JavaScript - Kurzreferenz

Objektreferenz

Die Nummerierung vor den Objekten symbolisiert die JavaScript-Objekthierarchie und hat sonst keine weitere Bedeutung.

1	**window**	Anzeigefenster
1.1	**frames**	Frame-Fenster
1.2	**document**	Dokument im Anzeigefenster
1.2.1	**HTML-Elementobjekte**	Alle HTML-Elemente des Dokuments
1.2.2	**node**	Alle Knoten des Elementenbaums
1.2.3	**all**	Alle HTML-Elemente des Dokuments - Microsoft
1.2.3.1	**style**	CSS-Attribute von HTML-Elementen
1.2.4	**anchors**	Verweisanker im Dokument
1.2.5	**applets**	Java-Applets im Dokument
1.2.6	**forms**	Formulare im Dokument
1.2.6.1	**elements**	Formularelemente eines Formulars
1.2.6.1.1	**options**	Optionen einer Auswahlliste eines Formulars
1.2.7	**images**	Grafikreferenzen im Dokument
1.2.8	**embeds**	Multimedia-Referenzen im Dokument
1.2.9	**layers**	Layer im Dokument - Netscape
1.2.10	**links**	Verweise im Dokument
1.3	**event**	Anwendereereignisse
1.4	**history**	besuchte Seiten
1.5	**location**	URIs
2	**Array**	Ketten von Variablen
3	**Boolean**	Ja/Nein-Variablen
4	**Date**	Datum und Uhrzeit
5	**Function**	JavaScript-Funktionen
6	**Math**	Berechnungen
7	**navigator**	Browser-Informationen
7.1	**mimeTypes**	MIME-Typ-Informationen
7.2	**plugins**	installierte Plugins
8	**Number**	numerische Werte
9	**RegExp**	reguläre Ausdrücke
10	**screen**	Bildschirm-Informationen
11	**String**	Zeichenketten

Objekthierarchie

Es gibt zwei Gruppen von **vordefinierten Objekten** im klassischen JavaScript.

Die eine Gruppe ist jene, die das **gesamte Browser-Fenster** - markiert durch das **window**-Objekt - zum Ausgangspunkt hat. Das, was innerhalb davon als HTML-Seite angezeigt wird, gilt als Dokument, markiert durch das **document**-Objekt. Letzteres ist wiederum für das Document Object Model (DOM) das oberste Ausgangsobjekt. Im DOM ist der Zugriff auf **alle Elemente** eines Dokuments möglich - nach Syntax des Internet Explorers markiert durch das **all**-Objekt, und nach DOM-Syntax markiert durch bestimmte Zugriffsmethoden des `document`-Objekts, durch die **HTML-Elementobjekte** und das **node**-Objekt. (Objektnummern 1 bis 1.5)

Die zweite Gruppe von JavaScript-Objekten sind jene, die nichts direkt mit dem Geschehen im Anzeigefenster zu tun haben, aber wichtige andere Daten liefern oder Funktionen ausüben. Netscape bezeichnet sie als **core objects** (*Kernobjekte*). Typische Vertreter dieser zweiten Gruppe sind etwa das **Date**-Objekt für Datums- und Uhrzeitberechnungen, das **navigator**-Objekt für Informationen zum verwendeten Browser, oder das **Array**-Objekt zum Erzeugen von Serienvariablen. Auch bei dieser Gruppe gibt es ein paar hierarchische Beziehungen. So ist beispielsweise das **plugins**-Objekt ein Unterobjekt des `navigator`-Objekts. (Objektnummern 2 bis 11)

Objekthierarchien werden stets dadurch ausgedrückt, dass Objekte aneinander gereiht und durch Punkte getrennt werden. Typische Notationen sehen aus wie `window.document.images.length`.

Dabei steht `window` für das Browser-Fenster, in dem das JavaScript läuft, `document` für das in diesem Fenster angezeigte Dokument, `images` für die Gesamtheit der in dem Dokument enthaltenen Grafikreferenzen, und `length` für eine Eigenschaft des `images`-Objekts. Neben dieser einfachen Grundregel haben viele Objekte jedoch noch ihre eigenen Besonderheiten.

window

Eigenschaften:

closed (geschlossenes Fenster)
defaultStatus
(Normalanzeige in der Statuszeile)
innerHeight (Höhe des Anzeigebereichs)
innerWidth (Breite des Anzeigebereichs)
locationbar (Adresszeile)
menubar (Menüleiste)
name (Fenstername)
outerHeight (Höhe des gesamten Fensters)
outerWidth (Breite des gesamten Fensters)
pageXOffset (Fensterstartposition von links)
pageYOffset (Fensterstartposition von oben)
personalbar (Zeile für Lieblingsadressen)
scrollbars (Scroll-Leisten)
statusbar (Statuszeile)
status (Inhalt der Statuszeile)
toolbar (Werkzeugleiste)

Unterobjekte:

document
event
history
location

Methoden:

alert() (Dialogfenster mit Infos)
back() (zurück in History)
blur() (Fenster verlassen)
captureEvents() (Ereignisse überwachen)
clearInterval() (zeitliche Anweisungsfolge abbrechen)
clearTimeout() (Timeout abbrechen)
close() (Fenster schließen)
confirm() (Dialogfenster zum Bestätigen)
disableExternalCapture() (Fremdüberwachung verhindern)
enableExternalCapture() (Fremdüberwachung erlauben)
find() (Text suchen)
focus() (Fenster aktiv machen)
forward() (vorwärts in History)
handleEvent() (Ereignis übergeben)
home() (Startseite aufrufen)
moveBy() (bewegen mit relativen Angaben)
moveTo() (bewegen mit absoluten Angaben)
open() (neues Fenster öffnen)
print() (ausdrucken)
prompt() (Dialogfenster für Werteingabe)
releaseEvents() (Ereignisse abschließen)
resizeBy() (Größe verändern mit relativen Angaben)
resizeTo() (Größe verändern mit absoluten Angaben)
routeEvent() (Event-Handler-Hierarchie durchlaufen)
scrollBy() (Scrollen um Anzahl Pixel)
scrollTo() (Scrollen zu Position)
setInterval() (zeitliche Anweisungsfolge setzen)
setTimeout() (Timeout setzen)
stop() (abbrechen)

frames

Da jedes Frame-Fenster aus Sicht des Browsers ein eigenes Fenster darstellt, ist das `frames`-Objekt lediglich eine Variante des **window**-Objekts, also des allgemeinen Fensterobjekts. Alle Eigenschaften und Methoden, die zum `window`-Objekt gehören, aber auch alle anderen, die in der Hierarchie unterhalb des `window`-Objekts stehen, zum Beispiel Formulare, lassen sich auch auf das `frames`-Objekt, also auf einzelne Frame-Fenster anwenden. Beispiel:

```
window.document.forms[0].elements[0].value = "Stefan"
parent.frames[1].document.forms[0].elements[0].value = "Stefan"
```

Eigenschaft:

length (Anzahl Frames)

Unterobjekte:

document
event
history
location

document

Das `document`-Objekt bezieht sich auf den Inhalt, der in einem Browser-Fenster angezeigt wird. In der Objekthierarchie von JavaScript liegt es unterhalb des **window**-Objekts.
Im Document Object Model (DOM) des W3-Konsortiums ist das `document`-Objekt das Ausgangsobjekt für den Elementenbaum. Die Elemente eines HTML-Dokuments stellen dem DOM zufolge also Unterobjekte des `document`-Objekts dar. Die einzelnen HTML-Elemente können dabei eigene Eigenschaften und Methoden haben. Diese werden im Abschnitt über **HTML-Elementobjekte** beschrieben. Entscheidend für den Zugriff auf den HTML-Elementenbaum sind beim `document`-Objekt die DOM-Methoden **getElementById** und **getElementsByName**.

Darüber hinaus enthält das `document`-Objekt selber eine Reihe wichtiger Eigenschaften und Methoden, die dokumentglobale Eigenschaften speichern oder Funktionen ausführen. Die meisten davon werden auch vom DOM unterstützt.

Eigenschaften:

alinkColor (Farbe für Verweise beim Anklicken)
bgColor (Hintergrundfarbe)
charset (verwendeter Zeichensatz)
cookie (beim Anwender speicherbare Zeichenkette)
defaultCharset (normaler Zeichensatz)
fgColor (Farbe für Text)
lastModified (letzte Änderung am Dokument)
linkColor (Farbe für Verweise)
referrer (zuvor besuchte Seite)
title (Titel der Datei)
URL (URL-Adresse der Datei)
vlinkColor (Farbe für Verweise zu besuchten Zielen)

Unterobjekte:

HTML-Elemente
node
all
anchors
applets
forms
images
layers
links
plugins

Methoden:

captureEvents() (Ereignisse überwachen)
close() (schließen)
createAttribute() (Attributknoten erzeugen)
createElement() (Elementknoten erzeugen)
createTextNode() (Textknoten erzeugen)
getElementById() (HTML-Elementzugriff über id-Attribut)
getElementsByName() (HTML-Elementzugriff über name-Attribut)
getElementsByTagName() (HTML-Elementzugriff über Elementliste)
getSelection() (selektierter Text)
handleEvent() (Ereignisse verarbeiten)
open() (Dokument öffnen)
releaseEvents() (Ereignisse abschließen)
routeEvent() (Event-Handler-Hierarchie durchlaufen)
write() (ins Dokumentfenster schreiben)
writeln() (zeilenweise schreiben)

HTML-Elementobjekte

Elementobjekte

a	**basefont**	**center**	**dir**	**frame**	**img**	**link**	**optgroup**	**script**	**sup**	**title**
abbr	**bdo**	**cite**	**div**	**frameset**	**input**	**map**	**option**	**select**	**table**	**tr**
acronym	**big**	**code**	**dl**	**h1-h6**	**ins**	**menu**	**p**	**small**	**tbody**	**tt**
address	**blockquote**	**col**	**dt**	**head**	**isindex**	**meta**	**param**	**span**	**td**	**u**
applet	**body**	**colgroup**	**em**	**hr**	**kbd**	**noframes**	**pre**	**strike**	**textarea**	**ul**
area	**br**	**dd**	**fieldset**	**html**	**label**	**noscript**	**q**	**strong**	**tfoot**	**var**
b	**button**	**del**	**font**	**i**	**legend**	**object**	**s**	**style**	**th**	
base	**caption**	**dfn**	**form**	**iframe**	**li**	**ol**	**samp**	**sub**	**thead**	

Der HTML-Variante des Document Object Models (DOM) zufolge stellt jedes HTML-Element in einer HTML-Datei ein Objekt dar. Wichtig ist dabei zu wissen, wie mit einer Script-Sprache wie JavaScript auf ein solches HTML-Elementobjekt zugegriffen werden kann.

- Zugriff über Elementnamen sowie Elementnummer oder name-Attribut (ältere Elementlisten)
- Zugriff über das name-Attribut (getElementsByName):
- Zugriff über id-Attribut (getElementById):

- Zugriff über Elementnamen sowie Elementnummer, name-Attribut oder id-Attribut (getElementsByTagName)

Jedes HTML-Element hat Eigenschaften. Und zwar stellt jedes erlaubte Attribut eines HTML-Elements eine DOM-Eigenschaft dieses Elements dar. So hat beispielsweise das HTML-Element `input` ein erlaubtes Attribut `value`, und das HTML-Element `h1` hat ein erlaubtes Attribut `align`. Im DOM gibt es demnach also ein `input`-Elementobjekt mit der Eigenschaft `value`, und ein `h1`-Elementobjekt mit der Eigenschaft `align`.

Darüber hinaus definiert das DOM für einige der HTML-Elemente auch Methoden. So kann gibt es für das `form`-Elementobjekt (also das DOM-Objekt des HTML-Elements `form`) die Methoden `submit()` (Formular absenden) und `reset()` (Formulareingaben verwerfen).

Jedes HTML-Element stellt gemäß dem DOM außerdem einen **Knoten** im Elementenbaum dar. Deshalb gelten für jedes HTML-Element auch alle Eigenschaften und Methoden des **node**-Objekts.

Beachten Sie unbedingt die Groß-/Kleinschreibung der Eigenschaften und Methoden. Fehler bei der Groß-/Kleinschreibung führen zu Fehlern in JavaScript.

Universaleigenschaften (Wert von Universalattributen) – alle für Lesen und Schreiben geeignet:

Eigenschaft	Bedeutung
`className`	CSS-Klassenname
`dir`	Schreibrichtung
`id`	dokumentweit eindeutiger Name
`lang`	Landessprache (de, en, fr, it usw.)
`title`	Titel

node

Das `node`-Objekt ist das zentrale Objekt des Document Object Models (DOM) (*node* = *Knoten*). Hintergrund ist das Modell, dass ein Auszeichnungssprachen-Dokument, egal ob in HTML oder einer anderen, XML-basierten Auszeichnungssprache geschrieben, aus einem Baum von Knoten besteht.

Jedes Element, jedes Attribut und alle Zeichendaten stellen eigene Knoten dar. Diese Knoten bilden die Baumstruktur. Das `node`-Objekt stellt Eigenschaften und Methoden bereit, um auf die einzelnen Knoten zuzugreifen, egal, wie tief diese Knoten in der Baumstruktur liegen.

Das `node`-Objekt stellt damit die allgemeinere und für alle XML-gerechten Sprachen gültige Variante dessen dar, was die **HTML-Elementobjekte** speziell für HTML darstellen. Sie können in JavaScripts, die in HTML-Dateien notiert oder eingebunden sind, sowohl mit den HTML-Elementobjekten als auch mit dem `node`-Objekt arbeiten. Manches ist über die HTML-Elementobjekte bequemer zu lösen, für andere Aufgaben eignet sich wiederum das `node`-Objekt besser. Das `node`-Objekt gilt unter Puristen allerdings als das "reinere" DOM, eben weil es nicht auf HTML beschränkt ist.

Um auf die Eigenschaften und Methoden des `node`-Objekts zugreifen zu können, benötigen Sie einen Knoten. Um auf vorhandene Knoten im Dokument zuzugreifen, werden die Methoden des **document**-Objekts **getElementById**, **getElementsByName** und **getElementsByTagName** verwendet. Ausgehend davon können Sie die Attributknoten, Textknoten und weitere Element-Kindknoten eines Elements ansprechen.

Eigenschaften:

attributes (Attribute)
childNodes (Kindknoten)
data (Zeichendaten)
firstChild (erster Kindknoten)
lastChild (letzter Kindknoten)
nextSibling
(nächster Knoten auf derselben Ebene)
nodeName (Name des Knotens)
nodeType (Knotentyp)
nodeValue (Wert/Inhalt des Knotens)
parentNode (Elternknoten)
previousSibling (vorheriger Knoten auf derselben Ebene)

Methoden:

appendChild() (Kindknoten hinzufügen)
appendData() (Zeichendaten hinzufügen)
cloneNode() (Knoten kopieren)
deleteData() (Zeichendaten löschen)
getAttribute() (Wert eines Attributknotens ermitteln)
getAttributeNode() (Attributknoten ermitteln)
hasChildNodes() (auf Kindknoten prüfen)
insertBefore() (Knoten einfügen)
insertData() (Zeichendaten einfügen)
removeAttribute() (Attribut löschen)
removeAttributeNode() (Attributknoten löschen)
removeChild() (Knoten löschen)
replaceChild() (Kindknoten ersetzen)

replaceData() (Zeichendaten ersetzen)
setAttribute() (Wert eines Attributknotens setzen)
setAttributeNode() (Attributknoten erzeugen)

all

Das Objekt `all`, ist der Schlüssel zu **Dynamischem HTML** nach dem Ansatz des Internet Explorers ab Version 4.0. Mit Hilfe des `all`-Objekts haben Sie Script-Zugriff auf alle einzelnen Elemente und Inhalte einer HTML-Datei. Die meisten Eigenschaften können Sie lesen und ändern. Methoden des `all`-Objekts erlauben unter anderem das Einfügen oder Entfernen von HTML-Tags und von Angaben innerhalb eines HTML-Tags. Auf diese Weise ist der dynamische Zugriff auf alle Bestandteile einer Datei möglich.

Das `all`-Objekt gehört nicht zum offiziellen JavaScript-Sprachstandard. Es wurde von Microsoft für den Internet Explorer 4.0 implementiert. Das `all`-Objekt funktioniert zwar auch prima innerhalb von Script-Bereichen, die mit "JavaScript" ausgezeichnet sind, doch es ist bislang eigentlich nur Bestandteil von JScript, dem Microsoft-Derivat von JavaScript. Mit den neuen Browser-Generationen und dem Document Object Model (DOM) wird das `all`-Objekt durch die **HTML-Elementobjekte** und das **node**-Objekt verdrängt. Es hat also keine Zukunft mehr und sollte allenfalls noch aus Gründen der Rückwärtskompatibilität eingesetzt werden.

Eigenschaften:

className (Stylesheet-Klassenname eines Elements)
dataFld (Datenfeld bei Datenanbindung)
dataFormatAs (Datentyp bei Datenanbindung)
dataPageSize (Anzahl Datensätze bei Datenanbindung)
dataSrc (Datenquelle bei Datenanbindung)
id (id-Name eines Elements)
innerHTML (Inhalt eines Elements als HTML)
innerText (Inhalt eines Elements als Text)
isTextEdit (Editierbarkeit eines Elements)
lang (Sprache eines Elements)
language (Script-Sprache für ein Element)
length (Anzahl Elemente)
offsetHeight (Höhe eines Elements)
offsetLeft (Links-Wert der linken oberen Elementecke)
offsetParent (Elternelement-Position)
offsetTop (Obenwert der linken oberen Elementecke)
offsetWidth (Breite eines Elements)
outerHTML (Elementinhalt plus äußeres HTML)
outerText (Elementinhalt plus äußerem Text)
parentElement (Elternelement)
parentTextEdit (Editierbarkeit des Elternelements)
recordNumber (Datensatznummer bei Datenanbindung)
sourceIndex (wie vieltes Element)
tagName (HTML-Tag des Elements)
title (Titel des Elements)

Methoden:

click() (Element anklicken)
contains()
(Zeichenkette in Element enthalten)
getAttribute()
(Attribut in einem Element ermitteln)
insertAdjacentHTML() (Element einfügen)
insertAdjacentText() (Text einfügen)
removeAttribute()
(Attribut aus Element entfernen)
scrollIntoView() (zu Element scrollen)
setAttribute()
(Attribut eines Elements einfügen)

style

Das Objekt `style` liegt in der JavaScript-Objekthierarchie nach dem Objektmodell des Internet Explorers ab Version 4.0 unterhalb des **all**-Objekts und regelt den Zugriff auf **Stylesheet-Angaben**, die für ein HTML-Element definiert sind. Sie können alle Sheet-Angaben auslesen und dynamisch ändern. So können Sie HTML-Elemente mit Hilfe von Scripts nach Belieben umformatieren. Der Zugriff auf HTML-Elemente erfolgt dabei genau so wie beim `all`.

Auch im Document Object Model (DOM) der Version 2.0 gibt es das `style`-Objekt. Um auf die Eigenschaften und Methoden des `style`-Objekts nach DOM-Syntax zugreifen zu können, benötigen Sie einen Elementknoten. Um auf vorhandene Elementknoten im Dokument zuzugreifen, werden die Methoden des **document**-Objekts **getElementById**, **getElementsByName** und

getElementsByTagName verwendet. Ausgehend davon können Sie angesprochene Element mit Hilfe von Scripts umformatieren.

Methoden:

getAttribute() (Stylesheet-Angabe ermitteln)
removeAttribute() (Stylesheet-Angabe entfernen)
setAttribute() (Stylesheet-Angabe einfügen)

Style-Eigenschaften

Eine wichtige Regel müssen Sie kennen: Wenn in einem Script eine CSS-Angabe ausgelesen oder verändert werden soll, entfällt der Bindestrich, und der erste Buchstabe des Wortes hinter dem Bindestrich wird großgeschrieben. Die CSS-Eigenschaft `background-color` wird innerhalb eines JavaScripts also zu `backgroundColor`.

Die folgende Tabelle listet Style-Eigenschaften auf. Die Tabelle enthält links die Stylesheet-Angabe, wie Sie sie in JavaScript im Zusammenhang mit dem Style-Objekt notieren müssen. In der rechten Spalte steht eine Kurzbeschreibung, was die Angabe bewirkt.

JavaScript-Angabe	Kurzbeschreibung
`background`	Hintergrundbild
`backgroundAttachment`	Wasserzeichen-Effekt
`backgroundColor`	Hintergrundfarbe
`backgroundImage`	Hintergrundbild
`backgroundPosition`	Position des Hintergrundbilds
`backgroundRepeat`	Wallpaper-Effekt
`border`	Rahmen
`borderBottom`	Rahmen unten
`borderBottomColor`	Rahmenfarbe unten
`borderBottomStyle`	Rahmenart unten
`borderBottomWidth`	Rahmendicke unten
`borderColor`	Rahmenfarbe
`borderLeft`	Rahmen links
`borderLeftColor`	Rahmenfarbe links
`borderLeftStyle`	Rahmenart links
`borderLeftWidth`	Rahmendicke links
`borderRight`	Rahmen rechts
`borderRightColor`	Rahmenfarbe rechts
`borderRightStyle`	Rahmenart rechts
`borderRightWidth`	Rahmendicke rechts
`borderStyle`	Rahmenart
`borderTop`	Rahmen oben
`borderTopColor`	Rahmenfarbe oben
`borderTopStyle`	Rahmenart oben
`borderTopWidth`	Rahmendicke oben
`borderWidth`	Rahmendicke
`bottom`	Position von unten
`captionSide`	Tabellenbeschriftung
`clear`	Fortsetzung bei Textumfluss
`clip`	Anzeigebereich eingrenzen
`color`	Textfarbe
`cursor`	Mauszeiger
`direction`	Schreibrichtung
`display`	Sichtbarkeit (ohne Platzhalter)
`emptyCells`	Darstellung leerer Tabellenzellen
`cssFloat`	Textumfluss
`font`	Schrift
`fontFamily`	Schriftart

fontSize	Schriftgröße
fontStretch	Schriftlaufweite
fontStyle	Schriftstil
fontVariant	Schriftvariante
fontWeight	Schriftgewicht
height	Höhe eines Elements
left	Position von links
letterSpacing	Zeichenabstand
lineHeight	Zeilenhöhe
listStyle	Listendarstellung
listStyleImage	Grafik für Aufzählungslisten
listStylePosition	Listeneinrückung
listStyleType	Darstellungstyp der Liste
margin	Abstand/Rand
marginBottom	Abstand/Rand unten
marginLeft	Abstand/Rand links
marginRight	Abstand/Rand rechts
marginTop	Abstand/Rand oben
maxHeight	Maximalhöhe
maxWidth	Maximalbreite
minHeight	Mindesthöhe
minWidth	Mindestbreite
overflow	übergroßer Inhalt
padding	Innenabstand
paddingBottom	Innenabstand unten
paddingLeft	Innenabstand links
paddingRight	Innenabstand rechts
paddingTop	Innenabstand oben
pageBreakAfter	Seitenumbruch danach
pageBreakBefore	Seitenumbruch davor
position	Positionsart
right	Position von rechts
scrollbar3dLightColor	Farbe für 3D-Effekte (Scrollbars)
scrollbarArrowColor	Farbe für Verschiebepfeile (Scrollbars)
scrollbarBaseColor	Basisfarbe der Scroll-Leiste (Scrollbars)
scrollbarDarkshadowColor	Farbe für Schatten (Scrollbars)
scrollbarFaceColor	Farbe für Oberfläche (Scrollbars)
scrollbarHighlightColor	Farbe für oberen und linken Rand (Scrollbars)
scrollbarShadowColor	Farbe für unteren und rechten Rand (Scrollbars)
scrollbarTrackColor	Farbe für freibleibenden Verschiebeweg (Scrollbars)
tableLayout	Tabellentyp
textAlign	Ausrichtung
textDecoration	Textdekoration
textIndent	Texteinrückung
textTransform	Text-Transformation
Top	Position von oben
verticalAlign	vertikale Ausrichtung
visibility	Sichtbarkeit (mit Platzhalter)
width	Breite eines Elements
wordSpacing	Wortabstand
zIndex	Schichtposition bei Überlappung

anchors

Mit dem Objekt `anchors` haben Sie Zugriff auf Verweisanker, die in einer HTML-Datei definiert sind. Ein Verweisanker in HTML ist beispielsweise:
`<a name="top">Hier beginnt die Seite</a>`.

Verweisanker können Sie auf zwei Arten ansprechen:

- mit einer Indexnummer:
  Bei Verwendung von Indexnummern geben Sie `document.anchors` an und dahinter in eckigen Klammern, den wievielten Anker in der Datei Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. den ersten Verweisanker sprechen Sie mit `anchors[0]` an, den zweiten Verweisanker mit `anchors[1]` usw. Beim Zählen gilt die Reihenfolge, in der die Verweisanker in der Datei notiert sind. Beispiel: document.anchors[0].name
- mit Namen des Ankers als Indexnamen:
  Dabei notieren Sie wie beim Ansprechen mit Indexnummer hinter `document.anchors` eckige Klammern. Innerhalb der eckigen Klammern notieren Sie in Anführungszeichen den Namen, den Sie bei der Definition des Verweisankers im einleitenden <a>-Tag im `name`-Attribut angegeben haben. Beispiel: document.anchors["oben"].text

Im Internet Explorer ist es nicht möglich, mit dem Schema 2 auf einen Verweisanker zuzugreifen. Sie können aber jeden Anker über das **all**-Objekt und dessen Eigenschaften ansprechen.

Eigenschaft:

name (Name des Verweisankers)
length (Anzahl der Verweisanker)
text (Text des Verweisankers)
x (horizontale Position des Verweisankers)
y (vertikale Position des Verweisankers)

applets

Mit dem Objekt `applets` haben Sie Zugriff auf Java-Applets, die in einer HTML-Datei definiert sind. Die einzige gewöhnliche JavaScript-Objekteigenschaft des applet-Objekts ist die Anzahl der Java-Applets in einer Datei.

Über das applets-Objekts haben Sie jedoch auch **Zugriff auf Code in Java-Applets**. Dazu müssen Sie das gewünschte Java-Applet ansprechen. Java-Applets können Sie auf drei Arten ansprechen:

- mit einer Indexnummer:
  Bei Verwendung von Indexnummern geben Sie `document.applets` an und dahinter in eckigen Klammern, das wie vielte Java-Applet in der Datei Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. das erste Java-Applet sprechen Sie mit `applets[0]` an, das zweite Java-Applet mit `applets[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<applet>`-Befehle in der Datei notiert sind. Beispiel: document.applets[#].Code();
- mit Namen des Java-Applets als Indexnamen:
  Dabei notieren Sie wie beim Ansprechen mit Indexnummer hinter `document.applets` eckige Klammern. Innerhalb der eckigen Klammern notieren Sie in Anführungszeichen den Namen, den Sie bei der Definition des Java-Applets im einleitenden `<applet>`-Tag im Attribut `name` angegeben haben. Beispiel: document.applets["AppletName"].Code();
- mit dem Namen des Java-Applets direkt:
  Dabei geben Sie mit `document.AppletName` den Namen an, den Sie bei der Definition des Java-Applets im einleitenden `<applet>`-Tag im Attribut `name` angegeben haben. Beispiel: document.AppletName.Code();

Eigenschaft:

length (Anzahl Java-Applets)

forms

Mit dem Objekt `forms` haben Sie Zugriff auf Formulare, die in einer HTML-Datei definiert sind. Formulare können Sie auf drei Arten ansprechen:

- mit einer Indexnummer:
  Bei Verwendung von Indexnummern geben Sie `document.forms` an und dahinter in eckigen Klammern, das wie vielte Formular in der Datei Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. das erste Formular sprechen Sie mit `forms[0]` an, das zweite Formular mit `forms[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<form>`-Tags in der Datei notiert

sind. Sie können zwischen den eckigen Klammern auch eine Number-Variable notieren, die die Indexnummer enthält. Beispiel: Ziel = document.forms[0].action;

- mit dem Namen des Formulars direkt:
  Dabei geben Sie mit `document.Formularname` den Namen an, den Sie bei der Definition des Formulars im einleitenden `<form>`-Tag im Attribut `name` angegeben haben. Beispiel: Ziel = document.Testformular.action;
- mit Namen des Formulars als Indexnamen:
  Dabei notieren Sie wie beim Ansprechen mit Indexnummer hinter `document.forms` eckige Klammern. Innerhalb der eckigen Klammern notieren Sie in Anführungszeichen den Namen, den Sie bei der Definition des Formulars im einleitenden `<form>`-Tag im Attribut `name` angegeben haben. Sie können zwischen den eckigen Klammern auch eine String-Variable notieren, die den Formularnamen enthält. Beispiel: document.forms["Testformular"].reset();

Die Eigenschaften und Methoden des forms-Objekts betreffen nur Bestandteile des gesamten Formulars, etwa die Kodiermethode. Um auf einzelne Eingabefelder, Buttons usw. zuzugreifen, steht das **elements**-Objekts mit seinen Unterobjekten zur Verfügung.

Eigenschaften:

action (Empfängeradresse bei Absenden)
encoding (Kodierungstyp)
length (Anzahl Formulare in Datei)
method (Übertragungsmethode)
name (Formularname)
target (Zielfenster für CGI-Antworten)

Methoden:

handleEvent() (Ereignis verarbeiten)
reset() (Formulareingaben löschen)
submit() (Formular abschicken)

Unterobjekte:

elements

elements

Formularelemente können Sie auf folgende Arten ansprechen:

- mit einer Indexnummer (wie in Schema 1 / Beispiel 1)
  Beispiel: document.forms[0].elements[0].value = "Unsinn";
- mit Namen (wie in Schema 2 / Beispiel 2)
  Dabei geben Sie mit `document.FormularName.ElementName` den Namen des Formulars und des Elements an, den Sie bei der Definition des Formulars und des Elements in den entsprechenden HTML-Tags im Attribut `name` angegeben haben.
- mit Namen (wie in Schema 3 / Beispiel 3)
  Dabei geben Sie mit `document.forms["Formularname"].elements["Elementname"]` den Namen des Formulars und des Elements als String an. Diese Notation wird insbesondere für den Zugriff auf Formularelemente benötigt, deren Namen Sonderzeichen enthalten, welche den Zugriff nach Schema 2 unmöglich machen. Sie können zwischen den eckigen Klammern auch eine String-Variable notieren, die den Formularnamen bzw. den Elementnamen enthält.
- Ein Mischen der Zugriffsvarianten 1, 2 und 3 für die Adressierung des Formulars und seiner Elemente ist möglich. Beispiel:
  `document.Formularname.elements["Elementname"].value`

Checkboxen und Radio-Buttons bilden normalerweise Gruppen mehrerer zusammen gehöriger Elemente, die einen gleich Wert im `name`-Attribut haben und sich nur durch das `value`-Attribut unterscheiden. Solche Gruppen mit gleichen Namen bilden in JavaScript wiederum einen Array. Beispiel: document.Formular.Favoriten[2].checked = true;

Eigenschaften:

checked (Angekreuzt)
defaultChecked (vorangekreuzt)
defaultValue (voreingegebener Wert)
form (Name des zugehörigen Formulars)
length (Anzahl der Elemente eines Formulares)
name (Elementname)
type (Elementtyp)
value (Elementwert/-inhalt)

Methoden:

blur() (Element verlassen)
click() (Element anklicken)
focus() (auf Element positionieren)
handleEvent() (Ereignis verarbeiten)
select() (Text selektieren)

Unterobjekte:

options

Mit dem Objekt `options`, das in der JavaScript-Objekthierarchie unterhalb des **elements**-Objekts liegt, haben Sie Zugriff auf **Auswahllisten** innerhalb eines Formulars. Sie können dabei auf jede einzelne Auswahlmöglichkeit der Auswahlliste zugreifen. Auswahllisten sind ganz normale Formularelemente. Auswahllisten sprechen Sie also an wie andere Formularelemente auch:

- mit einer Indexnummer (wie in Schema 1 / Beispiel 1)
- mit Namen (wie in Schema 2 / Beispiel 2)
- mit Namen (wie in Schema 3 / Beispiel 3)

Sie können innerhalb eines JavaScripts neue Elemente zu einer Auswahlliste hinzufügen oder vorhandene Einträge durch neue ersetzen. Dazu müssen Sie mit Hilfe von JavaScript ein neues `option`-Objekt erzeugen und es einer Auswahlliste zuordnen:

```
function Hinzufuegen () {
  NeuerEintrag = new Option(document.Testform.neu.value,
document.Testform.neu.value, false, true);
  document.Testform.Auswahl.options[document.Testform.Auswahl.length] =
NeuerEintrag;
  document.Testform.neu.value = "";
}
// new Option() kennt vier Parameter (die drei letzten Parameter optional).
// 1. text = angezeigter Text in der Liste
// 2. value = zu übertragender Wert der Liste
// 3. defaultSelected = true übergeben, wenn der Eintrag der
//    defaultmäßig vorselektierte Eintrag sein soll, sonst false
// 4. selected = true übergeben, wenn der Eintrag selektiert werden soll
```

Eigenschaften:

defaultSelected (voreingestellte Auswahl)
length (Anzahl der Auswahlmöglichkeiten)
selected (aktuelle Auswahl)
selectedIndex (Index der aktuellen Auswahl)
text (Auswahltext)
value (Auswahlwert)

images

Mit dem Objekt `images`, das in der JavaScript-Objekthierarchie unterhalb des **document**-Objekts liegt, haben Sie Zugriff auf alle Grafiken, die in einer HTML-Datei definiert sind. Dabei können Sie auch vorhandene Grafiken dynamisch durch andere ersetzen.

Ein neues Grafik-Objekt wird automatisch erzeugt, wenn der Web-Browser eine **Grafik** in der HTML-Datei vorfindet. Grafikobjekte können Sie auf zwei Arten ansprechen:

- mit einer Indexnummer (wie in Schema 1 / Beispiel 1)
- mit dem Namen der Grafik (wie in Schema 2 / Beispiel 2)
- mit dem Namen der Grafik (wie in Schema 3 / Beispiel 3)

Für Grafiken, die Sie nachträglich mit JavaScript anzeigen möchten, müssen Sie jedoch eigene neue Grafikobjekte in JavaScript erzeugen. Das ist besonders dann wichtig, wenn Sie Grafiken dynamisch durch andere Grafiken ersetzen wollen:

```
Zweitbild = new Image(104, 102);
Zweitbild.src = "beispiel2.gif";
function Bildwechsel () {
```

```
}
// Die Objektfunktion Image() kennt zwei optionale Parameter:
// 1. width Breite des Bildes
// 2. height Höhe des Bildes
```

Eigenschaften:

border (Rahmen)
complete (Ladezustand)
height (Höhe)
hspace (horizontaler Abstand)
length (Anzahl Grafiken)
lowsrc (Vorschaugrafik)
name (Name)
src (Grafikdatei)
vspace (vertikaler Abstand)
width (Breite)

Methode:

handleEvent() (Ereignis verarbeiten)

embeds

Mit dem Objekt `embeds`, das in der JavaScript-Objekthierarchie unterhalb des **document**-Objekts liegt, haben Sie Zugriff auf alle Multimedia-Elemente, die in einer HTML-Datei mit dem Netscape-Element `<embed>...</embed>` definiert sind. Dabei können Sie z.B. den Abspielvorgang von Sound-Dateien oder Videos dynamisch starten.

Ein neues Objekt dieser Art wird automatisch erzeugt, wenn der Web-Browser eine **Multimedia-Referenz nach Netscape-Syntax** in der HTML-Datei vorfindet.

Solche Objekte können Sie auf zwei Arten ansprechen:

- mit einer Indexnummer (wie in Schema 1 / Beispiel 1)
  Bei Verwendung von Indexnummern geben Sie `document.embeds` an und dahinter in eckigen Klammern, das wie vielte `embed`-Element in der Datei Sie meinen. Jedes Objekt, das in HTML mit dem `<embed>`-Tag notiert wurde, zählt. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. die erste Multimedia-Referenz sprechen Sie mit `embeds[0]` an, die zweite mit `embeds[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<embed>`-Tags in der Datei notiert sind.
- mit dem Namen der Referenz (wie in Schema 2 / Beispiel 2)
  Dabei geben Sie mit `document.embeds["Objektname"]` den Namen an, den Sie bei der Definition der Multimedia-Referenz im einleitenden `<embed>`-Tag im Attribut `name` angegeben haben.

Eigenschaften:

height (Höhe des eingebetteten Objekts)
hspace (horizontaler Abstand des eingebetteten Objekts)
length (Anzahl eingebetteter Objekte)
name (Name des eingebetteten Objekte)
src (Quelle des eingebetteten Objekts)
width (Breite des eingebetteten Objekts)
type (MIME-Typ des eingebetteten Objekts)
vspace (vertikaler Abstand des eingebetteten Objekts)

Methoden:

play()
stop()

layers

Mit dem Objekt `layers`, das in der JavaScript-Objekthierarchie unterhalb des **document**-Objekts liegt, haben Sie Zugriff auf alle Layer, die in einer HTML-Datei definiert sind. Dieses Objekt ist - ebenso wie die entsprechenden HTML-Tags - Netscape-4-spezifisch. Es ist die Grundlage für **Dynamisches Positionieren bei Netscape 4.x** (dort finden Sie auch weitere Beispiele). Ein neues Layer-Objekt wird automatisch erzeugt, wenn der Web-Browser einen **Layer** in der HTML-Datei vorfindet.

Das `layers`-Objekt wird von Netscape ab Version 6.0 **nicht mehr** unterstützt. Es sollte also nur noch aus Gründen der Rückwärtskompatibilität zu Netscape 4.x Verwendung finden, da kein einziger moderner Browser dieses Objekt interpretiert.

Eigenschaften:

above (oberhalb liegender Layer)
background (Hintergrundbild eines Layers)
bgColor (Hintergrundfarbe eines Layers)
below (unterhalb liegender Layer)
clip (Anzeigebereich eines Layers)
document (document-Objekt eines Layers)
left (Links-Wert der linken oberen Ecke relativ)
length (Anzahl Layer)
name (Name eines Layers)
pageX (Links-Wert der linken oberen Ecke absolut)
pageY (Oben-Wert der linken oberen Ecke absolut)
parentLayer (Objekt des Eltern-Layers)
siblingAbove (Objekt des oberhalb liegenden Layers)
siblingBelow (Objekt des unterhalb liegenden Layers)
src (Externe Datei eines Layers)
top (Oben-Wert der linken oberen Ecke relativ)
visibility (Sichtbarkeit eines Layers)
zIndex (Schichtposition eines Layers)

Methoden:

captureEvents() (Ereignisse überwachen)
handleEvent() (Ereignisse behandeln)
load() (externe Datei laden)
moveAbove() (über einen anderen Layer legen)
moveBelow() (unter einen anderen Layer legen)
moveBy() (bewegen um Anzahl Pixel)
moveTo() (bewegen zu Position relativ)
moveToAbsolute() (bewegen zu Position absolut)
releaseEvents() (Ereignisüberwachung beenden)
resizeBy() (Breite und Höhe verändern um Anzahl Pixel)
resizeTo() (Breite und Höhe auf Anzahl Pixel setzen)
routeEvent() (Event-Handler-Hierarchie durchlaufen)

links

Der Zugriff auf die Verweise erfolgt mit Indexnummern. Dabei geben Sie `document.links` an und dahinter in eckigen Klammern, den wie vielten Verweis in der Datei Sie meinen, beginnend mit 0. Neben den verweisspezifischen Eigenschaften kennt das Link-Objekt auch alle Eigenschaften des **location**-Objekts.

Eigenschaft:

name (Name des Verweises)
length (Anzahl Verweise)
target (Zielfenster des Verweises)
text (Text des Verweises)
x (horizontale Position des Verweises)
y (vertikale Position des Verweises)

event

Mit dem Objekt `event` können Sie Einzelinformationen zu Anwenderereignissen wie Mausklicks oder Tasteneingaben ermitteln und weiterverarbeiten. So können Sie z.B. bei einem Mausklick die genaue Position ermitteln, wo der Mausklick erfolgte, oder bei einem Tastendruck die gedrückte Taste abfragen.

Anwenderereignisse können Sie entweder überwachen, indem Sie in einem erlaubten HTML-Tag einen **Event-Handler** notieren, oder, indem Sie direkt mit Hilfe von JavaScript eine Ereignisüberwachung programmieren. Für den Fall, dass das überwachte Ereignis eintritt, können Sie eine Handler-Funktion schreiben, die das Ereignis "behandelt", also verarbeitet. Die Handler-Funktion wird automatisch aufgerufen, wenn das Ereignis eintritt. Innerhalb einer Handler-Funktion besteht auch die Möglichkeit, Eigenschaften des eingetretenen Ereignisses abzufragen.

Die meisten Browser kennen das Event-Objekt, aber leider ist die Implementierung völlig unterschiedlich gelöst. Nicht alle Browser kennen dieselben Eigenschaften und auch in der Syntax zur Überwachung von Ereignissen unterscheiden sie sich. Das gilt sowohl bei der Ereignisüberwachung per Event-Handler in HTML als auch für die direkte Ereignisüberwachung in JavaScript.

Das allein genügt jedoch noch nicht zur Ereignisüberwachung. Damit die Handler-Funktionen bei Eintritt des Ereignisses automatisch aufgerufen werden, muss die Ereignisüberwachung gestartet

werden. Beispiel: `document.onkeydown = TasteGedrueckt;`. In den meisten Fällen reicht es jedoch aus, Ereignisse nicht dokumentweit, sondern nur für einzelne Elemente zu überwachen.

Eigenschaften:

altKey, ctrlKey, shiftKey (Sondertasten/Microsoft)
button (Maustastencode/Microsoft)
clientX, clientY (Bildschirmkoordinaten/Microsoft)
keyCode (Tastaturcode/Microsoft)
layerX, layerY (objekt-relative Koordinaten/Netscape)
modifiers (Sondertasten/Netscape)
offsetX, offsetY (objekt-relative Koordinaten/Microsoft)
pageX, pageY (fenster-relative Koordinaten/Netscape)
screenX, screenY (Bildschirmkoordinaten/Netscape)
which (Tastatur-/Maustastencode/Netscape)
type (Art des Ereignisses/Netscape)
x,y (Elternelement-relative Koordinaten/Microsoft)

history

Über das Objekt **history** haben Sie Zugriff auf die besuchten WWW-Seiten des Anwenders. Maßgeblich ist dabei die Liste, wie sie in der History-Liste des Web-Browsers gespeichert ist. Mit diesem Objekt können Sie z.B. Verweise vom Typ "springe zur zuletzt besuchten Seite" konstruieren. JavaScript erlaubt zum Schutz der Seitenbesucher nur einen begrenzten Zugriff auf das `history`-Objekt. Das bedeutet, Sie erhalten keinen Zugriff auf die URIs der besuchten Seitenadressen, und Sie können die History auch nicht löschen. Neben der Bewegung in der History ist es lediglich erlaubt, mit der Methode **location.replace()** den `history`-Eintrag der zuletzt besuchten Seite zu überschreiben.

Eigenschaften:

length (Anzahl besuchter Seiten)

Methoden:

back() (zurückspringen)
forward() (vorwärtsspringen)
go() (zu URI in History springen)

location

Über das Objekt `location` haben Sie Zugriff auf den vollständigen **URI** der aktuell angezeigten Web-Seite. Sie können den URI oder Teile davon zur Weiterverarbeitung abfragen und ändern. Beim Ändern führt der Web-Browser einen Sprung zu einem neuen URI aus, genau so wie bei einem Verweis.

Eigenschaften:

hash (Ankername innerhalb eines URI)
host (Domain-Name innerhalb eines URI)
hostname (Domain-Name innerhalb eines URI)
href (URI / Verweis zu URI)
pathname (Pfadname innerhalb eines URI)
port (Portangabe innerhalb eines URI)
protocol (Protokollangabe innerhalb eines URI)
search (Parameter innerhalb eines URI)

Methoden:

reload() (neu laden)
replace() (History-Eintrag überschreiben)

Array

Das Objekt `Array` ist als "Container" für Ketten gleichartiger Variablen gedacht. In der Programmiersprache spricht man auch von einem "Vektor". Wenn Sie beispielsweise die 16 Grundfarben speichern wollen, brauchen Sie keine 16 Variablen, sondern ein Array-Objekt, in dem Sie 16 gleichartige Werte (im Beispiel: Farbwerte) speichern können.

Eine Objektinstanz von `Array` speichern Sie in einem selbst vergebenen Objektnamen. Hinter dem Namen folgt ein Gleichheitszeichen. Dahinter folgt das reservierte Wort `new` und der Aufruf der Objektfunktion `Array()`.

Benutzen Sie Schema 1, wenn Sie zum Zeitpunkt der Definition noch nicht wissen, wie viele Elemente in dem Variablenvektor gespeichert werden sollen:

```
Objektname = new Array();
```

Benutzen Sie Schema 2, wenn Sie zum Zeitpunkt der Definition bereits wissen, wie viele Elemente in dem Variablenvektor gespeichert werden sollen. Die Anzahl können Sie der Objektfunktion als Parameter übergeben:

```
Objektname = new Array(Zahl);
```

Benutzen Sie Schema 3, um den Variablenvektor gleich mit Anfangswerten vorzubelegen. Bei den Varianten 1 und 2 bleiben die einzelnen Variablen des Variablenvektors leer, bis ihnen im Programmverlauf ein Wert zugewiesen wird:

```
Objektname = new Array(Element0, Element1, ..., element_n);
```

Nachdem Sie eine Instanz des Array-Objekts erzeugt haben, können Sie dies in Ihrem JavaScript-Code verwenden.

Sie können auch mehrdimensionale Arrays erzeugen. Das Beispiel definiert zunächst einen Array `a` mit 4 Elementen. Dann wird in einer **for-Schleife** für jedes dieser Elemente ein neuer Array definiert, wobei für jeden Array 10 leere Elemente erzeugt werden. Anschließend können Sie durch eine Angabe wie `a[3][1]` das zweite Element (1) im vierten Array (3) ansprechen:

```
var a = new Array(4);
for (var i = 0; i < a.length; ++i)
  a[i] = new Array(10);
a[3][1] = "Hallo";
alert(a[3][1]);
```

Als assoziative Arrays bezeichnet man Sammlungen von Elementen, bei denen der Zugriff auf einzelne Elemente mit Hilfe einer Zeichenkette erfolgt. Die Zeichenkette wird als *Schlüssel* für den Zugriff bezeichnet. **Im Gegensatz zu anderen Programmiersprachen gibt es in JavaScript keine assoziativen Arrays.** Arrays in JavaScript erlauben den Zugriff auf die Elemente lediglich über Indexnummern. Man kann jedoch mit Hilfe von `Object()` das Verhalten eines assoziativen Arrays teilweise nachbauen.

Eigenschaften:

length (Anzahl Elemente)

Methoden:

concat() (Arrays verketten)
join() (Array in Zeichenkette umwandeln)
pop() (letztes Array-Element löschen)
push() (neue Array-Elemente anhängen)
reverse() (Elementreihenfolge umkehren)
shift() (Erstes Array-Element entfernen)
slice() (Teil-Array extrahieren)
splice() (Elemente löschen und hinzufügen)
sort() (Array sortieren)
unshift() (Elemente am Array-Anfang einfügen)

Boolean

Das Objekt `Boolean` ist zum Erzeugen von JavaScript-Standardwerten `true` (*wahr*) und `false` (*falsch*) gedacht. Solche Werte sind vor allem als Rückgabewerte für Funktionen gedacht. Boole'sche Objekte werden bei der Definition immer mit einem der beiden möglichen Werte initialisiert und behalten diesen Wert auch. Die Variablen, in denen der Initialisierungswert gespeichert wird, stellen also Konstanten dar. Beispiele:

```
wahr = new Boolean(true); Fehler = new Boolean(false);
```

Date

Das Objekt `Date` ist für alle Berechnungen mit Datum und Zeit zuständig. Dabei gibt es, wie in der EDV üblich, einen fixen historischen Zeitpunkt, der intern als Speicherungs- und Berechnungsbasis dient. In JavaScript ist dies - wie in der C- und Unix-Welt - der 1. Januar 1970, 0.00 Uhr. Die Einheit, in der in JavaScript intern Zeit berechnet wird, ist eine Millisekunde.

Bevor Sie über JavaScript Zugriff auf die eingebauten Datums- und Uhrzeitfunktionen haben, müssen Sie ein neues `Date`-Objekt erzeugen. Dabei gibt es mehrere Varianten.

Benutzen Sie die Variante 1, wenn Sie eine Objektinstanz erzeugen wollen, in der das zum Zeitpunkt der Programmausführung aktuelle Datum und die aktuelle Uhrzeit gespeichert werden soll:

```
Objektname = new Date();
```

Benutzen Sie eine der anderen Varianten, wenn Sie das neue Datumobjekt mit bestimmten Werten (also einem bestimmten Datum und einer bestimmten Uhrzeit) initialisieren wollen. Alle Initialisierungswerte wie Monat oder Minuten müssen Sie in Zahlenform angeben beginnend mit 0, also etwa `9` für Monat Oktober.

```
Objektname = new Date(Jahr,Monat,Tag [,Stunden,Minuten,Sekunden]);
```

Methoden:

getDate() (Monatstag ermitteln)
getDay() (Wochentag ermitteln)
getFullYear() (volles Jahr ermitteln)
getHours() (Stundenteil der Uhrzeit ermitteln)
getMilliseconds() (Millisekunden ermitteln)
getMinutes() (Minutenteil der Uhrzeit ermitteln)
getMonth() (Monat ermitteln)
getSeconds() (Sekundenteil der Uhrzeit ermitteln)
getTime() (Zeitpunkt ermitteln)
getTimezoneOffset() (Zeitzonenabweichung der Lokalzeit ermitteln)
getUTCDate() (Monatstag von UTC-Zeit ermitteln)
getUTCFullYear() (volles Jahr von UTC-Zeit ermitteln)
getUTCHours() (Stundenteil der UTC-Uhrzeit ermitteln)
getUTCMilliseconds() (Millisekundenteil der UTC-Uhrzeit ermitteln)
getUTCMinutes() (Minutenteil der UTC-Uhrzeit ermitteln)
getUTCMonth()(Monat von UTC-Uhrzeit ermitteln)
getUTCSeconds()(Sekundenteil der UTC-Uhrzeit ermitteln)
getYear() (Jahr ermitteln)
parse() (Millisekunden seit dem 1.1.1970 ermitteln)
setDate() (Monatstag setzen)
setFullYear() (volles Jahr setzen)
setHours() (Stunden der Uhrzeit setzen)
setMilliseconds() (Millisekunden setzen)
setMinutes() (Minuten der Uhrzeit setzen)
setMonth() (Monat setzen)
setSeconds() (Sekunden der Uhrzeit setzen)
setTime() (Datum und Uhrzeit setzen)
setUTCDate() (Monatstag für UTC-Zeit setzen)
setUTCFullYear() (volles Jahr für UTC-Zeit setzen)
setUTCHours() (Stunden der UTC-Zeit setzen)
setUTCMilliseconds() (Millisekunden der UTC-Zeit setzen)
setUTCMinutes() (Minuten der UTC-Zeit setzen)
setUTCMonth() (Monat für UTC-Zeit setzen)
setUTCSeconds() (Sekunden der UTC-Zeit setzen)
setYear() (Datum und Uhrzeit setzen)
toGMTString() (Zeitpunkt in GMT-Format konvertieren)
toLocaleString() (Zeitpunkt in lokales Format konvertieren)
UTC() (GMT-Zeit seit dem 1.1.1970 ermitteln)

Function

Über das Objekt `Function` haben Sie Zugriff auf Eigenschaften einer JavaScript-**Funktion**. JavaScript-Funktionen werden dadurch also selbst Gegenstand von JavaScript-Anweisungen:

```
Farbe = new Function("NeueFarbe", "document.bgColor = NeueFarbe;");
```

Eine Variable, in der eine Funktion gespeichert ist, wie im Beispiel die Variable `Farbe`, können Sie genauso aufrufen wie eine Funktion.

```
function Farbe (NeueFarbe) { document.bgColor = NeueFarbe;}
```

Sinnvoll ist das Arbeiten mit dem Function-Objekt beispielsweise im Zusammenhang mit einer variablen Anzahl von Parametern in einer Funktion.

Eigenschaften:

arguments (Argumentnamen-Array)
arity (Anzahl Argumente)
caller (Namen der aufrufenden Funktion)

Math

Mit dem Objekt `Math` können Sie Berechnungen, auch komplexe wissenschaftlicher oder kaufmännischer Art, durchführen. Dazu stehen Ihnen verschiedene mächtige Methoden und Funktionen sowie einige Eigenschaften zur Verfügung.

Eine Instanz von `Math` brauchen Sie nicht eigens erzeugen. Sie können Eigenschaften und Methoden von `Math` direkt verwenden: `Zahl = 10 * Math.PI`

Eigenschaften:

E (Eulersche Konstante)
LN2 (natürlicher Logarithmus von 2)
LN10 (natürlicher Logarithmus von 10)
LOG2E (konstanter Logarithmus von 2)
LOG10E (konstanter Logarithmus von 10)
PI (Konstante PI)
SQRT1_2 (Konstante für Quadratwurzel aus 0,5)
SQRT2 (Konstante für Quadratwurzel aus 2)

Methoden:

abs() (positiver Wert)
acos() (Arcuscosinus)
asin() (Arcussinus)
atan() (Arcustangens)
ceil() (nächsthöhere ganze Zahl)
cos() (Cosinus)
exp() (Exponentialwert)
floor() (nächstniedrigere ganze Zahl)
log() (Anwendung des natürlichen Logarithmus)
max() (größere von zwei Zahlen)
min() (kleinere von zwei Zahlen)
pow() (Zahl hoch Exponent)
random() (0 oder 1 per Zufall)
round() (kaufmännische Rundung einer Zahl)
sin() (Sinus)
sqrt() (Quadratwurzel)
tan() (Tangens)

navigator

Über das Objekt `navigator` können Sie in einem JavaScript Informationen darüber ermitteln, welchen Web-Browser der Anwender verwendet sowie einige nähere Spezifikationen dazu. Das kann zum Beispiel interessant sein, um die Ausführung von JavaScript-Anweisungen davon abhängig zu machen, welchen Browser der Anwender benutzt. Eine zuverlässigere Möglichkeit, um Fehlermeldungen in Browsern zu vermeiden, die bestimmte verwendete JavaScript-Befehle nicht kennen, bietet die **Abfrage, ob ein Objekt existiert**.

Eigenschaften und Methoden von `navigator` können Sie direkt ansprechen. Beispiel: `navigator.appName`.

Eigenschaften:

appCodeName (Spitzname des Browsers)
appName (offizieller Name des Browsers)
appVersion (Browser-Version)
cookieEnabled (Cookies erlaubt)
language (Browser-Sprache)

Methoden:

javaEnabled() (Java-Verfügbarkeit überprüfen)

Unterobjekte:

mimeTypes

platform (Plattform, auf der der Browser läuft)
userAgent (HTTP-Identifikation des Browsers)

plugins

mimeTypes

Über das Objekt `mimeTpyes`, das in der JavaScript-Objekthierarchie unterhalb des **navigator**-Objekts liegt, können Sie ermitteln, welche Dateitypen der Browser des Anwenders kennt, und ob ein Plugin zum Anzeigen oder Abspielen eines Dateityps vorhanden ist (Indexnummer oder Name):

```
Beschreibung = navigator.mimeTypes["MIME-Typ"].Eigenschaft
```

Eigenschaften:

description (Beschreibung eines MIME-Typs)
enabledPlugin (Plugin vorhanden)
length (Anzahl MIME-Typen)
suffixes (Dateiendungen)
type (MIME-Typ)

plugins

Über das Objekt `plugins`, das in der JavaScript-Objekthierarchie unterhalb des **navigator**-Objekts liegt, können Sie ermitteln, welche Plugins im Sinne der Netscape-Plugin-Technik der Anwender installiert hat (Indexnummer oder Name):

```
Beschreibung = navigator.plugins["Name"].Eigenschaft
```

Eigenschaften:

description (Beschreibung eines Plugins)
filename (Dateiname eines Plugins)
length (Anzahl Plugins)
name (Produktname eines Plugins)

Number

Über das Objekt `Number` haben Sie Zugriff auf Eigenschaften numerischer Werte. So können Sie ermitteln, ob ein Wert eine gültige Zahl ist, oder welches die maximale positive oder negative Zahl ist, die in einer numerischen Variablen gespeichert werden kann.

Eigenschaften des Number-Objekts können Sie direkt ansprechen, indem Sie `Number` davor notieren.

Eigenschaften:

MAX_VALUE (größte speicherbare Zahl)
MIN_VALUE (kleinste speicherbare Zahl)
NaN (keine gültige Zahl)
NEGATIVE_INFINITY (Zahl zu klein)
POSITIVE_INFINITY (Zahl zu groß)

Methoden:

toExponential()
toFixed()
toPrecision()
toString()

RegExp

Reguläre Ausdrücke dienen dazu, Suchausdrücke zu formulieren, um in Zeichenketten nach Entsprechungen zu suchen und gefundene Stellen durch andere zu ersetzen.

Reguläre Ausdrücke können Sie in JavaScript direkt innerhalb entsprechender Methoden des **string**-Objekts anwenden. Das trifft auf die Methoden:

match(),
replace() und
search() zu.

Dort wird beschrieben, wie und wo Sie den regulären Ausdruck genau verwenden können, um Zeichenketten zu durchsuchen und Teile darin zu ersetzen.

Das RegExp-Objekt von JavaScript brauchen Sie dagegen nur, wenn Sie reguläre Ausdrücke zur Laufzeit des Scripts dynamisch erzeugen und ändern wollen. Dazu können Sie eine Instanz eines RegExp-Objekts definieren. Auf diese Instanz können Sie anschließend die Eigenschaften und Methoden des RegExp-Objekts anwenden, die hier beschrieben werden.

Syntax regulärer Ausdrücke

Die folgende Übersicht zeigt, aus welchen Bestandteilen Sie einen regulären Ausdruck zusammensetzen können.

Bestandteil	Beispiel	Beschreibung
	`/aus/`	findet "aus", und zwar in "aus", "Haus", "auserlesen" und "Banause".
`^`	`/^aus/`	findet "aus" am Anfang des zu durchsuchenden Wertes, also in "aus" und "auserlesen", sofern das die ersten Wörter im Wert sind.
`$`	`/aus$/`	findet "aus" am Ende des zu durchsuchenden Wertes, also in "aus" und "Haus", sofern das die letzten Wörter im Wert sind.
`*`	`/aus*/`	findet "au", "aus", "auss" und "aussssss", also das letzte Zeichen vor dem Stern 0 oder beliebig oft hintereinander wiederholt.
`+`	`/aus+/`	findet "auss" und "aussssss", also das letzte Zeichen vor dem Plus-Zeichen mindestens einmal oder beliebig oft hintereinander wiederholt.
`.`	`/.aus/`	findet "Haus" und "Maus", also ein beliebiges Zeichen an einer bestimmten Stelle.
`.+`	`/.+aus/`	findet "Haus" und "Kehraus", also eine beliebige Zeichenfolge an einer bestimmten Stelle. Zusammensetzung aus *beliebiges Zeichen* und *beliebig viele davon*.
`\b`	`/\baus\b/`	findet "aus" als einzelnes Wort. \b bedeutet eine Wortgrenze.
`\B`	`/\Baus\B/`	findet "aus" nur innerhalb von Wörtern, z.B. in "hausen" oder "Totalausfall". \B bedeutet *keine Wortgrenze*.
`\d`	`/\d.+\B/`	findet eine beliebige ganze Zahl. \d bedeutet eine Ziffer (0 bis 9)
`\D`	`/\D.+/`	findet "-fach" in "3-fach", also keine Ziffer.
`\f`	`/\f/`	findet ein Seitenvorschubzeichen.
`\n`	`/\n/`	findet ein Zeilenvorschub-Zeichen.
`\r`	`/\r/`	findet ein Wagenrücklaufzeichen.
`\t`	`/\t/`	findet ein Tabulator-Zeichen.
`\v`	`/\v/`	findet ein vertikales Tabulator-Zeichen.
`\s`	`/\s/`	findet jede Art von white space, also \f\n\t\v und Leerzeichen.
`\S`	`/\S.+/`	findet ein beliebiges einzelnes Zeichen, das kein white space ist, also kein \f\n\t\v und kein Leerzeichen.
`\w`	`/\w.+/`	findet alle alphanumerischen Zeichen und den Unterstrich (typische Bedingung etwa für programmiersprachengerechte selbstvergebene Namen).
`\W`	`/\W/`	findet ein Zeichen, das nicht alphanumerisch und auch kein Unterstrich ist (typisch zum Suchen nach illegalen Zeichen bei programmiersprachengerechten selbstvergebenen Namen).
`()`	`/(aus)/`	findet "aus" und merkt es sich intern. Bis zu 9 solcher Klammern (Merkplätze) sind in einem regulären Ausdruck erlaubt.
`/.../g`	`/aus/g`	findet "aus" so oft wie es in dem gesamten zu durchsuchenden Bereich vorkommt. Die Fundstellen werden intern in einem Array gespeichert.
`/.../i`	`/aus/i`	findet "aus", "Aus" und "AUS", also unabhängig von Groß-/Kleinschreibung.
`/.../gi`	`/aus/gi`	findet "aus", so oft wie es in dem gesamten zu durchsuchenden Bereich vorkommt (`g`) und unabhängig von Groß-/Kleinschreibung (`i`).

Eigenschaften:

$[1..9] (geklammerte Unterausdrücke)

Methoden:

exec() (Regulären Ausdruck anwenden)
test() (Regulären Ausdruck probehalber anwenden)

Screen

Mit dem Objekt `screen` können Sie Angaben zum Bildschirm des Anwenders ermitteln.

```
alert(screen.width + "x" + screen.height);
```

Diese Angaben werden oft dazu verwendet, das Layout einer Web-Seite oder die Fenstergröße an die zur Verfügung stehenden Bildschirmgröße anzupassen. Das `screen`-Objekt liefert jedoch im Hinblick darauf unzuverlässigen Werte, daher hat seine Bedeutung abgenommen. Das Auslesen der Größe desjenigen Bereiches im Browserfenster, in dem letztlich die Web-Seite letztendlich dargestellt wird, ist z.B. über **window.innerWidth** und **document.body.offsetWidth** möglich.

Eigenschaften:

availHeight (verfügbare Höhe)
availWidth (verfügbare Breite)
colorDepth (Farbtiefe)
height (Höhe)
pixelDepth (Farbauflösung)
width (Breite)

String

Die Eigenschaften und Methoden des Objektes `String` können Sie auf alle Zeichenketten anwenden. So können Sie in einer Zeichenkette z.B. alle darin enthaltenen Kleinbuchstaben in Großbuchstaben umwandeln, oder HTML-Formatierungen auf die Zeichenkette anwenden.

Eigenschaften:

length (Anzahl Zeichen)

Methoden:

anchor() (Verweisanker erzeugen)
big() (großen Text erzeugen)
blink() (blinkenden Text erzeugen)
bold() (fetten Text erzeugen)
charAt() (Zeichen an einer Position ermitteln)
charCodeAt() (Latin-1-Codewert an einer Position)
concat() (Zeichenketten zusammenfügen)
fixed() (Teletyper-Text erzeugen)
fontcolor() (Schriftfarbe erzeugen)
fontsize() (Schriftgröße erzeugen)
fromCharCode() (Latin-1-Zeichenkette erzeugen)
indexOf() (Position eines Zeichens ermitteln)
italics() (kursiven Text erzeugen)
lastIndexOf() (letzte Position eines Zeichens ermitteln)
link() (Verweis erzeugen)
match() (Regulären Ausdruck anwenden)
replace() (Regulären Ausdruck anwenden und ersetzen)
search() (Suchen mit Regulärem Ausdruck)
slice() (Teil aus Zeichenkette extrahieren)
small() (kleinen Text erzeugen)
split() (Zeichenkette aufsplitten)
strike() (durchgestrichenen Text erzeugen)
sub() (tiefgestellten Text erzeugen)
substr() (Teilzeichenkette ab Position ermitteln)
substring() (Teilzeichenkette ermitteln)
sup() (hochgestellten Text erzeugen)
toLowerCase() (alles klein schreiben)
toUpperCase() (alles groß schreiben)

Objektunabhängige Funktionen

Objektunabhängige Funktionen sind im Gegensatz zu selbst definierten **Funktionen** in JavaScript bereits vordefiniert. Sie können diese Funktionen also jederzeit aufrufen.

Es handelt sich um bestimmte, oft mächtige, JavaScript-Befehle, die in keines der Objekte passen und deshalb nicht der objektorientierten Richtung von JavaScript folgen.

decodeURI()	kodierten URI dekodieren: Dekodiert einen **URI**, der mit **encodeURI()** kodiert wurde.
decodeURIComponent()	kodierten URI dekodieren – II: Wie **decodeURI()**. Sollte aber nur auf Adressen oder Adressteile angewendet werden, die mit mit **encodeURIComponent()** kodiert wurden.

encodeURI()	URI kodieren:
	Verschlüsselt einen **URI** so, dass alle Sonderzeichen in ASCII-Zeichensequenzen umgewandelt werden. Kodiert werden also beispielsweise deutsche Umlaute und Sonderzeichen, auch Leerzeichen, eckige und geschweifte Klammern usw. außer den folgenden Zeichen:
	`0` bis `9`
	`A` bis `Z`
	`a` bis `z`
	`- _ . ! ~ * ' ( )`
	`, / ? : @ & = + $`

encodeURIComponent()	URI kodieren – II: Wie **encodeURI()**, es werden aber auch folgende Zeichen kodiert: `, / ? : @ & = + $`
eval()	Ausdruck interpretieren: Interpretiert ein zu übergebendes Argument und gibt das Ergebnis zurück. Wenn das übergebene Argument als Rechenoperation interpretierbar ist, wird die Operation berechnet und das Ergebnis zurückgegeben. Dabei sind auch komplexe Rechenausdrücke mit Klammerung möglich. Diese Funktionalität ist sehr praktisch, um als Zeichenketten notierte Rechenausdrücke mit einem einzigen Befehl errechnen zu lassen. Wenn das übergebene Argument als Objekt oder Objekteigenschaft interpretiert werden kann, wird das Objekt bzw. die Objekteigenschaft zurückgegeben.
escape()	ASCII-Zeichen in Zahlen umwandeln: Wandelt Steuersequenzen (Steuerzeichen mit den ASCII-Werten 0 bis 31) und Sonderzeichen wie z.B. deutsche Umlaute in ihre ASCII-Zahlenwerte in hexadezimaler Form um. Setzt vor jeden Wert das Trennzeichen "%" und gibt die so erzeugte Zeichenkette zurück. Soll in Zukunft durch **encodeURI()** ersetzt werden!
isFinite()	auf numerischen Wertebereich prüfen: Ermittelt, ob ein Wert sich innerhalb des Zahlenbereichs befindet, den JavaScript verarbeiten kann, also aus Sicht von JavaScript eine gültige Zahl darstellt.
isNaN()	auf numerischen Wert prüfen: Ermittelt, ob ein übergebener Wert eine ungültige Zahl ist (NaN = Not a Number). Ist `true`, wenn der Wert **keine** Zahl ist, und `false`, wenn es eine Zahl ist.
parseFloat()	in Kommazahl umwandeln: Wandelt eine Zeichenkette in eine Kommazahl und gibt diese als numerischen Wert zurück.
parseInt()	in Ganzzahl umwandeln: Wandelt eine Zeichenkette in eine Ganzzahl um und gibt diese als Ergebnis zurück.
Number()	auf numerischen Wert prüfen: Konvertiert den Inhalt eines Objekts in eine Zahl und gibt die Zahl zurück.
String()	in Zeichenkette umwandeln: Konvertiert den Inhalt eines Objekts in eine Zeichenkette und gibt die Zeichenkette zurück.
unescape()	Zahlen in ASCII-Zeichen umwandeln: Wandelt alle Zeichen der zu übergebenden Zeichenkette in normale ASCII-Zeichen um und gibt die so erzeugte Zeichenkette zurück. Soll in Zukunft durch **decodeURI()** ersetzt werden!