JScript

Was ist JScript?

JScript ist die Microsoft-Implementierung der Sprachspezifikation ECMA 262 (ECMAScript Edition 3). JScript ist mit nur wenigen kleinen Ausnahmen (für die Abwärtskompatibilität) eine vollständige Implementierung des ECMA-Standards. Diese Übersicht erläutert die ersten Schritte mit JScript.

Verwenden von JScript

JScript ist eine interpretierte objektbasierte Skriptsprache. Obgleich sie über einen geringeren Funktionsumfang als umfangreichere objektorientierte Sprachen wie C++ verfügt, ist JScript für die beabsichtigten Zwecke leistungsfähig genug.

JScript ist weder eine eingeschränkte Version noch eine Vereinfachung einer anderen Sprache (sie ist z. B. nur entfernt und indirekt mit Java verwandt). Sie unterliegt jedoch Einschränkungen. So können Sie keine selbstständigen Anwendungen in ihr schreiben, und sie bietet keine integrierte Unterstützung zum Lesen und Schreiben von Dateien. Darüber hinaus können JScript-Skripten nur bei vorhandenem Interpreter oder "Host" ausgeführt werden, wie z. B. Active Server Pages (ASP), Internet Explorer (IE) oder Windows Script Host (WSH).

JScript ist eine lose typisierte Sprache. Lose typisiert bedeutet, dass Sie den Datentyp von Variablen nicht explizit deklarieren müssen. Tatsächlich geht JScript noch einen Schritt weiter. Es besteht in JScript keine Möglichkeit zur Deklaration von Datentypen. Darüber hinaus führt JScript in vielen Fällen Konvertierungen automatisch nach Bedarf durch. Wenn Sie z. B. eine Zahl einem Element hinzufügen, das aus Text besteht (eine Zeichenfolge), wird die Zahl in Text konvertiert.

Das weitere Benutzerhandbuch bietet eine Übersicht über die JScript-Funktionen. Ausführliche Informationen zur Sprachimplementierung finden Sie im Sprachverzeichnis.

> **Anmerkung** Der Code vieler folgender Beispiele ist ausführlicher und weniger kompakt als der Code, den Sie wahrscheinlich auf realen Webseiten finden. Die verfolgte Absicht ist das Verdeutlichen der Konzepte, nicht die Darstellung einer optimalen Codierung hinsichtlich Prägnanz und Stil. Es ist auf jeden Fall eine gute Sache, Code zu schreiben, den Sie lesen und leicht verstehen können, und zwar sechs Monate, nachdem Sie ihn geschrieben haben.

> **Anmerkung 2:** Das vorliegende Skript bezieht sich zwar auf die ECMA-Script-Variante „Jscript“ von Microsoft, ist aber in weiten Teilen genauso für alle anderen Implementierungen (z. B. in Opera- oder Firefox-Browsern) gültig.

Schreiben von JScript-Code

Wie bei vielen anderen Programmiersprachen auch wird der Code von Microsoft JScript-Programmen im Textformat geschrieben und besteht aus Anweisungen, Blöcken von zusammengehörigen Anweisungen und Kommentaren. In Anweisungen können Sie Variablen, Daten (wie Zeichenfolgen und Zahlen (auch "Literale" genannt)) und Ausdrücke verwenden.

Anweisungen

Ein JScript-Programm ist eine Gruppe von Anweisungen. Eine JScript-Anweisung entspricht einem vollständigen englischen Satz. JScript-Anweisungen kombinieren Ausdrücke so, dass sie einen vollständigen Task ausführen.

Eine Anweisung besteht aus einem oder mehren Ausdrücken, Schlüsselwörtern oder Operatoren (Symbolen). In der Regel wird eine Anweisung auf einer einzelnen Zeile geschrieben, obwohl eine Anweisung auch über zwei oder mehr Zeilen geschrieben werden kann. Es können auch zwei oder mehrere Anweisungen auf derselben Zeile geschrieben werden, indem die Anweisungen durch Semikolon voneinander getrennt werden. Im Allgemeinen beginnt jede neue Zeile mit einer neuen Anweisung. Es wird empfohlen, die Anweisungen ausdrücklich zu beenden. Verwenden Sie dafür das Semikolon (;), das JScript-Endzeichen für Anweisungen. Im Folgenden finden Sie zwei Beispiele für JScript-Anweisungen.

```
einVogel = "Amsel"; // Weisen Sie den Text "Amsel" zur Variable einVogel zu
var heute = new Date(); // Weisen Sie dem heutigen Datum die Variable heute zu
```

Eine von geschweiften Klammern ({}) eingeschlossene Gruppe von JScript-Anweisungen wird als Block bezeichnet. Anweisungen, die in einem Block gruppiert sind, können im Allgemeinen wie eine einzelne Anweisung verwendet werden. Das bedeutet, dass Sie Blöcke an den meisten Stellen verwenden können, an denen JScript eine einzelne Anweisung erwarten würde. Zu den wichtigen Ausnahmen gehören die Kopfzeilen von **for**- und **while**-Schleifen. Beachten Sie, dass die einfachen Anweisungen in einem Block mit einem Semikolon beendet werden, der Block selbst jedoch nicht.

Im Allgemeinen werden Blöcke in Funktionen und Bedingungen verwendet. Beachten Sie, dass, abweichend von C++ und anderen Programmiersprachen, JScript Blöcke nicht als einen neuen Gültigkeitsbereich ansieht; nur Funktionen erstellen einen neuen Gültigkeitsbereich. Im folgenden Beispiel beginnt die erste Anweisung die Definition einer Funktion, die aus einem Block mit fünf Anweisungen besteht. Dem Block folgen drei Anweisungen, die nicht in Klammern eingeschlossen werden. Diese Anweisungen sind kein Block und deshalb auch kein Bestandteil der Funktionsdefinition.

```
function Konvertieren(zoll)  {
   fuss = zoll / 12;  //  Diese fünf Anweisungen stehen in einem Block.
   meilen = fuss / 5280;
   seemeilen = fuss / 6080;
   cm = zoll * 2.54;
   meter = zoll / 39.37;
}
km = meter / 1000;  // Diese drei Anweisungen stehen nicht in einem Block.
kradius = km;
mradius = meilen;
```

Kommentare

Ein einzeiliger JScript-Kommentar beginnt mit zwei Schrägstrichen (//).

```
eineGuteIdee = "Kommentieren Sie Ihren Code sorgfältig.";  //  Dies ist ein
einzeiliger Kommentar.
```

Ein mehrzeiliger JScript-Kommentar beginnt mit einem Schrägstrich und einem Sternchen (/*) und endet mit der umgekehrten Kombination (*/).

```
/*
Dies ist ein mehrzeiliger Kommentar, der das vorherige Codebeispiel erläutert.

Die Anweisung weist der Variablen eineGuteIdee einen
Wert zu. Der in Anführungszeichen eingeschlossene
Wert wird als Literal bezeichnet. Ein Literal enthält
 explizite Informationen, es verweist anders als andere
 Konstrukte nicht indirekt auf Informationen.
(Die Anführungszeichen sind nicht Bestandteil des Literals.)
*/
```

> **Anmerkung** Wenn Sie versuchen, einen mehrzeiligen Kommentar in einen anderen einzubetten, interpretiert JScript den daraus resultierenden mehrzeiligen Kommentar nicht wie erwartet. Das Sternchen und der Schrägstrich (*/), die das Ende des eingebetteten Kommentars markieren, werden als das Ende des gesamten mehrzeiligen Kommentars interpretiert. Dies bedeutet, dass der Text, der dem eingebetteten mehrzeiligen Kommentar folgt nicht auskommentiert wird, sondern als JScript-Code interpretiert wird, wodurch Syntaxfehler erzeugt werden.

Es wird empfohlen, dass Sie alle Kommentare als Blöcke mit einzeiligen Kommentaren schreiben. Auf diese Weise können Sie später große Codesegmente mit einem mehrzeiligen Kommentar auskommentieren.

```
// Ein weiterer mehrzeiliger Kommentar, als Folge von einzeiligen Kommentaren
geschrieben.
// Nachdem die Anweisung ausgeführt wurde, können Sie auf den Inhalt der
Variablen eineGuteIdee
// verweisen, indem Sie den Namen verwenden. Dies wird in der nächsten
Anweisung gezeigt. In dieser
// wird der Variablen eineGuteIdee ein Zeichenfolgenliteral hinzugefügt, um
eine neue Variable zu erstellen.

var erweiterteIdee = eineGuteIdee + "Sie wissen nicht, wann Sie herauskriegen
müssen, wie der Code funktioniert.";
```

Zuweisungen und Gleichheit

Das Gleichheitszeichen (=) wird in JScript-Anweisungen verwendet, um Werte zu Variablen zuzuweisen; es ist ein Zuweisungsoperator. Der linke Operand des Operators = ist stets ein Lvalue. Beispiele für Lvalues sind:

- Variablen,
- Arrayelemente und
- Objekteigenschaften.

Der rechte Operand des Operators = ist stets ein Rvalue. Rvalues können ein willkürlicher Wert eines beliebigen Typs sein, einschließlich des Wertes für einen Ausdruck. Im Folgenden sehen Sie ein Beispiel für eine JScript-Zuweisungsanweisung.

```
eineZahl = 3;
```

Der JScript-Compiler interpretiert diese Anweisung wie folgt: "Weise der Variablen **eineZahl** den Wert 3 zu", oder "**eineZahl** nimmt den Wert 3 an".

Sie müssen den Unterschied zwischen dem Operator = (Zuweisung) und dem Operator == (Gleichheit) verstehen. Wenn Sie zwei Werte vergleichen möchten, um festzustellen, ob diese gleich sind, verwenden Sie zwei Gleichheitszeichen (==). Eine genaue Beschreibung finden Sie unter Steuern des Programmablaufs.

Ausdrücke

Ein JScript-Ausdruck ist ein "Satz" von JScript, den ein JScript-Interpreter zum Erzeugen eines Wertes auswerten kann. Der Wert kann ein beliebiger gültiger JScript-Typ sein (eine Zahl, eine Zeichenfolge, ein Objekt usw.). Die einfachsten Ausdrücke sind Literale. Im Folgenden sehen Sie einige Beispiele für JScript-Literalausdrücke.

```
3.9                       // numerisches Literal
"Hallo!"                  // Zeichenfolgenliteral
false                     // Boolesches Literal
null                      // Nullwert des Literals
{x:1, y:2}                // Objektliteral
[1,2,3]                   // Arrayliteral
function(x){return x*x;}  // Funktionsliteral
```

Kompliziertere Ausdrücken können Variablen, Funktionsaufrufe und weitere Ausdrücke enthalten. Ausdrücke können kombiniert werden, um komplexe Ausdrücke mithilfe der Operatoren zu erstellen. Beispiele für Operatoren sind:

```
+  // Addition
-  // Subtraktion
*  // Multiplikation
/  // Division
```

Im Folgenden sehen Sie einige Beispiele für komplexe JScript-Ausdrücke.

```
var einAusdruck = 3 * (4 / 5) + 6;;
var einZweiterAusdruck = Math.PI * radius * radius;
var einDritterAusdruck = einZweiterAusdruck + "%" + einAusdruck;
var einVierterAusdruck = "(" + einZweiterAusdruck + ") % (" + einAusdruck +
")";
```

JScript-Variablen

In Programmiersprachen wird ein Teil der Daten zum Quantifizieren eines Konzepts verwendet.

```
Wie alt bin ich?
```

In JScript ist eine Variable der Name, den Sie dem Konzept gegeben haben; es stellt den Wert zu einem gegebenen Zeitpunkt dar. Wenn Sie die Variable verwenden, dann drücken Sie die Daten aus, für die diese Variable steht. Beispiel:

```
AnzahlDerVerbleibendenTage = EndeDatum – HeutigesDatum;
```

In mechanischem Sinne können Sie die Variablen zum Speichern, Einfügen und Ändern aller unterschiedlichen Werte verwenden, die in Ihrem Skript angezeigt werden. Erstellen Sie stets sinnvolle Namen für die Variablen. Dies erleichtert das Verständnis zur Funktion Ihrer Skripten für andere.

Deklarieren von Variablen

Wenn eine Variable das erste Mal in Ihrem Skript angezeigt wird, wird sie deklariert. Durch diese erste Erwähnung der Variable wird diese im Speicher erfasst, damit Sie später in Ihrem Skript darauf verweisen können. Deklarieren Sie Variablen stets, bevor Sie sie verwenden. Verwenden Sie dazu das Schlüsselwort **var**.

```
var Anzahl;  // eine einfache Deklaration.
var Anzahl, Menge, Ebene;  // mehrere Deklarationen mit einem einzelnen
Schlüsselwort var.
var Anzahl = 0, Menge = 100;  // Deklaration und Initialisierung der Variable
in einer Anweisung.
```

Wenn Sie die Variable nicht in einer Anweisung **var** deklarieren, nimmt die Variable automatisch den JScript-Wert **undefined** an. Obwohl es nicht sicher ist, so vorzugehen, ist es zulässige JScript-Syntax, wenn das Schlüsselwort **var** aus Ihrer Deklarationsanweisung entfernt wird. Wenn Sie so vorgehen, blendet der JScript-Interpreter den globalen Gültigkeitsbereich der Variable ein. Wenn Sie eine Variable auf der Prozedurebene deklarieren, obwohl Sie nicht möchten, dass diese im globalen Gültigkeitsbereich eingeblendet wird, müssen Sie in diesem Fall das Schlüsselwort **var** in der Deklaration der Variablen verwenden.

Benennen von Variablen

Der Variablenname ist ein Bezeichner. In JScript werden Bezeichner verwendet zum:

- Benennen von Variablen,
- Benennen von Funktionen und
- Bereitstellen von Bezeichnungen für Schleifen.

JScript unterscheidet zwischen Groß- und Kleinschreibung. Dies bedeutet, dass ein Variablenname, wie z. B. *meineZahl* sich vom Variablenname *MEINEZahl* unterscheidet. Variablennamen können beliebig lang sein. Die Regel zum Erstellen von zulässigen Variablennamen ist wie folgt:

- Das erste Zeichen muss ein ASCII-Buchstabe (Groß- oder Kleinbuchstabe) oder ein Unterstrich (_) sein. Beachten Sie, dass eine Zahl nicht als erstes Zeichen verwendet werden kann.
- Für die weiteren Zeichen sind nur Buchstaben, Zahlen oder Unterstriche zulässig.
- Der Variablenname darf kein reserviertes Wort sein.

Einige Beispiele für gültige Variablennamen:

```
_Seitenzahl
Teil9
Anzahl_Elemente
```

Einige ungültige Variablennamen:

```
99Ballons // Beginnt mit einer Zahl.
Smith&Wesson // Ein kaufmännisches Und-Zeichen (&) ist kein gültiges Zeichen
für Variablennamen.
```

Wenn Sie eine Variable deklarieren und initialisieren möchten, ihr jedoch keinen bestimmten Wert zuweisen möchten, dann weisen Sie ihr den JScript-Wert null zu. Beispiel:

```
var bestesAlter = null;
var vielZuAlt = 3 * bestesAlter; // vielZuAlt hat den Wert 0.
```

Wenn Sie eine Variable deklarieren, ohne ihr einen Wert zuzuweisen, existiert sie zwar, hat aber den JScript-Wert undefined. Beispiel:

```
var aktuelleZahl;
var endgültigeZahl = 1 * aktuelleZahl; // endgültigeZahl hat den Wert NaN, da
aktuelleZahl den Wert undefined hat.
```

Beachten Sie, dass in JScript der Hauptunterschied zwischen **null** und **undefined** darin liegt, dass sich **null** wie die Zahl 0 verhält, wohingegen **undefined** sich wie das Sonderzeichen **NaN** (Not a Number) verhält. Bei einem Vergleich sind die Werte **null** und **undefined** stets gleich. Sie können eine Variable deklarieren, ohne dass Sie das Schlüsselwort **var** in der Deklaration verwenden müssen, und ihm einen Wert zuweisen. Dies ist eine implizite Deklaration.

```
garKeineZeichenfolge = ""; // Die Variable garKeineZeichenfolge ist impliziert
deklariert.
```

Sie können keine Variable verwenden, die noch nie deklariert wurde.

```
var Datenträger = Länge * Breite; // Fehler – Länge und Breite sind noch nicht
vorhanden.
```

Erzwungene Konvertierungen

Der JScript-Interpreter kann lediglich Ausdrücke auswerten, in denen die Datentypen von Operanden gleich sind. Ohne erzwungene Konvertierung würde ein Ausdruck, der versucht, eine Operation auf zwei verschiedenen Datentypen (z. B. eine Zahl und eine Zeichenfolge) durchzuführen, ein fehlerhaftes Ergebnis erzeugen. Dies ist mit JScript nicht der Fall.

JScript ist eine Sprache mit flexibler Typbindung. Das bedeutet, dass ihre Variablen nicht von einem vorbestimmten Typ sind (im Gegensatz zu Sprachen mit strikter Typbindung wie C++). Stattdessen sind JScript-Variablen von einem Typ, der dem Typ des in ihnen enthaltenen Wertes entspricht. Ein Vorteil dieses Verhaltens liegt darin, dass es Ihnen die Flexibilität bietet, einen Wert so zu behandeln, als sei er von einem anderen Typ.

In JScript können Sie Operationen auf Werten unterschiedlicher Typen durchführen, ohne befürchten zu müssen, dass der JScript-Interpreter eine Ausnahme auslöst. Stattdessen ändert (konvertiert) der JScript-Interpreter automatisch einen der Datentypen in den des anderen Wertes und führt anschließend die Operation durch. Beispiele:

Operation	Ergebnis
Addition einer Zahl und einer Zeichenfolge	Die Zahl wird in eine Zeichenfolge konvertiert.
Addition eines booleschen Wertes und einer Zeichenfolge	Der boolesche Wert wird in eine Zeichenfolge konvertiert.
Addition einer Zahl und eines booleschen Wertes	Der boolesche Wert wird in eine Zahl konvertiert.

Betrachten Sie das folgende Beispiel.

```
var x = 2000;      // Eine Zahl.
var y = "Hallo";   // Eine Zeichenfolge.
x = x + y;         // Die Zahl wird in eine Zeichenfolge umgewandelt.
document.write(x); // Gibt 2000Hallo aus.
```

Verwenden Sie die parseInt-Methode, um eine Zeichenfolge explizit in eine Ganzzahl zu konvertieren. Verwenden Sie die parseFloat-Methode, um eine Zeichenfolge explizit in eine Zahl zu konvertieren. Beachten Sie, dass Zeichenfolgen zum Zweck des Vergleichs automatisch in äquivalente Zahlen konvertiert werden, jedoch für die Addition (Verkettung) Zeichenfolgen bleiben.

JScript-Datentypen

In JScript gibt es drei primäre Datentypen, zwei zusammengesetzte Datentypen und zwei spezielle Datentypen.

Die primären (einfachen) Datentypen sind:

- Zeichenfolgen (String)
- Zahlen (Number)
- Boolesche Werte (Boolean)

Die zusammengesetzten (Referenz-) Datentypen sind:

- Objekte (Object)
- Array

Spezielle Datentypen sind:

- Null
- Undefined

String-Datentyp

Ein Zeichenfolgenwert besteht aus einer Kette von null oder mehr Unicode-Zeichen (Buchstaben, Ziffern und Satzzeichen), die *aneinander gereiht* sind. Der String-Datentyp dient zur Darstellung von Text in JScript. Sie können Zeichenfolgenliterale in Ihre Skripte einfügen, indem Sie sie in übereinstimmende einfache oder doppelte Anführungszeichen einschließen. Doppelte Anführungszeichen können in Zeichenfolgen enthalten sein, die von einfachen Anführungszeichen umgeben sind, und einfache Anführungszeichen können in Zeichenfolgen enthalten sein, die von doppelten Anführungszeichen umgeben sind. Nachstehend sind einige Beispiele für Zeichenfolgen aufgeführt:

```
"Froh zu sein bedarf es wenig, und wer froh ist, ist ein König!"
'"Macht, dass ihr wegkommt!" brüllte der Techniker.'
"42"
'c'
```

Beachten Sie, dass JScript nicht über einen Typ zur Darstellung von einzelnen Zeichen verfügt. Um ein einzelnes Zeichen in JScript darzustellen, erstellen Sie eine Zeichenfolge, die aus nur einem Zeichen besteht. Eine Zeichenfolge, die null Zeichen enthält (""), ist eine leere Zeichenfolge (mit der Länge Null).

Number-Datentyp

In JScript wird nicht zwischen Ganzzahl- und Fließkommawerten unterschieden; eine JScript-Zahl kann beides sein (intern stellt JScript alle Zahlen als Fließkommawerte dar).

Ganzzahlwerte

Ganzzahlwerte können positive ganze Zahlen, negative ganze Zahlen und 0 sein. Sie können zur Basis 10 (dezimal), zur Basis 8 (oktal) und zur Basis 16 (hexadezimal) dargestellt werden. Die meisten Zahlen in JScript werden als Dezimalzahlen geschrieben. Oktale Ganzzahlen werden bezeichnet, indem ihnen eine führende "0" (null) vorangestellt wird. Sie können nur die Ziffern 0 bis 7 enthalten. Eine Zahl mit einer führenden "0", die die Ziffern "8" und/oder "9" enthält, wird als Dezimalzahl interpretiert.

Hexadezimale ("hex") Ganzzahlen werden bezeichnet, indem ihnen ein führendes "0x" (null und x|X) vorangestellt wird. Sie können nur die Ziffern 0 bis 9 und die Buchstaben A bis F (als Klein- oder Großbuchstaben) enthalten. Mit den Buchstaben A bis F werden die Zahlen 10 bis 15 zur Basis 10 als einzelne Ziffern dargestellt, d. h. 0xF ist äquivalent zu 15, und 0x10 ist äquivalent zu 16.

Sowohl Oktal- als auch Hexadezimalzahlen können negativ sein. Sie können jedoch weder Nachkommastellen besitzen noch in wissenschaftlicher (exponentieller) Notation geschrieben werden.

Fließkommawerte

Fließkommawerte können ganze Zahlen mit einem Nachkommaanteil sein. Zusätzlich können sie in wissenschaftlicher Notation ausgedrückt werden. Das heißt, dass mit einem "e" in Groß- oder Kleinschreibung die Exponentialschreibweise zur Basis 10 dargestellt wird. JScript stellt Zahlen unter Verwendung des 8-Byte-Fließkommastandards IEEE 754 für die numerische Darstellung dar. Dies bedeutet, dass Sie Zahlen schreiben können, die so groß sind wie $\pm 1.7976931348623157 \times 10^{308}$ oder so klein wie $\pm 5 \times 10^{-324}$. Eine Zahl, die mit einer einzelnen "0" beginnt und ein Dezimalkomma enthält, wird als dezimale Fließkommazahl interpretiert.

Beachten Sie, dass eine Zahl, die mit "0x" oder "00" beginnt und ein Dezimalkomma enthält, einen Fehler erzeugt. Hier sind einige Beispiele für JScript-Zahlen.

Zahl	Beschreibung	Dezimales Äquivalent
,0001; 0,0001; 1e-4; 1,0e-4	Vier äquivalente Fließkommazahlen.	0,0001
3,45e2	Eine Fließkommazahl.	345
42	Eine Ganzzahl.	42
0378	Eine Ganzzahl. Obwohl sie wie eine Oktalzahl aussieht (sie beginnt mit einer Null), ist 8 keine gültige Oktalziffer; daher wird die Zahl als Dezimalzahl behandelt.	378
0377	Eine oktale Ganzzahl. Beachten Sie, dass sich der Wert dieser Zahl erheblich von dem der oben genannten Zahl unterscheidet, obwohl die Zahl nur um eins niedriger zu sein scheint.	255
0,0001	Eine Fließkommazahl. Obwohl diese Zahl mit einer Null beginnt, handelt es sich nicht um eine Oktalzahl, da sie ein Dezimalkomma enthält.	0,0001
00,0001	Dies stellt einen Fehler dar. Die beiden führenden Nullen markieren die Zahl als Oktalzahl, aber Oktalzahlen dürfen keine Nachkommakomponente besitzen.	N/A (Compilerfehler)
0Xff	Eine hexadezimale Ganzzahl.	255
0x37CF	Eine hexadezimale Ganzzahl.	14287
0x3e7	Eine hexadezimale Ganzzahl. Beachten Sie, dass das 'e' nicht als Potenzierung behandelt wird.	999
0x3,45e2	Dies stellt einen Fehler dar. Hexadezimalzahlen dürfen keine Nachkommastellen besitzen.	N/A (Compilerfehler)

Zusätzlich enthält JScript Zahlen mit speziellen Werten. Diese lauten wie folgt:

- NaN (Not a Number). Dieser Wert wird verwendet, wenn eine mathematische Operation auf falschen Daten durchgeführt wird, z. B. auf Zeichenfolgen oder dem Wert undefined.
- Plus-Unendlich. Dieser Wert wird verwendet, wenn eine positive Zahl zu groß ist, um in JScript dargestellt zu werden.
- Minus-Unendlich. Dieser Wert wird verwendet, wenn eine negative Zahl zu klein ist, um in JScript dargestellt zu werden.
- Plus und minus 0. JScript unterscheidet zwischen plus und minus Null.

Boolean-Datentyp

Während die Datentypen String und Number eine praktisch unbegrenzte Anzahl verschiedener Werte haben können, kann der Boolean-Datentyp nur zwei Werte haben. Dies sind die Literale **true** und **false**. Bei einem booleschen Wert handelt es sich um einen Wahrheitswert — er drückt die Gültigkeit einer Bedingung aus (gibt an, ob die Bedingung erfüllt ist oder nicht).

Vergleiche, die Sie in Ihren Skripten durchführen, haben stets ein Ergebnis vom Typ Boolean. Betrachten Sie die folgende JScript-Codezeile.

```
y = (x == 2000);
```

Hier wird der Wert der Variablen x darauf getestet, ob er gleich der Zahl 2000 ist. Wenn ja, ist das Ergebnis des Vergleichs der boolesche Wert **true**. Dieser wird der Variablen y zugewiesen. Wenn x nicht gleich 2000 ist, so ist das Ergebnis des Vergleichs der boolesche Wert **false**.

Boolesche Werte sind besonders in Steuerstrukturen von Nutzen. Hier kombinieren Sie einen Vergleich, der einen booleschen Wert erstellt, direkt mit einer Anweisung, die diesen Wert verwendet. Betrachten Sie das folgende JScript-Codebeispiel.

```
if (x == 2000)
    z = z + 1;
else
    x = x + 1;
```

Die **if / else**-Anweisung in JScript führt eine Aktion durch, wenn ein boolescher Wert **true** ist (in diesem Fall z = z + 1), und eine alternative Aktion, wenn der boolesche Wert **false** ist (x = x + 1).

Sie können einen beliebigen Ausdruck als Vergleichsausdruck verwenden. Jeder Ausdruck, der 0, null, undefined oder eine leere Zeichenfolge ergibt, wird als **false** interpretiert. Ein Ausdruck, der einen beliebigen anderen Wert ergibt, wird als **true** interpretiert. Beispielsweise könnten Sie einen Ausdruck wie den folgenden verwenden:

```
if (x = y + z) // Dies führt nicht zu dem erwarteten Ergebnis - siehe unten!
```

Beachten Sie, dass die oben genannte Zeile **nicht** überprüft, ob x gleich y + z ist, da nur ein einzelnes Gleichheitszeichen (Zuweisung) verwendet wird. Stattdessen weist der oben genannte Code den Wert von y + z der Variablen x zu und überprüft anschließend, ob das Ergebnis des gesamten Ausdrucks (der Wert von x) null ist. Um zu überprüfen, ob x gleich y + z ist, verwenden Sie den folgenden Code.

```
if (x == y + z) // Dies unterscheidet sich von dem oben genannten Code!
```

Weitere Informationen zu Vergleichen finden Sie unter Steuern des Programmablaufs.

Null-Datentyp

Der **null**-Datentyp hat nur einen Wert in JScript: null. Das null-Schlüsselwort darf nicht als Name einer Funktion oder Variablen verwendet werden.

Eine Variable, die null enthält, enthält "keinen Wert" oder "kein Objekt". Anders gesagt, sie enthält keine gültige Zahl, keine gültige Zeichenfolge, keinen gültigen booleschen Wert, kein gültiges Array und kein gültiges Objekt. Sie können den Inhalt einer Variablen löschen (ohne die Variable selbst zu löschen), indem Sie ihr den Wert null zuweisen.

Beachten Sie, dass null in JScript nicht das Gleiche wie 0 ist (wie dies in C und C++ der Fall ist). Beachten Sie außerdem, dass der **typeof**-Operator in JScript bei null-Werten meldet, dass sie vom Typ **Object** sind, nicht vom Typ null. Dieses potenziell irreführende Verhalten wird aus Gründen der Abwärtskompatibilität bereitgestellt.

Undefined-Datentyp

Der Wert **undefined** wird zurückgegeben, wenn Sie eines der folgenden Elemente verwenden:

- Eine Objekteigenschaft, die nicht vorhanden ist.
- Eine Variable, die deklariert wurde, der aber noch nie ein Wert zugewiesen wurde.

Beachten Sie, dass Sie nicht auf das Vorhandensein einer Variablen testen können, indem Sie sie mit undefined vergleichen, obwohl Sie überprüfen können, ob sie vom Typ undefined ist. Nehmen wir im folgenden Codebeispiel an, dass der Programmierer testen möchte, ob die Variable **x** deklariert wurde:

```
// Diese Methode funktioniert nicht
if (x == undefined)
    // beliebige Anweisung

// Diese Methode funktioniert ebenfalls nicht - es muss eine Überprüfung auf
// die Zeichenfolge "undefined" durchgeführt werden
if (typeof(x) == undefined)
    // beliebige Anweisung

// Diese Methode funktioniert
if (typeof(x) == "undefined")
    // beliebige Anweisung
```

Betrachten Sie den Vergleich des undefined-Wertes mit null.

```
einObjekt.prop == null;
```

Dieser Vergleich ist **true**, wenn eine der folgenden Bedingungen erfüllt ist:

- Die Eigenschaft einObjekt.prop enthält den Wert null.
- Die Eigenschaft einObjekt.prop ist nicht vorhanden.

Um zu überprüfen, ob eine Objekteigenschaft vorhanden ist, können Sie den neuen **in**-Operator verwenden:

```
if ("prop" in einObjekt)
    // einObjekt besitzt die Eigenschaft 'prop'
```

JScript-Operatoren

JScript verfügt über eine Reihe von Operatoren, wie arithmetische, logische, bitweise Operatoren und Zuweisungsoperatoren. Des Weiteren gibt es einige sonstige Operatoren, die nicht kategorisiert werden können.

Arithmetische Operatoren

Beschreibung	Symbol
Unäre Negation	-
Inkrement	++
Dekrement	—
Multiplikation	*
Division	/
Modulo	%
Addition	+
Subtraktion	-

Logische Operatoren

Beschreibung	Symbol
Logisches NOT	!
Kleiner als	<
Größer als	>
Kleiner oder gleich	<=
Größer oder gleich	>=
Gleichheit	==
Ungleichheit	!=
Logisches AND	&&
Logisches OR	||
Bedingung (ternär)	?:
Komma	,
Strikte Gleichheit	===
Strikte Ungleichheit	!==

Bitweise Operatoren

Beschreibung	Symbol
Bitweises NOT	~
Bitweises Linksschieben	<<
Bitweises Rechtsschieben	>>
Vorzeichenloses bitweises Rechtsschieben	>>>
Bitweises AND	&
Bitweises XOR	^
Bitweises OR	|

Zuweisungsoperatoren

Beschreibung	Symbol
Zuweisung	=
Verbundzuweisung	*OP*=

Sonstige Operatoren

Beschreibung	Symbol
delete	delete
typeof	typeof
void	void
instanceof	instanceof
new	new
in	in

Der Unterschied zwischen == (Gleichheit) und === (Strikte Gleichheit) besteht darin, dass der Gleichheitsoperator Werte verschiedener Typen konvertiert, bevor die Gleichheitsüberprüfung durchgeführt wird. Beispielsweise ergibt ein Vergleich der Zeichenfolge "1" mit der Zahl 1 true. Der *Strikte Gleichheit*-Operator konvertiert dagegen Werte nicht in andere Typen. Daher ergibt ein Vergleich der Zeichenfolge "1" mit der Zahl 1, dass die Werte ungleich sind.

Einfache Zeichenfolgen, Zahlen und boolesche Werte werden mit ihrem Wert verglichen. Wenn sie den gleichen Wert aufweisen, ergibt ein Vergleich, dass sie gleich sind. Objekte (einschließlich der Objekte **Array**, **Function**, **String**, **Number**, **Boolean**, **Error**, **Date** und **RegExp**) werden mit ihrem Verweis verglichen. Selbst wenn zwei Variablen dieser Typen den gleichen Wert haben, ergibt ein Vergleich nur dann true, wenn sie auf exakt dasselbe Objekt verweisen.

Beispiele:

```
// Zwei einfache Zeichenfolgen mit dem gleichen Wert.
var Zeichenfolge1 = "Hallo";
var Zeichenfolge2 = "Hallo";

// Zwei String-Objekte mit dem gleichen Wert.
var StringObjekt1 = new String(Zeichenfolge1);
var StringObjekt2 = new String(Zeichenfolge2);

// Dies ergibt true.
if (Zeichenfolge1 == Zeichenfolge2)
     // beliebige Anweisung (diese wird ausgeführt)

// Dies ergibt false.
if (StringObjekt1 == StringObjekt2)
    // beliebige Anweisung (diese wird nicht ausgeführt)

// Um den Wert von String-Objekten zu vergleichen,
// verwenden Sie die Methoden toString() oder valueOf().
if (StringObjekt1.valueOf() == StringObjekt2)
     // beliebige Anweisung (diese wird ausgeführt)
```

Operatorvorrang

Beim Operatorvorrang handelt es sich um eine Reihe von Regeln in JScript. Er steuert die Reihenfolge, in der Operationen beim Auswerten eines Ausdrucks durchgeführt werden. Operationen mit einem höheren Vorrang werden vor solchen mit einem niedrigeren Vorrang durchgeführt. Beispielsweise wird eine Multiplikation vor einer Addition durchgeführt.

Die folgende Tabelle führt die JScript-Operatoren vom höchsten bis zum niedrigsten Vorrang auf. Operatoren mit dem gleichen Vorrang werden von links nach rechts ausgewertet.

Operator	Beschreibung
. [] ()	Feldzugriff, Arrayindizierung, Funktionsaufrufe und Gruppieren von Ausdrücken
++ — - ~ ! typeof new void delete	Unäre Operatoren, Rückgabedatentyp, Objekterstellung, nicht definierte Werte
* / %	Multiplikation, Division, Modulo
+ - +	Addition, Subtraktion, Zeichenfolgenverkettung
<< >> >>>	Bit-Verschiebung
< <= > >= instanceof	Kleiner als, Kleiner oder gleich, Größer als, Größer oder gleich, instanceof (Instanz von)
== != === !==	Gleichheit, Ungleichheit, Strikte Gleichheit und Strikte Ungleichheit
&	Bitweises AND
^	Bitweises XOR
|	Bitweises OR
&&	Logisches AND
||	Logisches OR
?:	Bedingungen
= *OP*=	Zuweisung, Verbundzuweisung
,	Mehrfache Auswertung

Zum Ändern der durch den Operatorvorrang festgelegten Auswertungsreihenfolge werden Klammern verwendet. Dies bedeutet, dass ein Ausdruck in Klammern vollständig ausgewertet wird, bevor sein Wert im Rest des Ausdrucks verwendet wird.

Beispiel:

```
z = 78 * (96 + 3 + 45)
```

In diesem Ausdruck sind fünf Operatoren enthalten: =, *, (), + und ein weiteres +. Gemäß den Regeln des Operatorvorrangs werden sie in der folgenden Reihenfolge ausgewertet: (), +, +, *, =.

1. Die Auswertung des Ausdrucks in den Klammern findet zuerst statt. In diesen Klammern befinden sich zwei Additionsoperatoren. Da beide Additionsoperatoren den gleichen Rang besitzen, werden sie von links nach rechts ausgewertet. 96 und 3 werden als Erstes addiert. Anschließend wird 45 zu dieser Summe addiert. Das Ergebnis ist der Wert 144.
2. Als Nächstes findet die Multiplikation statt. 78 wird mit 144 multipliziert. Das Ergebnis ist der Wert 11232.
3. Als Letztes findet die Zuweisung statt. 11232 wird z zugewiesen.

Steuern des Programmablaufs

Normalerweise werden Anweisungen in einem JScript-Skript nacheinander in der Reihenfolge ausgeführt, in der sie geschrieben wurden. Diese so genannte sequenzielle Ausführung ist die Standardrichtung des Programmablaufs.
Bei einer Alternative zur sequenziellen Ausführung wird der Programmablauf an einen anderen Teil des Skripts übertragen. Dies bedeutet, dass anstelle der nächsten Anweisung in der Sequenz eine andere Anweisung ausgeführt wird.
Damit ein Skript von Nutzen ist, muss diese Übertragung der Steuerung auf logische Weise erfolgen. Die Übertragung der Programmsteuerung wird auf der Basis einer Entscheidung durchgeführt, deren Ergebnis eine Wahrheitsanweisung ist (und den booleschen Wert **true** oder **false** zurückgibt). Sie erstellen einen Ausdruck und testen dann, ob das Ergebnis **true** ist. Es gibt zwei Hauptarten von Programmstrukturen, mit denen dies erreicht werden kann.

Die erste ist die Auswahlstruktur. Sie wird verwendet, um alternative Richtungen des Programmablaufs anzugeben und so eine Abzweigung im Programm zu erstellen (wie die Gabelung einer Straße). In JScript stehen vier Auswahlstrukturen zur Verfügung.

- Die Einzelauswahlstruktur (**if**).
- Die Zweifachauswahl-Struktur (**if/else**).
- Der ternäre Inlineoperator **?:**.
- Die Mehrfachauswahl-Struktur (**switch**).

Die zweite Art von Programmsteuerstruktur ist die Wiederholungsstruktur. Sie wird verwendet, um anzugeben, dass eine Aktion wiederholt werden soll, solange eine bestimmte Bedingung **true** bleibt. Wenn die Bedingungen der Steueranweisung erfüllt sind (in der Regel nach einer bestimmten Anzahl von Iterationen), wird die Steuerung an die nächste Anweisung nach der Wiederholungsstruktur übergeben. In JScript stehen vier Wiederholungsstrukturen zur Verfügung.

- Der Ausdruck wird am Anfang der Schleife getestet (**while**).
- Der Ausdruck wird am Ende der Schleife getestet (**do/while**).
- Die Schleife operiert auf jeder Eigenschaft eines Objekts (**for/in**).
- Durch einen Zähler gesteuerte Wiederholung (**for**).

Durch Schachteln und Stapeln von Auswahl- und Wiederholungssteuerstrukturen können Sie relativ komplexe Skripte erstellen.

Eine dritte Form des strukturierten Programmablaufs wird von der Ausnahmebehandlung bereitgestellt, die in diesem Dokument jedoch nicht behandelt wird.

Bedingungsanweisungen

JScript unterstützt die Bedingungsanweisungen **if** und if…else. In **if**-Anweisungen wird eine Bedingung getestet. Besteht die Bedingung den Test, wird der relevante JScript-Code ausgeführt. Bei der **if...else**-Anweisung wird ein anderer Code ausgeführt, wenn die Bedingung den Test nicht besteht. Die einfachste Form einer **if**-Anweisung kann vollständig in einer Zeile geschrieben werden. Wesentlich häufiger sind jedoch mehrzeilige **if**- und **if...else**-Anweisungen.

Die folgenden Beispiele zeigen Syntaxkonstrukte, die mit den **if**- und **if...else**-Anweisungen verwendet werden können. Das erste Beispiel zeigt die einfachste Form eines booleschen Testes. Falls das Element in Klammern mit **true** ausgewertet wird (oder erzwungenermaßen **true** wird) (und nur dann), erfolgt die Ausführung der Anweisung bzw. des Anweisungsblockes hinter **if**.

```
// Die Zerschmeissen()-Funktion wird an einer anderen Stelle im Code definiert.
// Boolescher Test, ob neuesSchiff wahr ist.
if (neuesSchiff)
   Zerschmeissen(champagnerFlasche,rumpf);

// Beim folgenden Beispiel misslingt der Test, wenn nicht beide Bedingungen
wahr sind.
if (schale.color == "dunkel gelb" && schale.textur == "große und kleine
Falten")
{
   dieAntwort = ("Ist es eine Crenshaw-Melone?");
}

// Beim nächsten Beispiel ist der Test erfolgreich, wenn mindestens eine von
beiden Bedingungen wahr ist.
var dieReaktion = "";
if ((Wochentag == "Samstag") || (Wochentag == "Sonntag"))
{
   dieReaktion  = ("Ich bin am Strand!");
}
else
   dieReaktion = ("Ich muss zur Arbeit!");}
}
```

Bedingter (ternärer) Operator

JScript unterstützt auch eine implizite Form für die Bedingung. Diese verwendet ein Fragezeichen nach der zu testenden Bedingung (anstelle des Wortes **if** vor der Bedingung). Es werden auch zwei Alternativen angegeben, von denen eine verwendet wird, wenn die Bedingung erfüllt wird. Die andere wird verwendet, wenn dem nicht so ist. Diese Alternativen müssen durch einen Doppelpunkt getrennt werden.

```
var stunden = "";

// Mithilfe des folgenden Codes wird die Stundenangabe in Stunden in die US-
Schreibweise
// (mit AM und PM) konvertiert.

stunden += (dieStunde >= 12) ? (dieStunde - 12) + " PM" : dieStunde + " AM";
```

Wenn mehrere Bedingungen zusammen getestet werden sollen und Sie wissen, dass eine von ihnen den Test mit größerer Wahrscheinlichkeit besteht oder nicht besteht als die anderen, können Sie mithilfe eines Features namens 'Kurzschlussauswertung' die Ausführung des Skripts beschleunigen. Wenn JScript einen logischen Ausdruck auswertet, werden nur so viele Unterausdrücke ausgewertet, wie zur Erzielung eines Ergebnisses erforderlich sind.

Wenn Sie beispielsweise einen **And**-Ausdruck wie ((x == 123) && (y == 42)) haben, überprüft JScript als Erstes, ob x gleich 123 ist. Wenn dies nicht der Fall ist, kann der gesamte Ausdruck nicht **true** sein, selbst wenn y gleich 42 ist. Daher wird der Test für y nie durchgeführt, und JScript gibt den Wert **false** zurück.

Wenn nur eine von mehreren Bedingungen **true** sein muss (unter Verwendung des ||-Operators), wird auf ähnliche Weise das Testen beendet, sobald eine der Bedingungen den Test besteht. Dies ist dann effektiv, wenn die zu testenden Bedingungen die Ausführung von Funktionsaufrufen oder andere komplexe Ausdrücke beinhalten. Daher sollten Sie, wenn Sie **Or**-Ausdrücke schreiben, diejenigen Bedingungen an den Anfang setzen, die mit größter Wahrscheinlichkeit **true** sind. Wenn Sie **And**-Ausdrücke schreiben, sollten Sie diejenigen Bedingungen an den Anfang setzen, die mit größter Wahrscheinlichkeit **false** sind.

Einer der Vorteile, die Ihnen das Entwerfen Ihres Skripts auf diese Weise bietet, besteht darin, dass im folgenden Beispiel **danach()** nicht ausgeführt wird, wenn **zuerst()** entweder 0 oder **false** zurückgibt.

```
if ((zuerst() == 0) || (danach() == 0))
   // Beliebiger Code
}
```

Schleifen

Es gibt mehrere Möglichkeiten, eine Anweisung oder einen Anweisungsblock wiederholt auszuführen. Im Allgemeinen wird die wiederholte Ausführung *Schleife* oder *Iteration*. genannt. Eine Iteration ist die einmalige Ausführung einer Schleife. Diese wird normalerweise durch den Test einer Variable gesteuert, deren Wert sich bei jedem Ausführen der Schleife ändert. JScript unterstützt viele Typen von Schleifen: for-Schleifen, for...in-Schleifen, while-Schleifen und do...while-Schleifen.

for-Schleifen

Die **for**-Anweisung gibt eine Zählervariable, eine Testbedingung und eine Aktion an, die den Zähler aktualisiert. Vor jeder Iteration der Schleife wird die Bedingung getestet. Ist der Test erfolgreich, so wird der Code in der Schleife ausgeführt. Ist der Test nicht erfolgreich, so wird der Code in der Schleife nicht ausgeführt, und das Programm wird in der ersten Codezeile unmittelbar nach der Schleife fortgesetzt. Nach der Ausführung der Schleife wird die Zählervariable aktualisiert, ehe der nächste Durchlauf beginnt.

Sollte die Bedingung für die Schleife nie erfüllt werden, wird die Schleife zu keiner Zeit ausgeführt. Wird die Testbedingung stets erfüllt, entsteht eine Endlosschleife. Während ersteres in manchen Fällen wünschenswert ist, so gilt dies nicht für letzteres. Achten Sie daher bei Programmierung der Schleifenbedingung sorgfältig darauf, dass keine Endlosschleife entsteht.

```
/*
Der Aktualisierungsausdruck ("izaehler++" in den folgenden Beispielen) wird am
Ende der Schleife
ausgeführt, nachdem der Anweisungsblock, der den Rumpf der Schleife ausmacht,
ausgeführt wurde
und bevor die Bedingung getestet wird.
*/

var wieWeit = 10;  // Begrenzt die Schleife auf 10 Durchläufe.

var summe = new Array(wieWeit);  // Erstellt ein Array mit Namen "summe" und 10
Elementen (Indexwerte von 0 bis 9).
var dieSumme = 0;
summe[0] = 0;

for(var izaehler = 0; izaehler < wieWeit; izaehler++)  {        // Zählt in
diesem Fall von 0 bis 9.
dieSumme += izaehler;
summe[izaehler] = dieSumme;
}

var neueSumme = 0;
for(var izaehler = 0; izaehler > wieWeit; izaehler++)  {        // Wird niemals
ausgeführt, da izaehler nicht größer als wieWeit ist
neueSumme += izaehler;
}
```

```
var summe = 0;
for(var izaehler = 0; izaehler >= 0; izaehler++)  {        // Dies ist eine
Endlosschleife.
summe += izaehler;
}
```

for...in-Schleifen

JScript bietet eine besondere Schleife zum Durchlaufen aller benutzerdefinierten Eigenschaften eines Objekts, oder aller Elemente eines Arrays. Der Schleifenzähler in einer **for...in**-Schleife ist zudem eine Zeichenfolge und keine Zahl. Er enthält den Namen der aktuellen Eigenschaft oder den Index des aktuellen Arrayelements.

Das folgende Codebeispiel sollte in Internet Explorer ausgeführt werden, da es die **alert**-Methode verwendet, die nicht Teil von JScript ist.

```
// Erstellt ein Objekt mit einigen Eigenschaften
var meinObjekt = new Object();
meinObjekt.Name = "Klaus";
meinObjekt.Alter = "22";
meinObjekt.Telefon = "555 1234";

// Zählt alle Eigenschaften des Objekts auf (durchläuft sie)
for (prop in meinObjekt)
{
    // Hierdurch wird "Der Wert der Eigenschaft 'Name' lautet Klaus" usw.
angezeigt.
    window.alert("Der Wert der Eigenschaft '" + prop + "' lautet " +
meinObjekt[prop]);
}
```

Obwohl **for...in**-Schleifen den **For Each...Next**-Schleifen von VBScript ähneln, funktionieren sie nicht auf die gleiche Weise. Die **for...in**-Schleife von JScript iteriert über Eigenschaften von JScript-Objekten. Die **For Each...Next**-Schleife von VBScript iteriert über Elemente in einer Kollektion. Um Auflistungen in JScript zu durchlaufen, müssen Sie das **Enumerator**-Objekt verwenden. Obwohl einige Objekte, z. B. diejenigen in Internet Explorer, sowohl die **For Each...Next**-Schleifen von VBScript als auch die **for...in**-Schleifen von JScript unterstützen, ist dies bei den meisten Objekten nicht der Fall.

while-Schleifen

Eine **while**-Schleife ist einer **for**-Schleife ähnlich. Im Gegensatz zu dieser verfügt die **while**-Schleife über keine integrierte Zählervariable und keinen Aktualisierungsausdruck. Wenn Sie die wiederholte Ausführung einer Anweisung oder eines Anweisungsblocks steuern möchten, jedoch eine komplexere Regel als nur "führe diese Code n-mal aus" benötigen, verwenden Sie eine **while**-Schleife. Das folgende Beispiel verwendet das Objektmodell von Internet Explorer und eine **while**-Schleife, um dem Benutzer eine einfache Frage zu stellen.

```
var x = 0;
while ((x != 42) && (x != null))
{
    x = window.prompt("Wie lautet meine Lieblingszahl?", x);
}

if (x == null)
    window.alert("Sie haben aufgegeben!");
else
    window.alert("Genau - das ist die Antwort auf alle Fragen!");
```

Anmerkung: Da **while**-Schleifen nicht über explizit integrierte Zählervariablen verfügen, sind sie anfälliger für Endlosschleifen als andere Typen. Darüber hinaus ist es nur allzu leicht, eine **while**-Schleife zu schreiben, in der die Bedingung tatsächlich nie aktualisiert wird. Dies liegt insbesondere daran, dass es nicht immer einfach ist, herauszufinden, wo und wann die Schleifenbedingung aktualisiert wird. Seien Sie deshalb beim Schreiben von **while**-Schleifen extrem sorgfältig.

Wie oben erwähnt gibt es in JScript auch eine **do...while**-Schleife, die der **while**-Schleife ähnelt. Der Unterschied besteht darin, dass sie garantiert mindestens einmal ausgeführt wird, da die Bedingung am Ende der Schleife getestet wird und nicht am Anfang. Beispielsweise kann die oben genannte Schleife wie folgt neu geschrieben werden:

```
var x = 0;
do
{
    x = window.prompt("Wie lautet meine Lieblingszahl?", x);
} while ((x != 42) && (x != null));

if (x == null)
    window.alert("Sie haben aufgegeben!");
else
    window.alert("Genau - das ist die Antwort auf alle Fragen!");
```

break- und continue-Anweisungen

In Microsoft JScript wird die break-Anweisung verwendet, um die Ausführung einer Schleife anzuhalten, wenn eine bestimmte Bedingung erfüllt ist. (Beachten Sie, dass **break** auch verwendet wird, um einen **switch**-Block zu beenden.) Mit der continue-Anweisung können Sie sofort zur nächsten Iteration springen und den Rest des Codeblocks überspringen. Dabei wird gleichzeitig die Zählervariable aktualisiert, wenn es sich bei der Schleife um eine **for**- oder **for...in**-Schleife handelt.

Das folgende Beispiel baut auf dem vorhergehenden Beispiel auf und verwendet die Anweisungen **break** und **continue** zum Steuern der Schleife.

```
var x = 0;
do
{
    x = window.prompt("Wie lautet meine Lieblingszahl?", x);

    // Wurde die Aktion vom Benutzer abgebrochen? Wenn ja, Schleife anhalten
    if (x == null)
        break;

    // Wurde eine Zahl eingegeben?
    // Wenn ja, ist keine Aufforderung zum Eingeben einer Zahl erforderlich.
    if (Number(x) == x)
        continue;

    // Fordert den Benutzer dazu auf, nur Zahlen einzugeben
    window.alert("Bitte geben Sie nur Zahlen ein!");

} while (x != 42)

if (x == null)
    window.alert("Sie haben aufgegeben!");
else
    window.alert("Genau - das ist die Antwort auf alle Fragen!");
```

JScript-Funktionen

Microsoft JScript-Funktionen führen Aktionen durch; sie können außerdem Werte zurückgeben. Manchmal handelt es sich hierbei um die Ergebnisse von Berechnungen oder Vergleichen. Man nennt Funktionen auch "globale Methoden".

Funktionen kombinieren mehrere Operationen unter einem Namen. Hierdurch können Sie Ihren Code "kompakter" gestalten. Sie können eine Gruppe von Anweisungen schreiben, sie benennen und alle Anweisungen in dieser Gruppe jederzeit ausführen, indem Sie die Gruppe aufrufen und die erforderlichen Informationen übergeben.

Sie übergeben Informationen an eine Funktion, indem Sie die Informationen hinter dem Namen der Funktion in Klammern einschließen. An eine Funktion übergebene Informationen werden Argumente oder Parameter genannt. Einige Funktionen erhalten keine Argumente, während andere ein oder mehrere Argumente erhalten. Bei einigen Funktionen hängt die Anzahl der Argumente davon ab, wie Sie die Funktion verwenden.

JScript unterstützt zwei Arten von Funktionen: die in die Sprache integrierten und die von Ihnen erstellten Funktionen.

Spezielle integrierte Funktionen

Die JScript-Sprache enthält zahlreiche integrierte Funktionen. Einige ermöglichen Ihnen, Ausdrücke und Sonderzeichen zu behandeln, während andere Zeichenfolgen in numerische Werte konvertieren. Eine nützliche integrierte Funktion ist eval(). Diese Funktion wertet gültigen JScript-Code aus, der in Form einer Zeichenfolge übergeben wird. Die **eval()**-Funktion erhält ein Argument (den auszuwertenden Code). Nachfolgend finden Sie ein Beispiel, das diese Funktion verwendet.

```
var einAusdruck = "6 * 9 % 7";
var Gesamt = eval(einAusdruck); // Weist der Variablen Gesamt den Wert 5 zu.
var nochEinAusdruck = "6 * (9 % 7)";
Gesamt = eval(nochEinAusdruck) // Weist der Variablen Gesamt den Wert 12 zu.
// Weist nichtSoGut eine Zeichenfolge zu (beachten Sie die geschachtelten
Anführungszeichen)
var nichtSoGut = eval("'...umgeben von sinkenden Schiffen.'");
```

Weitere Informationen über diese und andere integrierte Funktionen finden Sie im Sprachverzeichnis.

Erstellen eigener Funktionen

Sie können eigene Funktionen erstellen und dort verwenden, wo Sie sie benötigen. Eine Funktionsdefinition enthält eine Funktionsanweisung und einen Block von JScript-Anweisungen.

Die **PruefeDreiZahlen**-Funktion im folgenden Beispiel verwendet als Argumente die Längen der Seiten eines Dreiecks. Aus diesen errechnet sie, ob das Dreieck ein rechtwinkliges Dreieck ist, indem sie überprüft, ob die drei Zahlen den Satz des Pythagoras erfüllen (das Quadrat der Länge der Hypotenuse eines rechtwinkligen Dreiecks ist gleich der Summe der Quadrate der Längen der anderen beiden Seiten). Die **PruefeDreiZahlen**-Funktion ruft eine von zwei anderen Funktionen auf, um diesen Test durchzuführen.

Beachten Sie die Verwendung einer sehr kleinen Zahl ("epsilon") als Testvariable in der Fließkommazahlversion des Tests. Aufgrund von Ungenauigkeiten und Rundungsfehlern in

Fließkommaberechnungen ist es nicht ratsam, einen direkten Vergleich durchzuführen, ob die drei Zahlen den Satz des Pythagoras erfüllen, es sei denn, alle drei Werte sind mit Sicherheit Ganzzahlen. Da ein direkter Test genauer ist, bestimmt der Code in diesem Beispiel, ob ein solcher Test angebracht ist, und führt ihn gegebenenfalls durch.

```
var epsilon = 0,00000000001; // Eine sehr kleine Zahl zum Testen.

// Die Testfunktion für Ganzzahlen.
function Ganzzahlpruefung(a, b, c)
{
   // Der eigentliche Test.
   if ( (a*a) == ((b*b) + (c*c)) )
      return true;

   return false;
} // Ende der Testfunktion für Ganzzahlen.

// Die Testfunktion für Fließkommazahlen.
function Fliesskommapruefung(a, b, c)
{
   // Erstellt die Testzahl.
   var delta = ((a*a) - ((b*b) + (c*c)))

   // Der Test erfordert den absoluten Wert
   delta = Math.abs(delta);

   // Wenn die Differenz kleiner als epsilon ist, ist der Wert sehr nahe dran.
   if (delta < epsilon)
      return true;

   return false;
} // Ende der Testfunktion für Fließkommazahlen.


// Die Funktion zur Dreiecksüberprüfung.
function PruefeDreiZahlen(a, b, c)
{
   // Erstellt eine temporäre Variable für das Vertauschen von Werten
   var d = 0;

   // Als erstes sollte die längste Seite zur Position "a" bewegt werden.

   // Vertauscht gegebenenfalls a und b
   if (b > a)
   {
      d = a;
      a = b;
      b = d;
   }

   // Vertauscht gegebenenfalls a und c
   if (c > a)
   {
      d = a;
      a = c;
      c = d;
   }

   // Testen aller 3 Werte. Sind es Ganzzahlen?
   if (((a % 1) == 0) && ((b % 1) == 0) && ((c % 1) == 0))
   {
      // Falls ja, präzise Überprüfung verwenden.
      return Ganzzahlpruefung(a, b, c);
```

```
   }
   else
   {
      // Falls nein, näherungsweise Überprüfung.
      return Fliesskommapruefung(a, b, c);
   }
} // Ende der Funktion zur Dreiecksüberprüfung.

// Die nächsten drei Anweisungen weisen Beispielwerte für Testzwecke zu.
var seiteA = 5;
var seiteB = 5;
var seiteC = Math.sqrt(50,001);

// Ruft die Funktion auf. Nach dem Aufruf enthält 'Ergebnis' das Ergebnis.
var Ergebnis = PruefeDreiZahlen(seiteA, seiteB, seiteC);
```

JScript-Objekte

JScript-Objekte sind Auflistungen von Eigenschaften und Methoden. Eine Methode ist eine Funktion, die ein Element eines Objekts ist. Eine Eigenschaft ist ein Wert oder eine Gruppe von Werten (in Form eines Arrays oder Objekts), der bzw. die ein Element eines Objekts ist. JScript unterstützt vier Arten von Objekten: intrinsische Objekte, von Ihnen erstellte Objekte, Hostobjekte, die vom Host bereitgestellt werden (z. B. **window** und **document** in Internet Explorer) und Active X-Objekte (externe Komponenten).

Objekte als Arrays

In JScript werden Objekte und Arrays nahezu gleich behandelt. Beiden können willkürliche Eigenschaften zugewiesen werden, und tatsächlich handelt es sich bei Arrays lediglich um eine spezielle Art von Objekt. Der Unterschied zwischen Arrays und Objekten besteht darin, dass Arrays über eine "magische" **length**-Eigenschaft verfügen, während dies bei Objekten nicht der Fall ist. Dies bedeutet Folgendes: Wenn Sie einem Element eines Arrays, das größer als alle anderen Elemente ist, einen Wert zuweisen – beispielsweise **meinArray[100] = "hallo"** –, wird die **length**-Eigenschaft automatisch auf 101 (die neue Länge) aktualisiert wird. Wenn Sie die **length**-Eigenschaft eines Arrays ändern, werden auf ähnliche Weise alle Elemente gelöscht, die nicht mehr zum Array gehören.

Alle Objekte in JScript unterstützen "expando"-Eigenschaften, d. h. Eigenschaften, die zur Laufzeit dynamisch hinzugefügt und entfernt werden können. Diese Eigenschaften können beliebige Namen haben, einschließlich Zahlen. Wenn es sich bei dem Namen der Eigenschaft um einen einfachen Bezeichner handelt, kann er mit einem Punkt nach dem Objektnamen geschrieben werden, z. B. wie folgt:

```
var meinObj = new Object();

// Fügt zwei expando-Eigenschaften hinzu, 'Name' und 'Alter'
meinObj.Name = "Heinz";
meinObj.Alter = 42;
```

Wenn es sich bei dem Namen der Eigenschaft nicht um einen einfachen Bezeichner handelt oder er zum Zeitpunkt der Skripterstellung unbekannt ist, können Sie einen willkürlichen Ausdruck in eckigen Klammern verwenden, um die Eigenschaft zu indizieren. Die Namen aller expando-Eigenschaften in JScript werden in Zeichenfolgen konvertiert, bevor sie zum Objekt hinzugefügt werden.

```
var meinObj = new Object();

// Fügt zwei expando-Eigenschaften hinzu, die nicht in der
// Objekt.Eigenschaft-Syntax geschrieben werden können.
// Die erste enthält ungültige Zeichen (Leerzeichen) und muss daher
// in eckigen Klammern geschrieben werden.
meinObj["kein gueltiger Bezeichner"] = "Dies ist der Wert der Eigenschaft";

// Der Name der zweiten expando-Eigenschaft ist eine Zahl und muss daher
ebenfalls
// in eckige Klammern platziert werden
meinObj[100] = "100";
```

Traditionell erhalten Arrayelemente numerische Indizes, die bei Null beginnen. Diese Elemente interagieren mit der **length**-Eigenschaft. Da jedoch alle Arrays auch Objekte sind, unterstützen sie außerdem expando-Eigenschaften. Beachten Sie jedoch, dass expando-Eigenschaften nicht mit der **length**-Eigenschaft interagieren. Beispiel:

```
// Ein Array mit drei Elementen
var meinArray = new Array(3);

// Fügt Daten hinzu
meinArray[0] = "Hallo";
meinArray[1] = 42;
meinArray[2] = new Date(2000, 1, 1);

// Hierdurch wird 3 angezeigt, die Länge des Arrays
window.alert(meinArray.length);

// Fügt expando-Eigenschaften hinzu
meinArray.expando = "JScript!";
meinArray["noch ein Expando"] = "Windows";

// Hierdurch wird immer noch 3 angezeigt, da die beiden expando-Eigenschaften
// nicht die Länge beeinflussen.
window.alert(meinArray.length);
```

Obwohl JScript multidimensionale Arrays nicht direkt unterstützt, können Sie in Arrayelementen alle Arten von Daten speichern – einschließlich anderer Arrays. Sie können also das Verhalten von multidimensionalen Arrays erzielen, indem Sie Arrays in den Elementen eines anderen Arrays speichern. Der folgende Code erstellt beispielsweise eine Multiplikationstabelle für die Zahlen bis 5:

```
// Ändern Sie diese Zahl, um eine größere Tabelle zu erhalten
var iMaxZahl = 5;
// Schleifenzähler
var i, j;

// Neues Array. iMaxZahl + 1 wird verwendet, da die Zählung in Arrays
// bei Null beginnt und nicht bei 1.
var Multiplikationstabelle = new Array(iMaxZahl + 1);

// Durchlaufen der Schleife für jede Zahl (jede Zeile in der Tabelle)
for (i = 1; i <= iMaxZahl; i++)
{
    // Erstellt die Spalten in der Tabelle
    Multiplikationstabelle[i] = new Array(iMaxZahl + 1);

    // Füllt die Zeile mit den Ergebnissen der Multiplikation
    for (j = 1; j <= iMaxZahl; j++)
    {
        Multiplikationstabelle[i][j] = i * j;
    }
}

window.alert(Multiplikationstabelle[3][4]); // Zeigt 12 an
window.alert(Multiplikationstabelle[5][2]); // Zeigt 10 an
window.alert(Multiplikationstabelle[1][4]); // Zeigt 4 an
```

Erstellen eigener Objekte

Um Instanzen eigener Objekte zu erstellen, müssen Sie zuerst eine Konstruktorfunktion für diese Objekte definieren. Eine Konstruktorfunktion erstellt ein neues Objekt und weist ihm Eigenschaften sowie gegebenenfalls Methoden zu. Das folgende Beispiel definiert beispielsweise eine Konstruktorfunktion für Pastaobjekte. Beachten Sie die Verwendung des **this**-Schlüsselwortes, mit dem auf das aktuelle Objekt verwiesen wird.

```
// Pasta ist ein Konstruktor, der vier Parameter erhält.
function Pasta(Getreide, Breite, Form, enthaeltEi)
{
    // Aus welchem Getreide?
    this.Getreide = Getreide;

    // Breite (Zahl)
    this.Breite = Breite;

    // Querschnitt (Zeichenfolge)
    this.Form = Form;

    // Ist Eigelb als Bindemittel enthalten? (boolescher Wert)
    this.enthaeltEi = enthaeltEi;
}
```

Nachdem Sie einen Objektkonstruktor definiert haben, erstellen Sie Instanzen des Objekts mit dem **new**-Operator.

```
var Spagetti = new Pasta("Weizen", 0,2, "kreisfoermig", true);
var Linguini = new Pasta("Weizen", 0,3, "oval", true);
```

Sie können Eigenschaften zu einer Instanz eines Objekts hinzufügen und so diese Instanz ändern. Jedoch werden diese Eigenschaften nicht Teil der Definition anderer Objekte, die mit dem gleichen Konstruktor erstellt werden. Sie erscheinen auch nicht in anderen Instanzen, es sei denn, Sie fügen sie explizit hinzu. Wenn die zusätzlichen Eigenschaften in allen Instanzen des Objekts erscheinen sollen, müssen Sie sie zur Konstruktorfunktion oder zum Prototypobjekt des Konstruktors hinzufügen (Prototypen werden in der Dokumentation "JScript für Fortgeschrittene" besprochen).

```
// Zusätzliche Eigenschaften für Spagetti.
Spagetti.Farbe = "helles Stroh";
Spagetti.trockenKochzeit = 7;
Spagetti.frischKochzeit = 0,5;

var essSpass = new Pasta("Reis", 3, "flach", false);
// Weder das essSpass-Objekt noch die anderen vorhandenen
// Pastaobjekte besitzen die drei neuen Eigenschaften, die zum
// Spagettiobjekt hinzugefügt wurden.


// Durch Hinzufügen der 'Lebensmittelgruppe'-Eigenschaft zum Pasta-
Prototypobjekt
// steht sie allen Instanzen von Pastaobjekten zur Verfügung,
// einschließlich der bereits erstellten.
Pasta.prototype.Lebensmittelgruppe = "Kohlenhydrate"

// Jetzt enthalten Spagetti.Lebensmittelgruppe, essSpass.Lebensmittelgruppe
usw. alle
// den Wert "Kohlenhydrate".
```

Einfügen von Methoden in die Definition

Sie können Methoden (Funktionen) in die Definition eines Objekts einfügen. Eine Möglichkeit, dies durchzuführen, besteht darin, in die Konstruktorfunktion eine Eigenschaft einzufügen, die auf eine an anderer Stelle definierte Funktion verweist. Das folgende Beispiel erweitert z. B. die oben definierte Pastakonstruktorfunktion und fügt eine **toString**-Methode ein, die aufgerufen wird, wenn der Wert des Objekts angezeigt werden soll.

```
// Pasta ist ein Konstruktor, der vier Parameter erhält.
// Der erste Teil ist der gleiche wie der oben Genannte
function Pasta(Getreide, Breite, Form, enthaeltEi)
{
    // Aus welchem Getreide?
    this.Getreide = Getreide;

    // Breite (Zahl)
    this.Breite = Breite;

    // Querschnitt (Zeichenfolge)
    this.Form = Form;

    // Ist Eigelb als Bindemittel enthalten? (boolescher Wert)
    this.enthaeltEi = enthaeltEi;

    // Hier wird die toString-Methode hinzugefügt (wird weiter unten
definiert).
    // Beachten Sie, dass nach dem Namen der Funktion keine Klammern gesetzt
    // werden; es handelt sich nicht um einen Funktionsaufruf, sondern um einen
    // Verweis auf die Funktion selbst.
    this.toString = PastaToString;
}

// Die eigentliche Funktion zum Anzeigen des Inhalts eines Pastaobjekts.
function PastaToString()
{
    // Gibt die Eigenschaften des Objekts zurück

    return "Getreide: " + this.Getreide + "\n" +
        "Breite: " + this.Breite + "\n" +
        "Form: " + this.Form + "\n" +
        "Ei?: " + Boolean(this.enthaeltEi);
}

var Spagetti = new Pasta("Weizen", 0,2, "kreisfoermig", true);
// Dies ruft toString() auf und zeigt die Eigenschaften
// des Spagettiobjekts an (Internet Explorer erforderlich).
window.alert(Spagetti);
```

Intrinsische Objekte

Microsoft JScript stellt elf intrinsische (oder "integrierte") Objekte bereit. Im Einzelnen sind dies die Objekte **Array**, **Boolean**, **Date**, **Function**, **Global**, **Math**, **Number**, **Object**, **RegExp**, **Error** und **String**. Jedes dieser intrinsischen Objekte besitzt zugehörige Methoden und Eigenschaften, die im Sprachverzeichnis detailliert beschrieben werden. Einige dieser Objekte werden auch im vorliegenden Abschnitt beschrieben.

Array-Objekt

Man kann sich die Indizes eines Arrays als Eigenschaften eines Objekts vorstellen. Auf sie kann mit ihrem numerischen Index verwiesen werden. Beachten Sie, dass einem Array hinzugefügte benannte Eigenschaften nicht über eine Zahl indiziert werden können; sie werden von den Arrayelementen getrennt verwaltet.

Um ein neues Array zu erstellen, verwenden Sie den **new**-Operator und den **Array()**-Konstruktor, wie im folgenden Beispiel.

```
var dieMonate = new Array(12);
dieMonate[0] = "Jan";
dieMonate[1] = "Feb";
dieMonate[2] = "März";
dieMonate[3] = "Apr";
dieMonate[4] = "Mai";
dieMonate[5] = "Jun";
dieMonate[6] = "Juli";
dieMonate[7] = "Aug";
dieMonate[8] = "Sep";
dieMonate[9] = "Okt";
dieMonate[10] = "Nov";
dieMonate[11] = "Dez";
```

Wenn Sie unter Verwendung des **Array**-Schlüsselwortes ein Array erstellen, nimmt JScript eine **length**-Eigenschaft in das Array auf, die die Anzahl der Einträge aufzeichnet. Wenn Sie keine Zahl angeben, wird length auf 0 gesetzt, und das Array enthält keine Einträge. Geben Sie eine Zahl an, so wird length auf diesen Wert gesetzt. Wenn Sie mehr als einen Parameter angeben, werden diese Parameter als Einträge für das Array verwendet. Der length-Eigenschaft wird zusätzlich die Anzahl der Parameter zugewiesen, wie im folgenden Beispiel, das äquivalent zu dem vorhergehenden Beispiel ist.

```
var dieMonate = new Array("Jan", "Feb", "März", "Apr", "Mai", "Jun",
"Jul", "Aug", "Sep", "Okt", "Nov", "Dez");
```

JScript ändert automatisch den Wert von **length**, wenn Sie Elemente zu einem Array hinzufügen, das mit dem **Array**-Schlüsselwort erstellt wurde. Arrayindizes beginnen in JScript immer bei 0 und nicht bei 1. Daher ist der Wert der length-Eigenschaft stets um eins höher als der größte Index im Array.

String-Objekt

In JScript können Sie Zeichenfolgen (und Zahlen) so behandeln, als seien es Objekte. Das string-Objekt verfügt über einige integrierte Methoden, die Sie mit Zeichenfolgen verwenden können. Eine dieser Methoden ist die substring-Methode, die einen Teil der Zeichenfolge zurückgibt. Diese verwendet als Argumente zwei Zahlen.

```
eineZeichenfolge = "0123456789";
var einTeil = eineZeichenfolge.substring(4, 7);  // Setzt einTeil auf "456".
var andererTeil = eineZeichenfolge.substring(7, 4);  // Setzt andererTeil auf
"456".
// Unter Verwendung des vorherigen Beispiels zur Arrayerstellung:
ersterBuchstabe = dieMonate[5].substring(0,1);  // Setzt die Variable
ersterBuchstabe auf "J".
```

Eine weitere Eigenschaft des **String**-Objekts ist die **length**-Eigenschaft. Diese Eigenschaft enthält die Anzahl der Zeichen in der Zeichenfolge (0 bei einer leeren Zeichenfolge). Dies ist ein numerischer Wert, der direkt in Berechnungen verwendet werden kann.

```
var wieLang = "Hallo Welt".length  // Setzt die Variable wieLang auf 10.
```

Math-Objekt

Das **Math**-Objekt verfügt über eine Anzahl von vordefinierten Eigenschaften und Methoden. Die Eigenschaften sind bestimmte Zahlen. Eine dieser Eigenschaften ist der Wert von pi (näherungsweise 3,14159...). Im folgenden Beispiel wird diese Eigenschaft, die **Math.PI**-Eigenschaft, gezeigt.

```
// Die Variable Radius wird deklariert, und dieser wird ein numerischer Wert
zugewiesen.
var Kreisflaeche = Math.PI * Radius * Radius;  // Beachten Sie die
Großschreibung von Math und PI.
```

Eine der integrierten Methoden des **Math**-Objekts ist die Potenzierungsmethode (oder **pow**), die eine Zahl in eine bestimmte Potenz erhebt. Das folgende Beispiel verwendet sowohl pi als auch die Potenzierung.

```
// Diese Formel errechnet das Volumen einer Kugel mit dem angegebenen Radius.
Volumen = (4/3)*(Math.PI*Math.pow(Radius,3));
```

Date-Objekt

Mit dem **Date**-Objekt können willkürliche Datums- und Uhrzeitangaben dargestellt, das aktuelle Systemdatum abgerufen und Differenzen zwischen Datumsangaben berechnet werden. Es verfügt über eine Anzahl von Eigenschaften und Methoden, die vordefiniert sind. Im Allgemeinen stellt das **Date**-Objekt Wochentag, Monat, Tag und Jahr sowie die Zeit in Stunden, Minuten und Sekunden zur Verfügung. Diese Informationen basieren auf der Anzahl der seit dem 1. Januar 1970, 00:00:00.000 GMT verstrichenen Millisekunden. GMT steht für "Greenwich Mean Time"; heute wird eher der Begriff UTC (Universal Coordinated Time, universelle koordinierte Zeit) verwendet, der sich auf die Signale des Weltzeitstandards bezieht. JScript kann Datumsangaben behandeln, die ungefähr im Bereich von 250.000 v. Chr. bis 255.000 n. Chr. liegen.

Verwenden Sie den **new**-Operator, um ein neues **Date**-Objekt zu erstellen. Das folgende Beispiel errechnet die Anzahl der vergangenen und noch verbleibenden Tage im aktuellen Jahr.

```
/*
Dieses Beispiel verwendet das bereits definierte Array der Monatsnamen.
Die erste Anweisung weist das heutige Datum im Format "Tag Monat Datum 00:00:00
Jahr"
der Variablen dasIstHeute zu.
*/
var dasIstHeute = new Date();
```

```
var heute = new Date();  // Heutiges Datum speichern.

// Extrahiert Jahr, Monat und Tag.
var diesesJahr = heute.getFullYear();
var dieserMonat = dieMonate[heute.getMonth()];
var dieserTag = dieserMonat + " " + heute.getDate() + ", " + diesesJahr);
```

Number-Objekt

Microsoft JScript stellt zusätzlich zu den speziellen numerischen Eigenschaften (z. B. **PI**) des **Math**-Objekts weitere Eigenschaften durch das **Number**-Objekt bereit.

Eigenschaft	Beschreibung
MAX_VALUE	Größte mögliche Zahl, ungefähr 1,79E+308; kann positiv oder negativ sein. (Der Wert variiert leicht von System zu System.)
MIN_VALUE	Kleinste mögliche Zahl, ungefähr 2,22E-308; kann positiv oder negativ sein. (Der Wert variiert leicht von System zu System.)
NaN	Spezieller nichtnumerischer Wert, "Not a Number".
POSITIVE_INFINITY	Jeder positive Wert, der größer als die größte positive Zahl (Number.MAX_VALUE) ist, wird automatisch in diesen Wert konvertiert; dargestellt als unendlich.
NEGATIVE_INFINITY	Jeder Wert, der negativer als die kleinste negative Zahl (-Number.MAX_VALUE) ist, wird automatisch in diesen Wert konvertiert; dargestellt als minus unendlich.

Number.NaN ist eine spezielle Eigenschaft, die als "Not a Number" ("keine Zahl") definiert ist. Beispielsweise wird bei einer Division durch Null **NaN** zurückgegeben. Beim Versuch, eine Zeichenfolge zu analysieren, die nicht analysiert werden kann, wird ebenfalls **Number.NaN** zurückgegeben. **NaN** ist bei Vergleichen immer ungleich jeder Zahl und auch ungleich sich selbst. Für den Test, ob das Ergebnis **NaN** vorliegt, dürfen Sie somit keinen Vergleich mit **Number.NaN** durchführen. Verwenden Sie stattdessen die **isNaN()**-Funktion.

Reservierte Wörter von JScript

In JScript gibt es eine Reihe von reservierten Wörtern, die nicht als Bezeichner verwendet werden dürfen. Reservierte Wörter besitzen eine spezielle Bedeutung in der JScript-Sprache, da sie Teil der Sprachsyntax sind. Wenn Sie ein reserviertes Wort verwenden, verursacht dies beim Laden des Skripts einen Kompilierungsfehler.

JScript hat außerdem eine Liste von zukünftigen reservierten Wörtern. Diese Wörter sind derzeit noch nicht Teil der JScript-Sprache; sie sind jedoch für eine zukünftige Verwendung reserviert.

Reservierte Wörter

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	in	this	void
continue	false	instanceof	throw	while
debugger	finally	new	true	with
default	for	null	try	

Zukünftig reservierte Wörter

abstract	double	goto	native	static
boolean	enum	implements	package	super
byte	export	import	private	synchronized
char	extends	int	protected	throws
class	final	interface	public	transient
const	float	long	short	volatile

Beim Auswählen von Bezeichnern ist es außerdem wichtig, Wörter zu vermeiden, die bereits Namen von intrinsischen JScript-Objekten oder -Funktionen sind, z. B. **String** oder **parseInt**.