Maßsysteme in CSS

- relative Angaben
  - em (Höhe von M), ex (Höhe von x) bezogen auf elementeigene Schrifthöhe, d.h. vom Leser beeinflussbar
  - % bezogen auf Standardwert (Fenstergröße, umschließende Box, Standardschrift)

eigentlich zu bevorzugen

- absolute Angaben
  - Pixel für Bildschirmlayout
    - üblich für Bilder und eingebettete Objekte
  - cm, mm, in (Inch) für Drucklayout
  - pt (Punkt) für Schriften, Randabstände, ...

die Auflösung und Pixelgröße des Ausgabegeräts muss bekannt sein!

Schriftgrößen je nach Browsereinstellung

kleine Schrift im Browser

8pt 75% 0.75em (mit Rundungsfehler!)
12pt 100% 1em
24pt 200% 2em
36pt 300% 3em

mittlere Schrift im Browser

8pt 75% 0.75em (mit Rundungsfehler!)
12pt 100% 1em
24pt 200% 2em
36pt 300% 3em

große Schrift im Browser

- pt hängt von der Windows-Schriftgröße ab und bleibt unverändert
- % ist relativ zu einer (den Block umgebenden) Größe (z.B. Schrift)
- em ist relativ und zeigt den Font mit der Höhe, die der Breite eines M's entspricht (mit Rundungsfehlern)

Der (übliche) Wunsch

Hier der Textinhalt, keine Position

Hier ein zweiter Textblock .

kleine Schrift

Hier der Textinhalt, keine Position

Hier ein zweiter Textblock .

mittlere Schrift

Hier der Textinhalt, keine Position

Hier ein zweiter Textblock .

große Schrift

Wunsch: "maßstabsgetreues Zoom"

- ⇨ unterstützt unterschiedliche Schrifteinstellung
- ⇨ funktioniert auf verschiedenen Ausgabemedien

Häufiges Ergebnis

Der Firefox ignoriert feste Zuweisungen (z.B. px) an die Schrift (und damit auch den Standard)!

Aussehen, das der Designer im Sinn hatte...

position:absolute; font-size:2em; top:0px; left:15px;

position:absolute; font-size:4em; top:30px; left:15px;

... und je nach Schriftgrößen-Einstellung oder Ausgabemedium wird daraus

position:absolute; font-size:2em; top

position:absolute;

Wo ist das Problem?

position:absolute; font-size:2em; top:0px; left:15px;

position:absolute; font-size:4em; top:30px; left:15px;

- Schriftgröße ist relativ festgelegt
- Abstände sind absolut festgelegt
- Die Blöcke sind mit *absolute* positioniert, d.h. relativ zur umgebenden Box – aber an fester Position
  - ⇨ die linke obere Ecke der Boxen liegt (bei fester Auflösung) fest
  - ⇨ je nach Schriftgröße verschiebt sich der untere Rand der Boxen

- Für die Bildschirmausgabe sollte die Schriftgröße relativ festgelegt werden, weil nur dann die persönlichen Einstellungen Auswirkungen zeigen!

...aber wie beschreibt man Abstände relativ
- und trotzdem mit einem festen Layout?

Lösung: Systematische Bemaßung mit em

```
<BODY style="font-size:0.1em">
```

definiert feinen Maßstab als einheitliche Bezugsgröße

```
<DIV style="position:absolute; top:21em; left:15.2em">
<! definiert nur Position und Größe>
  <DIV style="font-size:9.2em">
    Hier der Textinhalt, keine Position
  </DIV>
</DIV>
```

```
<DIV style="position:absolute; top:35.7em; left:7.9em">
  <DIV style="font-size:18em">
    Hier ein zweiter Textblock
  </DIV>
</DIV>
```

Schriftgröße und Position sind entkoppelt!
(Schriftgröße im inneren Block änderbar ohne Positionsänderung)

Systematische Bemaßung mit em - Ergebnis

Hier der Textinhalt, keine Position

Hier ein zweiter Textblock .

kleine Schrift

Hier der Textinhalt, keine Position

Hier ein zweiter Textblock .

mittlere Schrift

Hier der Textinhalt, keine Position

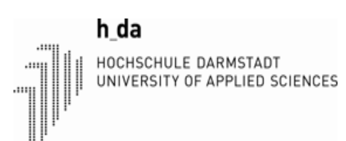
Hier ein zweiter Textblock .

große Schrift

Zusammenfassung

- CSS-Grundlagen
  - ⇨ Grundidee, Grundgerüst, HTML-Einbindung
  - ⇨ Schreibregeln und Syntax
  - ⇨ Formate modifizieren und definieren
- CSS-Attribute
  - ⇨ Farben, Hintergrund, Schriften, Ausrichtung, Ränder, Platzierung,...
  - ⇨ Layout
- CSS-Kaskadierung
  - ⇨ Hierarchie und Konflikte
- Maßeinheiten (wird noch fortgesetzt)

Jetzt wissen Sie alles um eine HTML-Seite mit einem <u>ordentlichen Design</u> zu entwickeln!



Entwicklung webbasierter Anwendungen

5. Kapitel: ECMA-Script und DOM

Quellenhinweis:
Viele Folien dieser Vorlesung entstammen der gleichnamigen Vorlesung von Prof. B. Kreling

© Ralf Hahn, SS 2006, h_da - Fachbereich Informatik

Interaktion mit Webseiten auf dem Client

- Seitenumschaltung
  - ⇨ Hotwords, (transparente) Schaltflächen, sensitive Grafik — HTML
- Eingabeformulare
  - ⇨ Listbox, Checkbox, Radiobutton — HTML
  - ⇨ Konsistenzprüfung der Eingabewerte (auf dem Client !) — ECMAScript
- objektbezogene Ereignisbehandlung
  - ⇨ Veränderung der HTML-Seite (auf dem Client !) — ECMAScript, DOM
- allgemeine Programmierung — Java Applet
  - ⇨ aufwändige Visualisierung, spezielle Widgets (TreeControl)

Wird in der Vorlesung nicht behandelt!

Zur Abgrenzung: Server-seitige Interaktion

Große Datenmengen bzw. persistente Daten werden auf dem Server gehalten!

- Durchsuchen großer Datenmengen
  - ⇨ Datenbankabfrage (z.B. Fahrplanauskunft)
  - ⇨ Volltextsuchmaschine
- Speicherung von Daten
  - ⇨ Gästebuch, Schwarzes Brett, Bestellungen
- Content Management System

- Realisierungsmöglichkeiten
  - ⇨ CGI, proprietäre Server-APIs (NSAPI, ISAPI)
  - ⇨ PHP, ASP, JSP
  - ⇨ Java Servlet
  - ⇨ Java Applet mit RMI (remote method invocation)
  - ⇨ ...

Skriptsprachen wurden ausgebaut

- Ursprung: Programmierung von Eingabeformularen
  - ⇨ Ereignisbehandlung war auf Formulare beschränkt
  - ⇨ komplexere Aufgaben erforderten Java,
    aber selbst damit kein Zugriff auf HTML-Dokument

- seit ca. 1999: Layout-Elemente als programmierbare Objekte
  - ⇨ Ereignisbehandlung mit zugeordneten Skripten
  - ⇨ alle Eigenschaften per Skript änderbar
  - ⇨ Seiten können vom Surfer modifiziert werden (z.B. Warenkorb)
    - ermöglicht Anzeige von Berechnungsergebnissen (z.B. Gesamtkosten)
    - ermöglicht Auf- und Zuklapp-Effekte
    - ermöglicht lokale Animationen
    - ...

DOM
ECMAScript

© Ralf Hahn, SS 2006, h_da - Fachbereich Informatik

Beispiel (eingebettet in HTML)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
  <html>
  <head>
  <title>Test</title>
  <script type="text/javascript">
  <!--
    var Hinweis = "Gleich werden Quadratzahlen ausgegeben";
    alert(Hinweis);

    function SchreibeQuadrate() {
        var SinnDesLebens = 42;
        var i, x;
        var Satzteil = "Das Quadrat von ";
        for(i=1; i <= SinnDesLebens; ++i) {
            x = i * i;
            document.write(Satzteil + i + " ist " + x + "<br>");
        }
    }
  //-->
  </script>
  </head>
  <body onload="SchreibeQuadrate()">
  </body>
  </html>
```

ältere Browser geben sonst das Skript als Text aus

Verwendung des DOM

Ausführung zu Beginn des Ladevorgangs

Quelle: SelfHTML

Historie

hatte mit Java nur C gemeinsam

- Ursprung: JavaScript (Netscape) in Navigator 2.0
  - ⇨ von Netscape an Microsoft lizensiert; MS hinkte hinterher
- JScript (Microsoft)
  - ⇨ lizenzunabhängige Sprachvariante mit MS-eigenen Erweiterungen (MSIE versteht JavaScript und JScript)
- ECMAScript (ECMA-262, herstellerunabhängig)
  - ⇨ European Computer Manufacturer's Association, Genf
    - aktuell: 3rd Edition, 1999
  - ⇨ autorisiert von W3C, übernommen als ISO / IEC 16262
  - ⇨ Netscape und Microsoft haben Einhaltung zugesagt
  - ⇨ http://www.ecma-international.org

darauf konzen-trieren wir uns

Overview

- Free-formatted with C-like syntax. Careful formatting is optional, unlike FORTRAN.
- Interpreted and loosely-typed. You don't have to wrestle with a pedantic compiler.
- Garbage collected with no pointers. Like Java, someone else cleans up your mess.
- Floating point numbers and Unicode strings. Basic types are kept simple.
- Arrays and objects.
  Objects are easy and informal, and have properties and methods.
- Object-based, not object-oriented. Complex object features are left out, unlike C++ or Java.
- Null and undefined special values. Variables and functions can be created anytime.
- Flexible functions.
  Bare statements without a main() will do.
  Variable parameters for functions.
- Highly portable. Hardware independent, so it can run anywhere, much like Java

Vergleich mit C

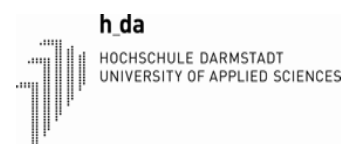
man kann einfach mal drauflos schreiben...

- Syntax sehr ähnlich
  - ⇨ Zuweisung, `if`, `switch`, `for`, `while`, `do`, `try catch`, `//`, `/*...*/`
    - – Besonderheiten: `with`, `for (var Prop in Objekt) {}`
  - ⇨ Konstanten, Operatoren (Stringverkettung mit `+`)
- Variablen nicht typisiert
  - ⇨ Zahlen sind immer Gleitpunktzahlen
  - ⇨ Schlüsselworte `var` bzw. `function` statt Typ in der Deklaration
- Objekterzeugung mit `new`
  - ⇨ wie in Java; kein delete
- nicht zeilenorientiert
  - ⇨ `;` wird am Zeilenende autom. eingefügt, falls es fehlt

    (kein Zeilenende hinter `return` oder vor `++` und `--` lassen !)

Konstanten in ECMAScript

- Notation generell wie in C
  - ⇨ auch `null`, `true`, `false`
- Besonderheit für Strings
  - ⇨ wahlweise mit `"..."` oder `'...'`
    ermöglicht String im String (z.B. String in HTML-Attributwert)
- für Farben leider uneinheitlich
  - ⇨ in HTML: `#FFD700` oder `gold`
  - ⇨ in CSS: `rgb(255,215,0)` oder wie HTML
  - ⇨ in ECMAScript: `0xFFD700`



Array

- dynamisch erzeugtes und erweiterbares Objekt
  - ⇨ ganzzahliger Index im Bereich 0..n
  - ⇨ Elementtyp beliebig und nicht notwendigerweise einheitlich

das kann C nicht...

- Erzeugung
  - ⇨ ohne Längenangabe für dynamische Erweiterung

```
var Vektor1 = new Array ();
```

  - ⇨ mit Längenangabe (eine Zahl)

```
var Vektor2 = new Array (27);
```

  - ⇨ mit Initialisierung (mehr als 1 Wert oder Objekt)

```
var Vektor3 = new Array ("abc", 55, "xyz");
```

- Zugriff

```
var Element = Vektor2[4];
Vektor1[0] = "text";
var AnzahlElemente = Vektor2.length;
```

Assoziatives Array

- dynamisch erzeugtes und erweiterbares Objekt
  - String als Index
  - Elementtyp beliebig und nicht notwendigerweise einheitlich
  - vgl. Datenstruktur / struct / Hashtabelle / map / dictionary
- Erzeugung, Erweiterung und Zugriff
  -
```
var Vektor = new Array ();
Vektor["posLeft"] = 45;
var Element = Vektor["posLeft"];
```
- Verarbeitung
  - häufig mit Hilfe von
```
for (var Element in Vektor) { ... }
```
- Arrays werden beim Zugriff auf DOM häufig gebraucht
  - Assoziative Arrays sind wirklich praktisch

das kann C nicht...

Funktionen

- Deklaration mit Schlüsselwort `function`
- Rückgabe von Werten aus Funktionen durch `return`
  - ⇨ ein Rückgabeparameter wird nicht deklariert
  - ⇨ Klammern nicht vergessen

- Beispiel

```
function Doppel (InParam) {
  var  OutParam = 2 * InParam;
  return OutParam;
}
```

Vordefinierte Funktionen

- Vordefinierte Funktionen können einfach aufgerufen werden
  - ⇨ `isFinite()` auf numerischen Wertebereich prüfen
  - ⇨ `isNaN()` auf numerischen Wert prüfen
  - ⇨ `parseFloat()` in Kommazahl umwandeln
  - ⇨ `parseInt()` in Ganzzahl umwandeln
  - ⇨ `Number()` auf numerischen Wert prüfen
  - ⇨ `String()` In Zeichenkette umwandeln
  - ⇨ `unescape()` Zahlen in ASCII-Zeichen umwandeln
  - ⇨ ...

© Ralf Hahn, SS 2006, h_da - Fachbereich Informatik

Ausnahmebehandlung

- wie in Java / C++, Ausnahmeobjekte jedoch untypisiert

```
try {
    ...
    if (FehlerAufgetreten)
        throw "Text oder Objekt";
    ...
}
catch (Ausnahme) {
    alert (Ausnahme);         // sofern es Text ist
}
```

- zusätzlicher `finally`-Block möglich wie in Java
  - ⇨ wird in jedem Fall ausgeführt
- sicherheitshalber einbauen, auch ohne eigenes throw
  - ⇨ manche Browser werfen bei manchen JavaScript-Fehlern Ausnahmen aus...

Objektbasierend statt objektorientiert

- "...Object-based, not object-oriented. Complex object features are left out..."
  - ⇨ ECMA-Skript kennt keine Klassen, verwirrt den Begriff "Objekt"
    - soll einfacher sein für Novizen ???
    - für OO-Programmierer sehr gewöhnungsbedürftig
  - ⇨ alles und jedes ist ein Objekt, selbst eine Funktion
    - Zitat aus dem Standard:
      "All functions including constructors are objects, but not all objects are constructors."
    - erklärt sich durch implementierungsnahe Sicht:
      Objekt gleichbedeutend mit *Speicherbereich*
  - ⇨ einige vordefinierte "Objekte" sind eigentlich Klassen
    - Boolean, String, Date, Array, ...

Klassen ?

- Klassen werden Objekte genannt
  - ⇨ verwirrend, wenn man mal den Unterschied verstanden hatte

- "Objekte" (eigentlich Klassen)
  - ⇨ `Array, Boolean (true, false), Date, Number, string`
- "Objekte" (eigentlich Funktionsbibliotheken)
  - ⇨ `Math, RegExp` (reguläre Ausdrücke)
- (echte) Objekte der Laufzeitumgebung
  - ⇨ `date, window, navigator, document` (⇨DOM)

Konstruktor statt Klassendeklaration

- Klassen sind nicht deklarierbar, aber Objekte kann man konstruieren
  - ⇨ Attribute werden im Konstruktor initialisiert und damit zugleich definiert
  - ⇨ Methoden werden im Konstruktor zugeordnet
  - ⇨ kein Zugriffsschutz (private / protected / public)

Destruktor gibt es nicht; i.a. nicht nötig wegen Garbage Collection wie in Java

- `Objekt = new KonstruktorFunktion (Parameter);`
  - `new` deutet dynamische Allokation an

`this.Attributname = Wert` definiert ein Attribut

`this.Methodenname = Funktion` definiert eine Methode

- eine solche Funktion darf wiederum intern `this` verwenden

Beispiel: Klasse Bruch

Anwendung der Klasse

```
function main ()
{
   var x = new CBruch (3, 5);
   var y = new CBruch (4, 7);
   var z = x.Mal (y);

   alert (z.m_Zaehler + "/" +
          z.m_Nenner);
}
```

Klassendefinition

```
function CBruch (Zaehler, Nenner)
{
   this.m_Zaehler = Zaehler;
   this.m_Nenner = Nenner;
   this.Mal = CBruch_Mal;
}

function CBruch_Mal (b) {
   var Erg = new CBruch (1,1);
   Erg.m_Zaehler =      this.m_Zaehler *
                        b.m_Zaehler;
   Erg.m_Nenner =       this.m_Nenner *
                        b.m_Nenner;
   return Erg;
}
```

Attribute

Konstruktor

Methode

Klassendeklarationen – wozu?

- Klassendeklaration in C++ / Java
  - ⇨ ermöglicht Typprüfung zur Compile-Zeit
    - –Typprüfung findet in ECMAScript generell zur Laufzeit statt
  - ⇨ definiert Speicherallokation
    - –in ECMAScript nicht nötig, da Objekte dynamisch erweiterbar sind
- Nachteil bei Verzicht: geringere Sicherheit
  - ⇨ kaum Überprüfungen zur Compile-Zeit möglich
  - ⇨ Schreibfehler in Attributnamen erzeugen neue Attribute
- Nachteil bei Verzicht: geringere Geschwindigkeit
  - ⇨ Laufzeittypprüfung kostet Rechenzeit
  - ⇨ es kann kein typspezifischer Code generiert werden

Vererbung

es gibt keine Mehrfachvererbung

- spezielle Eigenschaft jedes Objekts: `prototype`
  - ⇨ enthält Zeiger auf Objekt der Basisklasse; kann `null` sein
  - ⇨ wird im Konstruktor definiert:

    ```
    this.prototype = new Basisklassenkonstruktor(...);
    ```

- impliziter Zugriff auf Attribute + Methoden des prototype
  - ⇨ werden durch gleichnamige Attribute und Methoden des neuen Objekts verdeckt

- aber nicht für Objekte des DOM
  - ⇨ für die ist prototype eine normale benutzerdefinierte Eigenschaft ohne besonderes Verhalten

Platzierung von Skripten

"Globaler Code" wird während des Ladens der HTML-Datei ausgeführt. Funktionen erst bei Aufruf oder als Ereignis-Handler

- "extern" in eigener JS-Datei
  - ⇨ "Bibliothek" kann von mehreren HTML-Dateien genutzt werden

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

- "eingebettet" im HTML-Code
  - ⇨ brauchbar nur für diese eine HTML-Datei
  - ⇨

```
<script type="text/javascript">
<!--
    // ... ECMAScript Anweisungen ...
//-->
</script>
```

in <head>
oder <body>

DOM ist aber erst komplett für Ereignis-Handler !

Ereignisse und Handler

nicht ECMAScript, sondern HTML 4.0

- vordefinierte Ereignisse
  - Maus: onclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout
  - Tastatur: onkeydown, onkeyup, onkeypress
  - Formular: onchange, onfocus, onblur, onsubmit, onreset
  - Datei: onload, onunload, onabort

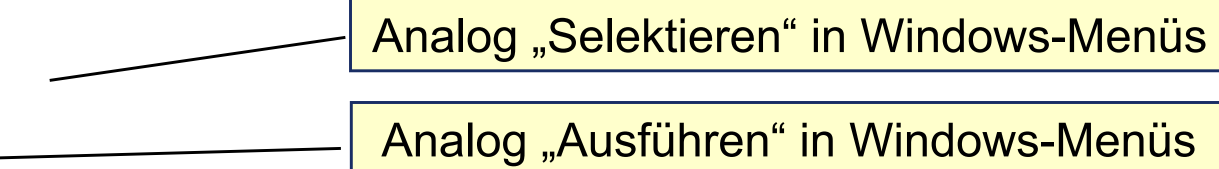
optimal für Initialisierung

- Zuordnung "inline" im HTML-Tag
  - normalerweise: Funktionsaufruf als Event-Handler (beliebige ECMAScript-Anweisungen sind aber möglich)

```
<p onclick="Funktionsaufruf(27)"> ein Text </p>
```

Reihenfolge von Maus-Ereignissen

- beim Schieben:
  - onmouseover — Cursor geht in den Objektbereich
  - onmousemove — wiederholt, solange in Objektbereich
- beim Klicken:
  - onmousedown
  - onmouseup
  - onmouseclick — down und up an derselben Stelle
  - onmousedblclick
- beim Verlassen
  - onmouseout — Cursor verläßt den Objektbereich



t

Überprüfung von Formulareingaben

- Script-Funktion mit boole'schem Ergebnis

```
function FormulardatenOK()
{
   if      (!Feld1_OK) return false;
   else if (!Feld2_OK) return false;
   else                return true;
}
```

- Zuordnung zum Ereignis OnSubmit
  - ⇨ das Abschicken der Formulardaten wird unterdrückt, wenn die Funktion `false` liefert

Sonderfall; nur bei onsubmit

```
<form onsubmit="return FormulardatenOK();">
  <input type="submit" value="Abschicken">
</form>
```

Vorsicht mit globalem Code !

- globale Anweisungen werden ausgeführt, sobald sie eingelesen sind
  - ⇨ d.h. während des Aufbaus der HTML-Seite
    - – die Anzeige der Seite wird ggfs. verzögert
  - ⇨ Achtung: im <head> existiert der DOM-Baum noch nicht !
  - ⇨ sicherer: Initialisierungen im <body> bei onl oad

- globale Anweisungen beschränken auf Deklaration globaler Variablen und deren Initialisierung
  - ⇨ alles andere im Handler für onl oad von <body>
  - ⇨ insbesondere Zugriff auf DOM nicht als globaler Code !

Initialisieren und Aufräumen

- besondere Ereignisse für <body> und <frameset>

```
<body   onload="Initialisieren();"
        onunload="Aufraeumen();">
```

in HTML

oder alternativ über DOM zuweisen

```
document.getElementsByTagName
   ("body")[0].onload = Initialisieren
document.getElementsByTagName
   ("body")[0].onunload = Aufraeumen
```

im Skript

Funktionszeiger (ohne Klammer!)

- onload ruft "Konstruktor" der Seite
  - ⇨ tritt ein, nachdem die Seite komplett geladen wurde
  - ⇨ Zugriff auf DOM jetzt möglich (ist nun komplett aufgebaut)
- onunload ruft "Destruktor" der Seite
  - ⇨ tritt ein, bevor die Seite verlassen wird

Zeitsteuerung

- Verzögerte Ausführung

```
window.setTimeout (Anweisung, Verzoegerung);
```

  - ⇨ Anweisung: beliebige JavaScript-Anweisung (meist Funktionsaufruf), geschrieben als String
  - ⇨ Verzögerung in msec

- Periodische Ausführung

```
var ID = window.setInterval(Anweisung, Periodendauer);
window.clearInterval (ID);
```

  - ⇨ Periodendauer in msec
  - ⇨ ID identifiziert Timer-Objekt (es können mehrere parallel laufen)