J A V A S C R I P T
Dokumentation

Inhalt

1. grundlagen

1.1 einleitung, beschreibung von Javascript seite 3
Javascript-anweisungen, bemerkungen

1.2 objektorientierung (überblick) seite 4

1.3 variable - numerisch, logisch seite 5

1.4 objekt **String** - zeichenkette seite 5
operatoren, eigenschaft length
methoden charAt, charCodeAt, concat, indexOf, lastIndexOf,
replace, search, substr, substring, fromCharCode

1.5 typ umwandeln – globale funktionen String, Number, toString, seite 7
parseInt, parseFloat
typ prüfen – methode isNaN

1.6 operatoren seite 8
arithmetisch, logisch, vergleich, bit verschieben

1.7 objekt **Array** seite 9
eindimensional, zweidimensional, eigenschaft length
methoden pop, push, reverse, shift, unshift, sort

1.8 Javascript in HTML seite 12
JS-script, JS-datei, link

2. programmierung

2.1 alternative (if, else, else if) seite 13

2.2 programmschleifen (for, while, do while) seite 14

2.3 methode **eval** seite 15

2.4 funktionen seite 16

2.5 fallunterscheidung (switch) seite 18

3. standard-objekte

3.1 objekt **window** seite 19
modale fenster alert, confirm, prompt, print

3.2 zeitgeber-methoden seite 21
setTimeout, clearTimeout, setInterval, clearInterval

3.3 navigieren seite 22
objekte **history, location**

3.4 object **document** seite 22
methode write
methoden getElementByID, event (hinweis)
eigenschaften forms, links, images (hinweis)

3.5 objekt **screen** seite 23
eigenschaften height, width, availHeight | Width | Left | Top

3.6 objekt **Date -** diverse methoden seite 24

3.7 objekt **Math -** diverse methoden, zufallszahlen seite 26

4. formular prüfen

4.1 syntax - event onSubmit seite 27

4.2 formular-zugriff seite 27
eigenschaften forms, elements

4.3 formularelement prüfen seite 27
zugriff zu eigenschaften, texteingabefeld, radiobutton
checkbox, auswahlliste

5. HTML-code ändern

5.1 änderung über ID seite 30
methode getElementById
eigenschaften style, className, innerHTML
5.2 formular ändern seite 34
5.3 HTML-element ein- ausblenden seite 36

6. eventhandler

6.1 übersicht seite 40
6.2 event objektbezogen seite 40
event onclick, onfocus, onblur, onkeyup
6.3 event dokumentbezogen seite 43
startfunktion – event onload
funktion für event onmousemove, onmouseover,
onmousedown, onclick, onkeypress
6.4 array document.links und event seite 48

7. objekt Image

7.1 objekt vereinbaren seite 51
eigenschaft images
7.2 bildwechsel seite 51
7.3 galerien seite 53
blättern mit link, onclick, zeitgeber
mit vorschaubildern, durch einblenden
7.4 objekt zoomen seite 60

8. animation

8.1 laufschrift seite 65
8.2 objekt in bewegung seite 68
8.3 objekt bewegen seite 73
8.4 katz und maus seite 75

9. objekt-orientierte programmierung

9.1 objekt vereinbaren seite 79
konstruktor, eigenschaften, methoden
9.2 namenlose eigenschaft seite 82
9.3 geschachtelte objekte seite 83

stichworte seite 85

hinweis

Diese beschreibung kann auf der homepage **www.hartard-bernhard.de** unter **doku/Javvascript** aufgerufen werden. Dort kann diese beschreibung heruntergeladen werden und es können alle beispiele ausgeführt und heruntergeladen werden.

autor **B. Hartard**

stand **2.5 / 01.09.2023**

Javascript

Seit sich **PHP** für die gestaltung von WEB-seiten durchgesetzt hat, ist die bedeutung von Javascript zurückgegangen, ganz verzichten kann man aber nicht darauf. Die nachfolgende beschreibung beschränkt sich daher auf die möglichkeiten, von denen noch immer gerne gebrauch gemacht wird. Ausgeklammert bleiben der einsatz von **frames** und das umfangreiche gebiet der **fenster** (öffnen, schließen, verschieben, gestalten). Beides gilt heute als veraltet.

1. grundlagen

1.1 einleitung

1.1.1 beschreibung von Javascript

Bei der folgenden beschreibung von Javascript gilt:

normalschrift	die angabe ist genau so zu schreiben.
kursiv oder ***kursiv***	das ist ein platzhalter für einen wert, der hier anzugeben ist.
[ ]	angaben in eckigen klammern können wahlweise gemacht werden. Teilweise gehören eckige klammern auch zur syntax der anweisungen.
. . .	die vorangehende angabe kann mehrfach wiederholt werden.
|	durch einen senkrechten strich getrennte angaben sind alternativ.
`Courier New`	code-beispiele sind mit der schriftart Courier New geschrieben

1.1.2 Javascript-anweisungen

Eine anweisung endet mit einem strichpunkt; es können, durch strichpunkt getrennt, mehrere anweisungen in einer zeile stehen.

```
a = 20;
a = 20; b = 30; c = 40;
```

1.1.3 bemerkung

Eine bemerkung ist eine anweisung, die nichts bewirkt, auch nicht irgendwo sichtbar wird, sondern dazu dient, den Javascript-code zu erläutern und zu dokumentieren. Bemerkungen können ein- oder mehrzeilig sein.

Eine **einzeilige** bemerkung beginnt mit dem doppelten schrägstrich und steht entweder in einer zeile für sich allein oder in einer zeile nach einer oder mehreren anweisungen. Eine **mehrzeilige** bemerkung beginnt mit **/***, erstreckt sich über beliebig viele zeilen und endet mit ***/**; sie kann auch nach einer anweisung beginnen.

```
//  das ist eine bemerkung
var a = 20;   // die bemerkung beginnt nach einer anweisung
/*  das ist eine bemerkung,
die sich über mehrere
zeilen erstreckt */
var = 20;  /*  die bemerkung beginnt nach einer anweisung
und endet erst in der nächsten zeile */
```

1.2 objekt-orientierung (überblick)

Javascript ist eine objektorientierte sprache; auf details wird hier nicht eingegangen, es werden nur einige begriffe erklärt, die in dieser unterlage verwendet werden. Etwas mehr dazu im abschnitt 9. Im vergleich zu "moderneren" sprachen ist bei Javascript die objektorientierung eher mager realisiert und es herrscht teilweise einige begriffsverwirrung und unschärfe, die hier nur notdürftig behoben werden kann.

1.2.1 klasse

In der objektorientierten programmierung gibt es den begriff der klasse; eine klasse definiert oder beschreibt die **eigenschaften** und **methoden** von **objekten**. Dabei entsprechen eigenschaften dem, was man sonst als variable bezeichnet und methoden sind nichts anderes als funktionen. Meist gehört zu den methoden eine spezielle funktion, der **konstruktor**, der beim erzeugen eines objekts, die eigenschaften des objekts vorab mit werten versorgt.

In Javascript gibt es **keine** klasse, es gibt nur eine spezielle funktion, die die aufgaben der klasse übernimmt, nämlich die **eigenschaften** und **methoden** eines objekts zu definieren und zudem ein entsprechendes objekt zu erzeugen. Nicht ganz korrekt bezeichnet man deshalb diese funktion auch als **konstruktor-funktion** und den namen der funktion verwendet man als eine art typenbezeichnung, manche nennen es auch eine pseudo-klasse.

1.2.2 objekt und instanz

Das erzeugen eines objekts einer klasse nennt man instanziieren, ein objekt ist also eine instanz einer klasse, d.h. die beiden begriffe sind praktisch synonym. Weil es bei Javascript aber den begriff einer klasse nicht gibt, drückt man sich etwas verschwommen aus und spricht von einem objekt mit dem typ der konstruktors-funktion oder von einer instanz eines objekttyps. Das klingt recht gestelzt, aber es ist eben so.

beispiel

Gegeben sei die konstruktor-funktion **Test**, die instanz **Instan** wird erzeugt mit

```
var Instan = new Test();
```

Oft sind in den runden klammern noch parameter (auch als argumente bezeichnet) angegeben, mit denen werte für eigenschaften des objekts übergeben werden. **Instan** ist der name der instanz oder auch der name des objekts. Um nicht allzuviel verwirrung zu stiften, wird in dieser unterlage meist von instanzen und weniger von objekten geredet.

1.2.3 zugriff

Der zugriff zu einem objekt sieht dann so aus, dass man dem namen einer eigenschaft oder einer methode durch einen punkt getrennt dem namen des objekts (namen der instanz) voranstellt.

```
var erg = Instan.eigen
```

```
var erg = Instan.func();
```

Die erste anweisung greift auf die eigenschat **eigen** der instanz **Instan** zu, die zweite anweisung ruft die methode (funktion) **funk** der instanz **Instan** auf.

hinweis

Es ist eine gewisse schludrigkeit, aber in dieser unterlage wird immer wieder davon gesprochen, irgendetwas ist eine **instanz des objekts xxx**. Gemeint ist natürlich es ist ein **objekt mit dem typ xxx**. Die objektorientierung ist in Javascript zwar allgegenwärtig, aber so seltsam realisiert, dass es ohne sprachliche ungenauigkeiten nicht geht, es sei denn man betreibt schlimme sprachverrenkungen, die dann keiner mehr versteht.

1.3 variable

1.3.1 deklaration

var *name*;	deklaration einer variablen
var *name1, name2* . . . ;	deklaration von mehreren variablen
var *name = wert;*	deklaration und wertzuweisung einer variablen
var *name1 = wert, var name2 = wert* . . . ;	deklaration und wertzuweisung von mehreren variablen
name = wert;	wertzuweisung an eine bereits deklarierte variable

wenn eine variable erstmalig verwendet wird, sollte man unbedingt **var** davor schreiben, bei weiteren zuweisungen, d.h. änderungen der variablen, entfällt **var**, weil sonst die variable neu deklariert wird.

numerische variable

[var] *numvar* = 5 | -5 | 123.45;
[var] *hexvar* = 0x1F;

logische variable

[var] *logvar* = true | false;

1.4 objekt String - zeichenkette

deklaration und wertzuweisung
Auch wenn es meist so aussieht, eine zeichenkette ist keine einfache variable, sondern eine **instanz** des objekt-typs **String**. Nachstehend drei möglichkeiten eine solche instanz mit dem namen **strvar** und dem inhalt **zeichenkette** zu erzeugen. Meist, so auch in dieser unterlage, spricht man von **String-variable** oder auch einfach **zeichenkette**, wenn eine instanz des objekt-typs **String** gemeint ist.

1. `var strvar = new String("zeichenkette");`
2. `var strvar = new String();`
   `strvar = "zeichenkette";`
3. `var strvar = "zeichenkette";`

Der erste fall zeigt die formal vollständige erzeugung und wertzuweisung einer instanz des objekt-typs **String** . Im zweiten fall wird zunächst eine leere instanz erzeugt und dann mit dem inhalt versorgt. Der dritte fall ist eine vereinfachte schreibweise, die bei instanzendes objekt-typs **String** zulässig ist und am häufigsten verwendet wird.

1.4.1 arbeiten mit instanzen des objekt-typs String (String-variable)

String-operator +
Mit dem operator + werden zeichenketten zu einer neuen zeichenkette zusammengefügt.

`var a = "zeichen";`	ergibt: "zeichen"
`var b = "kette"`	ergibt: "kette"
`var c = a + b;`	ergibt: "zeichenkette"

eigenschaft length
Eine **String-variable** hat die eigenschaft **length**, welche die anzahl der zeichen in einer zeichenkette liefert.

```
var kette = "das ist ein einmaliger text";
var erg = kette.length                    ergibt: 27
```

String indizieren

In einer zeichenkette kann man die einzelnen zeichen mit 0 beginnend indizieren.

`var kette = "Mastschwein";`	ergibt: "Mastschwein"
`var erg = kette[2];`	ergibt: "s"
`kette[1] = "i";`	ergibt: "Mistschwein"

1.4.2 String-methoden

Die wichtigsten methoden des objekt-typs **String** werden nachstehend dargestellt. Soweit nicht anders angegeben, wird dabei folgende zeichenkette verwendet:

```
var kette = "das ist ein einmaliger text";
```

charAt(*position*)
gibt das zeichen von der angegebenen ***position*** zurück; das erste zeichen hat die position 0..

```
var erg = kette.charAt(4);                    ergibt:  "i"
```

charCodeAt(*position*)
gibt den zeichencode von der angegebenen ***position*** zurück.

```
var erg = kette.charCodeAt(3);                ergibt:  32, d.h. zwischenraum
```

concat(*string*, *string*, ….)
hängt zeichenketten an eine zeichenkette)

```
var alt = "das ";                                       ergibt:  "das
var neu = alt.concat("ist ", "ein", " ", "mist");       ergibt:  "das ist ein mist"
```

indexOf(*teil* [, *start*])
gibt die position zurück an der ***teil*** zum erstenmal vorkommt. Die suche beginnt am anfang der zeichenkette oder an der angegebenen position ***start***

```
var erg = kette.indexOf("ein");               ergibt:  8
```

lastIndexOf(*teil* [, *ende*])
gibt die position zurück an der ***teil*** zum letzen mal vorkommt. Die suche beginnt am ende der zeichenkette und geht bis zum beginn oder der angegebenen position ***ende***.

```
var erg = kette.lastIndexOf("ein");           ergibt:  12
```

replace(*was*, *womit*)
ersetzt in einer zeichenkette die teilzeichenkette ***was*** durch die zeichenkette ***womit***.

```
var erg = kette.replace("einmaliger", "unsinniger");
                                              ergibt:  "das ist ein unsinniger text"
```

search(*was*)
Sucht in einer zeichenkette die teilzeichenkette ***was*** und liefert deren position

```
var erg = kette.search("einmal");             ergibt:  12
```

substr(*start*, *anzahl*)
liefert eine teilzeichenkette ab der position ***start*** mit ***anzahl*** zeichen

```
var erg = kette.substr(4,3)                   ergibt:  "ist"
```

substring(*start, ende*)
liefert eine teilzeichenkette ab der position ***start*** bis zum zeichen vor der position ***ende***

```
var erg = kette.substring(4, 7);              ergibt:  "ist"
```

String.fromCharCode(*wert1* [, *wert2* , . . .])
Die methode erzeugt eine neue instanz, deshalb muss hier dem namen der methode **String.** vorangestellt werden. Die *werte* sind numerisch und stellen jeweils den dezimalcode eines darstellbaren zeichens dar. Die werte werden in zeichen umgewandelt, das ergebnis ist eine zeichenkette aus diesen zeichen.

```
var erg = String.fromCharCode(65, 66, 67);      ergibt:  "ABC"
```

1.5 typ umwandeln

Häufig ergibt sich die notwendigkeit, einen numerischen wert in eine zeichenkette oder eine zeichenkette in einen numerischen wert umzuwandeln. Um eine zeichenkette, die als numerischer wert interpretiert werden kann, in eine numerische variable umzuwandeln, genügt eine multiplikation mit 1. Die umwandlung einer numerischen variablen in eine entsprechende zeichenkette bewirkt man mit der addition einer leeren zeichenkette. So kann man auch eine logische variable in eine zeichenkette umwandeln. Zur verdeutlichung sind bei den folgenden ergebnissen zeichenketten in anführungszeichen gestellt.

anweisung	***ergebnis***
`test = "25";`	"25"
`test *= 1;`	25
`test += "";`	"25"
`test = true;`	logischer wert
`test += "";`	"true"

Dem gleichen zweck dienen die globalen funktionen (nicht objektbezogene methoden) **String** und **Number** und für die umwandlung einer numerischen variablen in eine zeichenkette die String-methode **toString**.

anweisung	***ergebnis***
`numvar = 100;`	100
`kette  = String(numvar);`	"100"
`numvar = Number(kette);`	100
`numvar = 123.45;`	123.45
`kette  = numvar.toString();`	"123.45"

Beginnt eine zeichenkette mit einem numerisch interpretierbaren teil, kann man diesen teil mit den globalen funktionen **parseInt** (ganzzahliger wert) oder **parseFloat** (dezimalwert) herausziehen.

anweisung	***ergebnis***
`kette  = "123xxx";`	"123xxx"
`numvar = parseInt(kette);`	123
`kette  = "123.45xxx";`	"123.45xxx"
`numvar = parseInt(kette);`	123
`numvar = parseFloat(kette);`	123.45

typ prüfen

erg = **isNaN**(*variable* | *wert*);

Prüft, ob eine variable oder ein wert einen numerischen inhalt hat. Geprüft wird auf "nicht numerisch", d.h. ein numerischer wert liefer **false**, ein nicht numerischer **true**.

1.6 operatoren

1.6.1 arithmetische operatoren

+	addition	`a = b + c;`	`a += b;`
-	subtraktion	`a = b - c;`	`a -= b;`
*	multiplikation	`a = b * c;`	`a *= b;`
/	division	`a = b / c;`	`a /= b;`
%	modulo	`a = b % c;`	`a %= b;`
-	negation	`a = -b;`	
++	um 1 erhöhen	`a++;`	
--	um 1 vermindern	`a--;`	

1.6.2 logische operatoren && || !

zur erklärung dienen folgende anweisungen

```
a = true;
b = true;
c = false;
d = false;
```

&& UND

`x = a && b;`	ergibt true
`x = a && c;`	ergibt false

|| ODER

`x = a || b;`	ergibt true
`x = a || c;`	ergibt true
`x = c || d;`	ergibt false

! NICHT

`x = ! a;`	ergibt false
`x = ! c;`	ergibt true

1.6.3 vergleichsoperatoren

Die üblichen verdächtigen:

==	gleich
!=	nicht gleich
<	kleiner
>	größer
<=	kleiner oder gleich
>=	größer oder gleich

1.6.4 bit-verschiebe-operatoren

wert << n verschiebt um n stellen nach links, zieht von rechts 0 nach
wert >> n verschiebt um n stellen nach rechts, zieht von links das vorzeichenbit nach
wert >>> n verschiebt um n stellen nach rechts, zieht von links 0 nach

1.7 objekt Array

Ein array ist eine unter einem namen zusammengefaßte menge von variablen, die man auch als elemente des array bezeichnet. Die elemente eines array müssen nicht typengleich sein. Ein array ist ein objekt, d.h. eine **instanz** des objekt-typs **Array**.

1.7.1 eindimensionaler array

[var] *name* = **new Array**(*wert*, *wert*, . . .); vollständige schreibweise

[var] *name* **= new Array**(*nn*) instanz mit ***nn*** elementen ohne wert

name[0] = *wert;* wertzuweisung
name[1] = *wert*;
. . .

[var] *name* = **new Array**(); leere instanz
name[0] = *wert;* wertzuweisung
name[1] = *wert;*
. . .

Auch hier gibt es für die erzeugung einer instanz eine vereinfachte schreibweise

[var] *name* = (*wert*, *wert*, . . .) ; vereinfachte schreibweise

1.7.2 zweidimenionaler array

Erzeugt wird eine instanz mit elementen, die selbst wieder instanzen mit elementen und wertzuweisung sind.

var *name* = **new Array**(**new Array**(*wert*, . . .),
new Array(*wert*, . . .), . . .);

beispiel
Erzeugt wird die instanz **name** mit 10 leeren elementen. In einer schleife werden dann als werte für die elemente instanzen mit 6 leeren elementen erzeugt. In einem dritten schritt werden dann diesen elementen werte zugewiesen.

```
var name = new Array(10;                          instanz mit 10 leeren elementen
for (var x = 0; x < name.length; x++)             10 instanzen mit 6 leeren elementen
     name[x] = new Array(6);
name[0][0] = wert1;                               wertzuweisung
name[0][1] = wert2;
...
```

1.7.3 Array-eigenschaft length

liefert die anzahl der elemente eines array

```
var test = new Array("a", "c", "b", "z", "x");
var erg  = test.length;                            erg: 5
```

achtung
Bei einem zweidimensionalen array liefert **name.length** die anzahl der zeilen und **name[0].length** die anzahl der elemente in der ersten zeile. Die interaktive doku von Javascript enthält dazu ein ausführliches beispiel.

beispiel
In dem beispiel werden ein ein- und ein zweidimensionaler array angelegt und angezeigt. Der eindimensionale array wird geändert und erweitert und erneut angezeigt. Zum verständnis des beispiels wird auf abschnitt 1.8 verwiesen. Die methode **document.write** wird in kapitel 3.4 erklärt.

eindimensionaler array test

```
<script type="text/JavaScript">
   var test = new Array("A", "x", "y", "D");
   var ln = test.length;
   var anz = "<p>test enthält " + ln + " elemente: "
   for (var x = 0; x < ln; x++)
       anz = anz + test[x] + " ";
   anz += "</p>";
   document.write(anz);
   test[1] = "B";
   test[2] = "C";
   test[4] = "E";
   ln = test.length;
   anz = "test korrigiert enthält " + ln + " elemente: "
   for (x = 0; x < ln; x++)
       anz = anz + test[x] + " ";
   document.write(anz);
</script>
```

zweidimensionaler array feld

```
<script type="text/JavaScript">
   var feld = new Array(10);
   for (var x = 0; x < feld.length; x++)
        feld[x] = new Array(x+10, x+20, x+30, x+40, x+50, x+60);
   var erg = feld.length;
   var aus = "<p>feld enthält " + erg + " zeilen</p>";
   document.write(aus);
   for (x = 0; x < feld.length; x++)
   {   erg = feld[x].length;
       aus = "zeile " + x + " enthält " + erg + " elemente:   ";
       document.write(aus);
       for (var y = 0; y < erg; y++)
       {   aus = feld[x][y];
           document.write(aus + " / ");
       }
       document.write("<br />");
   }
</script>
```

eindimensionaler array **test**

test enthält 4 elemente: A x y D

test korrigiert enthält 5 elemente: A B C D E

zweidimensionaler array **feld**

feld enthält 10 zeilen

zeile 0 enthält 6 elemente: 10 / 20 / 30 / 40 / 50 / 60 /
zeile 1 enthält 6 elemente: 11 / 21 / 31 / 41 / 51 / 61 /
zeile 2 enthält 6 elemente: 12 / 22 / 32 / 42 / 52 / 62 /
zeile 3 enthält 6 elemente: 13 / 23 / 33 / 43 / 53 / 63 /
zeile 4 enthält 6 elemente: 14 / 24 / 34 / 44 / 54 / 64 /
zeile 5 enthält 6 elemente: 15 / 25 / 35 / 45 / 55 / 65 /
zeile 6 enthält 6 elemente: 16 / 26 / 36 / 46 / 56 / 66 /
zeile 7 enthält 6 elemente: 17 / 27 / 37 / 47 / 57 / 67 /
zeile 8 enthält 6 elemente: 18 / 28 / 38 / 48 / 58 / 68 /
zeile 9 enthält 6 elemente: 19 / 29 / 39 / 49 / 59 / 69 /

1.7.4 Array-methoden (auswahl)

Zur darstellung der methoden wird folgender array verwendet.

```
var test = new Array("a", "c", "b", "z", "x");
```

pop()
Das letzte element wird entfernt und zurückgegeben

```
var erg = test.pop();
```
erg: "x"
test: "a", "c", "b", "z"

push(*elem*, . . .)
Es werden elemente angefügt und die neue länge des array wird zurückgegeben

```
var erg = test.push("m", "n");
```
erg: 7
test: "a", "c", "b", "z", "x", "m", "n"

reverse()
Die reihenfolge der elemente wird umgekehrt

```
var neu = test.reverse();
```
neu: "x", "z", "b", "c", "a"

shift()
Das erste element wird entfernt und zurückgegeben

```
var erg = test.shift();
```
erg: "a"
test: "c", "b", "z", "x"

sort()
Die elemente werden alphanumerisch sortiert

```
var neu = test.sort();
```
neu: "a", "b", "c", "x", "z"

unshift(elem, . . .)
Es werden elemente am anfang eingefügt und die neue länge des array zurückgegeben

```
var erg = test.unshift("m", "n")
```
erg: 7
test: "m", "n", "a", "c", "b", "z", "x"

1.8 Javascript in HTML

In einer mit **HTML** beschriebenen seite können Javascript-anweisungen in HTML-anweiungen auftreten, meist sind sie aber in einem script (JS-script) zusammengefaßt oder sie sind in einer datei gespeichert, die in die seite eingebunden wird.

1.8.1 JS-script

In einem JS-script stehen die anweisungen von Javascript innerhalb eines **script**-tags; das tag verwendet das attribut **language** oder **type**. Das attribut language gilt als veraltet, ist aber immer noch gültig.

<script **type**="text/JavaScript" | **language**="JavaScript">
  anweisungen
</script>

Meist steht ein JS-script im **header** der seite und enthält dann eine oder mehrere funktionen (siehe 2.4) und | oder daten, die von den funktionen benötigt werden. Ein script kann aber auch im **body** der seite stehen und enthält dann entweder eine funktion oder auch anweisungen, mit denen direkt in die seite geschrieben wird (siehe 3.4 - document.write).

1.8.2 JS-datei

Die Javascript-anweisungen können in einer datei stehen, die in eine seite eingebunden wird. Das einbinden erfolgt meist im **header**, kann aber auch im **body** erfolgen. Eine JS-datei entspricht funktional einem JS-script Das einbinden erfolgt mit einem **script**-tag mit dem attribut **src**. Eine JS-datei hat üblicherweise die namens-erweiterung **.js** und ist eine einfache text-datei.

<script type="text/JavaScript | language="JavaScript"" src="*name*.js">
</script>

Die angabe language gilt als veraltet.

1.8.3 link

In einem link können Javascript-anweisungen stehen. Es ist zwar jede Javascript-anweisung zulässig, meist handelt es sich aber um den aufruf einer funktion. Wenn es sich dabei um eine eigene funktion handelt, muss sie im header oder body vorhanden sein oder in in einer eingebundenen JS-datei stehen.

 [*text*]

beispiel 1a – Javascript in HTML

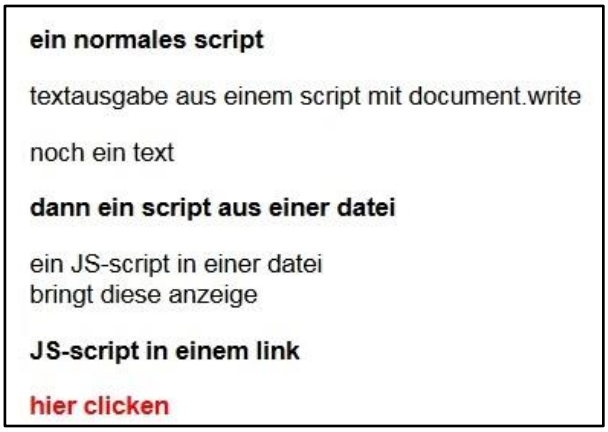
Das beispiel zeigt die verschiedenen möglichkeiten, Javascript zu verwenden, die dabei benütze methode **document.write** wird in kapitel 3.4 erklärt.

```
<p class="font10b">ein normales script</p>
<script type="text/JavaScript">
   document.write("textausgabe aus einem script mit document.write");
   var text = "<p class='font10'>noch ein text";
   document.write(text + "</p>");
</script>

<p class="font10b">dann ein script aus einer datei</p>
<script type="text/JavaScript" src="test1.js">
</script>

<p class="font10b">JS-script in einem link</p>
   <p><a href="JavaScript: alert('link auf alert');">
   hier clicken</a></p>
```

In dem beispiel wird die datei **test1.js** mit folgendem inhalt verwendet:

```
// das ist eine js-datei
document.write("ein JS-script in einer datei<br />");
document.write("bringt diese anzeige");
```

Und so sieht das dann aus:



2. programmierung

2.1 alternative

hier gilt das übliche; alternativen können geschachtelt werden.

if (*bedingung*)
{ *anweisungen*
}

else if (*bedingung*)
{ *anweisungen*
}

else
{ *anweisungen*
}

Die anweisungen **else if** und **else** sind nicht notwendig.
Wenn den anweisungen **if, else if** oder **else** nur eine anweisung folgt, können die geschweiften klammern entfallen.

2.2 programmschleifen

Der formalismus der programmschleifen gleicht weitgehend dem formalismus von **C** oder **PHP** und wird daher hier ohne weitere erklärungen an hand von beispielen gezeigt. Die verwendete methode **document.write** wird in kaptel 3.4 beschrieben.

beispiel 2a - schleifen

2.2.1 for-schleife

```
<script type="text/JavaScript">
document.write("<p><b>for-schleife
                </b><br />");
var zahl;
var zeile = " ";
for (zahl=0; zahl<=10; zahl++)
    zeile += zahl + "  ";
zeile = zeile + "<br />";
document.write(zeile);
```

unbedingte for-schleife

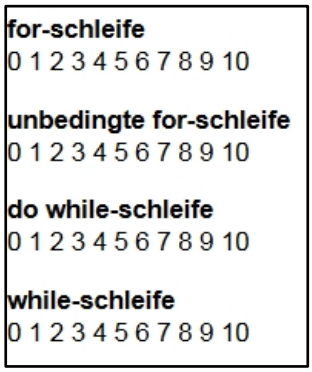
```
document.write("<p><b>unbedingte
            for-schleife</b><br />");
zahl  = 0;
zeile = " ";
for (;;)
{   zeile += zahl + "  ";
    zahl++;
    if (zahl > 10)
        break;
}
zeile = zeile + "<br />";
document.write(zeile);
```

2.2.2 while-schleife

```
document.write("<p><b>while-schleife
                          </b><br />");
zahl  = 0;
zeile = " ";
while (zahl <= 10)
{   zeile += zahl + "  ";
    zahl++;
}
zeile = zeile + "<br />";
document.write(zeile);
```

2.2.3 do while-schleife

```
document.write("<p><b>do while-
                  schleife</b><br />");
zahl  = 0;
zeile = " ";
do
{   zeile += zahl + "  ";
    zahl++;
}
while (zahl <= 10);
zeile = zeile + "<br />";
document.write(zeile);
</script>
```



.

2.3 methode eval

Die methode **eval** ist eine objekt-unabhängige methode, die als argument eine zeichenkette erhält, die sie auswertet und als Javascript-anweisung ausführt. Das bedeutet, dass man damit anweisungen ganz oder teilweise "bastelt". Zur erklärung soll folgender fall dienen: es gibt 10 variable mit den namen **artikel-1** bis **artikel-10**, deren inhalt ausgegeben werden soll. Man kann dazu zehn anweisungen schreiben nach dem muster;

```
document.write("Ergebnis: " + artikel-1);
```

Weniger schreibaufwendig geht das mit in einer for-schleife deren lauf-variable die werte 1 bis 10 annimmt:

```
for (IX=1; IX<=10; IX++)
    document.write("Ergebnis: " + eval("artikel-" + IX);
```

Hier erzeugt eval nacheinander string-variable mit dem inhalt "artikel-1", "artikel-2" usw.

Man kann auch gleich die ganze anweisung mit eval erzeugen, dann muss man allerdings mit einer kombination von anführungszeichen und apostrophen oder auch entwerteten anführungszeichen oder apostrophen arbeiten.

```
eval("document.write('Ergebnis: ' + artikel-" + IX + ")");      oder
eval("document.write(\"Ergebnis: \" + artikel-" + IX + ")");
```

Sehr übersichtlich ist das nicht, aber es funktioniert. Bei komplexeren anweisungen vermeidet man solche kunststücke besser und stellt den aufruf von eval möglichst "tief" in die anweisung. Das folgende beispiel soll das verdeutlichen, indem es auf vier verschiedene arten mit eval jeweils das gleiche ergebnis erreicht, das aussagt, dass es bananen, orangen und kartoffel zu kaufen gibt.

beispiel 2b - eval-methode

```
<script type="text/JavaScript">
var IX, name1, name2, name3, anf, ende, text;
name1 = "bananen ";
name2 = "orangen ";
name3 = "kartoffel ";
anf  = "hier gibt es ";
ende = " zu kaufen <br />";
for (IX=1; IX<=3; IX++)
{   eval("document.write('hier gibt es ' + name" + IX + "+ 'zu kaufen<br />')");

    eval("document.write(anf + name" + IX + "+ ende)");

    document.write(anf + eval("name" + IX) + ende);

    text =  anf + eval("name" + IX) + ende;
    document.write(text);
}
</script>
```

hier gibt es bananen zu kaufen
hier gibt es bananen zu kaufen
hier gibt es bananen zu kaufen
hier gibt es bananen zu kaufen
hier gibt es orangen zu kaufen
hier gibt es orangen zu kaufen
hier gibt es orangen zu kaufen
hier gibt es orangen zu kaufen
hier gibt es kartoffel zu kaufen
hier gibt es kartoffel zu kaufen
hier gibt es kartoffel zu kaufen
hier gibt es kartoffel zu kaufen

2.4 funktionen

2.4.1 deklaration und aufruf

Die konstruktion einer funktion geht in Javascript wie bei anderen sprachen (C, PHP). Etwas unklar ist die verwendung von variablen. Wird eine variable außerhalb aller funktionen deklariert, ist sie global und kann überall, also auch in funktionen gelesen und geschrieben werden. Wird eine variable mit **var** in einer funktion deklariert, ist sie lokal und steht nur in dieser funktion zur verfügung. Wird in einer funktion einer nicht deklarierten variablen (also ohne **var**) ein wert zugewiesen, ist sie global und hat auch außerhalb der funktion den zugewiesenen wert. Das ist verwirrend und man sollte davon nicht gebrauch machen, d.h. in funktionen lokale variable immer mit **var** deklarieren und globale variable immer außerhalb der funktionen vereinbaren.

deklaration

function *name* ([*par* [= *default*] , . . .])
{ deklaration von lokalen variablen
anweisungen
[return [*ergebnis*]] ;
}

Die return-anweisung am ende einer funktion ist nur nötig, wenn die funktion ein ergebnis zurückgibt. Eine funktion kann mehrere return-anweisungen haben.

aufruf

[*erg*] = *name* ([*arg*, . . .]);

2.4.2 parameter

Parameter können mit einem default-wert vorbelegt werden. Beim aufruf einer funktion muss die anzahl der übergebenen argumente nicht mit der anzahl der parameter in der funktionsdeklaration übereinstimmen. Ob für einen parameter tatsächlich ein wert übergeben wurde, kann man mit der prüfung auf **undefined** feststellen. Formal ist eine funktion ein objekt mit der eigenschaft **arguments**, die in der form eines array die übergebenen argumente enthält. Das ermöglicht folgende anweisungen:

anz = *funktionsname*.**arguments.length** anzahl der übergebenen argumente

erg = *funktionsname*.**arguments[***index***]** ein übergebene argument

2.4.3 beispiel 2c - funktionen

```
function funktest (par1="vorbelegt", par2)
{   var erg;
    if (par2 == undefined)
        par2 = "fehlt";
    document.write("par1: " + par1 + "<br />");
    document.write("par2: " + par2 + "<br />");
    erg = par1 + " / " + par2;
    return erg;
}

function testfunk()
{   var arg, x, z;
    arg = testfunk.arguments.length;
    if (arg <= 0)
        document.write("keine argumente übergeben <br />");
    else
    {   document.write(arg + " argumente übergeben<br />");
        for (x=0; x<arg; x++)
        {   z = x + 1;
            argu = testfunk.arguments[x];
            document.write("argument " + z +
            " - " + argu + "<br />");
        }
    }
    return;
}
```

Mit folgenden anweisungen werden die funktionen in einem **JS-script** aufgerufen:

```
<script type="text/JavaScript">
   var ergeb;
   document.write("funktest mit 2 parametern <br />");
   ergeb = funktest("das ist", "ein test");
   document.write(ergeb + "<br /><br />");
   document.write("funktest mit 1 parameter <br />");
   ergeb = funktest("aaaa");
   document.write(ergeb + "<br /><br />");
   document.write("funktest ohne parameter <br />");
   ergeb = funktest();
   document.write(ergeb + "<br /><br />");

   document.write("testfunk ohne parameter <br />");
   testfunk();
   document.write("<br />testfunk mit 4 parametern <br />");
   testfunk(3, 25, "test", "mist");
</script>
```

und so sieht das dann aus:

funktest mit 2 parametern
par1: das ist
par2: ein test
das ist / ein test

funktest mit 1 parameter
par1: aaaa
par2: fehlt
aaaa / fehlt

funktest ohne parameter
par1: vorbelegt
par2: fehlt
vorbelegt / fehlt

testfunk ohne parameter
keine argumente übergeben

testfunk mit 4 parametern
4 argumente übergeben
argument 1 - 3
argument 2 - 25
argument 3 - test
argument 4 - mist

Natürlich kann man eine funktion auch mit einem **link** aufrufen, wie in kapitel 1.8 gezeigt, aber meist ist das nicht sehr zweckmäßig, weil man bei beendigung der funktion nicht unbedingt dort landet, wo man es erwartet hat.

```
<p><a href="JavaScript: testfunk(3, 25, 'test', 'mist')">aufruf</a>
   funktion <b>testfunk</b> mit 4 parametern</p>
```

Man beachte, dass mit der vorstehenden anweisung ein **lin**k ausgeführt, d.h. eine neue seite aufgerufen wird, in die die funktion **testfunk** das ergebnis schreibt. Diese seite formatiert den text offenkundig anders als zuvor.

4 argumente übergeben
argument 1 - 3
argument 2 - 25
argument 3 - test
argument 4 - mist

2.5 fallunterscheidung

Abhängig vom inhalt einer variablen werden unterschiedliche folgen von anweisungen ausgeführt.

switch (*variable*)
{ case *wert1*:
anweisungen,
break;
case *wertn*:
anweisungen;
break;
default:
anweisungen;
}

beispiel 2d - fallunterscheidung

```
<script type="text/JavaScript">
function swstd(art)
{   var erg= " ";
    switch (art)
    {   case 1:
           erg = "Spaten"; break;
        case 2:
           erg = "Rechen"; break;
        case 3:
           erg = "Schaufel"; break;
        case 4:
           erg = "Beil"; break;
        default:
           erg = "unbekannt";
    }
    return erg;
}
 </script>
```

aufruf der funktion

```
<script type="text/JavaScript">
for (x=0; x<=5; x++)
{   var erg = swstd(x);
    document.write("artikel " + x + " ist <b>" + erg + "</b><br />");

}
</script>
```

standard-switch
artikel 0 ist **unbekannt**
artikel 1 ist **Spaten**
artikel 2 ist **Rechen**
artikel 3 ist **Schaufel**
artikel 4 ist **Beil**
artikel 5 ist **unbekannt**

3. standard-objekte und methoden

3.1 objekt window

Das aktuell geöffnete fenster wird als objekt window zur verfügung gestellt. Das objekt enthält zahlreiche eigenschaften und methoden, von denen hier nur die wichtigsten vorgestellt werden. Praktischerweise werden die methoden und unterobjekte des objekts window meist ohne diesen namen aufgerufen.

3.1.1 modale fenster

Ein **modales fenster** ist ein fenster, das im aktuellen fenster geöffnet wird und dieses solange blockiert, bis es wieder geschlossen wird. Modale fenster sind besonders praktisch bei der entwicklung einer neuen seite, hernach im normalbetrieb sollte man weitgehend auf sie verzichten, weil sie das erscheinungsbild einer seite doch sehr unschön "zerreißen". Das aussehen der modalen fenster unterscheidet sich bei den verschiedenen browsern. Für modale fenster gibt es einige methoden des objekts **window**.

alert - hinweis

Es wird ein modales fenster geöffnet, das eine zeichenkette und den button **OK** anzeigt. Mit dem button wird das fenster geschlossen.

alert(*zeichenkette*);

confirm - bestätigung

Es wird ein modales fenster geöffnet, das eine zeichenkette und zwei botton anzeigt. Mit dem button **OK** wird das fenster geschlossen und es wird **true** zurückgegeben. Mit dem button **abbrechen** wird beim schließen des fensters **false** zurückgegeben.

erg = **confirm**(*zeichenkette*);

prompt - eingabe

Es wird ein modales fenster geöffnet, das eine zeichenkette, ein einzeiliges eingabefeld und zwei button anzeigt. Mit dem button **OK** wird das fenster geschlossen und die eingabe wird zurückgegeben. Mit dem button **abbrechen** wird beim schließen des fensters **false** zurückgegeben.

erg = **prompt**(*zeichenkette*);

print - drucken

Es wird das modale fenster **Drucken** geöffnet, mit dem der ausdruck der aktuellen seite gesteuert wird. Mit dem button **OK** wird das fenster geschlossen und der ausdruck gestartet, mit dem button **abbrechen** wird das fenster geschlossen. Die methode wird oft in einer funktion aufgerufen, in der zunächst geprüft wird, ob der browser die methode **print** kennt. Das war früher unbedingt notwendig.

beispiel 3a - modale fenster

```
<script type="text/JavaScript">
function drucken()
{  if (window.print)
      print()
   else
      alert("Leider nicht
              möglich!")
}

alert("Hallo");
var erg = confirm("soll ich weitermachen ?");
if (erg)
    document.write("okey dann geht es weiter");
var benutzername = prompt("Wie heißen Sie?");
if (benutzername)
    alert("Guten Tag, Herr/Frau " + benutzername);
</script>
```

aufruf der funktion drucken

```
<p class="font10b">Drucken mit JavaScript</font></p>
<p class="font10b"><a href="javascript:drucken()">Drucken</a>
   seite drucken</p>
```

Die beispiele zeigen die modalen fenster des browsers Firefox

beispiel alert

beispiel confirm

beispiel prompt + alert

beispiel funktion drucken

Am ende der ausführung des beispiels wird angezeigt:

Drucken mit Javascript

Drucken seite drucken

Wenn man dann den link **Drucken** auslöst wird das modale fenster **Drucken** geöffnet, mit dem dann der ausdruck der seite (nicht des modalen fensters) ausgelöst werden kann. Auf die darstellung des modalen fensters wird verzichtet, weil die unterschiede bei den verschiednen browsern sehr groß sind.

3.2 zeitgeber

Das objekt **window** enthält sog. zeitgeber-methoden.

3.2.1 setTimeout

Mit dieser methode bewirkt man, dass eine anweisung erst nach ablauf einer bestimmten zeit ausgeführt wird. Die anweisung wird als zeichenkette, die zeit als numerischer wert (millisekungen) angegeben. Die methode gibt ein ergebnis zurück, das genau diesen Timeout identifiziert. Man benötigt das ergebnis ggf. für die methode **clearTimeout,** es muss daher in einer **globalen** variablen gespeichert werden. Es können mehrere Timeout gleichzeitig aktiv sein.

[*erg* =] **setTimeout**(*zeichenkette, wert*);

3.2.2 clearTimeout

Die methode stoppt einen laufenden Timeout; man benötigt dafür das ergebnis der methode **setTimeout**.

clearTimeout(*erg*);

3.2.3 setInterval

Die methode gleicht weitgehend der methode **setTimeout**, mit dem unterschied, dass die anweisung immer wieder nach ablauf der gesetzten zeit ausgeführt wird.

[*erg* =] **setInterval**(*zeichenkette, wert*);

3.2.4 clearInterval

Die methode stoppt einen laufenden interval-zeitgeber; man benötigt dafür das ergebnis der methode **setInterval**.

clearInterval(*erg*);

beispiel 3b - zeitgeber

In dem beispiel werden die zeitgeber-methoden in funktionen aufgerufen; die funktionen werden jeweils mit einem link gestartet.

```
<script type="text/JavaScript">
var ID
function tmoutset(zeit)
{    ID = setTimeout("location.href='js-test3-hilf.php'", zeit);
}
function tmoutstop(par)
{    clearTimeout(par);
     alert("Timeout gestoppt");
}
var IDI;
function interset()
{    ID = setInterval("intertxt()", 10000);
}
function intertxt()
{    alert("hallo");
}
function interstop(par)
{    clearInterval(par);
     alert("Interval gestoppt");
}
</script>
```

aufruf der funktionen

```
<p class="font10b"><a href="javascript:tmoutset(15000);">start</a>
   timeout starten</p>
<p class="font10b"><a href="javascript:tmoutstop(ID;);">stop</a>
   timeout stoppen</p>
<p class="font10b"><a href="javascript:interset();">start interval</a>
   interval starten</p>
<p class="font10b"><a href="javascript:interstop(IDI);">stop interval</a>
   interval stoppen</p>
```

3.3 navigieren

3.3.1 objekt history

Das objekt ist ein unterobjekt von **window** und enthält eine liste der besuchten seiten. In der liste kann man mit den methoden **back** oder **forward** rückwärts oder vorwärts navigieren und damit die entsprechenden seiten aufrufen. Das objekt enthält die eigenschaft **length** mit der anzahl der besuchten seiten.

```
<a href='Javascript: history.back();'>zurück</a>
<a href='Javascript: history.forward();'>vorwärts</a>
```

3.3.2 object location

Das objekt **location** ist ein unterobjekt von **window** und enthält die eigenschaft **href,** mit der eine neue seite geladen wird. Das objekt wird gerne in einem link verwendet. Je nachdem, wo die seite gespeichert ist, muss auch der pfadname angegeben werden.

location.href ="*name der seite*";

```
<a href="javascript:location.href='name.htm'">wechsel</a>
```

3.4 objekt document

Das objekt ist ein unterobjekt zu **window** und enthält die aktuelle seite; zu dem objekt gehören viele methoden und eigenschaften, die hier nicht alle erklärt werden, vielmehr wird meist nur auf die stelle verwiesen, wo in dieser beschreibung eine methode oder eigenschaft erstmals verwendet und dann auch erklärt wird.

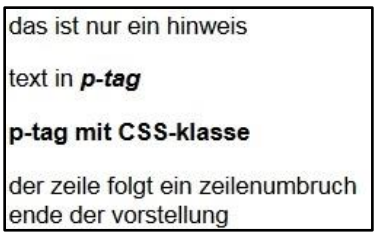
3.4.1 methode write

Die methode **write** gibt einen text in das aktuelle dokument (angezeigte seite) aus, der text kann auch HTML-tags enthalten, z.b. um den text zu formatieren. Wenn der text nicht explizit formatiert wird, wird die für die seite geltende formatierung verwendet.

document.**write***("text");*

beispiel 3f - write

```
<script type="text/JavaScript">
document.write("das ist nur ein hinweis");
document.write("<p>text in <b><i>p-tag</i></b></p>");
var text = "p-tag mit CSS-klasse";
document.write("<p class='font10b'>" + text + "</p>");
document.write("der zeile folgt ein zeilenumbruch
   <br/>");
document.write("ende der vorstellung");
</script>
```



3.4.2 document-eigenschaften

forms	array mit allen formularen derseite	vgl. 4.2	formular-zugriff
links	array mit allen links der seite	vgl. 6.4	links und event
images	array mit allen Image-objekten der seite	vgl. 7.4	objekt image

3.4.3 document-methoden

getElementByID	zugriff auf die ID eines elements der seite	vgl. 5.	HTML-code ändern
event	ereignisse	vgl. 6.3	eventhandler

3.5 objekt screen

Das objekt enthält als eigenschaften informationen über den aktuell verwendeten bildschirm.

screen.**availHeight**

screen.**availWidth**

Geliefert werden als numerische werte die zur verfügung stehende bildschirm-höhe und breite in pixel, also abzüglich taskleiste u.ä..

screen.**height**

screen.**width**

Geliefert werden als numerische werte die bildschirm-höhe und breite in pixel.

screen.**availLeft**

screen.**availTop**

Geliefert werden als numerische werte die x/y-koordinaten des linken obersten bildpunktes.

beispiel 3c - bildschirm-info

```
<script type="text/JavaScript">
    var gesh = screen.height;
    var gesw = screen.width;
    var zeile = "höhe / breite des bildschirms<br />";
    zeile += "verfügbar " + screen.availHeight + " / " ;
    zeile += screen.availWidth + "<br />";
    zeile += "gesamt " + gesh + " / " + gesw + "<br />";
    zeile += "oberster bildpunkt (left/top) " + screen.availLeft;
    zeile += " / " + screen.availTop;
    document.write(zeile);
</script>
```

bildschirm-info

höhe / breite des bildschirms
verfügbar 728 / 1024
gesamt 768 / 1024
oberster bildpunkt (left/top) 0 / 0

3.6 objekt Date

Der in Javascript definierte objekt-typ **Date** bietet mit seinen methoden die möglichkeit, mit datum und uhrzeit zu arbeiten.

3.6.1 datum, uhrzeit abfragen

Es muss zuerst eine instanz des objekt-typs erzeugt werden; es sind mehrere instanzen möglich.

instanz = new **Date**();

Das ergebnis ist ein verweis auf eine instanz des objekt-typs **Date**; die instanz kann dann mit folgenden methoden (auswahl) ausgewertet werden.

methode	***ergebnis***
getDate()	tag 1 - 31
getMonth()	monat 0 - 11 !! also 1 addieren
getYear()	jahr, aber sehr seltsam, wenn das ergebnis < 200 ist, addiert man 1900 und erhält 2000 oder größer, andernfalls ist das ergebnis 1999 oder kleiner
getFullYear()	jahr, vierstellig ohne die umstände wie zuvor
getDay()	nummer des tags 0 - 6, 0 ist Sonntag
getHours()	stunden 0 - 23
getMinutes()	minuten 0 - 59
getSeconds()	sekunden 0 - 59
getMilliseconds()	millisekunden 0 - 999
getTime()	die zeit (in millisekunden), die seit dem 1.1.1970 und dem erzeugen der aktuellen instanz vergangen ist. Das braucht man, wenn man mit datum und / oder uhrzeit rechnen will.

3.6.2 datum, uhrzeit setzen

instanz = new **Date**(*jahr, monat, tag* [, *stunden, minuten, sekunden*]);

Das ergebnis ist ein verweis auf eine instanz des objekt-typs **Date** mit dem angegebenen datum und ggf. der uhrzeit. Die instanz kann man dann mit den vorstehenden methoden auswerten. Das braucht man, wenn man mit datum und / oder uhrzeit rechnen will. Achtung, bei der monatsangabe muss man darauf achten, dass hier die werte 0 - 11 gelten.

beispiel 3d - datum

Das beispiel enthält zwei funktionen; die funktion **tagesdatum** ermittelt das aktuelle datum, bereitet es auf und gibt es als ergebnis zurück.

```
<script type="text/JavaScript">
function tagesdatum()
{
   var erg, tag, mon, jahr;
   var d, m, y;
   d   = new Date();                    //  tagesdatum
   tag = d.getDate();
   m   = d.getMonth();
   mon = monat[m];
   y   = d.getYear();
   if (y < 200)                         //  ab 2000 notwendig
       jahr = y + 1900;
   else
      jahr = y;
   erg = "<p class='font10b'>" + tag + ". " + mon + " " + jahr + "<br />";
   return erg;
}
</script>
```

Die funktion **rechnen** erzeugt die instanz **d0** mit dem aktuellen datum und holt daraus jahr, monat und tag. Damit wird die instanz **d1** erzeugt und daraus (mit getTime) der zeitwert in millisekunden ermittelt (**zeit1**). Das aktuelle jahr wird um 1 vermindert und damit, sowie mit monat und tag die instanz **d2** erzeugt, aus der ebenfalls der zeitwert (**zeit2**) ermittelt wird. Aus der differenz der beiden zeitwerte wird die anzahl der tage errechnet. Man darf für die errechnung des zeitwertes zeit1 nicht die instanz **d0** verwenden, denn da steckt nicht nur das datum, sondern auch die uhrzeit drin, während **d2** nur ein datum enthält. Das ergebnis wäre nicht grundfalsch, es würde halt keine gerade zahl von tagen heraus kommen, sondern eine zahl mit vielen stellen hinter dem komma.

```
<script type="text/JavaScript">
function rechnen()
{
   var d0, d1, d2, y;
   var tag1, mon1, jahr1, zeit1;
   var tag2, mon2, jahr2, zeit2;
   var diff, tage, text;

   d0   = new Date();                  //  tagesdatum
   tag1 = d0.getDate();
   mon1 = d0.getMonth();
   y    = d0.getYear();
   if (y < 200)
      jahr1 = y + 1900;
   else
      jahr1 = y;

   d1 = new Date(jahr1, mon1, tag1);   //  objekt mit tagesdatum
   zeit1 = d1.getTime();               //  datum in millisekunden
   jahr2 = jahr1 - 1;                  //  datum ein jahr vorher
   mon2  = mon1;                       //  in neues objekt
   tag2  = tag1;

   d2    = new Date(jahr2, mon2, tag2);
   zeit2 = d2.getTime();               //  datum in millisekunden
   diff  = zeit1 - zeit2;              //  differenz
   tage  = diff / 1000 / 60 / 60 / 24; //  in tage umrechnen
   mon2++;
   text = "ab " + tag2 + "." + mon2 + "." + jahr2 + " bis heute sind ";
   document.write(text + tage + " tage vergangen <br />");
}
</script>
```

So werden die funktionen auf der seite aufgerufen

```
<p>Guten Tag, heute ist der<br />
<script type="text/JavaScript">
   ergeb = tagesdatum();
   document.write(ergeb);
   rechnen();
</script>
```

und so sieht das ergebnis aus

Guten Tag, heute ist der

12. September 2017
ab 12.9.2016 bis heute sind 365 tage vergangen

3.7 objekt Math

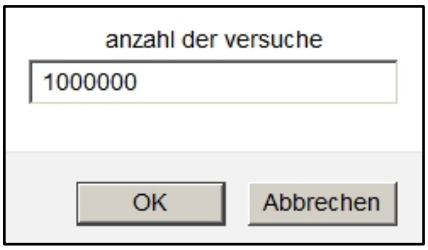
Die methoden dieses objekts dienen für allerlei mathematische berechnungen

Math.**ceil(**wert);	nächste größere ganzzahl π
Math.**floor**(wert);	nächste kleinere ganzzahl
Math.**round**(wert);	kfm. runden auf ganzzahl
Math.**abs**(wert);	absoluter zahlenwert, d.h. ohne vorzeichen
Math.**sqrt**(wert);	quadratwurzel
Math.**pow**(wert, potenz);	wert hoch potenz
Math.**random**();	zufallszahl zwischen 0 und 1
Math.**PI**;	eigenschaft; liefert die kreiszahl **π (Pi)** mit vielen nachkommastellen

beispiel 3e - zufallszahlen

In dem beispiel wird mit der funktion **anfordern** eine zahl eingegeben. Ensprechend dieser zahl wird dann die funktion **zufall** aufgerufen und dabei der wert 6 übergeben. Die funktion liefert dann eine zufallszahl zwischen 1 und 6; die funktion simuliert also einen würfel. Als ergebnis wird aufgelistet, wie oft 1, 2 usw geliefert wurde. Dass hier tatsächlich zufallszahlen geliefert werden erkennt man daran, dass bei verschiedenen ausführungen des beispiels mit gleichem startwert jedes mal andere ergebnisse geliefert werden.

```
<script type="text/JavaScript">
function zufall(n)
{
   var erg;
   erg = Math.random();      //  zahl zwischen 0 und 1
   erg *= n;                 //  * n auf niedr. ganzzahl gerundet
   erg = Math.floor(erg);    //  ergibt ganzzahl 0 bis n - 1
   erg += 1;                 //  + 1 ergibt 1 bis n
   return erg;
}

function anfordern()
{
   var erg = prompt("anzahl der versuche");
   erg = Number(erg);
   return erg;
}
```




```
var x, y, erg, ende;
var summen = new Array(0,0,0,0,0,0,0);
ende = anfordern();
for (x=0; x<ende; x++)
{   erg = zufall(6);
    for (y=1; y<=6; y++)
    {   if (y == erg)
        {   summen[y]++;
            break;
        }
    }
}
document.write("<p>ergebnis mit " + ende + " versuchen<br />");
for (y=1; y<=6; y++)
    document.write(y + ":   " + summen[y] + "<br />");
</script>
```

ergebnis mit 1000000 versuchen
1: 166625
2: 166757
3: 166484
4: 166812
5: 167014
6: 166308

4. formular prüfen

4.1 syntax

Mit Javascript können die eingaben eines formulars vor dem abschicken zur zielseite geprüft und ggf. vervollständigt oder geändert werden. Es ist zweckmäßig, aber nicht unbedingt nötig, dem formular und den formularelementen eindeutige namen zu geben, weil das den zugriff auf das formular und die elemente erleichtert. Das form-tag hat dazu folgende syntax.

<form [name="*formname*"] action="*zielseite*" method="POST | GET" **onSubmit="return *funk*()"**>

Beim abschicken des formulars (submit) wird das ereignis (event) **onSubmit** ausgelöst, d.h. die hier angegebene funktion ***funk*** wird ausgeführt. Die funktion muss den rückgabewert true oder false haben. Bei true wird das formular zur *zielseite* geschickt, bei false unterbleibt dies.

4.2 formular-zugriff

forms - zugriff zum formular

Das objekt **document** hat die eigenschaft **forms** in der form eines array, der referenzen auf alle formulare der seite enthält. Auf diese eigenschaft kann in der funktion **funk** (s.o.) zugegriffen werden. Abhängig davon, ob das formular einen namen hat, gibt es für den zugriff folgende möglichkeiten:

[var *fm* =] document.**forms**[*n*];	zugriff mit index des formulars im array **forms**
[var *fm* =] document.**forms**["f*ormname*"];	zugriff mit dem formularnamen als index im array **forms**
[var *fm* =] document.*formname*;	zugriff mit dem formularnamen

Bei allen folgenden erläuterungen und beispielen steht ***fm*** für eine variable mit einer referenz auf ein formular.

elements – zugriff zu formularelementen

Die referenzen auf die elemente des formulars werden mit der eigenschaft **elements** als array zur verfügung gestellt. Abhängig davon, ob das formular und das element namen haben gibt es verschiedene zugriffsmöglichkeiten:

[var *elem* =] document.**forms**[*n*].**elements**[*n*];
[var *elem* =] *fm*.**elements**[*n*];

[var *elem* =] document.**forms**["*formname*"].**elements**["*elemname*"];
[var *elem* =] *fm*.**elements**["*elemname*"];

[var *elem* =] document.*formname.elemname*

[var *elem* =] *fm.elemname*;

Die eigenschaft **elements** ist ein array mit der die eigenschaft **length**. Die anzahl der elemente im array **elements** erhält man mit:
anzahl = document.**forms**[*n*].**elements**.**length**;
anzahl = document.**forms**["f*ormname*"].**elements.length**;
anzahl = *fm*.**elements.length**;

4.3 formularelement prüfen

zugriff zu eigenschaften

Alle elemente haben die eigenschaft **type**, die weiteren eigenschaften hängen von **type** ab. Den wert einer eigenschaft erhält man mit (jetzt nur die kurzschreibweise):

wert = fm.elemname.**type**;
wert = elemname.**type**;

Für die anderen eigenschaften gilt das entsprechend. Welche eigenschaften vorhanden sind und wie man sie prüft und ggf. ändert hängt vom typ des elements ab.

4.3.1 texteingabefeld

eigenschaft type: "text" | "password" | "textarea"

Geprüft wird hier die eigenschaft **value**, d.h. der eingegebene bzw. der vorbelegte text.

```
if (elemname.value == "")
     var OK = false;
else
{    var OK = true;
     //  ggf jetzt weitere aktionen
}
```

Enthält das formular mehrere texteingabefelder mit gleichem namen, stehen die value-werte in dem array **elemname** zur verfügung, mit **elemname[n]** greift man dann auf die werte zu.

4.3.2 radiobutton

eigenschaft type: "radio"

Hier kann man die eigenschaft **checked** prüfen. Meist bilden mehrere radiobutton eine gruppe und haben einen gemeinsamen namen, die eigenschaften der button stehen dann im array ***elemname***. Um festzustellen, welcher button ausgewählt wurde, schreibt man am besten eine schleife.

```
var OK = false;
for (var x=0; x<fm.elemname.length; x++)
{    if (fm.elemname[x].checked)
     {    OK = true;
          break;
     }
}
```

Die variable **x** enthält die lfd nummer des ausgewählten radiobutton: Wie man an den zugehörigem text der beschriftung kommt, der ja in value stehen müßte, war nicht zu erfahren.

4.3.3 checkbox

eigenschaft type: "checkbox"

Auch hier prüft man die eigenschaft **checked**, allerdings haben mehrere checkboxen keinen gemeinsamen namen und man muss sie der reihe nach alle prüfen, zumal ja mehrere boxen ausgewählt sein können. Dafür ist es wieder einfach an den text der beschriftung zu kommen.

```
if (fm.elemname.checked)
     var text = fm.elemname.value;     //  text der beschriftung
```

4.3.4 auswahlliste

eigenschafr type: "select-one"

Hier macht es keinen sinn, den einzelnen optionen einen namen zu geben, vielmehr gibt man dem select-tag einen namen. Alle optionen stehen im array **options** und haben ggf. die eigenschaft **selected**. Man muss aber nicht diesen array abarbeiten, weil der index der ausgewählten option in der variablen **selectedIndex** angeboten wird. Es genügt also, zu schreiben:

```
if (fm.selectname.selectedIndex == 0)
     var OK = false;
else
{    var auswahl = fm.selectname.selectedIndex;   //  lfd nr der option
     var wert = fm.selectname.options[auswahl];   //  value-wert der option
     var OK = true;
}
```

4.3.5 mehrfach-auswahlliste

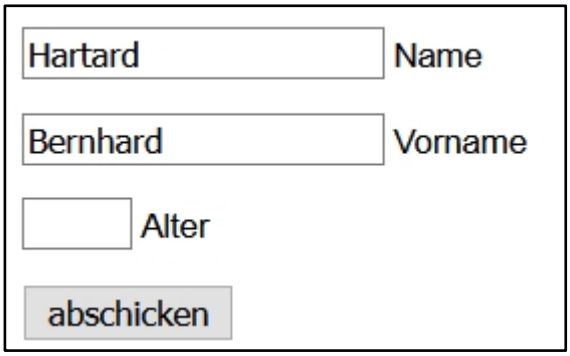
eigenschaft type: "select multiple"

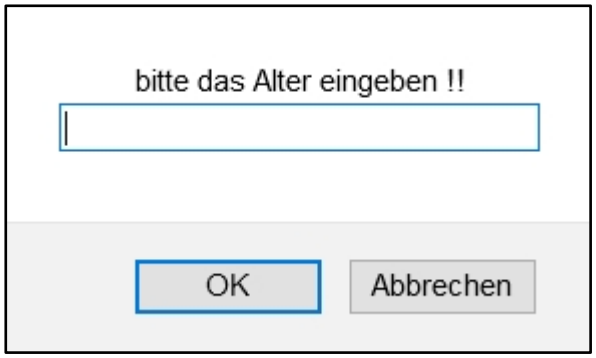
Hier steht der index der **ersten** auswahl in **selectedIndex**, sonst alles wie bei der einfachen auswahlliste.

4.3.6 beispiel 4 - formular prüfen

Das beispiel enthält das folgende formular mit drei eingabefeldern; die vor dem abschicken an die zielseite mit der funktion **check** darauf geprüft werden, ob eine eingabe gemacht wurde. Fehlende eingaben werden mit **prompt** angefordert.

```
<form name="fm1" action="js-test4.php" method="POST"
   onsubmit="return check()">
  <p><input type="text" name="name" size="20" maxlength="40" /> Name</p>
  <p><input type="text" name="vorn" size="20" maxlength="40" /> Vorname</p>
  <p><input type="text" name="alter" size="3" maxlength="3" /> Alter</p>
  <p><input type="submit" name="sender" value="abschicken" /></p>
</form>

<script type="text/JavaScript">
function check()
{   var erg = true;
    var param, x;
    var text = new Array("den Namen", "den Vornamen", "das Alter");
    var fm = document.fm1;                  //  referenz auf formular
    var anz = fm.elements.length;           //  anzahl formular-elemente
    for (x=0; x<anz; x++)                   //  alle elemente prüfen
    {   if (fm.elements[x].type == "text")  //  elementtyp ist text
        {   param = fm.elements[x].value;   //  value des elements
            if (param == "")                //  kein wert vorhanden
            {   param = prompt("bitte " + text[x] + " eingeben !!");
                if ((param == "") || (!param))
                    erg = false;
                else                        //  eingabe prompt übern.
                    fm.elements[x].value = param;
            }
        }
    }
    if (!erg)
        alert("das formular ist nicht vollständig");
    return erg;
}
</script>
```





5. HTML-code ändern

Der HTML-code einer seite wird vom browser dargestellt und die so angezeigte seite ist eigentlich eine statische angelegenheit. Mit Javascript kann man aber das aussehen einer angezeigten seite ändern.

5.1 änderung über ID

Ein HTML-tag kann u.a. das attribut **id** haben, beispielsweise:

```
<p id="tagid" class="font10">das ist ein text</p>
```

Über diese **id** kann man auf das HTML-tag der aktuellen seite (objekt **document**) zugreifen und es verändern, mit der wirkung, dass sich das aussehen der angezeigten seite entsprechen ändert. Der zugriff erfolgt mit der methode **getElementByID** des objekts **document**. Das ergebnis der methode sind verschiedene eigenschaften des HTML-tags. Diese eigenschaften kann man dann ändern. Man kann die eigenschaften, besser gesagt die referenzen auf eigenschaften auch in variablen sichern und damit zu einem späteren zeitpunkt die eigenschaft erneut ändern oder den ursprünglichen zustand wiederherstellen.

[var eigen =] document.**getElementByID**("*tagid*").*eigenschaft*

Die methode greift mit hilfe der element-id auf ein element der seite zu und liefert die referenz auf eine eigenschaft des elements. Mit dieser referenz kann man lesend und schreibend auf die eigenschaft zugreifen.

5.1.1 eigenschaft style

Die eigenschaft ermöglicht den lesenden und schreibenden zugriff zu den styles eines HTML-tags. Man kann direkt auf bestimmte styles zugreifen.

styles lesen
var erg1 = document.getElementByID("*tagid*").**style.color**;
var erg2 = document.getElementByID("*tagid*").**style.background** ;

styles schreiben
document.getElementByID("*tagid*").**style.color** = "*farbe*";
document.getElementByID("*tagid*").**style.background** = "*farbe*";

Hier werden die textfarbe und die hintergrundfarbe gespeichert bzw. geändert. Dabei stört es nicht, wenn in dem angesprochenen tag keine textfarbe oder hintergrundfarbe explizit gesetzt wurde, es ist dann eine browserspezifische standardfarbe gesetzt.

Häufig speichert man auch zunächst die referenz auf die eigenschaft **style** in einer variablen und greift dann auf die verschiedenen styles mit hilfe dieser variablen zu.

```
var stil = document.getElementByID("tagid").style;
erg1 = stil.color;
erg2 = stil.background;

stil.color = "red";
stil.background = "aqua";
```

achtung

Styles wie height, width, top. left usw. sind angaben mit einer masseinheit (z.b. 20px), sie sind also keine numerischen werte sondern zeichenketten. Will man damit rechenoperationen durchführen, muss man den numerischen teil aus der zeichenkette holen, am besten mit **parseInt** (vgl. 1.5). Hat man zur änderung solcher styles einen numerischen wert, muss man ihn in eine zeichenkette umwandeln und dabei die masseinheit anfügen, man schreibt dann **wert + "px"**

5.1.2 eigenschaft className

Styles werden meist mit CSS-klassen vereinbart. Man muss nun keineswegs die styles einzeln ändern, sondern man kann einfach eine andere klasse zuweisen. Styles, die dadurch nicht geändert werden, kann man zusätzlich wie zuvor beschrieben ändern.

erg = document.getElementById("t*agid*").**className**;

document.getElementById("t*agid*").**className** = "*klasse*";

5.1.3 eigenschaft innerHTML

HTML-tags wie hier das p-tag enthalten meist text, der angezeigt wird. Auch dieser text kann geändert werden.

erg = document.getElementById("*tagid*").**innerHTML**;

document.getElementById("*tagid*").**innerHTML** = "*text*";

5.1.4 beispiel 5a - HTML-ändern über ID

Das folgende beispiel ist sehr umfangreich und wird deshalb schrittweise erklärt. Zunächst wird der inhalt der seite (des body) gezeigt; hier gibt es eine kleine grafik und drei p-tags mit text, ein tag steht in einem div-container. Außerdem gibt es zwei links mit einem aufruf der funktionen **change** und **retour**.

```
<body>
<p id="p3" class="font10">das ist ein text</p>
<p id="p2" class="font10">auch das ist ein text</p>
<div id="hin1" style="width: 200px; background: aqua; padding: 10px;">
<p id="p1" class="font10">zeile in container </p>
</div>
<p><img src="im/hexe1.gif" style="height: 100px; width: 100px;" id="hexe"></p>
<p> </p>
<p><a href='Javascript: change();'>ändern</a></p>
<p><a href='Javascript: retour();'>ursprung</a></p>
</body>
```

Die ursprüngliche ansicht wird auf der übernächsten seite gezeigt.

funktion change

Die funktion steht als JS-script im header der seite und sichert zunächst in **globalen** variablen eigenschaften:

p-tag **p3:**	style und innerHTML
p-tag **p2** und **p1:**	style, innerHTML und className
div-tag **hin1:**	style, text- und hintergrundfarbe
grafik **hexe:**	breite und höhe

Vor der funktion stehen die **globalen** variablen

```
var p1style, p1text, p1class, h1back, h1width;
var p2style, p2text, p2class, p3style, p3text;
var hxb, hxh;
```

```
function change()
{   var zeile, hilf;

    p1style = document.getElementById("p1").style;
    p1text  = document.getElementById("p1").innerHTML;
    p1class = document.getElementById("p1").className;
    hilf    = document.getElementById("hin1").style;
    h1back  = hilf.background;
    h1width = hilf.width;
    p2style = document.getElementById("p2").style;
    p2text  = document.getElementById("p2").innerHTML;
    p2class = document.getElementById("p2").className;
    p3style = document.getElementById("p3").style;
    p3text  = document.getElementById("p3").innerHTML;
    hxw = document.getElementById("hexe").style.width
    hxh = document.getElementById("hexe").style.height;
```

Für das p-tag **p3** wird über i**nnerHTML** der text geändert, über die variable zeilakt (enthält referenz auf **style**) werden textfarbe, hintergrundfarbe und die breite geändert; ferner font, d.h. die schriftattribute dicke, stil, größe und schriftart, dabei können nicht die einzelattribute geändert werden, es muß vielmehr die kombinierte angabe gemacht werden (vgl CSS-doku 3.5.1).

Für das p-tag **p2** werden über **innerHTML** der text und über **className** die CSS-klasse geändert. Die variable **zeilakt** enthält eine referenz auf style, damit werden text- und hintergrundfarbe und die breite geändert.

```
    zeile = "dieser text wurde geändert";
    document.getElementById("p3").innerHTML = zeile;
    var zeilakt   = document.getElementById("p3").style;
    zeilakt.font  = "bold italic 30px Courier";
    zeilakt.color = "red";
    zeilakt.background = "aqua";
    zeilakt.width = "500px";

    document.getElementById("p2").innerHTML = zeile;
    document.getElementById("p2").className = "cour22b";
    var zeilakt   = document.getElementById("p2").style;
    zeilakt.color = "red";
    zeilakt.background = "aqua";
    zeilakt.width = "500px";
```

Für das p-tag **p1** im div-container werden über **innerHTML** der text und über **className** die CSS-klasse und mit hilfe der variablen vp1 die text- und die hintergrundfarbe geändert. Für den container selbst werden die hintergrundfarbe und die breite geändert.

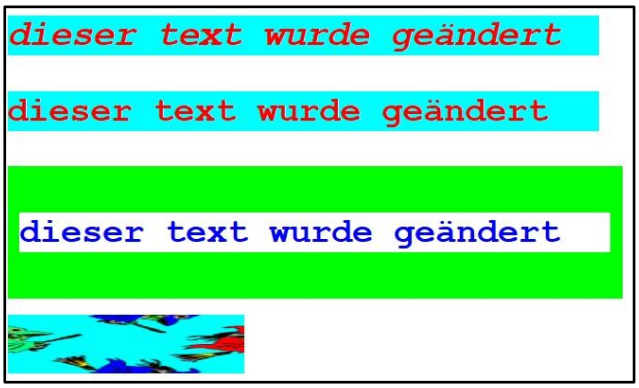
```
    document.getElementById("p1").innerHTML = zeile;
    document.getElementById("p1").className = "cour16b";
    var vp1   = document.getElementById("p1").style;
    vp1.color = "blue";
    vp1.background = "white";
    var cont = document.getElementById("hin1").style;
    cont.background = "lime";
    cont.width = "500px";
```

Für die grafik **hexe** wird in der variablen hilf die breite gespeichert, dann daraus mit **parseInt** der numerische teil herausgezogen und um 100 erhöht und zuletzt das ergebnis als zeichenkette in die breite zurückgeschrieben. Mit der höhe wird entprechend verfahren.

```
    hilf = document.getElementById("hexe").style.width;
    hilf = parseInt(hilf) + 100;
    document.getElementById("hexe").style.width = hilf + "px";
    hilf = document.getElementById("hexe").style.height;
    hilf = parseInt(hilf) - 50;
    document.getElementById("hexe").style.height = hilf + "px";
}
```

ursprüngliche ansicht

ansicht nach änderung



Die funktion **retour** stellt den ursprünglichen zustand wieder her und verwendet dazu die oben gezeigten globalen variablen.

```
function retour()
{   document.getElementById("p1").style = p1style;
    document.getElementById("p1").innerHTML = p1text;
    document.getElementById("p1").className = p1class;
    var hilf = document.getElementById("hin1").style;
    hilf.background = h1back;
    hilf.width = h1width;
    document.getElementById("p2").style = p2style;
    document.getElementById("p2").innerHTML = p2text;
    document.getElementById("p2").className = p2class;
    document.getElementById("p3").style = p3style;
    document.getElementById("p3").innerHTML = p3text;
    document.getElementById("hexe").style.width = hxw;
    document.getElementById("hexe").style.height = hxh;
}
```

5.2 formular ändern

Wie in abschnit 4. gezeigt wurde, erfolgt mit Javascript der zugriff zu formularen und ihren elemente nicht mit dem attribut **id**, sondern mit dem formular- oder den elementnamen. Mit der gleichen technik kann man in der aktuellen seite auch formular-elemente ändern

document.*formname*.*elemname*.**value**	value-wert eines elements
document.*formname*.*elemname*.**size**	größe eines elements
document.*formname*.*elemname*.**className**	styles, die in einer CSS-klasse definiert sind
document.*formname*.*elemname*.**color**	farbe

Es sind hier nicht alle teile eines formular-elements aufgeführt, die geändert werden können, es gibt ja vom typ abhängig sehr unterschiedliche. Es wurde auch noch längst nicht alles ausprobiert, was da vielleicht möglich ist.

beispiel 5b - formular ändern

Das beispiel zeigt ein formular, das nicht an eine zielseite übergeben wird, sondern nur dazu dient, einen hinweis anzuzeigen, der dann geändert wird. Verwendet wird dazu ein einzeiliges eingabefeld mit dem namen **txt**. Das ist eine beliebte methode, um, während eine seite angezeigt wird, auf grund von ereignissen unterschiedliche hinweise anzuzeigen. (Im **beispiel 5c** unter nr. **5.3** gibt es einen ähnlichen fall mit einem mehrzeiligen eingabefeld). Aufgerufen werden auch hier die funktion **change** zum sichern und ändern und die funktion **retour** zum wiederherstellen des ursprünglichen zustands.

```
<form  name="fm1">
<input type="text" name="txt" class="font10" value="text aus formular"
       readonly style="background: aqua">
</form>

<p><a href='Javascript: change();'>ändern</a></p>
<p><a href='Javascript: retour();'>ursprung</a></p>
```

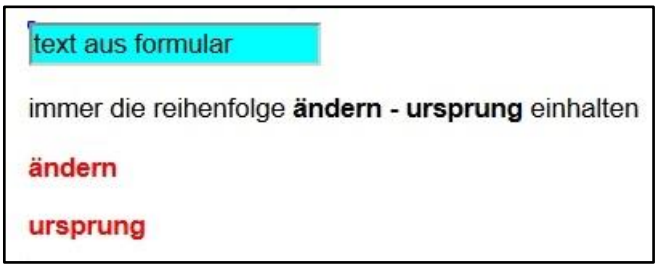
Die funktionen stehen auch hier als JS-script im header der seite. Die funktion **change** sichert für das eingabefeld **value, className** und **size**; size ist im formular nicht ausdrücklich gesetzt, aber dann ist automatisch ein standardwert gesetzt. Außerdem werden gesichert **style.color** und **style.backgrund**. Auch hier ist color nicht gesetzt und es gilt das gleiche wie bei size. Zu erwähnen ist noch, dass es nichts bringt, style selbst zu sichern, man kann damit nichts anfangen. Die anschließend vorgenommenen änderungen sind wohl selbsterklärend.

```
var txttext, txtsize, txtclass, txtback, txtcolor;

function change()
{    var zeile, hilf;
     //   sichern
     txttext  = document.fm1.txt.value;
     txtclass = document.fm1.txt.className;
     txtsize  = document.fm1.txt.size;
     txtback  = document.fm1.txt.style.background;
     txtcolor = document.fm1.txt.style.color;

     //   ändern
     zeile = "dieser text wurde geändert";
     document.fm1.txt.value = zeile;
     document.fm1.txt.className = "cour22b";
     document.fm1.txt.style.background = "lime";
     document.fm1.txt.style.color = "blue";
     document.fm1.txt.size = 30;
}
```

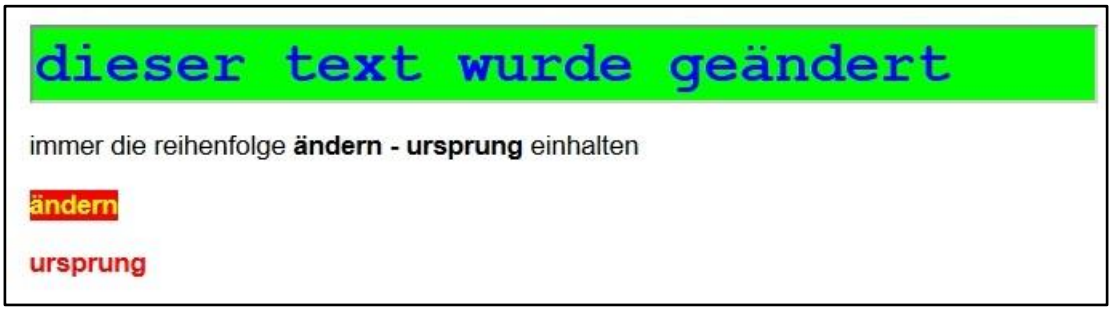
Die funktion **retour** stellt den ursprünglichen zustand wieder her.

```
function retour()
{
     document.fm1.txt.value = txttext;
     document.fm1.txt.className = txtclass;
     document.fm1.txt.style.background = txtback;
     document.fm1.txt.style.color = txtcolor;
     document.fm1.txt.size = txtsize;
}
</script>
```

ursprüngliche ansicht



ansicht nach der änderung



5.3 HTML-elemente ein- / ausblenden

Man kann HTML-elemente mit dem attribut **visibility** unsichtbar auf einer seite darstellen. Das macht nur sinn, wenn man sie gezielt sichtbar und ggf. wieder unsichtbar machen kann. Dazu benötigt man die methode **getElementById** und wiederum die eigenschaft **style**. Beides wurde schon unter 5.1 behandelt. Zu den dort gezeigten möglichkeiten kommt jetzt noch die möglichkeit, ein HTML-element ein- und auszublenden und zu positionieren.

styles lesen
erg = document.getElementById(nr).**style.visibility**;
oben = document.getElementById(nr).**style.top**;
links = document.getElementById(nr).**style.left**;

styles setzen
document.getElementById(nr).**style.visibility** = "visible" | "hidden";
document.getElementById(nr).**style.top** = "nn**px**";
document.getElementById(nr).**style.left** = "nn**px**";

Zu beachten ist auch hier, dass als werte für **top** und **left** zeichenketten anzugeben sind.

beispiel 5c - ein-ausblenden

Das beispiel ist sehr umfangreich und wird daher schritt für schritt erklärt. Es gibt die div-container **tab1** und **tab2**, die beide beim start des beispiels unsichtbar sind, aber mit einer entsprechenden menüauswahl sichtbar gemacht werden können. Der container tab1 enthält den button **ausblenden**, damit macht man ihn wieder unsichtbar, der container tab2 verschwindet nach 15 sekunden automatisch, d.h. beim einblenden des containers wird mit einer zeitgeber-methode bereits das ausblenden gestartet. Aus dem body der seite werden nur die rahmenden div-tags der container und die aufrufe der funktionen zum ein- und ausblenden gezeigt.

```
<div id="tab1" style="position: absolute; top:300px; left: 20px;
     visibility: hidden">
. . .
    <td style="text-align: right;">
        <a href="JavaScript:tabaus('tab1')">ausblenden</a></td>
</div>

<div id="tab2" style="position: absolute; top: 200px; left: 200px;
     visibility: hidden;">
. . .
</div>

<p><a href='Javascript: tabein("tab1");'>
   einblenden</a> <b>tabelle 1</b></p>
<p><a href='Javascript: tabein("tab2");'>
   einblenden</a> <b>tabelle 2</b></p>
```

funktionen zum ein-/ ausblenden; den funktionen wird beim aufruf als argument die ID des containers mitgegeben, der ein- oder ausgeblendet wird.

```
function tabein(nr)
{   var tabakt = document.getElementById(nr).style;
    tabakt.visibility = "visible";
    zeigen(1, nr);                          //  hinweis anzeigen
    if (nr == "tab2")
        setTimeout("tabaus('tab2')", 15000);
}

function tabaus(nr)
{   var tabakt = document.getElementById(nr).style;
    tabakt.visibility = "hidden";
    zeigen(2, nr);                          //  hinweis anzeigen
}
```

container bemerk

Beim ein- und ausblenden der container mit den funktionen **tabein** und **tabaus** wird mit der funktion **zeigen** der container **bemerk** mit verschiedenen hinweisen angezeigt. Die hinweise werden in einem mehrzeiligen eingabefeld eines formular ausgegeben. Die funktion **zeigen** macht den container **bemerk** sichtbar oder unsichtbar, außerdem ändert sie für das mehrzeilige eingabefeld **bem** den wert von **value**, obwohl dieses attribut gar nicht definiert ist, auch gar nicht definiert werden kann. Es ist trotzdem vorhanden. Der funktion wird beim aufruf als argument die nummer des anzuzeigenden hinweises und die ID eines containers mitgegeben.

```
<div id="bemerk" style="position: absolute; top: 100px; left: 700px;
     visibility: visible; ">
<p><b>hinweis</b><br />
<form name="fm2">
<textarea rows="3" name="bem" cols="20"
   readonly>hier werden bei bedarf hinweise ausgegeben</textarea>
</form></p>
</div>

<p class="font10b"><a href="JavaScript: zeigen(0)">hinweis ausblenden</a></p>
```

funktion zeigen im header der seite

```
var bemkz = true;

function zeigen(par1=0, par2)
{   var txt;

    if (par1 == 0)
    {   if (bemkz)
        {   akthin = document.getElementById("bemerk").style;
            akthin.visibility = "hidden";
            bemkz = false;
        }
    }
    else if (par1 == 1)
    {   txt = "A c h t u n g\n";
        txt += "der container\n" + par2 +  " wurde\neingeblendet";
        document.fm2.bem.value = txt;
        akthin = document.getElementById("bemerk").style;
        akthin.visibility = "visible";
        bemkz = true;
    }
    else if (par1 == 2)
    {   txt = "der container\n" + par2 +  " wurde\nausgeblendet";
        document.fm2.bem.value = txt;
        akthin = document.getElementById("bemerk").style;
        akthin.visibility = "visible";
        bemkz = true;
    }
    else
    {   txt = "S T O P\n\nhier geht es nicht weiter";
        document.fm2.bem.value = txt;
        akthin = document.getElementById("bemerk").style;
        akthin.visibility = "visible";
        bemkz = true;
    }
}
```

container uhr – digitale uhr

Es gibt einen weiteren div-container, der mit dem aufruf der funktion **uhr** eingeblendet wird und dann eine im sekundentakt laufende digitaluhr anzeigt.

```
<div id="uhr" style="position: absolute; top: 50px; left: 700px;
     visibility: hidden;">
<p id="uhrzeit">nn:nn:nn</p>
</div>

<p class="font10b"><a href='Javascript: uhr(1);'>start</a>
   uhr einschalten</p>
```

funktion uhr im header der seite

```
function uhr(par)
{   var erg, std, min, sec, zeile;

    erg = new Date();            //  aktuelles datum und zeit
    std = erg.getHours();        //  uhrzeit ermitteln
    if (std < 10)
        std = "0" + std;
    min = erg.getMinutes();
    if (min < 10)
        min = "0" + min;
    sec = erg.getSeconds();
    if (sec < 10)
        sec = "0" + sec;
    zeile = "<b>" + std + ":" + min + ":" + sec + "</b>";
    document.getElementById("uhrzeit").innerHTML = zeile;
    if (par == 1)                //  beim ersten aufruf einblenden
        document.getElementById("uhr").style.visibility = "visible";
    setTimeout("uhr(2)", 1000);
}
```

beweglicher container

Das beispiel enthält den weiteren div-container **tab3**, der nicht ein- oder ausgeblendet, sondern mit den funktionen **huepfen** und **ruckhupf** über den bildschirm bewegt wird. Das ist schon fast animation.

Bei der funktion **huepfen** wird verhindert, dass der container über den linken bildschirmrand verschoben wird, bei der funktion **ruckhupf** wird der container vor dem oberen rand gestoppt. Das muss nicht unbedingt sein, man kann HTML-elemente ziemlich weit in alle richtungen aus dem sichtbaren bereich positionieren, sie gehen dabei nicht verloren. Die bewegung des containers ist ein wenig ungeschickt programmiert, er kommt nie mehr an seine ausgangsposition zurück und läßt sich nach einigem hin und her gar nicht mehr bewegen.

```
<div id="tab3" style="position: absolute; top: 70px; left: 150px;
     visibility: visible;">
. . .
</div>

<p class="font10b"><a href='Javascript: huepfen();'>hupf</a>
   vorwärts</p>
<p class="font10b"><a href='Javascript: ruckhupf();'>hupf</a>
   rückwärts</p>
```

funktionen im header der seite

```
var oben, links;
avw = screen.availWidth;     //  vgl. 3.5

function huepfen()
{
    tabakt = document.getElementById("tab3").style;
    oben = tabakt.top;
    oben = parseInt(oben) + 30;
    links = tabakt.left;
    links = parseInt(links) + 100;
    if (links > avw - 200)
    {   alert("stop");
        links -= 100;
    }
    else
    {   tabakt.top  = oben  + "px";
        tabakt.left = links + "px";
    }
}

function ruckhupf()
{
    tabakt = document.getElementById("tab3").style;
    oben = tabakt.top;
    oben = parseInt(oben) - 20;
    links = tabakt.left;
    links = parseInt(links) - 50;
    if (oben < 0)
    {   alert("stop");
        oben += 20;
    }
    else
    {   tabakt.top  = oben + "px";
        tabakt.left = links + "px";
    }
}
```

6. eventhandler - ereignisse behandeln

6.1 übersicht

Der browser hat eine wichtige aufgabe: er muß alle ereignisse behandeln, die beim laden einer seite, bzw. während eine seite geladen ist, auftreten. Wenn ein ereignis (s.u.) eintritt, aktiviert der dafür vorhandene eventhandler eine entsprechende routine des browsers. Genau an dieser stelle kann man mit Javascript eingreifen, indem man für den eventhandler eine eigene funktion vorgibt, die aufgerufen werden soll.

Von den zahllosen eventhandlern, für die man Javascript-funktionen schreiben kann, werden hier nur die wichtigsten aufgeführt.

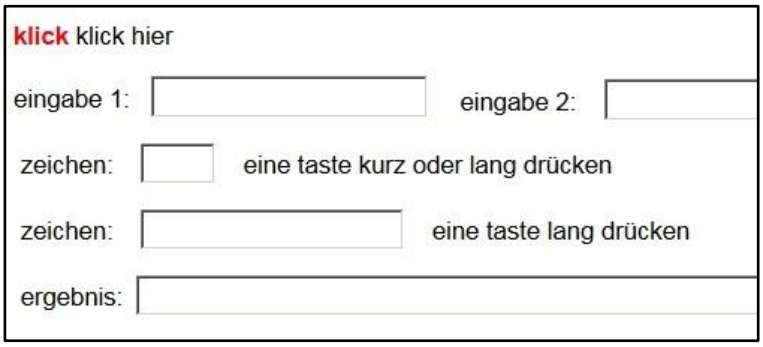
event-handler	***ereignis***	***beispiel***	
onblur	verlust des focus	beispiel 8	eventhandler objektbezogen
onclick	klick mit der maus	beispiel 8 beispiel 9	eventhandler objektbezogen eventhandler dokumentbezogen
ondblclick	doppelklick mit der maus		
onfocus	erhalt des focus	beispiel 8	eventhandler objektbezogen
onkeypress	eine taste drücken	beispiel 9	eventhandler dokumentbezogen
onkeyup	eine gedrückte taste loslassen	beispiel 8	eventhandler objektbezogen
onload	laden einer seite	beispiel 9	eventhandler dokumentbezogen
onmousdown	drücken einer maustaste	beispiel 9	eventhandler dokumentbezogen
onmousemove	bewegung der maus	beispiel 9	eventhandler dokumentbezogen
onmouseout	der mauszeiger verläßt ein objekt		
onmouseover	der mauszeiger über einem objekt	beispiel 9	eventhandler dokumentbezogen
onmouseup	loslassen einer maustaste		
onreset	ein formular wird zurückgesetzt		
onselect	ein element wird ausgewählt		
onsubmit	ein formular wird abgeschickt	beispiel 6	formular prüfen vgl. 4.3.6

6.2 eventhandler objektbezogen

Es gibt eine einfache und weit verbreitete methode, funktionen für eventhandler zur verfügung zu stellen: man ordnet einem element einer seite (auch als objekt bezeichnet) einen oder auch mehrere eventhandler zu und weist jeweils eine bestimmte funktion zu. Wenn man beispielsweise einem formular-element den eventhandler **onfocus** zuordnet, wird die zugewiesene funktion ausgeführt, wenn das element den focus erhält.

beispiel 6a – eventhandler objektbezogen

Das beispiel zeigt die verwendung einiger eventhandler, die jeweils einem bestimmten objekt zugeordnet sind. Es werden nur die teile der seite gezeigt, die für die beschreibung des jeweiligen eventhandlers relevant sind. Die benötigten funktionen stehen im header der seite.



```
<body>
<p><a href="#" onclick="pklick()";>klick</a>klick hier</p>

<form name="fm">
  <p>eingabe 1:  <input type="text" name="T1" size="50"
       onfocus="focusein(1)" onblur="focusaus(1)">
           eingabe 2:   
     <input type="text" name="T2" size="20" onfocus="focusein(2)"
       onblur="focusaus(2)"></p>
  <p> ein zeichen:   
     <input type="text" name="T3" size="3" onkeyup="tastein()" />
        eine taste kurz oder lang drücken</p>
  <p> mehr zeichen:   
     <input type="text" name="T3A" size="20" onkeyup="mehrein()" />
        eine taste lang drücken</p>
  <p> ergebnis:  <input type="text" name="T4"
     size="80" readonly /> </p>
</form>
</body>
```

6.2.1 onclick

Das p-tag mit der ID **P1** enthält einen link, dem der event-handler **onclick** zugeordnet ist und zwar in der art, dass bei einem klick auf das element die funktion **pklick** aufgerufen wird. Man beachte die merkwürdige schreibweise: das attribut **href** hat den wert "#", das muss so sein, sonst funktioniert der aufruf der funktion nicht.

Mit der funktion **pklick** werden die farben und der text des p-tags geändert. Leider geht dabei auch der link verloren, d.h. es erschein am bildschirm ein anderer text mit anderen farben, aber wirkungsvoll anklicken kann man den neuen text nicht.

```
function pklick()
{
   document.getElementById("P1").innerText = " hier wurde geklickt";
   document.getElementById("P1").style.color = "red";
   document.getElementById("P1").style.background = "aqua";
}
```

6.2.2 onfocus, onblur

Die eventhandler **onfocus** und **onblur** sind den feldern **eingabe 1** und **eingabe 2** (ID **T1** und **T2)** zugeordnet, die gleich behandelt werden. Es werden die funktionen **focusein** bzw. **focusaus** aufgerufen. Zur unterscheidung wird der funktion ein parameter übergeben. Die funktion **focusein** wird aufgerufen, wenn man ein eingabefeld anklickt oder den focus per tabulator auf das feld setzt. Die funktion stellt die hintergrundfarbe sicher und ändert sie dann. Außerdem wird bei value ein text eingetragen. Die funktion **focusaus** wird aufgerufen, wenn man den focus von dem eingabefeld nimmt; sie stellt die ursprüngliche hintergrundfarbe wieder her, läßt aber value unverändert.

```
var eing = "";             //  globale variable
var backcol;

function focusein(par)
{  var elem;
   if (par == 1)
       elem = document.fm.T1;
   else
       elem = document.fm.T2;
   backcol = elem.style.background;
   elem.value = "vorgabe";
   elem.style.background = "lime";
}
```

```
function focusaus(par)
{  var elem;
   if (par == 1)
       elem = document.fm.T1;
   else
       elem = document.fm.T2;
   elem.style.background = backcol;
}
```

6.2.3 onkeyup

Den feldern mit der ID **T3** und **T3A** ist der eventhandler **onkeyup** zugeordnet; aufgerufen werden die funktionen **tastein** bzw **mehrein**. Der eventhandler wird ausgelöst, wenn man eine taste drückt und wieder losläßt, d.h. zeichen eingibt während das feld den focus hat. Drückt man dabei eine taste etwas länger, gibt man das zeichen mehrfach ein. Die funktionen sind nun so gebaut, dass die funktion **tastein** wirklich nur ein zeichen übernimmt, während die funktion **mehrein** bis zu 20 zeichen übernimmt. Die eingegebenen zeichen werden in dem feld **ergebnis** (ID **T4**) angezeigt, wenn das feld voll ist wird es gelöscht und ggf neu beschrieben

```
function tastein()
{   var taste, inkey;
    taste = document.fm.T3.value;      //  alle zeichen
    inkey = taste.charAt(0);           //  erstes zeichen übernehmen
    if (eing.length >= 80)             //  bei mehr als 80 zeichen
        eing = "";                     //  ergebnis löschen
    eing += inkey;
    document.fm.T4.value = eing;
    document.fm.T3.value = "";
}

function mehrein()
{   var taste, anz, lang;
    taste = document.fm.T3A.value;    //  alle zeichen
    anz = taste.length;
    if (anz > 20)                     //  maximal 20 zeichen
    {   taste = taste.substr(0,20);
        anz = 20;
    }
    lang = eing.length;
    if (anz + lang >= 80)             //  bei mehr als 80 zeichen
        eing = "";                    //  eingabe löschen
    eing += taste;
    document.fm.T4.value  = eing;
    document.fm.T3A.value = "";
}
```

6.3 eventhandler dokumentbezogen

6.3.1 startfunktion

Anspruchsvoller ist es, mit eventhandlern zu arbeiten, die auf das aktuelle dokument, d.h. die gerade geladene seite bezogen sind. Beispielsweise soll jedesmal eine funktion ausgeführt werden, wenn an einer beliebigen stelle der angezeigten seite mit der maus geklickt wird oder wenn der mauszeiger bewegt wird usw. In diesem fall müssen entsprechende vereinbarungen schon beim laden der seite getroffen werden, d.h. man benötigt eine funktion, die beim laden der seite ausgeführt wird. Diese funktion wird im header der seite definiert. Der aufruf der funktion erfolgt im body-tag, das mit dem eventhandler **onload** verbunden wird oder im header.

```
<body onload="init()">
```

oder spezielle anweisung im header.

```
window.onload = init;
```

Zu beachten ist, dass bei der anweisung im header der name der funktion ohne die runden klammern anzugeben ist.

In der startfunktion wird für jeden benötigten eventhandler die zugehörige funktion vereinbart. Auch hier wird der name der funktion ohne die runden klammern angegeben.

document.***eventhandler*** = ***funktion***;

6.3.2 funktionen für eventhandler

Eine funktion, die von einem eventhandler ausgeführt wird, erhält als parameter eine referenz auf das objekt, das den eventhandlers auslöst. Die bezeichnung des parameters kann frei gewählt werden.

function *name*(*par*)

Mit hilfe des parameters kann auf die eigenschaften des objekts zugegriffen werden. Hier eine auswahl der wichtigsten eigenschaften

eigenschaft	***bedeutung***	
type	typ des events als zeichenkette	
clientX, clientY	horizontale und vertikale koordinaten des objekts in der seite	
pageX, pageY	alternative zu clientX, clientY	
screenX, screenY	horizontale und vertikale koordinaten am bildschirm	hinweise beachten
button	betätigte maustaste: 0 - links, 1 - mitte, 2 - rechts	
which	betätigte maustaste: 1 - links, 2 - mitte, 3 – rechts	
which charCode keyCode key	siehe hinweis siehe hinweis siehe hinweis siehe hinweis	
altKey	true wenn Alt-taste zusammen mit taste betätigt	nicht getestet
ctrlKey	true wenn Strg-taste zusammen mit taste betätigt	nicht getestet
shiftKey	true wenn umschalt-taste zusammen mit taste betätigt	nicht getestet
target target.name target.id	verweis auf das tag, das den event ausgelöst hat dazu gibt es zwei eigenschaften name des tag, falls vorhanden id des tag, falls vorhanden	

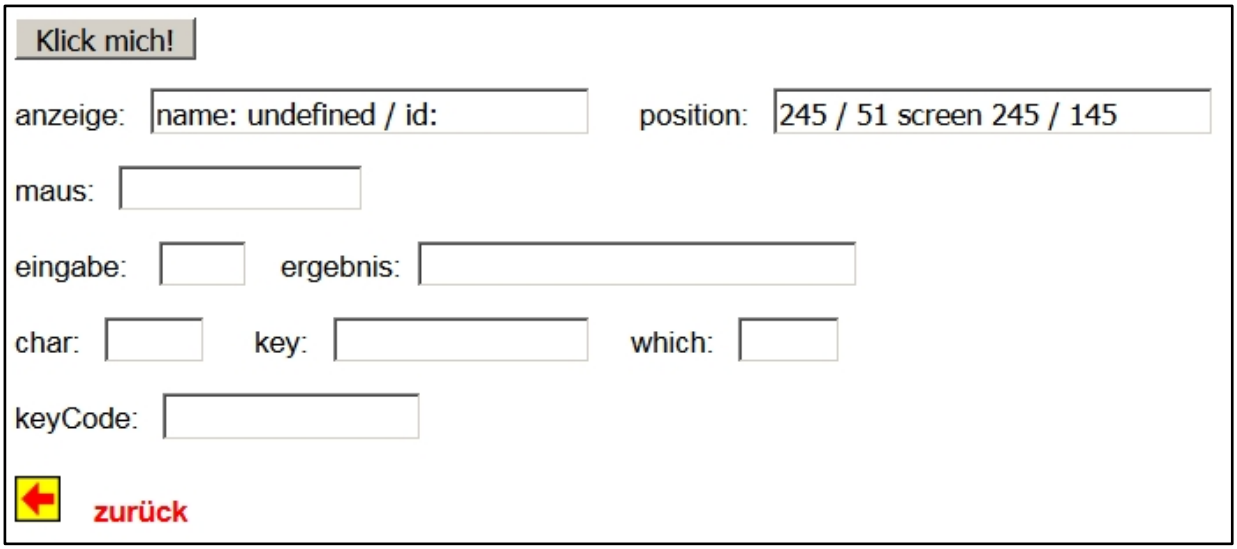
hinweise

Die eigenschaften which, charCode, keyCode, key liefern browserabhängig unterschiedliche ergebnisse, dabei ist zu unterscheiden zwischen tasten, die ein zeichen erzeugen, funktionstasten und der entertaste. Die browser **Chrome** und **Internet Explorer reagieren** auf funktionstasten gar nicht, zumindest nicht mit einem bekannten event. Aus der tabelle läßt sich ableiten, dass man auf die auswertung von funktionstasten wohl verzichten muss und ansonsten am besten mit **key** oder **which** arbeitet.

browser	***taste***	***charCode***	***key***	***which***	***keyCode***
Firefox	zeichentaste	dezimalcode	zeichen	dezimalcode	0
	funktionstaste	0	bezeichnung	0	dezimalcode
	Enter-taste	0	bezeichnung	dezimalcode	dezimalcode
Chrome Internet Explor.	zeichentaste	dezimalcode	zeichen	dezimalcode	dezimalcode
	funktionstaste	kein event	kein event	kein event	kein event
	Enter-taste	dezimalcode	bezeichnung	dezimalcode	dezimalcode

6.3.3 beispiel 6b - eventhandler dokumentbezogen

Das beispiel zeigt eine seite mit einem umfangreichen formular an.



- Alle elemente der seite haben einen namen, die kleine pfeilgrafik eine ID.
- Die elemente des formulars sind eingabefelder; mit ausnahme des feldes **eingabe** haben sie aber den status readonly, d.h. es kann etwas angezeigt, aber nichts eingegeben werden.
- Das feld **anzeige** zeigt an, in welchem element sich der mauszeiger befindet und ob eine maustaste betätigt wurde, das feld **maus** zeigt, welche maustaste betätigt wurde und das feld **position** zeigt die koordinaten des mauszeiger.
- Im feld **eingabe** kann man tatsächlich etwas eingeben, die eingabe wird im feld **ergebnis** gezeigt.
- In den feldern **char, key, which** und **keyCode** wird angezeigt, welche taste betätigt wurde, soweit das möglich ist. Angezeigt entweder das zeichen oder der zeichencode (vgl. hinweise).

```
<body onload="init()">
<form name="fm">
  <input type="button" name="button" value="Klick mich!">
  <p>anzeige:   <input type="text" name="T1"
     size="30" readonly>     
     position:   <input type="text" name="T4" size="30"
     readonly></p>
  <p>maus:   <input type="text" name="T5"
     size="15" readonly>     
  <p>eingabe:   
     <input type="text" name="T3" size="3" />   
     ergebnis:  <input type="text" name="T2" size="30" readonly /> </p>
  <p>char:   <input type="text" name="ch"
     size="4" readonly>     
     key:   <input type="text" name="ky" size="16"
     readonly>    
     which:   <input type="text" name="wi" size="4"</p>
  <p>keyCode:   <input type="text" name="kc"
     size="16" readonly></p>
</form>

<p><a href="Javascript: history.back()">
   <img src="im/n-links.gif" width="21" height="21" id="raus" />
   </a>  <a href="Javascript: history.back()" name="link">
   zurück</a></p>
</body>
```

onload - funktion init
Beim laden der seite wird die durch den eventhandler **onload** die funktion **init** aufgerufen, mit der für einige eventhandler funktionen vereinbart werden. Außerdem wird in das feld **anzeige** (ID T1) ein text eingetragen, der aber nie sichtbar wird, weil das feld sofort beim laden der seite überschrieben wird.

```
function init()
{
   document.onmousedown = mdown;
   document.onclick     = mclick;
   document.onmouseover = wechsel;
   document.onmousemove = mmove;
   document.onkeypress  = eingabe;
   document.forms.fm.T1.value = "info";
}
```

6.3.4 onmousemove - funktion mmove
Jede bewegung des mauszeigers löst den eventhandler **onmousemove** aus, der die funktion **mmove** aufruft. Die funktion liefert fortlaufend die position des zeigers und zeigt sie im feld **position** (ID T4) an.

```
function mmove(par)
{   var cl, ln, sccl, scln, pos;

    cl = par.clientX;
    ln = par.clientY;
    cl = Math.round(cl);   //  nötig für
    ln = Math.round(ln);   //  Internet Explorer
    sccl = par.screenX;
    scln = par.screenY;
    pos = cl + " / " + ln + " screen " + sccl + " / " + scln;
    document.forms["fm"].T4.value = pos;
}
```

6.3.5 onmouseover - funktion wechsel

Fast alle elemente (tags) der seite haben einen **namen** oder eine **id**. Wenn der mauszeiger über ein element geführt wird, löst das den eventhandler **onmouseover** aus, der die funktion **wechsel** aufruft. Die funktion liefert den namen oder die ID des elements. Bei unbenannten tags oder im "freien" raum der seite wird als name **undefined** gemeldet. Die anzeige erfolgt im feld **anzeige** (ID T1).

```
function wechsel(par)
{
    ziel = par.target;
    beznam = ziel.name;
    bezid  = ziel.id;
    if (bezid == "raus" || beznam == "link")
    {   zeile = "ausgang name: " + beznam + " / id: " + bezid;
        document.forms.fm.T1.value = zeile;
    }
    else                                //  alle andern
    {   zeile = "name: " + beznam + " / id: " + bezid;
        document.forms.fm.T1.value = zeile;
    }
}
```

6.3.6 onmousedown - funktion mdown

Wenn eine maustaste gedrückt wird, löst das den eventhandler **onmousedown** aus, der die funktion **mdown** aufruft. Die funktion ermittel mit der eigenschaft **which** informationen über den mauszeiger und zeigt im feld **maus** (ID T5) welche maustaste gedrückt wurde und im feld **anzeige** (ID TZ1) den event-typ und, falls vorhanden, den namen oder die ID des elements, auf dem der mauszeiger gedrückt wurde.

```
function mdown(par)
{
    maus = par.which;
    if (maus == 1)
        maus += " - linke taste";
    else if (maus == 2)
        maus += " - mittlere taste";
    else if (maus == 3)
        maus += " - rechte taste";
    else
        maus += " - ????";
    document.fm.T5.value = maus;
    ziel = par.target;
    beznam = ziel.name;
    bezid  = ziel.id;
    zeile = " " + par.type + "  " + beznam + " / " + bezid;
    document.fm.T1.value = zeile;
}
```

6.3.7 onclick - funktion mclick

Wenn eine maustaste geklickt wird (drücken und loslassen), löst das den eventhandler **onclick** aus, der die funktion **mclick** aufruft. Die funktion gleicht der funktion **mdown**, mit dem kleinen unterschied, dass die info über die maustasten aus der eigenschaft **button** gewonnen wird. Da einem mausklick zwingend ein drücken einer maustaste vorangeht, werden die beiden funktionen immer nacheinander ausgeführt.

```
function mclick(par)
{
    maus = par.button;
    if (maus == 0)
        maus += " - linke taste";
    else if (maus == 1)
        maus += " - mittlere taste";
    else if (maus == 2)
        maus += " - rechte taste";
    else
        maus += " - ????";
    document.fm.T5.value = maus;
    ziel = par.target;
    beznam = ziel.name;
    bezid  = ziel.id;
    zeile = " " + par.type + "  " + beznam + " / " + bezid;
    document.fm.T1.value = zeile;
}
```

6.3.8 onkeypress - funktion eingabe

Wenn eine taste gedrückt, d.h. ein zeichen eingegeben wird, löst das den eventhandler **onkeypress** aus, der die funktion **eingabe** aufruft. Die funktion liefert den wert der eigenschaften **charCode, key, which** und **keyCode** in den entsprechenden feldern. Wenn **which** einen wert > 0 , d.h. den dezimalcode eines zeichens oder der Enter-taste liefert und die eingabe in feld **eingabe** (ID T3) erfolgt, wird der wert mit **CharCode** in das entsprechende zeichen umgewandelt und das zeichen in das feld **ergebnis** (ID T1) eingetragen. Das eingabe-feld wird gelöscht. Beim versuch an einer anderen stelle der seite ein zeichen einzugeben, erfolgt eine fehlermeldung.

```
function eingabe(par)
{   var inkey;

    ziel  = par.target;
    document.fm.T5.value = " ";        //  maus-anzeige löschen
    document.forms.fm.ch.value = par.charCode;
    document.forms.fm.ky.value = par.key;
    document.forms.fm.wi.value = par.which;
    document.forms.fm.kc.value = par.keyCode;
    if (par.which != 0)
    {   taste = par.which;                //  tastencode
        if (ziel.name == "T3")            //  eingabe nur in T3
        {   inkey = String.fromCharCode(taste);
            if (eing.length >= 20)        //  bei mehr als 20 zeichen
                eing = "";                //  eingabe löschen
            eing += inkey
            document.forms.fm.T2.value = eing;
            document.forms.fm.T3.value = " ";
        }
        else
            document.forms.fm.T1.value = "hier nichts eingeben";
    }
}
```

6.4 array document.links

Für jede seite enthält das objekt **document** die eigenschaft **links**, die in der form eines array informationen über alle links auf der seite enthält.

6.4.1 array links

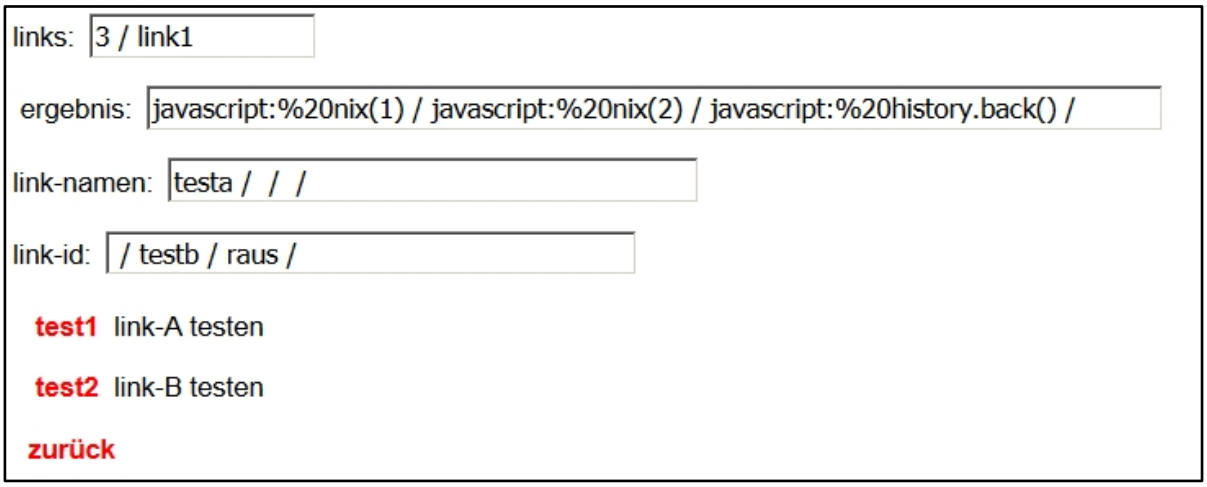
beispiel 6c - array links

Das beispiel hat noch gar nichts mit eventhandler zu tun, es enthält drei links, wovon zwei die funktion **nix** aufrufen. Alle links haben einen namen oder eine id, das dient hier nur ein wenig zum spielen..

```
<form name="fm">
  <p>links:  <input type="text" name="T7"
     size="15" readonly /> </p>
  <p> ergebnis:  <input type="text" name="T8"
     size="80" readonly /> </p>
  <p>link-namen:  <input type="text" name="T5"
     size="40" readonly /> </p>
  <p>link-id:  <input type="text" name="T6"
     size="40" readonly /> </p>
</form>
<p>  <a href="Javascript: nix(1)" name="testa" >
   test1</a>  link-A testen</p>
<p>  <a href="Javascript: nix(2)" id="testb" >
   test2</a>  link-B testen</p>
<p>  <a href="Javascript: history.back()"
   id="raus">zurück</a></p>
```

Die funktion **nix** ermittelt die anzahl der vorhandenen links und zeigt dies an. Dann zeigt sie, auf was die links verweisen und den namen bzw die id der links. Bei der anzahl der links wird noch angezeigt, von welchem link die funktion aufgerufen wurde. Braucht man so was wirklich ?

```
<script type="text/JavaScript">
function nix(par)
{   var x, anz, zeile, zeilea, zeileb;

    anz = document.links.length; //  anzahl der links
    zeile  = "";
    zeilea = "";
    zeileb = "";

    for (x=0; x<anz; x++)
    {   zeile += document.links[x].href + " / ";
        zeilea += document.links[x].name + " / ";
        zeileb += document.links[x].id + " / ";
    }
    if (par == 1)
        anz += " / link1";
    else if (par == 2)
        anz += " / link2";
    else
        anz += " / unbenannt";
    document.fm.T7.value = anz;
    document.fm.T8.value = zeile;
    document.fm.T5.value = zeilea;
    document.fm.T6.value = zeileb;
}
</script>
```



6.4.2 array links und event

Einzelne links aus dem array **document.links** können mit der funktion eines eventhandlers verbunden werden. Wenn das tag mit dem link einen namen oder eine id hat, kann darüber der link im array angesprochen werden, andernfalls geht es über den index. Eine bedauerliche einschränkung gibt es: der Internet Explorer kennt diese möglichkeit nicht.

document.links["*name*" | "*id*" | *index*].*event* = *funktion*;

beispiel 6d - array links und event

```
<body onload="init()">
<form name="fm">
  <p>link-href   <input type="text" name="T1"
     size="80" readonly></p>
  <p>link-name:  <input type="text" name="T2"
     size="30" readonly /> </p>
  <p>link-id:  <input type="text" name="T3"
     size="30" readonly /> </p>
  <p>anzeige:  <input type="text" name="T4"
     size="30" readonly /> </p>
</form>
<p>  <a href="Javascript: mist()" name="test1">test</a>
     link-1 testen</p>
<p>  <a href="Javascript: mist()" id="test2">test</a>
     link-2 testen</p>
<p>  <a href="Javascript: mist()" id="test3">test</a>
     link-3 testen</p>
```

Beim laden der seite wird die funktion **init** aufgerufen, mit der für drei links der eventhandler **onclick** mit funktionen vereinbart wird. Man beachte, dass ein link einen namen hat und die beiden anderen eine id haben. Das ziel der links ist die funktion **mist**, die zwar nach der event-funktion ausgeführt wird, aber nichts weiter tut als einen bedeutungslosen text auszugeben..

```
function init()
{  document.links["test1"].onclick = linktest;
   document.links["test2"].onclick = linktest;
   document.links["test3"].onclick = linksprung;
}
```

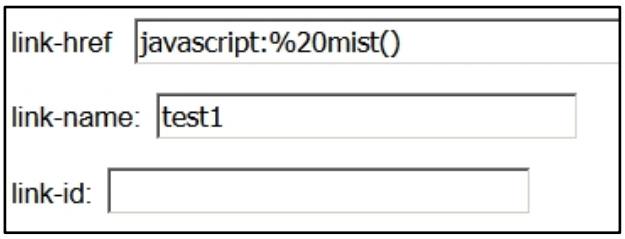
Die funktion **linktest** ermittel das ziel des links (href) und name bzw. id des tags, das den link enthält und zeigt die ergebnisse an.

```
function linktest(par)
{    ziel = par.target;
     document.fm.T1.value = ziel.href;
     document.fm.T2.value = ziel.name;
     document.fm.T3.value = ziel.id;
}
```

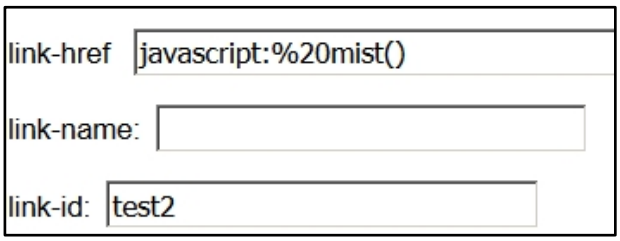
Die funktion **linksprung** tut zunächst das gleiche wie **linktest**, ändert dann aber das ziel des links ab und ruft das ziel auf. Die funktion **mist** wird dann natürlich nicht ausgeführt.

```
function linksprung(par)
{    ziel = par.target;
     document.fm.T1.value = ziel.href;
     document.fm.T2.value = ziel.name;
     document.fm.T3.value = ziel.id;
     ziel.href = "js-test4-hilf.php";
}
```

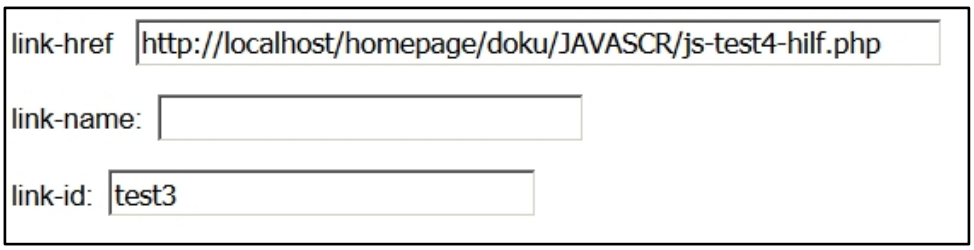
anzeige nach nach test link-1



anzeige nach test link-2



anzeige nach test link-3
Wenn der link test-3 ausgelöst wird, wird die seite js-test4-hilf.php angezeigt. Erst nach der rückkehr von dieser seite sieht man die folgende anzeige.



7. objekt Image

7.1 objekt vereinbaren

Eine grafik ist eine instanz des objekt-typs **Image** und wird mit dem **img**-tag in eine seite gestellt Die grafik wird entweder mit der tatsächlichen größe oder mit der vereinbarten breite und höhe angezeigt. Breite und höhe werden mit dem **style**-attribut vereinbart, als maßeinheit (me) verwendet man am besten pixel (**px)**. Außerdem benötigt man ggf. das **namen-** oder das **id**-attribut und manchmal sogar beides; je nach bedarf kann man für beide die gleiche bezeichnung wählen.

Für jede seite hat das objekt **document** die eigenschaft **images** in der form eines array mit informationen über die vorhandenen **Image-objekte**. Dadurch ergeben sich drei möglichkeiten, die grafiken einer seite anzusprechen:

document.***name***
document.images["***name***"]
document.images[***index***]

Dass es sich bei einer grafik um ein instanz des objekt-typy **Image** handelt, wird bei den verschiedenen schreibweisen nicht klar erkennbar und es ist meist nicht nötig, für eine grafik explizit eine instanz zu vereinbaren. Für eine reihe der nachstehend gezeigten basteleien mit Javascript ist es aber oft zweckmäßig. Bei dieser vereinbarung einer instanz des objekt-typs **Image** wird der eigenschaft **src** der pfadname einer grafik zugewiesen. Diese methode hatte früher große bedeutung für das sog. vorladen von grafiken und wird heute immer noch im zusammenhang mit Javascript angewendet.

```
var grafik = new Image();
grafik.src = "pfadname";
```

7.2 bildwechsel

Eine einfache anwendung von Javascript ist der sog. bildwechsel, d.h. in einer angezeigten seite wird durch eine aktion eine grafik ausgetauscht.

beispiel 7a - bildwechsel

Das beispiel zeigt drei möglichkeiten:

- Mit einem **link** wird die funktion **wechsel1** aufgerufen, die den bildwechsel durchführt.
- Durch anklicken der grafik wird der eventhandler **onclick** ausgelöst, der die funktion **wechsel2** aufruft.
- Wenn der mauszeiger über die grafik geführt wird, bzw. die grafik wieder verläßt wird der eventhandler **onmouseover** bzw. **onmouseout** ausgelöst. In beiden fällen wird die funktion **wechsel3** aufgerufen. Ein parameter teilt der funktion mit, welcher eventhandler aktiv war.

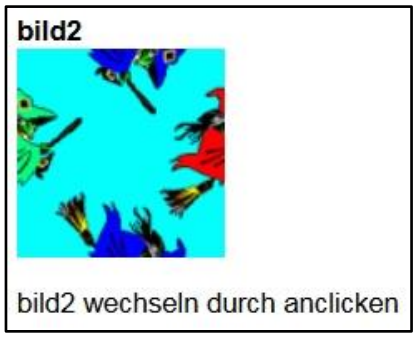
Auf der seite gibt es drei img-tags, d.h. es gibt drei instanzen des objekt-typ **Image**. Die grafiken müssen nicht alle die gleiche größe haben. Die angabe von breite und höhe wie bei der zweiten vereinbarung ist nicht nötig, aber wenn vorhanden sollte sie zur größe der grafik passen, weil sonst ggf. eine grafik verzerrt angezeigt wird. Die funktionen zeigen die drei möglichkeiten, die objekte mit den grafiken anzusprechen. Der wechsel der grafiken erfolgt dadurch, dass der eigenschaft **src** des **img**-tags jeweils eine andere grafik zugewiesen wird.

```
<p><b>bild1</b><br /><img src="im/hexe1.gif" name="bild1" /></p>
<p class="font10"><a href="JavaScript: wechsel1();">
     bild1 wechseln</a></p>

<p><b>bild2</b><br /><img src="im/hexe1.gif" name="bild2"
   style="width: 100px; height: 100px;" onclick="wechsel2();" /></p>

<p><b>bild3</b><br /><img src="im/hexe1.gif" name="bild3"
   onmouseover="wechsel3(0);" onmouseout="wechsel3(1);" /></p>
```

Die variable **kipp1** wird beim laden der seite auf 0 gesetzt und dann bei jedem aufruf umgestellt; sie zeigt damit an, welche grafik angezeigt wird. Das **img**-tag wird direkt mit seinem **namen** angegeben.

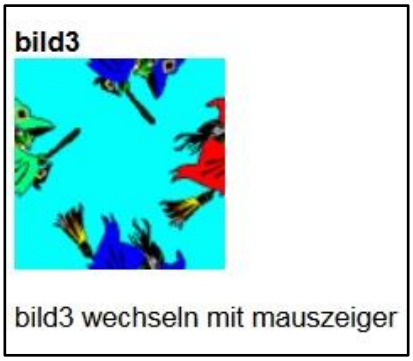

```
var kipp1 = 0;
function wechsel1()
{
   if (kipp1 == 0)
   {   document.bild1.src="im/hexe0.gif";
       kipp1 = 1;
   }
   else
   {   document.bild1.src="im/hexe1.gif";
       kipp1 = 0;
   }
}
```

Die variable **kipp2** hat die gleiche funktion, wie bei der vorigen funktion. Hier dient der **name** des **img**-tags als index im array der eigenschaft **images**.




```
var kipp2 = 0;
function wechsel2()
{
   if (kipp2 == 0)
   {   document.images["bild2"].src="im/hexe0.gif";
       kipp2 = 1;
   }
   else
   {   document.images["bild2"].src="im/hexe1.gif";
       kipp2 = 0;
   }
}
```

Der funktion wird der parameter **kipp3** übergeben, der aussagt von welchem eventhandler die funktion aufgerufen wurde. Das **img**-tag mit dem namen **bild3** ist das dritte auf der seite. Im array **images** wird das tag entsprechend indiziert. Es wäre hier nicht nötig, dass das tag einen namen hat.




```
function wechsel3(kipp3)
{
   if (kipp3 == 0)            //  onmouseover
       document.images[2].src="im/hexe0.gif";
   else                       //  onmouseout
       document.images[2].src="im/hexe1.gif";
}
```

7.3 galerien

Bei einer galerie wird bei der anzeige nicht zwischen zwei verschiedenen grafiken gewechselt, sondern es werden nacheinander mehrere grafiken angezeigt. Auch hier gibt es verschiedene möglichkeiten die dazu notwendigen funktionen zu gestalten. Die folgenden beispiele zeigen jeweils eine möglichst einfache konstruktion. In der praxis können funktionen erheblich komplizierter sein, wenn beispielsweise zu den verschiedenen grafiken noch unterschiedliche unterschriften oder titel angezeigt werden sollen oder erkennbar sein soll, ob man eine galerie im aktuellen zustand in beide richtungen oder nur vorwärts oder rückwärts durchblättern kann u.ä.

Die beispiele verwenden zumeist die methode **eval** (siehe2.3).

einfache galerien

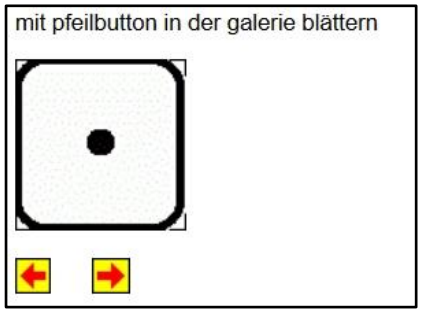
beispiel 7b - galerien typ 1, typ 2 und typ 3

Gemeinsam ist diesen typen, dass für die grafiken explizit instanzen des objekt-typs **Image** eingerichtet werden. Außerdem haben die instanzen einen namen nach dem muster grafik***n***, wobei ***n*** eine mit 0 beginnende fortlaufende nummer ist. Das geht auch anders, aber dann wird es sofort sehr viel komplizierter.

```
<script type="text/JavaScript">
var grafik0 = new Image();
grafik0.src = "im/hexe0.gif";
. . .
var grafik5 = new Image();
grafik5.src = "im/wappen.jpg";
</script>
```

7.3.1 galerie typ 1 - blättern mit link

Bei dieser galerie sind zwei kleine button vorhanden, die jeweils über einen link eine funktion aufrufen, mit der vorwärts oder rückwärts geblättert werden kann. Wenn dabei der anfang oder das ende der galerie erreicht ist, löst der entsprechende button keine aktion aus. Der funktion **galerie1** wird der parameter **richt** übergeben, der aussagt, ob vorwärts (rechts) oder rückwärts (links) geblättert werden soll. Die variable **pos1** sagt aus, welche grafik gezeigt werden soll, sie wird beim laden der seite auf 0 gesetzt und dann bei jedem aufruf entsprechend neu gesetzt.




```
<p>mit pfeiltasten in der galerie blättern</p>
<p><img src="im/hexe0.gif" name="gal1" /></p>
<p><a href="JavaScript: galerie1(1);"><img src="im/n-links.gif" /></a>
       
   <a href="JavaScript: galerie1(2);"><img src="im/n-rechts.gif" /></a></p>

<script type="text/JavaScript">
var pos1 = 0;
function galerie1(richt)
{
    if (richt == 1)                    //  nach links
    {   if (pos1 > 0)
        {   pos1--;
            document.gal1.src = eval("grafik" + pos1 + ".src");
        }
    }
    else                               //  nach rechts
    {   if (pos1 < 4)
        {   pos1++;
            document.gal1.src = eval("grafik" + pos1 + ".src");
        }
    }
}
</script>
```

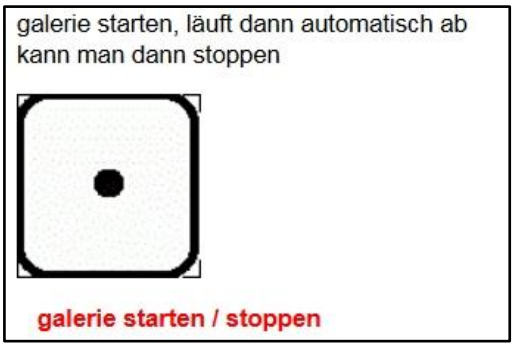
7.3.2 galerie typ 2 - blättern mit onclick

In der galerie wird geblättert, indem man die angezeigte grafik anklickt und damit den eventhandler **onclick** auslöst, der die funktion **galerie2** aufruft. Es wird nur vorwärts geblättert, wenn das ende der galerie erreicht wird, geht es beim anfang weiter. Die galerie enthält eine grafik mit abweichender größe. Abhängig von der variablen **pos2** wird eine grafik angezeigt. Die variable wird beim laden der seite auf 0 gesetzt und dann bei jedem aufruf um 1 erhöht bzw. wieder auf 0 zurückgesetzt.


```
<p>in der galerie blättern<br />durch anklicken</p>
<p><img src="im/hexe0.gif" name="gal2" onclick="galerie2();" /></p>

<script type="text/JavaScript">
var pos2 = 0;
function galerie2()
{
    if (pos2 < 5)
        pos2++;
    else
        pos2 = 0;
    document.gal2.src = eval("grafik" + pos2 + ".src");
}
</script>
```

7.3.3 galerie typ 3 - blättern mit zeitgeber

Mit einem link wird die funktion **galerie3** aufgerufen, die das blättern per zeitgeber startet. Ein erneuter aufruf per link stoppt das blättern. Auch hier enthält die galerie eine grafik mit abweichender größe, im img-tag ist aber eine unpassende breite und höhe angegeben, daher wird diese grafik verzerrt angezeigt.



Die funktion **galerie3** ist etwas komplizierter konstruiert: Beim laden der seite wird die variable **anf** auf true gesetzt. Die variable **pos3** zeigt an, welche grafik zuletzt angezeigt wurde, beim start der seite wird der variablen noch kein wert zugewiesen. Der funktion wird der parameter **par** übergeben, der aussagt, ob die funktion per link (0) oder per zeitgeber (1) aufgerufen wurde.

Wird die funktion per link aufgerufen und **anf** ist true, wird **anf** auf false und **pos3** auf 1 gesetzt. Dann wird die **grafik1** angezeigt und die funktion mit dem zeitgeber aufgerufen.

Wird die funktion per link aufgerufen und **anf** hat den wert false, wird der zeitgeber angehalten, die **grafik0** wird angezeigt und **anf** wird auf true gesetzt.

Wird die funktion per zeitgeber aufgerufen, wird **pos3** um 1 erhöht oder, wenn das ende der galerie erreicht ist, auf 1 gesetzt, die entsprechende grafik angezeigt und die funktion erneut per zeitgeber aufgerufen.

```
<p><img src="im/hexe0.gif" name="gal3"
   style="width: 100px; height: 100px;" /></p>
<p><a href="Javascript: galerie3(0)">  
   galerie starten / stoppen</a></p>
```

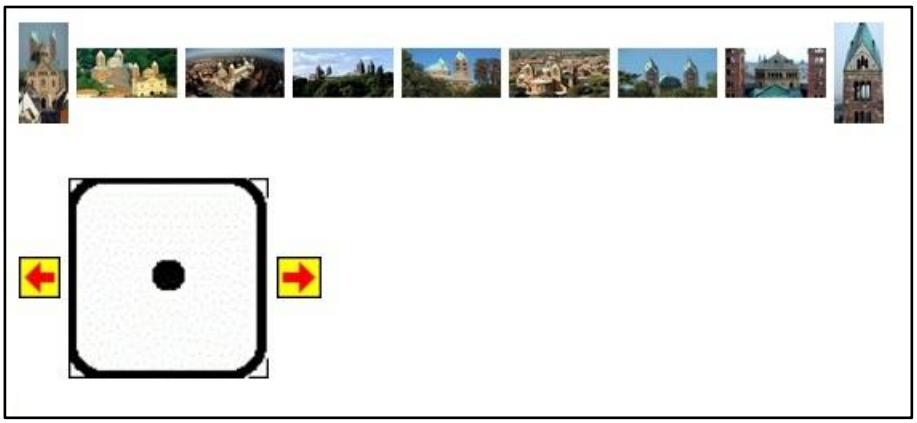
funktion galerie3

```
<script type="text/JavaScript">
var pos3;
var anf = true;
var ID3;
function galerie3(par)
{
    if (par == 0)                       //  start/stop betätigt
    {   if (anf)                        //  galerie starten
        {   anf = false;
            pos3 = 1;                   //  erste bild
            document.gal3.src = eval("grafik" + pos3 + ".src");
            ID3 = setTimeout("galerie3(1)", 1000);
        }
        else                            //  galerie stoppen
        {   clearTimeout(ID3);
            pos3 = 0;                   //  standbild
            document.gal3.src = eval("grafik" + pos3 + ".src");
            anf = true;                 //  start wieder möglich
        }
    }
    else                                //  aufruf durch zeitgeber
    {   if (pos3 < 5)                   //  nächstes bild
            pos3++;
        else
           pos3 = 1;                    //  erstes bild
        document.gal3.src = eval("grafik" + pos3 + ".src");
        ID3 = setTimeout("galerie3(1)", 1000);
    }
}
</script>
```

7.3.4 galerie typ 4 - blättern mit vorschaubildern

beispiel 7c - galerien typ 4

Beim laden der seite werden eine galerie mit kleinen vorschaubildern und die startgrafik der galerie angezeigt. Durch anklicken eines vorschaubildes wird statt der startgrafik die dem vorschaubild entsprechende grafik in originalgröße angezeigt. Zusätzlich kann man in der galerie vorwärts und rückwärts blättern und wenn man eine angezeigte grafik anklickt, wird sie ausgeblendet und die startgrafik wird wieder angezeigt. Dabei sind die grafiken auf der seite nur einmal vorhanden, werden aber permanent in verkleinertem maßstab als vorschaubild und nach auswahl in originalgröße angezeigt. Die galerie enthält grafiken im quer- und im hochformat und zudem in unterschiedlichen größen.



Der einbau der galerie in die seite ist nicht einfach. Zunächst müssen die vorschaubilder eingebaut werden, dabei soll es bilder im hoch- und im querformat geben, außerdem wird jedes img-tag mit einem aufruf der funktion **galerie4** verbunden. Der aufruf enthält als parameter die nummer der anzuzeigenden grafik. Statt der links im a-tag könnte man hier im img-tag auch mit **onclick** arbeiten. Das ganze ist zunächst einfach, aber mit schreibarbeit verbunden und damit fehleranfällig. Deshalb wird für den einbau der vorschau-tabelle zunächst eine einfache lösung gezeigt. Dann eine komplexe variante mit PHP, bei der die vorschau-tabelle nicht fix und fertig in die seite gestellt wird, sondern erst beim aufruf der seite auf dem server mit **PHP** erzeugt wird. Zum verständnis dieser bastelei wird auf die PHP-dokumentation verwiesen.

einbau der vorschaubilder

einfache lösung
Man beachte hier, dass das erste vorschaubild im hochformat und das nächste im querformat angezeigt. wird.

```
<table>
<tr>
<td><a href='Javascript: galerie4("1")'> <img src='im/dom1.jpg'
    style='width: 25px; height: 50px;'> </a></td>
<td><a href='Javascript: galerie4("2")'> <img src='im/dom2.jpg'
    style='width: 50px; height: 25px;'> </a></td>
. . .
</tr>
</table>
```

lösung mit PHP
Der array $dimen legt das format der vorschaubilder fest.

```
<?php
  $dimen = array(" ", "H", "Q", "Q", "Q", "Q", "Q", "Q", "Q", "H");
  echo "<p><b>galerie typ 4</b></p>" . $ende;
  echo "<table>" . $ende;
  echo "<tr>" . $ende;
  for ($x=1; $x<=9; $x++)
  {   $bild = "im/dom" . $x . ".jpg";
      if ($dimen[$x] == "H")                  //  hochformat
          $vbild = "<img src='$bild' style='width: 25px; height: 50px;'> ";
      else                                    //  querformat
          $vbild = "<img src='$bild' style='width: 50px; height: 25px;'> ";
      $td = "<td><a href='Javascript: galerie4(\"" . $x . "\")'> ";
      $td = $td . $vbild . "</a></td>";
      echo $td . $ende;
  }
  echo "</tr>" . $ende;
  echo "</table>" . $ende;
?>
```

Der restliche einbau der galerie erfolgt dann mit einer tabelle, die die beiden button für rückwärts und vorwärts blättern und die startgrafik enthält, jeweils verbunden mit dem aufruf der funktion **galerie4a**. Das img-tag mit der startgrafik hat den namen **gal4**. Bei diesem tag wird hernach bei der anzeige der einzelnen grafiken für die eigenschaft **src** die grafik gewechselt.

```
<table>
   <tr>
      <td><a href="JavaScript: galerie4a(1);">
          <img src="im/n-links.gif" /></a></td>
      <td><a href="JavaScript: galerie4a(0);">
          <img src='im/hexe0.gif' name='gal4' /></a></td>
      <td><a href="JavaScript: galerie4a(2);">
          <img src="im/n-rechts.gif" /></a></td>
   </tr>
</table>
```

funktionen im header

Für die anzuzeigenden grafiken sind instanzen des objekts **Image** notwendig. Die namen der instanzen enthalten auch hier eine fortlaufende nummer. Der funktion **galerie4** wird mit dem parameter **nr** die nummer der anzuzeigenden grafik übergeben. Mit dieser grafik wird im img-tag mit dem namen **gal4** ein einfacher bildwechsel durchgeführt. Außerdem wird die nummer der variablen **pos4** zugewiesen, die damit zeigt, welche grafik gerade angezeigt wird.

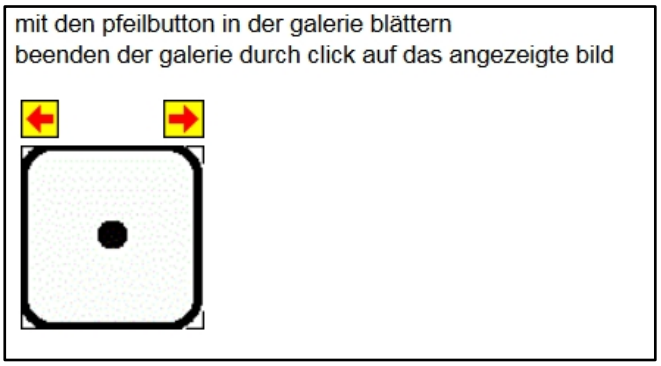
Der funktion **galerie4a** wird der parameter **richt** übergeben, der aussagt, ob rückwärts (1) oder vorwärts (2) geblättert werden soll oder ob die startgrafik (0) angezeigt werden soll. Die variable **pos4** wird entsprechend gesetzt und die grafik mit einem einfachen bildwechsel angezeigt.

```
<script type="text/JavaScript">
var dom0 = new Image();
dom0.src = "im/hexe0.gif";
var dom1 = new Image();
dom1.src = "im/dom1.jpg";
. . .

var pos4 = 0;

//   bildwechsel über vorschaubild
function galerie4(nr)
{  document.gal4.src = eval("dom" + nr + ".src");
   pos4 = nr;
}


//   bildwechsel über pfeiltasten
function galerie4a(richt)
{   if (richt == 0)                    //  galerie schließen
    {   pos4 = 0;                      //  standbild
        document.gal4.src = eval("dom" + pos4 + ".src");
    }
    else if (richt == 1)               //  nach links
    {   if (pos4 > 1)
        {   pos4--;                    //  vorheriges bild
            document.gal4.src = eval("dom" + pos4 + ".src");
        }
    }
    else                               //  nach rechts
    {   if (pos4 < 9)
        {   pos4++;                    //  nächstes bild
            document.gal4.src = eval("dom" + pos4 + ".src");
        }
    }
}
</script>
```

7.3.5 galerie typ 5 - blättern durch einblenden

beispiel 7d - galerien typ 5

Dieser typ unterscheidet sich völlig von den vorhergehenden galerien: die grafiken einschließlich der button zum vorwärts und rückwärts blättern sind jeweils in einen **unsichtbaren** div-container eingebettet, nur die startgrafik ist nach dem laden der seite sichtbar. Beim blättern wird der aktuell angezeigte container ausgeblendet und der nächste wird eingeblendet. Durch click auf eine angezeigte grafik wird wieder die startgrafik gezeigt.



Das aufwendigste ist hier der einbau der container mit den grafiken. Die container müssen namen nach einem einheitlichen muster haben (cont1, cont2 usw.) gleiches gilt für die grafiknamen (dom1.jpg, dom2.jpg). Wenn man sich daran nicht hält, wird das ganze uferlos kompliziert. In diesem beispiel wird übrigens konsequent mit **onclick** gearbeitet. Das einrichten von instanzen für die grafiken entfällt hier. Man muss für jede grafik einen container aufbauen. Diese aufwendige schreibarbeit kann man vermeiden, indem man alle container mit **PHP** aufbaut.

einfach, aber aufwendig

```
<div id="cont1" style="position: absolute; top: 150px; left: 400px;
     visibility: hidden">
<table><tr>
<td><img src="im/n-links.gif" onclick="galerie5(1);"/></td>
<td>   </td>
<td style="text-align: right"><img src="im/n-rechts.gif"
    onclick="galerie5(2);" /></td></tr>
<tr><td colspan='3'><img src="im/dom1.jpg" onclick="galerie5(0);"></td></tr>
</table>
</div>
```

lösung mit PHP

```
<?php
   for ($x=1; $x<=9; $x++)
   {   $idi = "cont" . $x;
       $bild = "im/dom" . $x . ".jpg";
       echo "<div id=\"$idi\" style=\"position: absolute; top: 150px; "
             . "left: 400px; visibility: hidden\">" . $ende;
       echo "<table>" . $ende;
       echo "<tr>" . $ende;
       echo '<td><img src="im/n-links.gif" onclick="galerie5(1);"/></td>' . $ende;
       echo "<td>   </td>" . $ende;
       echo '<td style="text-align: right"><img src="im/n-rechts.gif" '
            . 'onclick="galerie5(2);" /></td>' . $ende;
       echo "</tr>" . $ende;
       echo "<tr>"  . $ende;
       echo "<td colspan='3'>"
            . "<img src=\"$bild\" onclick=\"galerie5(0);\"></td>" . $ende;
       echo "</tr>" . $ende;
       echo "</table>" . $ende;
       echo "</div>" . $ende;
   }
?>
```

Der container mit der startgrafik unterscheidet sich von den übrigen und wird direkt in die seite eingebaut.

```
<div id="cont0">
<table>
  <tr>
     <td><img src="im/n-links.gif" onclick="galerie5(1);"/></td>
     <td>   </td>
     <td style="text-align: right"><img src="im/n-rechts.gif"
         onclick="galerie5(2);" /></td>
  </tr>
  <tr>
     <td colspan="3"><img src="im/hexe0.gif"/ ></td>
  </tr>
</table>
</div>
```

Die funktion **galerie5** wird mit dem parameter **richt** aufgerufen, der anzeigt in welche richtung geblättert werden soll (1 oder 2) oder ob der angezeigte container ausgeblendet werden soll (0). Beim laden der seite wird die variable **pos5** auf 0 gesetzt und zeigt an, welcher container gerade angezeigt wird. Wenn die funktion aufgerufen wird, wird der container, auf den die variable pos5 zeigt, ausgeblendet. Dann wird pos5 je nach blätterrichtung um 1 erhöht oder vermindert oder auf 0 (startgrafik) gesetzt und der entsprechende container wird sichtbar gemacht.

```
<script type="text/JavaScript">
var pos5 = 0;
function galerie5(richt)
{   var nr = pos5;
    divnr = "cont" + pos5;
    tabakt = document.getElementById(divnr).style;
    tabakt.visibility = "hidden";
    if (richt == 0)                        //  ausblenden
        pos5 = 0;
    else                                   //  blättern
    {   if (richt == 1)                    //  rückwärts
        {   if (pos5 > 1)
                pos5--;
        }
        else                               //  vorwärts
        {   if (pos5 < 9)
                pos5++;
        }
    }
    divnr = "cont" + pos5;                 //  einblenden
    tabakt = document.getElementById(divnr).style;
    tabakt.visibility = "visible";
}
</script>
```

7.4 objekt zoomen

Beim zoomen eines objekts (elements) einer seite, beispielsweise einer grafik, ist nicht gemeint per bildwechsel (vgl. 7.2) eine kleine grafik gegen eine größere auszuwechseln, vielmehr wird dabei eine grafik verkleinert angezeigt (vorschaubild) und durch irgend eine aktion größer angezeigt, d.h. es ist nur eine grafik vorhanden, die in unterschiedlicher größe angezeigt wird. Das funktioniert nur, wenn breite und höhe der grafik mit styles definiert sind.

7.4.1 zoome - einfaches zoomen

Für das zoomen wird hier die funktion **zoome** zur verfügung gestellt, mit der beliebige elemente gezoomt werden können. Während eine seite geladen ist, können mehrere elemente gleichzeitig gezoomt werden. Die funktion sammelt für jedes gezoomte element informationen in einem array, dabei wird für ein element im array nur ein eintrag erzeugt, gleichgültig, wie oft es gezoomt wird.

beispiel 7e - zoomen

Das beispiel zeigt das zoomen von grafiken. Die grafik, die gezoomt werden soll, stellt man mit einem img-tag in die seite, das tag muss eine **id** haben, außerdem muss man **breite** und **höhe** für das vorschaubild angeben. Der aufruf der funktion erfolgt dann entweder mit einem **link** oder mit **onclick**. Das beispiel ist so konstruiert, dass die grafik durch anklicken gezoomt wird. Als parameter enthält der funktionsaufruf die **id** des img-tags und breite und höhe, zu der die grafik gezoomt werden soll. Bei dieser konstruktion wird die funktion auch aufgerufen, wenn die gezoomte grafik angeklickt wird, um die grafik wieder zu verkleinern. Wenn man das mit einem eigenen funktionsaufruf erreichen will, muss man ebenfalls die drei parameter angeben.

```
<p><img id="bild1" style="width: 75px; height: 60px"
     src="im/dom5.jpg" onclick="zoome('bild1', 500, 372)"></p>
<p><img id="bild2" style="width: 75px; height: 60px"
         src="im/dom5.jpg" onclick="zoome('bild2', 250, 186)"></td>
<p><a href="Javascript: zoome('bild3', 390, 490)">
    <img id="bild3" style="width: 60px; height: 75px"
    src="im/blumen.jpg"></a></p>
```

funktion zoome

Die funktion prüft zunächst die anzahl der übergebenen parameter, wurden zu wenig übergeben, bricht die funktion ab. Im code der funktion wird leider von objekten statt von elementen geprochen, es wird gebeten diese schludrigkeit zu entschuldigen.

```
<script type="text/JavaScript">
//   im array zometab werden je objekt vier werte gespeichert
//   0 | 1         objekt nicht gezoomt | gezoomt
//   höhe, breite  aus den styles des objekts
//   id            des objekts
var zometab = new Array(new Array(0,0,0," "));


function zoome(par, breit, hoch)
{  var kz, obj, len, x;
   kz = true;
   if (zoome.arguments.length < 3)              //  zu wenig parameter
   {   alert("zoome: parameterfehler");
       kz = false;
   }
```

Dann wird geprüft, ob für das element bereits informationen gespeichert sind. Ist das der fall und und das element ist gezoomt, wird es mit den gespeicherten werten verkleinert, andernfalls mit den parameterwerten gezoomt

```
   if (kz)
   {   obj = document.getElementById(par).style;
       if (zoometab[0][0] == 0)                  //  kein objekt gespeichert
           len = 0;
       else                                      //  objekte gespeichert
       {   len = zoometab[0][0];                 //  anzahl objekte
           for (x=1; x<=len; x++)
           {   if (zoometab[x][3] == par)        //  geklicktes obj. gesp.
               {   if (zoometab[x][0] == 1)      //  objekt gezoomt
                   {   zoometab[x][0] = 0;       //  objekt verkleinern
                       obj.height = zoometab[x][1];
                       obj.width  = zoometab[x][2];
                   }
                   else                          //  objekt zoomen
                   {   zoometab[x][0] = 1;
                       obj.height = hoch  + "px"; ;
                       obj.width  = breit + "px";
                   }
                   kz = false;
                   break;
               }
           }
       }
```

Sind für das element noch keine informationen gespeichert, wird ein neuer eintrag im array angelegt, die informationen werden gespeichert und das element wird mit den parameterwerten gezoomt.

```
       if (kz)                                   //  geklicktes obj. speichern
       {   len++;                                //  und zoomen
           zoometab[0][0] = len;
           zoometab[len]  = new Array(0,0,0," ");
           zoometab[len][0] = 1;
           zoometab[len][1] = obj.height;
           zoometab[len][2] = obj.width;
           zoometab[len][3] = par;
           obj.width  = breit + "px";            //  breite und höhe auf
           obj.height = hoch  + "px";            //  parameterwerte setzen
       }
   }
}
</script>
```

7.4.2 zoom - komplexes zoomen

Es gibt die funktion **zoom**, mit der ebenfalls elemente gezoomt werden können, nun aber in der weise, dass immer nur ein element gezoomt werden kann, d.h. wenn ein element gezoomt ist und ein weiteres soll gezoomt werden, wird das gezoomte element auf die ursprungliche größe verkleinert.

beispiel 7f - zoomen

Das beispiel zeigt das zoomen von grafiken mit der funktion **zoom**. Es stehen die gleichen grafiken zur verfügung, wie beim vorigen beispiel. Auch der aufruf der funktion erfolgt wie bei der funktion **zoome.**

Die funktion **zoom** benötigt einige variable, die wohl selbsterklärend sind. Beim aufruf der funktion wird geprüft, ob genügend parameter übergeben wurden, im fehlerfall bricht die funktion ab, ist bereits ein element gezoomt, wird es zuvor mit den sichergestellten werten verkleinert (funktion **rezoom**). Wurde die funktion durch anklicken des gezoomten elemennts aufgerufen, wird die funktion nach dem verkleinern des elements beendet.

```
 <td><img id="dom1" style="width: 75px; height: 60px"
         src="im/dom5.jpg" onclick="zoom('dom1', 500, 372)"></td>

var zoomobj;                                  //  objekt
var zoomkz  = false;                          //  kennz. kein objekt gezoomt
var objid = " ";                              //  aktuelle id
var objhoch, objbreit;                        //  höhe, breite des objekts
```

funktion zoom

```
function zoom(par, breit, hoch)
{  var kz;

   kz = true;
   if (zoom.arguments.length < 3)             //  zu wenig parameter
   {   alert("zoom: parameterfehler");
       if (zoomkz)                            //  ein objekt ist gezoomt
           rezoom();                          //  container verkleinern
       kz = false;
   }
   if (zoomkz)                                //  ein objekt ist gezoomt
   {   if (objid == par)                      //  gezoomtes objekt angeklickt
       {   rezoom();                          //  objekt verkleinern
           kz = false;
       }
       else                                   //  klick auf anderes objekt
       {   rezoom();                          //  gezoomtes objekt verkleinern
           kz = true;
       }
   }
```

Die id und breite und höhe des angeklickten, noch nicht gezoomten elements werden sichergestellt, dann wird es mit den parameterwerten gezoomt.

```
   if (kz)                                    //  angeklicktes objekt zoomen
   {   zoomkz = true;                         //  kennz. objekt gezoomt
       objid  = par;                          //  id des objekts speichern
                                              //  styles des objekts lesen
       zoomobj  = document.getElementById(par).style;
       objhoch  = zoomobj.height;             //  höhe und breite speichern
       objbreit = zoomobj.width;
       zoomobj.width  = breit + "px";         //  breite und höhe auf übergebene
       zoomobj.height = hoch  + "px";         //  werte setzen
   }
}
```

funktion rezoom

Die funktion **rezoom** wird nur von der funktion **zoom** aufgerufen und verkleinert mit den sichergestellten werten das gezoomte element.

```
function rezoom()
{
   objid = " ";                               //  gespeich. objekt-id löschen
   zoomkz  = false;                           //  kennz. kein objekt gezoomt
   zoomobj.width  = objbreit;                 //  breite und höhe des objekts
   zoomobj.height = objhoch;                  //  auf ursprüngliche werte
}
```

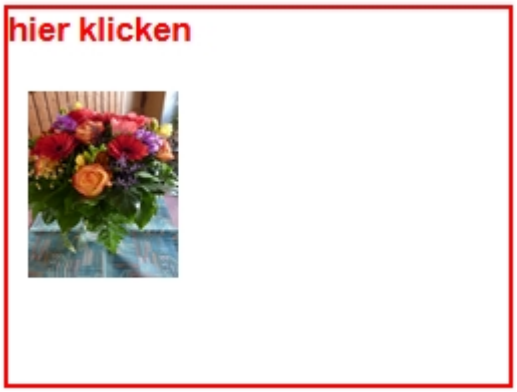
7.4.3 weitere beispiele

beispiel 7g - zoomen

Das beispiel enthält zwei weitere anwendungsfälle für das zoomen von elementen, hier von div-containern. Im ersten fall ist der container so klein dimensioniert, dass sein inhalt, eine grafik, nicht vollständig angezeigt werden kann und auch keine scrollbalken eingefügt werden (vgl CSS-doku 7.3). Die funktion **zoom** ist mit **onclick** im container so positioniert, dass man zum zoomen bzw. verkleinern den container an jeder beliebigen stelle anklicken kann. Gezoomt wird hier der container, der inhalt des containers bleibt unverändert.

```
<div id="zoomdiv" style="width: 200; height: 100; overflow: hidden;
     border-width: 2px; border-style: solid; border-color: red"
     onclick="zoom('zoomdiv', 550, 450)">
<p><img src="im/dom5.jpg"></a></p>
</div>
```

Der zweite fall in diesem beispiel ist kurios: ein zoombarer container enthält eine grafik, die ebenfalls gezoomt werden kann, wobei das zoomen für container und grafik jeweils durch einen aufruf der funktion **zoome** erfolgt. Man kann allerdings beim container den aufruf der funktion nicht mit **onclick** auslösen, weil dann ein click auf die grafik ebenfalls das zoomen des containers bewirkte. Also wurde hier ein link (hier klicken) eingebaut. Das spiel funktioniert auch, wenn man für ein element die funktion **zoom** und für das andere **zoome** verwendet. Für beide kann man **zoom** nicht verwenden, weil dann ein klick auf ein element immer das verkleinern des anderen auslöst.




```
<div id="zoomdiva" style="width: 200px; height: 150px; overflow: hidden;
     border-width: 2px; border-style: solid; border-color: red">
     <a href="Javascript: zoome('zoomdiva', 400, 600)">hier klicken</a>
     <p>  <img id="tebi3" src="im/blumen.jpg"
     style="width: 60px; height: 75px" onclick="zoome('tebi3', 390, 500)"></p>
</div>
```

8. Animation

Für die verschiedenen möglichkeiten animierter darstellungen auf einer seite werden viele bisher beschriebene techniken verwendet.

8.1 laufschrift

Eine laufschrift wird durch ein ereignnis auf der seite ausgelöst, häufig schon beim laden der seite. Oft ist es sinnvoll, eine laufschrift gezielt zum stehen zu bringen. Laufschriften sollte man sparsam verwenden, sie können sonst nerven

beispiel 8a - laufschrift

Das beispiel zeigt vier verschiedene laufschriften:

- Eine unbenannte schrift läuft in einem textfeld des formulars **fmt** von rechts nach links, wenn die schrift links aus dem feld läuft, kommt sie von rechts wieder herein.
- Die **laufschrift 1** steht in dem roten container **lauf1**, der sich auf einem roten balken von rechts nach links bewegt. Wenn er links aus dem bild gelaufen ist, kommt er von rechts wieder herein, d.h.die schrift bewegt sich nicht, sondern nur der container.
- Die **laufschrift 2** läuft in dem blauen container **lauf2** von rechts nach links; sie läuft links aus dem container und kommt von rechts wieder herein.
- Die **laufschrift 3** steht in dem roten container **lauf3**, der sich von rechts nach links und von links nach rechts bewegt.

Die bewegung aller schriften wird mit **start** (aufruf der funktion **init**) gestartet und mit **stopp** (aufruf der funktion **STOPPEN**) angehalten.

```
<body>
<form name="fmt">
  <p><input type="text" name="T1" size="40" onFocus="this.blur();"
     readonly="readonly"></p>
</form>
<div id="hilf" style="position: absolute; color: yellow;
   background-color: red; top: 200px; height: 20px"> 
</div>
<div id="lauf1" style="position: absolute; color: yellow;
   background-color: red; top: 200px"><b>das ist die laufschrift 1</b>
</div>
<div id="lauf2" style="position: absolute; color: yellow;
   background-color: blue; text-align: center; top: 150px"><b>lauf2</b>
</div>
<div id="lauf3" style="position: absolute; color: yellow;
   background-color: red; text-align: center; top: 250px">
   <b>das ist die laufschrift 3</b>
</div>
<p><a href="JavaScript: init()">start</a>
   <b>laufschriften starten</b></p>
<p class="font10b"><a href="JavaScript: STOPPEN()">stopp</a>
   <b>laufschriften stoppen</b></p>
</body>
```

funktion init

Die funktion **init** bereitet die laufschriften vor und startet sie. Mit der funktion **ANFA** werden die laufschriften 1 und 3, mit der funktion **laufen** die schriften 2 und im textfeld gesteuert. Beide funktionen werden per zeitgeber aufgerufen. Alle funktionen stehen im header der seite, vor den funktionen sind einige globale variable definiert.

```
<script type="text/JavaScript">
var breite, zeit, links3
var lang1, lang2, lang3
var lpos1, lpos2, lpos3, tpos1, tpos2, tpos3;
var urtext = "  +++   das ist eine laufschrift   +++   ";
var ltext, ltext2;
var kz = false;

function init()
{
    if (kz)                        //  laufschriften schon eingeschaltet
        return;
    kz     = true;                  //  laufschriften einschalten
    breite = screen.width;
    links3 = true;                 //  laufschrift 3 nach links
    lang1  = 150;                  //  länge der schriften
    lang2  = 300;
    lang3  = 200;
    lpos1  = breite + 10;           //  position der schriften
    lpos2  = breite - 400;
    lpos3  = breite - lang3 - 40;
    zeit   = 10;
    ltext2 = "  +++ das ist die laufschrift 2 +++  ";

    var hilf = document.getElementById("hilf").style; //  roter balken für lauf1
    hilf.width  = screen.width + "px";
    hilf.left   = "1px";

    hilf = document.getElementById("lauf2").style;
    hilf.width = lang2 + "px";
    hilf.left  = lpos2 + "px";
    hilf.innerHTML  = '<font face="Arial" size="2"><b>' + ltext2 + '</b></font>';

    setTimeout ('ANFA()', zeit);          //  laufschrift 1 und 3 starten

    ltext = urtext;                       //  laufschrift in textfeld eintr.
    document.forms["fmt"].T1.value = ltext;   //  und zusammen mit
    setTimeout("laufen()", 100);              //  laufschrift 2 starten
}
```

funktion ANFA

Die funktion **ANFA** steuert die laufschriften 1 und 2, die fest in einem container stehen indem sie die container zunächst schrittweise mit 10 pixel von rechts nach links bewegt. Wenn der container **lauf1** (laufschrift 1) über den linken bildschirmrand hinausgelaufen ist, wird er von rechts wieder ins bild gezogen. Der container **lauf3** (laufschrift 3) ändert am linken, bzw. rechten rand die laufrichtung. Nach jedem schritt werden die funktionen erneut per zeitgeber aufgerufen (zeittakt 10 msec).

```
function ANFA()
{   var hilf;

    hilf = document.getElementById("lauf1").style;
    hilf.width = lang1 + "px";
    hilf.left  = lpos1 + "px";
    hilf = document.getElementById("lauf3").style;
    hilf.width = lang3 + "px";
    hilf.left  = lpos3 + "px";
    if (lpos1 > -lang1)                     //  schrift 1 weiter nach links
        lpos1 -= 1;
    else                                    //  schrift 1 wieder von rechts
        lpos1 = breite + 10;
    if (links3)                             //  schrift 3 läuft nach links
    {   if (lpos3 >= 10)                    //  weiter nach links
            lpos3 -= 1;
        else                                //  richtungswechsel
        {   lpos3 += 1;                     //  nach rechts
            links3 = false;
        }
    }
    else                                    //  schrift 3 läuft nach rechts
    {   if (lpos3 < breite - lang3 - 30)    //  weiter nach rechts
            lpos3 += 1;
        else                                //  richtungswechsel
        {   lpos3 -= 1;
            links3 = true;                  //  nach links
        }
    }
    if (kz)                                 //  weiterlaufen
        setTimeout ('ANFA()', zeit);
}
```

funktion laufen

Die funktion **laufen** steuert die laufschrift 2 im container **lauf2** und die schrift im textfeld. Bei jedem aufruf wird das erste zeichen der schrift abgeschnitten, die schrift um ein zeichen nach links gezogen und das abgeschnittene zeichen am ende angefügt. Dann wird die funktion erneut per zeitgeber aufgerufen (zeittakt 100 msec)

```
function laufen()                           //  text um ein zeichen nach links
{   ltext  = ltext.substring(1, ltext.length) + ltext.charAt(0);
    ltext2 = ltext2.substring(1, ltext2.length) + ltext2.charAt(0);
    document.fmt.T1.value = ltext;          //  text in textfeld
                                            //  text in container
    document.getElementById("lauf2").innerHTML = ltext2;

    if (kz)                                 //  weiterlaufen
        setTimeout("laufen()", 100);
}
```

funktion STOPPEN

Die funktion **STOPPEN** setzt die variable **kz** auf false und stoppt dadurch alle laufschriften.

```
function STOPPEN()
{
    kz = false;
}
```

8.2 objekt in bewegung

Darunter wird verstanden, dass ein auf einer seite angezeigtes element durch ein ereignis in bewegung gesetzt wird und diese bewegung ohne weitere eingriffe beibehält, bis durch ein weiteres ereignis die bewegung angehalten wird.

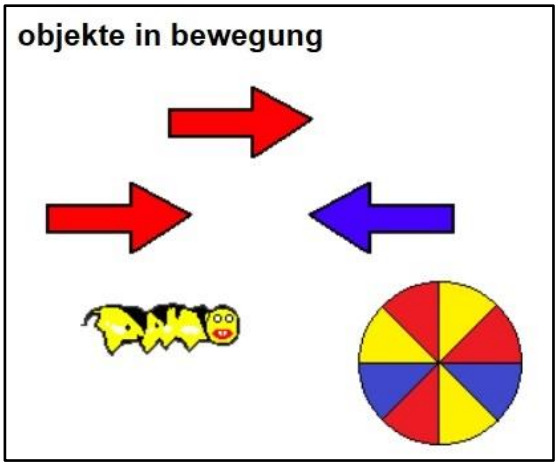
beispiel 8b - objekte in bewegung

Das beispiel zeigt sechs elemente, die in bewegung gesetzt werden; bei den elementen handelt es sich jeweils um div-container. Das muss nicht so sein, ist aber am einfachsten zu handhaben.

- Der container **ueber** enthält einen text und wandert vom unteren bildschirmrand nach oben und bleibt dort stehen.
- Der container **dreher** enthält die grafik **ball**, die mit zunehmender geschwindigkeit rotiert, bis sie eine bestimmte geschwindigkeit erreicht hat. Dann beginnt die rotation mit der anfangsgeschwindigkeit erneut.
- Der container **phinher** enthält den roten pfeil **hinher** und wandert von links nach rechts über den bildschirm. Wenn der rechte rand überschritten ist, wird die grafik gewechsel und der container wandert wieder zurück und setzt das endlos fort. Der start- und wendepunkt liegt außerhalb des bildschirms.
- Die container **rot** und **blau** enthalten jeweils einen pfeil. Sie bewegen sich gegeneinander, stoßen zusammen und wandern zurück und wiederholen das endlos. Die grafiken sind unbenannt.
- Der container **kriech** enthält die grafik **wurm**; der wurm kriecht von links nach rechts, schlägt einen purzelbaum und kriecht wieder zurück und setzt das endlos fort.

Mit **start** (aufruf der funktion **ANFANG)** werden die bewegungen gestartet, mit **stopp** (aufruf der funktion **BEWSTOP)** werden sie gestopt,

```
<div id="ueber" style="position: absolute; top: 600px">     //  text
   <p class="font14b">objekte in bewegung</p>
</div>                                                      //  ball
<div id="dreher" style="position: absolute; top: 180px; left: 750px">
   <img src="im/ball1.png" style="width: 100px; height: 100px;" name="ball">
</div>                                                      //  pfeil
<div id="phinher" style="position: absolute; top: 100px; left: -150px">
   <img src="im/p-links.gif" style="width; 94px; height: 51px;" name="hinher">
</div>                                                      //  zwei pfeile
<div id="rot" style="position: absolute; left: 10px; top: 200px">
   <img src="im/pfeilr.gif" style="width: 94px; height: 51px;">
</div>
<div id="blau" style="position: absolute; left: 600px; top: 200px">
   <img src="im/pfeill.gif" width="94" height="51">
</div>                                                      //  wurm
<div id="kriech" style="position: absolute; left: 10px; top: 300px">
   <img src="im/wurm1.gif" name="wurm">
</div>
<p><a href="JavaScript: ANFANG()">start</a>
   <b>bewegung starten</b></p>
<p class="font10b"><a href="JavaScript: BEWSTOP()">stopp</a>
   <b>bewegung stoppen</b></p>
```



funktion ANFANG
Die funktion steuert die bewegungen, dabei wird der erste teil der funktion nur beim ersten aufruf ausgeführt. Im ersten teil werden startwerte für die verschiedenen bewegungen gesetzt bzw. ermittelt und die bewegungen werden gestartet. Der zweite teil der funktion wird nur ausgeführt, wenn vor dem aufruf der funktion die bewegungen angehalten wurden; sie werden dann fortgesetzt. Vor den funktionen stehen einige globale variable.

```
<script type="text/JavaScript">
var a, l, r, atop, breit, zeit, dreh, drehpos, drehzahl;
var wurmnr, wurmpos, wricht, wnr, wz;
var anf   = true;
var kzbew = false;
var wm1 = new Image();      //  ebenso für wm2, wm3, wm4
wm1.src = "im/wm1.gif";
var ball1 = new Image();    //  ebenso für ball2 – ball8
ball1.src = "im/ball1.png";

function ANFANG()
{   var hilf;
    if (anf)                                //  erster aufruf
    {   anf = false;
        kzbew = true;                       //  alles in bewegung
        breit = screen.width;               //  breite des bildschirms
        zeit  = 30;                         //  zeit-takt
        atop  = screen.height + 100;        //  startpos. titelzeile
        setTimeout('TITO()', 100);          //  titelzeile hochziehen
        hilf = document.getElementById("phinher").style.left;
        a = parseInt(hilf);                 //  start-pos pfeil
        BEWEGRE();                          //  pfeil startet
        hilf = document.getElementById("rot").style.left;
        l = parseInt(hilf);                 //  start-pos. roter pfeil
        hilf = document.getElementById("blau").style.left;
        r = parseInt(hilf);                 //  start-pos blauer pfeil
        GEGEN();                            //  pfeile starten
        dreh  = 200;                        //  anfangstempo für ball
        drehpos  = 1;
        drehzahl = 0;
        DREHEN();                           //  ball startet
        hilf = document.getElementById("kriech").style.left;
        wurmpos = parseInt(hilf);           //  start-pos. wurm
        wurmnr  = 1;                        //  wurmbild 1
        WURMRE();                           //  wurm startet
    }
    else if (!kzbew)                        //  bewegung fortsetzen
    {   kzbew = true;
        atop = screen.height + 100;         //  titelzeile an startpos.
        setTimeout('TITO()', 100);          //  titelzeile hochziehen
        if (richt)                          //  pfeil nach rechts
            window.setTimeout ('BEWEGRE()', zeit);
        else                                //  oder nach links
            window.setTimeout ('BEWEGLI()', zeit);
        if (l+70 < r)                       //  pfeile gegeneinander
            setTimeout ('GEGEN()', zeit);
        else                                //  oder auseinander
            setTimeout ('DAVON()', zeit);
        DREHEN();                           //  ball dreht weiter
        if (wricht)                         //  wurm nach rechts
            window.setTimeout ('WURMRE()', zeit);
        else                                //  oder links
            window.setTimeout ('WURMLI()', zeit);
    }
}
```

funktion TITO
Die funktion zieht den container **ueber** mit dem text nach oben; beim aufruf der funktion wird der container auf die aktuelle position in **atop** gesetzt, dann wird atop um 10 verringert. Solange der wert in atop größer ist als 10, wird die funktion per zeitgeber erneut aufgerufen (zeittakt 100 msec).

```
function TITO()
{
    document.getElementById("ueber").style.left = "50px";
    document.getElementById("ueber").style.top  = atop + "px";
    atop -= 10;
    if (atop >= 10)
    {
        setTimeout("TITO()", 100);
    }
}
```

funktion DREHEN
Die funktion läßt den ball im container **dreher** rotieren; für den ball gibt es acht grafiken, die durch den aufruf der funktion nacheinander angezeigt werden, bei jeder anzeige wird der ball um den achten teil eines kreises gedreht. Dann wird per zeitgeber die funktion erneut aufgerufen. Nach 16 bewegungen, d.h. zwei umdrehungen wird der zeittakt um 10 msec verringert. Wenn eine taktzeit von 10 msec unterschritten wird, wird der takt wieder auf 200 msec gesetzt.

```
function DREHEN()
{
    if (kzbew)
    {   if (drehpos < 8)
            drehpos++;
        else
            drehpos = 1;
        drehzahl++;
        if (drehzahl >= 16)                 //  nach zwei umdrehungen
        {   drehzahl = 0;
            dreh-=10;                       //  tempo erhöhen
            if (dreh < 10)                  //  bei höchstgeschwindigkeit
                dreh = 200;                 //  auf anfangstempo
        }
        document.ball.src = eval("ball" + drehpos + ".src");
        window.setTimeout('DREHEN()', dreh);
    }
}
```

funktionen GEGEN und DAVON
Die funktion **GEGEN** bewegt die container **rot** und **blau** mit den pfeilen gegeneinander; bei jedem aufruf rückt der container **rot** um 2 pixel nach rechts und der container **blau** um 3 pixel nach links. Abhängig von der errreichten position wird per zeitgeber entweder die funktion **GEGEN** oder die funktion **DAVON** aufgerufen. Die funktion **DAVON** bewegt die container auseinander, sie ist das spiegelbild der funktion **GEGEN.** Von der funktion DAVON werden nur die anweisungen gezeigt, die anders als bei der funktion GEGEN sind.

```
function GEGEN()                                        DAVON
{   if (kzbew)
    {   document.getElementById("rot").style.left  = l + "px";
        document.getElementById("blau").style.left = r + "px";
        l+=3;                                           l-=6;
        r-=2;                                           r+=4;
        if (l+70 < r)                                   if (l < 10 || r > 700)
            setTimeout ('GEGEN()', zeit);
        else
            setTimeout ('DAVON()', zeit);
    }
}
```

funktionen BEWEGRE und BEWEGLI
Die funktion **BEWEGRE** bewegt den container **phinher** mit dem roten pfeil um 2 pixel nach rechts, solange der container nicht den bildschirm verlassen hat und ruft dann die funktion erneut per zeitgeber auf. Wenn der pfeil vom bildschirm verschwunden ist, wird die grafik **hinher** gewechselt und die funktion **BEWEGLI** aufgerufen. Die variable **richt** zeigt die richtung an, in der die bewegung erfolgt. Die funktion **BEWEGLI** ist das spiegelbild von **BEWEGRE** und steuert die bewegung nach links.

```
function BEWEGRE()                                      BEWEGLI
{   if (kzbew)
    {   if (a <= breit + 50)                            if (a >= -150)
        {   richt = true;                               richt nicht ändern
            a+=2;                                       a-=2;
            document.getElementById("phinher").style.left = a + "px";
            window.setTimeout('BEWEGRE()', zeit);       BEWEGLI  aufrufen
        }
        else                        //  richtung wechseln
        {   richt = false;                              richt = true;
            document.hinher.src="im/p-rechts.gif";
            window.setTimeout('BEWEGLI()', zeit);       BEWEGRE aufrufen
        }
    }
}
```

funktionen WURMRE und WURMLI
Die funktion **WURMRE** bewegt den container **kriech** mit dem wurm schrittweise (10 pixel) nach rechts, wechselt bei jedem schritt die grafik und ruft erneut die funktion WURMRE auf. Hat der container den rechten bildschirmrand erreicht, wird die funktion **PURZEL** aufgerufen. Die variable **wricht** sagt aus, in welche richtung die bewegung geht. Die funktion **WURMLI** ist das spiegelbild zur funktion **WURMRE** und steuert die bewegung nach links.

```
function WURMRE()                                       WURMLI
{   if (kzbew)
    {   wurmnr++;
        wurmpos += 10;                                  wurmpos -= 10;
        if (wurmpos > breit - 150)                      if (wurmpos < 10)
        {   wnr = 1;         //  purzelbaum
            wz  = 1;
            window.setTimeout('PURZEL()', 30);          PURZEL();  ohne Timeout
        }
        else                                    //  andernfalls einen schritt
        {   if (wurmnr > 2)                     //  weiter
                wurmnr = 1;
            wricht = true;                              wricht = false;
            document.getElementById("kriech").style.left = wurmpos + "px";
            eval ("document.wurm.src='im/wurm" + wurmnr + ".gif'");
            window.setTimeout('WURMRE()', 200);         WURMLI
        }
    }
}
```

funktion BEWSTOP
Die funktion setzt die variable **kzbew** auf false und stoppt dadurch die bewegungen.

```
function BEWSTOP()
{
    kzbew = false;
}
```

funktion PURZEL
Die funktion **PURZEL** steuert den purzelbaum des wurms; dazu werden vier grafiken benötigt, die nacheinander angezeigt werden, d.h. nach einer anzeige wird die funktion per zeitgeber erneut aufgerufen. Nach 10 purzelbäumen wird abhängig von **wricht** der wurm nach rechts oder nach links gerichtet angezeigt und die funktion **WURMRE** oder **WURMLI** aufgerufen.

```
function PURZEL()
{
    document.wurm.src = eval("wm" + wnr + ".src");
    wnr++;
    if (wnr > 4)                            //  ein purzelbaum fertig
    {   wnr = 1;
        wz++;                               //  bei weniger 10
        if (wz <= 10)                       //  nächster purzelbaum
            window.setTimeout('PURZEL()', 30)
        else
        {   if (wricht)                     //  wurm kriecht nach links
            {   wurmpos = breit - 100;
                document.wurm.src="im/wurm1a.gif";
                window.setTimeout('WURMLI()', 300);
            }
            else                            //  wurm lriecht nach rechts
            {   wurmpos = 10;
                document.wurm.src="im/wurm1.gif";
                window.setTimeout('WURMRE()', 300);
            }
        }
    }
    else                                    //  purzelbaum nicht fertig
        window.setTimeout('PURZEL()', 30);
}
</script>
```

8.3 objekt bewegen

Darunter wird verstanden, ein in einer seite angezeigtes element (hier auch als objekt bezeichnet) durch eine aktion (beispielsweise mausklick) auf dem bildschirm zu bewegen oder mit dem mauszeiger zu ziehen.

beispiel 8c - objekt bewegen

Das beispiel zeigt eine kleine grafik, die man entweder per mausklick bewegen oder mit dem mauszeiger ziehen kann. Die seite zeigt den div-container **katze** mit der grafik **imgkatze** und vier links. Mit den links kann man den container schrittweise (10 pixel) in jede beliebige richtung bewegen. Man kann aber auch die katze anklicken. Dann ändert sich das aussehen des mauszeigers und er wird mit der katze "verbunden", d.h. wenn man nun den zeiger bewegt, "zieht" er die katze mit. Ein erneuter klick löst die verbindung.


```
<body onload="init()">
<p><a href="Javascript: links()">nach links</a><br />
   <a href="Javascript: rechts()">nach rechts</a><br />
   <a href="Javascript: oben()">nach oben</a><br />
   <a href="Javascript: unten()">nach unten</a></p>
<div id="katze" style="position:absolute; width: 90; height: 35; top: 20px;
     left: 200px; cursor: crosshair;"><img src="im/katz.gif" id="imgkatze">
</div>
```

funktion init

Die funktion wird beim laden der seite aufgerufen und ermittel die position und größe des containers mit der katze. Außerdem werden für die ereignisse **mousedown** und **mousemove** funktionen vereinbart. Vor den funktionen stehen einige globale variable.

```
<script language="JavaScript">
var katzex, katzey,dx,dy;              //  koordinaten katze
var katzeh, katzew;                    //  höhe, breite katze
var obj;                               //  objekt-adresse katze
var bereit = false;
var breite = screen.width;
var hoehe  = screen.availHeight - 90;

function init()
{
    obj = document.getElementById("katze").style;     //  styles katze lesen
    katzex = parseInt(obj.left);                      //  linke obere ecke katze
    katzey = parseInt(obj.top);
    katzeh = parseInt(obj.height);                    //  höhe, breite katze
    katzew = parseInt(obj.width);
    document.onmousedown = mousestat;                 //  mouse-taste gedrückt
    document.onmousemove = mousebew;                  //  mouse-zeiger bewegt
}
```

bewegung per link
Mit den folgenden funktionen wird die katze schrittweise (10 pixel) bewegt. Es erfolgt keine bewegung, wenn die katze dabei den bildschirmrand überschreiten würde.

```
function links()
{   if (katzex - 10 > 1)
    {   katzex -= 10;
        obj.left = katzex + "px";
    }
}

function rechts()
{   if (katzex + 10 + katzew < breite)
    {   katzex += 10;
        obj.left = katzex + "px";
    }
}
```

```
function oben()
{   if (katzey - 10 > 1)
    {   katzey -= 10;
        obj.top = katzey + "px";
    }
}

function unten()
{   if (katzey + katzeh + 10 < hoehe)
    {   katzey += 10;
        obj.top = katzey + "px";
    }
}
```

ziehen mit dem mauszeiger
Beim anklicken der katze ruft das ereignis **onmousedown** die funktion **mousestat** auf. Wenn man dann den zeiger bewegt, ruft das ereignis **onmousemove** die funktion **mousebew** auf. Die funktion **mousestat** reagiert nur, wenn sich beim klicken der mauszeiger über der katze (grafik **imgkatze**) befindet. Wenn die variable **bereit** den inhalt true hat, wird die katze schon festgehalten, die variable wird auf false gesetzt und die katze wird losgelassen. Andernfalls wird die variable auf true gesetzt und die distanz zwischen der maus-position und der linken, oberen ecke des containers wird errechnet. Von jetzt an folgt die katze dem mauszeiger.

```
function mousestat(e)
{   var aktx, akty, ev, ziel;

    ev = e;
    ziel = ev.target;
    if (ziel.id == "imgkatze")              //  katze wurde angeklickt
    {   if (bereit)                         //  wenn die katze festgehalten wird,
            bereit = false;                 //  wird sie nun freigegeben
        else
        {   bereit = true;                  //  katze wird festgehalten
            aktx = ev.clientX;              //  position mousezeiger
            akty = ev.clientY;
            dx  = aktx - logox;             //  distanz mousezeiger - katze
            dy  = akty - logoy;
        }
    }
}
```

Die funktion **mousebew** wird bei jeder bewegung des mauszeigers aufgerufen. Aus der aktuellen position des zeigers und den in **mousestat** errechneten distanzen wird die neue position für den container **katze** errechnet und der container wird neu positioniert.

```
function mousebew(e)
{   var aktx, akty, ev;

    ev = e;
    if (bereit)                             //  katze hängt am mouse-zeiger
    {   aktx = ev.clientX;
        akty = ev.clientY;
        logox = aktx - dx;                  //  neue position katze
        logoy = akty - dy;
        obj.left = logox + "px";
        obj.top  = logoy + "px";
    }
}
```

8.4 katz und maus

beispiel 8d

Das beispiel ist eine umfangreiche erweiterung des vorhergehenden; es ist wieder die katze im spiel, aber jetzt gibt es noch eine maus, die von der katze gejagt wird. Das beispiel wird hier gezeigt, aber nur notdürftig erklärt, den rest muss man sich erarbeiten.

Der container **katze** mit der katze hat eine **id** und einen **namen** mit der gleichen bezeichnung. Der container **maus** mit der maus hat nur eine **id**. Es sind noch vier container (tx1 – tx4) mit hinweisen vorhanden, die hier nicht gezeigt werden.

```
<div id="katze" style="position:absolute; width: 90px; height: 35px; top: 250px;
   left: 10px; visibility: visible; cursor: crosshair;">
   <img src="im/katz.gif" id="katzegfx" name="katzegfx">
</div>

<div id="maus" style="position:absolute; width: 40px; height: 20px; top: 250px;
  left: 250px; visibility: visible"><img src="im/maus.gif" id="mausgfx">
</div>
```

Vor den funktionen im header stehen stehen auch hier einige globale variable.

```
//   globale variable
var katzex, katzey, katzer, katzeu;     //  koordinaten katze
var mausx, mausy, mausr, mausu;         //  koordinaten maus
var katzeh, katzew;                     //  höhe, breite katze
var maush, mausw;                       //  höhe, breite maus
var dx, dy, mxa, mya;                   //  koordinaten mouse-zeiger
var obj, obja;                          //  objekt-adressen katze, maus
var bereit = false;
var text1, text2, text3, text4;
.
```

funktion init
Die funktion wird beim laden der seite aufgerufen und stellt styles der container sicher. Außerdem werden für die ereignisse **onmousedown** und **onmousemove** funktionen vereinbart.

```
function init()
{
    obj = document.getElementById("katze").style;  //  styles katze lesen
    katzeh = parseInt(obj.height);                 //  höhe, breite katze
    katzew = parseInt(obj.width);
    katzex = parseInt(obj.left);            //  katze ecke links, oben
    katzey = parseInt(obj.top);
    katzer = katzex + katzew;               //  katze ecke rechts, unten
    katzeu = katzey + katzeh;
    objа = document.getElementById("maus").style;  //  styles maus lesen
    maush = parseInt(obja.height);                 //  höhe, breite maus
    mausw = parseInt(obja.width);
    mausx = parseInt(obja.left);            //  maus ecke links, oben
    mausy = parseInt(obja.top);
    mausr = mausx + katzew;                 //  maus ecke rechts unten
    mausu = mausy + katzeh;
    text1 = document.getElementById("tx1").style;  //  styles texte lesen
    text2 = document.getElementById("tx2").style;
    text3 = document.getElementById("tx3").style;
    text4 = document.getElementById("tx4").style;
    document.onmousedown = setzestatus;          //  mouse-taste gedrückt
    document.onmousemove = dragdrop;             //  mouse-zeiger bewegt
}
```

funktion setzestatus
Die funktion wird aufgerufen, wenn eine maustaste gedrückt wird. Sie ermittel die position des mauszeigers u.ä., d.h. sie hält die katze fest. Wird die katze schon festgehalten, wird sie losgelassen.

```
function setzestatus(ev)
{   var aktx, akty, ziel;

    text1.visibility = "hidden";            //  alle texte ausblenden
    text2.visibility = "hidden";
    text3.visibility = "hidden";
    text4.visibility = "hidden";
    ziel = ev.target;
    if (bereit)                             //  wenn katze festgehalten,
    {   bereit = false;                     //  nun frei geben
        text1.visibility = "visible";
    }
    else                                    //  katze nicht festgeh.
    {   if (ziel.id == "katzegfx")          //  katze wurde angeklickt
        {   aktx = ev.clientX;              //  position mouse-zeiger
            akty = ev.clientY;
            dx  = aktx - katzex;            //  dist. mouse-zeiger - katze
            dy  = akty - katzey;
            mxa = aktx;                     //  posit. mouse-zeiger sichern
            mya = akty;
            bereit = true;                  //  katze hängt am mouse-zeiger
            text2.visibility = "visible";
        }
        else if (ziel.id == "mausgfx")      //  maus wurde angeklickt
            text3.visibility = "visible";
        else                                //  es wurde irgendwo geklickt
            text4.visibility = "visible";
    }
}
```

funktion dragdrop
Die funktion wird aufgerufen, wenn der mausezeiger bewegt wird. Es wird zunächst die bewegungsrichtung ermittelt und ggf. die grafik für die katze gewechselt. Dann wird die neue position des containers **katze** mit der katze berechnet, aber noch nicht positioniert. Mit der funktion **posmaus** wird die maus neu positioniert. Erst dann wird die katze neu positioniert.

```
function dragdrop(ev)
{   var aktx, akty;
    var rix = 0;                            //  bewegungsrichtung
    var riy = 0;

    if (bereit)                             //  katze hängt am mouse-zeiger
    {   aktx = ev.clientX;                  //  position mouse-zeiger
        akty = ev.clientY;
        if (aktx > mxa)                     //  bewegungsrichtung:
        {   rix = 1;                        //  nach links
            document.katzegfx.src="im/katz.gif";
        }
        else if (aktx < mxa)                //  nach rechts
        {   rix = -1;
            document.katzegfx.src="im/katza.gif";
        }
        if (akty > mya)                     //  nach unten
            riy = 1;
        else if (akty < mya)                //  nach oben
           riy = -1;
        mxa = aktx;                         //  pos. mouse-zeiger sich.
        mya = akty;
        aktx = aktx - dx;                   //  neue position katze
        akty = akty - dy;
        posmaus(aktx, akty, rix, riy);      //  maus neu setzen
        obj.left = aktx + "px";             //  katze neu setzen
        obj.top  = akty + "px";
        katzex = aktx;                      //  koordinaten katze
        katzey = akty;                      //  berechnen
        katzer = katzex + katzew;
        katzeu = katzey + katzeh;
    }
}
```

funktion posmaus
Die funktion errechnet die neue position für den container **maus** mit der maus und positioniert ihn neu. Übergeben werden mit x und y die koordinaten der linken oberen ecke des containers **katze** und mit rx und ry informationen, in welche richtung die katze sich bewegt. Die funktion ist so konstruiert, dass die maus der katze immer entkommt.

```
function posmaus(x, y, rx, ry)
{  var dax = 0;
   var day = 0;

   //  wenn sich die koordinaten von katze und maus berühren
   //  wird die maus in die bewegungsrichtung der katze
   //  um 20 pixel bewegt
   //  zum besseren verständnis:
   //  katzex, katzey   -   linke obere ecke container katze
   //  katzer, katzeu   -   rechte untere ecke
   //  mausx, mausy     -   linke obere ecke container maus
   //  mausr, mausu     -   rechet untere ecke
   //
```

```
   if (rx > 0)                              //  nach rechts schieben
   {   if (katzer > mausx && katzer < mausr)
       {   if ((katzey >= mausy && katzey <= mausu)
                || (katzeu >= mausy && katzeu <= mausu))
               dax = 20;
       }
   }
   if (rx < 0)                              //  nach links schieben
   {   if (katzex < mausr && katzex > mausx)
       {   if ((katzey >= mausy && katzey <= mausu)
                || (katzeu >= mausy && katzeu <= mausu))
               dax = -20;
       }
   }
   if (ry > 0)                              //  nach unten schieben
   {   if (katzeu > mausy && katzeu < mausu)
       {   if ((katzex >= mausx && katzex <= mausr)
                || (katzer >= mausx && katzer <= mausr))
               day = 20;
       }
   }
   if (ry < 0)                              //  nach oben schieben
   {   if (katzey < mausu && katzey > mausy)
       {   if ((katzex >= mausx && katzex <= mausr)
                || (katzer >= mausx && katzer <= mausr))
               day = -20;
       }
   }

   if (dax != 0 || day != 0)           //  maus muss sich bewegen
   {   mausx += dax;                   //  neue position der maus
       mausy += day;
       if (mausx < 0)                  //  maus springt über die katze, wenn
           mausx = katzer + 20;        //  sie über den rand geschoben würde
       else if (mausx > screen.width - mausw - 20)
           mausx = katzex - 20;
       if (mausy < 0)
           mausy = katzeu + 20;
       else if (mausy > screen.height - maush - 220)
           mausy = katzey - maush - 20;
       mausr = mausx + katzew;
       mausu = mausy + katzeh;

       obja.left = mausx + "px";       //  neue position maus
       obja.top  = mausy + "px";
   }
}
```

9. objektorientierte programmierung

9.1 objekt vereinbaren

Obwohl in Javascript durchgehend mit objekten gearbeitet wird, ist im vergleich mit PHP oder modernen programmiersprachen die objektorientierte programmierung recht schlicht realisiert. Wie schon unter nr. 1.2 ausgeführt, geht es bei den begriffen etwas durcheinander: es gibt **keine klasse,** mit der üblicherweise die **eigenschaften** und **methoden** eines objekts definiert oder beschrieben werden, es gibt nur eine funktion, mit der eigenschaften und ggf. methoden vereinbart werden und die auch dazu dient, objekte, also instanzen mit dem von der funktion definierten typ zu erzeugen. Oft wird die funktion auch als **konstruktor-funktion** bezeichnet.

9.1.1 konstruktor-funktion

function *klasse* ([*param* [= *default*] , . . .])
{
    this.*eigenschaft* = *param* | *wert*; einer eigenschaft einen wert zuweisen
    this.*methode* = *methfunk*; eine methode vereinbaren
    [return *erg*;]
}

klasse	name der funktion; es gibt bei Javascript keine klasse, aber man tut sich leichter, wenn man so tut als ob und den funktionsnamen wie eine klassebezeichnung betrachtet.
param	vereinbarung eines parameters, für den beim aufruf ein argument übergeben wird
default	wert, wenn für den parameter nichts übergeben wird
eigenschaft	eigenschaft des objekts; man kann einen parameter, eine in der funktion vereinbarte variable oder direkt einen wert zuweisen. Oft wird einfach 0 zugewiesen. Man kann beliebig viele eigenschaften vereinbaren. Eigenschaften entsprechen dem, was man sonst als variable bezeichnet.
methode	name einer methode; man kann keine oder beliebig viele methoden vereinbaren
methfunk	name einer funktion (methoden-funktion), die der methode zugewiesen wird; die funktion wird ohne ( ) angegeben. Man erleichtert sich den umgang mit den methoden des objekts, wenn man für die methode und die methoden-funktion die gleiche bezeichnung wählt.
erg	rückgabewert der funktion; in einer konstruktor-funktion eher unüblich.

9.1.2 methode

Für eine methode ist eine funktion notwendig; die funktion kann beliebige anweisungen enthalten, wichtig sind anweisungen, mit denen eigenschaften des objekts gelesen oder geschrieben werden.

function *methfunk* ([*param* [= *default*] , . . .])
{
    var *wert* = **this.***eigenschaft*;
    this.*eigenschaft* = *irgendetwas*;
    var ferg = **this**.mname([wert, …]);
}

methfunk	name der funktion, der in der konstruktor-funktion vereinbart wurde, also nicht der vereinbarte name der methode. Man erleichtert sich das leben, wenn man für beides den gleichen namen verwendet.
eigenschaft	eigenschaft (variable), die in der konstruktor-funktion vereinbart wurde
mname	aufruf einer methode, die in der konstruktor-funktion vereinbart wurde; es ist der name der methode, nicht der name der methoden-funktion anzugeben.

9.1.3 objekt erzeugen

Durch den aufruf der konstruktor-funktion wird eine instanz, d.h. ein objekt mit den in dieser funktion definierten eigenschaften und methoden erzeugt. Man kann beliebig viele instanzen anlegen.

var *instanz* **new** *klasse*([*param* , . . .]);

instanz	name der instanz bzw. des objekts
klasse	name der konstruktor-funktion.
param	parameter, für den beim anlegen der instanz werte übergeben werden.

9.1.4 zugriff zu eigenschaften

Die eigenschaften eines objekts sind global, daher kann man darauf überall zugreifen, aber nur auf dem weg über eine vorhandene instanz.

var *wert* = *instanz.eigenschaft*;	lesen einer eigenschaft
instanz.eigenschaft = *irgendetwas*;	schreiben einer eigenschaft

9.1.5 verwendung einer methode

Eine methode kann wie jede andere funktion verwendet werden, allerdings kann man eine methode immer nur für eine vorhandene instanz des objekts aufrufen.

[*erg*] = *instanz.methode*([*param*, . . .]);

achtung

Eine methode wird mit dem namen der methode und nicht mit dem namen der methoden-funktion aufgerufen. Das ist der grund, weshalb man für beides die gleiche bezeichnung wählen sollte.

beispiel 9a

objekt-typ vereinbaren

Es wird die konstruktor-funktion **PERS** mit einigen eigenschaften und den beiden methoden **EIN** und **AUS** vereinbart. Für den konstruktor PERS sind zwei parameter vorgesehen, es müssen daher zwei argumente übergeben werden, es können aber auch mehr sein, weil im konstruktor die anzahl der übergebenen werte geprüft und dann entsprechend verfahren wird.

```
<script type="text/JavaScript">
//   konstruktor
function PERS (par1, par2)
{
    this.name = par1;
    this.vorn = par2;
    var x = PERS.arguments.length;
    if (x > 2)
        this.abt = PERS.arguments[2];
    else
        this.abt = 0;
    this.beitr = 0;
    this.EIN = eingabe;    //  methode EIN - funktion eingabe
    this.AUS = AUS;        //  methode AUS - funktion AUS
}
```

methoden

Die methode **EIN** (funktionsname **eingabe)** erwartet zwei argumente, die in eigenschafren des objekts geschrieben werden. Die methode **AUS** (funktionsname **AUS**) zeigt eigenschaften des objekts an.

```
function eingabe(abteil, bei)
{
    if (this.abt == abteil)
        this.beitr = bei;
    else
    {   this.abt   = abteil;
        this.beitr = bei;
    }
}

function AUS()
{
    var zeile = this.name + " " + this.vorn;
    zeile += " " + this.abt + " " + this.beitr;
    document.write(zeile + "<br />");
}
</script>
```

Es werden drei instanzen von **PERS** erzeugt, deren eigenschaften dann mit der methode **EIN** mit werten versorgt und dann mit der methode **AUS** angezeigt werden. Zuletzt wird die eigenschaft **name** der instanz **fall3** geändert und die eigenschaften der instanz erneut angezeigt. Die änderung erfolgt nicht mit hilfe einer methode, sondern durch einen direkten zugriff auf die eigenschaft.

```
<script type="text/JavaScript">
   var fall1 = new PERS("Valentin", "Karl", 1);
   var fall2 = new PERS("Karlstadt", "Liesel");
   var fall3 = new PERS("Rmbrmdng", "Wrdlbrmft", 3);
   fall1.EIN(1, 15.30);
   fall2.EIN(3, 10.00);
   fall3.EIN(2, 12.00);
   fall1.AUS();
   fall2.AUS();
   fall3.AUS();
   fall3.name = "Rembremerdeng";
   fall3.AUS();
</script>
```

```
Valentin Karl 1 15.3
Karlstadt Liesel 3 10
Rmbrmdng Wrdlbrmft 2 12
Rembremerdeng Wrdlbrmft 2 12
```

9.2 namenlose eigenschaft

In der konstruktor-funktion kann als erste eigenschaft eine eigenschaft ohne namen vereinbart werden. Gerne wird diese eigenschaft verwendet, um damit einen array zu vereinbaren, gelegentlich in der art, dass der array bei den verschiedenen instanzen eine unterschiedliche größe hat. Der umgang mit dieser eigenschaft ist gewöhnungsbedürftig und wird hier an einem kleinen beispiel vorgeführt.

beispiel 9b - namenlose eigenschaft
Es wird die konstruktor-funktion **VEREIN** vereinbart, der als erste eigenschaft einen unbenannten array enthält. Die größe des arrays wird durch einen aufruf-pararmeter bestimmt. Der array soll die anzahl der mitglieder in den einzelnen abteilungen enthalten.

```
<script type="text/JavaScript">
//   konstruktor
function VEREIN(n, par1, par2)
{    for (var x=0; x<=n; x++)    //  alle elemente auf 0
        this[x] = 0;
    this[0]    = n;              //  anzahl der abteilungen
    this.name  = par1;           //  vereinsname
    this.ort   = par2;           //  sitz des vereins
    this.FUELL = FUELL;          //  methode FUELL
    this.ZEIG  = ZEIG;           //  methode ZEIG
}
```

Die methode **FUELL** trägt für eine abteilung die anzahl der mitglieder in den unbenannten array ein.

```
function FUELL(n, zahl)
{
    this[n] = zahl;
}
```

Die methode **ZEIG** summiert die anzahl der mitglieder aller abteilungen und zeigt sie an.

```
function ZEIG()
{   var anz, x, zahl = 0;

    anz = this[0];               //  anzahl der abteilungen
    for (x=1; x<=anz; x++)       //  mitglieder summieren
        zahl += this[x];
    zeile = this.name + " in " + this.ort + "<br />";
    zeile += "hat in " + anz + " abteilungen <br />";
    zeile += zahl + " mitglieder";
    document.write(zeile);
}
</script>
```

Es wird die instanz **club** von **VEREIN** angelegt. Dann wird für die abteilungen 1 und 2 mit der methode **FUELL** die zahl der mitglieder eingetragen. Für die abteilung 4 wird die zahl der mitglieder direkt eingetragen, die schreibweise **club[4]** ist verwirrend, es sieht so aus als sei **club** ein array, ist es natürlich nicht, aber hinter dieser angabe verbirgt sich der unbenannter array in der instanz **club**, der so angesprochen wird. In der abteilung 3 gibt es keine mitglieder. Zuletzt erfolgt mit der methode **ZEIG** die anzeige.

```
<script type="text/JavaScript">
   var club = new VEREIN(4, "Lahme Ente", "Hintertupfing");
   club.FUELL(1, 120);
   club.FUELL(2, 80);
   club[4] = 65;
   club.ZEIG();
</script>
```

9.3 geschachtelte objekte

In ein objekt können weitere objekte geschachtelt werden, das geht recht einfach, die konstruktor-funktion eines geschachtelten objekts muss nur vor der konstruktor-funktion des objekts stehen, in das es geschachtelt ist.

beispiel 9c - geschachtelte objekte

Die konstruktor-funktion **ABTEIL** definiert die eigenschaft **abt** und drei instanzen von **FUNKT** in form des array **leit**. Außerdem werden die methoden **ZUGANG** und **ANZEIG** vereinbart. Die konstruktor-funktion **FUNKT** definiert nur zwei eigenschaften.

konstruktoren

```
function FUNKT()
{
    this.name = " ";
    this.vorn = " ";
}

function ABTEIL()
{
    this.abt = " ";
    this.leit = new Array(new FUNKT(), new FUNKT(), new FUNKT());
    this.ZUGANG = ZUGANG;
    this.ANZEIG = ANZEIG;
}
```

methoden

Die methode **ZUGANG** setzt in einer instanz des array **leit** die eigenschaften **name** und **vorn**. Man beachte die schreibweise, mit der man in ABTEIL eine eigenschaft von FUNKT erreicht.

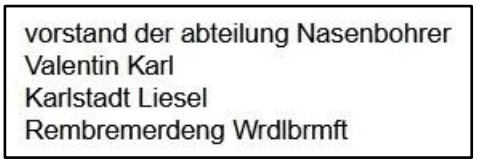
```
function ZUGANG (n, par1, par2)
{
   this.leit[n].name = par1;
   this.leit[n].vorn = par2;
}
```

Die methode **ANZEIG** gibt die eigenschaft **abt** von **ABTEIL** und die eigenschaften der instanzen von **FUNKT** in dem array **leit** aus.

```
function ANZEIG()
{   var anz, x;

    anz = this.leit.length;
    zeile = "vorstand der abteilung " + this.abt + "<br />";
    for (x=0; x<anz; x++)
        zeile += this.leit[x].name + " " + this.leit[x].vorn + "<br />";
    document.write(zeile);

}
```

Es wird die instanz **abteilung** von **ABTEIL** erzeugt; damit werden zugleich drei instanzen von **FUNKT** erzeugt. Die eigenschaft **abt** der instanz **abteilung** wird direkt versorgt. Mit der methode **ZUGANG** werden die beiden ersten instanzen von **FUNKT** im array **leit** versorgt. Die dritte instanz wird direkt versorgt. Zuletzt wir alles angezeigt.

```
<script type="text/JavaScript">
   abteilung = new ABTEIL();
   abteilung.abt = "Nasenbohrer";
   abteilung.ZUGANG(0, "Valentin", "Karl");
   abteilung.ZUGANG(1, "Karlstadt", "Liesel");
   abteilung.leit[2].name = "Rembremerdeng";
   abteilung.leit[2].vorn = "Wrdlbrmft";
   abteilung.ANZEIG();
</script>
```



stichworte

Die stichworte verweisen nicht auf seiten, sondern auf abschnitte.

alert()	3.1	objekt	1.2, 9.1
array	1.7	objekt bewegen	8.2, 8.3
array links	6.4	onblur	6.2
array-methoden	1.7	onclick	6.2, 6.3
ausblenden	5.3	onfocus	6.2
bewegtes objekt	8.2, 8.3	onkeypress	6.3
bildwechsel	7.2	onkeyup	6.2
charAt()	1.4	onload	6.3
charCodeAt()	1.4	onmousedown	6.3
className	5.1	onmousemove	6.3
clearInterval	3.2	onmouseover	6.3
clearTimeout	3.2	pop()	1.7
concat()	1.4	print()	3.1
confirm()	3.1	prompt()	3.1
einblenden	5.3	push()	1.7
eval	2.7	replace()	1.4
eventhandler	6.1	reverse()	1.7
forms	4.2	search()	1.4
+	5.2	setInterval	3.2
ormular ändern			
formular prüfen	4.3	setTimeout	3.2
formular-zugriff	4.2	shift()	1.7
galerien	7.3	sort()	1.7
image-objekt	7.1	StringfromCharCode()	1.4
indexOf()	1.4	String-Methoden	1.5
innerHTML	5.1	string-variable	1.3
instanz	1.2, 9.1	styles lesen	5.1
katz und maus	8.4	styles schreiben	5.1
klasse	1.2	substr()	1.4
konstruktor	9.1	substring()	1.4
laufschrift	8.1	unshift()	1.7
length	1.4	window-objekt	3.1
logische variable	1.3	zeichenkette	1.3
modale fenster	3.1	zeitgeber	3.2
numerische variable	1.3	zoomen	7.4