2 JavaScript-Grundlagen

In diesem Kapitel werden Sie Ihr erstes Script erstellen, abspeichern und ausführen. Danach lernen Sie, wie man Daten in Variablen ablegt und mit diesen weiter arbeitet. Um aber nicht den zweiten Schritt vor dem ersten zu machen, beginnen wir mit einem Programmierbeispiel.

2.1 Hello World!

Seit jeher ist es Tradition, dass das erste Programm in einer neuen Programmiersprache die Welt begrüßt. Da die meisten Programmiersprachen in Englisch gehalten sind, geben wir nun den Schriftzug `Hello World!` auf dem Bildschirm aus. Wenn Sie diese Ausgabe sehen, haben Sie erfolgreich Ihr erstes Programm generiert und ausgeführt. Also frisch ans Werk ...

Starten Sie Ihren Lieblings-Texteditor und tragen Sie die folgenden Zeilen ein.

Listing 2.1: Unser erstes lauffähiges Script

```
<html>
<script language="JavaScript">
  document.write("Hello World!");
</script>
</html>
```

Das Tag `<html>` markiert, wie Sie schon gelernt haben, den HTML-Teil des Dokuments. Damit der Browser später weiß, dass wir ein JavaScript programmiert haben, müssen wir das durch das `<script>`-Tag kenntlich machen. Zwischen `<script>` und `</script>` steht also der eigentliche Quelltext des Scripts. Da wir nichts anderes realisieren wollen, als die Worte `Hello World!` über ein JavaScript in das HTML-Dokument einzufügen, rufen wir in der Klasse `document` (engl. für »Dokument«) die Funktion `write` (engl. für »schreiben«) auf. Dieser übergeben wir dann einfach die beiden Wörter. Was genau Klassen und Funktionen sind, erfahren Sie noch in späteren Kapiteln des Buches. Auch auf die Syntax gehen wir noch genau ein. Bitte glauben Sie uns zuerst, dass die Zeile `document.write("Hello World!");` genau diesen Effekt hat.

Damit wir das Ergebnis ansehen können, müssen wir den Quelltext auf der Festplatte abspeichern. Legen Sie dazu bitte ein Verzeichnis namens `JS` auf Ihrer Festplatte an. Speichern Sie dann unser erstes Script in die Datei `HelloWorld.html`.

Bitte beachten Sie, dass der Quelltext im Textformat abgespeichert werden muss. Wenn Sie ein Script zum Beispiel im Word-Format abspeichern, werden zusätzliche Steuerzeichen abgespeichert, die eine HTML-Datei unbrauchbar machen.

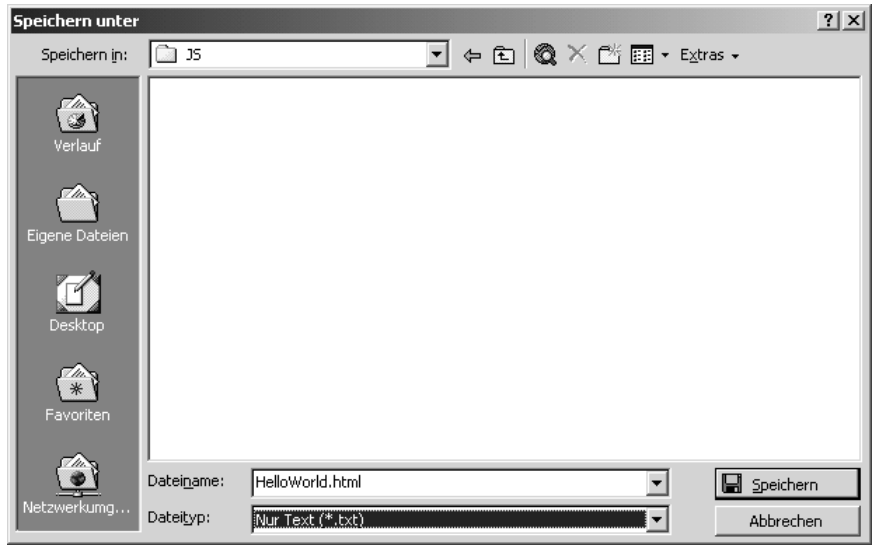

Abbildung 2.1: HTML-Dateien sind immer im Textformat abzuspeichern.

Unser Programm befindet sich nun auf Ihrer Festplatte. Aber wie schaut man sich jetzt das Ergebnis an? Wenn Sie einen Internetbrowser – wie zum Beispiel den Microsoft Internet Explorer oder den Netscape Communicator – installiert haben, können Sie Ihre Datei über den Explorer durch einen simplen Doppelklick öffnen.

Abbildung 2.2: Die Ausgabe unseres Scripts

2.2 Kommentare

Kommen wir nun noch zu einigen Dingen, die eigentlich nicht ganz so viel mit der Programmierung von JavaScript zu tun haben, aber trotzdem für die Verständlichkeit vieler Programme sehr wichtig sind: Kommentare. In JavaScript haben Sie mehrere Möglichkeiten, Kommentare einzufügen. Eine Möglichkeit ist die Benutzung von //, um dem Browser klar zu machen, dass jetzt ein Kommentar folgt. Mit // wird alles, was in der Zeile hinter // steht, vom Browser als Kommentar gewertet. Manchmal hat man aber nicht nur einen kurzen Kommentar, sondern möchte zu einer Funktion oder einem anderen Programmteil ausführlich über mehrere Zeilen beschreiben, wie die Funktion oder der Quelltext funktioniert oder was für Übergabewerte erwartet werden. Für diesen Fall stellen uns die Browser eine Vereinfachung der Kommentare zur Verfügung. Mit der Zeichenfolge /* gibt man dem Browser zu verstehen, dass alle folgenden Zeilen im Quelltext als Kommentar zu behandeln sind und nicht übersetzt werden müssen. Damit aber nicht der gesamte Quellcode als Kommentar behandelt wird, nachdem wir /* eingefügt haben, brauchen wir noch eine Zeichenfolge, die dem Browser wieder klar macht, dass der Kommentar beendet ist und von dieser Position wieder alles übersetzt werden muss. Die Zeichenfolge zum Beenden des Kommentars lautet */. Nach dieser Zeichenfolge wird das Programm wieder übersetzt.

2.3 Nicht JavaScript-fähige Browser

Da JavaScript eine sehr schnell weiterentwickelte Sprache ist, gibt es viele Browser, die noch nicht alle Standards unterstützen, und somit Ihre Programme nicht zwingend auf diesem Browser angezeigt werden. Um zu verhindern, dass ein Browser bei einem JavaScript, das er nicht übersetzen kann, unkontrollierte Dinge ausführt, gibt es Möglichkeiten, vor dem Start des Programms zu überprüfen, welche JavaScript-Version der Browser des Anwenders unterstützt. Um an dieser Stelle aber nicht schon Inhalte aus den nachfolgenden Kapiteln einfach so in den Raum zu stellen, möchten wir hier nur darauf hinweisen, dass im Kapitel der Objekte ausführlich darauf eingegangen wird, wie solche Überprüfungen oder Beschränkungen vorgenommen werden. Eine Ihnen bereits bekannte Art der Festlegung, mit welcher Version von JavaScript ein Programm übersetzt werden soll, ist ja die Angabe des `language="JavaScript"`-Attributes im `<script>`-Tag in HTML. Es ist hier auch möglich, dem `language`-Attribut einen der folgenden Werte zu übergeben:

- `JavaScript1.1`
- `JavaScript1.2`
- `JavaScript1.3`

- JavaScript1.4
- JavaScript1.5

Ein Browser, der zum Beispiel nur JavaScript in der Version 1.1 unterstützt, wird ein Script, das für JavaScript 1.3 geschrieben wurde, ignorieren.

2.4 Variablen

In diesem Abschnitt wollen wir Ihnen zeigen, wie Sie unter JavaScript mit Variablen arbeiten und wie Sie gut lesbaren Quellcode schreiben. Wir gehen in diesem Abschnitt außerdem kurz auf die Schreibweise der JavaScript-Syntax ein.

2.4.1 Variablen deklarieren und initialisieren

In JavaScript haben Sie zwei Möglichkeiten, eine Variable zu deklarieren. Sie können der Variablen einfach einen Wert zuweisen:

```
x = 10
```

oder aber mit dem Schlüsselwort `var` kenntlich machen, dass jetzt eine Variable deklariert wird:

```
var x = 10
```

Beide Arten der Deklaration haben das gleiche Ergebnis: Der Variablen `x` wird der Wert `10` zugewiesen.

Wollen Sie bei der Definition von Variablen Platz sparen und mehrere Variablen hintereinander definieren, gilt folgende Syntax:

```
x = 10; y = 20;
var x = 10; var y = 20;
```

Bei dieser Schreibweise werden alle Befehle voneinander mit ; getrennt.

Folgende Schreibweise ist aber auch möglich:

```
var x = 10, y = 20
```

Diese Schreibweise gilt allerdings nur in Verbindung mit dem Schlüsselwort `var`.

Nun gibt es in JavaScript allerdings mehr als nur Variablen vom Typ Integer. Um beispielsweise ein Array zu initialisieren, bedarf es der folgenden Syntax:

```
new meinArray(10)
```

Mit diesem Befehl wird ein Array mit dem Namen `meinArray` erzeugt, das zehn Elemente beinhaltet, die im Moment allerdings alle noch keine Werte enthalten. Um den einzelnen Elementen des Arrays nun Werte zuzuweisen, verwendet man nachfolgende Syntax:

```
meinArray[0] = "Wert1"
meinArray[9] = "Wert10"
```

Diese Zuweisungen setzen den Wert des ersten Elementes des Arrays (`meinArray[0]`) auf `"Wert1"` und den Wert des letzten Elementes (`meinArray[9]`) auf `"Wert10"`.

Um nun abschließend einen String zu definieren, geht man wie folgt vor:

```
var name = new String("Müller")
```

Die Anweisung definiert die Variable `name` als `String`-Objekt und weist ihr den Wert `"Müller"` zu.

2.4.2 Namenskonventionen

In der folgenden Tabelle finden Sie die in JavaScript 1.3 reservierten Wörter, die Sie nicht für Variablen-, Methoden-, Funktions- oder Objektnamen verwenden dürfen. Alle hier aufgeführten Wörter werden in den bis jetzt erschienenen JavaScript-Versionen als Schlüsselwörter benutzt oder sind schon für spätere Versionen reserviert.

Tabelle 2.1: Reservierte Wörter in JavaScript 1.3

Reservierte Wörter			
abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true

Tabelle 2.1: Reservierte Wörter in JavaScript 1.3 (Forts.)

Reservierte Wörter			
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

In Zeichenketten gibt es auch reservierte Steuerzeichen, die Sie benutzen können um den Text zu formatieren oder um Spezialzeichen wie Anführungszeichen in eine Zeichenkette einzufügen.

Folgende Tabelle gibt eine Übersicht über spezielle Zeichen in Strings:

Tabelle 2.2: Sonderzeichen in Strings

Zeichen	Bedeutung
\b	Backspace
\f	Form Feed (Seitenvorschub)
\n	New Line (Zeilenvorschub)
\r	Carriage Return (Wagenrücklauf)
\t	Tab
\'	Apostrophzeichen (')
\"	Anführungszeichen oben (")
\\	Backslash Zeichen (\ )
\XXX	Ein Zeichen des Latin-1-Zeichensatzes, welches durch bis zu drei oktale Zahlen genauer angegeben wird. Bereich von 0 bis 377, wobei \251 z. B. für das Copyrightzeichen steht.
\xXX	Hexadezimaler Wert
\uXXXX	Unicode-Zeichen

Wichtig ist, dass alle Sonderzeichen mit einem \ beginnen, da sie sonst nicht als Steuerzeichen, sondern als normaler Text behandelt würden.

JavaScript unterscheidet Groß-/Kleinschreibung in Namen für Befehle, Variablen, Objekte usw., wo HTML eine freie Schreibweise zulässt.

2.4.3 Datentypen und Literale

In JavaScript sind folgende Datentypen bekannt:

- Zahlen
- Boolesche Werte
- Strings

Alle diese Datentypen stehen in zwei verschiedenen Varianten zur Verfügung, einmal als einfacher Wert und einmal als Objekte. Im Folgenden beschäftigen wir uns erst mit den Werten an sich und gehen im Kapitel 6 dann genauer auf die Objekte ein.

Bei Zahlen gibt es keine explizite Unterscheidung in ganzzahlige Werte und Kommazahlen. Die Benutzung von ganzen Zahlen in Ihren Programmen bedarf keiner besonderen Notation, Sie können einfach die bekannte Dezimaldarstellung benutzen. Es ist hier allerdings auch möglich, eine alternative Oktal- oder Hexadezimaldarstellung der Zahlen zu benutzen. Um diese Alternativen zu nutzen, müssen Sie eine bestimmte Schreibweise der Zahlen berücksichtigen. Bei Oktalzahlen ist dies die Kenntlichmachung durch eine vorangestellte `0` (Null). Bei Hexadezimalzahlen wird eine `0x` (oder `0X`) vorangestellt. Des Weiteren haben Sie natürlich bei Oktalzahlen nur die Möglichkeit, Ziffern im Bereich von `0-7` einzugeben, im Dezimalsystem sind hier Ziffern von `0-9` erlaubt. Beim Hexadezimalsystem sieht es schon wieder ganz anders aus, hier sind Ziffern von `0-9` und die Buchstaben `A-F` (oder `a-f`) erlaubt.

Dezimal	Oktal (Präfix 0)	Hexadezimal (Präfix 0x)
0	00	0x0
1	01	0x1
2	02	0x2
4	04	0x4
8	010	0x8
10	012	0xA
15	017	0xF
16	020	0x10

Tabelle 2.3: Beispiele für dezimale, oktale und hexadezimale Schreibweisen

Tabelle 2.3: Beispiele für dezimale, oktale und hexadezimale Schreibweisen (Forts.)

Dezimal	Oktal (Präfix 0)	Hexadezimal (Präfix 0x)
20	024	0x14
24	030	0x18
255	0377	0xFF

Eine vollständige Umrechnungstabelle aller Zahlen von 0 bis 255 finden Sie im Anhang.

Anders ist es bei Zahlen, die einen Nachkommaanteil besitzen. Hier gibt es zwei Schreibweisen.

Tabelle 2.4: Reelle Zahlen in Fix- und Fließkommanotation

Fixkomma	Fließkomma in JavaScript	in mathematischer Schreibweise
1.0	1e0.1e1	1*1000,1*101
0.0000002	2e-7.2e-6	2*10-70,2*10-6
-272120000000000	-2.7212e14	-2,7212*1014
-0.00012	-.12e-3	-0,12*10-3

Einfacher haben Sie es hier bei der Verwendung von Strings und booleschen Ausdrücken. Bei diesen Datentypen brauchen Sie nur eine Variable zu erstellen und ihr den gewünschten Wert zuzuweisen. Um also einen String zu erstellen genügt folgende Syntax:

```
var Text = "Diese Variable enthält Text."
```

oder aber für einen booleschen Ausdruck:

```
var JaNein = true
```

Allerdings muss an dieser Stelle darauf hingewiesen werden, dass Stringoperationen nur mit String-Objekten möglich sind. Näheres über die Erstellung und Verwendung von Objekten erfahren Sie im Kapitel Objekte.

2.4.4 Lesbarer Quellcode

Um Quellcode gut lesbar und übersichtlich zu gestalten, sollten Sie Ihre Listings strukturieren und mit vielen Kommentaren versehen.

Listing 2.2: Unstrukturierter Quellcode

```
<html>
<head>
<script>
a=10
b=100
a+=b
document.write("a ist gleich " + a + "<br>")
</script>
</head>
<body>
Die Zahl a wurde mit JavaScript errechnet.
</body>
</html>
```

Einfacher und übersichtlicher ist hier das Lesen des gleichen Quelltextes in strukturierter Form:

Listing 2.3: Strukturierter Quellcode

```
<html>
  <head>
    <script>
    var a=10;
    var b=100;
    a+=b;
    document.write("a ist gleich " + a +"<br>");
    </script>
  </head>
  <body>
  Die Zahl a wurde mit JavaScript errechnet.
  </body>
</html>
```

Um dem Leser des Quellcodes nun noch verständlich zu machen, was hier geschieht, können Sie noch Kommentare in das Listing einfügen:

Listing 2.4: Strukturierter und kommentierter Quellcode

```
<html>
  <head>
    <script>
      //a mit 10 initialisieren
    var a=10;
      //b mit 100 initialisieren
    var b=100;
      //a + b rechnen und das Ergebnis in a speichern
    a+=b;
      //Text ausgeben und in die nächste Zeile gehen
    document.write("a ist gleich " + a + "<br>");
    </script>
  </head>
  <body>
  Die Zahl a wurde mit JavaScript errechnet.
  </body>
</html>
```

Ein Nachteil, wenn Sie so ausführlichen Quelltext schreiben, ist natürlich, dass er viel mehr Speicherplatz benötigt und so längere Zeit beim Laden der Internetseite benötigt. Unser kleines Script hat in der strukturierten und kommentierten Variante einen Speicherplatzbedarf von 409 Byte. Wollen wir nun ein speicheroptimiertes Script im Web ablegen, so empfiehlt es sich die Kommentare und die Strukturierung wieder zu entfernen. Gekürzt würde unser Script etwa so aussehen:

Listing 2.5: Ein stark gekürztes Script

```
<html><head><script>a=10;b=100;a+=b;document.write("a ist gleich
"+a+"<br>");</script></head><body>Die Zahl a wurde mit JavaScript
errechnet.</body></html>
```

Dieses Script hat nun eine Größe von 155 Byte, ist allerdings, wie Sie sicher schon bemerkt haben, in keinster Weise mehr übersichtlich.

2.5 Operatoren

JavaScript verfügt über Zuweisungs-, Vergleichs-, arithmetische, bitweise, Logik-, String- und Spezialoperatoren. Eine wichtige Sache bei Operatoren ist die Schreibweise. Nehmen wir zum Beispiel den Inkrement-Operator (++). Soll ein Wert vor der Zuweisung erhöht werden, verwendet man `++x` (Präfix), wenn erst nach der Zuweisung inkrementiert werden soll, schreibt man `x++` (Postfix). Schauen wir uns ein Beispiel an:

```
var a, b, praef, postf;
prae = 10;
post = 10;
a=++praef;
b=postf++;
```

`praef` und `postf` haben zu Anfang den Wert `10`. `a=++praef;` heißt nichts anderes als: Erhöhe zuerst `praef` um `1` und weise das Ergebnis `a` zu. `a` und `praef` haben somit nach der Anweisung beide den Wert `11`. `b=postf++;` könnte man so übersetzen: Weise `b` den Wert von `postf` zu und erhöhe `postf` dann um `1`. `b` ist somit nach dieser Anweisung `10` und `postf` hat den Wert `11`.

Die nachfolgenden Tabellen geben einen Überblick über die Operatoren:

2.5.1 Arithmetische Operatoren

Tabelle 2.5: Arithmetische Operatoren

Operator	Beschreibung
+	(Addition) addiert zwei Zahlen
++	(Inkrement) erhöht den Wert einer Variablen um eins

Operator	Beschreibung
-	(Subtraktion) subtrahiert zwei Zahlen voneinander
--	(Dekrement) verringert den Wert einer Variablen um eins
*	(Multiplikation) multipliziert zwei Zahlen miteinander
/	(Division) dividiert zwei Zahlen miteinander
%	(Modulo) der ganzzahlige Rest einer Division zweier Zahlen

Tabelle 2.5: Arithmetische Operatoren (Forts.)

2.5.2 String-Operatoren

Operator	Beschreibung
+	(String-Addition) verknüpft zwei Strings miteinander
+=	verknüpft zwei Strings miteinander und schreibt das Ergebnis der Operation in die links stehende String-Variable

Tabelle 2.6: String-Operatoren

2.5.3 Logikoperatoren

Operator	Beschreibung
&&	(logisches UND) gibt dann `true` zurück, wenn beide Operanden den Wert `true` besitzen
||	(logisches ODER) gibt `true` zurück, wenn ein Operand den Wert `true` besitzt
!	(logisches NICHT) negiert den Operanden

Tabelle 2.7: Logikoperatoren

2.5.4 Bitweise Operatoren

Operator	Beschreibung
&	bitweises UND
^	bitweises XODER
|	bitweises ODER
~	bitweises NICHT
<<	Links-Verschiebung

Tabelle 2.8: Bitweise Operatoren

Tabelle 2.8: Bitweise Operatoren (Forts.)

Operator	Beschreibung
>>	Vorzeichen weiter reichende Rechts-Verschiebung
>>>	Null füllende Rechts-Verschiebung

2.5.5 Zuweisungsoperatoren

Tabelle 2.9: Zuweisungsoperatoren

Operator	Beschreibung
=	weist dem ersten Operanden den Wert des zweiten Operanden zu
+=	addiert zwei Zahlen und speichert das Ergebnis in die erste
-=	subtrahiert zwei Zahlen voneinander und speichert das Ergebnis in der ersten Zahl
*=	multipliziert zwei Zahlen miteinander und speichert das Ergebnis in der ersten Zahl
/=	dividiert zwei Zahlen durcheinander und weist das Ergebnis der ersten Zahl zu
%=	errechnet den Modulo von zwei Operanden und weist das Ergebnis dem ersten Operanden zu
&=	führt eine bitweise UND-Verknüpfung aus und speichert das Ergebnis im ersten Operanden
^=	führt eine bitweise XODER-Verknüpfung aus und speichert das Ergebnis im ersten Operanden
|=	führt eine bitweise ODER-Verknüpfung aus und speichert das Ergebnis im ersten Operanden
<<=	bitweise Zuweisung mit Links-Verschiebung
>>=	bitweise Zuweisung mit Rechts-Verschiebung (Vorzeichen-Weiterreichung)
>>>=	bitweise Zuweisung mit Rechts-Verschiebung (Null-füllend)

2.5.6 Vergleichsoperatoren

Tabelle 2.10: Vergleichsoperatoren

Operator	Beschreibung
==	gibt `true` zurück, wenn beide Operanden gleich sind
!=	gibt `true` zurück, wenn beide Operanden nicht gleich sind

Operator	Beschreibung
`===`	gibt `true` zurück, wenn beide Operanden gleich sind und beide Operanden den gleichen Datentyp haben
`!==`	gibt `true` zurück, wenn beide Operatoren nicht gleich sind und/ oder beide Operatoren nicht vom gleichen Datentyp sind
`>`	gibt `true` zurück, wenn der linke Operand größer ist als der rechte
`>=`	gibt `true` zurück, wenn der linke Operand größer oder gleich dem rechten Operanden ist
`<`	gibt `true` zurück, wenn der linke Operand kleiner ist als der rechte
`<=`	gibt `true` zurück, wenn der linke Operand kleiner oder gleich dem rechten Operanden ist

Tabelle 2.10: Vergleichsoperatoren (Forts.)

2.5.7 Spezielle Operatoren

Operator	Beschreibung
`?:`	vereinfachte Schreibweise für eine »`if...then...else`«-Abfrage
`,`	Komma trennt mehrere Elemente voneinander
`delete`	löscht ein Objekt, eine Property eines Objektes oder ein Element in einem Array
`new`	erzeugt eine Instanz eines benutzerdefinierten Objektes oder eines vordefinierten Objektes
`this`	Schlüsselwort, das Sie benutzen können, um auf das aktuelle Objekt zu verweisen
`typeof`	gibt einen String zurück, der den Typ des unausgewerteten Operanden enthält
`void`	definierte Rückgabe für Funktionen, die keinen Wert zurückliefern

Tabelle 2.11: Spezielle Operatoren

2.5.8 Hierarchie der Operatoren

Bezeichnung	Operator / Operatoren
Komma	`,`
Zuweisung	`= += -= *= /= %= <<= >>= >>>= &= ^= |=`
Bedingung	`?:`
logisches ODER	`||`

Tabelle 2.12: Operator-Präzedenz

Tabelle 2.12: Operator-Präzedenz (Forts.)

Bezeichnung	Operator / Operatoren
logisches UND	`&&`
bitweises ODER	`|`
bitweises XODER	`^`
bitweises UND	`&`
Gleichheit	`== !=`
Vergleich	`< <= > >=`
bitweises Verschieben	`<< >> >>>`
Addition / Subtraktion	`+ -`
Multiplikation / Division	`* / %`
Invertierung / Inkrementierung	`! ~ - + ++ -- typeof void delete`
Aufruf	`( )`
erstellende Instanz	`new`
member	`. [ ]`

2.6 JavaScript-Entities

In einigen Fällen kommt es vor, dass man Variablen, die in JavaScript definiert wurden, auch im HTML-Teil einer Seite benötigt. Hier kommen die JavaScript-Entities des Netscape Navigators zum Einsatz. Sie haben richtig gelesen. Die JavaScript-Entities werden nur von Netscape und nicht von Microsoft unterstützt. Um also eine Variable, die im `<script>`-Teil der Seite definiert ist, nutzen zu können, muss die Variable folgendermaßen angesprochen werden: `&{variable};`. In folgendem Beispiel wird die im `<script>`-Tag definierte Variable `breite` an ein Eingabefeld übergeben.

Listing 2.6: Ein Textfeld, das seine Größe über JavaScript-Entities bezieht

```
<html>
  <head>
    <script language="JavaScript">
    var breite = 10;
    </script>
  </head>
  <body>
  <form>
    <input type="text" size=&{breite};>
  </form>
  </body>
</html>
```

Im Browser sollte nun Folgendes zu sehen sein:

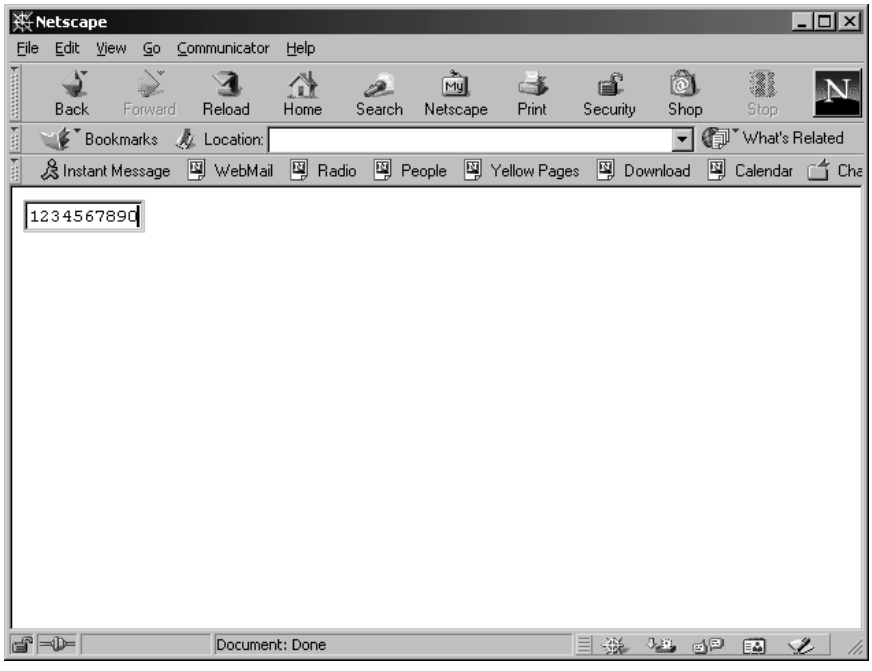

Abbildung 2.3: Ein Textfeld, das seine Breite aus einer JavaScript-Variablen bezieht

Seien Sie nicht verwundert, dass in Ihrem Textfeld noch keine Werte enthalten sind. Wir haben zur Veranschaulichung, dass auch wirklich eine Breite von 10 Zeichen vorhanden ist, die Zahlen eingetragen.

JavaScript-Entities sind sehr hilfreich, wenn Sie eine Seite nur für Netscape-Browser schreiben. Ansonsten sollte man aus Kompatibilitätsgründen versuchen, ohne sie auszukommen.