Einführung in Javascript

Hinweis:

Zum Bearbeiten der Beispiele und Übungen stehen Ihnen die entsprechenden Quelldateien und Materialien gesondert zur Verfügung.

Download dieser Dateien unter der Adresse: http://www.stz-isd.de/aik/jsdemos/

Die hier vorgestellten Seiten funktionieren zum Teil nur im Internet-Explorer.
Anleitungen zur Anpassung an andere Browser finden Sie im Anhang.

1. Einbinden von Javascript

Es gibt prinzipiell zwei Möglichkeiten JavaScript in HTML-Dateien einzubinden. Zum einen kann der JavaScript-Code direkt in der HTML-Datei integriert sein, zum anderen kann er auch extern in einer eigenen Script-Datei stehen. Letzteres hat den Vorteil, dass der gleiche Code für verschiedene Webseiten verwendet werden kann. Baut man beispielsweise eine zentrale Navigation mittels einer externen Javascript-Datei, so lässt sich diese leicht in alle Seiten integrieren. Bei Veränderungen der Navigation muss nur noch die Script-Datei modifiziert werden.
Die direkte Integration in eine HTML-Datei sieht so aus:

```
<html>
<head>
   <script type="text/javascript">
      // Hier wird einfach eine Alarmbox ausgegeben:
      alert("Hallihallo");
   </script>
</head>
<body>
</body>
</html>
```

Einbinden von JavaScript in HTML: intern

Der JavaScript-Teil wird über das Element `<script>` eingefügt. Dessen Attribut `type` gibt an, um was für eine Sorte Script es sich handelt – in diesem Fall um ein textcodiertes JavaScript. Anschließend folgt die Ausgabe einer Alarmbox.
Beachten Sie: Der Text hinter den zwei Schrägstrichen (`//`) ist ein Kommentar. In JavaScript werden Kommentare wie in Java formuliert:

1. Einzeilige Kommentare werden durch `//` eingeleitet. Alles was hinter den beiden Schrägstrichen in einer Zeile steht wird vom Browser ignoriert. Erst die nächste Zeile wird wieder verarbeitet. Diese Kommentare müssen nicht am Beginn einer Zeile stehen, sondern können auch hinter einer Script-Anweisung stehen.
2. Mehrzeilige Kommentare werden mit `/*` eingeleitet und mit `*/` beendet. Alles, was zwischen diesen beiden Zeichenkombinationen steht, wird vom Browser ignoriert.

Für das Einbinden einer externen JavaScript-Datei wird zunächst eine solche benötigt:

```
alert("hallihallo");
```

Einbinden von JavaScript: externe JavaScript- Datei „begruessung.js“

Hier wird eine Alarmbox mit dem Inhalt „hallihallo“ aufgerufen. Beachten Sie, dass Sie nach jeder Anweisung ein Semikolon setzen müssen.
Externe JavaScript-Dateien sind an der Extension `*.js` zu erkennen. Sie werden wie folgt in eine HTML-Seite eingebunden:

```
<html>
<head>
   <script src="begruessung.js" type="text/javascript"></script>
</head>
<body>
</body>
</html>
```

Einbinden der externen JavaScript-Datei „begruessung.js“

Das `<script>`-Element wird hier zusätzlich mit dem Attribut `src` versehen, welches einen Verweis auf die zu integrierende JavaScript-Datei enthält. Vergessen Sie nicht, die notwendigen Pfadangaben zu machen, wie Sie es vom Einbinden von CSS-Dateien, Bildern und Links gewohnt sind.

2. Funktionen

In Funktionen können Sie Programmabläufe definieren, die sie bei Bedarf aufrufen können. Sie können Funktionen jederzeit, wenn es nötig ist, auch mehrmals aufrufen. Der Vorteil von Funktionen liegt darin, dass sie bestimmte Abläufe nur einmal definieren müssen und mehrmals aufrufen können. Eine Funktion sieht in JavaScript wie folgt aus:

```
function begruessung(){
   alert("hallihallo");
}
```

JavaScript-Datei begruessung.js mit Funktion

Die Deklaration einer Funktion beginnt mit dem Schlüsselwort `function`. Anschließend folgt der Name der Funktion, hier: `begruessung`. Nach dem Namen definieren Sie in Klammern die Parameter, die der Funktion übergeben werden sollen. Parameter dienen dazu, die Funktion speziell für den gebrauchten Einsatz zu justieren. In diesem Beispiel werden keine Parameter übergeben. Was die Funktion genau tun soll definieren sie anschließend in geschwungenen Klammern. Hier wird wie oben einfach eine Alarmbox aufgerufen.
Um eine Funktion aufzurufen, stehen Ihnen prinzipiell zwei unterschiedliche Möglichkeiten zur Verfügung. Zum einen können Sie die Funktion einfach innerhalb eines Script-Bereichs aufrufen. Das sieht dann etwa so aus:

```
<html>
<head>
   <script src="begruessung.js" type="text/javascript"></script>
</head>
<body>
   <script type="text/javascript">
      begruessung();
   </script>
</body>
</html>
```

Aufruf einer Funktion aus einem Script

Für den Aufruf müssen Sie nur den Namen der Funktion aufrufen und die entsprechenden Parameter nennen. Vergessen Sie auch hier nicht das Semikolon am Ende des Aufrufs. (Die Deklaration der Funktion muss natürlich nicht in einer externen Datei stattfinden, sondern kann auch direkt in der gleichen HTML-Seite stehen.)
Auf diese Art und Weise können Sie jederzeit aus einem Script heraus Funktionen aufrufen. Mitunter ist es sogar nötig, aus einer Funktion heraus Unterfunktionen aufzurufen. Man spricht hierbei von Schachtelung.

Die zweite Methode, eine Funktion aufzurufen, ist sie mit einem HTML-Element zu verknüpfen. Dazu stehen ihnen eine Reihe von Befehlen zur Verfügung, zum Beispiel der Befehl `onload`:

```
<html>
<head>
   <script src="begruessung.js" type="text/javascript"></script>
</head>
<body onload=begruessung();>
</body>
</html>
```

Funktionsaufruf aus einem Element heraus

In diesem Beispiel wird die Funktion `begruessung()` aus dem `<body>`-Element heraus aufgerufen. Dies geschieht durch den Befehl `onload`, der bewirkt, das beim Laden des Elements die entsprechende Funktion aufgerufen wird. Ein anderer Befehl dieser Art lautet `onClick`, der den Funktionsaufruf erst bei einem Mausklick auf das entsprechende Element startet:

```
<html>
<head>
   <script src="begruessung.js" type="text/javascript"></script>
</head>
<body>
<h2 onclick=begruessung();>Begruessung</h2>
</body>
</html>
```

Funktionsaufruf mit onclick

In diesen Beispielen haben die Funktionen immer das Gleiche getan. Sie können Funktionen aber auch durch Parameter steuern. Dazu brauchen Sie zunächst eine Funktion, die Parameter erhalten kann:

```
function begruessung(name){
   alert("hallihallo"+name);
}
```

JavaScript-Datei begruessung.js mit parametrisierbarer Funktion

Diese Funktion können Sie nun mit verschiedenen Parametern aufrufen:

```
<html>
<head>
   <script src="begruessung.js" type="text/javascript"></script>
</head>
<body>
   <script type="text/javascript">
      begruessung("Monika Musterfrau");
      begruessung("Markus Mustermann");
   </script>
</body>
</html>
```

Zweifacher Funktionsaufruf mit verschiedenen Parametern

Beachten Sie bei diesem Beispiel, dass Javascript die Texte „hallihallo“ und die Namen mit dem + - Zeichen zu einem Text verbindet.
Die funktioniert natürlich mit jedem beliebigen Namen, wie folgendes Beispiel zeigt:

```
<html>
<head>
   <script src="begruessung.js" type="text/javascript"></script>
</head>
<body>
   <script type="text/javascript">
      begruessung(prompt("Bitte geben Sie Ihren Namen ein", "Name"));
   </script>
</body>
</html>
```

Namenseingabe mit prompt

Hier sind zwei Funktionen verschachtelt. Die Funktion `prompt()` führt zu einem Eingabefeld. Sie müssen zwei Parameter angeben. Erstens den Text, der über dem Eingabefeld stehen soll, und zweitens den Text, der in dem Eingabefeld vordefiniert stehen soll. Die Eingabebox liefert den Inhalt des Eingabefeldes als Ergebnis zurück. Dieses Ergebnis wird hier direkt als Parameter für die Funktion `begruessung()` verwendet. Solche Verschachtelungen sind anfangs vielleicht etwas schwer zu lesen und zu programmieren, führen aber zu sehr effizientem Programmcode.

3. Variablen

Variablen ermöglichen die Speicherung von Informationen, die für den Probgrammablauf benötigt werden. Dieses Speichern kann entweder für den gesamten Programmablauf vorgenommen werden; man spricht dann von globalen Variablen. Oder aber Variablen gelten nur innerhalb von Funktionen, dann spricht man von lokalen Variablen. Ein Beispiel für eine globale Varable:

```
var name;
function setzeName(n) { name=n; }
function begruessung() { alert("hallihallo "+name); }
```

berguessung.js mit globaler Variable name

Eine Variable wird durch das Schlüsselwort `var` deklariert. In diesem Beispiel steht die Deklaration außerhalb einer Funktion, d.h. die Variable gilt als global. Auf sie kann also von überall im Programmcode zugegriffen werden. Stünde die Variablendeklaration innerhalb einer Funktion, könnte nur diese eine Funktion auf die Variable zugreifen.
Wenn die Funktion `setzeName()` aufgerufen wird, dann wird der Variablen `name` ein Wert zugewiesen, und zwar der Wert, der der Funktion als Parameter übergeben wurde. Wird die Funktion `begruessung()` aufgerufen, so wird eine Alarmbox geöffnet, die wiederum den Inhalt der Variablen `name` verwendet.
Die Verwendung der Funktionen könnte wie folgt aussehen:

```
<html>
<head>
   <script src="begruessung.js" type="text/javascript"></script>
</head>
<body>
   <script type="text/javascript">
      setzeName(prompt("Bitte geben Sie Ihren Namen ein", "Name"));
      begruessung();
   </script>
</body>
</html>
```

Aufruf der Funktionen mit der Variablen name

Übung 1: Erweitern Sie die JavaScript-Datei und die HTML-Datei so, dass nicht nur der Name, sondern auch der Vorname abgefragt wird. Definieren Sie dafür eine neue Variable `vorname`, die entsprechende Funktion `setzeVorname()`, und modifizieren sie Aufrufe und Ausgabe entsprechend.

Variablen können neben textuellem Inhalt natürlich auch Zahlen enthalten. In diesem Fall können mit den Variablen die üblichen mathematischen Operationen durchgeführt werden:
Mit den Variablen

```
var x=3;
var y=2;
var z;
```

können beispielsweise folgende Operationen vorgenommen werden:

```
z=x+y    →5                          z=x-y        →1
z=x*y    →6                          z=9/x        →3
```

Besondere verkürzende Operatoren sind:

```
x=x+1    x++       →4 (inkrementieren)
x=x-1    x--       →2 (dekrementieren)

x=x+2       x+=2          →5
x=x-2       x-=2          →1
x=x*3       x*=3          →9
x=x/3       x/=3          →1
```

Übung 2: Erweitern Sie die JavaScript-Datei und die HTML-Datei so, dass nicht nur der Name und Vorname, sondern auch das Geburtsjahr abgefragt wird. Auf der Basis der Eingabe soll das Alter der Person ermittelt werden. Sie benötigen zusätzlich die globale Variable `alter`. Schreiben Sie eine Funktion `setzeAlter()`, die das heutige Alter auf Jahresbasis berechnet und geben Sie in der Funktion `begruessung()` das Alter mit aus.

4. Bedingte Anweisungen

Mit Hilfe von bedingten Anweisungen können Sie die Ausführung von Programmabläufen an Bedingungen knüpfen. Die am Häufigsten eingesetzte bedingte Anweisung ist die wenn-dann-Regel: Wenn eine bestimmte Bedingung vorliegt, dann wird eine bestimmte Anweisung vorgenommen. In Javascript wird diese Anweisung durch `if` ausgedrückt:

```
<script type="text/javascript">
   var x;
   x=prompt("Bitte geben Sie eine Zahl ein", "");
   if (x>0){
      alert("Sie haben eine positive Zahl eingegeben.");
   }else if (x<0){
      alert("Sie haben eine negative Zahl eingegeben.");
   }else if (x==0){
      alert("Sie haben eine 0 eingegeben");
   }else{
      alert("Sie haben keine gültige Zahl eingegeben.");
   }
</script>
```

if-Anweisung

In diesem Beispiel wird zunächst eine Variable `x` deklariert. Dieser wird über eine Eingabebox ein Wert zugewiesen. Anschließen folgt eine Untersuchung der Variablen durch `if`-Anweisungen. Dabei wird in den ersten drei Fällen die Variable `x` mit der Zahl 0 verglichen. Ist `x` größer Null wird der Text „Sie haben eine positive Zahl eingegeben.“ in einer Alarmbox ausgegeben. Entsprechendes gilt für den Fall, dass `x` kleiner 0 ist. Solche sukzessiven Abragen werden durch die Anweisung `else` ermöglicht. Sie tritt in Kraft, wenn die `if`-Abfrage negativ beschieden wurde.
Eine Besonderheit ist die Frage, ob `x` gleich 0 ist. Hier muss ein doppeltes Gleichheitszeichen verwendet werden. Ein doppeltes Gleichheitszeichen gilt als Vergleichsoperator, ein einfaches Gleichheitszeichen gilt als Zuweisungsoperator. Bei einem solchen würde hier also `x` auf 0 gesetzt. Ein Vergleich würde nicht gemacht, und das Script eine Fehlermeldung erzeugen. Zum Schluss folgt eine `else`-Regel für den Fall dass überhaupt keine Zahl eingegeben wurde. Sie fängt alle Eingaben ab, die nicht schon durch vorige `if`-Anweisungen bearbeitet wurden.
Wenn für einen bestimmten Programmschritt mehrere Bedingungen gleichzeitig gelten müssen, können Sie diese natürlich durch verschachtelte `if`-Anweisungen ausdrücken. Sie können aber auch eleganter eine `if`-Anweisung an mehrere Bedingungen gleichzeitig knüpfen. Wenn Sie beispielsweise in der letzten Aufgabe an die Berechnung des Alter denken: Dort sollte abgefragt werden, ob das angegebene Geburtsjahr überhaupt möglich ist. Ein Geburtsjahr nach 2005 ist sicher unsinnig und ein Geburtsjahr vor 1900 doch eher unwahrscheinlich. Beide Fälle können Sie so ausschließen:

```
function setzeAlter(j){
   if(j>2005||j<1900){
      alert("Sie haben keine gültige Zahl eingegeben!");
      alter="Alter unbekannt";
   }else{
      alter=2002-j;
   }
}
```

modifizierte Funktion setzeAlter()

Das besondere hier ist die Verbindung zweier Bedingungen durch den Doppelbalken `||`, mit dem das logische „oder“ ausgedrückt wird. Die Programmzeile `if(j>2005||j<1900){` liest sich also: „wenn j größer als 2005 oder wenn j kleiner als 1900, dann...“. Sie können beliebig viele solcher Verbindungen in der `if`-Anweisung unterbringen. Beachten Sie, dass das logische „oder“ einschließend ist, also mindestens eine der beiden Bedingungen, evtl. aber auch beide gelten können. Verwechseln Sie das logische „oder“ nicht mit dem natürlichsprachigen „oder“ im Sinne von „entweder ... oder“. Ein logisches „und“ beschreiben Sie durch ein doppelt gesetzten `&&`. In diesem Fall müssen alle verknüpften Bedingungen ausnahmslos zutreffen.

5. Schleifen

JavaScript bietet die Möglichkeit, bestimmte Programmsequenzen wiederholt auszuführen, bis das gewünschte Ergebnis erreicht ist. Eine dieser Möglichkeiten ist die `while`-Schleife:

```
<script type="text/javascript">
   var j="";
   while (j!="nta-isny"){
      j=prompt("Zugang gesperrt! Bitte geben Sie Das Passwort ein", "");
   }
   alert("Zugang freigegeben!");
</script>
```

while-Schleife

Die `while`-Schleife wird über das Schlüsselwort `while` eingeleitet. Anschließend wird in Klammern die Bedingung definiert, die erfüllt sein muss, damit die `while`-Schleife abbricht. In geschweiften Klammern steht der Programmteil, der immer wieder ausgeführt wird, bis die Schleife abbricht. In diesem Beispiel wird solange eine Eingabebox aufgerufen, bis die Eingabe „nta-isny“ lautet.

Übung 3: Formulieren Sie die JavaScript-Datei und die HTML-Datei so, dass die Eingabe des Alters über eine while-Schleife abgefangen wird. Nur ein zulässiges Jahr soll weitergeleitet werden. Bei unzulässigen Jahren soll die Eingabe erneut aufgerufen werden. Modifizieren Sie dafür zunächst die Funktion setzeAltert() so, dass kein Parameter mehr übergeben wird. Setzen sie die prompt-Anweisung und deren Verarbeitung anschließend in diese Funktion.

Eine andere Möglichkeit Schleifen zu definieren ist die `for`-Schleife. Hier wird einer variablen eine Reihe von Werten zugeordnet, die nacheinander abgearbeitet werden:

```
<script type="text/javascript">
   var i;
   for (i=1; i<=10; i++){
      alert(i);
   }
</script>
```

for-Schleife

Die `for`-Schleife wird durch das Schlüsselwort `for` eingeleitet. Anschließend werden in Klammern zunächst der Anfangswert, dann der Endwert und schließlich noch die Berechnung der Zwischenwerte der Variablen angegeben. In diesem Beispiel wird der Variablen `i` der Anfangswert 1 und der Endwert 10 zugeordnet. Die Zwischenwerte berechnen sich einfach aus einer steten Steigerung des Wertes von `i` um 1, also 1,2,3,4,5,6,7,8,9,10. Beachten Sie, dass Sie den Endwert mit <= angeben müssen. Das <-Zeichen sorgt dafür, dass alle Werte unter dem Endwert berücksichtigt werden.

6. Objekte

JavaScript ermöglicht die Programmierung mit Objekten. Sie können entweder selbst Objekte definieren oder aber Sie greifen auf Standardobjekte zurück, die von JavaScript unterstützt werden. Letzteres soll hier veranschaulicht werden.
Standardobjekte in JavaScript sind die Objekte auf die sie normalerweise bei der Programmierung referieren müssen. Solche Objekte sind zum einen der Browser in dem sich das Script abspielen soll, der `navigator`. Dann natürlich das Browserfenster, das `window`. Innerhalt des `window` gibt es das in der Praxis meist verwendete Objekt `document`. Das ist der Bereich der Browserfensters, in dem Ihre HTML-Seite erscheint, also all das was Sie bislang im `<body>` definiert haben.
Neben diesen auch auf dem Bildschirm greifbaren Objekten gibt es noch weitere Objekte die eine Reihe interessanter Standardfunktionen beinhalten: `Math` (eine Sammlung mathematischer Funktionen), `date` (alles was Sie zur Bestimmung von Datum und Uhrzeit brauchen), `history` (die History Ihres Browsers), und viele andere.
Objekte, ihre Eigenschaften und ihre Funktionen werden über die Punktnotation angesprochen. Wollen sie beispielsweise die Eigenschaft `width` eine Objekts `object` ansprechen, schreiben Sie: `oject.width`

```
<script type="text/javascript">
   alert(navigator.appCodeName);
   alert(navigator.appName);
   alert(navigator.appVersion);
   alert(navigator.userAgent);
   alert(screen.width);
   alert(screen.height);
</script>
```

Beispiel: Aufruf verschiedener Eigenschaften zur Ermittlung der Systemvoraussetzungen.

Mit Hilfe dieses Beispiels können Sie zwar einiges über Ihren Browser erfahren, interessant wird die Sache aber erst, wenn Sie auch die Inhalte modifizieren können. In der Praxis könnten Sie beispielsweise aufgrund der Informationen über die Bildschirmbreite die Elemente auf dem Bildschirm individuell anordnen.
Hier sollen zwei Methoden erklärt werden, um Inhalte auf dem Bildschirm zu modifizieren. Zunächst einmal das einfache Hinzufügen von Elementen:

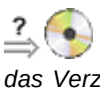
```
<script type="text/javascript">
   document.write("<h1>Ja sapperlott!</h1>");
</script>
```

write-Befehl

`document` bezeichnet den Teil der Browserfensters, in dem Sie den Inhalt definieren. Bisher haben Sie das immer anhand von HTML-Elementen gemacht. Sie können aber auch die JavaScript-Funktion `write()` dazu verwenden. Wenn Sie die `write`-Anweisungen in einer externen JavaScript-Datei definieren, können Sie so sehr elegant eine Navigation bauen, die Sie zentral pflegen können. Beachten Sie dabei, dass Sie Anführungszeichen des zu schreibenden Textes mit einem Schrägstrich speziell markieren müssen, damit JavaScript sie nicht für die Anführungszeichen des `write()`-Befehls hält. Schreiben Sie also:

```
document.write("<a href=\"verweis.html\">Ein Link</a><br>");
```

Umgang mit Anführungszeichen



Übung 4: Knöpfen Sie sich noch mal Ihre Rezeptdateien vor. Wenn Sie sie nicht mehr haben, gehen Sie in das Verzeichnis `Aufgaben-Materialien\Rezepte\`, dort finden Sie alle notwendigen Dateien: eine Indexdatei, eine CSS-Datei und die HTML-Rezeptdateien. Gehen Sie nun wie folgt vor:

1.: Schreiben Sie eine externe JavaScript-Datei `navigation.js`, in der Sie in einer Funktion `showMenu()` mit Hilfe des `write()`-Befehls eine Liste mit Links zu Ihren Rezepten definieren. Eine Vorlage finden Sie dazu in Ihrer Index-Datei.

2.: Rufen Sie auf jeder Rezeptseite an geeigneter Stelle die Funktion `showMenu()` auf, so dass Sie auf jeder Rezeptseite die vollständige Linksliste haben.

Eine sehr gängige Lösung zum dynamischen Verändern von HTML-Inhalten ist die Verwendung des -Elements in Verbindung mit Scripten. Das -Element dient lediglich zur Definition von Bereichen auf Ihrer Seite. Diese Bereiche können Sie mit Hilfe von JavaScript beliebig mit Inhalt füllen, erscheinen und verschwinden und auf dem Bildschirm wandern lassen.

```
<body>
   <div id="divMenu" style="position:absolute; top:400; left: 300; visibility:hidden;
   z-index: 2;"></div>

   <script type="text/javascript">
      var einInhalt="<h3>Hallihallohallöchen!</h3>";
      var zweiInhalt="Hier könnte Ihr Text stehen<br>";
      var dreiInhalt="<sub>und tschüss</sub>";

      function showInhalt(){
         document.all.divMenu.innerHTML=einInhalt+zweiInhalt+dreiInhalt;
         document.all.divMenu.style.visibility="visible";
         }
   </script>

   <a href="javascript:showInhalt()">Klicken Sie hier um den Inhalt zu sehen!</a>
</body>
```

-Füllen

In diesem Beispiel wird zunächst ein leerer -Bereich definiert. Er bekommt die ID „divMenu“. Die Vergabe einer ID ist sehr wichtig, da der Bereich über sie angesprochen werden kann. Anschließend folgen einige Style-Angaben zu Position und Sichtbarkeit.
In dem darauffolgenden Script werden zunächst drei Inhaltsvariablen definiert. Anschließend folgt eine Funktion, die zunächst dem -Bereich einen Inhalt zuweist, und ihn anschließend sichtbar macht.
Die Ansprache des -Bereichs ist Microsoft-spezifisch: Als oberster Objektknoten wird das `document` aufgerufen. Anschließen folgt die Ansprache aller Elemente des Dokuments durch `all`. Von diesen wiederum wird nur der Bereich `divMenu` benötigt, dem ein HTML-Code zugewiesen wird. Anschließend wird sein `style` verändert.
Aufgerufen wird die Funktion auf einem Link heraus.

Übung 5: Kreieren Sie nun für Ihre Rezeptseiten ein Popup-Menu. Gehen Sie wie folgt vor:

1.: Schreiben Sie in einer Test-HTML-Seite eine Tabelle, die das Erscheinungsbild eines Popups hat, d.h. ein viereckiger Kasten mit einer dünnen Umrahmung. für die dünne Umrahmung verwenden sie die Einpixelgraphik aus dem Verzeichnis `Aufgaben-Materialien/pic`. Ihre Tabelle besteht aus drei Zeilen und drei Zellen, wobei die drei Zellen der ersten und dritten Zeile mit `colspan` miteinander verbunden sind. Sie bilden den Rahmen oben und unten. Die erste und dritte Zelle der zweiten Zeile bildet den Rahmen links und rechts. Für den Rahmen setzen Sie in die entsprechenden Zellen die Einpixelgraphik. Definieren sie ihre Weite für den oberen und unteren Rahmen mit `width=250`.

2.: Modifizieren Sie nun Ihre Funktion `showMenu()` wie folgt: Definieren Sie drei Variablen `tabellenAnfang`, `tabellenInhalt` und `tabellenEnde`. In der Variablen `tabellenAnfang` steht der Teil der in 1 geschriebenen Tabelle bis zu dem <td>-Element, in das der Inhalt geschrieben wird. In der Variablen `tabellenInhalt` definieren Sie die Links zu den einzelnen Rezepten und in die Variable `tabellenEnde` schreiben Sie den Rest der Tabelle. Modifizieren Sie den -Bereich anschließen wie in obigem Beispiel gezeigt über `innerHTML` und `style.visibility`.

3.: Vergessen Sie nicht, nun auch in Ihren Rezept-Dateien einen Link zu setzen, der die Funktion `showMenu()` aufruft. Vergessen Sie ebenfalls nicht, einen -Bereich in den Seiten zu integrieren – möglichst bevor Sie die Funktion aufrufen.

7. Anhang

Welche Eventhandler gibt es?

Folgende Eventhandler treten auf	bei dieser Aktion:
onAboard	Abbruch des Ladevorgangs
onBlur	Feld verliert Fokus
onChange	Feld bekommt neuen Inhalt
onClick	Klick mit der Maus
onDblClick	Doppeltes Anklicken eines Elements
onError	Auftreten eines Fehlers
onFocus	Feld erhält Fokus
onKeyDown	Tastendruck
onKeyPress	Erfolgter Tastendruck
onLoad	Nach Laden einer HTML-Seite
onMouseDown	Drücken der Maus über einem Element
onMouseOut	Verlassen eines Elements mit der Maus
onMouseOver	Überfahren eines Elements mit der Maus
onMouseUp	Loslassen der Maus über einem Element
onMove	Bewegen des Fensters
onReset	Zurücksetzen eines Formulars
onResize	Verändern der Fenstergröße
onSubmit	Absenden eines Formular
onUnload	Verlassen einer HTML-Seite

Wie gestalte ich die Seiten browserunabhängig?
Die hier vorgestellten Seiten funktionieren zum Teil nur im Internetexplorer. Um die Seiten auch im Netscape 4.7 und Netscape 6.x zugänglich zu machen müssen Sie folgende Veränderungen vornehmen: Ändern Sie zunächst in den Rezept-Dateien das wie folgt:

```
<div id="divMenu" style="position:absolute; z-index: 2;"><layer id="layMenu"></layer></div>
```

Das zusätzliche `<layer>`-Element ist für Netscape 4.7. Dieser Browser kennt kein `<div>` und verwendet stattdessen die ähnlich funktionierenden Layer. Microsoft Internet-Explorer und Netscape 6.x hingegen verstehen nur `<div>`, nicht `<layer>`. Achten Sie auf die Reihenfolge der Verschachtelung!

Als nächstes müssen Sie das JavaScript umschreiben und alle drei Browsertypen dabei integrieren. Dies geschieht durch eine `if`-Anweisung:

```
function showMenu(){
   var tabellenAnfang= .....
   var tabellenInhalt= .....
   var tabellenEnde= .....

   if (document.all){  // Microsoft
      document.all.divMenu.innerHTML=tabellenAnfang+tabellenInhalt+tabellenEnde;
      document.all.divMenu.style.visibility="visible";
   }else if (document.getElementById){  //Netscape 6.x
   document.getElementById("divMenu").innerHTML=tabellenAnfang+tabellenInhalt+tabellenEnde;
      document.getElementById("divMenu").style.visibility="visible";
   }else if (document.layers){  //Netscape 4.7
      document.layers[0].document.write(tabellenAnfang+tabellenInhalt+tabellenEnde);
   }
}
```

Die Variablendeklaration ist in allen drei Browsern gleich. Sie steht daher außerhalb der `if`-Anweisung. In der `if`-Anweisung wird zunächst über die Bedingung `document.all` abgefragt, ob es sich um den Microsoft Internet-Explorer handelt, dann über die Bedingung `document.getElementsByID`, ob es sich um den neuen Netscape handelt, und schließlich mit `document.layers`, ob es sich um den alten Netscape handelt. Beachten Sie auch hier die Reihenfolge, um Fehler in der Abarbeitung zu verhindern. Die jeweils aufgerufenen Anweisungen sind browserspezifisch.