

# 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://www.w3schools.com/js/>

# (Web-)Skriptsprachen

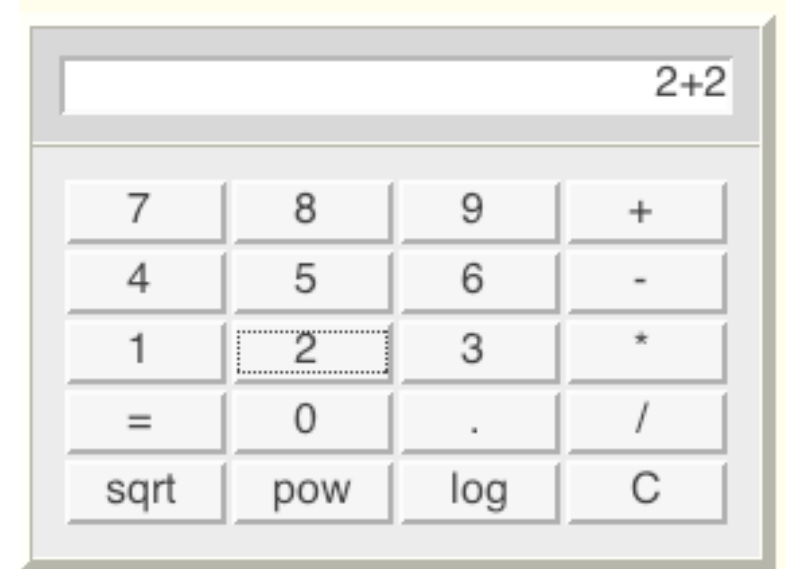
- Skriptsprache: Zur Programmierung von Abläufen in Computersystemen
  - Enge Integration mit Betriebssystem oder speziellem Anwendungssystem
  - Leicht zur Laufzeit zu definieren und zu ändern
- Moderne Skriptsprachen: Echte Alternative zu Programmiersprachen
  - z.B. Perl, Ruby
- Web-Skriptsprachen:
  - Werden im Web-Browser ausgeführt
  - Relativ sicher (kein Zugriff auf lokales Dateisystem und Betriebssystem)
  - z.B. JavaScript, JScript, VBScript

# Was ist JavaScript?

- Web-Skriptsprache
  - Ursprünglich auf Ausführung im Browser optimiert
  - Object-basiert, interpretiert, dynamisches Typing
- 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 ff. wird weiterentwickelt

# JavaScript: Funktionsumfang und Anwendungsbereich

- Traditionelle Beispiele für sinnvolle Anwendung von JavaScript:
  - Formulareingaben auf Plausibilität prüfen
  - Spezialitäten verschiedener Browser-Plattformen flexibel unterstützen
- Moderne Beispiele für sinnvolle Anwendung von JavaScript:
  - Universelle Softwarebasis für plattformübergreifende Anwendungen
- Funktionsumfang:
  - Vollwertige Programmiersprache (!)
  - Verarbeitung von Maus- und Tastatureingaben
  - Dynamische Erzeugung von (HTML-)Ausgabe
  - Zugriff auf Dokument-Struktur über das *Document Object Model (DOM)*
- Steigende Bedeutung, u.a. durch:
  - HTML5 + Canvas
  - WebGL
  - Umfangreiche Bibliotheken



# Einbettung von JavaScript in HTML

```
<!-- Script-Markup -->
```

```
<h1>  
  <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)
  - "Just-in-time" Compiler zur Optimierung
- 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
  - `<!--` *startet einen speziellen einzeiligen 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 anderes Modell von Datentypen:
  - Datentyp einer Variable muss nicht explizit deklariert werden!

# 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>
```



# 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;`

# QUIZ!

- Welches Ergebnis liefern diese Beispiele?
  - `"Apollo"+"5"`;
  - `5 * 5`;
  - `null * 5`;
  - `null * null`;
- Welches Ergebnis liefern wohl diese Beispiele?
  - `"Apollo"+5`;
  - `null + "ify"`;
  - `"5" * 5`;
  - `"strawberry" * 5`;

# Objekte und Eigenschaften in JavaScript

- Eigenschaften (*properties*) für Werte: Paare (Schlüssel – 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;
```

- **QUIZ!** Was liefert wohl das folgende Beispiel?

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

# 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)
```

# 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



# Beispiel & Quiz

- Was macht diese Funktion?

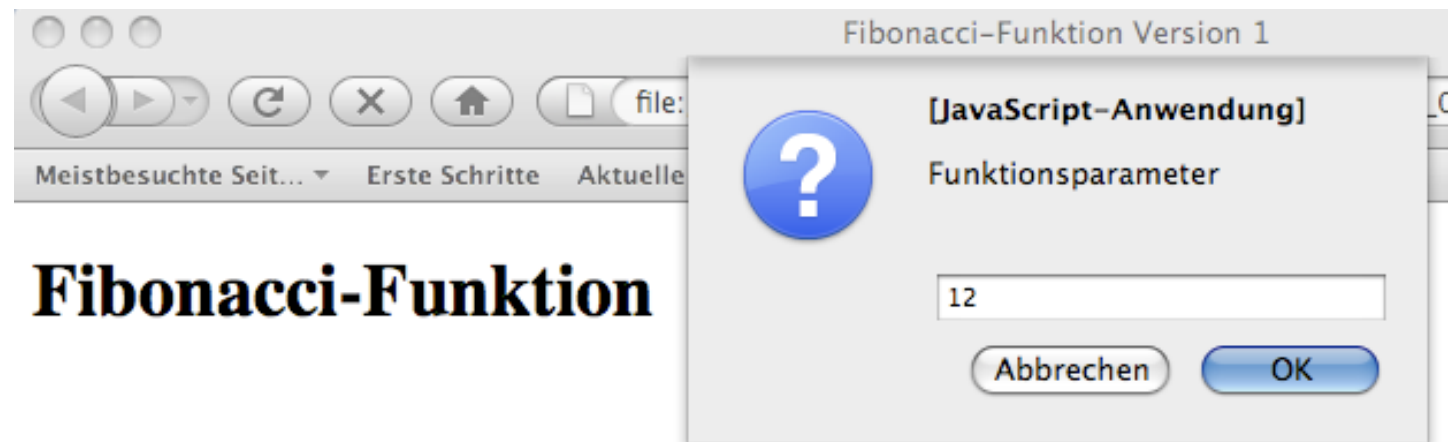
```
function check(Eingabe) {  
    var nur_das = "0123456789[]()-+*%/";  
    for (var i = 0; i < Eingabe.length; i++)  
        if (nur_das.indexOf(Eingabe.charAt(i)) < 0 )  
            return false;  
    return true;  
}
```

# 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

# Exkurs zu HTML: Formulare

- Benutzereingabe in HTML:

- `<form>`-Element

- Untergeordnete Elemente:

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

- Mögliche Typen (*typ*) (Auswahl):

- `checkbox` Wahl-Kästchen

- `radio` "Radio-Knöpfe" für Alternativen

- `text` Textzeile

- `textarea` Mehrzeiliges Textfeld

- `password` Textfeld zur Passwortabfrage

- `file` Dateiauswahl

- `button` Allgemeine Schaltfläche

- `submit` Schaltfläche zum Absenden des Formularinhalts

- `reset` 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>
```

```
        <input type="text" value="0"><br>
```

```
        <input type="submit" value="Berechnen">
```

```
    </form>
```

```
</h2>
```

```
</body>
```

fibonacci\_form\_only.html



**Fibonacci-Funktion**

**Bitte Zahlwert eingeben:**

- Wie wird eine Berechnung aufgerufen?
- Wie kommt der Eingabewert in den Aufruf?

# Eigenschaften von HTML-Objekten

- Eigenschaften von HTML-Objekten (lokale Variable)
  - Jedes Attribut des HTML-Elements ist eine Eigenschaft
  - Universalattribute (z.B. className, id)
- HTML-Objekte können in JavaScript als Objekte benutzt werden!

- Notation:

*objekt . Eigenschaft*

- Beispiel:

```
<input type="text" value="0">
```

sei repräsentiert als JavaScript-Objekt *o*

- Zugriff auf den Wert des Textfeldes:

```
o.value
```

- Setzen des Werts für das Textfeld:

```
o.value = Wert;
```

# Beispiel: Auslesen von HTML-Eigenschaft

```
...
<body>...
  <h2>
    Bitte Zahlwert eingeben:
    <form>
      <input id="eingabefeld" type="text" value="0"><br>
      <input type="submit" value="Berechnen">
    </form>
  </h2>

  <script type="text/javascript">
    ...
    function showResult() {
      var eing = eingabefeld.value;
      alert('fib('+eing+') ='+fib(eing));
    }
    ...
  </script>
</body>
```

Gültiges HTML5!  
Sicherer ist Zugriff mit  
"getElementById",  
siehe gleich!

fibonacci\_form\_compute.html



# HTML-Ereignisse

- Vielzahl von definierten Ereignissen wird vom Browser erkannt
  - Benutzer-Aktionen, z.B. Tastendruck, Mausbewegung, Schaltfläche
  - System-Ereignisse, z.B. Ende eines Ladevorgangs
- Ereignisbehandlung:
  - Auf Wunsch Code aufrufen, der das Ereignis behandelt
  - Code zur Ereignisbehandlung ist *Eigenschaft* von HTML-Element!
- Beispiel: Klicken (auf beliebigem Element)
  - Eigenschaft `onclick`
  - Wert = *Script-Code* (z.B. als Funktionsname)
- Beispiel:

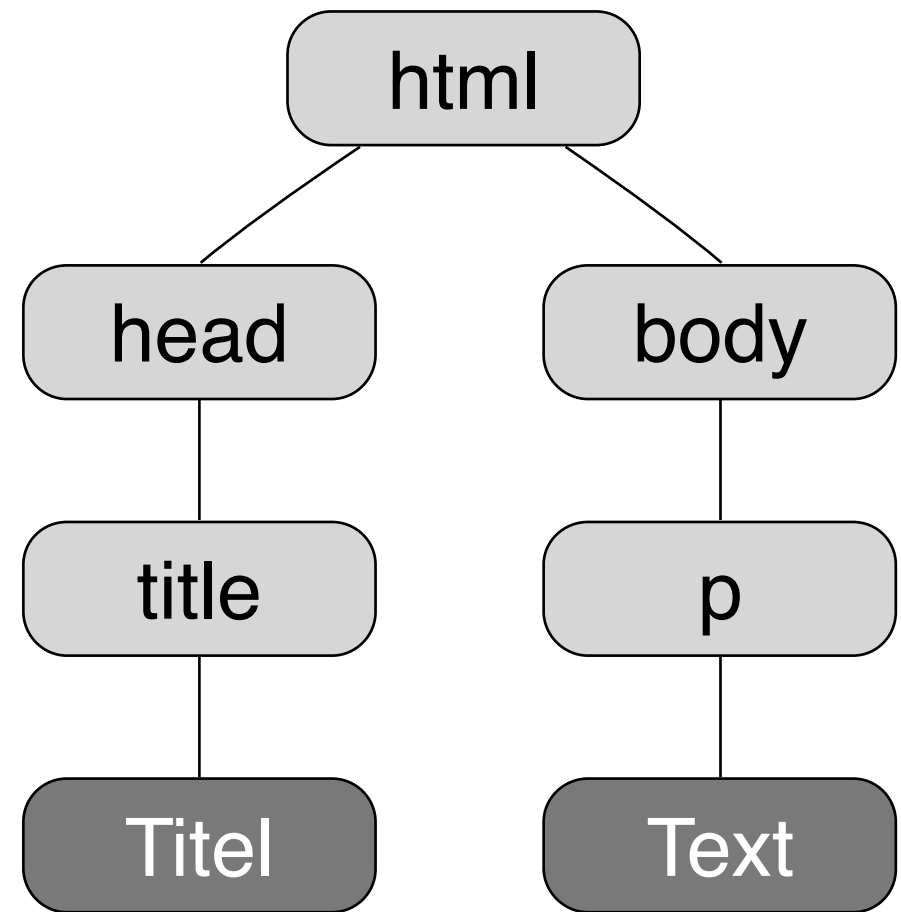
```
<form>  
  <input id="eingabeknopf" type="submit" value="Berechnen">  
</form>...  
<script>  
  eingabeknopf.onclick=showResult;  
</script>
```

fibonacci\_form\_compute.html

# Markup-Dokument als Baum

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

```
<html>  
  <head>  
    <title>  
      Titel  
    </title>  
  <body>  
    <p>Text</p>  
  </body>  
</html>
```



# Datenstruktur im Browser: DOM-Baum

- Diverse Werkzeuge zum Arbeiten mit DOM-Bäumen

DOM view ([hide](#), [refresh](#)):

```
DOCTYPE: html
HTML
  HEAD
    #text:
    TITLE
      #text: Fibonacci-Funktion mit Formular, mit Berechnung
    #text:
  #text:
  BODY
    #text:
    H1
      #text: Fibonacci-Funktion
    #text:
    H2
      #text: Bitte Zahlwert eingeben:
      FORM
        #text:
        INPUT id="eingabefeld" type="text" value="0"
        BR
        #text:
        INPUT id="eingabeknopf" type="submit" value="Berechnen"
        #text:
      #text:
    #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() { eing = eingabef
        eingabeknopf.onclick=showResult;
      #text:
```

<http://software.hixie.ch/utilities/js/live-dom-viewer/>

# Document Object Model (DOM)

- DOM (W3C-Standard)
  - Hilfsmittel für Programme, die mit *Bäumen* arbeiten, die *XML*- oder *HTML*-Dokumenten entsprechen
  - DOM ist eine standardisierte *Programmierschnittstelle (Application Programming Interface, API)*
  - Für viele verschiedene Programmiersprachen nutzbar
  - Funktionen und Eigenschaften (lesbare 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
- Bequemes Arbeiten mit DOM in JavaScript:
  - Einsatz von Bibliotheken, vor allem *jQuery*

# DOM API:

## Elementare Eigenschaften und Funktionen

- 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()`

# Selektieren von Elementen über id-Wert

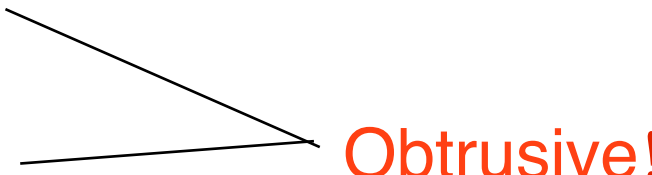
- Annahme: id-Deklaration
  - `<input type="text" id="eingabefeld" value="0">`
- Purer DOM-Stil (empfohlen):
  - `eing = document.getElementById("eingabefeld").value;`
- Direkter Zugriff (über window-Interface) in HTML5:
  - `eing = eingabefeld.value;`
- Ältere Stile ohne DOM-Nutzung:
  - Benutzen **name**-Attribut in Formularen und Eingabeelementen
  - (Historische) platzsparende Syntax

# Ereignisbehandlung in DOM

- Ereignisbehandlung über HTML-Attribut:
  - `eingabeknopf.onclick=showResult;`
- Purer DOM-Stil (empfohlen):
  - `eingabeknopf.addEventListener("click", showResult);`
- Prinzip des *unaufdringlichen* (***unobtrusive***) Codes:
  - Skript-Code getrennt vom Dokument-Gerüst (HTML)
  - Skript- und HTML-Code enthält wenig CSS-Definitionen
  - Purer DOM-Stil führt zu unaufdringlichen und stabilen Skripten
- Für sicheren Skript-Code:
  - Skript erst nach Laden aller DOM-Elemente ausführen
  - In Funktion, die `onload`-Ereignis (besser `DOMContentLoaded`) behandelt

# 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>
```



**Obtrusive!**



# Dynamische Veränderung von Seiteninhalt

```
<form>
  <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);
</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);
</script>
```

# 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);
    list_items[i].addEventListener("click", unHilite);
  }
</script>
```

**QUIZ: Was macht dieses Skript?**

highlight\_list.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;
```

# 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

# 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
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>"
    };
}
```

objects1.html

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

Aktuelle Objektinstanz: Schlüsselwort **this**

Attribute und Methoden gleichartig behandelt (Funktionsdatentyp)

*Konvention:* Konstruktornamen beginnt mit Großbuchstaben



# Konstruktoren 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 im Prototyp-Objekt gespeichert sind, sind für alle Objekte verfügbar, die zu dem Prototyp gehören.

```
// 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)

# JavaScript Object Notation (JSON)

- Notation für geschachtelte Objekte
- Kompatibel mit JavaScript Typsystem
- Beispiel:


```
{ "student": {  
  "name": "Mustermann",  
  "vorname": "Max",  
  "matrnr": "123456",  
  "belegung": [  
    { "code": "MINF-DM",  
      "LVtitel": "Digitale Medien" },  
    { "code": "INF-EiP",  
      "LVtitel": "Einf. in die Progr." }  
  ]  
}  
}}
```

# 8. Web-Skriptsprachen

8.1 Clientseitige Web-Skripte: JavaScript

8.2 Dokument-Objekte und DOM

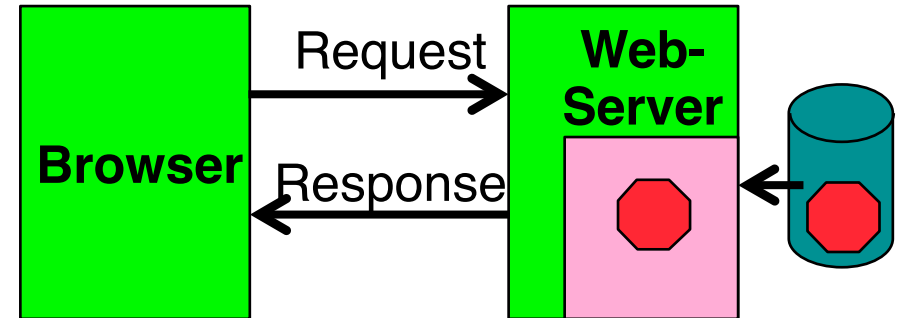
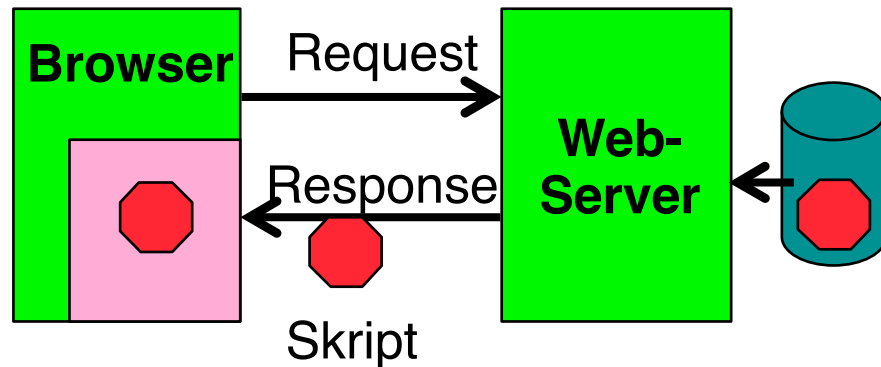
8.3 Objektorientierung in JavaScript

8.4 Serverseitige Web-Skripte (Prinzipien) 

Weiterführende Literatur:

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

# 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 (aber auch JavaScript!)

# 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



# Beispiel: Server-Skriptsprache PHP

- PHP:
  - **P**ersonal **H**ome **P**age Toolkit
  - **P**HP **H**ypertext **P**reprocessor
- 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!
  - spezialisiert auf Aufgaben der Webseiten-Programmierung

# "Hello World"-Beispiel mit serverseitiger Dynamik (PHP) und clientseitiger Dynamik (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

# "Hello World"-Beispiel mit serverseitiger Dynamik (Node.js / Express)

- JavaScript wird zunehmend auch auf Serverseite eingesetzt!
- Beispiel (von <http://expressjs.com>):

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('Hello World!')
})

var server = app.listen(3000, function () {

  var host = server.address().address
  var port = server.address().port

  console.log('Example app listening at http://%s:%s', host, port)

})
```