Informationen erfassen und bearbeiten.
Einfache und effektive Verfahren mittels Webformularen und JavaScript

Inhaltsverzeichnis

I Programmieren mit JavaScript

I.1 Einleitung

JavaScript wird allgemein als eine leicht zu erlernende Programmiersprache bezeichnet, andererseits aber eingeräumt, daß es nur für kleinere Programme, vor allem zur

erweiterten Programmierung von Webseiten geeignet sei. Was die Erlernbarkeit von JavaScript betrifft, so ist dieses in der Tat verglichen mit anderen Programmiersprachen sogar besonders leicht erlernbar. Das ist schon offensichtlich, wenn man es etwa mit Java, dem JavaScript in vielem ähnlich ist, vergleicht. Dazu kommt die wesentlich leichtere Handhabung von JavaScript. Während der Java-Code eigens von einem besonderen Programm, einem Compiler, übersetzt werden muß, bevor er auf dem Browser ausgeführt wird, da der Browser diesen Code nur in compilierter Form zu interpretieren fähig ist, können die beiden Browser Mozilla und Internet Explorer, jedenfalls in neueren Versionen, JavaScript genau so verstehen wie die Browsersprache HTML. JavaScript wird einfach in den HTML-Quellcode geschrieben und zusammen mit diesem als HTML-Datei ausgeführt. Man kann ein mit JavaScript erstelltes Programm sofort mit dem Browser starten. Das umständliche Deklarieren von Klassen, Datentypen und Variablen wie bei Java entfällt ebenfalls. Viele Sprachelemente sind aber in beiden Programmiersprachen ähnlich, sie finden sich teilweise auch in anderen Programmiersprachen wie z.B. in Perl und in C. Letztere sind ebenfalls wie Java und JavaScript objektorientierte Programmiersprachen, d.h. Sprachen, die sich am gegebenen Objekt orientieren unabhängig von der Hardware und im Falle von Java auch weitgehend unabhängig von der Software, mit dem das Programm ausgeführt wird. Diese Programmiersprachen sind schon wesentlich leicher zu handhaben als die früher üblichen maschinenabhängigen Programmiersprachen, wo ein Programm in viele einzelne kleine, auf die Hardware eines bestimmten Rechnertyps abgestimmte Befehlsfolgen zerlegt werden mußte.

Den Unterschied zwischen Java und JavaScript soll das folgende kleine Programm verdeutlichen. Dieses sieht in Java so aus:

```
package grundkurs.lang;
public class ErstesJava
{
public static void main (String[ ] args)
{
System.out.print ("Mit print kann man schreiben und rechnen:");
Sytem.out.print (" 5 * 10 ="+50);
}}
```

Dasselbe in JavaScript:

```
<script>
document.write ("Mit print kann man schreiben und rechnen:");
document.write (" 5 * 10 ="+50);
</script
```

Wenn Sie dieses kleine JavaScript, das hier in einer etwas verkürzten, aber voll funktionsfähigen Schreibweise steht, in einem Editor zwischen <html> und </html> einfügen, unter irgendeinem Namen speichern und die Datei starten, erscheint im Browserfenster als Ergebnis:

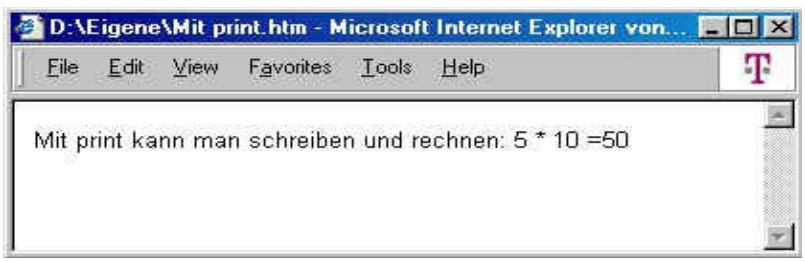


Abb. I.1

In diesem Fall bestehen die Unterschiede zu Java darin, dass *document.write* anstatt *System.out.print* geschrieben wird und dass vor allem die gesamten einleitenden Zeilen samt Klammern fortfallen.

JavaScript ist sogar noch leichter erlernbar, wenn man, wie ich es in den Beispielen dieser Darstellung getan habe, auf alle nicht unbedingt notwendigen Schreibweisen verzichtet. Außerdem ist die praxisnahe Methodik zu empfehlen, sich immer nur diejenigen Elemente einer Programmiersprache anzueignen, die man gerade für seine zu lösende Aufgabe braucht, und das sind für die meisten der hier gezeigten Beispiele relativ wenige, die sich zudem ständig wiederholen. Eine Sache ist auch wesentlich schneller erlernbar, wenn man das Gelernte gleich ausprobieren und in die Praxis umsetzen kann. Mögen die in diesem Grundlagenteil gezeigten Beispiele zu den einzelnen JavaScript-Anweisungen noch recht willkürlich und nutzlos erscheinen, so wird, wie ich hoffe, in den Beispielen des speziellen Teils wesentlich deutlicher, wofür man sie in der Praxis verwenden kann. Nicht nur für die Ausgestaltung von Webseiten, für Animationen, laufende Bildchen usw. Dazu gibt es bereits genügend Darstellungen in zahlreichen anderen Publikationen. Die hier gezeigten Beispiele dagegen sind vielmehr Anwendungen für die praktische wissenschaftliche und informationstechnische Arbeit, die für Studierende, Wissenschaftler, Informationspraktiker, Dokumentare, Bibliothekare usw. von Nutzen sein können. Die eine oder andere Anwendung kann sogar evtl. ein bereits vorhandenes kommerzielles Programm ersetzen oder dieses an individuelle Gegebenheiten anpassen.

Der Zweck dieser Darstellung ist also, möglichst schnell sich die grundlegenden Kenntnisse in JavaScript anzueignen, um dann anschließend mit diesen Grundkenntnissen für das Studium und die Berufspraxis sehr nützliche Programme entwickeln zu können. Das dabei Gelernte kann natürlich außerdem auch eine gute Basis für ein weitergehendes Studium von JavaScript und für ganz andersartige Anwendungen sein.

JavaScript besteht wie ein Baukasten aus lauter Einzelteilen, die man dementsprechend vielseitig zusammenfügen kann. Bemerkenswert ist, dass schon ein einzelner Baustein, eine einzelne Methode, z.B. die Sort-Methode, eine umfangreiche, aus einer großen Zahl von Maschinenbefehlen bestehende Computeraktion, in diesem Fall die alphabetische Anordnung einer Liste, auslösen kann. Wie schon angedeutet, wäre es nicht sehr sinnvoll, alle Bauteile im einzelnen zu studieren. Zunächst genügt ein Überblick über den Umfang des Angebotenen. Unverzichtbare Voraussetzung ist allerdings, dass man über die grundlegenden Bauregeln Bescheid weiß. Sehr wesentlich für die leichte Erlernbarkeit von JavaScript ist, dass der Umfang dieser Grundregeln im Verhältnis zu den gesamten Regeln und Befehlsanweisungen von JavaScript, das inzwischen auch bedeutende Erweiterungen erfahren hat, relativ klein ist.

Die ebenfalls weit verbreitete Meinung, JavaScript sei ausschließlich für kleinere Programme geeignet, widerlegen die im speziellen Teil dieses Buches gezeigten Beispiele. Sie zeigen, daß es möglich ist, JavaScript auch bei Aufgabengebieten anzuwenden, die allgemein als Domänen für größere, "professionellere" Programmier-sprachen gelten, natürlich mit gewissen Einschränkungen: Sehr komplexe und grosse Datenbanksysteme, umfangreiche Textbearbeitungen usw. lassen sich damit nicht oder nicht so gut verwirklichen. Früher scheiterte das an der begrenzten zugelassenen Kapazität von Formular-Dateien bei Browsern, besonders bei Netscape, wo JavaScript-Dateien nur bis etwa 70 KB unterzubringen waren. Das hat sich inzwischen wesentlich

geändert. Formulardateien können jetzt Daten bis zu mehreren Megabyte aufnehmen. Von einer gewissen Grösse an kann aber die längere Lade- und Bearbeitungszeit eine Grenze setzen.

Um einen mit JavaScript bearbeiteten Text, eine Liste usw. über einen Schalterklick in einer Datei zu speichern oder eine Datenbank zu verändern ist die Verwendung des ActiveXObjects des IE erforderlich, das ich in einem eigenen Kapitel (I.4) behandle.

Eine weitere wichtige Einschränkung: Der direkte Zugriff auf die Webseiten einer anderen Domain, also eines auswärtigen Rechners, der etwa die Datenbankergebnisse zur einer Rechercheabfrage liefert, ist mit den hier gezeigten Anwendungen nicht möglich. Der Zugriff auf die Webseite eines auswärtigen Rechners kann hier nur von außen erfolgen, und zwar durch das Einfügen des kopierten Inhalts in das Texteingabefeld eines Formulars. Das Formular dient bei allen diesen Beispielen quasi als Schnittstelle. Die dafür notwendigen drei Tastenbefehle sind sehr bequem und schnell auszuführen und dürften, einmal eingeübt, kein Hemmnis sein. Im Gegenteil: die geforderte Aktivität des Benutzers und die Transparenz des Verfahrens können willkommene Anregungen zum Nachvollziehen dieser Anwendungen und vor allem zum eigenen Programmieren sein.

I.2 Grundlagen von JavaScript

JavaScript wird ebenso wie der HTML-Quellcode in einen Editor geschrieben. Ein JavaScript-Programm kann zwar häufig auch in einer eigenen Script-Datei (mit der Dateiendung js) existieren, oft aber ist es Teil einer HTML-Datei. Eine HTML-Datei wird mit <html> eingeleitet und mit </html> beendet. Dazwischen schreiben Sie nun das JavaScript. Dieses beginnen Sie mit <script> und beenden es mit </script>. Da möglicherweise außer JavaScript noch weitere Schriftsprachen entstehen, sollte noch das Attribut *language="JavaScript"* hinzugefügt werden. Zur Zeit funktioniert JavaScript aber auch ohne dieses Attribut. Ich habe es deshalb in allen Beispielen weggelassen. Wegen einer besseren Übersicht habe ich auch die empfohlenen Kommentarhinweise <!... ...> für ältere Browser, die kein JavaScript verstehen, weggelassen, weil die hier gezeigten Anwendungen ohnehin nur für die neueren Browserversionen geeignet sind. Auf die Einteilung einer HTML-Seite in einen Head- und Bodyteil habe ich ebenfalls verzichtet. Bei der Verwendung eines Webeditors wird die Seite automatisch in einen Head- und Bodyteil unterteilt. JavaScript sollte zwar in den Headteil gesetzt werden, aber es funktioniert in allen angegebenen Beispielen genau so gut vom Bodyteil aus oder sogar auch dann, wenn die HTMLSeite nicht in einen Head- und Bodyteil unterteilt ist. Schließlich habe ich darauf verzichtet, alle in einem Script verwendeten Variablen zu Beginn des Scripts mit der Variblenbenennung *var* zu definieren, was gründliche Informatiker vielleicht bemängeln werden. Alle Variablen werden erst dann definiert, wenn sie im Laufe des Programms mit einem Wert belegt werden. Wird einer freien Benennung ein Wert zugewiesen, ist damit bereits diese Benennung als Variable definiert, ohne daß die feste Benennung var in Erscheinung tritt. Dabei muß man allerdings vermeiden, gleichnamige Variablen mehrdeutig zu verwenden. So ist darauf zu achten, ob eine Variable innerhalb eines ganzen Scripts, einer ganzen Funktion oder nur innerhalb eines Teils davon, z.B. in einer For-Schleife oder If-Bedingung, gelten soll.

Ein erstes kleines JavaScript

Das denkbar kleinste JavaScript, eingebettet in eine minimale HTML-Datei, könnte etwa so aussehen:

<html>
<script>
alert ("Hallo!");
</script>
</html>

Schreiben Sie dieses Skript in einen Editor, speichern Sie es unter irgendeinem Dateinamen, z.B. *Hallo.html* ab und öffnen Sie diese Datei vom Dateimanager oder Explorer aus, so startet der Browser und zeigt in einem kleinen Fenster die Meldung "Hallo".

Der Ausdruck *alert* ist eine der vielen Anweisungen oder Befehle von JavaScript, die auch Methoden genannt werden. Mit der Methode *alert* wird immer ein kleineres oder größeres Fenster geöffnet und darin ein Text angezeigt. Die Größe des Fensters richtet sich nach dem Umfang des Textes. Ein längerer Text kann über den Bildschirm hinausgehen. Der Text, der gemeldet werden soll, wird in der Code-Zeile innerhalb einer Klammer und in Anführungszeichen geschrieben. Die Anführungszeichen definieren einen wörtlich zu lesenden Text, einen sogen. String. Diese Anführungszeichen sind sehr wichtig. Läßt man sie weg, erscheint eine Fehlermeldung, die im IE so aussieht:

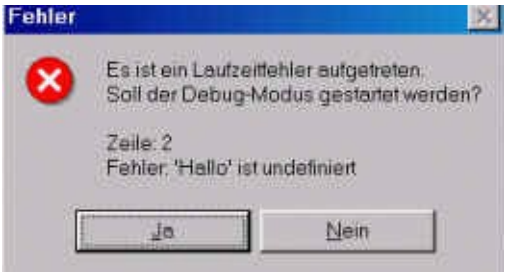


Abb I.2 Fehlermeldung im Internet Explorer

In diesem Fall, d.h. wenn man *alert (Hallo);* anstatt *alert ("Hallo");* schreibt, hat JavaScript *Hallo* wegen der fehlenden Anführungszeichen nicht als Stringobjekt, sondern als eine Variable oder als Objektbenennung interpretiert. Die Fehlermeldung erfolgt, weil eine vorangehende Definition der Variablen oder des Objekts *Hallo* fehlt.

I.2.1. Variablen

In etwas abgeänderter, erweiterten Weise kann das Skript auch so geschrieben werden, was dasselbe bewirkt:

1) <html>
2) <script>
3) v="Hallo";
4) alert(v);
5) </script>
6) </html>

Hier ist die Zeile 3 *v="Hallo"* eingefügt. In Zeile 4 steht innerhalb der Klammer nur *v.* Das kleine *v* ist hier eine frei gewählte Benennung, eine sogen. Variable. Hinter dem Gleichheitszeichen folgt die in Anführungszeichen stehende Zeichenfolge *Hallo*, ein sogen. String. Die Anweisung in Zeile 3 bewirkt, dass der Computer die in Anführungszeichen gesetzte Zeichenfolge *Hallo* in der Variablen *v* speichert. Beachten Sie hierbei das Gleichheitszeichen, das hier im Sinne einer Zuweisung gebraucht wird. Ein JavaScript besteht zu einem großen Teil aus solchen mit einem Gleichheitszeichen ausgedrückten

Zuweisungen, wobei jeweils eine Zeichenfolge, ein Wert oder ein Objekt einer Variablen zugewiesen wird. In diesem Beispiel wird die Variable mit der frei gewählten Benennung *v* bezeichnet. Anstatt *v* könnte hier auch eine andere Benennung stehen oder irgendein beliebiges Wort oder Zeichenfolge. Wichtig ist dabei, daß kein reserviertes Wort der Programmiersprache JavaScript gewählt wird, z.B. *string*, das die festgelegte Benennung für das Stringobjekt ist . Außerdem sollte für Sie selbst die Benennung merkfähig sein, um in einem längeren Programm nicht die Übersicht zu verlieren und um nicht etwa Benennungen doppelt zu vergeben. Im weiteren Programmverlauf beinhaltet *v* also immer die dieser Variablen zugewiesene Zeichenfolge *Hallo.* So können auch sehr lange Zeichenfolgen mit einer kurzen, frei gewählten Benennung, eben einer Variablen, bezeichnet werden. Im weiteren Verlauf kann das Programm auf diese Variable *v* wieder zugreifen und etwa den in ihr gespeicherten Wert oder wie hier den String *Hallo* verändern, so daß die Variable zu einer veränderten Variablen wird, während sie in diesem Beispiel unverändert bleibt.

Was geschieht in diesem Beispiel innerhalb des Computers? Die Zeichenfolge *Hallo* wird als Stringobjekt der Variablen *v* zugewiesen. Der Computer speichert dann in einem Speicherplatz mit dem Namen *v* diese Zeichenfolge. In der nächsten Zeile wird der Computer angewiesen mit der Methode *alert* auf diesen Speicherplatz *v* zuzugreifen, den in *v* gespeicherten Wert auszulesen und das Ergebnis in einem Meldefenster anzuzeigen.

Wie in anderen Programmiersprachen werden in JavaScript also sowohl feststehende Benennungen als auch frei gewählte Benennungen verwendet. Feststehende Benennungen werden für eine Anzahl von Objekten gebraucht, die im einzelnen noch erläutert werden, vor allem aber für die durchzuführenden Methoden oder Funktionen, die an einem Objekt (manchmal auch unabhängig von einem Objekt) ausgeführt werden. Solche Methoden besitzen ausschließlich feste Benennungen.

Für Variable verwenden Sie grundsätzlich nur frei gewählte Benennungen, denen Sie irgendwelche Werte oder Daten zuweisen. Das können Zahlenwerte, eine Zeichenfolge wie im vorigen Beispiel, Texte usw. sein, die im weiteren Programm entweder verändert werden oder auch unverändert bleiben können. Die Veränderung kann auf zweifache Weise geschehen:

1. Der in der Variablen gespeicherte Wert, z.B. eine Zeichenkette, ein String, wird verändert. Zum Beispiel aus "sehr schön" wird *"Sehr Schön"*. Dann muß der veränderte String in einer neuen Variablen, z.B. in *va*, gespeichert werden. Ein Beispiel:

```
v="datenbanken";
va=v.toUpperCase();
document.write(va);
```

In Zeile 1 wird die Zeichenfolge datenbanken in *v* gespeichert. In Zeile 2 wird der in *v* gespeicherte Wert mit der Methode *toUpperCase()* in Großbuchstaben verwandelt und in der neuen Variablen *va* gespeichert. In Zeile 3 wird *va* ausgeschrieben, d.h. auf dem Bildschirm erscheint *DATENBANKEN*.

2. Der Variablen werden im Programmverlauf verschiedene Werte zugewiesen. Zum Beispiel wird aus einer Suchergebnisliste von Aufsätzen zu jedem Treffer in einer Variablen das Jahr der Veröffentlichung gespeichert und dieses später wieder abgefragt. Diese Zuweisung verschiedener Werte zu einer Variablen wird in den Beispielen von Teil II ausführlich erläutert.

I.2.2 Objekte. Methoden. Eigenschaften

Ein weiteres kleines Programm kann etwa so aussehen:

```
1) <html>
2) <script>
3) v="Hallo";
4) document.write (v);
5) </script>
6) </html>
```

Hier wurde Zeile 4 des vorigen Beispiels in *alert (v);* in *document.write (v);* abgeändert. *document* ist die feste Benennung für das im Browserfenster geöffnete aktuelle Dokument, eines von vielen in JavaScript vordefinierten Objekten mit feststehenden Benennungen. Objekte sind einfach Gegenstände, die mit JavaScript bearbeitet werden können. Das können Zeichenfolgen, sogen. Stringobjekte, Frameabschnitte, Formularelemente, URL-Adressen, mathematische Objekte usw. sein.

Das hinter dem Punkt stehende *write* ist eine der vielen Methoden, die verursacht, daß der in Klammern angegebene Text oder der in der Variablen *v* gespeicherte Text im geöffneten Dokument ausgeschrieben wird. Wenn Sie das kleine Programm starten, so erscheint im Browserfenster das Dokument mit dem ausgeschriebenen *Hallo*. *write* ist eine festgelegte Methoden-Benennung und immer nur mit dem Objekt *document* verbunden. Die Anweisung *document.write* ist ziemlich häufig und taucht bei allen in diesem Buch besprochenen Beispielen auf. Methoden sind fast immer an ein bestimmtes Objekt gebunden und werden hinter einem Punkt an das zugehörige Objekt angefügt. Einige Methoden des Objektes *window* werden ohne vorhergehende Objektbenennung geschrieben. Dazu gehört das oben benutzte *alert*.

I.2.1.1 Objekthierarchie

Objekte von JavaScript sind wie in einer hierarchischen Struktur einander über- und untergeordnet. So ist das Objekt *document* dem Objekt *window*, das an der obersten Stelle dieser Struktur steht, untergeordnet. Diese Reihenfolge ist bei allen Anweisungen immer zu beachten. So ist das Formularobjekt *forms* dem Objekt *document* untergeordnet, und ein einzelnes Formularfeld, z.B. ein Texteingabefeld ist eine Objekteigenschaft von *forms*, die mit *elements* benannt wird.

Andererseits ist es nicht immer nötig, das oberste Objekt *window* anzugeben. So müßte die oben genannte Anweisung streng genommen *window.document.write* geschrieben werden. Es ist aber üblich, hier die Objektbenennung *window* einfach wegzulassen.

Die für ein bestimmtes Objekt möglichen Bearbeitungsmethoden oder Funktionen sowie Eigenschaften werden hinter der Objektbenennung hinter einem Punkt angefügt. Benennungen von Objekten können entweder festgelegt sein, wie etwa die Objekte *window, document* oder *location,* oder frei gewählt werden. Häufig wird ein Objekt mit einer Methode bearbeitet und das Ergebnis einer Variablen mit freier Benennung zugewiesen. In den folgenden Programmbeispielen werden besonders häufig String- und Array-Objekte verwendet. So ist die in Anführungszeichen gesetzte Zeichenkette *hallo*, bereits ein Stringobjekt und wird, ohne daß die feste Objektbezeichnung *String* verwendet wird, in einer freien Variablenbezeichnung gespeichert. Damit ist das Objekt bereits erzeugt, wie es in der Fachsprache heißt. Im weiteren Verlauf kann das Stringobjekt mit verschiedenen Methoden verändert und selbstgewählten Variablenbezeichnungen

zugewiesen werden.

Beim Arrayobjekt ist es ähnlich. Die Methode *split,* auf einen String angewandt, erzeugt bereits ein Array, ohne daß der Objektname *Array* verwendet wird. Das Array wird in einer beliebigen Variablenbezeichnung gespeichert und kann mit weiteren Methoden weiterbearbeitet werden, z.B. mit der Sort-Methode.

Beispiel:

```
<script>
1) T="AB BC BE AF GH BB";
2) TA=T.split(" ");
3) TB=TA.sort();
4) document.write(TB);
</script>
```

- In Zeile 1 wird die Zeichenkette *AB BC DE EF GH IJ* in Anführungszeichen gesetzt, womit ein Stringobjekt erzeugt wird, und in der Variablen *T* gespeichert.
- In Zeile 2 wird mit der Splitmethode dieser String in einzelne Teile zerlegt, d.h. ein Array wird erzeugt. In der Klammer hinter split ist das Trennungszeichen angegeben, in diesem Fall eine Leerstelle, ausgedrückt durch eine Leerstelle zwischen zwei Anführungszeichen. Das Ergebnis wird in der neuen Variablen *TA* gespeichert.
- In Zeile 3 wird mit der Sortmethode das Array alphabetisch geordnet. Das Ergebnis in *TB* gespeichert.
- In Zeile 4 wird das Ergebnis ausgeschrieben.
- Im Browser wird *AB,AF,BB,BC,BE,GH* angezeigt. Die einzelnen Teile eines Arrays werden durch Kommata voneinander getrennt.

Über die Erzeugung eines Arrays mit einem Konstruktor (Objektname + Schlüsselwort *new*) siehe unten.

Neben Methoden gibt es auch noch Eigenschaften von Objekten, die ebenfalls nach einem Punkt an das Objekt angehängt werden. Auf die Methodenbezeichnung folgt jeweils in Klammern die Parameterangabe, d.h. die zu bearbeitenden Werte. Eigenschaften sind von Methoden nicht immer leicht zu unterscheiden, außer formal dadurch, daß Eigenschaften keine Parameterwerte in Klammern aufweisen. Eigenschaften können herausgelesen und vielfach auch verändert werden.

Beim Programmieren in JavaScript wird ein gegebenes Objekt mit einer mit diesem Objekt verbundenen Methode oder Eigenschaft bearbeitet, verändert usw. und das Ergebnis in einer selbstgewählten Variablenbenennung gespeichert. Das Objekt entsteht entweder automatisch wie z.B. vielfach beim String- und Arrayobjekt, oder es wird durch einen Konstruktor erzeugt mit dem Schlüsselwort *new*, was häufig bei Arrays und sogen. regulären Ausdrücken (*regular expressions*) geschieht.

Objekte sind im Grunde nichts anderes als Variablen, auf die bestimmte Prozeduren oder Methoden angewandt werden. Variablen können andererseits zu Objekten werden. So wird eine Zeichenkette in einer Variablen gespeichert. Diese Variable wiederum steht dann für ein Stringobjekt, an dem weitere Methoden und damit Änderungen durchgeführt und das Ergebnis in einer neuen Variablen gespeichert wird, usw. Es gibt aber auch Variablen, auf die keine bestimmten Prozeduren angewandt werden können. So ist der bei einem String mit der Index-Methode ermittelte Wert ein Zahlenwert, mit dem wohl weiter verfahren, z.B. gerechnet, auf die aber keine Methode angewandt werden kann.

I.2.3 Funktionen

In den vorangehenden Beispielen wurde das JavaScript automatisch beim Starten des Browsers ausgeführt. In den kleinen Skripten befinden sich zwischen den beiden Tags *<script>* und *</script>* jeweils die einzelnen Befehle. Ein solches JavaScript ohne eine Einbettung in eine Funktion kann auch aus sehr vielen Befehlen oder Anweisungen bestehen, die einfach mit dem Laden der Datei automatisch ausgeführt werden. Das ist in der Praxis aber eher die Ausnahme. In den meisten Fällen wird das Ausführen eines Programms oder Programmteils von einer bestimmten Bedingung abhängig gemacht, je nachdem ob man diesen oder jenen Schalter oder Hyperlinkverweis anklickt z.B., oder ob innerhalb einer Folge von Anweisungen eine Bedingung enthalten ist, die zur Ausführung des Teilprogramms führt. In allen diesen Fällen muß das auszuführende Skript in einer Funktion mit entsprechender Funktionsbezeichnung eingebettet sein, damit das Skript aufgerufen und ausgeführt wird.

Ein JavaScript kann aus einer oder auch aus vielen Funktionen bestehen. Die auszuführenden Programmanweisungen werden mit *function* eingeleitet und erhalten einen Namen, der nach einer Leerstelle folgt als frei gewählte Bezeichnung mit folgenden Klammerzeichen (). Die Programmanweisungen werden außerdem durch die geschweiften Klammerzeichen { und } eingeleitet und abgeschlossen, so daß das oben angegebene Skript innerhalb einer Funktion in folgender Weise geschrieben wird:

```
1) <html>
2) <script>
3) function hallo()
4) {
5) v="Hallo";
6) document.write (v);
7) }
8) </script>
9) </html>
```

I 2.4 Eventhandler

Wenn Sie eine Datei mit diesen Zeilen als Quellcode starten, geschieht aber zunächst gar nichts. Es fehlt noch eine Anweisung, die veranlaßt, daß diese Funktion ausgeführt wird. Eine solche Anweisung, wenn sie sich außerhalb des Skriptbereichs, aber innerhalb des HTML-Codes befindet, wird auch als Event-Handler bezeichnet. Soll die Programmfunktion in diesem Beispiel zusammen mit dem Browser gestartet werden, so wird dies innerhalb der Bodytags des HTML-Codes so angegeben:

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

Die Anweisung *onload* ist einer der vielen Eventhandler von JavaScript. Hinter dem Ist-Zeichen folgt der Name der aufzurufenden Funktion in Anführungszeichen und mit Klammern, also *hallo().*

Die ganze Programmdatei in vollständiger Schreibweise:

```
1) <html>
2) <head>
3) <meta http-equiv="Content-Type"
4)  content="text/html; charset=iso-8859-1">
5) <meta name="GENERATOR" content="Microsoft FrontPage Express 2.0">
6) <title></title>
```

```
7) </head>
8) <body onload="hallo()">
9) <script language="JavaScript"><!--
10) function hallo()
11) {
12) v="Hallo";
13) document.write (v);
14) }
15) // --></script>
16) </body>
17) </html>
```

So sieht der vollständige Quellcode aus, wenn man den Script-Code und den HTML-Code in einen Webeditor, z.B. in den Front Page Express, schreibt und das Ganze in einer Datei speichert. Man erkennt die nicht unbedingt notwendigen und sonst von mir weggelassenen Ergänzungen zwischen *<head>* und *</head>,* den Zusatz *language="JavaScript"* und die Kommentarzeichen <!-- und // --> für Browser, die JavaScript nicht lesen können.

I.2.5 Das String-Objekt. Zeichenketten bearbeiten

Im vorigen Beispiel wurde eine Zeichenkette, ein sogen. String, bearbeitet. Das Stringobjekt ist eines der fundamentalen Objekte in JavaScript, das besonders viele Methoden besitzt. In den in diesem Buch dargestellten Beispielen spielt es eine hervorragende Rolle. Immer wenn Texte oder Listen zu bearbeiten sind, kommt das String-Objekt vorrangig zum Einsatz. Anstatt des sehr kurzen Strings *Hallo*, können Strings auch sehr lang sein. Es können ganze Sätze, ja sehr lange Texte und Listen von Hunderten von Seiten als String behandelt werden. Das Stringobjekt wird automatisch dadurch erzeugt, daß die zu bearbeitende Zeichenkette in Anführungszeichen gesetzt und einer Variablen mit einer frei gewählten Benennung zugewiesen wird. So z.B. in dem folgenden Beispiel:

```
<html>
<script>
S="Das ist aber sehr schön";
alert (S);
</script>
</html>
```

Wenn Sie dieses Skript als Datei speichern und danach öffnen, so wird in dem Alert-Fenster der String *S* angezeigt:

Das ist aber sehr schön

Sie können längere Sätze oder ganze Texte nur als String verwenden, wenn diese keinen manuellen Zeilenumbruch aufweisen. Sobald Sie einen solchen verwenden, z.B. in den Editor schreiben:

```
document.write ("Mit print kann man
schreiben und rechnen:")
```

erhalten Sie die Fehlermeldung:

SCRIPT1015: Nicht abgeschlossene Zeichenfolgenkonstante
Datei: Fehler.htm, Zeile: 4, Spalte: 12

I.2.5.1 Substring( )

Mit dieser sehr häufig angewandten Methode wird aus einem gegebenen String ein Teilstück ermittelt. Dieses Teilstück wird in Klammern mit zwei Positionen angegeben: die Position des ersten Zeichens des Teilstücks und die Position, die auf das letzte Zeichen des Teilstücks folgt.

Beispiel:

```
<script>
T="Das ist aber sehr schön";
TS=T.substring(0,4);
alert(TS);
</script>
```

Das Teilstück beginnt mit der Position des Anfangszeichens *D* des Strings *T* und endet vor der Position 4 des Strings. Position 4 ist hier die Leerstelle hinter *Das*. Als Teilstück wird mit alert *Das* angezeigt.

substr() stellt eine Variante der vorigen Methode dar. Hierbei gibt der zweite Wert in der Klammer die Länge des Teilstücks an, dessen Anfangsposition mit dem ersten Wert angegeben wird.

I.2.5.2 Slice ( )

Ist eine ähnliche Methode wie die vorigen, nur daß hier aus einer Zeichenkette ein Teilstück ab einer angegebenen Position herausgeschnitten und in einer Variablen gespeichert wird. Wird in der Klammer nur der Wert dieser Position angegeben, so wird das Teilstück von dieser Position an bis zum Ende herausgeschnitten. Wird nach einem Komma ein zweiter Wert angegeben, so gibt dieser die Position des letzten auszuschneidenden Zeichens an. Im Unterschied dazu gibt bei substring der zweite Wert die auf das Ende des Teilstücks folgende Position an.

```
<script>
T="Das ist aber sehr schön";
TS=T.slice(4);
alert(TS);
</script>
```

Mit alert wird der String *ist aber sehr schön* angezeigt.

I.2.5.3 Replace ( )

Eine der wichtigsten Methoden des Stringobjekts ist die Methode *replace*. Auch bei dieser Methode folgt eine Klammer. In dieser wird die Zeichenfolge (String), die ersetzt werden soll, und nach einem Komma die Zeichenfolge, die diese ersetzt, angegeben.

```
1) <html>
2) <script>
3) S="Das ist aber sehr schön und sehr gut";
4) Sa=S.replace("sehr", "nicht");
5) alert (Sa);
6) </script>
7) </html>
```

Der zu verändernde String wird der Vaiablen *S* zugewiesen (3), der geänderte String der Variablen *Sa* (4). In Klammern der zu ersetzende und der ersetzende String. Das Wort *sehr* wird durch *nicht* ersetzt. Schließlich wird der geänderte Satz durch die Alert-Methode angezeigt (5).

Sollen Zeichen, Wörter oder Teilstrings, so oft sie in einem Satz oder Text vorkommen, ersetzt werden, z.B. in dem Satz *Das ist aber sehr schön und sehr gut* das *sehr* überall durch *nicht* ersetzt werden, so muß Zeile 3 so geschrieben werden:

Sa=S.replace(/sehr/g, "nicht");

Die Schrägstriche bezeichnen einen sogen. regulären Ausdruck. Ein regulärer Ausdruck ist ein Suchmuster, mit dessen Hilfe Zeichenfolgen in einem Stringobjekt wie hier mit der Replace-Methode durch eine andere Zeichenfolge ersetzt werden. Das auf den Schrägstrich folgende *g* bedeutet, daß der reguläre Ausdruck global, also überall durch den in Anführungszeichen stehenden String ersetzt werden soll. Häufige Anwendungen des regulären Ausdrucks finden auch bei der Match- und Search-Methode statt (s.u.).

I.2.5.4 toUpperCase() -- toLowerCase()

Diese Methoden wandeln einen String in Groß- bzw. Kleinbuchstaben um. In den Klammern werden keine Parameter angegeben.

I.2.5.5 length

Eine Eigenschaft des Stringobjekts (deshalb ohne Klammer). Ermittelt die Länge, d.h. die Anzahl der Zeichen und Leerzeichen eines Strings.

I.2.5.4 link()

Mit dieser Methode wird ein String in einen Hyperlink verwandelt. In der Klammer wird die URL-Adresse mit Anführungszeichen angegeben. Bewirkt dasselbe wie in HTML über das Attribut *HREF* und die Verwendung der Anker *<a>* und **, zwischen denen der anklickbare Text gesetzt wird.

Beispiel:

```
1) <script>
2) ZS="Acta Informatica".link("http://link.springer.de/link/service/journals/00236/");
3) document.write(ZS);
4) </script>
```

In der 2. Zeile wird der Hyperlink gebildet, in der 3. Zeile ausgeschrieben. Die Abbildung zeigt den anklickbaren Zeitschriftentitel, der zur originalen Internetseite dieser Zeitschrift führt.

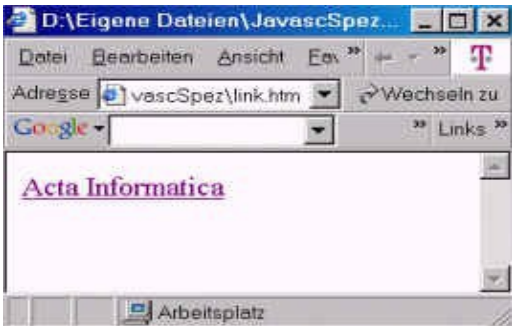


Abb. I.2

I.2.6 Das Objekt form

Bei den in den Kapiteln II bis V erläuterten Anwendungen sind Webformulare von wesentlicher Bedeutung. JavaScript verwendet dafür das Objekt *form* mit dem Unterobjekt *elements.* Unter *elements* können die einzelnen Schalter, Menus und Eingabefelder angesprochen werden. Mit folgenden wichtigen Eigenschaften und Methoden kann der Zugriff auf ein Formular bzw. auf eines der Elemente erfolgen:

- submit() das betreffende Formular wird abgeschickt
- action enthält die Adresse des anzuwählenden Ziels eines Formulars
- reset(), löscht alle Eingaben des Formulars
- click() erzeugt einen Mausklick bei einem Formularelement
- checked ermittelt, ob eine Checkbox ausgewählt wurde
- option Eintrag eines Menüs wird angesprochen
- selected ermittelt, ob ein Eintrag eines Menüs ausgewählt wurde
- selectedIndex ermittelt Indexnummer eines ausgewählten Eintrags eines Menüs
- value der Wert d.h. Inhalt eines Formularelements, kann ausgelesen und belegt werden

I.2.5.1. value

Eigenschaft des Form-Objekts. Wird in ein mehrzeiliges Eingabefeld *textarea* ein Text, eine Liste usw. eingefügt. so kann der Inhalt durch folgende Anweisung wieder ausgelesen werden:

a=document.forma.area.value,

Hier ist *area* die freie Bezeichnung des Textarea-Feldes. Der Inhalt des Feldes wird in der Varablen *a* als String gespeichert und kann anschliessend beliebig bearbeitet werden. Bemerkenswert ist dabei, dass z.B. umfangreiche Texte, Zeitschriftenlisten usw. in ein Textareafeld eingefügt werden können und dann als Stringobjekt zur Verfügung stehen. Der Objektname *string* tritt nicht in Erscheinung.

Umgekehrt kann der in einer Variablen *a* gespeicherte String in ein Eingabefeld abgelegt werden:

document.forma.area.value=a

In diesem Fall ist sowohl ein mehrzeiliges Textareafeld als auch in ein einzeiliges Textfeld, bezeichnet mit *<input type=text>* möglich. Auch dabei kann der String eine sehr umfangreiche Liste enthalten.

Umfangreiche Listen können ausserdem fest sowohl in mehrzeilige Textareafelder als auch in einzeilige Textfelder eingespeichert werden. Bei Textfeldern allerdings muss der String in Anführungszeichen gesetzt werden: *<input type=text value=".....">* und darf

selber keine Anführungszeichen enthalten. [1]

I.2.7 Die Suchmethoden indexOf, search und match

Es gibt eine Reihe von Suchmethoden, von denen hier die wichtigsten vorgestellt werden. Die beiden Methoden *indexOf* und *search* werden hierbei in ähnlicher Weise angewandt, während sich die Methode *match* wesentlich davon unterscheidet. Folgendes Beispiel zeigt die Anwendung von indexOf ( ):

```
1) <html>
2) <script>
3) S="Das ist aber sehr schön und sehr gut";
4) P=S.indexOf("sehr");
5) alert (P);
6) </script>
7) </html>
```

Wenn Sie diese Datei starten, zeigt *alert* in einem Fenster 13 an. Gesucht wird der String bzw. Zeichenkette *sehr* innerhalb des gesamten Strings *S*. Das erste Zeichen des gesamten Strings hat die Position *0. 13* ist die Position des ersten Zeichens *s* von *sehr,* gespeichert in der Variablen *P*. Ein Leerraum zwischen den Wörtern wird wie ein Zeichen gezählt.

Bei der indexOf-Methode kann man außerdem eine Zeichenfolge ab einer bestimmten Position suchen. Dies ist eine besondere Erweiterungsmöglichkeit dieser Methode, die bei der Search-Methode nicht vorhanden ist. Die maßgebliche Position wird nach einem Komma in der Klammer angegeben (s. Zeile 5):

```
1) <html>
2) <script>
3) S="Das ist aber sehr schön und sehr gut";
4) P1=S.indexOf("sehr");
5) P2=P2=S.indexOf("sehr",P1+1);
6) alert (P2);
7) </script>
8) </html>
```

P1 ist die Position des ersten Treffers, wie im vorigen Beispiel =13. P2 ist die Position des 2. Treffers, gesucht von der Position des 1. Treffers aus: P1+1. Die 1 muß hinzu addiert werden, sonst würde wieder nur der erste Treffer an der Position P1 gefunden. Die Position des 2. Treffers ist 28.

Das folgende Beispiel zeigt die Anwendung von search. Auch hierbei wird wie bei der IndexOf-Methode nach der Position des ersten Zeichens eines Strings gesucht.

```
1) <html>
2) <script>
3) S="Das ist aber sehr schön und sehr gut";
4) P=S.search("sehr");
5) alert (P);
6) </script>
7) </html>
```

1 Auch versteckte Textfelder lassen sich so mit Inhalt belegen: <input type=hidden value=".....">

Durch *alert(P)* wird der Wert *P* angezeigt, der die erste Position der Zeichenfolge sehr im String *S* angibt, in diesem Fall ist dies wieder 13. Mit search lässt sich immer nur die erste Position von mehrfach in einem String enthaltenen Zeichenfolgen suchen.

I.2.7.1 Suchen mit regulären Ausdrücken

search()

Diese Methode bietet ebenso wie die weiter unten dargestellte Match-Methode die Möglichkeit mit regulären Ausdrücken zu suchen. Ein Beispiel:

```
1) <html>
2) <script>
3) S="Das ist aber sehr schön und sehr gut";
4) P=S.search(/se\w+/);
5) alert (P);
6) </script>
7) </html>
```

In der Klammer ist innerhalb von Schrägstrichen / das zu suchende Suchmuster angegeben /se\w+/. Die Buchstaben se sind die zu suchenden Zeichen, gefolgt von einem oder mehreren beliebigen Zeichen, ausgedrückt durch \w+. Beliebige Ziffern, Buchstaben, Leerstellen oder Umbrüche werden mit dem Backlash \ eingeleitet und dahinter die entsprechende Bezeichung für Buchstaben, Ziffern, Leerstellen, Zeilenumbrüche, Neue Zeilen usw.

match()

Ein Beispiel:

```
1) <html>
2) <script>
3) S="Das ist aber sehr schön und sehr gut";
4) Treffer=S.match(/s\w+/g);
5) alert (Treffer);
6) </script>
7) </html>
```

Dieses Beispiel zeigt die Verwendung der Methode *match*, angewandt auf das in der Variablen *S* gespeicherte Stringobjekt. Auch hier wird die Variable wiederum zum Objekt. In der Klammer wurde zu dem regulären Ausdruck noch ein g hinzugefügt, was bedeutet, daß der reguläre Ausdruck global, d.h. im gesamten Text gesucht werden soll. Das Ergebnis wird in Treffer gespeichert und mit der Alertmethode angezeigt. In diesem Fall wird als Ergebnis sehr, sehr angezeigt. Beachten Sie, daß das Ergebnis ein sogen. Array ist, eine Folge von gleichartigen Elementen. Mehr darüber siehe weiter unten.

Wird der reguläre Ausdruck mit /s\w+/g angegeben, so erscheint als Ergebnis:

Abb. I.3

In diesem Fall werden alle Zeichenfolgen gesucht, die mit *s* beginnen und auf das ein oder mehrere Buchstaben bis zu einem Leerraum folgen. Das Wort *schön* wird nur bis zu den Buchstaben *sch* gelesen, weil der Umlaut ö ohne Umwandlung so nicht lesbar ist. Die einzelnen Treffer werden, wie bei einem Array üblich, durch Kommata getrennt.

Mit der Match-Methode kann auch ein längerer Text in seine einzelnen Wörter zerlegt werden. Das erreicht man, indem man den regulären Ausdruck (/\w+/g verwendet. Als Ergebnis wird angezeigt:

Abb. I.4

Hier ist der ganze Satz in Form eines Arrays angezeigt, die einzelnen Wörter durch Kommata getrennt. Mit dieser Methode lassen sich auch bestimmte Folgen von Zahlen herauslesen.

I.2.8 Reguläre Ausdrücke

Dies sind Suchmuster, mit denen Texte oder Datenbankinhalte sehr effektiv durchsucht werden können. Wollen wir z.B. in einem Text alle mit un beginnendenWörter suchen, so wäre der entsprechende reguläre Ausdruck dafür

/\bun\w+\s/g

oder:
new RegExp("\\bun\\w+\\s","g").

Bei der ersten Schreibweise werden die zu suchenden Buchstaben *un* mit den speziellen Bezeichnungen *\b* für Wortanfang, *\w+* für mehrere beliebige alphanumerische Zeichen und *\s* für eine Leerstelle zwischen Schrägstriche gesetzt, dahinter *g* als Kennzeichen für eine globale Suche. \bun\w+\s sucht also zunächst un am Wortanfang, dann die weiteren folgenden alphanumerischen Zeichen und schließlich eine Leerstelle, die das Wort im Text abschließt. Diese Schreibweise kann man direkt bei den Methoden *match, search* und *replace* anwenden, also z.B.

Tr=T.match(/\bun\w+\s/g);

Der reguläre Ausdruck wird hier als Suchmuster in dem Text T gesucht. Die Treffer werden als Array in *Tr* gespeichert.

Bei der zweiten Schreibweise muß der über das Schlüsselwort *new* erzeugte reguläre Ausdruck zunächst in einer Variablen gespeichert werden:

A=new RegExp("\\bun\\w+\\s","g");

Hier sind alle Kennzeichnungen in Anführungszeichen gesetzt. Die speziellen

Kennzeichnungen *\b, \w* und *\s* werden durch ein zusätzliches \ gekennzeichnet. Das *g* steht, durch ein Komma getrennt, ebenfalls in Anführungszeichen. Danach kann diese Variable bei einer Methode eingesetzt werden. Das obere Beispiel der Match-Methode kann dann so geschrieben werden:

Tr.T.match(A);

Bei einer vorgegebenen Zeichenkette (hier z.B. *un*) wird man die erste direkte Schreibweise bevorzugen. Ist dagegen die Zeichenkette eine noch unbekannte Variable, so kann bei dieser Art von Suchmuster nur die zweite Schreibweise, d.h. die mit dem Konstruktor *new,* angewandt werden.

I.2.8.1 Die wichtigsten Sonderzeichen bei reguären Ausdrücken

Zeichen Beschreibung

* Sucht das vorangehende Zeichen zero or more times. For example, "zo*" Sucht either "z" or "zoo"
+ Sucht das vorangehende Zeichen one or more times. For example, "zo+" Sucht "zoo" but not "z".
? Sucht das vorangehende Zeichen zero or one time. For example, "a?ve?" Sucht the "ve" in "never".
. Sucht jedes einzelne Zeichen except a newline character.
\b Sucht eine Zeichenfolge mit Wortbegrenzung, that is, the position between a word and a space. For example, "er\b" Sucht the "er" in "never" but not the "er" in "verb".
\B Sucht eine Zeichenfolge, das keine Wortbegrenzung ist. "ea*r\B" Sucht the "ear" in "never early".
\d Sucht ein Zahlenzeichen. Equivalent to [0-9].
\D Sucht ein Zeichen, das keine Zahl ist. Equivalent to [^0-9].
\n Sucht Steuerzeichen für eine neue Zeile
\r Sucht Steuerzeichen für Wagenrücklauf
\s Sucht ein Leerzeichen etc. Equivalent to "[\f\n\r\t\v]".
\S Sucht alles außer Leerzeichen. Equivalent to "[^ \f\n\r\t\v]".
\w Sucht alphanumerische Zeichen und den Unterstrich Equivalent to "[A-Za-z0-9_]".
\W Sucht nichtalphanumerische Zeichen. Equivalent to "[^A-Za-z0-9_]".
^ Sucht den Anfang einer Zeichenkette
$ Sucht das Ende einer Zeichenkette

I.2.9 Arrays

Das Arrayobjekt ist eines der wichtigsten JavaScript-Objekte, das in den hier dargestellten Anwendungen immer wieder vorkommt. Array (engl. = Feld) bedeutet soviel wie eine Folge von gleichartigen Teilen oder Elementen. Arrays können z.B. die einzelnen Wörter eines Satzes oder Textes sein, die einzelnen Sätze eines Textes oder die einzelnen Titel einer Bibliographie oder die einzelnen Abschnitte eines Aufsatzes usw.

Ein Array wird, wie in dem oberen Match-Beispiel gezeigt, automatisch durch die Anwendung der globalen Match-Methode erzeugt, ebenso bei der Anwendung der Split-Methode auf ein Stringobjekt. Diese kommt in den hier vorgestellten Beispielen besonders häufig vor. Dabei wird jedesmal eine längere oder kürzere Zeichenfolge, ein Text, eine Liste usw. in eine Reihe von gleichartigen Teilen zerlegt. Hinter der Methodenbezeichnung

split in Klammern das Trennungselement, das können Buchstabenfolgen, Zahlen, neue Zeilen, Leerstellen sein, wie hier z.B.:

```
S="Das ist aber sehr schön";
SA=S.split (" ");
alert(SA);
```

Abb. I.5

Das Ergebnis ist hier dasselbe wie bei der Match-Methode:

Unabhängig von der Erzeugung eines Arrays durch die Match- oder Split-Methode eines Strings kann ein Array auch durch das Schlüsselwort new, den Objektnamen Array und der Angabe der einzelnen Teile in Klammern entstehen:

```
A = new Array ("aber", "das", "ist", "doch", "so");
```

Hinter der festgelegten Konstruktorbezeichnung new Array in Klammern die einzelnen Elemente des Arrays, in diesem Fall einzelne Wörter, in Anführungszeichen, weil es sich hier um Zeichenfolgen handelt. Das Ergebnis wird in der Variablen *A* gespeichert.

Eine weitere Option, ein Array zu erzeugen, besteht in einer For-Schleife (s.u.).

Die einzelnen Elemente eines Arrays können durch eine Indexnummer in eckigen Klammern angesprochen werden. Der erste Teil eines Arrays hat immer die IndexNr 0, der zweite 1, der dritte 2 usw. Die einzelnen Teile werden also durchnumeriert, beginnend mit 0. Der 4. Teil des Array A wird entsprechend mit *A[3]* bezeichnet. Soll der vierte Teil des Arrays mit der Alert-Methode angewählt und gezeigt werden, so schreibt man:

```
alert (A[3]);
```

Im Alert-Fenster erscheint *doch*.

I.2.9.1 Methoden und Eigenschaften des Arrayobjekts

length

Mit dieser Eigenschaft wird die Anzahl der Elemente eines Arrays festgestellt.

Beispiel:

```
S="das sind immer wieder die gleichen Schritte";
SA=S.split(" ");
alert(SA.length);
```

Der in Anführungszeichen angegebene Satz ist ein Stringobjekt und wird in der Variablen *S* gespeichert.
Durch die Split-Methode mit einer Leerstelle als Trennungselement wird daraus der Array

SA.
Mit alert wird das Ergebnis von *SA.length*, d.h. die Zahl der Elemente angezeigt. Diese ist in diesem Fall 7.

join ( )

Mit dieser Methode werden die Elemente eines Arrays zu einem String zusammengefügt. In der Klammer wird ein Trennungszeichen angegeben. Wird z.B. " " für eine Leerstelle angegeben, so werden die Arrayelemente, durch Leerstellen getrennt, hintereinander gereiht. Ohne Trennungszeichen werden die einzelnen Arrayelemente durch Kommata von einander getrennt.

sort ( )

Diese Methode ist eine der effektivsten Methoden von JavaScript überhaupt. Mit einer einzigen kleinen Zeile können umfangreiche, in Arrays verwandelte Datenmengen im Nu sortiert werden.

Soll mit dieser Methode eine alphabetische oder lexikalische Anordnung eines Arrays erreicht werden, so werden in der Klammer keine Werte als Parameter eingegeben. Sollen Zahlen numerisch sortiert werden, so muß eine einfache Vergleichsfunktion verwendet werden, die mit dem frei gewählten Funktionsnamen *NumSort* so definiert wird:

```
function Numsort(a,b)
{
return a-b
}
```

Der Funktionsname wird als Parameter in die Klammer übernommen.

Beispiel:

```
1) S="1,3,5,4,9,8,12,10";
2) SA=S.split(",");
3) SB=SA.sort(Numsort);
4) function Numsort(a,b)
5) {
6) return a-b
7) }
8) alert(SB);
```

Abb. I.6

Mit alert wird die numerische Reihenfolge der Zahlen angezeigt:

slice ( )

Mit dieser Methode wird ein Teil aus einem Array extrahiert. In der Klammer werden die Indexwerte des ersten und des letzten zu extrahierenden Elementes angegeben.

concat ( )

Mit dieser Methode wird ein Array an einen anderen angehängt. In der Klammer steht der Name des anzuhängenden Arrays.

I.2.10 Statements

Nach der kurzen Darstellung des Arrayobjekts kommen wir zu einer anderen Gruppe von Sprachelementen einer Programmiersprache, den sogen. Statements. Dies sind Schlüsselwörter, die wesentlich für den Programmablauf sind, ohne die kein vollwertiges Programm denkbar ist. In den vorangehenden Beispielen von kleinen linearen Programmabläufen wurde davon noch kein Gebrauch gemacht. Unter einer Reihe von Statements sind es lediglich zwei, die beiden Schlüsselwörtern for und if, die grundlegend sind für die im zweiten Teil dargestellten umfangreichen und komplizierteren Beispiele.

I.2.10.1 Die For-Schleife

Sobald ein Array durch die o.g. Methoden erzeugt worden ist, können die einzelnen Teile des Arrays mit einer For-Schleife weiter bearbeitet werden. Dadurch werden die gleichen Prozeduren für alle Teile in gleicher Weise angewandt. Daher brauchen die entsprechenden Programmanweisungen nur einmal notiert zu werden. Soll z.B. der oben genannte Satz mit großen Anfangsbuchstaben geschrieben werden, so könnte man dies mit einer For-Schleife so erreichen:

```
1) <script>
2) S="Das ist doch alles sehr schön";
3) SA=S.split (" ");
4) for (x=0;x<SA.length;x++)
5) {
6) SB=SA[x].substring(0,1);
7) SC=SB.toUpperCase();
8) SD=SA[x].replace(SB,SC);
9) document.write(SD+" ");
10) }
11) </script>
```

4) In Klammern hinter *for* wird die Indexvariable *x* zunächst belegt mit 0. *x<SA.length* gibt die maximale Größe von *x* an, die kleiner sein muß als die Gesamtanzahl *SA.length* und schließlich *x*++, alles durch Semikolons voneinander getrennt. x=0 bedeutet, die For-Schleife beginnt mit dem Indexwert 0, d.h. dem ersten Teil des Arrays. *SA.length* ist die Anzahl der Elemente des Arrays *S. Lenght* ist eine Methode zur Bestimmung der Länge sowohl eines Strings als auch eines Arrays. *x<SA.length* gibt die maximal zu durchlaufende Anzahl der Durchläufe an, die hier kleiner sein soll als die Anzahl der Elemente des Arrays. Wenn der Satz 6 Wörter hat, wie hier, ist *SA.length* 6. Es finden also 6 Durchläufe statt. *x*++ ist eine Schreibweise dafür, daß der Indexwert *x* jedes mal um eine 1 erhöht wird, ein Inkrement, das Gegenteil davon wäre das Dekrement *x*--.
5) Nach der Klammer kein Semikolon, sondern eine öffnende geschweifte Klammer und dann die einzelnen Anweisungen, der Anweisungsblock.
6) *SB=SA[x].substring(0,1)*; aus dem Array-Element mit der IndexNr x wird das erste Zeichen mit der Substring-Methode herausgelesen und in *SB* gespeichert.

7) *SC=SB.toUpperCase();* das Stringobjekt *SB*, also das erste Zeichen wird mit *toUpperCase()* in einen Großbuchstaben verwandelt, falls kleiner Buchstabe. Ergebnis gespeichert in *SC*.
8) *SD=SA[x].replace(SB,SC);* in dem Arrayelement wird das erste Zeichen *SB* durch *SC* ersetzt und in *SD* gespeichert.
9) *document.write(SD+" ")*; ausgeschrieben wird *SD* gefolgt von einer Leerstelle. Nacheinander wird jedes einzelne Wort mit großem Anfangsbuchstaben und folgendem Zwischenraum ausgeschrieben. Ohne diese Leerstelle würden die Wörter aneinandergereiht ohne Zwischenraum geschrieben.

In einer For-Schleife wird eine Variable jeweils mit einem anderen Wert belegt. In manchen Anwendungen werden diese veränderten Variablen in einem neuen Array gespeichert und danach in der Fortsetzung des Programms abgerufen wird. Das kann durch folgenden Code in diesem Beispiel erreicht werden:

```
SA=S.split(">"); // eine Wörterliste als Array
SN=""; // Variable Anfangswert als Leerwert
for(x=0;x<SA.length;x++)
{
s=SA[x].match(/[A-Z][a-z]+/g); // aus den einzelnen Elementen werden mit
Grossbuchstaben beginnende Wörter ermittelt
SN=SN+">"+s; // nacheinander wird das ermittelte Wort s der Zeichenkette mit dem
Zeichen > hinzugefügt
}
SNA=SN.split() das Array SNA wird gebildet
```

Beim ersten Durchlauf der Schleife ist *SN=0*, d.h.es ist *SN=0+">"+s*, beim zweiten Duchlauf ist *SN=s,* d.h. es ist *SN=s+">"+s* usw. Schliesslich enthält SN die gesamte Zeichenkette der einzelnen veränderten Variabllen *s*. Dies ist noch kein Array, sondern ein String, der aber durch das Trennungsmerkmal > leicht in ein Array SNA verwandelt werden kann.

Eine andere umständlichere Möglichkeit kann darin bestehent, die einzelnen ermittelten Werte in ein Eingabefeld abzulegen durch den Code:

```
a=document.forma.area.value;
document.forma.area.value=a+">"+s;
```

Bei jedem Durchlauf wird der enthaltene Wert des Feldes zunächst ausgelesen und dann dieser Wert dem bereits vorhandenen Wert hinzugefügt. Nach der For-Schleife befinden sich in dem Feld *area* alle ermittelten s-Werte, gerennt durch >

I.2.10.2 Die If-Bedingung

Das Ausführen von bestimmten Programmanweisungen wird sehr oft von bestimmten Bedingungen abhängig gemacht. Sollen z.B. nur die Buchstaben a und i im vorigen Programm groß geschrieben werden, so wird diese Bedingung eingeleitet mit if, gefolgt von dem Gegenstand der Bedingung in Klammern, hier also.:

```
if(SB=="a"||SB=="i")
SC=SB.toUpperCase();
```

Beachten Sie == ist ein Gleichsetzungszeichen, = ist ein Zuweisungszeichen. || bedeutet oder. Hinter der schließenden Klammer folgt kein Semikolon. Wenn nur eine Anweisung

folgt, können die geschweiften Klammern, die bei mehreren Anweisungen stehen müssen, wegfallen.
7) und 8) Falls das erste Zeichen *SB* ein *A* oder I ist, soll dieses in Großbuchstaben geschrieben werden.
10) Im anderen Fall, ausgedrückt durch *else,* soll der Buchstabe nicht verändert werden, hier wird 11) *SB* unverändert zu *SC*, was man so schreibt: *SC=SB*. (Auch hier keine geschweifte Klammer, weil nur eine Anweisung.)

Das ganze Programm würde demnach lauten:

```
1) <script>
2) S="Das ist aber sehr schön";
3) SA=S.split (" ");
4) alert(SA);
5) alert(SA[3]);
6) for (x=0;x<SA.length;x++)
7) {
8) SB=SA[x].substring(0,1);
9) if(SB=="a"||SB=="i")
10) SC=SB.toUpperCase(); /*Zeichenkette SB wird in Großbuchstaben geschrieben und in SC gespeichert*/
11) else
12) SC=SB;
13) SD=SA[x].replace(SB,SC);
14) document.write(SD+" ");
15) }
16) </script>
```

Abb. I.7

Das ausgeschriebene Ergebnis:

i*f(SB=="a"||SB=="i") falls SB gleich a oder falls SB gleich i ist (|| = Oder-Operator)*

I.2.11 Operatoren

Nach Objekten, Methoden und Statements bilden die Operatoren eine weitere wichtige Gruppe jeder Programmiersprache. In den vorangehenden Beispielen wurden schon einige der wichtigsten Operatoren verwendet. Während, wie schon oben angedeutet, das einfache Gleichheitszeichen als Zuweisungsoperator dient, der am häufigsten in einem Skript verwendete Operator, wobei ein Wert einer Variablenbezeichnung zugewiesen wird, ist das doppelte Gleichheitszeichen einer der sogen. Vergleichsoperatoren. Weitere Vergleichsoperatoren sind != für ungleich, > für größer und < für kleiner. Der Oder-Operator gehört zu den logischen Operatoren, neben dem Und-Operator && und dem Nichtoperator ! Logische und Zuweisungsoperatoren finden z.B. in If-Bedingungen häufig

Verwendung. Außerdem gibt es noch die arithmetischen Operatoren +, -, * und ++ als Inkrementierung und -- als Dekrementierung.

I.3. Zusammenfassung der bisher besprochenen JavaScript-Elemente

Allgemeine Schreibweise

- JavaScript kann an beliebiger Stelle innerhalb eines HTML-Doluments stehen.
- Einleitung mit <script language=javascript> oder auch nur mit <script>, Ende mit </script>
- Jede Anweisung mit Semikolon abschließen, z.B. document.write(v);
- Daneben Verwendung von Kommata, Punkten und Ausrufungszeichen.
- Runde Klammern und eckige Klammern. Runde Klammern bei If-Bedingungen, For-Schleifen, bei vielen Stringobjekten. Eckige Klammern zur Indexangabe bei Arrays.
- Geschweifte Klammern für mehrere zusammengehörige Befehlszeilen,
- Jeder einleitenden Klammer muß eine abschließende Klammer entsprechen.
- Funktionen werden mit geschweiften Klammern eingeleitet und beendet.

Variablen

Frei gewählte Benennung für einen zu speichernden Wert (Zeichenfolge, Zahl, bestimmtes Frame oder Fenster, Formularfeld). Der unter einer Variablenbenennung gespeicherte Wert wird sehr oft zu einem Objekt, an dem Methoden, Funktionen, wertverändernde Aktionen ausgeführt werden. Die Variablenbenennung wird so zu einer gleichlautenden Objektbenennung, vor allem bei Stringobjekten, z.B.

```
v="das ist aber schön";
va=v.replace("schön", schlecht);
```

Eine Variable wird auch oft einer Methode als auszuführenden Wert zugewiesen, z.B.:

```
V= "das ist aber schön";
document.write(V);
```

Neben den frei gewählten Objektbenennungen wie für diese Stringobjekte gibt es vordefinierte Objekte in einer Stufenfolge von dem obersten Objekt window, zu den Objekten *document, frame, forms, elements* usw.

Methoden sind von wenigen Ausnahmen abgesehen an Objekte gebunden. Es gibt Methoden, die an ganz bestimmte Objekte gebunden sind, wie z.B. die Methode *write* an das Objekt *document*. Methoden sind immer vordefiniert. Eigenschaften sind ähnlich wie Methoden oft an bestimmte Objekte gebunden. Siekönnen aber auch frei definiert werden. Im Unterschied zu Methoden werden sie ohne folgende Klammer notiert.

Statements

If-Bedingung: nach if folgt in runden Klammern der Inhalt dieser Bedingung, z.B.: if(x<10), d.h. falls x kleiner als 10 ist, d.h. nur für Werte weniger als 10. For-Schleife: nach for folgen in runden Klammern die einer Variablen (z.B. x) zugewiesenen Indexanfangs- und Indexendwerte (eines Arrays) der zu durchlaufenden Schleife sowie der Zählfaktor , z.B.

x++, wenn jeweils der Indexwert um eins erhöht wird. Dazwischen steht jeweils ein Semikolon: for(x=0; x=A.length; x++)

Operatoren

- = Zuweisung
- == Gleichheitsoperator
- || oder
- > größer
- < kleiner
- && und
- != nicht

Eventhandler (innerhalb des HTML-Codes). Auswahl

- onclick beim Anklicken eines Schalters oder eines Textes
- onload beim Laden der Seite
- onMouseOver wenn die Maus auf einen bestimmten Bereich einer Seite befindet
- onKeyPress wenn eine Taste gedrückt wird

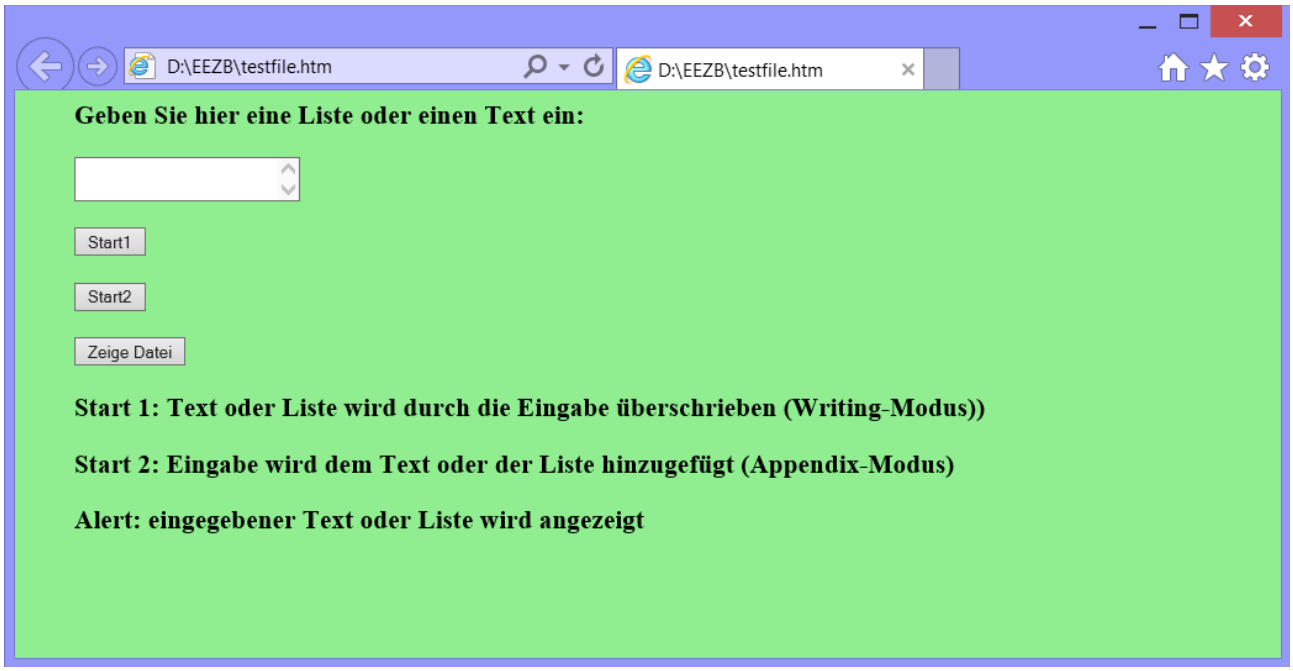
I.4 Anwendungen mit ActiveX

ActiveX stellt eine eigene umfassende Objekthierarchie dar. Es beinhaltet zahlreiche Objekte mit unterschiedlichen Funktionalitäten, von denen einige in der *Scripting Run-Time Library* zusammengefaßt werden. Dazu zählt das *FileSystemObject*, das für den direkten Zugriff auf eine beliebige vorhandene Datei zuständig ist, wodurch Daten ausgelesen, hinzugefügt oder verändert werden können. Und dieses soll uns allein in diesem Zusammenhang interessieren. Eine darüber hinaus gehende Darstellung des *ActiveX-Objekts* würde hier zu weit führen. Es ist sehr wichtig hierbei darauf hinzuweisen, daß die Anwendung eines der zur ActiveX-Technologie gehörenden Objekte, wie des *FileSystemObjects*, nur in bezug auf die im eigenen Computer gespeicherten Dateien möglich ist. Mit diesem Objekt geschriebene Programme sind über das Internet nicht ausführbar.

Der Button Start 1 ist durch den Eventhandler *onclick* mit der Funktion *datei1()* verknüpft, der Button *Start 2* mit der Funktion *datei2().* In diesem Beispiel wird die OpenTextFile-Methode des FileSystemObjects in 3 verschiedenen Modi angewandt: *ForReading* (Lesen), *ForAppending* (Hinzufügen) und *ForWriting* (Schreiben), mit den Nummern 1, 2 und 8 bezeichnet. In den jeweils folgenden Zeilen wird das FileSystemObject mit dem Constructor *new* als Teil des ActiveXObjects erzeugt und in der Variablen *fso* gespeichert. jeweils folgenden Zeilen wird die Objektvariable *fso* mit der Methode *OpenTextFile* verbunden, was wie üblich durch einen einfachen Punkt geschieht. In Klammern dahinter folgt zunächst in Anführungszeichen der Name der Datei, auf die der Zugriff erfolgen soll. Normalerweise muß hier der gesamte Pfad angegeben werden, also Laufwerk, Verzeichnis, Unterverzeichnis, Dateiname. In diesem Beispiel lautet der Pfadname *D:/EEZB/file1.txt*, d.h. die Datei *file1.txt* befindet sich im Laufwerk *D* und dem Ordner *EEZB.* In der Klammer steht hinter einem Komma der auszuführende Modus. Dieser kann entweder mit den numerischen Werten 1, 2 und 8 bezeichnet werden oder mit den Benennungen *ForReading, ForWriting* und *ForAppending*. Bei Start1 und Start2 werden zunächst die in das Formularfeld *area* eingeschriebenen oder eingefügten Daten in der

Variablen *E* gespeichert. Diese Daten können ein längerer Text oder eine längere Liste sein. Der Wert *E* stellt eine durchgehende Zeichenkette und damit ein Stringobjekt dar und kann als solches beliebig verändert werden, in diesem Beispiel durch Replaceanweisungen, wodurch Zeilenumbrüche durch das Zeichen > ersetzt werden. Der veränderte Wert *Eaa* wird über den Schalter1 durch die Anweisung *f.Write(Eaa);* und den *ForWriting Modus* in die Textdatei fest eingeschrieben. In Start2 wird durch *f.Write(" "+Eaa);* und den *ForAppendix-Modus* der Wert zum bereits vorhanden Inhalt hinzugefügt und gespeichert.

Über den Zeige-Button wird die neue oder ergänzte Textdatei angezeigt Hierbei wird mit dem *ForReading-Modus* die Textdatei aufgerufen und das Ergebnis in *f1* gespeichert. In der folgenden Zeile wird auf diese Objektvariable die Methode *ReadAll()* angewandt.und das Ergebnis in *A* gespeichert. Das Stringobjekt *A* wird nun durch die Split-Methode in ein Array *Aa* verändert und dieses in einer For-Schleife ausgeschrieben.



Abb, I.9

Die Abbildung zeigt eine einfache Anwendung des FileSystemObjects

Dies ist der vollständige Code:

```
<html>
<head>
</head>
<body style="background-color: lightgreen;">
<form style="margin-left: 40px;" name="forma">
<p><strong></strong></p>
<p><strong>Geben Sie hier eine Liste oder einen Text ein:</strong></p>
<textarea name="area"></textarea>
<p><strong><input value="Start1"
onclick="datei1()" type="button"> </strong></p>
<p><strong><input value="Start2"
onclick="datei2()" type="button"> </strong></p>
<input value="Zeige Datei" onclick="zeige()"
type="button">
```

```
<p><strong>Start 1: der enthaltene Text wird durch den
eingegebenen Text &uuml;berschrieben (durch Writing-Modus))</strong></p>
<p><strong>Start 2: zu dem enthaltenen Text wird der
eingegebene Text hinzugef&uuml;gt (durch Appendix-Modus)</strong></p>
<p><strong>In der Alertanzeige wird der eingegebene Text
angezeigt</strong></p>
</form>
<script>
function datei1()
{
ForReading = 1, ForWriting = 2;
fso = new ActiveXObject("Scripting.FileSystemObject");
f = fso.OpenTextFile("D:/EEZB/file1.txt", 2);
E=document.forma.area.value; //
alert(E);
Ea=E.replace(/\n/g,">");
Eaa=Ea.replace(/>>+/g,">");
alert(Eaa);
f.Write(Eaa);
window.location.reload();
}f
unction datei2()
{
ForReading = 1, ForWriting = 2 ; ForAppending = 8 ;
fso = new ActiveXObject("Scripting.FileSystemObject");
f = fso.OpenTextFile("D:/EEZB/file1.txt", 8); E=document.forma.area.value;
alert(E);
Ea=E.replace(/\n/g,">");
alert(Ea);
Eaa=Ea.replace(/>>+/g,">");
f.Write(" "+Eaa);
window.location.reload();
}f
unction zeige()
{
ForReading = 1, ForWriting = 2 ; ForAppending = 8 ;
fso = new ActiveXObject("Scripting.FileSystemObject");
f = fso.OpenTextFile("D:/EEZB/file1.txt", 1);
A=f.ReadAll();
Aa=A.split(">");
for(x=0;x<Aa.length;x++)
document.write("<br>"+Aa[x]);
window.location.reload();
}
</script>
</body>
</html>
```

In diesem einfachen Beispiel werden Daten in ein Eingabefeld eingegeben, die Daten werden bearbeitet und in einer Datei gespeichert. Das ist im Prinzip dieselbe Vorgehensweise wie bei den in Kap.V dargestellten Multiplen Linksystem und Einarbeitungssystem. Im ersten werden Aufsatznachweise durch Copy und Paste in ein Eingabefeld eingefügt, im zweiten Zeitschriftentitel aus EZB-Listen übernommen. In einer aufwendigen Bearbeitung werden die übernommenen Nachweise oder Titel bearbeitet und mit anderen, aus anderen Quellen übernommenen Daten verknüpft und das Ergebnis in einer dynamisch ergänzten Datei gespeichert.