24

€ 4,-

KnowWare PLUS

Was man so alles machen kann !

72 Seiten

JavaScript

für Fortgeschrittene

www.KnowWare.de

Martin Baier

Deutschland: 4,- EUR Österreich: 4,60 EUR
Schweiz: 8 SFR Luxemburg: 4,70 EUR Italien: 5,50 EUR

Acrobat Reader: Wie ...

F5/F6 öffnet/schließt die Ansicht **Lesezeichen**

Strg+F sucht

Im Menü Ansicht stellst du ein, wie die Datei gezeigt wird

STRG+0 = Ganze Seite **STRG+1** = Originalgrösse **STRG+2** = Fensterbreite

Im selben Menü kannst du folgendes einstellen:: **Einzelne Seite**, **Fortlaufend** oder **Fortlaufend - Doppelseiten** .. Probiere es aus, um die Unterschiede zu sehen.

Navigation

Pfeil Links/Rechts: eine Seite vor/zurück

Alt+ Pfeil Links/Rechts: Wie im Browser: Vorwärts/Zurück

Strg+**+** vergrößert und **Strg**+**–** verkleinert

Bestellung und Vertrieb für den Buchhandel

Bonner Pressevertrieb, Postfach 3920, D-49029 Osnabrück
Tel.: +49 (0)541 33145-20
Fax: +49 (0)541 33145-33
bestellung@knowware.de
www.knowware.de/bestellen

Autoren gesucht

Der KnowWare-Verlag sucht ständig neue Autoren. Hast du ein Thema, daß dir unter den Fingern brennt? - ein Thema, das du anderen Leuten leicht verständlich erklären kannst?

Schicke uns einfach ein paar Beispielseiten und ein vorläufiges Inhaltsverzeichnis an folgende Adresse:

lektorat@knowware.de

Wir werden uns deinen Vorschlag ansehen und dir so schnell wie möglich eine Antwort senden.

www.knowware.de

Vorwort

Nach dem durchschlagenden Erfolg von *JavaScript für Einsteiger*“ geht es jetzt in die zweite Runde: Willkommen bei *JavaScript für Fortgeschrittene*!

Nachdem du im ersten Heft die Grundlagen von JavaScript erlernt, mit Variablen, Funktionen und Schleifen gearbeitet und Anwendungen wie das Quiz oder den Euro-Rechner programmiert hast, geht es jetzt weiter:

Du lernst in diesem Heft beispielsweise, wie du Musik mit JavaScript steuerst. Auch ferngesteuerte Fenster, Cookies oder das Auslesen von Tastatureingaben stehen auf dem Programm. Zum Abschluss zeige ich dir anhand eines umfangreichen Spieles, wie du JavaScript und CSS zu DHTML verbindest und damit tolle Effekte zaubern kann!

Soviel? Schaffe ich das überhaupt? Natürlich schaffst du es! Denn das Wissen eignest du dir nicht trocken und theoretisch an, sondern bekommst es von mir häppchenweise in Form von praktischen Anwendungen präsentiert.

Um das Heft problemlos durcharbeiten zu können, benötigst du allerdings schon ein bisschen Vorwissen.

Um optimal vorbereitet zu sein, empfehle ich dir mein Heft *JavaScript für Einsteiger*, erschienen im Knowware-Verlag als Nummer PLUS 6. Du kannst es entweder an einem Kiosk in deiner Nähe kaufen oder über **www.knowware.de** bestellen.

Hast du anderweitig gute Vorkenntnisse in JavaScript erworben, so sollte dir dieses Heft aber auch keinerlei Probleme bereiten.

Wie schon beim Heft *JavaScript für Einsteiger* stehen auch alle Anwendungen aus diesem Heft zum Download als zip-Datei sowie zum Ausprobieren auf der Homepage des Verlages zur Verfügung:

www.knowware.de/javascript2

Auch die Lösungen zu den Übungsaufgaben findest du hier.

Bitte teste zuerst auf dieser Seite, ob ein Programm wirklich nicht funktioniert, bevor du mich mit Fragen bombardierst!

Als geübter JavaScript-Programmierer hast du bestimmt schon deinen bevorzugten Browser und Editor für die HTML-Dateien gefunden. Deshalb lediglich zu deiner Information: Die Screenshots in diesem Heft sind zeigen die Anwendungen mit dem Browser Netscape 6.2. Als Editor verwende ich den Standard-Editor von Windows.

Mit deinem Editor und Browser sollten alle Beispielprogramme aber genauso funktionieren.

Allerdings funktionieren leider noch nicht alle Anwendungen mit dem neuen Netscape 6. Deshalb weiche ich an manchen Stellen auf einen anderen Browser aus – ich weise dich dann auch darauf hin.

Schließlich noch eine Bitte: Schreib doch alles, was dir an diesem Heft gefallen hat, aber auch alles, was ich deiner Meinung nach verbessern könnte, in eine Mail und schick sie mir! Nur durch dieses Feedback kann ich meine Hefte so gestalten, wie du als Leser sie gerne hättest!

Auch Internet-Adressen, auf denen du dein JavaScript-Wissen anwendest, kannst du mir gerne senden, dann kann ich mal einen Blick auf den Erfolg des Heftes werfen.

Nun ist aber genug geredet, ich wünsche dir viel Erfolg und vor allem viel Spaß beim Programmieren mit JavaScript!

Erlangen, im Juli 2002

Martin Baier (martin.baier@gmx.net)

FensterIn

Wer kennt sie nicht, die kleinen (und oft leider auch größeren) Werbefenster, die sich ganz von alleine öffnen, wenn eine Seite im Browser aufgerufen wird? Dahinter steckt JavaScript. Wie auch du solcher Fenster Herr wirst, und wie du diese Fenster sinnvoll und nicht nur für Werbung einsetzt, erfährst Du in unserem ersten Kapitel.

Fenster öffnen

Weil das das einfachste Beispiel ist, fangen wir mit dem berühmt-berüchtigten Werbefenster an. Hier der Code für die HTML-Datei FENSTER1.HTML.

```
<html>
<head>
<title>Fenster öffnen</title>
<script language="JavaScript">
<!--
function open_window(){
var myWindow=window.open("werbung.html", "Werbefenster")
}
//-->
</script>
</head>
<body onLoad="open_window()">
Hier ist der Inhalt.<p>
Aber die Werbung dominiert.
</body>
</html>
```

Eine Datei namens WERBUNG.HTML muss natürlich auch vorhanden sein. Sie soll lediglich eine Werbegrafik enthalten. Dies sieht beispielsweise so aus:

```
<html>
<head>
<title>Werbung</title>
</head>
<body>
<img src="werbung.gif">
</body>
</html>
```

Willst du die Dateien nicht selber erstellen, lädst du sie unter **www.knowware.de/javascript2** aus dem Internet.

Und so sieht das ganze dann im Browser aus:

Was ist passiert? Du hast einfach eine Variable **myWindow** – der Name ist ohne Bedeutung – deklariert, der du mit **window.open("werbung.html", "Werbefenster")** eine Instanz des Objektes **window** zugewiesen hast. Über die Variable kannst du auf das Fenster zugreifen, es z.B. wieder schließen; mehr dazu später.

Fensterparameter

Bist du nun zufrieden mit deinem Werbefenster? Natürlich nicht! Und warum nicht? Na, weil ein Werbefenster keine Menüleisten enthalten soll, und außerdem soll es nur so groß sein wie das Werbebanner selbst!

Kein Problem mit JavaScript! Bis jetzt hast du an **window.open** nur die zwei Parameter übergeben, die obligatorisch sind: Der erste ist der Dateiname, der zweite ist ein Fenstername. Du kannst nun einen dritten Parameter hinzufügen, der die Eigenschaften des Fensters bestimmt. Die einzelnen Optionen trennst du durch Kommata.

Es stehen etwa zwanzig Eigenschaften zur Verfügung, hier nur die wichtigsten:

width=(Pixel)	Gibt die Breite des neuen Fensters an.
height=(Pixel)	Gibt die Höhe des neuen Fensters an.
resizable=yes/no	Gibt an, ob Breite und Höhe des Fensters verändert werden können.
menubar=yes/no	Gibt an, ob eine Menüleiste im neuen Fenster vorhanden sein soll.
toolbar=yes/no	Gibt an, ob eine Buttonleiste im neuen Fenster vorhanden sein soll.
location=yes/no	Gibt an, ob im neuen Fenster eine Adresszeile vorhanden sein soll.
status=yes/no	Gibt an, ob im neuen Fenster eine Statuszeile vorhanden sein soll.
scrollbars=yes/no	Gibt an, ob im neuen Fenster Bildlaufleisten vorhanden sein sollen.
dependent=yes/no	Gibt an, ob das neue Fenster vom aktuellen Fenster abhängig sein soll. Ist dem so, so wird das neue Fenster geschlossen, sobald das momentane Fenster geschlossen wird. Unter Windows wird solch ein abhängiges Fenster nicht in der Taskleiste angezeigt.

Willst du nun das Werbefenster optimieren, ersetzt du einfach die bisherige Zeile zum Öffnen des neuen Fensters durch folgende Zeile:

```
var
myWindow=window.open("werbung.html",
"Werbefenster", "width=300,height=60")
```

Schon sieht das Werbefenster viel besser aus:

Sobald du einen Parameter setzt (bei uns etwa **width**), werden alle anderen Eigenschaften (Menüleiste, Bildlaufleisten usw.) deaktiviert! Du musst sie dann manuell wieder hinzufügen. Willst du also in deinem Werbefenster eine Menüleiste haben, schreibst du folgende Zeile:

```
var
myWindow=window.open("werbung.html",
"Werbefenster","width=300,height=60,me
nubar=yes")
```

Es funktioniert:

So, nun weißt Du, wie die Sache funktioniert – jetzt bist du an der Reihe:

- Unser Werbefenster soll sich selbständig schließen, wenn das Hauptfenster geschlossen wird. Öffne ein abhängiges Fenster!
- Die Größe des Werbefensters soll veränderbar sein!
- Sorge dafür, dass neben der Menüleiste auch die Buttonleiste vorhanden ist!

Um auch mit den restlichen, hier nicht angegebenen Parametern zu programmieren, empfehle ich dir die (englischsprachige) Dokumentation von Netscape. Du findest sie hier:
http://developer.netscape.com/docs/manuals/javascript.html

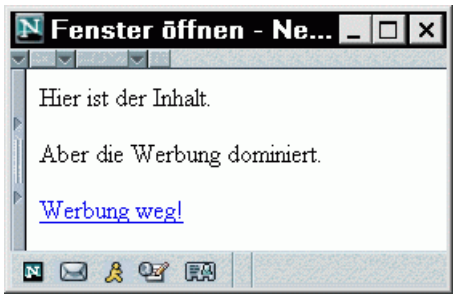
Fenster schließen

Wir möchten jetzt unserem Surfer eine Möglichkeit anbieten, das Werbefenster zu schließen. Und zwar soll dieser Vorgang von unserem Hauptfenster aus möglich sein!

Füge im Code des Hauptfensters einen Link ein:

```
...Aber die Werbung dominiert.<p>
<a href="javascript:myWindow.close()">
Werbung weg!</a>
</body>...
```

Jetzt weißt du, warum wir vorher zum Öffnen des Fensters eine Variable gebraucht haben! Dieser Variablen hatten wir eine Instanz von **`window`** zugewiesen, wir können also auf das Werbefenster zugreifen. Mit **`myWindow.close()`** wird also das Werbefenster geschlossen.



Geht nicht? Stimmt! Hast du aber *JavaScript für Einsteiger* gründlich gelesen, solltest du eigentlich herausfinden können, warum.

Schau dir mal an, wo die Variable **`myWindow`** deklariert wurde! Das war in der Funktion **`open_window`**! Es war also eine lokale Variable. Nachdem das Fenster geöffnet worden ist, wurde die Funktion beendet, und die Variable war nicht mehr gültig. Wollen wir aber später wieder auf das Fenster zugreifen, muss die Variable global sein. Du weißt schon, wie man das löst. Der JavaScript-Teil muss folgendermaßen aussehen – die Variable wird außerhalb der Funktion deklariert:

```
var myWindow
function open_window(){
myWindow=window.open("werbung.html",
"Werbefenster","width=300,height=60")
}
```

Nun solltest Du die Werbung ohne Probleme schließen können.

Beim ersten Mausklick auf den Verweis wird das Werbefenster ordnungsgemäß geschlossen. Klickst du allerdings ein zweites Mal darauf, gibt es einen JavaScript-Fehler. Das liegt daran, dass das Fenster ja schon geschlossen ist und demnach nicht mehr geschlossen werden kann. Die Lösung ist einfach. Ändere den Link folgendermaßen:

```
<a
href="javascript:if(!myWindow.closed)
{myWindow.close()}">Werbung weg!</a>
```

So wird das Fenster nur geschlossen, wenn (**`if`**) es noch nicht (**`!`**) geschlossen ist (**`.closed`**).

An dieser Stelle wieder eine kleine Aufgabe: Ist das Werbefenster schon geschlossen, soll diese Meldung ausgegeben werden: „Die Werbung ist doch schon weg!“ Verwende einfach die **`else`**-Anweisung.

Dynamische Fenster

Andere benutzen für solche Dinge serverseitige Skriptsprachen wie PHP oder ASP, wir schaffen es auch mit JavaScript: Den Inhalt eines Fensters dynamisch zu erzeugen. Bis jetzt haben wir nur unser immer gleich bleibendes Werbefenster geöffnet. Jetzt wollen wir dem Surfer ein Formular anbieten, in das er sein Geburtsdatum und seinen Namen einträgt. Je nach den Eingaben in diesem Formular wird dann in einem neuen Fenster sein persönliches Horoskop angezeigt.

Das HTML-Gerüst

Das Eingabeformular soll so aussehen:

Der zugehörige HTML-Code, noch ohne die JavaScript-Funktionen, sieht aus wie folgt:

```
<html>
<head>
<title>Dein Horoskop</title>
<script language="JavaScript">
<!--
...
//-->
</script>
</head>
<body>
<form>
Dein Name: <input name="name"><br>
Dein Geburtsdatum (Tag, Monat):
<input name="tag" size=2 maxlength=2>
<select name="monat">
<option value="Steinbock">Januar
<option value="Wassermann">Februar
<option value="Fisch">März
<option value="Widder">April
<option value="Stier">Mai
<option value="Zwilling">Juni
<option value="Krebs">Juli
<option value="Löwe">August
<option value="Jungfrau">September
<option value="Waage">Oktober
<option value="Skorpion">November
<option value="Schütze">Dezember
</select><p>
<input type="button" value="Horoskop
     erstellen"
onClick="javascript:horoskop()">
</form>
</body>
</html>
```

Das Formular ist selbsterklärend. Bei den Monaten wird als Wert immer das Sternzeichen für die erste Monatshälfte mit angegeben. Wie trotzdem jedem Geburtsdatum das korrekte Sternzeichen zugeordnet wird, sehen wir später.

Beim einem Klick auf die „Horoskop erstellen“-Schaltfläche wird nun lediglich die Funktion **`horoskop`** aufgerufen.

Das Schicksal

`schicksal` ist eine Funktion, die wir benötigen, um das Horoskop zu erstellen. Sie gibt zufällig „Glück“ oder „Pech“ zurück. Das ist der Code:

```
function schicksal(){
if (Math.random()>0.5){
return("Gl&uuml;ck")
}
else
{
return("Pech")
}
}
```

Hier hast du den Befehl **`return`** kennen gelernt. Bisher haben wir Funktionen nur so aufgerufen:

```
funktion()
```

Die Funktion hat also irgendeine Aufgabe erfüllt, konnte aber nichts zurückgeben. Jetzt können wir schreiben:

```
zufall=schicksal()
```

`zufall` enthält dann entweder „Glück“ oder „Pech“. **`return`** wird also in Funktionen eingebaut, um einen Wert zurück zu geben! Du kannst solche Funktionen auch direkt in andere Anweisungen mit einbauen:

```
alert("Du hast "+schicksal())
```

Hier wird zufällig die Meldung „Du hast Glück“ oder „Du hast Pech“ ausgegeben. Füge die Funktion **`schicksal`** in den JavaScript-Teil der Horoskop-Datei ein!

Das Horoskop

Jetzt zur eigentlichen Horoskop-Funktion:

```
function horoskop(){
var nr=document.forms[0].monat.selectedIndex
if (document.forms[0].tag.value>20){
nr++
}
if (nr==12){
nr=0
}
var horoskopWindow=window.open("", "Horoskop", "width=400,height=200")
horoskopWindow.document.writeln('<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">')
horoskopWindow.document.writeln("<html><head><title>")
horoskopWindow.document.writeln(document.forms[0].name.value+"s Horoskop")
horoskopWindow.document.writeln("</title></head>")
horoskopWindow.document.writeln("<body>")
horoskopWindow.document.writeln("Hallo, "+document.forms[0].name.value+"!<p>")
horoskopWindow.document.writeln("Dein Sternzeichen ist
"+document.forms[0].monat.options[nr].value+"!<p>")
horoskopWindow.document.writeln("Im n&auml;chsten Monat hast du "+schicksal()+"
in der Liebe")
horoskopWindow.document.writeln("und "+schicksal()+" im Beruf!")
horoskopWindow.document.close()
}
```

Diese Funktion bedarf wohl einiger Erklärungen:

Zunächst wird in der neu initialisierten Variablen **nr** die Kennzahl für den Monat gespeichert. Januar war der erste Eintrag in der Liste, dies entspricht der Zahl 0; Februar hat die Kennzahl 1, März hat 2, und so weiter. Ist nun der Tag größer als 20, soll der Einfachheit halber das Sternzeichen des nächsten Monats genommen werden. Die Variable **nr** wird um 1 erhöht. Für den Spezialfall „Ende Dezember“ muss die Januar-Nummer gesetzt werden.

Nun wird ein Fenster geöffnet, wie du es schon kennst. Allerdings wird keine HTML-Datei angegeben, da wir den Code ja selbst erzeugen wollen.

Mit **horoskopWindow.document.writeln** können wir nun Text (mit anschließendem Zeilenumbruch) in das neue Fenster schreiben. Es handelt sich dabei um ganz normalen HTML-Code. Lediglich Name (zwei mal), Sternzeichen und der Zufall (zwei mal) werden dynamisch generiert und eingefügt.

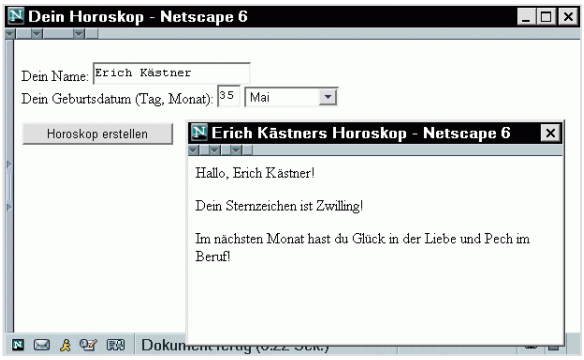
> Willst du Text ohne Zeilenumbruch in eine Datei schreiben, verwendest Du statt **document.writeln** einfach **document.write**.

Mit **horoskopWindow.document.close()** musst du das Dokument dann noch schließen. Du teilst so dem Browser mit, dass du mit dem Schreiben fertig bist.

Achte auf den Unterschied zwischen **document.close()** und **close()**. Im ersten Fall wird nur das Schreiben abgeschlossen, im zweiten Fall schließt du das Fenster!

Füge auch diese Funktion in den JavaScript-Teil der Horoskop-Datei ein. Nun ist die Datei komplett, Du kannst Dein Horoskop erstellen lassen.

So hätte vielleicht das Horoskop von Erich Kästner ausgesehen:



Jetzt bist du dran! Erstens natürlich, um dir die Zukunft vorhersagen zu lassen. Vor allem aber, um das Programm in folgenden Punkten zu verbessern:

- Es soll eine Meldung ausgegeben werden, falls kein Name angegeben wurde. Das Horoskop soll nur angezeigt werden, wenn ein Name angegeben wurde. Benutze eine **`if`**/**`else`**-Konstruktion in der Funktion **`horoskop`**.
- Ist das Horoskopfenster bereits geöffnet, soll es nicht möglich sein, ein neues Horoskop erstellen und damit das alte zu überschreiben zu lassen. Verwende eine **`if`**-Konstruktion und benutze **`horoskopWindow.closed`** in der Bedingung. Hierbei werden sich dir einige Probleme in den Weg stellen: Beachte zum Beispiel, dass **`horoskopWindow`** bis jetzt eine lokale Variable war. Des weiteren ist **`horoskopWindow.closed`** nicht definiert, bevor das Fenster zum ersten Mal geöffnet wurde. Hier musst du über eine Hilfsvariable gehen, die sich „merkt“, ob schon einmal ein Horoskop angezeigt wurde. **`horoskopWindow.closed`** darf nur abgefragt werden, wenn dies bereits der Fall war.
- Baue im Horoskop-Fenster selbst einen Link ein, der das Horoskop-Fenster schließt. So lautet der Verweis: **`javascript:close()`**. Achte darauf, dass Du zwei verschiedene Anführungszeichenpaare verwendest (`'`/`"`), da du zwei Funktionen verschachteln musst.

Wie du siehst, haben wir hier eine Seite komplett dynamisch erzeugt – es existiert für sie keine eigene HTML-Datei. Sogar der Seitentitel wurde dynamisch erstellt, was z.B. nicht möglich wäre, wenn das erstellte Fenster die Daten aus dem ursprünglichen Fenster abgefragt hätte.

Moment... Geht das auch? Daten von anderen Fenstern abfragen? Kommunikation zwischen Fenstern?

Kommunikation zwischen Fenstern

Fenster öffnen, Fenster dynamisch erzeugen, Fenster schließen, was will man mehr? Zum Beispiel folgendes: Du hast eine HTML-Seite, auf der lediglich ein Bild dargestellt werden soll. Ohne hässliche Formulare auf der Seite selber möchtest du es dem Benutzer erlauben, das Bild sowie den Text in der Statuszeile selbst zu wählen. Die Eingabe der Daten soll in einem zweiten Fenster erfolgen. Hier die Seite, zunächst mit dem Knowware-Banner als Bild:

Der Quelltext lautet folgendermaßen:

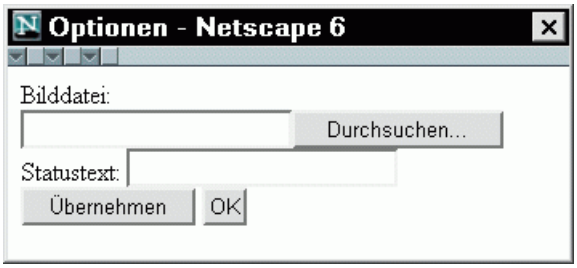
```
<html>
<head>
<title>Persönliche Seite</title>
<script language="JavaScript">
<!--
var Fenster_Optionen
function optionen(){
Fenster_Optionen=window.open("optionen.html","Optionen","width=350,height=120")
}
//-->
</script>
</head>
<body>
<h2>Hier ist dein Bild:</h2>
<img src="werbung.gif" name="bild"><p>
<form>
<input type="button" name="Button_Optionen" value="Optionen"
onClick="optionen()">
</form>
</body>
</html>
```

Es handelt sich also einfach um eine HTML-Seite mit einem Bild und einem Button. Klickst du auf den Button, wird die Funktion **optionen** ausgeführt, die die Datei OPTIONEN.HTML in einem neuen Fenster öffnet. Das funktioniert auf die gleiche Weise, wie wir es schon mehrmals in diesem Heft versucht haben.

Nun zur Datei OPTIONEN.HTML. Der Quelltext – noch ohne JavaScript – sieht so aus:

```
<html>
<head>
<title>Optionen</title>
<script language="JavaScript">
<!--
...
//-->
</script>
</head>
<body onRight="alert('test')">
<form>
Bilddatei: <input type="file"
name="bilddatei"><br>
Statustext: <input
name="statustext"><br>
<input type="button"
value="Übernehmen"
onClick="aktualisieren(false)">
<input type="button" value="OK"
onClick="aktualisieren(true)">
</form>
</body>
</html>
```

Vielleicht kennst du den Feldtype **file** noch nicht? Es handelt sich um ein einfaches Texteingabefeld, das mit einem zusätzlichen Button zum Durchsuchen des Computers nach Dateien ausgestattet ist. Schau dir einfach das Bild an:



Die beiden Buttons „Übernehmen“ und „OK“ haben fast die gleiche Funktion: Sie sollen das Bild und den Statustext im Hauptfenster aktualisieren. Bei einem Klick auf „OK“ soll das Optionen-Fenster geschlossen werden (der Parameter **aktualisieren** ist **true**), beim Klick auf „Übernehmen“ nicht (**false**).

Schau dir nun die JavaScript-Funktion **aktualisieren** an, die du im JavaScript-Teil der Datei OPTIONEN.HTML einfügen musst:

```
function aktualisieren(close){
opener.document.bild.src="file://"+
document.forms[0].bilddatei.value
opener.neuerstatus()
if (close==true){
window.close()
}
}
```

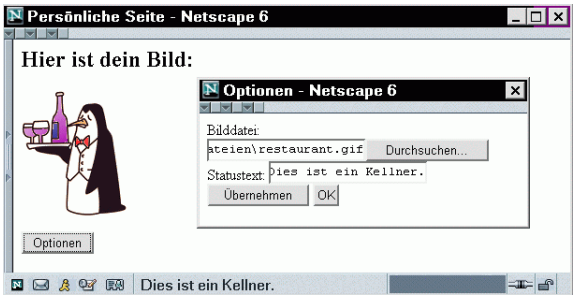
Du hast bereits gelernt, dass du erzeugte Fenster über den Objektnamen ansprechen kannst. Das erzeugte Fenster weiß allerdings den Namen seines Erzeugers nicht! Deshalb gibt es eine andere Möglichkeit: Mit **opener** sprichst du immer den Erzeuger an. Die erste Zeile der Funktion übergibt also an das Bild im Erzeugerfenster den Namen, der ins Dateifeld eingetragen wurde. In der nächsten Zeile wird die Funktion **neuerstatus** im Hauptfenster aufgerufen. Diese Funktion erstellen wir gleich noch. Schließlich wird das Optionen-Fenster geschlossen, falls der OK-Button gedrückt wurde.

Du siehst also, dass ein erzeugtes Fenster per JavaScript auf alle Elemente des Erzeugers zugreifen kann. Mit der Funktion **neuerstatus** möchte ich demonstrieren, dass es auch umgekehrt geht. Füge folgende Funktion, die den Statustext aktualisieren soll, in die Hauptdatei ein:

```
function neuerstatus(){
window.status=Fenster_Optionen.documen
t.forms[0].statustext.value
}
```

Hier wird ein Wert aus dem Tochterfenster in die Statuszeile geschrieben. Du musst einfach den Fensternamen und einen Punkt voranstellen, dann kannst du auf alle Funktionen, Variablen und Objekte eines anderen Fensters zugreifen!

Jetzt sind die beiden Dateien komplett – du kannst es ausprobieren:



Das Bild zeigt die Dokumente nach einem Klick auf „Übernehmen“. Beachte, dass sowohl das Bild als auch die Statuszeile geändert wurde.

Jetzt gibt es wieder drei Aufgaben für dich:

- Bis jetzt kannst du auch einfach überhaupt kein Bild angeben und auf „Übernehmen“ oder „OK“ klicken. Verhindere, dass die Seite in diesem Fall aktualisiert wird, und lasse eine Warnmeldung ausgeben. Benutze eine **if**/**else**-Konstruktion und den Befehl **alert**.
- Du hast gesehen, dass sowohl das erzeugte Fenster auf den Erzeuger zugreifen kann als auch umgekehrt. Es gibt noch eine Möglichkeit, den Text in die Statuszeile zu schreiben, die hier sinnvoll wäre. Rüste die Funktion **neuerstatus** mit einem Parameter aus, der den neuen Statustext enthalten soll. Übergib in der Funktion **aktualisieren** beim Aufruf von **neuerstatus** den Wert aus dem Formular.
- Füge dem Optionen-Fenster einen Abbrechen-Button hinzu, der das Fenster nur schließt, ohne Änderungen am Dokument vorzunehmen.

Andere Fenster

Jetzt kannst du also schon recht gut mit Fenstern arbeiten. Du könntest auch ohne Probleme ein Fenster programmieren, das nur einen Meldungstext und einen OK-Button enthält. Wie du aber sicher schon weißt, gibt es dafür den **alert**-Befehl, der das ganze vereinfacht.

Neben **alert** gibt es noch zwei weitere Befehle, die Standarddialoge anzeigen. Alle drei Dialoge haben die Eigenschaft, dass sie erst geschlossen werden müssen, bevor wieder auf das Browserfenster zugegriffen werden kann. Aber sieh selbst:

OK oder Abbrechen: confirm

Schau dir folgenden Code an:

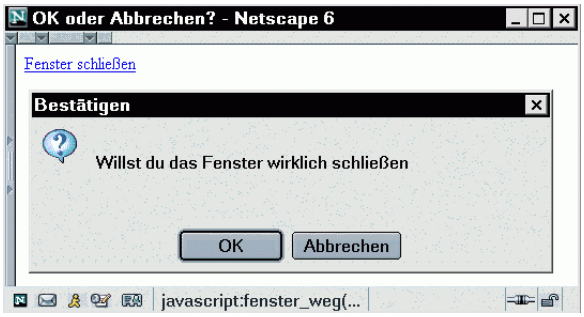
```
<html>
<head>
<title>OK oder Abbrechen?</title>
<script language="JavaScript">
<!--
function fenster_weg(){
if (confirm("Willst du das Fenster
wirklich schließen")){
window.close()
}
}
//-->
</script>
</head>
<body>
<a
href="javascript:fenster_weg()">Fenste
r schließen</a>
</body>
</html>
```

Die Seite enthält lediglich einen Verweis, der die Funktion **fenster_weg** aufruft. In dieser Funktion kommt die Funktion **confirm** zum Einsatz. Diese Funktion gibt entweder **true** (OK) oder **false** (Abbrechen) zurück. Je nach Ergebnis wird das Fenster also geschlossen oder nicht.

Vielleicht erscheint es dir etwas ungewöhnlich, dass in einer **if**-Abfrage eine Funktion steht – und nicht beispielsweise ein Vergleich zweier Werte. Du könntest genauso schreiben:
if (confirm("Willst du das Fenster wirklich schließen")==true){

Dieser Ausdruck ist ebenfalls wahr, wenn auf OK geklickt, und falsch, wenn ABBRECHEN gewählt wird. Nur ist er eben ein bisschen komplizierter.

Im Browser sieht das ganze dann so aus:



Zum Schluss wieder eine Aufgabe für dich:

- Baue diese Abfrage in das Beispiel „Fenster schließen“ aus dem letzten Kapitel ein. Dazu ist es sinnvoll, die beiden **if**-Abfragen („Ist das Fenster noch geöffnet?“ und „Hat der Benutzer mit dem OK-Button bestätigt?“) in eine eigene Funktion zu packen. Überlege genau, in welcher Reihenfolge du die **if**-Abfragen ineinander schachteln musst! Wenn das Fenster schon geschlossen ist, soll nicht nachgefragt werden, ob das Fenster geschlossen werden soll...

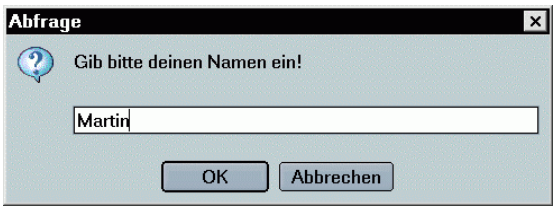
Texteingabe: prompt

Im nächsten Beispiel wollen wir den Surfer persönlich begrüßen. Dazu benötigen wir aber seinen Namen. Mit **prompt** ist das möglich: Die Funktion zeigt einen Eingabedialog mit einem Textfeld und gibt den eingegebenen Text zurück. Hier der Quelltext:

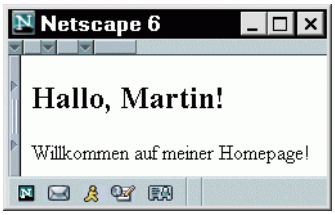
```
<html>
<head>
<title></title>
<script language="JavaScript">
<!--
name=prompt("Gib bitte deinen Namen ein!")
//-->
</script>
</head>
<body>
<h2>Hallo,
<script>document.write(name)</script>!
</h2>
Willkommen auf meiner Homepage!
</body>
</html>
```

Noch bevor Text auf dem Bildschirm angezeigt werden kann, wird die eine Zeile JavaScript-Code ausgeführt. **prompt** erzeugt ein Dialogfenster zur Texteingabe. Der übergebene Parameter ist die Eingabeaufforderung. Der Name des Surfers wird in die Variable **name** geschrieben, falls der Surfer diesen eintippt und auf OK klickt. Beim Erstellen der Seite wird der Name mittels **document.write** in die Überschrift eingefügt.

Und so interpretiert Netscape die **prompt**-Funktion:



Die persönliche Begrüßung sieht dann so aus:



Diesmal habe ich wieder drei kleine Aufgaben für dich:

- Ursula ist deine beste Freundin. Jedes Mal, wenn „Ursula“ eingegeben wird, soll zusätzlich folgender Satz erscheinen: „Schön, dass du auch mal wieder auf meiner Seite vorbeischaust!“
  Benutze eine **if**-Abfrage, die du in **<script>**-Tags vor **</body>** einbaust.
- Wenn du der Funktion **prompt** einen zweiten Parameter übergibst, wird dieser als Standardtext in die Textabfrage eingefügt. Da du weißt, dass Ursula die häufigste Besucherin deiner Seite ist, soll ihr Name als Voreinstellung im Dialog erscheinen.
- Damit du den dritten Befehl für Dialogfenster, den **alert**-Befehl nicht ganz vergisst: Sorge dafür, dass die Meldung „Du willst wohl anonym bleiben!“ ausgegeben wird, wenn kein Name eingegeben wird. Baue den **alert**-Befehl direkt nach der **prompt**-Funktion im JavaScript-Teil ein und benutze dafür eine **if**-Abfrage.

Cookies

Was sind Cookies?

Cookies sind kurze Textinformationen, die auf der Festplatte des Surfers gespeichert werden. Du als Programmierer kannst so bestimmte benutzerspezifische Daten speichern. Typische Anwendungen für Cookies sind das Speichern des Namens des Benutzers, um ihn jedes Mal persönlich zu begrüßen, oder das Zählen der Zugriffe des Surfers auf deine Site. Letzteres wollen wir im folgenden Beispiel ausprobieren.

> Cookies sind eine sehr umstrittene Technik, da sie tief in die Privatsphäre des Nutzers eingreifen. Deshalb deaktivieren viele Surfer Cookies in ihrem Browser. Deine Seite sollte also möglichst auch ohne Cookies problemlos funktionieren!

Natürlich müssen auch in deinem Browser die Cookies aktiviert sein, wenn du das Beispiel ausprobieren möchtest.

Unter Netscape aktivierst du unter BEARBEITEN / EINSTELLUNGEN in der Kategorie PRIVATSPHÄRE UND SICHERHEIT / COOKIES die Option ALLE COOKIES AKZEPTIEREN und bestätigst die Änderungen mit OK.

Benutzt du den Internet-Explorer, so klickst du auf EXTRAS / INTERNETOPTIONEN. In der Registerkarte SICHERHEIT klickst du dann unter SICHERHEITSSTUFE DIESER ZONE auf STUFE ANPASSEN. Aktiviere hier unter COOKIES die beiden Optionen AKTIVIEREN. Mit OK bestätigst du die Änderungen.

Cookies schreiben

Jedes Mal, wenn der Surfer deine Seite besucht, soll ihm eine Meldung angezeigt werden. Sie soll folgenden Satz enthalten: „Du bist zum x. mal auf dieser Seite!“ Natürlich wird „x“ durch eine Zahl ersetzt. Dazu benötigen wir ein Cookie, das auf der Festplatte des Surfers gespeichert wird, und die Zahl „x“ enthält. Der Quelltext ist einfach – das Cookie wird hier nur geschrieben:

```
<html>
<head>
<title>Besuchszähler</title>
<script language="JavaScript">
<!--
var zaehler=1
var auszeit=new Date()
auszeit=new
Date(auszeit.getTime()+1000*60*60*24*
365)
document.cookie =
"Zaehler="+zaehler+";
expires="+auszeit.toGMTString()+";"
alert("Dies ist dein " + zaehler + ".
Besuch auf dieser Seite!")
//-->
</script>
</head>
<body>
Inhalt der Seite.
</body>
</html>
```

Beim Aufruf der Seite wird nun die Variable **zaehler** zunächst auf 1 gesetzt, da es sich um deinen ersten Besuch handelt. Später werden wir an dieser Stelle das Cookie auslesen, falls die Seite schon einmal besucht wurde.

Ein Cookie hat immer einen Verfallszeitpunkt, zu dem es von der Festplatte gelöscht wird. Wir möchten, dass das Cookie für die Dauer eines Jahres gespeichert wird. Ein neues Datumsobjekt namens **auszeit** wird deklariert. Es enthält nun das aktuelle Datum und die aktuelle Uhrzeit. In der nächsten Zeile wird es neu deklariert – es enthält nun den Zeitpunkt um ein Jahr in der Zukunft.

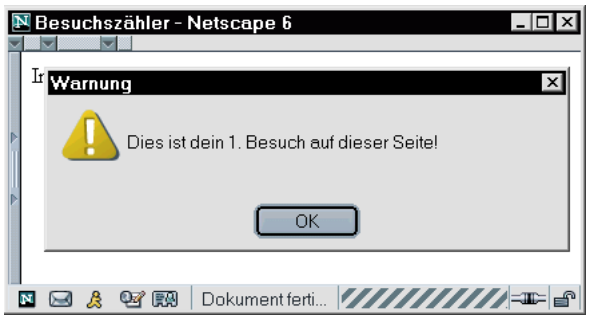
Zur aktuellen Zeit (**auszeit.getTime()**) wird ein Jahr (also 1000x60x60x24x365 Millisekunden) dazugezählt. Das Cookie wird nun zum angegebenen Zeitpunkt – also zur gleichen Zeit in einem Jahr – wieder gelöscht.

Nun wird das Cookie mit **`document.cookie=...`** geschrieben. Es wird eine Zeichenfolge folgender Struktur übergeben:

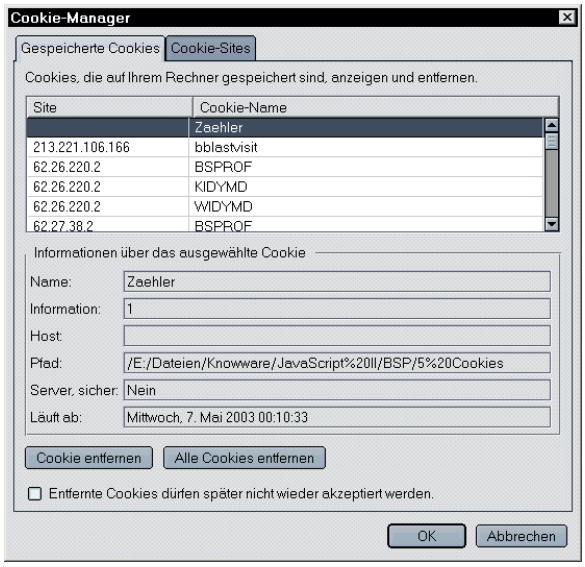
```
Zaehler=1; expires=Thu, 30 Aug 2003
10:24:34 GMT;
```

Das erste Element der Zeichenkette enthält Name und Wert des Cookies, das zweite Element den Verfallszeitpunkt. Mit der Funktion von Datumsobjekten **`toGMTString()`** wird das Datum richtig formatiert.

Das Programm funktioniert natürlich fehlerfrei, es wird die erwartete Meldung ausgegeben:



Viel wichtiger ist aber, dass nun ein Cookie geschrieben worden ist. Unter Netscape 6 kannst du alle Cookies mit Netscape unter AUFGABEN / PRIVATSPHÄRE UND SICHERHEIT / COOKIE-MANAGER / GESPEICHERTE COOKIES ANZEIGEN einsehen. Hier siehst du unser neu erstelltes Cookie:



Cookies lesen

Bis jetzt ist der Zähler natürlich nicht besonders sinnvoll, da er bei jedem Besuch „1“ in das Cookie schreibt und nie mehr wieder etwas aus dem Cookie ausliest. Also muss bei jedem Laden der Seite der Wert gelesen werden.

Hierzu musst du wissen, dass das Cookie nun folgendermaßen gespeichert ist:

```
Zaehler=1
```

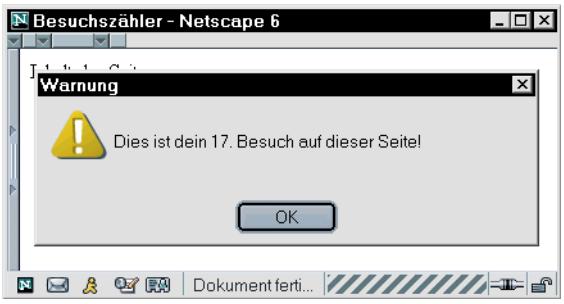
Folgender Quelltext, den du nach der Deklaration von **`zaehler`** einfügen musst, extrahiert die Zahl:

```
if (document.cookie){
var
strZaehler=document.cookie.substring
(document.cookie.indexOf("=")+1,
document.cookie.length)
zaehler=parseInt(strZaehler)+1
}
```

Die **`if`**-Bedingung dient dazu, dass die Zahl nur dann ausgelesen wird, wenn das Cookie schon vorhanden ist. **`document.cookie`** ist wahr, wenn diese Seite bereits ein Cookie gesetzt hat. Die Funktion **`substring`** kennst du schon aus dem ersten Heft. In diesem Fall liest sie alle Zeichen nach „=“ **`document.cookie.indexOf("=")+1`**) bis zum Ende des Strings (**`document.cookie.length`**). Die Zahl wird zunächst in der Variablen **`strZaehler`** gespeichert, es handelt sich noch um eine Zeichenkette. Im nächsten Schritt wird sie mit der Funktion **`parseInt`** in eine Zahl gewandelt und um eins erhöht der Variablen **`zaehler`** übergeben.

Wurde die Seite noch nie zuvor besucht, ist noch kein Cookie gesetzt. Die **`if`**-Bedingung ist also nicht wahr und die Variable **`zaehler`** behält ihren Wert 1. Bei jedem Aufruf der Seite wird das Cookie mit der aktuellen Anzahl der Besuche neu gesetzt. Somit beginnt das eine Jahr bis zum Ablauf des Cookies auch nach jedem Aufruf neu zu zählen.

Mit relativ wenig Code hast du jetzt einen voll funktionsfähigen Zähler gebastelt:



Um die doch gewöhnungsbedürftige Arbeit mit Cookies etwas zu trainieren, habe ich wieder einige Aufgaben für dich:

- Füge auf der Seite einen Link hinzu, der den Inhalt von **`document.cookie`** als Meldung ausgibt.
- Lese mit einem **`prompt`**-Befehl den Namen des Benutzers ein. Schreibe ihn analog zu der Anzahl der Besuche mit dem Namen **`Name`** in das Cookie. Dazu rufst du den Befehl **`document.cookie = "..."`** einfach noch einmal auf – diesmal natürlich mit geändertem Namen und geändertem Wert.
- Klickst du auf den erstellten Link, erfährst du, dass das Cookie etwa so heißt:
  `Zaehler=12; Name=Martin`
  Überlege, warum der Zähler trotzdem richtig weiterläuft. Eigentlich müsste das Ergebnis des Zählers **`12; Name=Martin`** lauten.
  Hinweis: Beachte die Funktion von **`parseInt`**!
- Es soll nun die Meldung in folgender Form ausgegeben werden: „Hallo [name], du bist zum [zaehler]. mal auf dieser Seite.“
  Überlege, wie du die Parameter der Funktion **`substring`** abwandeln musst, um den Namen zu extrahieren.
- Sorge dafür, dass der Name nur eingelesen wird, wenn die Seite noch nie besucht wurde, also noch nie Gelegenheit war, einen Namen einzugeben.