3 Syntax

JavaScript lernen
ISBN 3-8273-2087-9

Ziele dieses Kapitels

In diesem Kapitel lernen Sie die grundlegende Syntax von JavaScript kennen. Die vermittelten Informationen sind für die JavaScript-Programmierung unerlässlich. Nur wenn diese Grundlagen verstanden wurden, lassen sich erste Anwendungen erstellen. Besonderes Augenmerk ist auf den Abschnitt 3.3 zu richten. Hier werden die notwendigen Konventionen bezüglich der Namensvergabe gezeigt.

3.1 Kommentare

übersichtliche Programme

Durch Kommentare lässt sich die Lesbarkeit von JavaScript-Programmcode verbessern. Zusätzlich eignet sich der Einsatz von Kommentaren, um ein späteres Ändern der Syntax effizient zu gestalten und somit schnell auf die programmspezifischen Besonderheiten und deren Anforderungen reagieren zu können. Innerhalb der JavaScript-Spezifikation wurden zwei grundsätzliche Arten von Kommentar-Schreibweisen festgelegt:

- Zwei Schrägstriche kennzeichnen einen Kommentar bis zum Zeilenende.
- Durch die Zeichenabfolge /* und */ wird ein Blockkommentar, also ein Kommentar über mehrere Zeilen, spezifiziert.

Die zweite Kommentarform kann neben einem Blockkommentar auch einen einzeiligen Bereich kennzeichnen. Dies macht deutlich, dass für die Vereinheitlichung und eine verbesserte Überschaubarkeit des JavaScript-Quellcodes sehr wohl auf diese eine Art der Kommentar-Kennzeichnung zurückgegriffen werden kann und sollte. Es ergibt sich hieraus ein weitaus homogeneres Schriftbild, als dies durch die Zuweisung zweier unterschiedlicher Kommentarvarianten erzielt werden würde.

```
<SCRIPT type="text/JavaScript">
<!--
function goto(n)
{
this.length = n
return this
}
var urls = new goto(3)
/* es folgt die Definition
mehrerer URLs */
urls[0] = ""//leerer URL
urls[1] = "http://www.kss.de" //erster URL
urls[2] = "http://www.phoenix.de" //zweiter URL
function begin(url){
if (url.control.desert>0)
location.href = urls[url.control.desert]
}
//-->
</SCRIPT>
```

In dem Beispiel wurden beide Varianten der Kommentarauszeichnung angewandt. Neben den bereits erwähnten Einsatzmöglichkeiten eignen sich Kommentare zusätzlich für die Unterbringung von Urheberrechtsbestimmungen für das Script und für Änderungsanleitungen.

3.2 Anweisungen

Folge von Anweisungen

Ohne bereits an dieser Stelle ausführlich auf die zahlreichen Notationsvarianten von JavaScript einzugehen, kann dennoch die Aussage getroffen werden, dass es sich bei dieser Programmiersprache letztendlich um nichts anderes als eine Anordnung von Anweisungen handelt. Bei diesen Anweisungen handelt es sich um Befehle, die der WWW-Browser interpretiert und anschließend in auf dem System des Nutzers ausführbaren Maschinencode umwandelt. Es existieren freilich die verschiedensten Formen von Anweisungen wie zum Beispiel das Zuweisen von Werten zu einer Variablen. Aber auch die folgende Syntax stellt eine Anweisung dar:

```
alert("Corporate Identitiy");
```

Anweisungen sollten stets durch einen Strichpunkt abgeschlossen werden. Zwar ist dies nicht in jedem Fall notwendig, um eventuelle Fehlermeldungen bereits im Vorfeld einzudämmen, sollte dennoch jede Anweisung hierdurch beendet werden.

3.2.1 Anweisungsblöcke

geschweifte Klammern

Ein Anweisungsblock setzt sich aus mindestens zwei Anweisungen zusammen, die sich innerhalb einer übergeordneten Anweisung oder in Gänze innerhalb einer Funktion befinden. Das Charakteristische an Anweisungsblöcken ist das notorische Vorhandensein von geschweiften Klammern. Im Zusammenhang mit Funktionen ist zu sagen, dass deren Anfang und Ende jeweils durch eine geschweifte Klammer gekennzeichnet sind und es sich bei Funktionen somit um Anweisungsblöcke handelt. Ein Beispiel, welches einen Auszug einer Funktion zeigt, soll die für diese Notationsvariante typische Syntax demonstrieren.

```
function goto(n)
{
this.length = n
return this
}
```

Verschachtelungen

In JavaScript ist die Verschachtelung von Anweisungsblöcken erlaubt. Bei einer solchen Verschachtelung ist darauf zu achten, dass einer öffnenden geschweiften Klammer jeweils eine schließende geschweifte Klammer zugewiesen werden muss. Aus diesem Grund ist es sinnvoll, jede geschweifte Klammer innerhalb einer separaten Zeile zu notieren, um die Übersichtlichkeit, und dies gilt insbesondere im Zusammenhang mit komplexen JavaScript-Programmen, zu wahren.

3.3 Namensvergabe

Regeln für Namen

Während der Programmierung eines JavaScript-Programms müssen häufig selbst kreierte Namen vergeben werden. Dies kann beispielsweise bei eigenen Funktionen, Variabelendeklarationen oder im Zusammenhang mit eigenen Objekten der Fall sein. Diese Namen unterliegen gewissen Konventionen, ohne deren vollständige Berücksichtigung unweigerlich Fehlermeldungen des verwendeten Browsers ausgegeben werden. Die grundlegendsten und in jedem Fall zu berücksichtigenden Regeln sind hier in einer Übersicht zusammengefasst.

- Namen dürfen keine deutschen Umlaute enthalten.
- Namen dürfen keine Leerzeichen enthalten.
- Als einziges gestattetes Sonderzeichen ist der Unterstrich vorgesehen.
- Es dürfen keine reservierten Namen verwendet werden. (Mehr zu diesem Thema finden Sie im Abschnitt 3.3.1).
- Namen sollten prinzipiell aus nicht mehr als 32 Zeichen bestehen.
- Es wird zwischen Groß- und Kleinschreibung unterschieden.

- Namen sollten stets mit einem Buchstaben beginnen.
- Namen dürfen sowohl aus Buchstaben wie auch aus Zahlen oder einer Symbiose aus beiden Elementen bestehen.

Problematisch wird die Vergabe von Namen immer dann, wenn sich deren Bedeutung nicht auf den ersten Blick erschließt. Zwar ist es gang und gäbe, kryptische Namen einzusetzen, vor dieser Verfahrensweise sei an dieser Stelle dennoch gewarnt. So mag zwar ein bestimmter Name bzw. dessen Bedeutung während der ersten Programmierzeit noch geläufig sein, bei später anfallenden Korrekturen des Internetprojekts ist das dann allerdings häufig nicht mehr der Fall. Erschwerend kommt hinzu, dass an der Erstellung eines komplexen Projekts häufig mehrere Programmierer beteiligt sind. Und eben auch für die sollten die Bedeutung und auch die Schreibweise eines Namens keine zusätzliche Belastung darstellen.

3.3.1 Reservierte Namen

diese Namen nicht verwenden

Die folgende Tabelle enthält alle reservierten Namen, die innerhalb eines JavaScript-Programms nicht für selbst definierte Variablen oder Objekte verwendet werden dürfen. Wie die Auflistung zeigt, ist die Mehrzahl dieser Namen derzeit noch mit keiner Funktion belegt. Sie sollen aber für spätere Sprachweiterentwicklungen reserviert bleiben. Es ist in jedem Fall darauf zu achten, dass auch solche Namen, die noch nicht mit einer Funktion belegt sind, nicht als Variablen- oder Funktionsnamen verwendet werden dürfen.

Wort	Beschreibung
`abstract`	bislang nicht belegt
`boolean`	bislang nicht belegt
`break`	beendet eine Schleife
`byte`	bislang nicht belegt
`case`	einzelner Fall innerhalb einer Fallunterscheidung
`catch`	Fehlerbehandlung
`char`	bislang nicht belegt
`class`	bislang nicht belegt
`const`	Deklaration von Konstanten
`continue`	erzwingt den nächsten Schleifen-Durchlauf
`debugger`	bislang nicht belegt
`default`	wenn kein Fall bei einer Fallunterscheidung zutrifft
`delete`	löscht ein Objekt

Tabelle 3.1: Reservierte Namen in JavaScript-Programmen

Wort	Beschreibung
`do`	für `do-while`-Schleifen
`double`	bislang nicht belegt
`else`	Sonst-Fall bei Fallunterscheidungen
`enum`	bislang nicht belegt
`export`	Objekte für fremde Scripts ausführbar machen
`extends`	bislang nicht belegt
`false`	Rückgabewert von Funktionen
`final`	bislang nicht belegt
`finally`	bislang nicht belegt
`float`	bislang nicht belegt
`for`	für for-Schleife
`function`	Funktionsdeklaration
`goto`	bislang nicht belegt
`if`	Bedingung für bedingte Anweisungen
`implements`	bislang nicht belegt
`import`	erlaubt das Importieren von Objekten usw. aus signierten Scripts
`in`	für `for-in`-Schleifen
`instanceof`	überprüft, ob eine Variable eine Instanz eines Objekts ist
`int`	bislang nicht belegt
`interface`	bislang nicht belegt
`long`	bislang nicht belegt
`native`	bislang nicht belegt
`new`	definiert ein Objekt
`null`	setzt ein Objekt auf null
`package`	bislang nicht belegt
`private`	bislang nicht belegt
`protected`	bislang nicht belegt
`public`	bislang nicht belegt
`return`	Rückgabewert für Funktionen
`short`	bislang nicht belegt
`static`	bislang nicht belegt
`super`	bislang nicht belegt
`switch`	Fallunterscheidung
`synchronized`	bislang nicht belegt
`this`	nimmt auf das aktuelle Objekt Bezug
`throw`	nutzerdefinierte Ausnahme
`throws`	bislang nicht belegt

Tabelle 3.1: Reservierte Namen in JavaScript-Programmen (Forts.)

Wort	Beschreibung
`transient`	bislang nicht belegt
`true`	Rückgabewert von Funktionen
`try`	testet, ob eine Anweisung ausführbar ist
`typeof`	ermittelt den Typ eines Elements
`var`	Variabelendeklaration
`void`	erzeugt keinen Rückgabewert
`volatile`	bislang nicht belegt
`while`	für `while`-Schleifen
`with`	einem Objekt werden mehrere Anweisungen zugewiesen

Tabelle 3.1: Reservierte Namen in JavaScript-Programmen (Forts.)

Diese Tabelle sollte bei der Programmierung eines jeden JavaScripts berücksichtigt werden, da auch die Verwendung von derzeit noch nicht belegten Namen zu einer JavaScript-Fehlermeldung führen kann. Das soll Abbildung 3.1 demonstrieren. Zu der Fehlermeldung kam es, da eine Funktion mit dem Namen `long()` definiert wurde und `long` ein reservierter Name ist.

Abbildung 3.1: Fehlermeldung im Netscape Navigator 7

3.4 Fragen und Übungen

1. Welche der folgenden selbst vergebenen Namen sind korrekt?

bär

1bär

Fotos2

final

neues:Foto

2. Woran erkennen Sie einen Anweisungsblock?