- dynamisch typisiert mit wenigen einfachen Typen
- Notation orientiert sich an Perl und C
- große Funktionsbibliothek
  - Z.B. umfangreiche Unterstützung von Datenbanken
- über 80% von durch server-seitige Programmierspachen generierte Webseiten verwenden PHP*

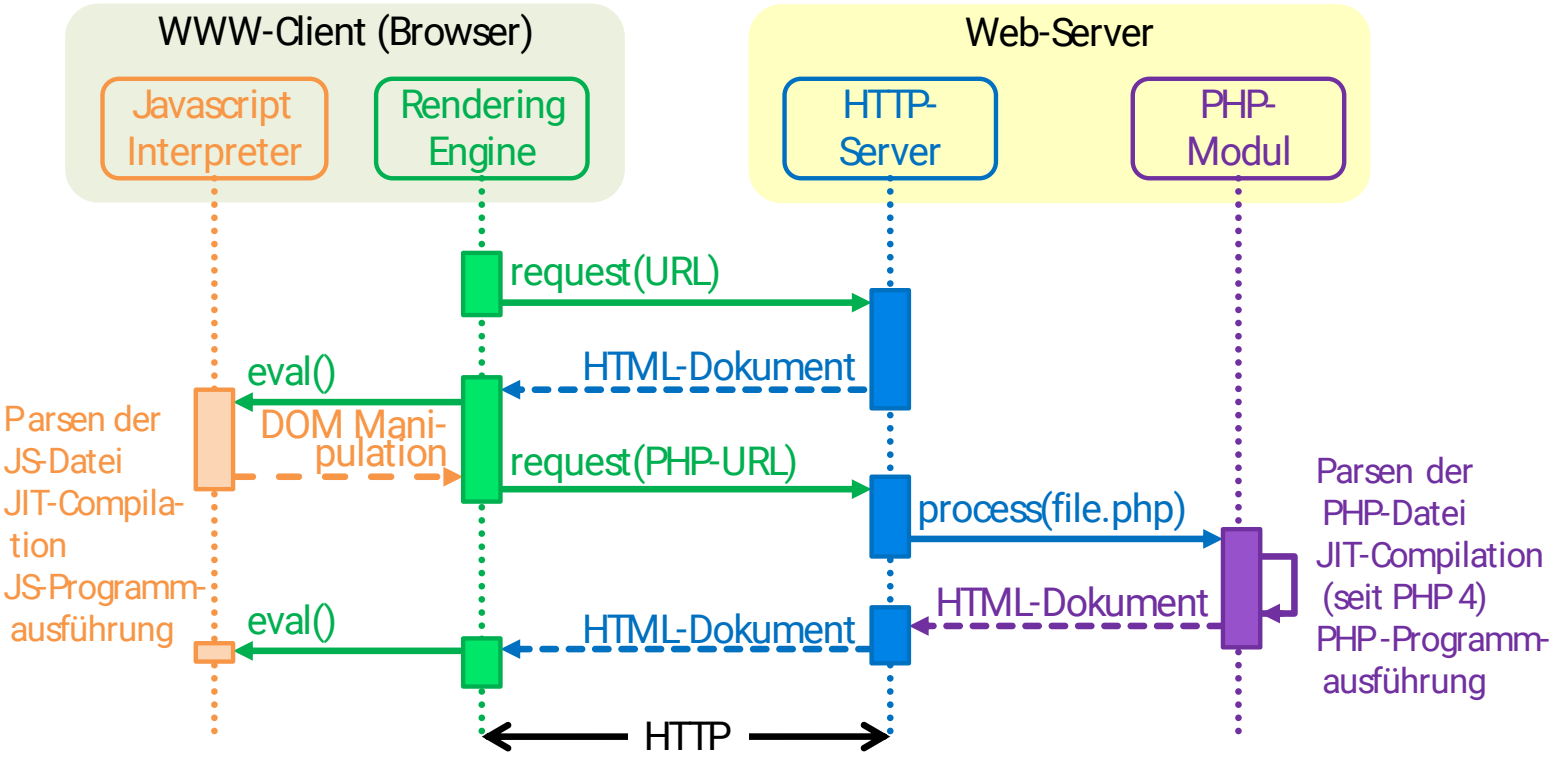
* W3Techs: Usage of server-side programming languages for websites

PHP - Anwendungen und Standardkonfiguration

- kleine private bis mittelgroße kommerzielle Projekte
- Schwerpunkt:
  - Generation von Webseiten basierend auf Daten in Datenbanken
- Beispielanwendungen
  - Content Management Systeme
    - Typo3, Wordpress, MediaWiki, Joomla!, Drupal, ...
- Standardkonfiguration mit freier Software (bis auf Windows)

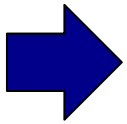
Komponente	LAMP	WAMP
Betriebssystem	Linux	Windows
Web Server	Apache Web Server	
Datenbankmanagementsystem	MySQL	
Serverseitige Skriptsprache	PHP	

Zusammenspiel von Browser & Server



Beispiel: Webanwendung mittels Javascript

```
<!DOCTYPE html>
<html>
 <head>
  <title>JavaScript Example</title>
  <script type="text/javascript">
   var number = Math.floor(Math.random()*6)+1;
   function check() {
    var guessed = document.GuessForm.inputfield.value;
    if(guessed==number){
     alert("Congratulations! The dice value is "
           + number + ".");
    } else {
     if(number<guessed) alert("My number is smaller!");
     else alert("My number is bigger!");
    }
   }
  </script>
 </head>
 <body>
  I have thrown a dice. Please guess the number:
  <form name="GuessForm">
   <input type="number" name="inputfield"
                size="1" min="1" max="6"/>
   <input type="button" value="Guess!" onClick="check()"/>
  </form>
 </body>
</html>
```



I have thrown a dice.
Please guess the number:

[ ] Guess!

Einbindung von JavaScript in HTML

1. Als Script-Bereich innerhalb eines HTML-Dokuments:

```
<script type="text/javascript">
… // your javascript code goes here…
</script>
```

2. Innerhalb von HTML-Tags (zur Definition von Bedienern, die bei bestimmten Ereignissen (Events) ausgelöst werden):
   a. Kodierung des Ereignisses als Attribut: Attributwert enthält eine Anweisungsfolge, die beim Eintritt des Ereignisses ausgeführt wird:

```
<input type="button" value="…" onClick="checkGuess()"/>
```

   b. Angabe einer Anweisungsfolge in einem Anker-Element (anstatt einer URL) mit `javascript:`, die beim Klicken ausgeführt wird:

```
<a href="javascript:checkGuess()">…</a>
```

3. In einer separaten Datei:

```
<script src="file.js" type="text/javascript"></script>
```

Bemerkungen zum Einbinden von JavaScript-Code

- Mehrfache Verwendung des `<script>`-Elementes möglich
- Sofortige Ausführung des JavaScript-Codes beim Einlesen des Dokumentes vom Browser
  - Keine sofortige Ausführung von Funktionsdefinitionen sowie Ereignisbedienern
- Angabe mehrerer Anweisungen in HTML-Attributen durch Semikola (;) getrennt möglich, aber unübersichtlich
- Ladezeitverringerung durch Platzierung von JavaScript-Bereichen **am Ende** des Dokumentes
- Separate Datei (ausschliesslich) mit JavaScript-Code: ASCII-Datei mit Dateinamenerweiterung `.js`

Einbinden von jQuery

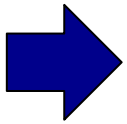
- Einbinden wie andere JavaScript-Dateien:

```
<script src="jquery-path" type="text/javascript"></script>
```

- Pfad zu der jQuery-Datei =
  - Pfad zu jQuery in Content Delivery Network (CDN):
    z.B. **Google CDN**
    oder:
  - lokale Datei auf dem eigenen Webserver nach
    Download von jQuery

PHP - Beispieleinbettung

```
<!DOCTYPE html>
<html>
  <head><title>Tree</title></head>
  <body>
    <pre>
      <?php
        $middle = 6;
        $width = 1;
        while ($middle >= 0) {
          $i=0;
          while($i<$middle){
            echo ' ';
            $i++;
          }
          $i=0;
          while($i<$width){
            echo '*';
            $i++;
          }
          $middle--;
          $width += 2;
          echo "\n";
        }
      ?>
    </pre>
  </body>
</html>
```



```
     *
    ***
   *****
  *******
 *********
***********
*************
```

PHP – Einbettung in HTML-Dokumente

- HTML-Anteile ohne PHP-Code werden nicht interpretiert und unverändert in das Ergebnis aufgenommen
- Interpretation nur von ausgezeichneten PHP-Code
- Syntaxalternativen der Auszeichnung
  - Standard-Tags: `<?php … ?>`
  - Sprachspezifische Script-Deklaration:
    `<script language="php"> … </script>`
  - ASP-Stil: `<% … %>`
  - Kurzschreibweise: `<? … ?>`
- Konfiguration im PHP-Modul notwendig bei Verwendung des ASP-Stils oder der Kurzschreibweise

Javascript - Grundlagen der Syntax

Bezeichner

```
identifier = {letter|"$"|"_"}{letter|"$"|"_"|digit}*
```

- einheitliche Schreibweise für alle Arten von Bezeichnern
- Unterscheidung von Klein-/Großschreibung (Case Sensitive)

Anweisungen

```
{ var k = 42; document.writeln (5*k); }
```

- **optionales** Semikolon am Zeilenende
- Semikolon erforderlich zwischen Anweisungen in derselben Zeile
- Eine Anweisungsfolge definiert keinen Scope:
  eine `var`-Deklaration gilt nicht nur in der Anweisungsfolge, sondern in umgebender Funktion bzw. im umgebenden Programm

Bedingte Anweisung

```
if (a < b) {min = a;} else {min = b;}
```

- Bei einzelnen Anweisungen sind die `{}`-Klammern optional

while-Schleife

```
i = 0; while (i < n) {document.write ("+"); i++;}
```

for-Schleife

```
for (i = 0; i < n; i++) {document.write ("+");}
```

return-Anweisung

```
return n*42;     return "+";
```

Parameterübergabe

- standardmäßig mittels **call-by-value**

Kommentare

- Einzeilig: `// Kommentar`
- Mehrzeilig: `/* Kommentar … */`

PHP - Grundlagen der Syntax

Bezeichner

```
identifier = [a-zA-Z_][a-zA-Z_0-9]*
```

- **Variablen**namen beginnen immer **mit** `$`
- **Konstanten- und Funktion**snamen **ohne** `$`
- Groß-/Kleinschreibung
  - bei Variablennamen (& Konstanten via `const`) unterschieden
  - bei Funktionsnamen (& Konstanten via `define`) **nicht** unterschieden

Anweisungen

```
<?php echo "Hello!";$i++; ?> ≡ <?php echo "Hello!";$i++ ?>
```

- Ende einer Anweisung bei **Semikolon** oder **schließendes Tag** `?>`

Bedingte Anweisung

```
if($a < $b) { $min = $a; } else { $min = $b; }
```

- Bei einzelnen Anweisungen sind die `{}`-Klammern optional

while-Schleife

```
$i=0; while($i<$width) { echo '*'; $i++; }
```

for-Schleife

```
for($i=0; $i<$width; $i++) { echo '*'; }
```

return-Anweisung

```
return $n*42;     return '+';
```

Parameterübergabe

- standardmäßig mittels **call-by-value**
- optional: **call-by-reference**
  - Übernahme der Änderungen des Wertes der Parametervariablen innerhalb der Funktion zu der Variablen des Funktionsaufrufes
  - Kennzeichnung in der Funktionsdefinition durch Notation von „&“

Kommentare

- Mehrzeilig: `/* Kommentar … */`
- Einzeilig: `// Kommentar`

Javascript - Variablen

- Variable kann Werte **beliebigen** Typs annehmen
- In der Regel Deklaration mit dem Schlüsselwort `var`
- Lokale Variable
  - Deklaration innerhalb einer Funktion
  - Geltungsbereich: nur innerhalb ihrer Funktion
- Globale Variable
  - Deklaration außerhalb von Funktionen
  - Deklaration innerhalb einer Funktion ohne Verwendung des Schlüsselwortes `var`: **Achtung**, Fehlerquelle!
  - Geltungsbereich: ganzes Programm
    - Ausnahme: Überdeckung von einer lokalen Variable
- Initialisierung durch Zuweisung eines Anfangswertes in Form eines Literals, einer anderen Variable oder einer komplexen Formel

Javascript - Markiere die Geltungsbereiche von Variablen


```
1  var number = Math.floor(Math.random()*6)+1;
2  function checkGuess() {
3   var guessed = document.GuessForm.inputfield.value;
4   function compare(number, guessed){
5     if(number == guessed){
6       return true;
7     } if(guessed < number){
8       compare = -1;
9     } else {
10      compare = 1;
11    }
12    return false;
13  }
14  isEqual = compare(number, guessed);
15  if(isEqual){
16   alert("Congratulations! The dice value is "+number+".");
17  } else {
18   if(compare > 0){
19    alert("My number is smaller!");
20   } else { alert("My number is bigger!");}
21  }
22 }
```

PHP - Variablen

- Definition durch Initialisierung
- Annahme von Werten beliebigen Typs
- Unterscheidung zwischen
  - lokalen Variablen
  - globalen und superglobalen (vordefinierte) Variablen
    - gelten im ganzen Programm
    - superglobale Variablen sind immer sichtbar
    - Definition von globalen Variablen
      - außerhalb des Bindungsbereiches einer Funktion
      - Verwendung des Schlüsselwortes `global` innerhalb des Bindungsbereiches einer Funktion
  - statischen Variablen
    - existieren nur im Bindungsbereich einer Funktion
    - Wert geht beim Verlassen dieses Bereichs nicht verloren

PHP - Illustration von Geltungsbereichen


```php
<?php
     $a = 1;
     function globaltest() {
      $a = 2;
      echo $a;
     }
     globaltest();
     echo $a;
?>
```

Ausgabe: ???

```php
<?php
     $a = 1;
     function globaltest() {
      global $a;
      $a = 2;
      echo $a;
     }
     globaltest();
     echo $a;
?>
```

Ausgabe: ???

Javascript - Primitive Datentypen

`number`

- **Keine** Unterscheidung (nach außen) zwischen
  - Ganzzahlen gespeichert in 4 Byte
  - Gleitpunktzahlen gespeichert in 8 Byte nach IEEE754
- Wert `NaN` (not a number): ein undefiniertes Ergebnis
- Wert `Infinity`: Wert, der größer als die größte repräsentierbare Zahl ist

`string`

- Zeichenkettenliterale mit einfachen oder doppelten Anführungszeichen
- Konkatenation wie in Java: `var s = "Hello " + 'world!';`
- Objektorientierte Notation für Zeichenkettenfunktionen
  Beispiele: `s.length`, `s.indexOf(substr)`, `s.charAt(i)`

`boolean`

- Literale: `true` und `false`
  (insbesondere **nicht**: True bzw. False)
- Operatoren: Konjunktion `&&`, Disjunktion `||`, Negation `!`

PHP - Primitive Datentypen

`integer`
`float`

- Dezimale, hexadezimale oder oktale Notation von Ganzzahlen
- Bei Überlauf Konvertierung nach `float`

`string`

- Zeichenkettenliterale mit
  - einfachen Anführungszeichen
    - Alle Zeichen stehen für sich selbst (nur `'` muss „escaped“ werden: `\'`)
  - doppelten Anführungszeichen
    - Parsen des Strings und Ersetzen von vorkommenden Variablen und Escape-Folgen, Bsp.: `"Sum of $year = \t$sum EUR"`
- Konkatenation mit Punkt: `"Hello" . "world!"`
- Ausgabe durch `echo` (von durch Kommata getrennte Ausdrücke) oder `print` (1 Parameter)

`boolean`

- Literale: `true` und `false` (keine Unterscheidung von Groß-/Kleinschreibung)
- Operatoren: Konjunktion `&&` bzw. `and` (kleinere Präzedenz), Disjunktion `||` bzw. `or` (k.P.), Negation `!`, Exklusiv-Oder `xor`

Javascript - Datentypen: Arrays

Datentyp	Jedes Feld (Array) ist vom Datentyp `Array`
Bedeutung	Abbildung von Indizes auf Werte
Einträge	Paare (k, v) mit k numerischer oder String-Schlüssel und zugeordneter Wert v

Erzeugung von Arrays mit dem Konstruktor `Array( )`

Liste von Werten

```
// indiziert von 0 an:
monatsName = new Array("", "Jan", …, "Dez");
```

Erweiterung eines leeren Arrays

```
monatsName = new Array();
monatsName[1]= "Jan"; monatsName[2]= "Feb"; …
```

Assoziatives Array

```
monatsNr = new Array();
monatsNr["Jan"] = 1; monatsNr["Feb"] = 2; …
```

- fügt Eigenschaften `"Jan"`, `"Feb"` etc. zum Objekt `monatsNr`
- „Ass. Array“ mittels Objektliteral: `monatsNr = {jan:1,feb:2,…};`

Zugriff

Aufzählung aller Elemente

```
for(n in monatsNr) { console.log(n+"=>"+monatsNr[n]); }
```

PHP - Datentypen: Arrays

Bedeutung	Abbildung von Indizes auf Werte
Einträge	Paare (k, v) mit k ganzzahl. (evtl. von anderen primitiven Typen konvertiert) oder String-Schlüssel und zugeordneter Wert v
	Erzeugung von Arrays
Liste von Werten	`// indiziert von 0 an:` `$monthName = array("", "Jan", "Feb", "Mar");`
Liste von Index-Wert-Paaren	`$monthName = array(1 => "Jan", 2 => "Feb", 3 => "Mar");`
Explizite Indizierung	`$monthName[1]= "Jan"; $monthName[2]= "Feb"; …`
Assoziatives Array	`$monthName = array("Jan" => 1, "Feb" => 2, "Mar" => 3);`
	Zugriff
Aufzählung aller Elemente	`foreach($monthName as $k => $v){ echo $k." => ".$v." "; }`

Javascript - Funktionsdatentyp

- Funktionen sind Objekte vom Typ `Function`

	Erzeugung von Funktionen
function-Statement	`function f(x, y) { return x * y; }`

Funktionsliteral	`// anonym:`
	`var p = function (x, y)  { return x * y; };`
	`// benannt:`
	`var p = function fn(x, y) { return x * y; };`

Konstruktor Function()	`var p = new Function("x", "y", "return x * y;");`

Javascript - Objekt-Datentyp

Datentyp	Alle Objekte sind vom Datentyp `Object`
Bestandteile	Komponenten, die jeweils Namen und Wert besitzen
	Erzeugung
Objektliteral	`var student = { matrNr:4711, name:"P. Smell" };`
Standardkonstr. `Object()`	`var student = new Object();`

Benutzer-definierter Konstruktor `Object(…)`	`function MyStudent(m, n) { // Nameskonvention: Groß!`
	`  this.matrNr = m; // this = aktuelle Objektinstanz`
	`  this.name = n;`
	`}`
	`var student = new MyStudent(4711,"P. Smell");`

	Hinzufügen von weiteren/Abändern bestehender Komponenten
Zuweisung	`student.semester = 5; student.matrNr = 4711;`
	Zugriff auf Komponenten
Pfadausdruck	mittles *Objekt.Komponente*: `student.matrNr` liefert `4711`

Javascript - Methoden

- Funktionen als Objektkomponenten heißen Methoden
  - Bei Aufruf i.d.R. Lesen und Verändern der Objektkomponenten
- Alle andersartigen Komponenten heißen Eigenschaften

Definition

```
function MyCircle(r) {
 this.radius=r;
 this.area=getArea; // Zuweisen von globaler Fkt. getArea
 this.circ=function() {return (Math.PI*this.radius*2);};
}
function getArea() {
 return (Math.PI * this.radius * this.radius);
}
```

Aufruf

```
c = new MyCircle(3);
document.writeln("Area = " + c.area());
document.writeln("Circumference = " + c.circ());}
```

Javascript - Klassen und ECMAScript 6

- ECMA standardisiert ECMAScript (Name des offiziellen Standards)
  - JavaScript ist dessen meistbekannteste Implementation
- Weitergehende objektorient. Sprachkonstrukte ab ECMAScript 6
  - Wird **nicht** von jedem Browser unterstützt
  - Alte und neue Sprachkonstrukte parallel verwendbar

```
class Shape {
  constructor(x_pos,y_pos) {
    this.x = x_pos;
    this.y = y_pos;
  }
}
class MyCircle extends Shape {
  constructor(x_pos,y_pos,r) {
    super(x_pos,y_pos);
    this.radius=r;
  }
  getArea() {
    return (Math.PI * this.radius * this.radius);
  }
}
var c = new MyCircle(3);
console.log(c.getArea());
```

PHP - Objektorientierte Programmierung

- Vergleichbare Mächtigkeit von OOP-Sprachkonstrukten von PHP zu anderen OO-Programmiersprachen
- Details differieren

```php
abstract class Mammal {
  public function getNumberOfLegs() {
    return $this -> legs; // Achtung: Zugriff auf Variable der Kindsklasse
  }
}
class Dog extends Mammal {
  protected $legs = 4;
}
$waldi = new Dog;
echo $waldi -> getNumberOfLegs() . " legs";
```

PHP - Funktionsbibliothek

- ausgereift
- > 700 Funktionen
- Beispiele der Abdeckung der PHP-Funktionsbibliothek:
  - Arrays
  - Protokolle
  - Datenbanken
  - Datum/Uhrzeit
  - Dateiverzeichnisse
  - Dateien
  - Grafik
  - HTTP
  - IMAP
  - LDAP
  - Mathematik
  - MCAL
  - Mcrypt
  - Mhash
  - PDF
  - POSIX
  - reguläre Ausdrücke
  - Strings
  - Variablenmanipulation
  - XML

Javascript - Funktionsbibliothek

- Implementation des Großteils der Funktionsbibliothek in Form von Objekten
- ergänzt durch vordefinierte Funktionen

Art	Beschreibung	Beispiele
Kern-objekte	Funktionalität unabhängig von Browser und Dokument	`Array`, `Date`, `Function`, `Math`, `Object`, `RegExp`
DOM-Objekte	Repräsentation von und Zugriff auf Dokumente. Standardisierte DOM-Objekte nach der W3C Spezifikation mit Zugriff über DOM-API.	`Document`, `Element`, `Node`
Browser-Objekte	Repräsentation von und Zugriff auf Browser-Fenster	`History`, `Location`, `Navigator`, `Screen`, `Window`
Vordefi-nierte Funktionen	Funktionalität unabhängig von Browser und Dokument	`eval`, `isFinite`, `isNaN`, `parseInt`

Javascript - Funktionsbibliothek

Objekt	Beschreibung
`window`	• höchstes Objekt und Ausgangspunkt der DOM- und Browser-Objekte-Hierarchie
`document`	• Abkürzung für `window.document` • Wurzelknoten des HTML/XML-Dokuments • oberstes Ausgangsobjekt für das Document Object Model (DOM)
`documentElement`	• Abkürzung für `window.document.documentElement` • Wurzelelement des HTML/XML-Dokuments

- Wert einer Eigenschaft kann wiederum ein Objekt sein
  $\Rightarrow$ Mittels Pfadausdruck Zugriff auf tief inneliegende verschachtelte Eigenschaften: `window.document.images.length`
- verbindliche Standards in HTML 5 für Browser-Objekte
- Alternative Bezeichnungen für Kernobjekte (Global Objects, Built-in Objects, Objects in the Global Scope) und vordef. Funktionen (Global Functions, Built-in Functions)

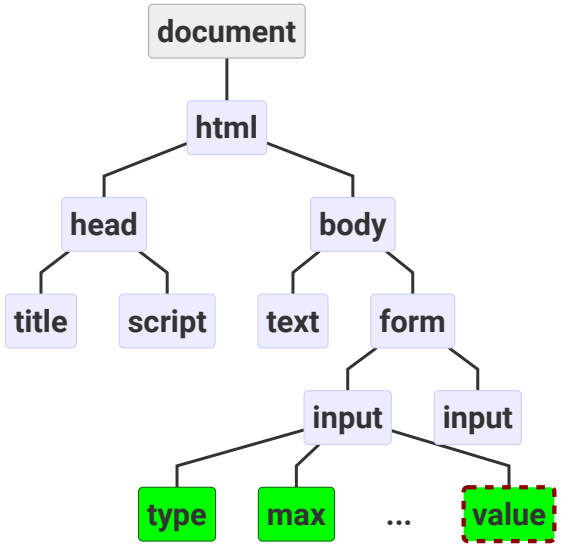
Javascript - Funktionsbibliothek: Zugriff auf HTML-Elementobjekte und deren Komponenten

- Eindeutig qualifizierender Name gemäß DOM-Dokumentbaum
- DOM-API, um Elementmenge auszuwählen
  - `getElementsByName()`
  - `getElementById()`
  - `getElementsByTagname()`
  - `querySelector()` / `querySelectorAll()` (mittels CSS-Selektor)
  - …
- Vordefinierte Arrays, die alle Elemente eines Typs enthalten
  - `document.forms`
  - `document.images`
  - …
- Kombiniert: Auswahl von Elementmenge plus weitere Spezialisierung mittels qualifizierendem Namen

Javascript - Fkt.-bib.: DOM-Objekte – Bsp.

I have thrown a dice. Please guess the number:
3 Guess!

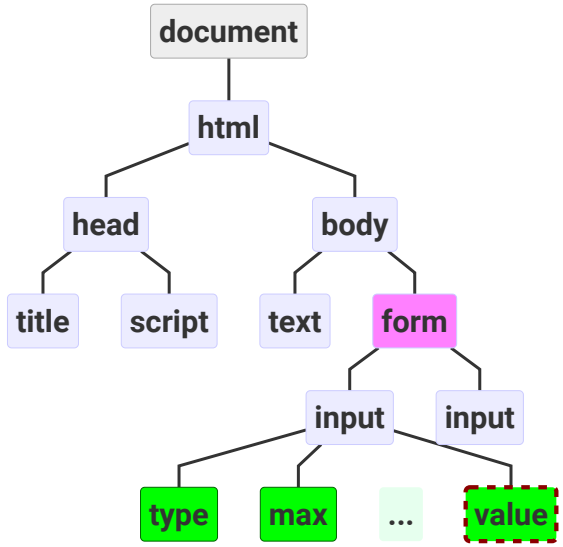
```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Example</title>
  <script type="text/javascript">
    var number = Math.floor(Math.random()*6)+1;
    function checkGuess() { … }
  </script>
</head>
<body>
  I have thrown a dice. Please guess the number:
  <form name="GuessForm" id="GF">
    <input type="number"
           name="inputfield"
           size="1" min="1" max="6"/>
    <input type="button" value="Guess!"
           onClick="checkGuess()"/>
  </form>
</body>
</html>
```



Javascript - Funktionsbibliothek: DOM-Objekte – Bsp.

I have thrown a dice. Please guess the number:
3 Guess!

```
…
<form name="GuessForm" id="GF">
  <input type="number" name="inputfield" …/>
  …
</form>
…
```



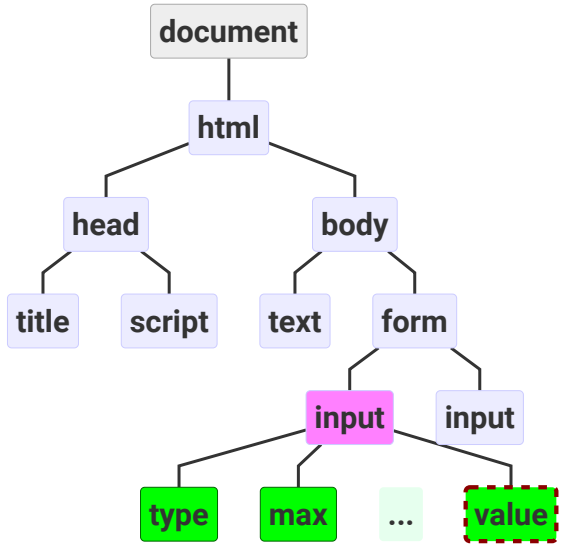
Zugriffsmöglichkeiten auf das (erste) Formelement:

`document.GuessForm`	qualifizierender Name
`document.getElementsByTagName("form")[0]`	DOM-API
`document.getElementById("GF")`	DOM-API

Javascript - Funktionsbibliothek: DOM-Objekte – Bsp.

I have thrown a dice. Please guess the number:
3 Guess!

```
…
<form name="GuessForm" id="GF">
  <input type="number" name="inputfield" …/>
  …
</form>
…
```



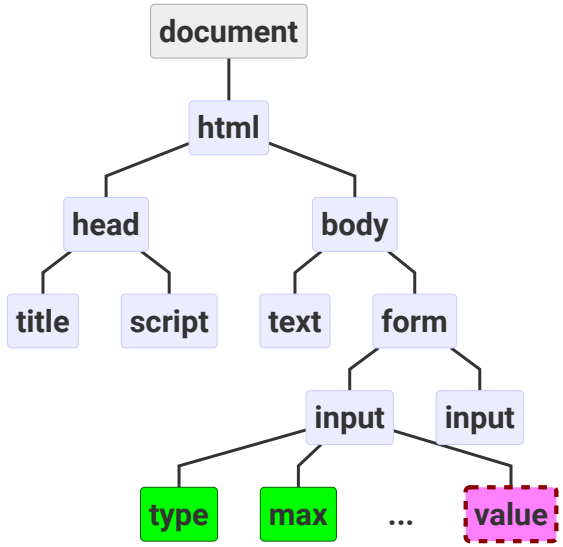
Zugriffsmöglichkeiten auf das (erste) Eingabeelement:

`document.GuessForm.inputfield`	qualifizierender Name
`document.getElementsByTagName("form")[0].inputfield`	DOM-API
`document.getElementsByName("inputfield")[0]`	DOM-API

Javascript - Funktionsbibliothek: DOM-Objekte – Bsp.

I have thrown a dice. Please guess the number:
3 Guess!

```
…
<form name="GuessForm" id="GF">
  <input type="number" name="inputfield" …/>
  …
</form>
…
```



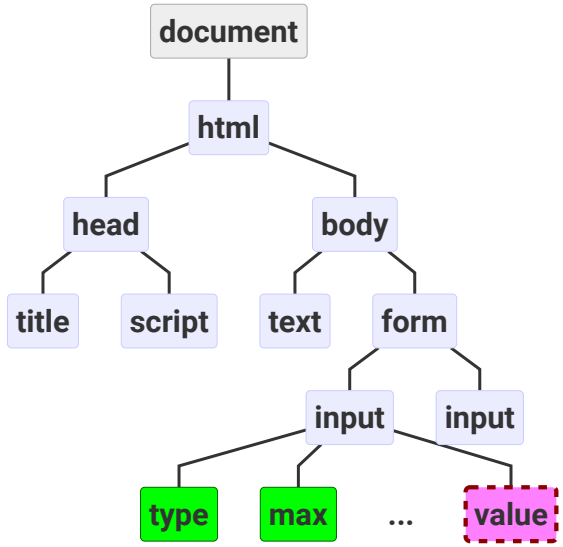
Zugriffsmöglichkeiten auf das Eingabefeld im ersten Eingabeelement:

```
document.GuessForm.inputfield.value
document.getElementsByTagName("form")[0].inputfield.value
document.getElementsByName("inputfield")[0].value
```

Javascript - Funktionsbibliothek: DOM-Objekte – Bsp.

I have thrown a dice. Please guess the number:
3 Guess!

```
…
<form name="GuessForm" id="GF">
  <input type="number" name="inputfield" …/>
  …
</form>
…
```



Zugriffsmöglichkeiten mittels CSS-Selektoren:

```
document.querySelector("[name=inputfield]").value
document.querySelectorAll("form input")[0].value
document.querySelector("form#GF > input[name=inputfield]").value
```

Javascript - DOM-Objekte

- Zugriffe erfolgen gemäß der DOM-API
  - über Methoden des generischen `Node`-Interfaces wie `getChildNodes()`
  - über Methoden des `Document`-Interfaces wie `getElementsByName()`, `getElementsByTagName()` oder `getElementById()`
- Jedes erlaubte Attribut eines HTML-Elements: Eigenschaft im entsprechenden DOM-Objekt
  - HTML-Element `<input>` hat ein erlaubtes Attribut value
    $\Rightarrow$ DOM-Hierarchie enthält `input`-Elementobjekt mit der Eigenschaft `value`
- Setzen von Werten analog:
  document.GuessForm.Eingabe.value = 2;
- DOM-Methoden für einige der Element-Objekte
  - Z.B. `submit()` und `reset()` im DOM-Objekt des HTML-Elements `<form>`

Javascript - Ereignisbehandlung

- Ereignis (Event): Wahrnehmung einer Zustandsänderung
  - Ereignisgetriebene Programmierung ordnet Ereignissen Operationen zu
  - Setzen von Bedienern für ein Ereignis:

```
<!DOCTYPE html>
<html>
<head><title>Event</title></head>
<body>
  <form name="eventForm">
    <input type="button" value="Click me!"
           onclick="alert('ping!');">
    <input type="button" value="Click me, too!"
           name="pongButton">
  </form>
  <script type="text/javascript">
  document.eventForm.pongButton.onclick =
    function(){ alert("pong!") };
  </script>
</body>
</html>
```

Javascript - Ereignisbehandlung

Event-Handler	HTML-Elemente	Semantik
`onclick`	Knopf, Checkbox, Anker	Element wird angeklickt
`onchange`	Textfeld, Textbereich, Auswahl	Wert wird geändert
`onkeydown` `onkeyup` `onkeypress`	Dokument, Bild, Anker, Textfeld	Taste gedrückt/losgelassen
`onload`	Body	Nach dem Laden eines Dokuments
`onmousedown` `onmouseup`	Dokument, Knopf, Anker	Maustaste gedrückt/losgelassen
`onmouseout`	Bereiche, Anker	Mauszeiger verlässt einen Bereich
`onmouseover`	Anker	Mauszeiger über Anker
`onreset` `onsubmit`	Formular	Reset/Submit für ein Formular
`onselect`	Textfeld, Textbereich	Element wird ausgewählt
`onfocus` `onblur`	Fenster, alle Formularelemente	Eingabefokus wird dem Element gegeben/entzogen

- Ereignisbehandlung (z.B. von Maus-Ereignissen) weitgehend auch mittels **CSS** realisierbar

Superglobale Variablen (vordefinierte Arrays)

Superglobale Variable	Bemerkung
`$GLOBALS`	(Name, Wert)-Paare von globalen Variablen
`$_SERVER`	Informationen über Server und Ausführungsumgebung
`$_GET`	HTTP GET-Variablen
`$_POST`	HTTP POST-Variablen
`$_FILES`	HTTP Dateiupload-Variablen
`$_COOKIE`	HTTP Cookies
`$_SESSION`	Sessionvariablen
`$_REQUEST`	Ein assoziatives Array, das standardmäßig den Inhalt von `$_GET`, `$_POST` und `$_COOKIE` enthält
`$_ENV`	Umgebungsvariablen

- Vordefinierte Variablen können **nicht** als variable Variablen verwendet werden

Unterstützung von Sitzungen (Sessions)

- Möglichkeit, Daten während einer Folge von Aufrufen einer Website festzuhalten
  (bis der Browser geschlossen wird)
  - Session-Daten werden in `$_SESSION` gespeichert
  - Zuordnung einer eindeutigen Session-ID zu einem Besucher beim Aufruf der Website
  - Bei Aufruf der Website prüft PHP je nach Konfiguration automatisch oder durch Aufruf von `session_start()`, ob eine Session-ID mitgesendet wurde

    $\Rightarrow$ Wiederherstellung einer zuvor gespeicherten Session-Umgebung

Beispiel zu Sessions: Login-Skript

HTML-Seite für den Login

```
<form action="login.php" method="post">
  User Name:<br/>
  <input type="text" size="24" maxlength="50" name="name"/><br/><br/>
  Password:<br/>
  <input type="password" size="24" maxlength="50" name="passwd"/><br/>
  <input type="submit" value="Login"/>
</form>
```

User Name:
groppe
Password:
••••••••••
Login

login.php

```
<?php
session_start();
$name = $_POST["name"];
$passwd = $_POST["passwd"];
$hash = md5($passwd);
// following condition can be replaced
// with database requests or file lookup
if($name == 'Peter' && $hash == 'XYZ'){
  $_SESSION['username'] = $name;
  echo "Login successful. <a href=\"secret.php\">Secret area</a>";
} else { echo "Access Denied <a href=\"login.html\">Back</a>"; }
?>
```

Erfolgreiche Anmeldung:

Login successful.
Secret area

Falscher Benutzer/Passwort:

Access Denied Back

Anmerkungen zum Login-Skript

- Kommunikation zu `login.php` und nachfolgenden Seiten sollte gesichert über HTTPS geschehen
  - Ansonsten Mitlesen der Session-ID und damit Übernahme der Session möglich
- Auf nachfolgenden PHP-Seiten (wie `secret.php`) muss jeweils überprüft werden, ob `$_SESSION['username']` gesetzt ist
  - Ansonsten ist der Benutzer nicht eingeloggt!

Quellen zum Nachlernen und Nachschlagen im Web

- Javascript
  - ECMA, Standard ECMA-262: ECMAScript Language Specification
  - MDN, JavaScript
  - SELFHTML e.V. JavaScript
  - Tarquin, JavaScript Tutorial
  - W3 Schools, JavaScript Tutorial
- PHP
  - PHP Dokumentation (The PHP Group)
  - SELFPHP (SELFPHP OHG)
  - PHP Tutorial (W3 Schools)
  - PHP 101: PHP For the Absolute Beginner (Zend)
  - Reguläre Sprachen, reguläre Ausdrücke (Leibniz-Rechenzentrum)

Zusammenfassung JQuery & PHP

- Javascript
  - Interpreter von Sprachen
  - Einführung in JavaScript
  - Eigenschaften, Anwendungen und Sprachkonstrukte (wie Variablen, Datentypen)
  - DOM-API und Ereignisbehandlung
- PHP
  - Hauptanwendung: Generation von Webseiten basierend auf Daten in Datenbanken
  - Dafür ausgelegte Sprachkonzepte & Funktionsbibliothek
  - Meist verwendete Serverseitige Programmiersprache
  - Standard-Webserver-Architektur: **L**(W)**AMP = L**inux(Win)+**A**pache Webserver+**M**ySQL+**P**HP
  - Einführung in imperative Sprachkonstrukte von PHP
  - Sessions