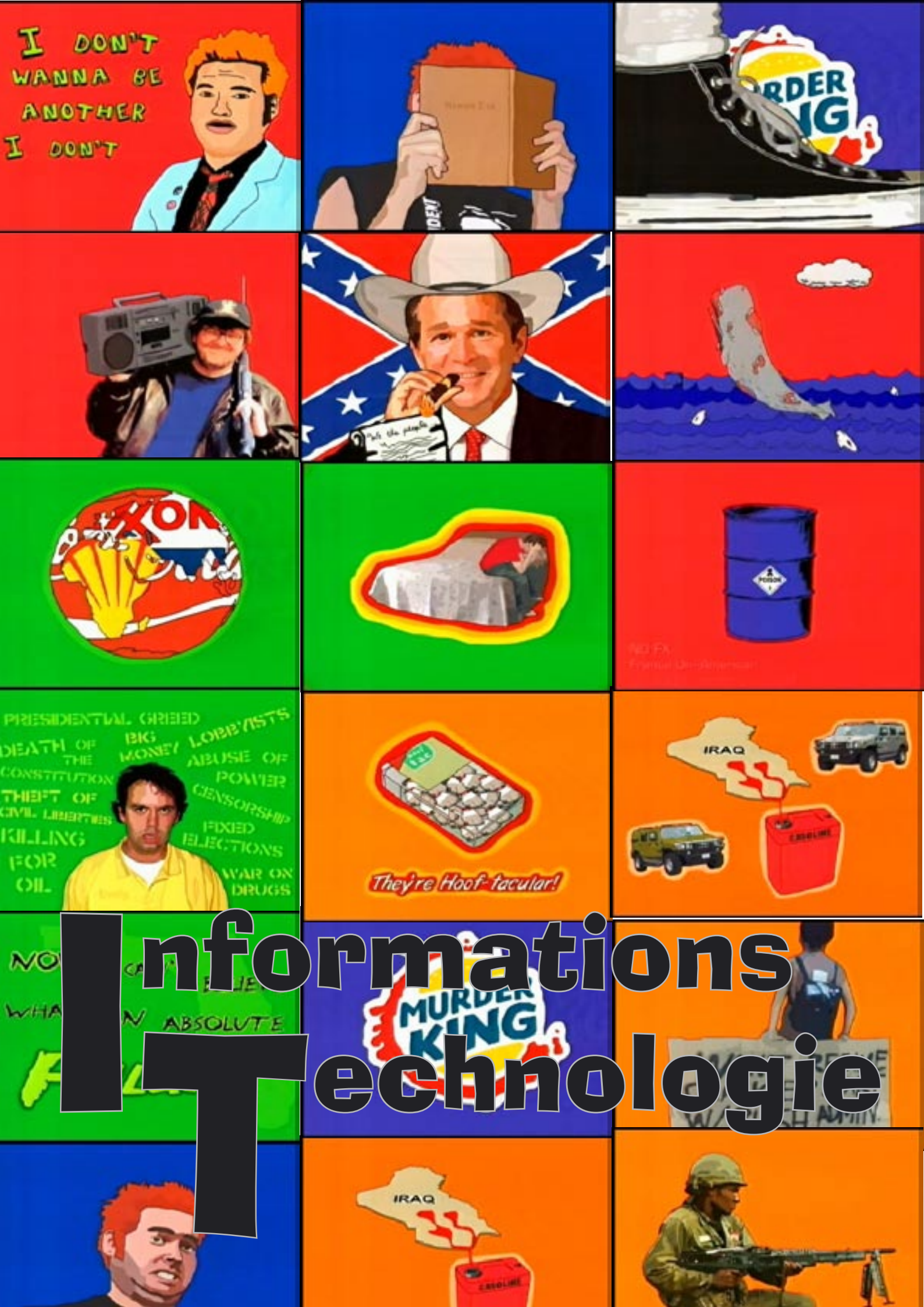

Informations Technologie

Inhalt

Informations Technick

1.0. Javascript

1.1. Javascript Grundlagen

Für JavaScript-Programmabschnitte können Sie in HTML Bereiche definieren.
Beispiel:

Anzeigebeispiel:

```
<html>
<head><title>Test</title>
<script type="text/javascript">
<!--
 alert(„Hallo Welt!“);
//-->
</script>
</head>
<body>
</body>
</html>
```

Erläuterung

Mit <script type="text/javascript"> leiten Sie einen Bereich für JavaScript innerhalb einer HTML-Datei ein (script = Quelltext, type = Mimetype). Die Angabe zum Mimetype ist seit HTML4.0 Pflicht. Mit type="text/javascript" bestimmen Sie den typischen Mime-Type für JavaScript-Dateien.
Zusätzlich können Sie zur Kennzeichnung der JavaScript-Sprachversion das language-Attribut verwenden (z.B. language="JavaScript" oder language="JavaScript1.2"). Jedoch gehört dieses Attribut zu den missbilligten Attributen und wird auch nicht von allen Browsern korrekt interpretiert.
Dahinter - am besten in der nächsten Zeile - sollten Sie mit <!-- einen HTML-Kommentar einleiten. Dadurch erreichen Sie, dass ältere WWW-Browser, die JavaScript nicht kennen, den folgenden JavaScript-Code ignorieren und nicht irrtümlich als Text innerhalb der HTML-Datei interpretieren.

Im obigen Beispiel wird mit Hilfe von JavaScript ein Meldungsfenster mit dem Text „Hallo Welt!“ am Bildschirm ausgegeben.

Den JavaScript-Bereich schließen Sie mit einem einzeiligen JavaScript-Kommentar //, gefolgt vom schließenden HTML-Kommentar --> und </script> ab. Der JavaScript-Kommentar ist erforderlich, um Fehlermeldungen in scriptfähigen Browsern zu unterdrücken. Berücksichtigen Sie, dass vor und nach dem HTML-Kommentar ein Zeilenumbruch zwingend erforderlich ist.

1.2. JavaScript-Anweisungen in HTML-Tags

JavaScript kann auch innerhalb herkömmlicher HTML-Tags vorkommen. Das ist dann kein komplexer Programmcode, sondern in der Regel nur der Aufruf bestimmter Methoden, Funktionen, Objekte, Eigenschaften. Für den Aufruf gibt es so genannte Event-Handler. Das sind Attribute in HTML-Tags, über die sich JavaScripts aktivieren lassen. Für jeden der möglichen Event-Handler ist festgelegt, in welchen HTML-Tags er vorkommen darf.
Beispiel:

Anzeigebeispiel: So sieht‘s aus

```
<html>
<head>
<title>Test</title>
<script type=“text/javascript“>
<!--
function Quadrat() {
   var Ergebnis = document.Formular.Eingabe.value * document.For-
mular.Eingabe.value;
   alert(„Das Quadrat von „ + document.Formular.Eingabe.value + „
= „ + Ergebnis);
}
//-->
</script>
</head>
<body>
<form name=“Formular“ action=““>
<input type=“text“ name=“Eingabe“ size=“3“>
<input type=“button“ value=“Quadrat errechnen“
onClick=“Quadrat()“>
</form>
</body>
</html>
```

Erläuterung

Das obige Beispiel zeigt eine komplette HTML-Datei. Im Dateikopf ist ein JavaScript-Bereich definiert. Innerhalb dieses Bereichs steht eine selbst definierte Funktion, die Funktion mit dem Namen Quadrat(). Die Funktion errechnet das Quadrat einer Zahl, die der Anwender in dem weiter unten notierten Formular im Eingabefeld mit dem Namen Eingabe eingibt. Der JavaScript-Code dieser Funktion wird aber nicht automatisch ausgeführt, sondern nur dann, wenn die Funktion explizit aufgerufen wird. Im obigen Beispiel erfolgt der Aufruf mit Hilfe eines Klick-Buttons. Wenn der Anwender auf den Button klickt, wird das Quadrat zur eingegebenen Zahl ausgegeben. Dazu ist im HTML-Tag des Klick-Buttons das Attribut onClick= notiert - ein Event-Handler mit der Bedeutung „beim Anklicken“.

Beachten Sie:

Im Abschnitt Event-Handler wird beschrieben, welche Event-Handler es außer onClick= noch gibt, und in welchen HTML-Tags sie vorkommen dürfen.

1.3. Anweisungen notieren

JavaScript besteht letztendlich aus einer kontrollierten Anordnung von Anweisungen. Das sind Befehle, die der JavaScript-Interpreter des WWW-Browsers bewertet und in Maschinencode umsetzt, der auf dem Rechner des Anwenders ausführbar ist.

Es gibt einfache und komplexere Anweisungen.
Beispiel 1:

```
Zahl = 42;
```

Beispiel 2:

```
Quadrat = Zahl * Zahl;
```

Beispiel 3:

```
if(Zahl > 1000)
  Zahl = 0;
```

Beispiel 4:

```
alert(„Das Quadrat von „ + Zahl + „ = „ + Ergebnis);
```

Erläuterung:

Eine Anweisung in JavaScript besteht immer aus einem Befehl, der mit einem Strichpunkt ; oder einem Zeilenumbruch abgeschlossen wird. In neueren Netscape-Dokumentationen zu JavaScript wird der Strichpunkt am Ende von einfachen Anweisungen zwar häufig weggelassen, aber um unnötige Fehler zu vermeiden, ist es ratsam, sich von vorneherein anzugewöhnen, alle Anweisungen auf diese Weise abzuschließen.

Eine Anweisung ist zum Beispiel:
wenn Sie einer Variablen einen Wert zuweisen, wie in Beispiel 1.
wenn Sie mit Variablen oder Werten eine Operation durchführen, wie in Beispiel 2.
wenn Sie einen Befehl nur unter bestimmten Bedingungen ausführen, wie in Beispiel 3 (siehe hierzu die Abschnitte über bedingte Anweisungen und über Schleifen).
wenn Sie eine selbst definierte Funktion oder eine Objektmethode aufrufen, bzw. wenn Sie eine Objekteigenschaft auslesen oder ändern (siehe hierzu den Abschnitt Objekte, Eigenschaften und Methoden), wie in Beispiel 4.

1.4. Anweisungsblöcke notieren

Ein Anweisungsblock besteht aus einer oder mehreren Anweisungen, die innerhalb einer übergeordneten Anweisung oder innerhalb einer Funktion stehen. So können Anweisungsblöcke beispielsweise unterhalb einer bedingten Anweisung oder einer Schleife stehen. Auch alle Anweisungen, die innerhalb einer selbst definierten Funktion stehen, bilden einen Anweisungsblock.
Beispiel 1:

```
if(Zahl > 1000) {
  Zahl = 0;
  Neustart();
}
```

Beispiel 2:

```
while(i <= 99) {
  Quadrat(i);
  i = i + 1;
}
```

Beispiel 3:

```
function SageQuadrat(x) {
  var Ergebnis = x * x;
  alert(Ergebnis);
}
```

Beispiel 4:

```
function SagEinmaleins(x) {
  var Ergebnis = x * x;
  if(Ergebnis > 100) {
    Ergebnis = 0;
    Neustart();
  }
  alert(Ergebnis);
}
```

Erläuterung:

Ein Anweisungsblock wird durch eine öffnende geschweifte Klammer { begonnen und durch eine schließende geschweifte Klammer } beendet. Sie können die geschweiften Klammern jeweils in eine eigene Zeile schreiben, so wie in den obigen Beispielen. Es ist aber auch erlaubt, die Klammern in der gleichen Zeile zu notieren wie die Anweisungen.

Bei bedingten Anweisungen (wie in Beispiel 1) oder bei Schleifen (wie in Beispiel 2) müssen Sie solche Anweisungsblöcke notieren, sobald mehr als eine Anweisung von der Bedingung oder der Schleifenbedingung abhängig ausgeführt werden soll. Bei Funktionen (wie in Beispiel 3) müssen Sie Anfang und Ende der Funktion durch geschweifte Klammern markieren. Alles, was innerhalb der Funktion steht, ist daher ein Anweisungsblock.

Anweisungsblöcke können auch verschachtelt sein, so wie in Beispiel 4 oben.

1.5. Selbstvergebene Namen

An vielen Stellen in JavaScript müssen Sie selbst Namen vergeben, zum Beispiel für selbst definierten Funktionen, eigene Objekte oder Variablen.
Beispiel:

Anzeigebeispiel: So sieht‘s aus

```
<html>
<head>
<title>Test</title>
<script type=“text/javascript“>
<!--
function Schlaue_Frage() {
 var i = 1;
 var Eingabe = „“;
 while(Eingabe != „SELFHTML“ && i <= 3) {
   Eingabe = window.prompt(„Wie heißt dieses Dokument?“,““);
   i++;
 }
}
//-->
</script>
</head>
<body>
<a href=“javascript:Schlaue_Frage()“>wollen Sie eine Frage beantworten?</a>
</body>
</html>
```

Erläuterung:

Im Beispiel ist in einem JavaScript-Bereich eine Funktion mit Namen Schlaue_Frage notiert. Darin werden zwei Variablen mit den Namen i und Eingabe deklariert und verwendet. Das Beispiel ruft beim Anklicken des HTML-Verweises die Funktion Schlaue_Frage auf. Darin wird ein Dialogfenster ausgegeben, in dem der Anwender bis zu drei mal auf die Frage antworten kann, wie dieses Dokument heißt. Namen vergeben müssen Sie also recht häufig in JavaScript.

Bei selbst vergebenen Namen gelten folgende Regeln:
sie dürfen keine Leerzeichen enthalten
sie dürfen nur aus Buchstaben und Ziffern bestehen - das erste Zeichen muss ein Buchstabe sein; es sind Groß- und Kleinbuchstaben erlaubt. Groß- und Kleinschreibung werden unterschieden!
sie dürfen keine deutschen Umlaute oder scharfes S enthalten
sie dürfen als einziges Sonderzeichen den Unterstrich „_“ enthalten

sie dürfen nicht mit einem reservierten Wort identisch sein.

Vergeben Sie sprechende Namen, die Ihnen auch ein halbes Jahr, nachdem Sie das Script geschrieben haben, noch signalisieren, welche Bedeutung sie haben. Es dürfen ruhig auch deutschsprachige Wörter sein, solange die genannten Regeln eingehalten werden.

1.6. Kommentare in JavaScript

Bei komplexeren Programmteilen können Sie Kommentare benutzen, um einzelne Anweisungen zu erklären. Auch wenn Sie Ihre Copyrightangaben innerhalb eines selbst geschriebenen JavaScript-Codes unterbringen wollen, können Sie dies mit Hilfe eines Kommentars tun. Kommentare werden vom JavaScript-Interpreter des WWW-Browsers ignoriert.
Beispiel:

```
while(i <= 99) {
  Quadrat = i * i;   /* solange i kleiner gleich 99, Quadrat von
i bilden */
  i = i + 1;        // i um eins erhoehen, damit es irgendwann 99
ist
}
```

Erläuterung:

Einen wahlweise ein- oder mehrzeiligen Kommentar leiten Sie mit /*, also einem Schrägstrich, gefolgt von einem Sternzeichen, ein. Mit der umgekehrten Folge */, also einem Sternzeichen, gefolgt von einem Schrägstrich, beenden Sie den Kommentar.

Einen einzeiligen Kommentar starten Sie mit der Zeichenfolge //. Der Browser ignoriert dann den nachfolgenden Text bis zum nächsten Zeilenende.

1.7. Variablen definieren

Variablen sind Speicherbereiche, in denen Sie Daten, die Sie im Laufe Ihrer Programmprozeduren benötigen, speichern können. Der Inhalt, der in einer Variablen gespeichert ist, wird als „Wert“ bezeichnet. Sie können den Wert einer Variablen jederzeit ändern. Um mit Variablen arbeiten zu können, müssen Sie die benötigten Variablen zuerst definieren.
Beispiel:

Anzeigebeispiel:

```
<html>
<head>
<title>Test</title>
<script type=“text/javascript“>
<!--
  var Hinweis = „Gleich werden Quadratzahlen ausgegeben“;
  alert(Hinweis);

  function SchreibeQuadrate() {
      var SinnDesLebens = 42;
      var i, x;
      var Satzteil = „Das Quadrat von „;
      for(i=1; i <= SinnDesLebens; ++i) {
          x = i * i;
          document.write(Satzteil + i + „ ist „ + x + „<br>“);
      }
  }
//-->
</script>
</head>
<body onLoad=“SchreibeQuadrate()“>
</body>
</html>
```

Erläuterung:

Das Beispiel gibt beim Aufruf der Seite eine Meldung aus, dass gleich Quadratzahlen ausgegeben werden. Bestätigt der Anwender das Meldungsfenster, werden die Quadrate der Zahlen von 1 bis 42 ausgegeben.

Es gibt globale Variablen und lokale Variablen. Eine lokale Variable erhalten Sie durch die Deklaration der Variablen mit var innerhalb einer Funktion. Im obigen Beispiel sind die Variablen SinnDesLebens, i, x und Satzteil innerhalb der Funktion SchreibeQuadrate() als lokale Variablen notiert. Diese Variablen sind deshalb nur innerhalb dieser Funktion gültig. Man spricht in diesem Zusammenhang auch von der „Lebensdauer“ von Variablen. Parameter, die einer Funktion übergegeben werden, werden ebenfalls als lokale Variablen behandelt.
Die Variable Hinweis ist dagegen eine global Variable. Sie ist im gesamten Dokument gültig und steht jederzeit zur Verfügung. Wenn

Sie innerhalb von Funktionen Variablen ohne das Schlüsselwort var deklarieren, dann sind diese Variablen global.

Jede Variablendeklaration wird mit einem Strichpunkt abgeschlossen, da sie eine ganz normale Anweisung darstellt.

Variablen können mit oder ohne weitere Wertzuweisung deklariert werden. Im obigen Beispiel wird etwa der Variablen SinnDesLebens bei der Definition gleich ein Wert zugewiesen, nämlich 42. Auch die Variable Satzteil erhält eine anfängliche Wertzuweisung, nämlich den Wert „Das Quadrat von „. Die Variablen i und x werden dagegen nicht mit einem Anfangswert versehen. Beim Zuweisen eines Wertes notieren Sie hinter dem Variablennamen ein Istgleichzeichen und dahinter den Wert, den Sie der Variablen zuweisen wollen.

Sie können mehrere Variablen auf einmal definieren, so wie die beiden Variablen i und x im Beispiel. Dazu trennen Sie die Variablennamen durch Kommata. Das ist natürlich auch in Verbindung mit zugewiesenen Anfangswerten möglich.

Es gibt numerische Variablen und Variablen für Zeichen bzw. Zeichenketten. Im obigen Beispiel sind die Variablen SinnDesLebens, i und x numerische Variablen. Die Variablen Hinweis und Satzteil sind dagegen Zeichenkettenvariablen. Dies ist daran erkennbar, dass der ihnen zugewiesene Wert, ein Text, in Anführungszeichen gesetzt wird. Sie könnten z.B. eine Variable Nummer = 1; und eine Variable Zeichen = „1“; definieren. Der Unterschied ist, dass Sie mit der Variablen Nummer Rechenoperationen anstellen können, mit der Variablen Zeichen nicht. Dafür können Sie mit Zeichenkettenvariablen Zeichenkettenoperationen durchführen, etwa das Extrahieren einer Teilzeichenkette, was mit numerischen Variablen nicht möglich ist.
Beachten Sie:

Bei der Vergabe von Variablennamen gelten die Regeln für selbstvergebene Namen.

Variablen in JavaScript sind nicht so streng „getypt“ wie in vielen anderen Programmiersprachen. Einfache Variablentypen, wie Zahlen, Zeichenketten oder Wahrheitswerte, werden lediglich nach nummerischen und nicht-numerischen Variablen eingeteilt. Kommazahlen und Ganzzahlen benötigen keine unterschiedlichen Typen. Der Inhalt von numerischen Variablen kann ohne vorherige Konvertierung in Zeichenketten auf den Bildschirm oder in Meldungsfenster geschrieben werden. Umgekehrt können Sie aber mit Zeichenketten, z.B. Werten aus Formularfeldern, nicht immer automatisch rechnen, sondern müssen sie vorher explizit in Zahlen umwandeln. Für die explizite Typumwandlung gibt es verschiedene objektunabhängige Funktionen.

1.8. Werte von Variablen ändern

Wertänderungen von Variablen sind das A & O bei der Programmierung. Sie werden nur dann erfolgreich eigene Programme schreiben können, wenn Sie jederzeit den Überblick haben, was in einer Variablen an einem bestimmten Punkt des Programmablaufs steht. Besonders, wenn Sie mit bedingten Anweisungen oder Schleifen arbeiten, werden Sie schnell feststellen, wie leicht der Überblick über den aktuellen Zustand einer Variablen verloren gehen kann. Beispiel:

Anzeigebeispiel:

```
<html>
<head>
<title>Sinn des Lebens zum Quadrat</title>
<script type=“text/javascript“>
<!--
  function SchreibeTabelle() {
      var SinnDesLebens = 42;
      var i, x, y;
      var Satzteil = „Das Quadrat von „;
      document.close();
      document.open(„text/html“);
      document.writeln(‚<table border=“1“><tr>‘);
      document.writeln(‚<td bgcolor=“EEEEEE“>Wert<\/td>‘);
      document.writeln(‚<td bgcolor=“EEEEEE“>Wert<sup>2<\/sup><\/
td>‘);
      document.writeln(‚<td bgcolor=“EEEEEE“>Wert<sup>3<\/sup><\/
td><\/tr>‘);
      for(i=1; i <= SinnDesLebens; ++i) {
          x = i * i;
          y = i * i * i;
          document.writeln(‚<tr><td>‘ + i + ‚<\/td>‘);
          document.writeln(‚<td>‘ + x + ‚<\/td>‘);
          document.writeln(‚<td>‘ + y + ‚<\/td><\/tr>‘);
      }
      document.writeln(‚<\/table>‘);
  }
//-->
</script>
</head>
<body onLoad=“SchreibeTabelle()“>
</body>
</html>
```

Erläuterung:

Das Beispiel leert das Anzeigefenster des Browsers und erzeugt dynamisch eine Tabelle mit 3 Spalten und 42 Zeilen plus einer Kopfzeile. In die Zellen der Tabelle werden für die Zahlen von 1 bis 42 der Quadratwert und der Kubikwert geschrieben. Zum dynamischen Schreiben in das Browserfenster benutzt das Script Methoden des Objekts document.

Die Variablen SinnDesLebens und Satzteil werden während des Programmablaufs zwar benutzt, aber ihr Wert wird nicht geändert. Die Variablen i, x und y dagegen ändern ihren Wert laufend. Das liegt daran, dass sie innerhalb einer for-Schleife bei jedem Schleifendurchlauf neue Werte zugewiesen bekommen.

Die Wertzuweisung erfolgt, indem Sie den Variablennamen, dahinter ein Istgleichzeichen und dahinter den gewünschten Wert notieren. Bei dem Wert, den Sie zuweisen, können Sie anstelle einer bestimmten Zahl oder einer Zeichenkette auch Namen anderer Variablen notieren. So wird im Beispiel der Variablen x bei jedem Schleifendurchlauf als Wert das Ergebnis der mit sich selbst multiplizierten Variablen i zugewiesen und y das Ergebnis von i * i * i.

1.9. Vordefinierte JavaScript-Funktionen

Es gibt ein paar Funktionen, die bereits in JavaScript integriert sind. Solche Funktionen können Sie aufrufen, ohne sie selbst zu definieren.
Beispiel:

Anzeigebeispiel:

```
<html>
<head>
<title>Test</title>
<script language=“JavaScript“ type=“text/javascript“>
<!--
function Rechne(Operation) {
  var Ergebnis = eval(Operation);
  alert(„Ergebnis: „ + Ergebnis);
}
//-->
</script>
</head>
<body>
<form name=“Formular“>
<p>Geben Sie eine Rechenaufgabe (z.B. 8*5) ein:</p>
<input type=“text“ name=“Eingabe“>
<input type=“button“ value=“OK“
onClick=“Rechne(document.Formular.Eingabe.value)“>
</form>
</body>
</html>
```

Erläuterung:

Das obige Beispiel zeigt eine HTML-Datei mit einem JavaScript-Bereich, in dem wiederum eine Funktionen Rechne() definiert ist. Innerhalb des Dateikörpers der HTML-Datei ist ein Formular mit einem Eingabefeld notiert. In dem Eingabefeld kann der Anwender

eine Rechenaufgabe eingeben, z.B. 1 + 1 oder (28.76 - 0.00001) * 7. Beim Anklicken des Klickbuttons wird die Funktion Rechne() aufgerufen. Sie erwartet als Parameter eine Rechenaufgabe. Deshalb wird ihr der Inhalt des Formulareingabefeldes beim Aufruf übergeben. Die Funktion Rechne bedient sich zur Berechnung des Ergebnisses der äußerst mächtigen vordefinierten Funktion eval() (eval = evaluate = berechne). Diese kann - unter anderem - Rechenoperationen als solche erkennen und ausrechnen. Das Rechenergebnis wird im Beispiel in einem Meldungsfenster ausgegeben.

1.10 Berechnungsoperatoren

Um mit numerischen Werten Berechnungen durchzuführen, brauchen Sie Berechnungsoperatoren.
Beispiel:

```
<script type=“text/javascript“>
<!--
  var Zwei = 1 + 1;
  var GarNix = 1 - 1;
  var AuchNix = 81 / 3 - 27;
  var WenigerAlsNix = 81 / (3 - 27);
  var SinnDesLebens = 6 * 7;
  var MachtAuchSinn = 84 / 2;
  var x = Jahr % 4;
  if(x == 0)
   Schaltjahr = true;

  /* Besondere Notationen: */

  var Zahl;
  Zahl+=3;
  Zahl++;
  Zahl-=2;
  Zahl--;
  Zahl*=4;
  Zahl/=3;
// -->
</script>
```

Erläuterung:

Mathematische Operatoren notieren Sie mit den dafür üblichen Zeichen. Mit + notieren Sie eine Addition, mit - eine Subtraktion, mit * eine Multiplikation, mit / eine Division. Eine Besonderheit stellt der Operator % dar. Damit wird eine so genannte Modulo-Division durchgeführt. Bei einer Modulo-Division werden zwei Werte dividiert. Das Ergebnis ist jedoch im Gegensatz zur normalen Division nur der Restwert der Division. Wenn Sie z.B. 13 % 5 notieren, erhalten Sie als Ergebnis 3, weil 13 geteilt durch 5 gleich 2 Rest 3 ergibt. Diese 3 ist es, die als Ergebnis einer Modulo-Division herauskommt.

Sie können mehrere Operationen in Reihe notieren. Dabei gilt die übliche „Punkt-vor-Strich-Regel“. Wenn Sie eine andere Regel erzwingen wollen, müssen Sie Klammern verwenden, so wie im vierten der obigen Beispiele.

Die besonderen Notationen, die in den obigen Beispielen vorkommen, können Sie verwenden, wenn Sie Additionen oder Subtraktionen abkürzen wollen:
Zahl+=3; ist eine Abkürzung für Zahl = Zahl + 3;
Zahl++; ist eine Abkürzung für Zahl = Zahl + 1;
Zahl-=2; ist eine Abkürzung für Zahl = Zahl - 2;
Zahl--; ist eine Abkürzung für Zahl = Zahl - 1;

Der Operator ++ wird auch als Inkrematonsoperator bezeichnet, der Operator -- als Dekrementationsoperator.

Operator zur Zeichenkettenverknüpfung

Mit einem einfachen Pluszeichen + können Sie eine Zeichenkette an eine andere anhängen.
Beispiel:

```
<script language=“JavaScript“ type=“text/javascript“>
<!--
 var Vorname = „Stefan „
 var Zuname = „Muenz“
 var Name = Vorname + Zuname + „, der Autor dieses Dokuments“
// -->
</script>
```

Erläuterung:

Sie können sowohl Zeichenkettenvariablen als auch direkte Zeichenkettenangaben mit + aneinanderhängen.

1.11 Einfache Entweder-Oder-Abfrage

Für einfache Entweder-Oder-Bedingungen gibt es eine spezielle Syntax, die Sie alternativ zur if/else-Anweisung verwenden können.

Anzeigebeispiel:

```
<html><head><title>Test</title>
<script type=“text/javascript“>
<!--
function Antwort() {
 var Ergebnis = (document.Formular.Eingabe.value == „42“) ?
„RICHTIG!“ : „FALSCH!“;
 document.Formular.Eingabe.value =
   „Die Antwort ist „ + Ergebnis;
}
// -->
</script>
</head><body>
<h1>Der Sinn des Lebens</h1>
<form name=“Formular“>
<p>Was ist der Sinn des Lebens?</p>
<input type=“text“ name=“Eingabe“ size=“40“>
<input type=“button“ value=“OK“ onClick=“Antwort()“>
</form>
</body></html>
```

Erläuterung:

Das Beispiel enthält eine JavaScript- Funktion namens Antwort(). Aufgerufen wird diese Funktion, wenn der Anwender in dem weiter unten notierten HTML-Formular auf den Klickbutton mit der Aufschrift OK klickt, und zwar mit dem Event-Handler onClick. Die Funktion prüft, ob der Wert des Eingabefeldes im Formular (document.Formular.Eingabe.value) den Wert 42 hat. Abhängig davon wird der Variablen Ergebnis entweder der Wert RICHTIG! oder FALSCH! zugewiesen. Anschließend wird in dem Textfeld des Formulars, das zur Eingabe diente, ein entsprechender Satz zusammengesetzt (siehe dazu auch Operator für Zeichenkettenverknüpfung).
Eine einfache Entweder-Oder-Abfrage wird mit einer Bedingung eingeleitet. Die Bedingung muss in Klammern stehen, im Beispiel (document.Formular.Eingabe.value == „42“). Dahinter wird ein Fragezeichen notiert. Hinter dem Fragezeichen wird ein Wert angegeben, der aktuell ist, wenn die Bedingung erfüllt ist. Dahinter folgt ein Doppelpunkt, und dahinter ein Wert für den Fall, dass die Bedingung nicht erfüllt ist. Da es sich um Werte handelt, die für die Weiterverarbeitung nur sinnvoll sind, wenn sie in einer Variablen gespeichert werden, wird einer solchen Entweder-Oder-Abrage in der Regel eine Variable vorangestellt, im Beispiel die Variable Ergebnis. Der Variablen wird durch diese Art von Anweisung das Ergebnis der Entweder-Oder-Abfrage zugewiesen.
Um Bedingungen zu formulieren, brauchen Sie Vergleichsoperatoren.

2.0. Datensicherungsverfahren

2.1. Vollständiger Bachup

·alle Dateinen werden gespeichert auch Betriebssystemdateien und Anwendungsdateien usw.
·Vorteile: Kompleter Wochenhistorie der Tagessicherung
·Nachteile: Täglich sehr hoher Zeitaufwand für den Backup

2.2. Differentielles Backup

·Nach einem Voll-Backup werden jeden Tag Backups durchgeführt, die bis zu diesem Voll-Backup zurückreichen. Bis der Nächste Voll-Backup durchgeführt wird.
·Vorteile: Kompleter Wochenhistorie der Tagessicherung
·Nachteile: Backup-Zeit wird im laufe der Woche länger

2.3. Inkrementeller Backup

·Nach einem Voll-Backup werden jeden Tag Backups durchgeführt, die nur die Tagesänderungen speichern. Bis der Nächste Voll-Backup durchgeführt wird.
·Vorteile: Kompleter Wochenhistorie der Tagessicherung
Kurze Backup-Zeit auch unter der Woche

2.4. Sicherungsplan Kind

·Anzahl der Bänder: 1-2 Bänder
·Maximale Lagerzeit: 1-2 Tage
·Durchführung: jeden Tag ein Voll-Backup
·Vorteile: Komplete Tageshistorie bzw. Zweitageshistorie
·Nachteil: Magnetbänder verschleisen schneller
Rückgesicherte Daten entsprechen immer nur dem letzten Tag
·Keine rationale Sicherungsmethode

2.5. Sicherungsplan Mutter/Kind

·Anzahl der Bänder: 6 Bänder
·Maximale Lagerzeit: 2 Wochen
·Durchführung: Wochenendes Voll-Backups (Band 1 und 2)
Wochentags Hinzufügende oder Differens Backups
4 Bänder für inkremetelle Sicherung
·Vorteile: Komplete Tageshistorie und Komplete Wochen-Backups
Länger Aufbewahrungsdauer, bis zu 2 Wochen
·Eignet sich für Netzwerkumgebungen bei denen hohe Anforderungen an den Datenschutz gestellt werden.

3.0. Betriebssysteme

3.1. Windows Partitionen

·Primäre Partion: C:\Boot-Manager
C:\WinXP (NTFS)
C:\Win 95-98 (FAT 32)

·Erweiterte Partition: E:\WinXP\Programme (NTFS)
F:\Win 95-98\Programme (FAT 32)
G:\DatenA (FAT 16)
H:\DatenB (NTFS)

3.2. FAT Dateisystem

FAT (File Allocation Tabele): -Dateizuordnungstabellen
-FAT 16 max. 2GByte
-FAT 32 max. 2TByte
-Auf Disketten und Festplatten

-Tabellen enthalten genauen Inhalte der Position und Adressen der Dateien sowie Angaben über den Umfang, die Art der Dateiorganisation und den Zugriff.

3.3. NTFS Dateisystem

NTFS (Zugriffssteuerungsliste): -Gleiches System wie bei FAT
-Zusätzliche Funktionen
-Festlegung der Benutzerrechte

3.4. Linux Architektur

·Kernel:	Kern der verschiedenen Linux Distributionen
·Hauptspeicherverwaltung:	Pysikalischerspeicher wird durch Virtuellenspeicher erweitert. Dabei werden wie bei Windows Programme, die aktuell nicht benötigt werden in Auslagerungsbereiche der Harddisk (Bei Linux Swap-Partitionen, Auslagerungsdateien bei Windows) ausgelagert
·Peripheriegeräte:	Gerätedateien stellen die Verbindung her Geräte müssen wie Laufwerke „gemountet“ werden
·Festplattenverwaltung:	Hierarchisches Dateisystem
·Programm-& Prozessverwaltung:	Linux garantiert, dass einzelne Programme unabhängig von einander laufen

4.0. Netzwerke/Internet

4.1. Netzwerkarten

·MAN`s (Motropolitan Area Network)
·WAN`s (Weitverkehrsnetzwerke)
·LAN`s (Local Area Network)

·Client/ Server:	-Herarchie -Hohe Sicherheit -Hoher Material Aufwand
·Pear to Pear Netz:	-Gleichberechtigt -Wenig Aufwand -Keine Verwaltung -Geringe Sicherheit

4.2 LAN/Subnetze

Komponenten des lokalen Netzes	
Name	Funktion
Netzwerkkarte	Empfang/Senden von Daten; MacIP
Kupfer- oder Glasfaserkabel	Leiter
Switch/Hub 1000/100MB`s	Verteiler/Weiche
Switch/Backbone 1000/1000MB`s	Verteiler/Weiche
Hub	Verteiler
(File Server)	
Kommunikations Firewall	
Router 2000MB`s	

4.3. TCP/IP Referenzmodell

·Besteht aus folgenden Komponenten:

FTP - File Transfer Protocol
HTTP - Hypertext Transfer Protocol
SMTP - Simple Mail Transfer Protocol
DNS - Domain Name System
TFTP - Trivial File Transfer Protocol

TCP/IP-Modell Schichten		
Anwendungs-schicht	TCP (Transmission Control Protocol) UDP (User Data Protocol)	Protokolle
Internetschicht	IP	Netze
Netzschicht	Internet LAN/WAN	

4.4. OSI Referenzmodell

Wichtigste Modell der Netzwerkkommunikation und Datenaustausch

OSI-Modell Schichten		
Anwendungsschicht		Kommuniziert mit dem Benutzer
Darstellungsschicht	Daten	Umsetzung von Zeichen in verständliche Formate
Sitzungsschicht		Baut eine Verbindung zum Netz auf
Transportschicht	Segmente	Teilt Daten in einzelne Segmente auf
Vermittlungsschicht	Päckchen	Versieht Päckchen mit Quell- und Zieladresse (Routing/ IP/ logische Verbindung)
Sicherungsschicht	Frames	Sicherung des Transportes von Gerät zu Gerät (Mac Adresse/ physikalische Verbindung)
Bitübertragungsschicht	Bit	Übertragung von hoher und niedriger Spannung 0/1

5.0 IP-Adressen

5.1 IP-Adressklassen

KL.	Dezimalbereich des Ersten Oktetts	Höherwertige Bits des ersten Oktetts	Netz-/ Host-ID	Standart Subnetzmaske	Anteil der Netze	Host pro Netz (verwendete Adressen)
A	1-126*	0	N.H.H.H.	255.0.0.0	126(2^7-2)	16,777,214 (2^{24}-2)
B	128-191	10	N.N.H.H.	255.255.0.0	16,382 (2^{14}-2)	65,534 (2^{16}-2)
C	192-223	110	N.N.N.H.	255.255.255.0	2,097,150 (2^{21}-2)	254(2^8-2)
D	224-239	1110	Reserviert für Multicasting			
E	240-254	11110	Experimentell, wird für Forschungszwecke verwendet			

*Adresse 127 der Klasse A kann nicht verwendet werden und ist für Prüfschleifen und Diagnoseschleifen reserviert.

5.2. Private IP-Adressen

10.0.0.0 - 10.255.255.255
172.16.0.0 - 172.31.255.255
192.168.0.0 - 192.168.255.255

5.3. Subnetze 200.10.57.0

SN	Netz-adres-se	Subnetzmaske	Subnetzadres-se	Bereich der mögli-chen Host-IP-Adres-sen	Broadcast Adressen
0	0	255.255.255.0	200.10.57.0	1-30	200.10.57.31
1	32	255.255.255.32	200.10.57.32	33-62	200.10.57.63
2	64	255.255.255.64	200.10.57.64	65-94	200.10.57.95
3	96	255.255.255.96	200.10.57.96	97-126	200.10.57.127
4	128	255.255.255.128	200.10.57.128	129-158	200.10.57.159
5	160	255.255.255.160	200.10.57.160	161-190	200.10.57.191
6	192	255.255.255.192	200.10.57.192	193-222	200.10.57.223
7	224	255.255.255.224	200.10.57.224	225-254	200.10.57.256

5.4. Subnetmask

IP-Adresse von Host X	11001000.00000001.00000001.00000101
Subnetmaske 255.255.255.0	11111111.11111111.11111111.00000000
Ergebnis der AND-Operation (200.10.1.0)	11001000.00000001.00000001.00000000

Das Ergebnis der AND-Operation Ergibt die Subnetmask der Adresse mit der die Eigene verglichen wird.