

# 8. Web-Skriptsprachen

8.1 Clientseitige Web-Skripte: JavaScript



8.2 Dokument-Objekte und DOM

8.3 Objektorientierung in JavaScript

8.4 Serverseitige Web-Skripte (Prinzipien)

Literatur:

Stefan Koch: JavaScript: Einführung, Programmierung und Referenz , dpunkt Verlag, 6. Auflage 2011

Marijn Haverbeke: Die Kunst der JavaScript-Programmierung, dpunkt Verlag, 2012

<http://de.selfhtml.org/>

# Gliederung (neu)

1. Grundbegriffe
2. Digitale Codierung und Übertragung
3. Zeichen und Schrift
4. Signalverarbeitung
5. Ton und Klang
6. Licht, Farbe und Bilder
7. Bewegte Bilder
- 8. Web-Skriptsprachen**
9. Web-Dokumente
10. Computergrafik (Einführung)
11. Varianten und Weiterentwicklungen bei Bild/Video

# Skriptsprachen

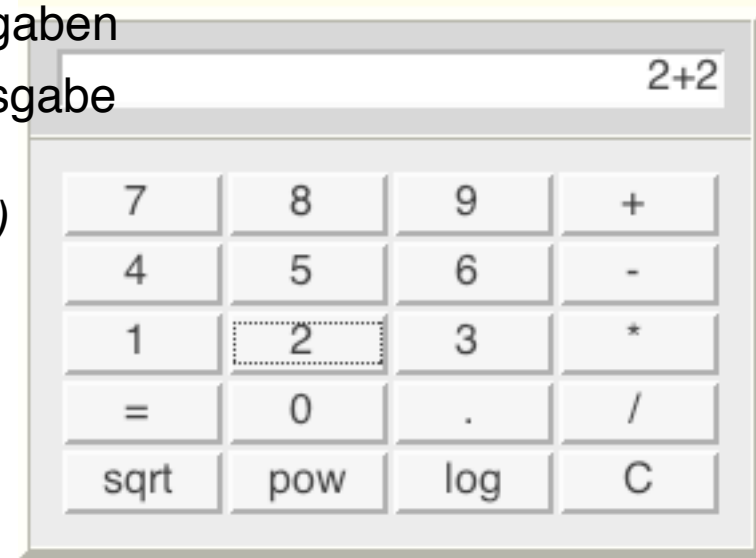
- Sprache zur Programmierung von Abläufen in Computersystemen
- Enge Integration mit Betriebssystem oder speziellem Anwendungssystem
- Meist interpretiert, leicht zur Laufzeit zu definieren und zu ändern
- Moderne Skriptsprachen durchaus Alternative zu Programmiersprachen
- Beispiele:
  - Betriebssystem-Skripte: Unix Shells, DOS Batch-Dateien, AppleScript
  - Clientseitige Web-Skripte: JavaScript, VBScript
  - Serverseitige Web-Skripte: PHP
  - Skripte für Multimedia-Player: Flash ActionScript
  - Universelle Skripte: Perl, Python, Ruby, TCL

# Was ist JavaScript?

- Schlanke Programmiersprache zur integrierten Ausführung in Web-Browsern (und -Servern)
  - interpretiert
  - lokale Ausführung
  - objektbasiert (nicht echt objektorientiert, z.B. keine Klassen/Vererbung)
  - schwach typisiert
  - dynamisch gebunden
  - relativ sicher (kein Zugriff auf lokales Dateisystem und Betriebssystem)
- JavaScript hat außer einer gewissen Syntaxähnlichkeit keine Beziehung zu Java! (Originalname: "LiveScript")
- Geschichte:
  - Entwickelt von Netscape 1995 (ab Browserversion 2)
  - Unterstützung in Microsoft Internet-Explorer ab Version 3 ("JScript")
  - Standardisiert als ECMAScript (ECMA-262) (European Computer Manufacturers Association) bzw. als ISO-10262
  - Moderne Browser weitgehend kompatibel zum ECMA-Standard (Edition 3)
  - ECMAScript Edition 4 wurde aufgegeben, Edition 5 wird weiterentwickelt

# JavaScript: Funktionsumfang und Anwendungsbereich

- Beispiele für sinnvolle Anwendung von JavaScript:
  - Formulareingaben auf Plausibilität prüfen
  - Spezialitäten verschiedener Browser-Plattformen flexibel unterstützen
  - Client-seitige Voraussetzungen prüfen  
z.B. zum Abspielen einer Multimedia-Datei
- Funktionsumfang:
  - Klassische Funktionen für Arithmetik und Zeichenreihenverarbeitung
  - Verarbeitung von Maus- und Tastatureingaben
  - Dynamische Erzeugung von (HTML-)Ausgabe
  - Zugriff auf Dokument-Struktur  
über das *Document Object Model (DOM)*
- Gestiegene Bedeutung:
  - HTML5+JavaScript
  - WebGL



# Einbettung von JavaScript in HTML

```
<h1>
<!-- Script-Markup -->
  <script type="text/javascript">
    document.write("Hello World!");
  </script>
</h1>
<!-- Externe Datei -->
<h2>
  <script type="text/javascript" src="hello.js"></script>
</h2>
<!-- URI -->
<h2>
  <a href="javascript:alert('Hallo');">Hallo sagen</a>
</h2>
<!-- Eventhandler -->
<h2 onClick="confirm('Halli');">
  Hier klicken...
</h2>
```

# JavaScript-Beispiele ausführen

- JavaScript wird *interpretiert* (nicht übersetzt/compiliert)
- Der Interpreter für Programme ist in Browser integriert
  - “Clientseitig interaktive Webseiten”
  - HTML-Seite mit eingebettetem JavaScript
- Fehlersuche (“Debugging”):
  - Mit Hilfe der JavaScript-Konsole (console)
  - Mit Hilfe spezieller Plug-ins
    - » z.B. “Firebug” für Firefox
- Diverse Spielarten:
  - z.B. eigenständige JavaScript-Engines
  - z.B. serverseitiges JavaScript (node.js)

# JavaScript: Elementare Namenskonventionen

- Variablennamen beginnen mit Buchstaben, Dollar oder Unterstrich
- Groß- und Kleinschreibung wird unterschieden
  - Häufig benutzt (“Camel Case”): *nameAusMehrerenWorten*
- Kommentarzeilen:
  - beginnen mit `//` oder werden in `/* . . . */` eingeschlossen
  - `<!--` ist ein spezieller einzeiliger Kommentar in JavaScript.
- *Ähnlich zu, aber nicht identisch mit Java-Syntax!*
  - Viele Schlüsselwörter und Sprachkonstrukte identisch (z.B. `if`, `while`, `for`)
  - Neue Schlüsselwörter im Vergleich zu Java (z.B. `var`, `function`)
- Grundsätzlich anderer Zugang zu Datentypen:
  - Datentyp einer Variable muss nicht explizit deklariert werden



# Skripte und Kommentare

- Für Browser, die die Skriptsprache JavaScript nicht erkennen:
  - JavaScript in HTML-Kommentar einschließen
  - Spezieller einzeiliger JavaScript-Kommentar `<!--`
  - HTML-Kommentarzeichen für JavaScript auskommentieren

- Beispiel:

```
<script type="text/javascript">
  <!--
    document.write("Hello World!");
  // -->
</script>

<noscript>
  <!-- Meldung falls Skript nicht unterstützt. -->
  <i>Bitte möglichst JavaScript
    einschalten, danke.</i>
</noscript>
```

# Programm-Beispiel: Fibonacci-Funktion

```
<script type="text/javascript">
```

```
function fib(n){  
  if (n==0)  
    return 0;  
  else  
    if (n==1)  
      return 1;  
    else  
      return (fib(n-1)+fib(n-2));  
}
```

```
document.writeln("fib(3) =" + fib(3) + "<br>");
```

```
document.writeln("fib(8) =" + fib(8) + "<br>");
```

```
</script>
```

fibonacci1.html

# Einfache Datentypen in JavaScript

- Zahlen (Ganzzahlen, Gleitkommazahlen)
  - Dezimale Ganzzahlen: `0`, `22`, `-1000`
  - Oktalzahlen mit `0` beginnend: `026` (= dezimal 22)
  - Hexadezimalzahlen mit `0x` beginnend: `0x16` (= dezimal 22)
  - Fließkommazahlen: `33.333`, `123.`, `6.24e-12`
- Zeichenketten
  - Beliebig lange Folgen von Zeichen
  - Literale Angabe: Wahlweise in *einfachen oder doppelten* Anführungszeichen
  - Sonderzeichen `\b`, `\n`, `\t`, ...
- Wahrheitswerte
  - `true`, `false`
  - Operationen `!`, `&&`, `||`
- Sonderwerte
  - `undefined`: uninitialized variable, function not returning a value
  - `null`: special object of type null
  - `NaN`: “not a number” - non-numeric value in numeric context

# Schwache Typisierung

- Jede Variable und jeder Funktionsparameter kann uneingeschränkt Werte eines jeden in JavaScript bekannten Datentyps annehmen:
  - Zahl (Ganzzahl, Fließkomma)
  - Zeichenreihe
  - Wahrheitswert
  - Array
  - Objekt
  - Funktion (!)
- Ergebnisse von Funktionen werden mit `return` übergeben; ebenfalls keine Typdeklaration
- Variablendeklaration:
  - explizit (empfohlen!): `var i; var i = 1;`
  - implizit bei Verwendung: `i = 12;`
- Abfrage des aktuell zugewiesenen Datentyps:
  - `typeof v`

# Achtung automatische Typumwandlung!

- Ungewöhnliche Beispiele, liefern alle "true":
  - `false == 0;`
  - `"" == 0;`
  - `"5" == 5;`
- Vergleich ohne Typumwandlung mit "===" (bzw. "!==");
  - `false === 0;`
  - `"" === 0;`
  - `"5" === 5;`
- Was liefern wohl diese Beispiele?
  - `"Apollo"+5;`
  - `null + "ify";`
  - `"5" * 5;`
  - `"strawberry" * 5;`

# Objekte und Eigenschaften in JavaScript

- Eigenschaften (*properties*) für Werte:  
Paare aus (Schlüssel – zugeordneter Wert)
  - Beispiel: Länge einer Zeichenkette `s`
  - Äquivalente Schreibweisen: `s["length"]` und `s.length`
- Datentyp `object` in JavaScript:
  - Träger beliebiger Eigenschaften
  - Eigenschaften können hinzugefügt, entfernt und im Wert verändert werden

- Beispiel:

```
var cat = {"color": "grey", "name": "Spot"};  
cat.size = 47;  
cat.name = "Spottie";  
delete cat.size;
```

- Was liefert wohl das folgende Beispiel?

```
var prop1 = "name";  
var prop2 = "length";  
document.writeln(cat[prop1][prop2]);
```

[properties.html](#)

# Arrays in JavaScript

- Arrays sind spezielle Objekte:
  - Vordefinierte Eigenschaft `length` (und diverse Methoden)
  - Eigenschafts-Schlüssel sind natürliche Zahlwerte (ab 0)
- Erzeugen von Arrays:
  - Alternativ `var a = new Array(1, 2, 3, 4);`
  - oder `var a = [1, 2, 3, 4];`
- Enthaltene Werte in einem Array müssen nicht vom gleichen Typ sein!
- Beispiel:

```
function show(a) {  
    document.writeln(a+"<br>");  
    document.writeln("<hr>");  
}  
var a = [1, 2, 3, 4]; show(a);  
a[2] = "drei"; a[3] = 4.01; show(a)
```

arrays.html

# Zeichenreihen (Strings)

- Viele vordefinierte Eigenschaften und Funktionen, z.B.:
  - `length`: Länge der Zeichenreihe
  - `concat`: Verkettung von Zeichenreihen
  - `indexOf`: Position einer Teilzeichenreihe
  - `substring`: Ausschneiden einer Teilzeichenreihe
  - `search`, `match`, `replace`: Suchen und Ersetzen von Teilzeichenreihen, die über *reguläre Ausdrücke* spezifiziert sind (z.B. `/dm.* /`)
- Aufruf in “objektorientiertem” Stil: *Objekt . Funktion*
- Detaillierteres Beispiel:
  - `split(begrenzer)`: Teilt Zeichenreihe in ein Array von Teilzeichenreihen gemäß dem Trennzeichen *begrenzer*

```
s = ("Fritz;Eva;Franz;Maria");  
a = s.split(";");  
ergibt  
a = ["Fritz", "Eva", "Franz", "Maria"]
```



# Ablaufstrukturen in JavaScript

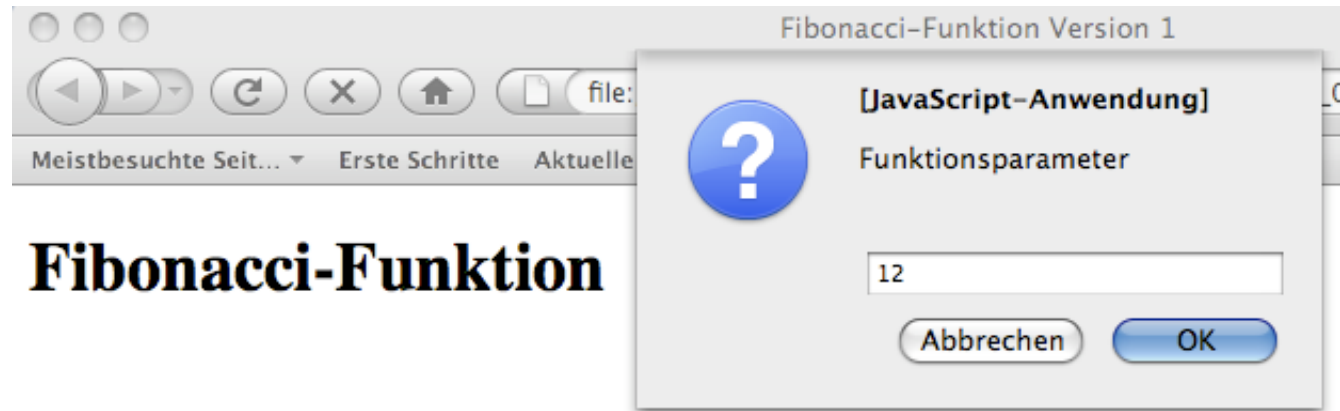
- Ablaufsteuerung ist analog zu Java-Syntax und Semantik, z.B.:
  - if/else
  - for
  - while
  - switch
  - return
  - break
  - continue

# JavaScript-Funktionen für modale Dialoge

- Dialogtypen:
  - *modal*: System wartet auf Antwort, bevor Verarbeitung fortgesetzt wird
    - » Typisches Beispiel: Öffnen-Dialog mit Dateiauswahl
  - *nicht-modal*: Dialogbearbeitung wird parallel zur normalen Arbeit fortgeführt
    - » Typisches Beispiel: Objektinspektor in Entwicklungsumgebungen
- Standardtypen von modalen Dialogen:
  - Hinweis:
    - » Sicherstellen, dass Information vom Benutzer wahrgenommen wurde  
JavaScript: `alert(String)` (meist "OK"-Knopf)
  - Bestätigung:
    - » Bestätigung oder Ablehnung durch Benutzer  
JavaScript: `confirm(String)` (meist "OK"- und "Cancel"-Knöpfe)
  - Abfrage:
    - » Bestimmte Eingabe vom Benutzer abrufen  
JavaScript: `prompt(String, StandardwertString)`

# Beispiel: Fibonacci-Programm mit Prompt

```
<body>...  
  <h2>  
    <script type="text/javascript">  
  
      function fib(n) {  
        ...  
      }  
  
      eing = prompt("Funktionsparameter", "0");  
      document.writeln("fib("+eing+") = "+fib(eing)+"<br>");  
    </script>  
  </h2>  
</body>
```



fibonacci\_prompt.html

# 8. Web-Skriptsprachen

8.1 Clientseitige Web-Skripte: JavaScript

8.2 Dokument-Objekte und DOM



8.3 Objektorientierung in JavaScript

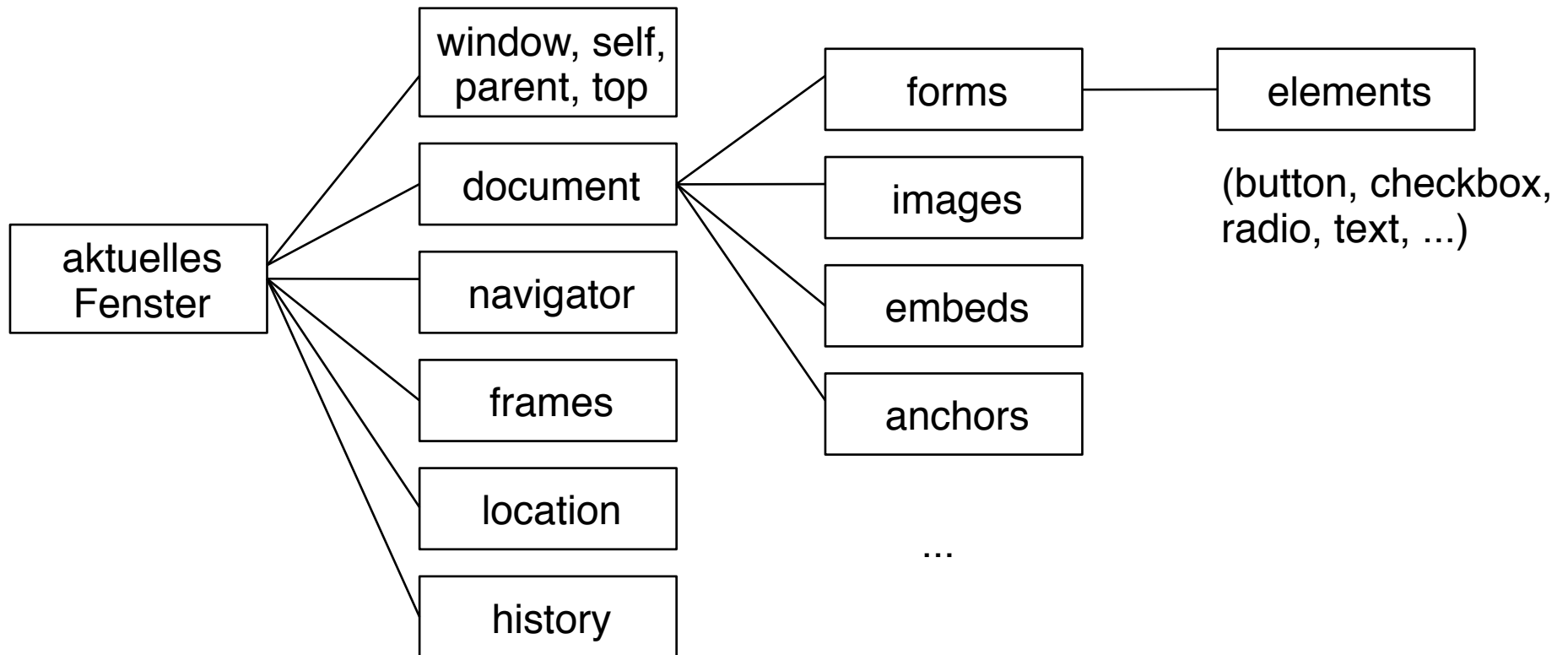
8.4 Serverseitige Web-Skripte (Prinzipien)

*Weiterführende Literatur:*

Jeremy Keith, Jeffrey Sambells: DOM Scripting: Web Design with JavaScript and the Document Object Model, Friends of ed, revised edition 2010

# Traditionell: Vordefinierte JavaScript-Objekte

- Direkter Pfad von Objekt zu Objekt:
  - Häufigstes Ausgangsobjekt "document"-Objekt
  - Objekte stellen Array von Unterobjekten zur Verfügung
  - Unterste Unterobjekte sind HTML-Elemente



# Exkurs zu HTML: Formulare

- Benutzereingabe in HTML:

- `<form>`-Element

- Untergeordnete Elemente:

- `<input type=typ name=name>`

- Mögliche Typen (*typ*) (Auswahl):

<code>checkbox</code>	Wahl-Kästchen
<code>radio</code>	"Radio-Knöpfe" für Alternativen
<code>text</code>	Textzeile
<code>textarea</code>	Mehrzeiliges Textfeld
<code>password</code>	Textfeld zur Passwortabfrage
<code>file</code>	Dateiauswahl
<code>button</code>	Allgemeine Schaltfläche
<code>submit</code>	Schaltfläche zum Absenden des Formularinhalts
<code>reset</code>	Schaltfläche zum Zurücksetzen des Formularinhalts

- `<select name=name>`

- Liste von Optionen: Untergeordnete Elemente vom Typ `<option>`

- `<option selected>` bestimmt "vorselektierten" Standardwert

# Beispiel: Fibonacci-Programm mit HTML-Eingabe

```
<body>...
  <h2>
    Bitte Zahlwert eingeben:
    <form name="formular">
      <input type="text" name="eingabe" value="0"><br>
      <input type="submit" value="Berechnen"
        onClick="
          var eing = document.formular.eingabe.value;
          alert('fib('+eing+') =' +fib(eing));">
    </form>
  </h2>
</body>
```

fibonacci\_form\_trad.html



# Eigenschaften von HTML-Objekten

- Jedes HTML-Objekt hat Eigenschaften (lokale Variable)
  - Jedes Attribut des HTML-Elements ist eine Eigenschaft
  - Universaleigenschaften (z.B. className, id)
  - Auch Code (Methoden) als Eigenschaft zulässig (z.B. onclick)
- Notation:  
*objekt . Eigenschaft*
- Beispiel:  
`<input type="text" name="eingabe" value="0">`  
sei repräsentiert als JavaScript-Objekt `o`
  - Zugriff auf den Wert des Textfeldes:  
`o.value`



# Zugriff auf HTML-Objekte: Traditioneller Stil

- **Achtung: Heutzutage besser Document Object Model verwenden!**

- » Siehe später

- Was bedeutet `var eing = document.formular.eingabe.value;`?

- Kurzschreibweise für

```
var eing =  
document.forms.formular.elements.eingabe.value;
```

- Äquivalent zu

```
var eing =  
document.forms["formular"].elements["eingabe"].value;
```

- Was bedeutet `onClick="..."`?

- » Spezielle Eigenschaft zur Ereignisbehandlung

- » **Heutzutage besser als eigene Funktion definieren und als *Event Listener* registrieren (siehe später)!**

# Beispiel: Ereignisbehandlung in Funktion

```
<script type= "text/javascript ">
...
function showResult() {
    var eing = document.formular.eingabe.value;
    alert('fib('+eing+') =' +fib(eing));
}
</script>
...
<body>...
    <h2>
        Bitte Zahlwert eingeben:
        <form name="formular">
            <input type="text" name="eingabe" value="0"><br>
            <input type="submit" value="Berechnen"
                onClick="showResult();">
        </form>
    </h2>
</body>
```

fibonacci\_form\_trad\_f.html

# Auslesen von Kontextinformation

- Vordefinierte JavaScript-Objekte ermöglichen die dynamische Abfrage von Information

- z.B. über die Browser-Version:

```
var UserAgent = navigator.userAgent;  
var BrowserName = navigator.appName;  
var BrowserCodeName = navigator.appCodeName;  
var BrowserVersion = navigator.appVersion;  
var Platform = navigator.platform;
```

- Hinweis: Moderne Browsernamen (Firefox, Safari etc.) sind in der Regel als Teilzeichenreihe in *userAgent* und/oder *appVersion* codiert.

- z.B. über die Quelldatei:

```
var Location = location;
```

# Markup-Dokument als Baum

- Ein HTML- (und XML-) Dokument im Browser entspricht einem *Baum*

DOM view (hide, refresh):

<http://livedom.validator.nu>

```
DOCTYPE: html
HTML
  HEAD
    #text:
    TITLE
      #text: Fibonacci-Funktion mit Formular, traditionell
    #text:
    SCRIPT type="text/javascript"
      #text: function fib(n){ if (n==0) return 0; else if (n==1) return 1; else return(fib(n-1)+fib(n-2)); }function showResult() { var ei
        alert('fib('+eing+') ='+fib(eing)); }
    #text:
  #text:
  BODY
    #text:
    H1
      #text: Fibonacci-Funktion
    #text:
    H2
      #text: Bitte Zahlwert eingeben:
      FORM name="formular"
        #text:
        INPUT type="text" name="eingabe" value="0"
        BR
        #text:
        INPUT type="submit" name="knopf" value="Berechnen" onclick="showResult();"
        #text:
      #text:
    #text:
```

# Was ist DOM?

- DOM (*Document Object Model*), W3C-Standard
- DOM ist eine Sammlung von Hilfsmitteln für Programme, die mit **Bäumen** arbeiten, die XML- oder HTML-Dokumenten entsprechen
  - Level 2 in modernen Browsern realisiert
  - Level 3 (u.a. XPath-Anbindung) seit April 2004 verabschiedet
- DOM ist eine standardisierte *Programmierschnittstelle* (Application Programming Interface, API)
  - Für viele verschiedene Programmiersprachen nutzbar
  - Funktionen und Eigenschaften (les- und setzbare Werte)
- Wichtigste Funktionen:
  - Navigation im Dokumentbaum
  - Finden von markierten Elementen (z.B. über ID, Elementname)
  - Verwaltung von Inhalten und Attributen
  - Zugriff auf Style Sheets
  - Ereignismodell, z.B. für Benutzeraktionen

# DOM: Ausgewählte Eigenschaften und Funktionen

- Beispiele von Funktionen und Eigenschaften
- Finden von Knoten:  
`getElementById()`, `getElementsByTagName()`
- Elementare Knoteninformation:  
`nodeName`, `nodeValue`, `nodeType`, `attributes`
- Navigation im Dokumentbaum:  
`parentNode`, `hasChildNodes()`, `childNodes`, `firstChild`,  
`lastChild`, `previousSibling`, `nextSibling`;
- Ändern des Dokumentbaums:  
`insertBefore()`, `replaceChild()`, `removeChild()`,  
`appendChild()`

# Dynamische Veränderung von Seiteninhalt (Moderne Version mit DOM)

```
<form name="formular">
  <input type="text" id="eingabe" value="0"><br>
  <input type="button" id="knopf" value="Berechnen">
</form>
<p id="ergebnis">
  Kein Ergebnis bisher.
</p>
```

...

```
<script type="text/javascript"
  ...
  function fibCompute() {
    var eingWert =
      document.getElementById("eingabe").value;
    var ergNode =
      document.getElementById("ergebnis").firstChild;
    ergNode.nodeValue = "fib("+eingWert+") = "+fib(eingWert);
  }
  knopf = document.getElementById("knopf");
  knopf.addEventListener("click", fibCompute, false);
</script>
```

fibonacci\_dom.html

# Dynamische Veränderung von Stilinformation

- CSS-Attribute lassen sich durch DOM/JavaScript manipulieren
- Damit können z.B. Anzeigebestandteile ein/ausgeblendet, umformatiert und bewegt werden.
- Beispiel:

```
<form name="formular">
  <input type="text" id="eingabe" value="0"><br>
  <input type="button" id="knopf" value="Berechnen">
  <span id="hint" style="visibility:hidden;color:red;">
    Zeigt Ergebnis durch dynamische Textver&auml;nderung
  </span>
</form>...
<script type="text/javascript">
  function showHint(){
    document.getElementById("hint").
      style.visibility = "visible";
  } ...
  knopf = document.getElementById("knopf");
  knopf.addEventListener("mouseover", showHint, false);
</script>
```

fibonacci\_dom\_css.html



# Beispiel DOM/CSS: Highlighting in Listen (1)

- HTML-Code:

```
<head>
  <title>List Highlighting with JavaScript</title>
  <style>
    li      {font-family:Helvetica; font-size:20pt}
    .hilite {color:red}
  </style>
</head>

<body>
<ul>
  <li>Listeneintrag 1</li>
  <li>Listeneintrag 2</li>
  <li>Listeneintrag 3</li>
  <li>Listeneintrag 4</li>
</ul>
</body>
```

# Beispiel DOM/CSS: Highlighting in Listen (2)

- JavaScript-Code:

```
<script type="text/javascript">  
    function setHilite(evt) {  
        evt.target.setAttribute("class","hilite");  
    }  
    function unHilite(evt) {  
        evt.target.setAttribute("class",null);  
    }  
  
    var list_items = document.getElementsByTagName("li");  
    for (var i = 0, len = list_items.length; i < len; i++) {  
        list_items[i].addEventListener  
            ("mouseover",setHilite,false);  
        list_items[i].addEventListener  
            ("mouseout",unHilite,false);  
    }  
</script>
```

highlight\_list.html

# 8. Web-Skriptsprachen

8.1 Clientseitige Web-Skripte: JavaScript

8.2 Dokument-Objekte und DOM

8.3 Objektorientierung in JavaScript 

8.4 Serverseitige Web-Skripte (Prinzipien)

Literatur:

Stefan Koch: JavaScript: Einführung, Programmierung und Referenz ,  
dpunkt Verlag, 6. Auflage 2011

Marijn Haverbeke: Die Kunst der JavaScript-Programmierung,  
dpunkt Verlag, 2012

<http://de.selfhtml.org/>

# Objektorientierung in JavaScript?

- JavaScript (in gängigen Versionen)
  - kennt keinen Klassen-Begriff
  - ist *nicht* im strengen Sinn objektorientiert
  - ist “prototyp-basiert” (und wird auch “objekt-basiert” genannt)
- JavaScript enthält das Schlüsselwort “**class**”!
  - (Derzeit) historisches Überbleibsel von ECMAScript Edition 4
    - » Aufgehobene Sprachversion
  - Von ECMAScript abgeleitete Sprachen enthalten zum Teil echte Klassen
    - » z.B. Adobe ActionScript 3 (Flash, Flex)
  - Evtl. wieder Klassen in kommenden Versionen von ECMAScript (“Harmony”)

# Konstrukturen für Objekte in JavaScript (1)

```
// Point constructor objects1.html
function Point(x,y) {
  this.x = x;
  this.y = y;
  this.distance = function(p) {
    return Math.sqrt(
      Math.pow(p.x-this.x,2)+Math.pow(p.y-this.y,2) )
  };
  this.display = function() {
    return "Punkt "+
      (x = "+this.x+", y = "+this.y+")<br>"
  };
}
```

Funktion ist dafür ausgelegt, mit **new** aufgerufen zu werden

Aktuelle Objektinstanz: Schlüsselwort **this**

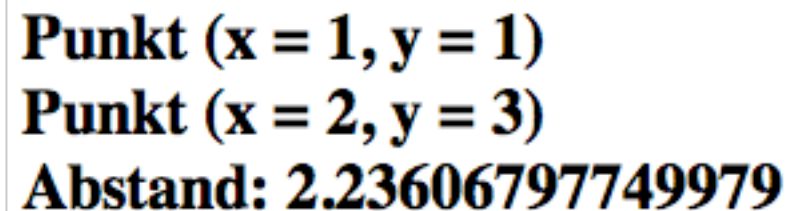
Attribute und Methoden gleichartig behandelt (Funktionsdatentyp)

*Konvention:* Konstruktornamen beginnen mit Großbuchstaben

# Konstrukturen für Objekte in JavaScript (2)

objects1.html

```
p1 = new Point(1,1);  
p2 = new Point(2,3);  
  
document.writeln(p1.display());  
document.writeln(p2.display());  
document.writeln("Abstand: "+p1.distance(p2));
```



```
Punkt (x = 1, y = 1)  
Punkt (x = 2, y = 3)  
Abstand: 2.23606797749979
```

# "Vererbung" in JavaScript (1)

- Jedes Objekt hat die Eigenschaft `prototype`:
  - Verweis auf sein sogenanntes *Prototyp-Objekt*.
- Eigenschaften (incl. Methoden), die nur einmal für alle Instanzen gespeichert werden sollen, kann man im Prototyp-Objekt speichern.

```
// Point constructor
function Point(x,y) {
    this.x = x;
    this.y = y;
}
Point.prototype.distance = function(p) {
    return Math.sqrt(sqr(p.x-this.x)+sqr(p.y-this.y))
}
Point.prototype.display = function() {
    return "Punkt "+"(x = "+this.x+", y = "+this.y+)"<br>"
}
```

objects2.html

# "Vererbung" in JavaScript (2)

```
//ColorPoint constructor
function ColorPoint(x,y,c) {
    this.x = x;
    this.y = y;
    this.color = c;
}
ColorPoint.prototype = new(Point);
ColorPoint.prototype.display = function() {
    return "Punkt "(x = "+this.x+", y = "+this.y",
        color = "+this.color+)"<br>"
}
...
cp1 = new ColorPoint(1,1,"red");
cp2 = new ColorPoint(2,3,"blue");
document.writeln(cp1.display());
document.writeln(cp2.display());
document.writeln("Abstand: "+cp1.distance(cp2));
```

objects2.html

- Objekte "erben" alle Eigenschaften, die in ihrem Prototyp festgelegt sind.



# Dynamische Erweiterung von Objekten

```
// Extending Point prototype
Point.prototype.shift = function(x,y) {
    this.x += x;
    this.y += y;
}
cp1.shift(5,5);
document.writeln("Verschoben: "+cp1.display());
```

objects3.html

- Prototypen können zu jedem Zeitpunkt um weitere Eigenschaften (auch Methoden) erweitert werden.
- Auch bereits bestehende Instanzen werden dadurch erweitert!
- Auch systemdefinierte Objekte haben Prototypen (und können erweitert werden).
- Standardprototyp für Objekte ist der allgemeine `Object()`-Prototyp (der z.B. die Methode `toString()` definiert)

# 8. Web-Skriptsprachen

8.1 Clientseitige Web-Skripte: JavaScript

8.2 Dokument-Objekte und DOM

8.3 Objektorientierung in JavaScript

8.4 Serverseitige Web-Skripte (Prinzipien) 

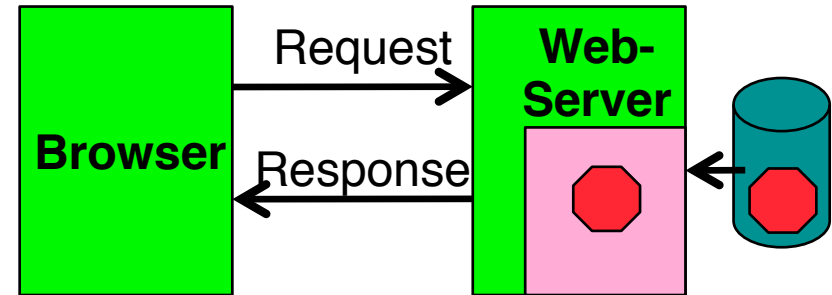
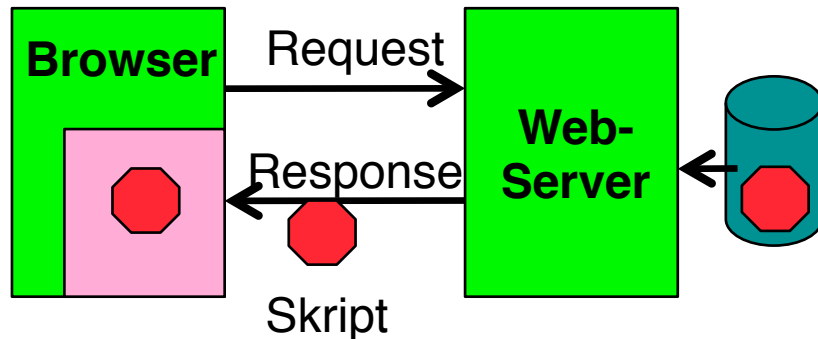
Weiterführende Literatur:

A. Trachtenberg, D. Sklar: PHP Cookbook, O'Reilly 2006

C. Wenz: Ajax - schnell und kompakt, entwickler.press 2007

<http://de.selfhtml.org/>

# Serverseitige vs. clientseitige Dynamik



- Clientseitige Dynamik:

- Browser enthält Ausführungsmaschine für Skripte
- Skript ist Teil der Antwort vom Server
- Web-Server muss Skriptsprache nicht kennen
- Beispiel: JavaScript

- Serverseitige Dynamik:

- Web-Server enthält Ausführungsmaschine für Skripte
- Skript wird vor Beantwortung der Anfrage ausgeführt und liefert HTML-Text
- Browser muss Skriptsprache nicht kennen
- Beispiel: PHP

# Beispiel: Server-Skriptsprache PHP

- PHP:
  - Personal Home Page Toolkit
  - PHP Hypertext Preprocessor
- OpenSource Entwicklung:
  - siehe [www.php.net](http://www.php.net)
  - lizenzfrei benutzbar
- Syntax an C angelehnt, aber mehrere Syntax-Varianten unterstützt
- Einfache Kernsprache, umfangreiche Funktionsbibliothek
  - über 500 Funktionen!
  - etwas unübersichtlich
  - spezialisiert auf Aufgaben der Webseiten-Programmierung

# Beispiel: "Hello World" in PHP und JavaScript

```
<html>
<head><title>Hello World mit JavaScript</title></head>
<body>
  <h1>
    <script type="text/javascript">
      document.write("Hello World!");
    </script>
  </h1>
</body>
</html>
```

JavaScript

```
<html>
<head><title>Hello World mit PHP</title></head>
<body>
  <h1>
    <?php
      echo "Hello World!";
    ?>
  </h1>
</body>
</html>
```

PHP

# Server-Skripte vs. Client-Skripte

## Client-Skripte

Schnelle Reaktion  
Funktion auch ohne Netzanbindung  
Unabhängigkeit von Server-Software

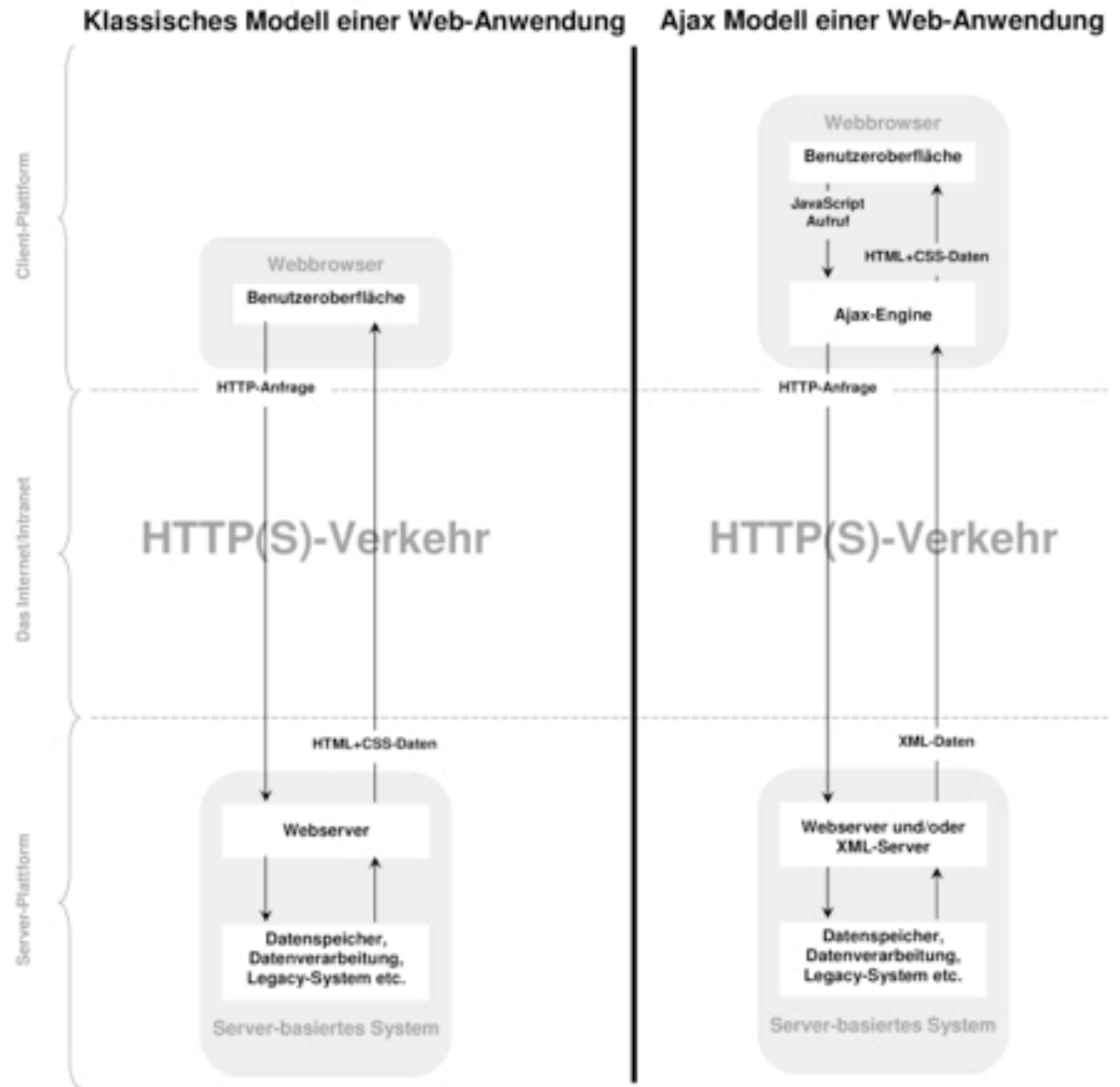
Berechnung von Seiteninhalt aus  
Benutzereingaben und anderen  
äußeren Umständen

## Server-Skripte

Datenhaltung auf Server  
Zugriff auf zentrale Ressourcen (z.B. zur Weiterverarbeitung)  
Unabhängigkeit von Browser-Software

# AJAX und Comet

- AJAX: Asynchronous JavaScript and XML
- Konzept der asynchronen Datenübertragung zwischen Server und Browser
- HTTP-Anfrage wird an Server geschickt ohne Seite neu zu laden
  - (XML-)Antwort wird in JavaScript ausgewertet
- Comet: "reverse AJAX"
  - Push-Technologie, Server sendet Information asynchron zum Browser



Bildquelle: Wikipedia

# WebSockets

- Programmier-Schnittstelle (API), vom World Wide Web Consortium (W3C) definiert
- Ermöglicht es einer Web-Anwendung, mit einem beliebigen Host einen *full-duplex* Kommunikationskanal zu unterhalten
- Noch in Entwicklung
  - JavaScript-Version des API existiert
  - Integriert in den HTML5 Standard
  - Sicherheitsbedenken bremsen den praktischen Einsatz derzeit
- Prinzipiell ein Weg, Ineffizienzen und unnötige Verkehrslast derzeitiger Ajax-ähnlicher Anwendungen zu reduzieren