JavaScript

GRUNDSÄTZLICHES

- JavaScript ist eine von Netscape in den 90-er Jahren entwickelte Programmiersprache zum Optimieren von HTML Quellcode, sie kann mit jedem beliebigen Text- oder HTML-Editor geschrieben werden und wird vom Browser zur Laufzeit interpretiert.
- Lösungen, bei denen die Daten mehrerer beteiligten Anwender gefragt sind, sind aufgrund der Interpretation im Browser nicht möglich (Gästebücher, Seitenzähler, Web-Foren, …)
- Weitere Infos: Schulinternes Intranet: Tutorial "JavaScript" von Ulrich Hirschmann, http://de.selfhtml.org, http://msdn.microsoft.com (> WebDevelopment/Scripting/JScript)

Konventionen (Befehle, Bezeichner, …)

- JavaScript ist Case-Sensitive (unterscheidet zw. Groß- u. Kleinbuchstaben: z.B. meinAuto)
- weiters gelten im Schnitt die Regeln, die auch für HTML gelten. ($ und _ sind jedoch erlaubt)
- JavaScript hat reservierte Schlüsselwörter, die in den gängigen Referenzen gesammelt sind.
- Jeder Js-Befehl wird mit einem Strichpunkt abgeschlossen!

DAS JAVASCRIPT-GRUNDGERÜST

JavaScript in eine externe Datei auslagern

- Die externe Datei wird unter name**.js** gespeichert und enthält keine weiteren Einleitungen, …
- Eingebunden wird die externe js-Datei im HEAD der jeweiligen HTML-Datei. Der gesamte Inhalt der externen js-Datei steht jetzt in der HTML-Datei als JavaScript-Code zur Verfügung.

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

JavaScript-Bereiche in HTML (HEAD od. BODY)

```
<script type="text/javascript">
<!--
   window.alert("Hallo Welt!");
//-->
</script>
```

HTML-Kommentare verhindern, dass ältere Browser den Code als Text interpretieren. JavaScript-Kommentar u. HTML-Kommentar.

JavaScript-Bereiche in HTML-TAGS

- JavaScript kann auch innerhalb herkömmlicher HTML-Tags vorkommen. Das ist dann kein komplexer Programmcode, sondern in der Regel nur der Aufruf bestimmter Methoden, Funktionen, Objekte, Eigenschaften. Für einen solchen Aufruf gibt es so genannte ***Event-Handler***[1] (siehe unten). Das sind Attribute in HTML-Tags, über die sich JavaScripts aktivieren lassen.
- Weiters kann bei Links auch ***JavaScript:***[2] (Fußnote auf Seite 2) notiert werden:

```
<a href="#" onClick="document.bgColor='#E0FFE0'">Hintergrund wechseln</a>
```

---

[1] EVENT-HANDLER

- Event-Handler (Ereignisbehandler) sind ein wichtiges Bindeglied zwischen HTML und JavaScript. Diese HTML-Attribute beginnen immer mit *on* (`onClick=""`,…). In den Anführungszeichen steht dann eine JavaScript-Anweisung. Wenn mehrere Anweisungen ausgeführt werden sollen, dann definiert man am besten in einem JavaScript-Bereich eine `function` (siehe weiter unten) und ruft diese auf, also z.B. `onClick="Umrechnen()"`.
- Jeder Event-Handler steht für ein bestimmtes Anwenderereignis, `onClick` etwa für das Ereignis "Anwender hat mit der Maus geklickt". Der Anzeigebereich des HTML-Elements, in dem der Event-Handler steht ist, ist das auslösende Element. Hier einige Beispiele

`onAbort` (bei Abbruch)	`onKeydown` (bei gedrückter Taste)	`onMouseover` (beim Überfahren des Elements)
`onBlur` (beim Verlassen)	`onKeypress` (bei gedrückt gehaltener Taste)	`onMouseUp` (bei losgelassener Maustaste)
`onChange` (bei erfolgter Änderung)	`onKeyup` (bei losgelassener Taste)	`onReset` (beim Zurücksetzen des Formulars)
`onClick` (beim Anklicken)	`onLoad` (beim Laden einer Datei)`onMousedown` (bei	`onSelect` (beim Selektieren von Text)
`onDblClick` (bei doppeltem Anklicken)	gedrückter Maustaste)	`onSubmit` (beim Absenden des Formulars)
`onError` (im Fehlerfall)	`onMousemove` (bei weiterbewegter Maus)	`onUnload` (beim Verlassen der Datei)
`onFocus` (beim Aktivieren)	`onMouseout` (beim Verlassen des Elements)	*Details: siehe http://de.selfhtml.org*

[2] javascript: (bei Verweisen)

- Bei `javascript:` handelt es sich um eine Syntax, mit der JavaScript-Code als Verweisziel notiert werden kann `<a href="javascript:alert(document.lastModified)">Letzter Update</a>`

```
<a href="JavaScript:window.close()">Fenster schlie&szlig;en</a>
```

ALLGEMEINE REGELN FÜR JAVASCRIPT

Kommentare in JavaScript

```
//  Zeilenweises Kommentieren und Kommentarblöcke: /*  ...  */
```

Anweisungsblöcke notieren

- Eine od. mehrere Anweisungen stehen innerhalb übergeordneter Anweisungen/Funktionen; an den **{}** erkennt das System, welche Befehle zum Anweisungsblock gehören

```
if (Zahl > 1000)
{   document.write("Sie brauchen keine Versandkosten zu bezahlen.");
    versandkosten=0;}
```

VARIABLEN

Variablen definieren

- Variablen sind Speicherbereiche, denen verschiedene Werte (Inhalte) zugeordnet werden können, auf die später wieder zurückgegriffen werden kann.
- Variablen werden mittels "Deklaration" eingeführt, es wird ein Speicher unter dem angegebenen Variablennamen reserviert und sie stehen dann zur Verfügung.

```
var x;                                          //Deklaration
var Hinweis = "Eine Variable enthält Werte.";   //Deklaration u. Zuweisung
var Rabatt1,Rabatt2,Rabatt3;                    //Deklaration mehrerer Var.
x=4;                                            //reine Zuweisung
window.alert(Hinweis,x);                        //Ausgabe
```

- Es gibt lokale Variablen, die z.B. nur in einer Funktion (siehe unten) verwendet werden und globale, die im ganzen Programm zur Verfügung stehen. Fixiert wird diese Unterscheidung durch die Stelle, an der die Variable definiert wird, zB. innerhalb einer Funktion.

Typen von Variablen

- Zahlen: Zuweisung ohne Anführungszeichen und "." als Komma
  z.B. x=3.4; b=1E9; (1.000.000.000) c=2E-3; (0,003)
- Strings: Texte, werden in Anführungszeichen geschrieben (z.B. y="Guten Morgen!")
- Boolsche Werte: enthalten ja/nein Anweisungen (true/false) (z.B. n=false)
  null: Einer Variable ist noch kein Wert zugewiesen, bei einer Ausgabe erscheint "undefined"

Variablen in JavaScript sind nicht so streng "getypt" wie in vielen anderen Programmiersprachen. Einfache Variablentypen, wie Zahlen, Zeichenketten oder Wahrheitswerte, werden lediglich nach numerischen und nicht-numerischen Variablen eingeteilt. Kommazahlen und Ganzzahlen benötigen keine unterschiedlichen Typen. Der Inhalt numerischer Variablen kann ohne Konvertierung in Zeichenketten auf den Bildschirm ausgegeben werden. Allerdings kann mit Zeichenketten (z.B. Werten aus Formularfeldern), nicht immer gerechnet werden, sondern sie müssen vorher explizit in Zahlen umgewandelt werden (siehe unten)!

Werte von Variablen ändern

- Variablenwerte können jederzeit geändert werden (vgl. Schleifen, …)

```
var Einkauf=window.prompt("Bitte Einkaufssumme eingeben:","Hier eingeben");
var Rabatt=10;
if (Einkauf > 2000) Rabatt=20;
document.write("Gratulation! <br />Ihr Rabatt: <b>" +Rabatt+ " %!</b><br />");
```

Überprüfung von Variablentypen: typeof

- Vorbemerkung: Methoden fertiger JavaScript-Objekte (siehe weiter unten) sind an feste Variablentypen (boolean, string, number, function, object, undefined) gebunden.
- Mit typeof können Variablentypen überprüft werden: `var Zahl = 2005; window.alert(typeof Zahl);`

Umwandeln

- parseFloat(), parseInt() wandelt Texte in Zahlen um: z.B. x = parseFloat(x);
- string() wandelt in einen Text um
- in der Klammer der Methoden steht jeweils das, was umgewandelt werden soll.

OPERATOREN

Arithmetische Operatoren

+|-	Addition, Subtraktion: a=12-10 oder b=-20+30
*|/|%	Multiplikation, Division, Modulo (Restberechnung): a=10*5 oder b=7%4 (also 3)
++	Inkrement, erhöht den Wert um 1: b++ ist das selbe wie b=b+1
--	Dekrement, erniedrigt den Wert um 1: b-- ist das selbe wie b=b-1

Zuweisungsoperatoren

=	Weist der Variablen auf der linken Seite den Wert der rechten Seite zu: z.B. a=a+4
+=	Addiert den linken und rechten Operanden und weist das Ergebnis dem linken Operanden zu Gleiches gilt für -= | *= | /= | %= (/= ist das Ergebnis der Division und %= ergibt den Rest einer Division) Schreibweise: a=a+b ist dasselbe wie a+=b

Vergleich:

==	Gibt "true" zurück, wenn beide Operanden gleich sind: if (a=="Frau")
!=	Gibt "true" zurück, wenn beide Operanden nicht gleich sind
<|>	Gibt "true" zurück, wenn der linke Operand kleiner (größer) ist als der rechte
<=|>=	Gibt "true" zurück, wenn der linke Operand kleiner gleich (größer gleich) ist als der rechte

Stringoperatoren:

+	Hängt zwei Strings zu einem zusammen: Name=Vorname+" "+Zuname
+=	Hängt den linken und rechten String zusammen und weist das Ergebnis der linken Variablen zu

Logische Operatoren:

&&	Logisches UND: Es wird "true" zurückgegeben, wenn beide Operanden "true" sind
||	Logisches ODER: Es wird "true" zurückgegeben, wenn einer der beiden Operanden oder beide Operanden "true" sind
!	Logisches NICHT: Es wird "true" zurückgegeben, wenn der Operand "false" ist und umgekehrt

EXKURS 1: HTML - FORMULARE ENTWERFEN

- Mit <form ...> definiert man ein Formular (form = Formular). Alles, was zwischen diesem einleitenden Tag und dem abschließenden Tag </form> steht, gehört zum Formular: Eingabefelder, Auswahllisten, Buttons, ... Zum Plazieren von Elementen brauche ich jedoch auch Textabsätze, erzwungene Zeilenumbrüche, Bilder, Tabellen, div-Container, ...
- Im einleitenden <form>-Tag wird mit action= angegeben, was mit den ausgefüllten Formulardaten passieren soll, wenn der Anwender das Formular abschickt.

Formular definieren	<form action="name@domain.xy" method="post" enctype="text/plain"> <!-- Formularelemente --> </form>
	enctype="text/plain" wird nicht von allen Browsern interpretiert. mailto-Formulare (metod="post") funktionieren bei Browsern ohne eingerichtetem Mail-Client nicht!
Einzeilige	<input size=”x” maxlength=”x” name="Elementname" value="Text" />

Eingabefelder und Textvorbelegung	Anzeigebreite mit size= (Anzahl Zeichen) bestimmen, auf Wunsch maximale Eingabelänge mit maxlength= (Anzahl Zeichen). Name für JavaScript-Verarbeitung interessant. Textvorbelegung mit value= bestimmen.
Passwort Eingabefelder	`<input type=”password” size=”x” maxlength=”x” name="Elementname" />` Blindeingabe mit type=password bestimmen.

Mehrzeilige Eingabefelder	`<textarea cols="x" rows="x" name="Elementname">`
	`*Optionale Textvorbelegung*`
	`</textarea>`
	Breite des Eingabebereichs mit cols= (Anzahl Zeichen) angeben, Höhe mit rows= (Anzahl Zeilen). Name für JavaScript-Verarbeitung interessant.
	Wenn eine Textvorbelegung gewünscht ist, zwischen Einleitungs- und End-Tag notieren.
	Automatischen Zeilenumbruch bei Eingabe mit wrap=“virtual“ erlauben.
	Sollte das Feld nur zum Lesen gedacht sein u. keine angabe möglich: readonly=“readonly“

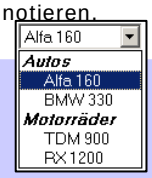
Auswahllisten	`<select name="Elementname">`
	`  <option> Eintrag </option>`
	`  <option> anderer Eintrag </option>`
	`</select>`
	Einfache Auswahlliste > Drop-Down-Menue
	Mehrfachauswahl: <selct multiple=“multiple“ ...> Auswahlfeld statt Drop-Down
	Vorauswahl treffen: Attribut selected=“selected“ in gewünschtem <option>-Tag notieren.
	Absendewerte bei Listenfeldern eintragen: <option value="Mitgabewert">

Menüstruktur	`<select>`
	`  <optgroup label="Gruppenüberschrift">`
	`  <option> Eintrag`
	`  </optgroup>`
	`</select>`
	So viele <optgroup>-Untermenüs notieren wie gewünscht. Verschachtelung nicht möglich.

Radiobuttons	`<input type=”radio” name="Name" value="Wert" /> Text` Gemeinsame Namen für alle zusammengehörigen Radiobuttons vergeben, aber versch. Values. Einträge vorselektieren: checked=“checked“ Elemente ausgrauen (gilt für alle input-Types): <input ... disabled>
Checkboxen	`<input type=”checkbox” name="Name" value="Wert" /> Text` Gemeinsame Namen für alle zusammengehörigen Checkboxen vergeben, aber versch. Values. Einträge vorselektieren: checked=”checked”
Klickbuttons	`<input type=”button” name="Name" value="Beschriftung" onClick="..." />` Sinnvoll zum Ausführen von JavaScript-Code bei onClick=.

Img-Buttons	`<button type=”button” name="Name" value="Beschriftung" onClick="...">`
	`  <img src="URL" />`
	`</button>`
	Sinnvoll z.B. um Grafiken als Button-Beschriftung zu benutzen.

Dateibuttons zum Hochladen von Dateien	`<input type=”file” name="Name" maxlength=”x” accept="MimeType" />` Wichtig: bei <form>-Tag die Angabe enctype="multipart/form-data" notieren. Dateigröße mit maxlength= (Bytes) begrenzen. Dateityp mit accept= eingrenzen, z.B. accept="text/*"
Button zum Absenden/ Abbrechen	`<input type=... value="Beschriftung" />` type=“submit“ schickt das Formular ab, type=“reset“ löscht alle Eingaben. Grafischer Absendebutton: <input type=image src="URL" />



EXKURS 2: EINFACHE EIN- UND AUSGABEN

Eingabefenster

```
var Name=window.prompt("Bitte Namen eingeben:","Ihr Name");
```

Der erste String in der Klammer von window.prompt() ist die Eingabeaufforderung im Eingabefenster, der zweite String kann einen Default-Eintrag vorgeben, der vorerst im Eingabefeld steht, bis eine andere od. keine Eingabe folgt => oft muss dann mit einem parse…- Befehl der Typ der Variable gändert werden (zB von Text auf Zahl)

Ausgabe im Dokument-Fenster

```
document.write(Name + ", ich grüße Sie!<br /><hr />");
```

Mit "+" können Variableninhalte und Strings nacheinander ausgegeben werden. HTML-Tags können innerhalb von Strings verwendet werden und werden selbstverständlich vom Browser interpretiert.

Ausgabe in einem Fenster: alert

```
window.alert("Sie haben das falsche Passwort eingegeben!\nNeuer Versuch!");
```

Mit \n kann ein Zeilenwechsel gesetzt werden.

Ein- und Ausgabe mit HTML-Formularen

```
<form name="meinform">
   Zahl 1 <input type="text" name="zahl1" value="0" size="4" maxlength="4" />
   Zahl 2 <input type="text" name="zahl2" value="0" size="4" maxlength="4" />
   <input type="button" name="add" value="+"
      onClick="document.meinform.ergebnis.value=
      parseFloat(document.meinform.zahl1.value) +
      parseFloat(document.meinform.zahl2.value)" />
   <input type="button" name="add" value="-"
      onClick="document.meinform.ergebnis.value=
      parseFloat(document.meinform.zahl1.value) -
      parseFloat(document.meinform.zahl2.value)" />
   <input type="text" name="ergebnis" value="0" size="20" maxlength="20" />
</form>
```

Mit Hilfe von Formularen und Buttons lassen sich Eingaben eleganter realisieren.
Dabei kann durch Angabe der kompletten Objekthierarchie und der korrekten selbst vergebenen Namen jedes Formularfeld angesprochen werden.
z.B. `document.meinform.ergebnis.value`
Eine weitere Möglichkeit ist der Zugriff via `document.getElementByID("")`, dann muss natürlich jedes Formularelement eine eindeutige ID haben.

FUNKTIONEN IN JAVASCRIPT

Einfache Funktionen

- Mit Hilfe von Funktionen (vgl. Methoden/Prozeduren) können Programmteile gekapselt und mehrfach aufgerufen werden. Funktionen haben einen Funktionsnamen.

```
function Gratulation() {
  gs = gs + 100;  //Gewinnsumme erhöhen
  document.write("Gratulation! Sie stehen jetzt schon bei " + gs + "€");}
```

- Funktionsaufruf zB. an einer anderen Stelle im Programm: `Gratulation();`

Funktionen mit Parameterübergabe

- Oft ist es gut, wenn Funktionen einen Wert von außen für ihre Aufgaben zur Verfügung haben

```
function Flaechenberechnung(laenge, breite) {
   document.write(Fläche = " + laenge*breite); }
```

- Funktionsaufruf zB an einer anderen Stelle im Programm: **`Flaechenberechnung(10,20);`**

Funktionen mit Parameterübergabe und Rückgabewert

- Oft ist es gut, wenn Funktionen einen Wert von außen für ihre Aufgaben zur Verfügung haben

```
function Flaechenberechnung(laenge, breite) {
   return laenge*breite; }
```

- Funktionsaufruf zB an einer anderen Stelle im Programm:
  `document.write("Fläche = " + Flaechenberechnung(10,20));`
- Oder:

```
function fett(txt){
  return "<font color='#FF9E00' size='4' face='verdana'><b>"+txt+"</font></b>"}
document.write(fett("Dieser Text sollte formatiert sein."));
```

- Allgemein:

```
function Funktionsname(Parameter1, Parameter2, ...) {
   ...    return Rückgabewert;  }
```

BEDINGTE ANWEISUNGEN (IF-ELSE/SWITCH)

Dieser Passwortschutz ist natürlich nicht ernst zu nehmen, ein echter Schutz muss serverseitig realisiert werden

Wenn-Dann-Bedingung: if

- Anweisungen können von Bedingungen abhängig gemacht werden:

```
function Geheim() {
   var Passwort = "Traumtaenzer";
   var Eingabe = window.prompt("Passwort eingeben","***");
   if(Eingabe != Passwort) {
      alert("Falsches Passwort!");
      Geheim();   }
 else {
   document.location.href="geheim.htm";  }  }
```

- Mit `if(...)` wird die WENN-Bedingung abgefragt: Wird die Bedingung erfüllt, setzt das Programm mit den Befehlen im folgenden Anweisungsblock `{...}` fort, ansonsten wird gleich nach dem Anweisungsblock fortgesetzt. Hier *kann* mit `else` eine SONST-Anweisung angegeben werden.
- Allgemein: `if (Bedingung) { ... } else { … }`
- If-Anweisungen können auch mit "UND" oder mit "ODER" kombiniert werden

```
if (a==b && a==c) { document.write("Alle Werte sind gleich!"); }  // und
if (a==b || a==c) { document.write("Zwei Werte sind!"); }         // oder
```

(Einfache Entweder-Oder-Anweisung)

```
function Antwort() {
   var Ergebnis = (document.Formular.Eingabe.value == "Traumtaenzer")
   ? "RICHTIG!" : "FALSCH!";
   document.Formular.Eingabe.value = "Die Antwort ist " + Ergebnis;    }
```

- Die Funktion überprüft, ob Ergebnis den richtigen Wert zugewiesen hat
- Syntax für oben: (Bedingung) ?=wahr :=falsch

Fallunterscheidung: switch

- Oft gilt es mehrere Fälle zu unterscheiden, was eine lange Kette an ifs bewirken würde

```
var Eingabe = window.prompt("Geben Sie eine Zahl zwischen 1 und 4 ein:","");
switch(Eingabe) {
 case "1":  alert("Sie sind sehr bescheiden");  break;
 case "2":  alert("Sie sind ein aufrichtiger Zweibeiner");  break;
 case "3":  alert("Sie haben ein Dreirad gewonnen");  break;
 case "4":  alert("Werden Sie bescheidener"); break;
 default:   alert("Sie bleiben leider dumm"); break; }
```

- Mit `case "Wert":` werden Bedingungen abgefragt. `default` deckt alle anderen Fälle ab.
- Allgemein:
  `switch (Variable) { case “Wert“ : Aktion; break; Fall …; default : Aktion ; break; }`

SCHLEIFEN (WHILE/FOR/DO-WHILE)

Schleifen mit while (kopfgesteuerte Schleife)

- Mehrere Programmanweisungen werden so lange wiederholt, bis die Bedingung, die in der Schleife formuliert ist, erfüllt wird:

```
var i=1;
while (i<=20) {document.write(i+"Sch. sind "+i/13.7603+" &euro;<br />"); i++;}
```

- Allgemein: `while (Abbruchbedingung) {...}`
- Achte darauf, dass die Bedingung wirklich auch eintritt, sonst wird das Programm nie enden, hier sorgt dafür der Eintrag `i++;`

Schleifen mit for (Zählschleife)

- Schleife mit einer bestimmten Anzahl an Durchläufen
```
for (var i=6;i<=50;i=i+2) {
   document.write('<span style="font-size:'+i+'px;">Schriftgröße:
   '+i+'px</span> <br />'); }
```
- Allgemein: for (Variable u. Startwert; Abbruchbedingung; Schrittweite) {...}
- Schleifen können auch verschachtelt werden, sodass innerhalb einer Zählschleife eine zweite gestartet und abgearbeitet wird und anschließend der nächste Durchgang der äußeren Schleife usw.

Schleifen mit do-while (fußgesteuerte Schleife)

- Hier handelt es sich um eine Variante der while-Schleife. Bei do-while wird der Code mindestens einmal durchlaufen, bei while nicht unbedingt.
```
var i=0;
do {  i++;
   var antwort = window.prompt(i+" .Versuch: Tag für Zeilenwechsel",0) }
while (antwort != "<br />");
document.write ("RICHTIG! Sie hatten <b>" +i+ "</b> Versuche");
```
- Allgemein: do { … } while (Abbruchbedingung)

Kontrolle innerhalb von Schleifen – break|continue

- Mit break wird eine Schleife sofort beendet
```
if (eingabe=="e") break;
```
- Mit continue wird ein bestimmter Durchlauf (if-Abfrage) einer Schleife nicht durchgeführt, die Schleife aber fortgesetzt.
```
var zahl=window.prompt("Ausgangszahl",1);
for (var i=-10;i<=10;i++)  {
   if (i==0) { document.write("Division durch 0 unmöglich!<br />"); continue; }
   document.write(zahl+" : " +i+ " = " +zahl/i+" <br />");  }
```

WEITERE GRUNDLAGEN

Das Objekt Array

- Das Objekt Array ist als "Container" für Ketten gleichartiger Variablen gedacht. In der Programmierersprache spricht man auch von einem "Vektor". Man könnte also z.B. für das Anlegen aller SchülerInnen einer Klasse mit Arrays arbeiten.
- (1) Objektname = **new Array()**: Anzahl der gleichartigen Variablen ungewiss
  (2) Objektname = new Array (100): 100 Variablenwerte sollen gespeichert werden
  (3) Objektname = new Array ("Fritz", "Franz", "Ferdl"): Plätze werden gleich belegt
- Ausgabe zu (3): alert Objektname[0] ➔ *Fritz* und alert Objektname[2] ➔ *Ferdl*
- Mehrdimensionale Arrays: z.B. für ein Quiz. Dimensionen Frage, Antwort
```
var a = new Array(4);
for (var i = 0; i < a.length; i++)          // erste Dimension
   a[i] = new Array(10);                    // des Arrays anlegen
a[0][1]="Hallo";  a[0][2]="Fritz"; a[1][0]="Grüß Gott"
alert(a[0][2]);  ➔ Fritz
```

String-Funktionen

- Zeichenketten können mit Hilfe von String-Funktionen weiteres be-/verarbeitet werden.
- zB Können Zeichenketten in Groß-/Kleinbuchstaben umgewandelt (toLowerCase(), toUpperCase()), Zeichenketten zusammengefügt (concat()), Positionen bestimmter Zeichen ermittelt (indexOf()), verschiedene Texte erzeugt werden (Anker, Schriftart, -farbe, -größe …), Teilzeichenketten ermittelt (slice()), Zeichen gesucht (search()) werden, …
- ACHTUNG: Sonderzeichen bereiten Probleme bei der Arbeit mit Strings; z.B. Anführungszeichen, Schrägstriche, usw.
  Beim Suchen, … muss man also teilweise mit "regulären Ausdrücken" (siehe Objekt RegExp in der Referenz http://de.selfhtml.org) arbeiten.

Objektunabhängige Funktionen

- JavaScript stellt viele fertige Funktionen zur Verfügung: z.B. Umwandlung von Zeichenketten in Komma- oder Ganzzahlen (`parseFloat(), parseInt()`), Umwandung in Strings (`string()`), Codieren von URIs (`encordeURI()`), Interpretieren von Ausdrücken (`eval()`), auf (nicht)nummerische Werte prüfen (`isFinite(), isNaN()`)
- Weitere Funktionen findet man in JavaScript-Referenzen

OBJEKTE, EIGENSCHAFTEN, METHODEN

Vordefinierte JavaScript-Objekte

- Objekte sind fest umgrenzte Datenelemente mit Eigenschaften und oft auch mit objektgebundenen Funktionen (Methoden – siehe unten). Über vordefinierte Objekte kann man z.B. auf Elemente von HTML-Formularen zugreifen.
- Analogie: Ein fiktives Objekt Auto hat Eigenschaften (zB Farbe, Größe, Marke, Sitzplätze, …) und Methoden (kann also etwas tun: fahren, beschleunigen, bremsen, …). Ein bestimmtes Auto (aus der Gesamtheit des Objekts Auto – zweifelsohne gibt es ja viele Autos) ist dann eine Instanz des abstrakten Objektes Auto, die zuerst erstellt werden muss.
- Ein Objekt kann weiters eine Teilmenge eines übergeordneten Objekts sein. Die JavaScript-Objekte sind deshalb in einer Hierarchie geordnet:
  `window`: hierarchiehöchstes Objekt – hat Eigenschaften wie: Name, Größe, …
  `document`: ist in der Regel der Fensterinhalt einer Datei u. kann z.B. Formulare, Links, Grafikreferenzen, … enthalten. Für solche untergeordneten Elemente gibt es wieder eigene JavaScript-Objekte, zum Beispiel das Objekt forms für Formulare.
  `elements`: Ein Formular besteht wiederum aus verschiedenen Elementen (z.B. Formularfelder), die angesprochen werden können
- Neben den hierarchisch geordneten JavaScript-Objekten gibt es auch solche, die nicht direkt in die Hierarchie passen. Das sind zum Beispiel Objekte für Datums- und Zeitrechnung, für mathematische Aufgaben oder für Zeichenkettenverarbeitung.
- Übersicht über JavaScript-Objekte [Quelle: http://de.selfhtml.org/javascript/objekte/index.htm]:

`window`	Anzeigefenster
`frames`	Frame-Fenster
`document`	Dokument im Anzeigefenster
`HTML-Elementobj.`	Alle HTML-Elemente des Dokuments
`node`	Alle Knoten des Elementbaums
`all`	Alle HTML-Elemente des Dokuments – Microsoft
`style`	CSS-Attribute von HTML-Elementen
`anchors`	Verweisanker im Dokument
`applets`	Java-Applets im Dokument
`forms`	Formulare im Dokument
`elements`	Formularelemente eines Formulars
`options`	Optionen einer Auswahlliste eines Formulars
`images`	Grafikreferenzen im Dokument
`embeds`	Multimedia-Referenzen im Dokument
`layers`	Layer im Dokument - Netscape

`links`	Verweise im Dokument
`event`	Anwendereregnisse
`history`	besuchte Seiten
`location`	URIs
`Array`	Ketten von Variablen
`Boolean`	Ja-Nein-Variablen
`Date`	Datum und Uhrzeit
`Function`	JavaScript-Funktionen
`Math`	Berechnungen
`navigator`	Browser-Informationen
`mimeTypes`	MimeType-Informationen
`plugins`	installierte Plugins
`Number`	numerische Werte
`RegExp`	reguläre Ausdrücke
`Screen`	Bildschirm-Informationen
`string`	Zeichenketten

Eigenschaften von Objekten

- Objekte haben Eigenschaften, die jederzeit ausgelesen werden können (`Math.PI`: `PI` als Eigenschaft des Objektes `Math`).
  `document.write("Sie verwenden den " + navigator.appName);`
- Bei manchen Objekten können Eigenschaften auch geändert werden (Browser-Fenster)

Methoden von Objekten

- Methoden sind Funktionen, die Aktionen ausführen, aber an ein bestimmtes Objekt gebunden sind. Vordefinierte JavaScript-Objekte haben oft Methoden.
- Im Objekt `history` sind zB die besuchten URLs eines Browserfensters gespeichert
  `<a href="javascript:history.back();">Geh hin wo Du herkommst</a>`
- Ob ein Browser ein best. Objekt interpretiert, kann abgefragt werden:
  `if(! document.images)` // ! bedeutet nicht
- `document.getElementById()` greift auf ein HTML-Element zu, dabei muss dann z.B. noch das Attribut angegeben werden. Einige Beispiele: `document.getElementById("bild").src`, oder

.value (bei Formularen) oder .innerHTML für z.B. den Text, der zw. zwei Elementen steht: Ökoenergie im Vormarsch - hier kann so auf den Text „Ökoenergie im Vormarsch" zugegriffen werden.

- An dieser Stelle möchte ich empfehlen, sich einmal einen Überblick über die in JavaScript vorhanden Objekte, Methoden und Eigenschaften mit Hilfe der Referenzen zu verschaffen!

Vordefinierte Objekte verwenden, ein Beispiel

```
var Jetzt = new Date();
var Tag = Jetzt.getDate();
var Monat = Jetzt.getMonth() + 1;
var Jahr = Jetzt.getYear();
var Stunden = Jetzt.getHours();
var Minuten = Jetzt.getMinutes();
var NachVoll  = ((Minuten < 10) ? ":0" : ":");
if (Jahr<2000) Jahr=Jahr+1900;
document.write("<h2>Guten Tag!<\/h2><b>Heute ist der "+ Tag + "." + Monat + "."
   + Jahr + ". Es ist jetzt "+ Stunden + NachVoll + Minuten + " Uhr<\/b>");
```

- Der Variable Jetzt wird mit new eine neue Instanz des Objektes Date zugewiesen.
- Um einzelne Daten der Objektinstanz anzusprechen, stehen Methoden (siehe unten) zur Verfügung, deren Rückgabewert hier ein Datum/Uhrzeit-Bestandteil ist.
- Die Anweisung NachVoll dient einer sauberen Formatierung der Uhrzeit.
- Mit document.write wird eine Ausgabe getätigt

Eigene Objekte definieren

- Um ein eigenes Objekt anzulegen, sind zwei Schritte nötig. Zuerst muss ich das Objekt selbst und seine Eigenschaften "deklarieren". Im zweiten Schritt kann eine Instanz dieses Objektes definiert werden. Mit der Objekt-Instanz können dann Programmprozeduren durchgeführt werden. Das folgende Beispiel sollte dies verdeutlichen:

```
function Farbe(R, G, B) {        // Funktionen => siehe weiter unten
  this.R = R;                    // this ist ein Bezug auf das eigene akt. Objekt
  this.G = G;
  this.B = B;                    // R,G,B erhalten ihren Wert unten (new Farbe …)
  this.hex="#";                  // hex bleibt für jede Instanz des Objektes gleich
}
function HintergrundWechseln(){
  var Hintergrund = new Farbe("E0","FF","E0");
  document.bgColor=Hintergrund.hex+Hintergrund.R+Hintergrund.G+Hintergrund.B;
}
```

im BODY:

```
<a href="javascript:HintergrundWechseln()"><b>Hintergrundfarbe wechseln</b></a>
```

- Sobald das Objekt angelegt ist, kann man mit Variable=new … jederzeit neue Instanzen dieses Objektes definieren.
- Die Variable Hintergrund enthält hier also Daten des Objekts Farbe und bekommt auch gleich die Werte für die neue Farbe mit, die dann bei document.bgColor= zu einem bekannten Farbcode zusammengesetzt werden.
- Beim Link wird die Funktion HintergrundWechseln() ausgelöst.

Mehrere Anweisungen mit einem Objekt ausführen (with)

```
with(document) {
   open();
      write("Diese Seite hat den Titel " + title);
   close();}
```

- Hier wird das Objekt document mehrfach verwendet: open, write, title, close
- Es ist auch möglich, untergeordnete Objekte auf diese Weise anzusprechen:
  with(document.MeinFormular.Feld_1) value = Stefan

Auf das aktuelle Objekt Bezug nehmen (this)

```
<form name="Eingabe">
<input type="text" name="Feld">
<input type="button" value="OK" onClick="alert(this.form.Feld.value)">
</form>
```

- Mit dem Schlüsselwort `this` kann ich auf ein aktuelles Objekt Bezug nehmen:
  `this.form` statt `document.Eingabe`

Beispiel: Das window-Objekt

- Man unterscheidet zwischen einer Objektklasse (z.B. Fenster) und einer daraus erzeugten Objektinstanz (z.B. MeinFenster).

Beispiel: OBJEKT WINDOW – Eigenschaften	**Beispiel: OBJEKT WINDOW – Methoden**
closed (geschlossenes Fenster) defaultStatus (Normalanzeige in der Statuszeile) innerHeight (Höhe des Anzeigebereichs) innerWidth (Breite des Anzeigebereichs) locationbar (URL-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)	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() (zeitlische Anweisungsfolge setzen) setTimeout() (Timeout setzen) stop() (abbrechen)

- Will man eine Instanz eines Objektes erzeugen, so geschieht dies im allgemeinen mit:
  `MeinObjekt = new Objekt(var1,var2,..)`
  (Ausnahme: MeinFenster=window.open("ich.htm",ich,"width=200,hight=100");)
  Der Zugriff auf Eigenschaften und Daten erfolgt über den Objektnamen gefolgt von einem Punkt und der Eigenschaft (variabel) bzw. Methode (Funktion):
  z.B. window.close(), window.status="Das kommt in die Statuszeile"

OBJEKTHIERARCHIE

- Informationen zur Dokumenthierarchie und die Unterschiede zw. „klass. JavaScript“ und DOM findest du unter http://de.selfhtml.org