CARL HANSER VERLAG

Ralf Siegel

Programmierhandbuch ActionScript

Leitfaden für die Programmierung von Flash MX-Anwendungen

3-446-22116-6

www.hanser.de

7 Schleifen

7.1 Jetzt wird Achterbahn gefahren

Zugegeben, ich hab' da möglicherweise etwas übertrieben, aber im Grunde genommen haben Schleifen in ActionScript denselben Erholungseffekt wie Achterbahnfahren auf dem Jahrmarkt. Anschnallen, zurücklehnen und schauen, was passiert. Schleifen stellen eine immens wichtige Kontrollstruktur dar und sind Ihre besten Freunde, wenn es darum geht, einen ganzen Sack voll Variablen und Objekte im Film zu erzeugen, zu verändern oder auch zu löschen. Mit einem verschmitzten Lächeln setzt Ihnen eine Schleife eben mal hundert Movieclips auf die Bühne und positioniert diese dann auch noch genau auf das Pixel. Schleifen kämpfen sich durch Berge an Daten in Arrays und bringen komplizierte Berechnungen fertig, die vor einhundert Jahren noch eine ganze Generation von Mathematikern beschäftigt hätte. Wir unterscheiden dabei vier verschiedene Formen von Schleifen in ActionScript:

- Die *while*-Schleife als bedingte Schleife, wobei die Bedingung am Anfang eines jeden Loops überprüft wird;
- die *do-while*-Schleife als bedingte Schleife, wobei die Bedingung hier erst am Ende eines jeden Loops überprüft wird;
- die *for*-Schleife als Zählschleife mit einer Schleifenvariablen als Zähler, welche individuell manipuliert werden kann;
- die *for-in*-Schleife als Ableger der *for*-Schleife mit einer Schleifenvariablen, welche sich durch die Eigenschaften eines Objektes arbeitet.

7.2 Die *while*-Schleife

Die *while*-Schleife ist in Ihrer Art der bedingten *if*-Anweisung recht ähnlich. Bei einer *if*-Anweisung wird der Anweisungsblock immer dann ausgeführt, wenn die aufgestellte Bedingung erfüllt wurde. Bei einer *while*-Schleife hingegen wird der Anweisungsblock **so lange** ausgeführt, wie die aufgestellte Bedingung erfüllt ist. Ein Blick auf die allgemeine Syntax macht die Seelenverwandtschaft zur *if*-Anweisung deutlich.

```
while ( bedingung ) anweisung;
```

Solange der Ausdruck *bedingung* in der runden Klammer den Booleschen Wert *true* ergibt, so lange wird die nachfolgende Anweisung *anweisung* ausgeführt.

Listing 7.1 Einfache *while*-Schleife

```
var i=10;
while(i>0) trace(i--);
```

```
10
9
8
7
6
5
4
3
2
1
```

Auch hier gilt, wie schon bei der *if*-Anweisung besprochen, dass erstens mehrere Anweisungen in geschweiften Klammern zusammengefasst werden müssen und zweitens auch einzeilige Anweisungen nach Möglichkeit in geschweiften Klammern stehen sollten, um das Skript lesbarer zu gestalten.

Listing 7.2 Einfache *while*-Schleife in der empfohlenen Formatierung

```
var i = 10;
while (i > 0)
{
   trace(i--);
}
```

Was passiert hier genau? Der Interpreter trifft auf die *while*-Anweisung und überprüft die aufgestellte Bedingung *i > 10* in der runden Klammer. Da die Variable *i* beim Eintritt in die Schleife den Wert 10 besitzt und damit zweifellos größer als 0 ist, ist die Bedingung erfüllt und der Interpreter tritt in den Anweisungsblock oder auch Schleifenkörper ein. Hier geben die Anweisungen den Wert der Variablen im Ausgabefenster aus und verringern danach diesen um eins. Falls Ihnen das etwas spanisch vorkommt, das Skript hätte man auch so schreiben können:

Listing 7.3 Dieselbe Schleife, etwas anders geschrieben

```
var i = 10;
while (i > 0)
{
```

```
   trace(i);
   i--; // der Wert der Variablen i wird um eins verringert
}
```

Nicht viel Neues werden Sie sagen, aber nun passiert der eigentliche Zauber. Der Interpreter verlässt nach Abarbeiten der Anweisungen nicht etwa die Schleife, sondern springt an den Beginn der Schleife zurück und überprüft die Bedingung erneut. Diesmal hat die Variable `i` nur noch den Wert 9 und so nimmt die ganze Sache ihren Lauf, bis *i* den Wert 0 angenommen hat und damit die Bedingung nicht mehr erfüllt ist. Merken Sie sich auch, dass der Interpreter gar nicht erst in eine *while*-Schleife verzweigt, wenn die Bedingung am Anfang nicht erfüllt ist. Dies ist ein Unterschied zur *do-while*-Schleife, die Sie bald kennen lernen werden.

Listing 7.4 Die Schleifenbedingung ist von Anfang an nicht erfüllt

```
var i = -1;
while (i > 0)
{
   trace(i--);
}
```

Schleifen sind, was die Loyalität zur Bedingung angeht, unglaubliche Dickköpfe, deshalb möchte ich Sie an dieser Stelle schon einmal vor dem Sensenmann in Gestalt einer Endlosschleife warnen. Denn glauben Sie nicht, dass Schleifen die Puste ausgeht, wenn sich die aufgestellte Bedingung als permanent wahr herausstellt. Einmal in Gang gekommen, laufen Schleifen hier ins Verderben. Um dies zu demonstrieren, wird im folgenden Skript die Variable *i* nicht heruntergezählt, wodurch diese den Wert *10* behält. Dies bedeutet, die Bedingung $i > 10$ ist immer erfüllt. Bahn frei für die Endlosschleife.

Listing 7.5 Provozieren einer Endlosschleife

```
// Achtung, ausführen auf eigene Gefahr !!!
// Dieses Skript legt Ihren Rechner für mehrere Sekunden lahm
var i = 10;
while (i > 0)
{
   trace(i); // i wird hier nicht heruntergezählt und ist immer 10
}
```

Nach etwa 20–30 Sekunden erscheint eine Meldung des Interpreters, und fragt Sie, ob Sie das Skript abbrechen möchten. Wählen Sie nach Möglichkeit „Ja“. Wenn Sie sich für „Nein“ entscheiden, wird die Schleife weiterhin ausgeführt, was tatsächlich bis zum Absturz des Rechners führen kann.

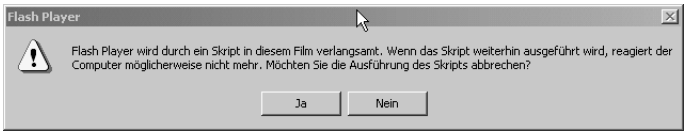

Abbildung 7.1 Der Flash Player bricht eine Endlosschleife ab

Dieser Schutzmechanismus des Flash Player ist primär nicht einmal für Sie als Entwickler gedacht, obwohl ich persönlich diesem Schutz sehr verbunden bin, sondern in erster Linie für den wehrlosen Besucher Ihrer Anwendung. So, lieber Leser, nachdem Sie nun die Schattenseite der Schleife kennen gelernt haben, können wir uns wieder der Sonnenseite zuwenden: *„Komm, wir malen eine Sonne“*. Das folgende Beispiel greift auf die Zeichenmethoden des *movieclip*-Objekts vor, was hoffentlich auch in Ihrem Sinne ist. Wenn Sie neugierig sind, können Sie natürlich gerne in der Referenz stöbern, was die Zeichenmethoden *lineStyle()*, *moveTo()*, *lineTo()* im Detail machen, oder Sie warten bis zum entsprechenden Kapitel.

Listing 7.6 Sonne zeichnen mit einer *while*-Schleife

```
var strahlen = 36, radius = 100;
var x0 = y0 = 100, b = 0, x, y;
var schrittweite = 2 * Math.PI / strahlen;
this.lineStyle(5, 0xFFFF00, 100);  // Linienstil definieren
while (strahlen > 0)
{
   this.moveTo(x0, y0);            // Stift zum Mittelpunkt bewegen
   b += schrittweite;              // Bogenmass ermitteln
   x = radius * Math.sin(b);       // x-Koordinate berechnen
   y = radius * Math.cos(b);       // y-Koordinate berechnen
   this.lineTo(x0 + x, y0 + y);    // Linie zum Punkt zeichnen
   strahlen--;
}
```

Stellen Sie sich vor, Sie müssten dieses Skript ohne Schleife realisieren. Aber vielleicht haben Sie auch Lust, das Ganze in eine Funktion zu packen? Denken Sie auch immer daran, dass es sehr wohl einen Unterschied ausmacht, **wann** Sie die Variable oder Bedingung verändern, also ob vor oder nach den Anweisungen. So werden im folgenden Skript nicht wie bisher die Zahlen von 1 bis 10 ausgegeben, sondern von 0 bis 9.

Listing 7.7 Variable i wird erst dekrementiert und danach der Wert ausgegeben

```
var i = 10;
while (i > 0)
{
   trace(--i); // vorher stand hier trace(i--)
}
```

```
9
...
1
0
```

Oder wieder in der anderen Schreibweise, um das Ganze zu veranschaulichen.

Listing 7.8 Dieselbe Schleife, anders geschrieben

```
var i = 10;
while (i > 0)
{
   i--;
   trace(i);
}
```

break

Übrigens gibt es auch für die *while*-Schleife einen geschickten Weg, um die Bedingung im Schleifenkörper selber zu formulieren und später mittels des Schlüsselwortes *break* aus der Schleife auszubrechen. Sicherlich erinnern Sie sich noch an die *switch*-Auswahlanweisung, in der es auch das Schlüsselwort *break* gab und wo ich Ihnen etwas Ähnliches gezeigt habe.

Listing 7.9 Schleife mit break beenden

```
var i = 10;
while (true)
{
   trace(i--);
   if (i <= 0) break;
}
```

Hierbei ist die Schleifenbedingung der *while*-Schleife für sich genommen immer erfüllt. Der Interpreter macht ja nicht mehr und nicht weniger, als den Ausdruck in den runden Klammern auf seinen Wahrheitsgehalt zu überprüfen, und dieser Ausdruck liefert, egal was passiert, immer den Wert *true*. Praktisch gesehen also eine Endlosschleife, die letztlich nur durch die zusätzlich aufgestellte *if*-Anweisung im Schleifenkörper samt *break* im Zaum gehalten wird. Solch eine Herangehensweise kann durchaus sinnvoll sein, wenn Sie komplexere Bedingungen aufstellen wollen, ohne den Komfort einer Schleife einbüßen zu wollen, und funktioniert auch mit den anderen Schleifentypen.

continue

Möchten Sie die Anweisungen für einzelne Werte in einer Schleife überspringen, dann hilft Ihnen das Schlüsselwort *continue* weiter. Aber passen Sie hier besonders auf, dass Sie dabei nicht die Anweisungen zum Manipulieren der Schleifenvariable überspringen, sonst finden Sie sich schnell in den Fängen einer Endlosschleife wieder. Wie schon *break*, funktioniert auch *continue* nicht nur mit einer *while*-Schleife, sondern kann auch in allen anderen Schleifentypen verwendet werden. Im folgenden Beispiel werden alle Zahlen von 0 bis 9 ausgegeben, mit einer Ausnahme – die Zahl 7 wird nicht angezeigt.

Listing 7.10 Wenn i den Wert 7 angenommen hat, wird die *trace*-Anweisung übersprungen

```
var i = 10;
while (i > 0)
{
   i--;
   if (i==7) continue; // wenn i=7 wird die trace-Anweisung übersprungen
   trace(i);
}
```

```
9
8
6
5
4
3
2
1
0
```

7.3 Die *do-while*-Schleife

Die allgemeine Syntax einer *do-while*-Schleife sieht wie eine verdrehte *while*-Schleife aus. Für einzeilige Anweisungen gilt folgende Schreibweise:

```
do anweisung while ( bedingung )
```

Mehrzeilige Anweisungen müssen auch hier in geschweiften Klammern gesetzt werden.

```
do { anweisung } while ( bedingung )
```

Der Unterschied zur *while*-Schleife besteht darin, dass die Bedingung erst nach dem Schleifendurchlauf überprüft wird. Das folgende Beispiel offenbart aber tatsächlich vorerst keinen Unterschied.

Listing 7.11 Einfache *do-while*-Schleife

```
var i = 10;
do
{
   trace(i--);
} while (i>0);
```

Dies wirkt sich erst aus, wenn die Schleifenbedingung von Beginn an nicht erfüllt ist. Denn die Anweisungen werden im Gegensatz zur *while*-Schleife wenigstens einmal ausgeführt.

Listing 7.12 Die Schleifenbedingung ist von Anfang an nicht erfüllt

```
var i = -1;
do
{
   trace(i--); // Ausgabe -1
} while (i > 0);
```

7.4 Die *for*-Schleife

Es kann wohl behauptet werden, dass die *for*-Schleife der gegenwärtig am häufigsten benutzte Schleifentyp in der ActionScript-Programmierung ist. Plausibel erklären lässt sich das eigentlich nicht, doch die Fakten sprechen für sich. Wenn man sich in einschlägigen Foren umschaut, haben *for*-Schleifen eindeutig die Oberhand. Vielleicht ist aber das Prinzip, Zählervariable und Schleifenbedingung an einem Platz zu haben, für die meisten Fälle überschaubarer und somit gefälliger. Und so sieht der Schleifenliebling für einzeilige Anweisungen aus:

```
for (init_ausdruck; bedingung; mod_ausdruck) anweisung
```

Analog dazu die Schreibweise mit geschweiften Klammern, zwingend für einen Schleifenkörper mit mehreren Anweisungen, empfohlen auch für einzeilige Anweisungen:

```
for (init_ausdruck; bedingung; mod_ausdruck) { anweisung }
```

Lassen Sie uns vor dem ersten Beispiel diese Begriffe noch etwas genauer betrachten, denn hier kursieren mitunter recht ungenaue Vorstellungen hinsichtlich der Aufgaben der einzelnen Ausdrücke:

Tabelle 7.1 Begriffsbestimmung der Bestandteile einer *for*-Schleife

Ausdruck	Nicht ganz korrekt	Genauer gesagt
init_ausdruck	Startwert. Hier wird der Startwert der Schleifenvariablen definiert.	Initialisierungs-Ausdruck. Es handelt sich hierbei um einen Ausdruck, und nicht um einen Wert. Dieser Ausdruck initialisiert in den meisten Fällen eine Variable. Es lassen sich aber auch mehrere Variablen initialisieren oder auch eine Funktion aufrufen. Wird nur einmal zu Beginn ausgeführt. *Optionaler Ausdruck.*
bedingung	Prüft immer, ob die Zählervariable ihren Endwert erreicht hat.	Es handelt sich allgemein um eine Bedingung, die auch prüfen könnte, ob es Tag oder Nacht ist. Ist diese Bedingung erfüllt, wird die *for*-Schleife weiter fortgesetzt. Diese Bedingung prüft zwar in den meisten Fällen den Endwert der Zählervariable, dies ist jedoch nur eine von vielen Möglichkeiten. *Optionaler Ausdruck.*
mod_ausdruck	Hier wird immer die Zählervariable um eins erhöht.	Das ist meistens der Fall, aber hier ist allgemein Platz für Anweisungen, die nach jedem Schleifendurchlauf ausgeführt werden. Es könnte hier zum Beispiel auch ein Funktionsaufruf stehen. *Optionaler Ausdruck.*

Das folgende Beispiel zeigt so eine überaus kompakte und praktische *for*-Schleife, welche einmalig eine Zählervariable *i* mit dem Startwert 1 initialisiert. In der aufgestellten Bedingung wird **vor** jedem Schleifendurchlauf getestet, ob der Wert der Zählervariablen *i* größer oder gleich 10 ist. Und **nach** jedem Schleifendurchlauf wird der Wert der Variablen *i* um eins erhöht.

Listing 7.13 Einfaches Beispiel für eine *for*-Schleife

```
for(var i=1; i<=10; i++) {
   trace(i);
}
```

Es ist gar nicht so unbedeutend, sich vor allem auch zu merken, **wann** welcher Teil dieser Schleife ausgeführt wird, sonst stellt man wie im obigen Skript eventuell überrascht fest, dass die Variable *i* nach Beenden der Schleife zum Beispiel den Wert 11 besitzt und nicht etwa den Wert 10.

Listing 7.14 Es ist wichtig, sich zu merken, welche Ausdrücke der Schleife wann an der Reihe sind

```
for(var i=1; i<=10; i++) {
   trace(i);
```

```
}
trace("i hat jetzt den Wert " + i); // ergibt 11 und nicht 10 !
```

Wie ich in der Tabelle oben schon vermerkt hatte, sind alle drei Ausdrücke als Bestandteil einer *for*-Schleife immer optional. Sie könnten also zum Beispiel die monotone Aufgabe des Inkrementierens der Zählervariable in den Schleifenkörper verlagern:

```
for(var i=1; i<=10; ) {
  trace(i);
  i++;
}
```

Oder den Ausdruck der Bedingung:

```
for(var i=1; ; ) {
  if (i>10) break;
  trace(i);
  i++;
}
```

Oder die Schleife ohne alles dastehen lassen:

```
var i=1;
for(; ; ) {
  if (i>10) break;
  trace(i);
  i++;
}
```

Wobei Sie an dieser Stelle auch sehr deutlich erkennen können, wie die Aufgabenverteilung hinter der Fassade dieser Schleife in Wirklichkeit funktioniert. Auch eine Endlosschleife ist natürlich wieder mit sehr einfachen Mitteln drin.

Listing 7.15 Provozieren einer Endlosschleife

```
// Achtung, ausführen auf eigene Gefahr !!!
// Dieses Skript legt Ihren Rechner für mehrere Sekunden lahm
for( ; ; ) {}
```

Interessant ist auch, dass Sie nicht nur mit *break* aus der Schleife ausbrechen können, sondern auch mit Schlüsselwort *return*, welches Ihnen noch von den Funktionen her bekannt sein dürfte, aber das eigentlich hier nur am Rande.

Listing 7.16 Schleifenabbruch mit return

```
for(var i=1; ; i++) {
   trace(i);
   if (i==5) return;
}
```

Ein nahezu krisenfester Job für *for*-Schleifen in ActionScript ist es, Objekte wie zum Beispiel Bilder oder Quadrate in einer Art zweidimensionalem Feld auf der Bühne zu platzieren. Ähnlich also dem Aufbau und Aussehen eines Schachbretts.

Listing 7.17 Schachbrett zeichnen mit zwei *for*-Schleifen

```
// Funktion zum Zeichnen von Quadraten
// z: Position auf der z-Achse | a: Kantenlänge
function drawQuader(farbe, z, a)
{
   // neuen leeren Movieclip erstellen
   var q = this.createEmptyMovieClip("quader_" + z, z);
   // Zeichenmethoden
   q.lineStyle(2, 0xFF0000, 100);
   q.beginFill(farbe, 100);
   q.moveTo(0, 0);
q.lineTo(a, 0);
   q.lineTo(a, a);
   q.lineTo(0, a);
   q.lineTo(0, 0);
   // gibt Referenz auf den neuen Movieclip zurück
   return q;
}
// Zähler für die Anzahl der Quadrate
var index = 0;
var kantenlaenge = 25;
// Zwei verschachtelte for-Schleifen
for (var x = 1; x <= 10; x++)
{
   for (var y = 1; y <= 10; y++)
   {
      var farbe = ((x+y) % 2) * 0xFFFF00
      neuesQuadrat = drawQuader(farbe, index, kantenlaenge);
      neuesQuadrat._x = x * kantenlaenge;
      neuesQuadrat._y = y * kantenlaenge;
```

```
      index++;
   }
}
```

Probieren Sie doch auch einfach mal, die Schrittweite der beiden Schleifen von eins auf zwei zu verändern, was dann in etwa so aussehen würde:

```
for (var x = 1; x <= 10; x+=2)
{
  for (var y = 1; y <= 10; y+=2)
  {
    ...
  }
}
```

Wenn Sie dieses Beispiel mit dem Schachbrett ohne Durchfall verdaut haben, kann ich Ihnen wirklich nur gratulieren, denn in der Tat findet sich hier, bis auf die Zeichenmethoden, ein Großteil dessen wieder, was Sie bisher schon gelernt haben. Die neu in ActionScript MX eingeführten Zeichenmethoden sind insofern auch in diesem Stadium sehr praktisch, da Sie hier einfach nur das Skript einzutippen brauchen, es modifizieren können und sofort ein Ergebnis sehen. Und ich an Ihrer Stelle hätte sowieso schon längst vorgeblättert.

7.5 Die *for-in*-Schleife

Als letzter Typ im schlagkräftigen Schleifenensemble fehlt noch die *for-in*-Schleife, welche eine besondere Stellung in ActionScript einnimmt, da sie nur im Zusammenhang mit Objekten, und hier besonders gerne auch mit Arrays verwendet wird. Ein Thema aber, welches noch aussteht. Allgemein sieht die Schreibweise der *for-in*-Schleife wie folgt aus:

```
for (eigenschaft in objekt) { anweisungen }
```

Das Besondere an dieser Kontrollstruktur besteht nun darin, dass sie geradezu detektivisch genau sämtliche Eigenschaften eines Objektes aufspürt, ohne dass wir auch nur die leiseste Ahnung über Art und Anzahl dieser Objekt-Eigenschaften mitbringen müssten. Das ist so, als ob Sie einen Mechaniker im Blaumann neben sich stehen haben, der Ihnen mal eben sämtliche Teile und technische Daten Ihres klapprigen Autos auflistet. In der Tat müssen wir nicht einmal wissen, ob das Objekt überhaupt Eigenschaften besitzt oder ob das Auto schrottreif ist. In dem Fall verzweigt die *for-in*-Schleife nämlich gar nicht erst in den Schleifenkörper. Das ist sozusagen die versteckte Bedingung dieses Typs: *Verzweige nur dann in den Schleifenkörper, wenn das Objekt Eigenschaften besitzt, und durchlaufe die Schleife so oft, wie du noch Eigenschaften*

findest. Dem mitlaufenden Index wird dabei wie von Geisterhand bei jedem Schleifendurchlauf der Name der gerade gefundenen Eigenschaft zugeordnet.

Listing 7.18 Durchlaufen der Eigenschaften eines selbst erstellten Objektes *auto*

```
var auto = new Object();
auto.farbe = "rot";
auto.motor = "10 Zylinder";
auto.ps = 220;
for (teil in auto) {
  trace(teil + " : " + auto[teil]);
}
```

```
ps : 220
motor : 10 Zylinder
farbe : rot
```

So kann man sich mit einfachsten Mitteln auch alle Eigenschaften und die dazugehörigen Werte eines schon vordefinierten Objektes, wie etwa *System.capabilities*, anzeigen lassen.

Listing 7.19 Anzeigen aller Systemeigenschaften mit einer *for-in*-Schleife

```
for (prop in System.capabilities) {
  trace(prop + " : " + System.capabilities[prop]);
}
```

Der Name der Objekt-Eigenschaft wird im obigen Beispiel bei jedem Schleifendurchlauf in der Variablen *prop* abgelegt. Dies geschieht tatsächlich ohne Ihr Zutun. Nur der Interpreter kennt die Namen aller Eigenschaften eines Objektes. Den Wert der Eigenschaft können Sie leicht mit dem *[]*-Operator oder auch Zugriffsoperator erhalten, hier mit *System.capabilities[prop]*. Die beiden ersten Zeilen ergeben auf meinem Rechner zum Beispiel Folgendes, werden aber auf Ihrem Rechner möglicherweise abweichen.

```
language : de
os : Windows 2000
...
```

Hier ist *language* der Name der ersten gefundenen Eigenschaft und „de“ der dazugehörige Wert dieser Eigenschaft. Aha, ein deutschsprachiges Betriebssystem, gut zu wissen. Beim nächsten Durchlauf findet der Interpreter eine Eigenschaft mit dem Namen *os*, deren Wert wieder eine Zeichenkette zu sein scheint – „Windows 2000“. Das geht so weiter bis zum Auflisten der Bildschirmeigenschaften und diversen anderen Details. Wir werden auf Objekte und Eigenschaften aber wie gesagt noch genauer eingehen, deshalb zum Abschluss nur eine kleine Vorschau und

ein Beispiel mit einem speziellen Objekt, einem Array. Speziell deswegen, weil die Eigenschaften eines Array-Objektes laut den heiligen ECMAScript-Schriften immer **ganze Zahlen** wie 0, 1, 2 oder 3 sind und nicht etwa Zeichenketten wie „*language*“ oder „os“. Fragen Sie mich jetzt aber nicht nach „assoziativen“ und „verschachtelten“ Arrays – Sie werden es bald selbst herausbekommen.

Listing 7.20 Durchlaufen eines Arrays mit einer *for-in*-Schleife

```
var obst_und_gemuese = ["Äpfel", "Birnen", "Tomaten", "Gurken"];
for (vitamin_index in obst_und_gemuese) {
   trace(vitamin_index + ". " + obst_und_gemuese[vitamin_index]);
}
```

```
3. Gurken
2. Tomaten
1. Birnen
0. Äpfel
```

Jetzt, da Sie die Grundlagen von ActionScript kennen, und sogar schon am Autolack gekratzt haben, wird es erst richtig spannend. Das ist wie wenn man beim Gitarrespielen nicht mehr aufs Griffbrett schauen muss, sondern sich völlig der berauschenden Tiefe des Spielens hingeben kann.

Plug it in.