*Java***Script**

Einführung in die Scriptsprache Java**Script** basierend auf Arbeiten von Stefan Münz

Inhaltsverzeichnis

1 JavaScript & HTML

JavaScript ist kein direkter Bestandteil von HTML, sondern eine eigene Programmiersprache. Diese Sprache wurde jedoch eigens zu dem Zweck geschaffen, HTML-Autoren ein Werkzeug in die Hand zu geben, mit dessen Hilfe sich WWW-Seiten optimieren lassen.

JavaScript-Programme werden wahlweise direkt in der HTML-Datei oder in separaten Dateien notiert. Sie werden nicht - wie etwa Java-Programme - kompiliert, sondern als Quelltext zur Laufzeit interpretiert, also ähnlich wie Batch-Dateien bzw. Shellscripts. Dazu besitzen moderne WWW-Browser wie Netscape oder Microsoft Internet Explorer entsprechende Interpreter-Software.

In einer Programmiersprache wie JavaScript gibt es für Anfänger viele verwirrende Elemente: Sonderzeichen, Variablen, Wenn-Dann-Anweisungen, Schleifen, Funktionen, Methoden, Parameter, Objekte, Eigenschaften und anderes mehr. Um mit diesen Elementen richtig umgehen zu können, müssen Sie lernen sich vorzustellen, was im Computer passiert, wenn der Programmcode ausgeführt wird. Das ist ein langwieriger Lernprozess, den Anfänger nur durch viel Übung bewältigen. JavaScript ist dazu allerdings hervorragend geeignet, weil es eine vergleichsweise einfache Sprache ist, bei der viele Aufgabenbereiche einer "grossen" Programmiersprache fehlen, z.B. Dinge wie Arbeitsspeicherverwaltung oder Dateioperationen. Ausserdem setzt JavaScript auf einer bestimmten Umgebung auf, nämlich auf einer anzuzeigenden oder angezeigten WWW-Seite.

Hinweise für die Praxis

Bevor man selber beginnt zu programmieren, sollte man sich im WWW umsehen, ob es nicht schon frei verfügbare JavaScript-Beispiele gibt, die genau Ihr Problem lösen. Denn es ist immer besser, auf Code zurückzugreifen, der sich in der Praxis bereits bewährt hat, als neuen Code zu erstellen, dessen "heimliche Tücken" einem noch nicht bekannt sind. In vielen Fällen genügt es, vorhandene JavaScripts den eigenen Erfordernissen anzupassen. Eine gute Einstiegsadresse für die Suche nach JavaScripts ist das WWW-Angebot ***http://javascript.seite.net/*** *von Johannes Gamperl. Bei vorhandenen JavaScripts müssen Sie allerdings so viel von der Sprache verstehen, dass Sie genau wissen, welche Variablen oder festen Werte Sie verändern oder welche Funktion Sie ergänzen wollen.*

Wenn Sie JavaScript für wichtige WWW-Seiten verwenden wollen, sollten Sie auf jeden Fall mehrere Browser zum Testen einsetzen. Denn leider sind die JavaScript-Interpreter der beiden JavaScript-fähigen WWW-Browser Netscape und MS Internet Explorer sehr unterschiedlich in ihrer Leistung.

Stellen Sie keine WWW-Seiten mit ungeprüftem JavaScript-Code ins WWW. Für einen Anwender ist es sehr ärgerlich, wenn er Fehlermeldungen des JavaScript-Interpreters am Bildschirm erhält und in schlimmeren Fällen einen Programmabsturz des WWW-Browsers oder gar einen Systemabsturz hat, was bei Online-Sitzungen besonders unangenehm ist.

1.1 JavaScript-Bereiche in HTML definieren

JavaScript-Programmabschnitte können Sie im HTML Bereiche definieren.

Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript">
<!--
  alert("Hallo Welt!");
//-->
</script>
</head><body></body></html>
```

Erläuterung:

*Mit **<script language="JavaScript">** leiten Sie einen Bereich für JavaScript innerhalb einer HTML-Datei ein (script = Quelltext, language = Sprache). Dahinter - am besten in der nächsten Zeile - sollten Sie mit **<!--** einen 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.*

*Am Ende eines JavaScript-Bereichs schliessen Sie mit **//-->** den Kommentar und mit **</script>** den Bereich für den Programmcode.*

Beachten Sie:

Es gibt keine festen Vorschriften dafür, an welcher Stelle einer HTML-Datei ein JavaScript-Bereich definiert werden muss. Es ist unter JavaScript-Programmierern zur Gewohnheit geworden, einen solchen Bereich im Kopf der HTML-Datei, also zwischen **<head>** und **</head>** zu definieren. Dadurch ist am ehesten sichergestellt, dass der Code vom WWW-Browser bereits eingelesen ist und zur Verfügung steht, wenn er ausgeführt werden soll.

2 JavaScript Sprachelemente

2.1 Allgemeine Notationsregeln

2.1.1 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: Zuweisung eines Wertes an eine Variable

```
Zahl = 42;
```

Beispiel 2: Durchführen einer Operation mit Variablen oder Werten

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

Beispiel 3: Ausführung einer Anweisung nur unter bestimmten Bedingungen

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

Beispiel 4: Aufrufen einer selbst definierten Funktion oder Objektmethode

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

Erläuterung:

Eine Anweisung in JavaScript besteht immer aus einem Befehl, der mit einem Strichpunkt **`;`** *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 vorne herein anzugewöhnen, alle Anweisungen auf diese Weise abzuschliessen.*

2.2 Anweisungsblöcke notieren

Ein Anweisungsblock besteht aus zwei oder mehreren Anweisungen, innerhalb einer übergeordneten Anweisung oder innerhalb einer Funktion stehen. So können Anweisungsblöcke beispielsweise innerhalb einer bedingten Anweisung, innerhalb 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** oben) oder bei Schleifen (wie in **Beispiel 2** oben) 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** oben) müssen Sie Anfang und Ende der Funktion durch geschweifte Klammern markieren. Alles, was innerhalb der Funktion steht, ist daher ein Anweisungsblock.*

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

2.2.1 Selbstvergebene Namen

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

Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript">
  var DiesIstEinVariableName;
  var Dies_Auch;

  function Dies_Ist_Ein_Funktionsname()
  {
    alert("Wir sind in einer Funktion");
  }
</script>
</head><body></body></html>
```

Erläuterung:

Bei selbst vergebenen Namen gelten folgende Regeln:

- *sie dürfen keine Leerzeichen enthalten*
- *sie sollten maximal 32 Zeichen Länge haben*
- *sie dürfen nur aus Buchstaben und Ziffern bestehen - das erste Zeichen sollte 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 entsprechende 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.

2.2.2 Kommentare in JavaScript

Bei komplexeren Programmteilen können Sie Kommentare benutzen, um einzelne Anweisungen zu erklären. Auch wenn Sie Ihre Copyright-Angaben 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:

```
/* Dies ist ein langer JavaScript-Kommentar, der sich über mehrere
Zeilen
   hinweg erstreckt i */
// Dies ist ein Kommentar, der nur auf dieser Zeile gültig ist
```

Erläuterung:

Einen Kommentar, der über mehrere Programmzeilen geht 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. Kommentare, die kürzer als eine Zeile sind, können mit doppeltem Schrägstrich `//` begonnen werden, wie in C++ und Java.

2.3 Variablen und Werte

2.3.1 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:

```
<html><head><title>Sinn des Lebens zum Quadrat</title>
<script language="JavaScript">
  // Variablen- und Funktionsdefinitionen
  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;
      /* mit dem Funktionsaufruf document.write() wird der Inhalt in
den
         Klammern direkt auf die HTML-Seite geschrieben. Der HTML-Tag
<BR>
         erzwingt einen Zeilenumbruch */
      document.write(Satzteil + i + " ist " + x + "<br>");
    }
  }
</script></head><body>
<script language="JavaScript">
  // Aufruf der Funktion
  SchreibeQuadrate();
```

```
</script></body></html>
```

Erläuterung:

*Variablen können mit Hilfe des reservierten Wortes **var** definiert werden. Dieses Schlüsselwort ist zwar nicht zwingend notwendig, macht den Quellcode aber lesbarer, da es signalisiert, dass an der betreffenden Stelle eine neue Variable definiert wird. Jede Variablendefinition wird mit einem Strichpunkt abgeschlossen, genau wie eine Anweisung (betrachten Sie eine Variablendefinition einfach als eine Anweisung!).*

*Es gibt globale Variablen und Variablen, die innerhalb einer Funktion definiert sind. Im obigen Beispiel ist die Variable **Hinweis** global, während die Variablen **SinnDesLebens, i, x** und **Satzteil** funktionsgebunden sind. Eine globale Variable steht jederzeit zur Verfügung, während eine Variable, die innerhalb einer Funktion definiert wird, nur innerhalb dieser Funktion zur Verfügung steht. Man spricht in diesem Zusammenhang auch von der "Lebensdauer" von Variablen.*

*Variablen können mit oder ohne weitere Wertzuweisung definiert 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 Ist-Gleich-Zeichen (**=**) 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. Ausser der Unterscheidung von numerischen und nicht-numerischen Variablen gibt es keine Unterscheidung. 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.

2.3.2 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.

Erläuterung:

*Im letzten Beispiel werden die Variablen **SinnDesLebens** und **Satzteil** während des Programmablaufs zwar benutzt, aber ihr Wert wird nicht geändert. Die Variablen **i** und **x** 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 Ist-Gleich-Zeichen (**=**) und dahinter den gewünschten Wert notieren. Bei dem Wert, den Sie zuweisen, können Sie an Stelle 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.*

Aufgabe 1:

- Ändern Sie das letzte Programm so ab, dass nur die Quadratwerte für die ungeraden Zahlen dargestellt werden.

2.4 Objekte, Eigenschaften und Methoden

2.4.1 Vordefinierte JavaScript-Objekte

Objekte sind fest umgrenzte Datenelemente mit Eigenschaften und oft auch mit objektgebundenen Funktionen (Methoden).

Dass JavaScript eine Erweiterung von HTML darstellt, liegt vor allem an den vordefinierten Objekten, die Ihnen in JavaScript zur Verfügung stehen. Über diese vordefinierten Objekte können Sie beispielsweise einzelne Elemente eines HTML-Formulars abfragen oder ändern.

Ein Objekt kann eine Teilmenge eines übergeordneten Objekts sein. Die JavaScript-Objekte sind deshalb in einer Hierarchie geordnet. Das hierarchiehöchste Objekt ist in JavaScript das Fenster-Objekt (**`window`**). Fenster haben Eigenschaften wie einen Titel, eine Grösse usw. Der Inhalt eines Fensters ist das nächst niedrigere Objekt, nämlich das im Fenster angezeigte Dokument (in JavaScript das Objekt **`document`**). In der Regel ist der Fensterinhalt eine HTML-Datei. Eine solche Datei kann bestimmte, durch HTML-Tags definierte Elemente enthalten, wie zum Beispiel Formulare, Verweise, Grafikreferenzen usw. Für solche untergeordneten Elemente gibt es wieder eigene JavaScript-Objekte, zum Beispiel das Objekt **`forms`** für Formulare. Ein Formular besteht seinerseits aus verschiedenen Elementen, zum Beispiel aus Eingabefeldern, Auswahllisten oder Buttons zum Absenden bzw. Abbrechen. In JavaScript gibt es dafür ein Objekt **`elements`**, mit dem Sie einzelne Felder und andere Elemente innerhalb eines Formulars ansprechen können.

Neben den hierarchisch geordneten JavaScript-Objekten gibt es auch solche, die nicht direkt in die Hierarchie passen. Das sind zum Beispiel Objekte für Datums- und Zeitrechnung, für mathematische Aufgaben oder für Zeichenkettenverarbeitung.

Eine Übersicht der vordefinierten JavaScript-Objekte finden Sie in der JavaScript Objekt-Referenz. Dort werden zu jedem Objekt die verfügbaren Eigenschaften und Methoden vorgestellt.

2.4.2 Vordefinierte JavaScript-Objekte verwenden

JavaScript-Objekte können Sie jederzeit verwenden.

Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript">
  function SagDatumUndZeit()
  {
    var Jetzt = new Date();
    var Tag = Jetzt.getDate();
    var Monat = Jetzt.getMonth() + 1;
    var Jahr = Jetzt.getYear();
    var Stunden = Jetzt.getHours();
    var Minuten = Jetzt.getMinutes();
    var NachVoll  = ((Minuten < 10) ? ":0" : ":");
    alert("Guten Tag!\nHeute ist der "
    + Tag + "." + Monat + "." + Jahr + "\nEs ist jetzt "
    + Stunden + NachVoll + Minuten + " Uhr");
  }
</script></head>
<body>
<script language="JavaScript">
  // Aufruf der Funktion
  SagDatumUndZeit();
</script></body></html>
```

Erläuterung:

*Im Beispiel wird innerhalb eines JavaScript-Bereichs mit Hilfe des vordefinierten JavaScript-Objekts **Date** das aktuelle Datum und die aktuelle Uhrzeit ermittelt. Beim Einlesen der Datei wird das Ergebnis zusammen mit einem Begrüssungstext sauber formatiert in einem Meldungsfenster ausgegeben.*

*Zuerst muss dazu eine neue Variablen angelegt werden. Im Beispiel ist dies die Variable **Jetzt**. Diese Variable soll auf Daten des **Date**-Objekts zugreifen können. Dies geschieht durch ein Ist-Gleich-Zeichen (**=**) hinter dem Variablennamen. Dahinter folgt das reserviert JavaScript Wort **new** und dahinter, durch ein Leerzeichen getrennt, der Aufruf von **Date()** zum Erzeugen einer neuen Instanz des Objekts **Date**.*

*Um die einzelnen Daten der Objektinstanz von **Date**, also Tag, Monat, Jahr usw. anzusprechen, stehen entsprechende Methoden zur Verfügung. Diese Methoden, z.B. **getDate()** oder **getHours()**, haben als Rückgabewert jeweils einen Datums/Uhrzeit-Bestandteil. So liefert **getDate()** beispielsweise den aktuellen Tag des Monats und **getHours()** die aktuelle Stundenzahl des Tages. Im Beispiel wird für jeden der benötigten Bestandteile eine Variable definiert. In der Variablen **Tag** wird beispielsweise durch Aufruf von **Jetzt.getDate()** der aktuelle Tag des Monats gespeichert.*

*Die Anweisung im Beispiel, die mit **NachVoll ...** beginnt, kann an dieser Stelle nicht näher erläutert werden. Die Anweisung ist eine Vorbereitung zur sauberen Formatierung der Uhrzeit.*

Beachten Sie:

*Die einzelnen Methoden des im Beispiel verwendeten Objekts **Date**, wie zum Beispiel **getDate()**, werden bei der Referenz des Objekts **Date** beschrieben.*

Aufgabe 2:

- Führen Sie das oben beschriebene Beispiel aus
- Suchen Sie das vordefinierte JavaScript-Objekt zur Manipulation für Bilder.

2.4.3 Eigenschaften von Objekten

Objekte können Eigenschaften haben. Ein selbst definiertes Objekt "Mensch" könnte zum Beispiel die Eigenschaften Name, Alter, Geschlecht und Muttersprache haben. Vordefinierte JavaScript-Objekte haben ebenfalls Eigenschaften. So hat das Objekt **Math** zum Beispiel eine Eigenschaft "PI" (**Math.PI**). Auf diese Weise lässt sich mit der mathematischen Konstante PI rechnen, ohne deren genauen Wert zu kennen.

Eigenschaften von Objekten können Sie innerhalb Ihres JavaScript-Codes jederzeit auslesen, und in vielen Fällen können Sie die Werte von Eigenschaften auch ändern. So können Sie beispielsweise dem im Browser-Fenster angezeigten Dokument eine neue, gültige URL-Adresse zuweisen. Dadurch bewirken Sie, dass der WWW-Browser einen Sprung zu der zugewiesenen URL-Adresse ausführt, genau so, wie wenn der Anwender auf einen entsprechenden Verweis klicken würde.

Beispiel:

```
<html><head><title>Browser des Anwenders auslesen</title>
<script language="JavaScript">
  var BrowserName = navigator.appName;
  var BrowserVersion = navigator.appVersion;
  alert("Ah ja, Sie verwenden also den " + BrowserName +
  ", und zwar in der Version " + BrowserVersion);
</script>
</head><body></body></html>
```

Erläuterung:

Im Beispiel werden innerhalb einesJavaScript-Bereichs zwei Eigenschaften des vordefinierten JavaScript-Objekts ***navigator*** *in zwei entsprechenden Variablen gespeichert. Das* ***navigator****-Objekt stellt Ihnen über seine Eigenschaften verschiedene Informationen über den verwendeten WWW-Browser des Anwenders zur Verfügung. Im obigen Beispiel werden die Eigenschaften des Browser-Namens (gespeichert in der Objekteigenschaft* ***navigator.appName****) und der Browser-Version (gespeichert in der Objekteigenschaft* ***navigator.appVersion****) ermittelt. Anschliessend werden die ermittelten Daten in einem Meldungsfenster am Bildschirm ausgegeben.*

Objekteigenschaften sprechen Sie an, indem Sie zuerst den Namen des Objekts notieren, dahinter einen Punkt (**.** *), und dahinter den Namen der Eigenschaft. Dabei sind keine Leerzeichen erlaubt!*

Aufgabe 3:

- Schreiben Sie alle JavaScript-Eigenschaften des Obkjets **navigator** auf.

2.4.4 Objekte-Methoden

Objekte können Methoden haben. Das sind Funktionen, die Aktionen ausführen, aber im Gegensatz zu allein stehenden Funktionen an ein bestimmtes Objekt gebunden sind. Viele vordefinierte JavaScript-Objekte haben Methoden. So gibt es zum Beispiel das vordefinierte JavaScript-Objekt **document**, das sich auf den Inhalt, der in einem Browser-Fenster angezeigt wird bezieht. Dazu gibt es eine Methode **document.write()**, mit der ein Text in ein Browser-Fenster geschrieben werden kann.

Beispiel:

```
<html><head><title>Methoden</title></head>
<body>
<script language="JavaScript">
  document.write("Aufruf der Methode write der Klasse document");
</script></body></html>
```

Erläuterung:

Das Beispiel enthält einen Verweis mit einer speziellen Syntax. Diese Syntax erlaubt Ihnen, beim Anklicken des Verweises JavaScript-Code aufzurufen. Im Beispiel ist das ein Aufruf der Methode ***`write()`*** *des Objekts* ***`document`****.*

Objektmethoden sprechen Sie an, indem Sie zuerst den Namen des Objekts notieren, dahinter einen Punkt, dahinter den Namen der Methode, und dahinter eine öffnende und eine schliessende Klammer. Dabei sind keine Leerzeichen erlaubt! Einige Methoden können auch Parameter beim Aufruf erwarten. Diese Parameter müssen Sie dann zwischen der öffnenden und der schliessenden Klammer übergeben.

Aufgabe 4:

- Führen Sie das oben beschriebene Beispiel aus.
- Schreiben Sie alle JavaScript-Methoden des Obkjets **`document`** auf.

2.5 Funktionen

2.5.1 Funktionen definieren

Mit Hilfe von Funktionen können Sie eigene, in sich abgeschlossene JavaScript-Prozeduren programmieren, die Sie dann über den Aufruf der Funktion ausführen können. JavaScript-Code, der nicht innerhalb einer Funktion steht, wird beim Einlesen der Datei vom WWW-Browser sofort ausgeführt!

Eine Funktion ist ein Anweisungsblock. Sie können eigene Funktionen innerhalb einesJavaScript-Bereiches oder in einer separaten JavaScript-Datei definieren. An erlaubten Stellen, z.B. innerhalb der einleitenden HTML-Tags **`<body...>`** und **`<a href...>`**, oder in einem Formular-Tag wie **`<input...>`** , können Sie eine solche selbst definierte Funktion dann mit Hilfe eines Event-Handlers (siehe weiter unten) aufrufen. Oder Sie rufen eine Funktion innerhalb einer anderen Funktion auf.

Beispiel:

```
function PrimzahlCheck(Zahl)
{
  var Grenzzahl = Zahl / 2;
  var Check = 1;
  for(i = 2; i <= Grenzzahl; i++)
  {
    if(Zahl % i == 0)
    {
     alert(Zahl + " ist keine Primzahl, weil teilbar durch " + i);
     Check = 0;
    }
  }
  if(Check == 1) alert(Zahl + " ist eine Primzahl!");
}
```

Erläuterung:

*Mit dem Schlüsselwort **`function`** leiten Sie die Definition einer Funktion ein. Dahinter folgt, durch ein Leerzeichen getrennt, ein frei wählbarer Funktionsname, im Beispiel: **`PrimzahlCheck`**. Vergeben Sie einen Funktionsnamen, der das, was die Funktion leistet, ungefähr beschreibt. Beachten Sie dabei die Regeln für selbst vergebene Namen.*

*Unmittelbar hinter dem Funktionsnamen folgt eine öffnende Klammer (**`(`**). Wenn die Funktion keine Parameter erwarten soll, notieren Sie dahinter sofort wieder eine schliessende Klammer (**`)`**). Wenn die Funktion Parameter übergeben bekommen soll, notieren Sie innerhalb der Klammer die Namen der Parameter. Die Namen der Parameter sind frei wählbar. Bei den Parameternamen gelten die gleichen Regeln wie beim Funktionsnamen. Mehrere Parameter werden durch Kommata voneinander getrennt. Im obigen Beispiel erwartet die Funktion **`PrimzahlCheck`** einen Parameter **`Zahl`**.*

*Der gesamte Inhalt der Funktion wird in geschweifte Klammern (**`{`**) und (**`}`**) eingeschlossen. Diese Klammern dürfen niemals fehlen!*

Die Anweisungen innerhalb der Funktion können sehr unterschiedlicher Natur sein. Da können Sie z.B. Objekte manipulieren, übergebene Parameter verarbeiten und ändern, Berechnungen anstellen usw. Sie können innerhalb von Funktionen auch andere Funktionen aufrufen. Welche Anweisungen innerhalb einer Funktion stehen, hängt davon ab, was die Funktion leisten soll.

Aufgabe 5:

- Beschreiben Sie, was in der oben stehenden Funktion genau geschieht.

2.5.2 Funktionen aufrufen

Sie können eine selbst definierte Funktion aufrufen, um den darin enthaltenen JavaScript-Code auszuführen. Die Funktion rufen Sie mit ihrem Funktionsnamen auf. Dahinter folgt die öffnende Klammer. Wenn die Funktion keine Parameter erwartet, notieren Sie hinter der öffnenden gleich eine schliessende Klammer. Wenn die Funktion Parameter erwartet, müssen Sie für jeden Parameter einen erlaubten Wert übergeben.

Beispiel:

```
// Funktionsaufruf
PrimzahlCheck(Zahl);
```

Aufgabe 6:

- Schreiben Sie ein vollständiges HTML-Dokument, indem die oben erwähnte Funktion **`PrimzahlCheck`** eingebaut wird.

2.5.3 Funktionen mit Rückgabewert und solche Funktionen aufrufen

Eine Funktion kann einen ermittelten Wert an die aufrufende Instanz zurückgeben. Dies geschieht mit dem **`return`**-Befehl.

Beispiel:

```
<html><head><title>Bruttobetrag aus Nettobetrag errechnen</title>
<script language="JavaScript">
  function BruttoBetrag(Netto, Prozente)
  {
    var Ergebnis = Netto * (1 + (Prozente / 100));
    return Ergebnis;
  }

  function SchreibeBrutto(Betrag, Prozentsatz)
  {
    var Wert;
    Wert = BruttoBetrag(Betrag, Prozentsatz);
    alert("Der Bruttobetrag von " + Betrag + " ist bei " + Prozentsatz
+ " = " + Wert);
  }
</script></head>
<body>
<script language="JavaScript">
  Wert = 1000;
  Prozentsatz = 5;
  SchreibeBrutto(Wert, Prozentsatz);
</script></body></html>
```

Aufgabe 7:

- Ändern Sie das Beispiel so ab, dass der berechnete Wert direkt in der **`alert`**-Anweisung aufgerufen wird und das **`Ergebnis`** direkt in der **`return`**-Anweisung berechnet wird.

2.6 Steuerzeichen und besondere Notationen

2.6.1 Steuerzeichen bei Zeichenketten

Bei Zeichenkettenvariablen gibt es die Möglichkeit, Steuerzeichen in den Variablenwert einzufügen.

Beispiel:

```
<script language="JavaScript">
  var Variable1 = "Hier erfolgt ein\nZeilenumbruch";
  var Variable2 = "Hier erfolgt ein\fWagenrücklauf";
  var Variable3 = "Hier erfolgt ein\bBackspace";
  var Variable4 = "Hier erfolgt ein\rDOS-Extra-Zeilenumbruch";
  var Variable5 = "Hier erfolgt ein\tTabulator";
  var Variable6 = "Hier erfolgt ein\"Anführungszeichen";
</script>
```

Erläuterung:

Steuerzeichen dieser Art werden durch das Zeichen **`\`** *eingeleitet. Dahinter folgt ein Buchstabe, der das Steuerzeichen markiert.*

Das Steuerzeichen **`\n`** *ist z.B. in* ***`alert(...)-`*** *Meldungen sinnvoll, um innerhalb des auszugebenden Textes einen Zeilenumbruch einzufügen.*

Das Steuerzeichen **`\t`** *ist z.B. sinnvoll, um etwa innerhalb einer* ***`alert(...)-`*** *Meldung tabellarische Information auszugeben.*
Das Steuerzeichen **`\r`** *sollten Sie zusätzlich zu dem Steuerzeichen* **`\n`** *notieren, wenn* **`\n`** *alleine nicht funktioniert.*
Das Steuerzeichen **`\"`** *müssen Sie notieren, wenn Sie innerhalb einer Zeichenkette ein Anführungszeichen verwenden möchten.*

Aufgabe 8:

- Schreiben Sie ein sinnvolles, funktionierendes JavaScript-Programm, dass alle oben erwähnten Steuerzeichen beinhaltet.

2.6.2 Notation numerischer Werte

Sie können Zahlen ganz normal notieren. Beachten Sie dabei nur, dass bei Kommazahlen an Stelle eines Kommas ein Punkt notiert werden muss. So wird die Zahl Pi beispielsweise als 3.1415 notiert. Für sehr hohe und sehr niedrige Zahlen und für komplexe Kommazahlen gibt es daneben aber noch andere Notationsmöglichkeiten.

Beispiel:

```
<script language="JavaScript">
  var a = 1E1;
  var b = 1.2345E4;
  var c = 2e-3;
</script>
```

Erläuterung:

*Mit **e** oder **E** bestimmen Sie die Anzahl Nullen, die hinter der Zahl vor dem **e** bzw. **E** stehen.*

*Die erste Zahl im Beispiel, **1E1**, ist eine **1** mit einer **0** dahinter, also **10**.*
*Die zweite Zahl im Beispiel, **1.2345E4**, ist eine andere Schreibweise für **12345**. Der Dezimalpunkt wird also einfach um so viele Stellen nach rechts verschoben, wie hinter dem **E**-Zeichen angegeben.*
*Die dritte Zahl im Beispiel, **2e-3**, ist eine andere Schreibweise für **0.002**. Der Dezimalpunkt wird also einfach um so viele Stellen nach links verschoben, wie hinter dem **E**-Zeichen angegeben. Diese umgekehrte Richtung wird durch das Minuszeichen bewirkt, dass hinter dem **e** folgt.*

Aufgabe 9:

- Schreiben Sie ein sinnvolles, funktionierendes JavaScript-Programm, dass alle oben erwähnten Notationen für numerische Werte enthält.

2.7 Operatoren

2.7.1 Zuweisungsoperator

Sie können zum Beispiel einer Variablen einen Wert zuweisen. Der Zuweisungsoperator dafür ist ein Ist-Gleich-Zeichen (**=**).

Beispiel:

```
<script language="JavaScript">
  var SinnDesLebens = 42;
</script>
```

Erläuterung:

*Im Beispiel wird eine Variable **SinnDesLebens** definiert. Der Variablen wird mit dem Zuweisungsoperator (**=**) der Wert 42 zugewiesen.*

2.7.2 Vergleichsoperatoren

Vergleichsoperatoren brauchen Sie, wenn Sie zwei Werte vergleichen wollen, z.B. den aktuellen Inhalt einer Variablen mit einem fixen Wert.

Beispiel:

```
<script language="JavaScript">
  if(SinnDesLebens == 42) return 1;
  if(SinnDesLebens != 42) return 0;
  if(SinnDesLebens > 42) return 0;
  if(SinnDesLebens < 42) return 0;
  if(Alter >= 18) alert("SIE duerfen das hier sehen!");
  if(Alter <= 17) alert("SIE duerfen das hier NICHT sehen!");
</script>
```

Erläuterung:

Um abzufragen, ob zwei Werte gleich sind, notieren Sie zwei Ist-Gleich-Zeichen **==** *nebeneinander.*
Um abzufragen, ob zwei Werte unterschiedlich sind, notieren Sie zwischen beiden Werten die Zeichen ***!=****.*
Um abzufragen, ob ein Wert grösser oder gleich ist als ein anderer, notieren Sie die Zeichen **>=***.*
Um abzufragen, ob ein Wert in jedem Fall grösser ist als ein anderer, notieren Sie das Zeichen **>***.*
Um abzufragen, ob ein Wert kleiner oder gleich ist als ein anderer, notieren Sie die Zeichen **<=***.*
Um abzufragen, ob ein Wert in jedem Fall kleiner ist als ein anderer, notieren Sie das Zeichen **<***.*
Nähere Information zu der ***If****-Abfrage erhalten Sie im Abschnitt über bedingte Anweisungen.*

2.7.3 Berechnungsoperatoren

Um mit numerischen Werten Berechnungen durchzuführen, brauchen Sie Berechnungsoperatoren.

Beispiel:

```
<script language="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;
  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 Inkrementationsoperator bezeichnet, der Operator **--** als Dekrementationsoperator*

Aufgabe 10:

- Schreiben Sie ein sinnvolles, funktionierendes JavaScript-Programm, dass die Inhalte der oben erwähnten Variablen bei jeder Änderung des Variableninhalts auf den Bildschirm ausgeben.

2.7.4 Logische Operatoren

Logische Operatoren brauchen Sie, wenn Sie komplexere Bedingungen für bedingte Anweisungen oder Schleifen formulieren wollen.

Beispiel:

```
<script language="JavaScript">
  if(PLZ >= 8000 && PLZ <= 8200)
   alert("Sie wohnen wohl in Zuerich oder Umgebung!")

  if(x > 100 || y == 0)
   break;
</script>
```

Erläuterung:

*Mit dem logischen Operator **&&** verknüpfen Sie zwei oder mehrere Bedingungen durch "und", d.h. beide bzw. alle Bedingungen müssen erfüllt sein, damit die gesamte Bedingung erfüllt ist.*

*Mit dem logischen Operator **||** verknüpfen Sie zwei oder mehrere Bedingungen durch "oder", d.h. es genügt, wenn eine der Bedingungen erfüllt ist, damit die gesamte Bedingung erfüllt ist.*

2.7.5 Bit-Operatoren

Bit-Operatoren sind nur etwas für Profis. Um Bit-Operatoren richtig einzusetzen, müssen Sie viel von computerinternen Speichervorgängen verstehen. Deshalb werden die Bit-Operatoren hier nur kurz erwähnt.

- **>>** verschiebt Bits nach rechts
- **<<** verschiebt Bits nach links
- **&** definiert in einer Bitmaske eine Und-Bedingung
- **|** definiert in einer Bitmaske eine inklusive Oder-Bedingung
- **^** definiert in einer Bitmaske eine exklusive Oder-Bedingung

2.7.6 Operatoren zur Zeichenverknüpfung

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

Beispiel:

```
<script language="JavaScript">
  var Vorname = "Hans " ;
  var Zuname = "Muster " ;
  var Name = Vorname + Zuname + ", der Autor dieses Dokuments" ;
</script>
```

Erläuterung:

*Sie können sowohl Zeichenkettenvariablen als auch direkte Zeichenkettenangaben mit **+** aneinander hängen*.

2.7.7 Operatorenrangfolge

Unter den Operatoren von JavaScript gibt es eine festgelegte Rangordnung. Wenn Sie komplexe Rechenoperationen durchführen oder mehrere Bedingungen miteinander verknüpfen, gilt bei der internen Auflösung solcher komplexen Ausdrücke die folgende Rangordnung:

Rangstufe: **,** (Aneinanderreihung)
Rangstufe: **= += -= <<= >>= &= ^= |=**
Rangstufe: **?:** (Entweder-Oder-Bedingung)
Rangstufe: **||**
Rangstufe: **&&**
Rangstufe: **|**
Rangstufe: **^**
Rangstufe: **&**
Rangstufe: **== !=**
Rangstufe: **< <= > >=**
Rangstufe: **<< >> >>>**
Rangstufe: **+ -**
Rangstufe: *** / %**
Rangstufe: **! ~ - ++ --**
Rangstufe: **() [] .** (Klammerung und Vektoren)

Mit Hilfe von Klammern, die absichtlich die unterste Rangstufe in der Prioritätshierarchie darstellen, können Sie die Rangfolge bei den Operatoren beeinflussen und Ausdrücke so bewerten, wie Sie es wünschen.

Beispiel:

```
<script language="JavaScript">
  var OffizielleStatistik = 3.29 * 3 + 4.71;
  var MeineStatistik = 3.29 * (3 + 4.71);
</script>
```

Erläuterung:

Das Beispiel zeigt, wie Sie durch Setzen von Klammern das Ergebnis einer Rechenoperation beeinflussen können.

Aufgabe 11:

- Schreiben Sie ein sinnvolles, funktionierendes JavaScript-Programm, dass die Inhalte der oben erwähnten Variablen auf den Bildschirm ausgeben.
- Erklären Sie das Resultat

2.8 Bedingte Anweisungen (if-else / switch)

2.8.1 Wenn-Dann-Bedingungen mit “if“

Sie können die Ausführung von Anweisungen von Bedingungen abhängig machen.

Beispiel:

```
<script language="JavaScript">
  var Passwort = "Traumtaenzer";

  UserEingabe = window.prompt("Bitte geben Sie das Passwort ein","");
  if(UserEingabe != Passwort)
  {
    alert("Falsches Passwort!");
  }
  else
  {
    alert("Richtiges Passwort!");
  }
</script>
```

Erläuterung:

*Mit **`if`** leiten Sie eine Wenn-Dann-Bedingung ein (if = wenn). Dahinter folgt, in Klammern stehend, die Formulierung der Bedingung. Um solche Bedingungen zu formulieren, brauchen Sie Vergleichsoperatoren und in den meisten Fällen auch Variablen. Für Fälle, in denen die Bedingung nicht erfüllt ist, können Sie einen "andernfalls"-Zweig definieren. Dies geschieht durch **`else`** (else = sonst).*

*Der **`else`**-Zweig ist nicht zwingend erforderlich. Wenn Sie mehr als eine Anweisung unterhalb und abhängig von `if` oder **`else`** notieren wollen, müssen Sie die Anweisungen in geschweifte Klammern einschliessen (siehe auch den Abschnitt über Anweisungsblöcke.*

Das obige Beispiel stellt eine einfache Passwortabfrage dar.

Aufgabe 12:

- Vervollständigen Sie das oben stehende Programm damit es auch funktionstüchtig ist.
- Erklären sie den Befehl **`window.prompt`** mit Hilfe der SELFHTML-Dokumentation.

2.8.2 Einfache Entweder-Oder-Abfrage

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

Beispiel:

```
<script language="JavaScript">
  var DuBistEin = (Antwort == "42") ? "Genie" : "Dummkopf";
  alert("Deine Antwort zeigt mir, dass Du ein " + DuBistEin + " bist!");
</script>
```

Erläuterung:

Eine einfache Entweder-Oder-Abfrage wird mit einer Bedingung eingeleitet. Die Bedingung muss in Klammern stehen, im Beispiel ***`(Antwort == "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-Abfrage in der Regel eine Variable vorangestellt, im Beispiel die Variable* ***`DuBistEin`****. Der Variablen wird durch diese Art von Anweisung das Ergebnis der Entweder-Oder-Abfrage zugewiesen.*

Um Bedingungen zu formulieren, brauchen Sie Vergleichsoperatoren.

Aufgabe 13:

- Vervollständigen Sie das oben stehende Programm damit es auch funktionstüchtig ist.

2.8.3 Fallunterscheidung mit "switch"

Mit **`if`** und **`else`** können Sie genau zwei Fälle unterscheiden. Wenn Sie feiner differenzieren, also zwischen mehreren Fällen unterscheiden wollen, können Sie zwar mehrere **`if`**-Abfragen hintereinander notieren, aber es gibt noch eine elegantere Möglichkeit: die Fallunterscheidung mit "**`switch`**

Beispiel:

```
<script language="JavaScript">
  Eingabe = window.prompt("Zahl zwischen 1 und 4","");
  switch(Eingabe)
  {
    case "1": alert("Sie sind sehr bescheiden");
              break;
    case "2": alert("Sie sind ein aufrichtiger Zweibeiner");
              break;
    case "3": alert("Sie haben ein Dreirad gewonnen");
              break;
    case "4": alert("Gehen Sie auf allen Vieren " +
                    "und werden Sie bescheidener");
             break;
    default: alert("Sie bleiben leider dumm");
             break;
  }
</script>
```

Erläuterung:

*Mit **`switch`** leiten Sie eine Fallunterscheidung ein (switch = Schalter). Dahinter folgt, in runde Klammern eingeschlossen, eine Variable oder ein Ausdruck, für dessen aktuellen Wert Sie die Fallunterscheidung durchführen. Im Beispiel ist das die Variable mit dem Namen **`Eingabe`**. Diese Variable wird vor der Fallunterscheidung mit einem Wert versorgt, denn ein Dialogfenster (**`window.prompt()`**) mit einem Eingabefeld fordert den Anwender auf, eine Zahl zwischen 1 und 4 einzugeben. Der eingegebene Wert wird in `Eingabe` gespeichert.*

*Die einzelnen Fälle, die Sie abfragen möchten, werden innerhalb geschweifter Klammern notiert. Jeden einzelnen Fall leiten Sie mit **`case`** ein (case = Fall). Dahinter folgt die Angabe des Wertes, auf den Sie prüfen wollen. Die Anweisung **`case "1":`** im obigen Beispiel bedeutet dann so viel wie: 'wenn die Variable Eingabe den Wert "1" hat'. Im Beispiel wird für jeden Fall eine individuelle Meldung ausgegeben.*

*Wichtig ist dabei auch das Wort **`break`** am Ende jedes Falls (break = abbrechen). Wenn Sie das Wort weglassen, werden nämlich alle nachfolgenden Fälle auch ausgeführt, aber das wollen Sie ja in der Regel nicht.*

*Für den Fall, dass keiner der definierten Fälle zutrifft, können Sie am Ende der Fallunterscheidung den Fall **`default`**: definieren. Die darunter stehenden Anweisungen werden ausgeführt, wenn keiner der anderen Fälle zutrifft.*

Aufgabe 14:

- Vervollständigen Sie das oben stehende Programm damit es auch funktionstüchtig ist.

2.9 Schleifen (while / for / do-while)

2.9.1 Schleifen mit “while“

Mit Hilfe von **`while`**-Schleifen können Sie Programmanweisungen solange wiederholen, wie die Bedingung, die in der Schleife formuliert wird, erfüllt ist.

Beispiel:

```
<script language="JavaScript">
  var Passwort = "Traumtaenzer";
  var UserEingabe = "";
  var Zaehler = 0;
  while((UserEingabe != Passwort)&&(Zaehler <= 3))
  {
    UserEingabe = window.prompt(Zaehler + ". Versuch: geben Sie das
Passwort ein","");
    Zaehler++;
  }
  if(UserEingabe != Passwort)
  {
    alert("Falsches Passwort!");
  }
  else
  {
    alert("Richtiges Passwort!");
  }
</script>
```

Erläuterung:

*Eine **`while`**-Schleife beginnt mit dem Wort `while` (while = solange). Dahinter folgt, in Klammern stehend, die Bedingung. Um eine Bedingung zu formulieren, brauchen Sie Vergleichsoperatoren. Der Inhalt der Schleife wird solange wiederholt, wie die Schleifenbedingung wahr ist.*

*In der Regel enthält eine **`while`**-Schleife mehrere Anweisungen, die innerhalb der Schleife stehen. Notieren Sie alle Anweisungen innerhalb geschweifter Klammern **`{`** und **`}`**, so wie im Beispiel (siehe auch den Abschnitt über Anweisungsblöcke).*

Das obige Beispiel stellt eine einfache Passwortabfrage dar. Der Anwender hat drei Versuche, das Passwort richtig einzugeben. Dazu wird eine Schleife eingesetzt.

Beachten Sie:

Achten Sie bei solchen Schleifen immer darauf, dass es mindestens eine Möglichkeit gibt, um die Schleife nach einer angemessenen Zeit zu beenden. Andernfalls erzeugen Sie eine so genannte "Endlosschleife", aus der der Anwender nur durch gewaltsames Beenden des WWW-Browsers herauskommt. Das ist besonders bei Online-Sitzungen im WWW sehr ärgerlich!

Um Endlosschleifen zu vermeiden, brauchen Sie irgendetwas, das irgendwann zu einem Ausweg aus der Schleife führt. Meistens werden zu diesem Zweck so genannte "Zähler" definiert, im Beispiel die Variable **`Zaehler`**. Diese Variable hat im Beispiel einen Anfangswert von 0 und wird innerhalb der Schleife bei jedem Durchgang mit der Anweisung **`Zaehler++;`** um 1 erhöht. Wenn im Beispiel der Zählerstand grösser gleich 3 ist, wird abgebrochen.

Weitere Möglichkeiten, um Schleife abzubrechen, werden weiter unten beschrieben.

Aufgabe 15:

- Vervollständigen Sie das oben stehende Programm damit es auch funktionstüchtig ist.

2.9.2 Schleifen mit "for"

Mit Hilfe von for-Schleifen vermeiden Sie die Probleme von **`while`**-Schleifen. Die Schleifenbedingung einer **`for`**-Schleife sieht von vorne herein einen Zähler und eine Abbruchbedingung vor.

Beispiel:

```
<script language="JavaScript">
  for(i = 1; i <= 100; i++)
  {
    var x = i * i;
    document.write("<br>Das Quadrat von " + i + " ist " + x);
  }
</script>
```

Erläuterung:

*Eine for-Schleife beginnt mit dem Wort **`for`**. Dahinter wird, in Klammern stehend, die Schleifenbedingung formuliert. Bei der **`for`**-Schleife gilt dabei eine feste Syntax. Innerhalb der Schleifenbedingung werden drei Anweisungen notiert. In der ersten Anweisung wird ein Schleifenzähler definiert und initialisiert. Im Beispiel wird ein Zähler **`i`** definiert und mit dem Wert **`1`** initialisiert. Die zweite Anweisung enthält die Bedingung, ab der die Schleife beendet wird. Dazu brauchen Sie Vergleichsoperatoren. In der dritten Anweisung wird der Schleifenzähler so verändert, dass er irgendwann die in der zweiten Anweisung notierte Bedingung erfüllt. Im Beispiel wird **`i`** bei jedem Schleifendurchgang um **`1`** erhöht, so dass der Wert irgendwann grösser **`100`** ist und damit die Bedingung der zweiten Anweisung erfüllt.*

Das Beispiel benutzt den Schleifenzähler, um bei jedem Schleifendurchgang das Quadrat des aktuellen Werts zu ermitteln. Das Ergebnis wird dann HTML-formatiert ins aktuelle Dokumentfenster geschrieben.

Aufgabe 16:

- Vervollständigen Sie das oben stehende Programm damit es auch funktionstüchtig ist.

2.9.3 Schleifen mit "do-while"

Die do-while-Schleife ist eine Variante der normalen **`while`**-Schleife. Der Unterschied zwischen beiden ist, dass bei der normalen **`while`**-Schleife vor dem Ausführen des Codes die Schleifenbedingung überprüft wird, während bei der **`do-while`**-Schleife zuerst der Code ausgeführt und erst danach die Schleifenbedingung überprüft wird. Auf diese Weise können Sie erzwingen, dass Anweisungen innerhalb der Schleife auf jeden Fall mindestens einmal ausgeführt werden, auch wenn sich die Schleifenbedingung gleich am Anfang als unwahr herausstellt.

Beispiel:

```
Einmal so:
<script language="JavaScript">
  var x = 10;
  do
  {
    document.write("<br>x * x = " + (x * x));
    x = x + 1;
  }
  while(x < 10);
</script>
<p>
Und einmal so:
<script language="JavaScript">
  var x = 10;
  while(x < 10)
  {
    document.write("<br>x * x = " + (x * x));
    x = x + 1;
  }
</script>
```

Erläuterung:

*Im Beispiel werden zwei kleine JavaScript-Bereiche definiert. In beiden Bereichen wird eine Variable **`x`** definiert und mit dem Wert **`10`** vorbelegt. Im ersten Bereich wird solange das Quadrat von **`x`** (das bei jedem Schleifendurchlauf um **`1`** erhöht wird) geschrieben, wie **`x`** kleiner als **`10`** ist. Da **`x`** ja schon am Beginn den Wert **`10`** hat, ist die Abbruchbedingung eigentlich schon von vorne herein erfüllt. Trotzdem wird ein mal das Quadrat von **`x`** ausgegeben, da die Überprüfung der Schleifenbedingung erst nach dem Ausführen der Anweisungen innerhalb der Schleife erfolgt.*

*Im zweiten Script-Bereich herrschen die gleichen Bedingungen, jedoch wird dort eine normale **`while`**-Schleife notiert. Da **`x`** von vorne herein nicht kleiner als **`10`** ist, werden die Anweisungen der while-Schleife kein einziges mal ausgeführt. Die Überprüfung der Schleifenbedingung, die am Anfang stattfindet, verhindert dies.*

Aufgabe 17:

- Vervollständigen Sie das oben stehende Programm damit es auch funktionstüchtig ist und erklären Sie das Resultat

2.9.4 Kontrollen innerhalb von Schleifen mit break und continue

Schleifen sind "kritische Faktoren" innerhalb eines Scripts. Bei komplizierteren Aufgaben ist es manchmal nicht einfach, eine Schleife so zu programmieren, dass die Schleife in jedem Fall irgendwann mal abgebrochen wird. Deshalb gibt es zusätzliche Befehle, um innerhalb einer Schleife das Geschehen zu kontrollieren.

Beispiel 1:

```
<script language="JavaScript">
  var i = 0;
  while (i < 6)
  {
    if (i == 3) break;
    i++;
  }
  alert("i = " + i);
</script>
```

Erläuterung:

*Mit **break** können Sie eine Schleife sofort beenden. Dazu müssen Sie innerhalb des Schleifenkörpers eine **if**-Abfrage notieren und abhängig davon das Wort **break** als Anweisung notieren. Im Beispiel bricht die Schleife bereits ab, wenn **i** den Wert **3** hat, obwohl laut Schleifenbedingung das Hochzählen bis **6** erlaubt.*

Beispiel 2:

```
<script language="JavaScript">
<!--
  var i = 0, j = 0;
  while (i < 6)
  {
    i++;
    if (i == 3) continue;
    j++;
  }
  alert("i ist gleich " + i + " und j ist gleich " + j);
</script>
```

Erläuterung:

*Mit **continue** erzwingen Sie sofort den nächsten Schleifendurchlauf. Nachfolgende Anweisungen innerhalb der Schleife werden bei diesem Schleifendurchlauf nicht mehr ausgeführt. Im Beispiel werden zwei Zähler **i** und **j** bei jedem Schleifendurchlauf um **1** erhöht. Wenn **i** gleich **6** ist, wird die Schleife abgebrochen. Zwischendurch hat **i** auch mal den Wert **3**. Dieser Fall wird mit einer **if**-Abfrage behandelt. Wenn **i** gleich **3** ist, wird sofort der nächste Schleifendurchgang gestartet. Die Anweisung `j++;` wird dadurch in diesem Schleifendurchlauf nicht mehr ausgeführt. Am Ende hat dadurch **i** den Wert **6** und `j` nur den Wert **5**.*

Aufgabe 18:

- Vervollständigen Sie das oben stehende Programme damit sie auch funktionstüchtig ist und erklären Sie die Resultate

3 Einige HTML-Erweiterungen

3.1 Einzeilige Eingabefelder

Einzeilige Eingabefelder dienen zur Aufnahme von einem oder wenigen Wörtern oder einer Zahl.

Beispiel 1:

```
<html><head><title>Formularfelder</title></head><body>
Gib deinen Spitznamen ein: <input type=text  name="Spitzname" size=20 maxlength=20>
</body></html>
```

Erläuterung:

<input ... > *definiert ein einzeiliges Eingabefeld (input = Eingabe). Der Vollständigkeit halber können Sie die Angabe* ***type=text*** *dazu setzen, d.h. alles was Sie in das Feld eingeben wird als Text interpretiert.*

Jedes Eingabefeld muss einen internen Bezeichnernamen erhalten, und zwar mit der Angabe ***name=****. Der Name sollte nicht zu lang sein und darf keine Leerzeichen und keine deutschen Umlaute enthalten. Verwenden Sie als Sonderzeichen höchstens den Unterstrich "_". Setzen Sie den Namen in Anführungszeichen.*

Ferner sollten Sie bei einzeiligen Eingabefeldern immer die Anzeigelänge in Zeichen mit ***size=*** *sowie die interne Feldlänge in Zeichen* ***maxlength=*** *bestimmen. Beide Angaben bedeuten die Anzahl Zeichen (size = Grösse, maxlength = maximal length = maximale Länge). Wenn die interne Feldlänge* ***maxlength*** *grösser ist als die angezeigte Feldlänge* ***size****, dann wird bei längeren Eingaben automatisch gescrollt.*

Beispiel 2:

```
<table><tr>
<td align=right>Ihr Vorname:</td>
<td><input type=text size=40 maxlength=40></td>
</tr><tr>
<td align=right>Ihr Nachname:</td>
<td><input type=text size=40 maxlength=40></td>
</tr>
</table>
```

Erläuterung:

Das Beispiel zeigt, wie Sie mit Hilfe einer Tabelle Beschriftung und Eingabefelder eines Formulars ordentlich formatieren können.

Aufgabe 19:

- Vervollständigen Sie das oben stehende Programm damit es auch funktionstüchtig ist.
- Schlagen Sie in SELFHTML die Bedeutung der Tags **<table>**, **<td>** und **<tr>** nach.
- Bauen Sie eine eigene Tabelle mit verschiedenen Eingabefeldern auf.

3.1.1 Textvorbelegung bei einzeiligen Eingabefeldern

Sie können ein einzeiliges Eingabefeld mit einem Inhalt vor belegen.

Beispiel:

```
<html><head><title>Formularfelder</title></head><body>
Gib deinen Spitznamen ein:
<input  name="Spitzname" size=20 maxlength=20 value= "Held">
</body></html>
```

Erläuterung:

Eingabefelder mit vorbelegtem Inhalt werden wie gewöhnliche Eingabefelder definiert. Mit dem zusätzlichen Attribut ***value=*** *können Sie einen Text angeben, mit dem das Feld vorbelegt wird (value = Wert). Der Text muss in Anführungszeichen stehen.*

3.1.2 Verarbeitung der Eingabe eines Eingabefeldes

Der Inhalt eines Eingabefeldes kann verarbeitet werden.

Beispiel 1:

```
<html><head><title>Formularfelder</title></head><body>
Gib deinen Spitznamen ein:
<input type=text  name="Spitzname" size=20 maxlength=20 value="Held"
onBlur="alert('Hallo '+ value)">
</body></html>
```

Erläuterung:

onBlur= *ist ein Ereignis (siehe Ereignisbehandlung weiter hinten), mit dem Sie den Wert eines Eingabefeldes beim Verlassen des Eingabefeldes als Variablenwert (****value****) ins Programm übernehmen können. In unserem Beispiel wird der Wert von* ***value*** *mit einer* ***alert****-Anweisung auf den Bildschirm ausgegeben.*

In Beispiel 1 wird die Aktion auf das Ereignis (**onBlur**) direkt ausgeführt. Die Ausführung kann auch in einer Funktion ausgeführt werden.

Beispiel 2:

```
<html><head><title>Formularfelder</title></head><body>
<script language="JavaScript">
  function Ausfuehrung(Wert)
  {
    alert("Hallo " + Wert);
  }
</script>

Gib deinen Spitznamen ein:
<input type=text  name="Spitzname" size=20 maxlength=20 value="Held"
onBlur="Ausfuehrung(value)">
</body></html>
```

Erläuterung:

Die Aktion des Ereignisses ***onBlur*** *wird in einer eigenen Funktion* ***Ausfuehrung(Wert)*** *ausgeführt. Die Daten (****value****) des Eingabefeldes können also im Programm an beliebigen Stellen weiter verwendet werden. (Siehe auch das Kapitel Formulare weiter unten)*

Aufgabe 20:

- Ändern Sie in Beispiel 2 die Vorbelegung des Eingabefeldes.

3.1.3 Eingabefeld für Passwörter

Für die Eingabe von Geheimnummern, Passwörtern usw. gibt es einen speziellen Typ von Eingabefeld. Die eingegebenen Zeichen werden dabei durch Platzhalter (meistens Sternchen) dargestellt, so dass Personen im Raum des Anwenders nicht zufällig das eingegebene Passwort mit lesen können.

Beispiel:

```
Passwort eingeben: <input type=password name="Zugangsnummer"
maxlength=10 size=10>
```

Erläuterung:

Eingabefelder für Passwörter werden wie gewöhnliche Eingabefelder definiert. Mit der zusätzlichen Angabe ***type=password*** *bestimmen Sie, dass es sich um ein Passwort-Feld handelt.*

Beachten Sie:

Passwörter werden trotz der verdeckten Eingabe im Klartext über das Internet übertragen. Weisen Sie Anwender in ernsthaften Zusammenhängen auf diese Tatsache bitte explizit hin.

Aufgabe 21:

- Vervollständigen Sie das oben stehende Programm so, dass nach dem Eingeben des Passwortes mit dem Ereignis **onBlur** eine Funktion aufgerufen wird, die das Passwort anzeigt.

3.2 Schaltflächen (Buttons)

Sie können anklickbare Schaltflächen definieren. Sinnvoll sind solche frei definierbaren Buttons nur in Verbindung mit JavaScript. In der Regel werden sie dazu verwendet, Verweise oder andere JavaScript-gesteuerte Befehle auszuführen.

Aufbau:

```
<input type=button value="Hallo" onClick="Ausgabe()">
```

Erläuterung:

*Mit **<input type=button ...>** definieren Sie einen Button (input = Eingabe). Die Beschriftung des Buttons bestimmen sie mit der Zusatzangabe **value=** (value = Wert). Die Angabe muss in Anführungszeichen stehen. Um anzugeben, was passieren soll, wenn der Button angeklickt wird, können Sie beispielsweise den JavaScript-EventHandler (siehe weiter unten) **onClick=** verwenden. Hinter dem Ist-Gleich-Zeichen geben Sie einen JavaScript-Befehl ein, z.B. den Aufruf einer selbst geschriebenen JavaScript-Funktion, z.B. **Ausgabe()**, oder einen einfachen JavaScript-Befehl, z.B. a **lert()**. Ein funktionierendes JavaScript-Programm mit einem Button sehen Sie in Beispiel 2.*

Beispiel 1:

```
<html><head><title>Button</title></head><body>
<input type="button" value="Hallo" onClick="alert('Hallo Du')">
</body></html>
```

Erläuterung:

*Sobald die Schaltfläche gedrückt wird, wird die Funktion **alert()** nach dem Befehl **onClick** ausgeführt.*

Aufgabe 22:

Schon wieder wurde mit **type=** etwas definiert, hier eine Schaltfläche.

- Schauen Sie unter dem Tag **<input>** in SELFHTML nach, was Sie alles mit **type=** definieren können.

Beispiel 2:

```
<html><head><title>Button</title></head><body>
<script language="JavaScript">
  function Ausgabe(Text)
  {
    alert(Text);
  }
</script>
<input type="button" value="Hallo" onClick="Ausgabe('Hallo Du')">
</body></html>
```

Erläuterung:

*In diesem Beispiel wird das Drücken der Schaltfläche nicht direkt in einer Funktion nach **onClick=** ausgeführt, sondern in einer eigenen JavaScript-Funktion **Ausgabe()**.*

Aufgabe 23:

- Warum steht die Textkonstante **'Hallo Du'** in Hochkomma und nicht in Anführungs- und Schlusszeichen **"Hallo Du"**?

Beispiel 3:

```
<table><tr>
<td width=30><input type=button value=" A " onClick="alert('Schalter
A')"></td>
<td width=40><input type=button value=" B " onClick="alert('Schalter
B')"></td>
<td width=50><input type=button value=" C " onClick="alert('Schalter
C')"></td>
<td width=60><input type=button value=" D " onClick="alert('Schalter
D')"></td>
</tr></table>
```

Erläuterung:

In diesem Beispiel werden verschiedenen Schaltflächen dargestellt. Der Tag ***<td width= ..>*** *bestimmt den Abstand zwischen den Schaltflächen. Die anderen HTML-Begriffe kennen Sie bereits.*

Aufgabe 24:

- Schreiben Sie ein Programm, indem die **alert**-Befehle des Beispiels 3 durch entsprechende Funktionsaufrufe (mit den entsprechenden Funktionen) ersetzt werden.

3.3 Formulare

Damit Eingabefelder und Schaltflächen miteinander verknüpft werden können, werden sogenannte Formulare eingesetzt. In Formularen kann der Anwender Eingabefelder ausfüllen, in mehrzeiligen Textfeldern Text eingeben, aus Listen Einträge auswählen und Buttons anklicken. Wenn das Formular fertig ausgefüllt ist, kann der Anwender auf einen Button klicken, um das Formular zu verarbeiten, z.B. als e-mail versenden etc.

Aufbau:

```
<form name="Formular" ...>
... Elemente des Formulars wie Eingabefelder, Auswahllisten, Buttons
usw. ...
</form>
```

Erläuterung:

Mit ***<form ...>*** *definieren Sie ein Formular (form = Formular). Alles, was zwischen diesem einleitenden Tag und dem abschliessenden Tag* ***</form>*** *steht, gehört zum Formular. Das sind hauptsächlich Elemente des Formulars wie Eingabefelder, Auswahllisten oder Buttons. Um die Elemente zu plazieren, brauchen Sie jedoch auch Textabsätze (Absatzschaltungen) und erzwungene Zeilenumbrüche. Darüber hinaus können Sie zwischen* ***<form..>*** *und* ***</form>*** *auch Text eingeben und diesen Text wie üblich mit HTML-Befehlen formatieren. Auch Grafiken, Verweise, Tabellen, Multimedia-Elemente sind mitten im Formular erlaubt. So können Sie Ihr Formular optisch aufwerten und mit erklärendem Text usw. versehen.*

Sie können dem Formular mit ***name=*** *einen Namen geben. Der Name sollte in Anführungszeichen stehen und kann dazu verwendet werden, um die einzelnen Elemente der Formulare in Funktionen zu verarbeiten.*

Beispiel 1:

```
<html><head><title>Formularverarbeitung</title></head><body>
<script language="JavaScript">
  function Verarbeitung()
  {
    alert("Hallo " + document.Formular.Spitzname.value);
  }
</script>

<form name="Formular">
Gib deinen Spitznamen ein: <input  name="Spitzname" size=20
maxlength=20 value= "Held" >
<input type="button" value="Hallo" onClick="Verarbeitung()">
</form>
</body></html>
```

Erläuterung:

Indem man dem Formular einen Namen ***"Formular"*** *gibt, kann man über diesen Namen auf die einzelnen Elemente des Formulars, über die Objektreferenz* ***document*** *zugreifen, z.B.* ***document.Formular.Spitzname.value****. Dies spricht das Feld* ***value*** *des Eingabefeldes an, also die Eingabe, die Sie gemacht haben. Über das Objekt* ***document*** *kann also auf die einzelnen Felder des Formulars (über ihre Namen) zugegriffen werden.*

Beachten Sie:

Formulare können noch weit mehr bewirken, z.B. Aktionen wie Versenden eines vollständig ausgefüllten Formulars mit verschiedenen Feldern an eine bestimmte e-mail-Adresse oder an ein Server-Programm zur Weiterverarbeitung (siehe SELFHTML).

Aufgabe 25:

- Schauen Sie in SELFHTML nach, wie die Objektreferenz **`document`** aufgebaut ist (Unter Kapitel : Allgemeines zur Verwendung).
- Schreiben Sie ein Programm, dass Sie nach einem Passwort fragt und das Passwort mit einem vorher bestimmten Wert (Variable) vergleicht. Öffnen Sie in einer Funktion, wie oben beschrieben, mit **`alert()`** ein Fenster mit der Antwort **`richtiges Passwort`**, resp. **`falsches Passwort`**, nachdem Sie das eingetippte Passwort mit dem vorbestimmten Passwort verglichen haben.

3.4 JavaScript-Code verstecken:

Es macht natürlich keinen Sinn, dass ein Passwort in einer Variablen abgespeichert ist. Jeder, der auf den Quellcode des HTML-Dokumentes zugreifen kann, kann das Passwort lesen. Um das Passwort wenigstens ein wenig zu verstecken, kann man den JavaScript-Code in eine eigene Datei auslagern, die nicht mehr von jedermann einsehbar ist.

Betrachten wir einen bekannten HTML-Quellcode.

Beispiel 1: HTML-Quellcode:

```
<html><head><title>Formularverarbeitung</title></head><body>
<script language="JavaScript">
  function Verarbeitung()
  {
    alert("Hallo " + document.Formular.Spitzname.value);
  }
</script>

<form name="Formular">
Gib deinen Spitznamen ein: <input  name="Spitzname" size=20
maxlength=20 value= "Held" >
<input type="button" value="Hallo" onClick="Verarbeitung()">
</form>
</body></html>
```

Durch Aufteilung in einen HTML- und eine JavaScript-Teil, kann das JavaScript-Programm für den Betrachter eines HTML-Quellencodes unsichtbar gemacht werden.

Beispiel 2: Aufteilung in JavaScript-Programm (abgespeichert unter `verarbeitung.js`):

```
function Verarbeitung()
{
  alert("Hallo " + document.Formular.Spitzname.value);
}
```

und HTML-Programm

```
<html><head><title>Formularverarbeitung</title></head><body>
<script language="JavaScript" src="verarbeitung.js">
</script>

<form name="Formular">
Gib deinen Spitznamen ein: <input  name="Spitzname" size=20
maxlength=20 value= "Held" >
<input type="button" value="Hallo" onClick="Verarbeitung()">
</form>
</body></html>
```

Erläuterung:

Das JavaScript-Programm wird unter ***verarbeitung.js*** *im gleichen Ordner, wie der HTML-Quellcode abgespeichert. Das JavaScript-Programm wird nicht in den HTML-Code geschrieben, sondern das JavaScript-Programm wird mit dem Verweis* ***src="verarbeitung.js"*** *in das HTML-Programm eingebunden.*

Das JavaScript-Programm wird für den Betrachter des HTML-Quellcodes unsichtbar.

Aufgabe 26:

- Machen Sie das Passwort in Aufgabe 25 und die Verarbeitung des Passworts für den Betrachter des HTML-Codes unsichtbar.
- Erweitern Sie die Passwortabfrage aus Aufgabe 25 dahingehend, dass maximal 3 Mal ein Passwort eingegeben werden kann, bevor das Programm verlassen wird. Falls das Passwort richtig eingegeben wird, wird das Programm mit einer entsprechenden Meldung sofort verlassen; nach dreimaliger Falscheingabe wird eine entsprechende Meldung auf den Bildschirm ausgegeben. Das Passwort muss für den Anwender unsichtbar sein!
  - Tipp: Arbeiten Sie mit einer **`while`**-Schleife und einem Zähler (Variable) als Abbruchbedingung.

4 Ereignisbehandlung (Event-Handler)

4.1 Allgemeines zu Ereignissen

JavaScript-Ereignisse können Sie sich wie Ereignisse (events) im wirklichen Leben vorstellen. Es sind eine Art Unterbrechungen des normalen Ablaufs, die jederzeit und beliebig oft eintreten können. Bereits kennengelernt haben Sie das JavaScript-Ereignis **`onBlur`** und **`onClick`**. Beim Verlassen eines Feldes und beim Drücken der Maustaste werden diese Ereignisse ausgelöst.

Jedes Ereignis hat einen Namen, dem Sie eine JavaScript-Funktion zuweisen können. Diese Funktion wird ausgeführt, wenn das Ereignis eintritt. Bei der Behandlung von Ereignissen gibt es einige Punkte, die zu beachten sind:

- Ereignisse werden innerhalb eines HTML-Tags aufgerufen, z.B. innerhalb des Tags **`<input …>`**:
  - **`<input type="button" value="Drücken" onClick="alert('Meldung')">`**
- Setzen Sie die JavaScript-Funktion mit allen Parametern in Anführungszeichen **`"alert('Meldung')"`**, sonst wird die Funktion im Netscape Navigator nicht ausgeführt. Texte als Parameter müssen mit Hochkommas (**`'Meldung'`**) übergeben werden, weil zwei Gänsefüsschen (**`""`**) hintereinander als Textbegrenzung betrachtet werden würden.
- Definieren Sie alle Funktionen, die Sie mit einem Ereignis aufrufen wollen in den Kopfbereich (vor dem Ereignis) der Seite, damit Sie schon geladen sind, wenn das Ereignis eintritt.
- Programmieren Sie das Beispiel und führen Sie es aus

Beispiel:

```
<html><head><title>EventHandler</title></head><body>
<input type="button" value="Drücken" onClick="alert('Ich habe mit
type=="button" und value="Drücken" ein Schaltfläche mit dem Namen
Drücken aufgebaut\n\n Wenn ich mit der Maus auf diese Schaltfläche
klicke, so wird das Ereignis: onClick aktiviert und der Befehl nach
onClick ausgeführt\n\n Das Ergebnis sehen Sie hier')">
</body></html>
```

Die Ereignisbehandlung ist ein zentraler Punkt bei JavaScript und stellt eine Art Schnittstelle zwischen HTML und JavaScript dar. Über die Ereignisbehandlung werden in der Regel die einzelnen JavaScript-Funktionen aufgerufen. Sie werden innerhalb des HTML-Codes verwendet und können auch nur innerhalb des gültigen HTML-Tags aufgerufen werden.

4.1.1 onBlur (beim Verlassen)

Für den Fall, dass ein Element zuvor aktiviert war und der Anwender es jetzt verlässt. (Siehe Eingabefelder)

Beispiel:

```
<html><head><title>onBlur</title></head><body>
<script language="JavaScript">
  function CheckInhalt(Feld)
  {
    if(Feld == "")
    {
      alert("Namensfeld muss einen Inhalt haben!");
      document.Test.Eingabe.focus();
      return false;
    }
    else alert("Ok! Namensfeld hat einen Inhalt!");
  }
</script>
<form name="Test">
  Name eingeben : <input type=text name="Eingabe" value=""
  onBlur="CheckInhalt(value)">
</form>
<script language="JavaScript">
  document.Test.Eingabe.focus();
</script></body></html>
```

Erläuterung:

*Im Beispiel wird ein Formular definiert, das ein Eingabefeld enthält. Unterhalb des Formulars ist ein JavaScript-Bereich notiert. Der Bereich wird deshalb unterhalb des Formulars definiert, weil zu Beginn des Bereichs gleich eine Anweisung ausgeführt wird, die die Existenz des Formulars bereits voraussetzt. Diese Anweisung (**`document.Test.Eingabe.focus();`**) setzt den Cursor in das Eingabefeld. Dort soll der Anwender seinen Namen eingeben. Klickt er irgendwo anders hin, wird der EventHandler **`onBlur`** aktiv, der im HTML-Tag des Eingabefeldes notiert ist. Dabei wird die Funktion **`CheckInhalt()`** aufgerufen, die ebenfalls in dem JavaScript-Bereich notiert ist. Diese Funktion fragt ab, ob die ihr übergebene Zeichenkette, der Inhalt des Namensfeldes, leer ist. Wenn ja, wird ein Meldungsfenster ausgegeben, und der Cursor wird wieder in das Feld positioniert.*

4.1.2 onClick (beim Anklicken)

Wie Sie gesehen haben, wird **onClick=** eingesetzt um einen Mausklick auf ein Element zu bearbeiten.

Beispiel:

```
<html><head><title>onCLick</title></head><body>
<script language="JavaScript">
  function Verarbeitung()
  {
    alert("Sobald Sie auf die Schaltfläche drücken, wird dieses
Fenster eingeblendet");
  }
</script>
<input type="button" value="Drücken" onClick="Verarbeitung()">
</body></html>
```

Erläuterung:

*Im Beispiel wird eine Schaltfläche **Drücken** aufgebaut. Beim Drücken der Schaltfläche wird die Funktion **Verarbeitung()** aufgerufen und eine **alert**-Anweisung ausgeführt.*

4.1.3 onDblClick (beim Doppelklicken)

Für den Fall, dass ein Anwender einen Doppelklick auf ein Element ausführt.

Beispiel:

```
<html><head><title>onDblClick</title>
</head><body>
<form name="Rechnen">
  Wert: <input size=10 name="Wert">
  <input type=button value="Doppelklick = Wert Potenzieren"
   onDblClick = "document.Rechnen.Wert.value = document.Rechnen.Wert.value *
   document.Rechnen.Wert.value">
</form>
<script language="JavaScript">
  document.Rechnen.Wert.focus();
</script></body></html>
```

Erläuterung:

*Im Beispiel wird der Fokus auf ein Eingabefeld gesetzt (**document.Rechnen.Wert.focus();**). Nach der Eingabe eines Wertes kann mit einem Doppelklick auf die Schaltfläche (**Doppelklick = Wert Potenzieren**) die Potenz des eingegebenen Wertes berechnet (**document.Rechnen.Wert.value * document.Rechnen.Wert.value**) und im Eingabefeld angezeigt werden (**document.Rechnen.Wert.value**).*

Aufgabe 27:

- Schreiben Sie ein das Beispiel so um, dass auf einen einfachen Klick (**`onClick`**) das Quadrat eines eingegebenen Wertes berechnet wird, und auf Doppelklick (**`onDblClick`**) die 3. Potenz des Wertes.

4.1.4 onFocus (beim Aktivieren)

Tritt ein, wenn der Anwender ein Element aktiviert.

Beispiel:

```
<html><head><title>OnFocus</title></head><body>
<form>
 <input size=30 onFocus="value='Hier Ihren Namen eingeben'"><br>
 <input size=30 onFocus="value='Hier Ihren Wohnort eingeben'"><br>
 <input size=30 onFocus="value='Hier Ihr Alter eingeben'"><br>
</form>
</body></html>
```

Erläuterung:

In dem Beispiel wird ein Formular definiert, das drei Eingabefelder enthält. Da die Felder unbeschriftet sind, hat der Anwender keine Ahnung, was er in die einzelnen Felder eingeben kann. Bewegt er den Cursor aus Neugier doch in eines der Eingabefelder, wird der Event-Handler ***`onFocus=`*** *des jeweiligen Feldes aktiv. Dabei wird in das jeweilige Feld eine Aufforderung geschrieben, was in dem Feld einzugeben ist.*

4.1.5 onLoad (beim Laden einer Datei)

Tritt ein, wenn eine HTML-Datei geladen wird.

Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript">
 function NaviFenster()
 {
   Navigation = window.open("TestHTML.html", "Navigation","height=100,
   width=300");
   Navigation.focus();
 }
</script>
</head>
<body onLoad="NaviFenster()">
</body></html>
```

Zusätzliche HTML-Seite

```
<html><head><title>TestHTML-Seite</title></head>
<body>
<h1>Dies ist eine Testseite</h1>
</body></html>
```

Erläuterung:

*Im Beispiel wird beim Starten des Programms mit **onLoad** ein zweites Fenster mit der HTML-Seite **TestHTML.html** geöffnet. Diese Seite muss im gleichen Ordner existieren! **OnLoad** ruft die Funktion **NaviFenster()** auf. Innerhalb dieser Funktion steht der Befehl zum Öffnen des Zweitfensters (**window.open()**). Das Fenster erhält auch gleich den Fokus (wird zum aktiven Fenster), so dass es im Vordergrund des Hauptfensters zu sehen ist.*

Aufgabe 28:

- Führen Sie das Beispiel aus und versuchen Sie jede Zeile des HTML-Codes zu verstehen.
- Schauen Sie in SELFHTML nach was über die Objektreferenz **window** geschrieben steht.
- Schreiben Sie alle Methoden auf, die Sie für die Objektreferenz **window** finden.

4.1.6 OnMouseover (beim Überfahren des Elements mit der Maus)

4.1.7 OnMouseout (beim Verlassen des Elements mit der Maus)

Tritt ein, wenn die Maus über ein Element gefahren wird, resp. wenn ein aktives Element von der Maus verlassen wird.

Beispiel:

```
<html><head><title>onMouseover</title>
</head><body>
 <h1 id="Test"
 onMouseover="document.all.Test.innerText='Sehen Sie?'"
 onMouseout="document.all.Test.innerText='Ich bin dynamisch'">Ich bin
dynamisch</h1>
</body></html>
```

Erläuterung:

*In dem Beispiel wird eine Überschrift erster Ordnung definiert. Innerhalb der Überschrift sind die Event-Handler **onMouseover=** und **onMouseout=** notiert. Der Event-Handler **onMouseover=** tritt in Aktion, wenn der Anwender die Maus in den Anzeigebereich der Überschrift bewegt, und **onMouseout=** wird aktiv, wenn er die Maus wieder aus dem Anzeigebereich heraus bewegt. Mit Hilfe des **all**-Objekts (siehe SELFHTML) und der Eigenschaft **innerText** wird bei jedem Aktiv werden eines der beiden Event-Handler der Text der Überschrift dynamisch ausgetauscht. Bei **onMouseover=** wird ein anderer Text angezeigt, bei **onMouseout=** wieder der ursprüngliche Text.*

Aufgabe 29:

- Führen Sie das Beispiel aus und versuchen Sie jede Zeile des HTML-Codes zu verstehen.

4.1.8 OnReset (beim Verlassen des Elements mit der Maus)

Tritt ein, wenn der Anwender Eingaben in einem Formular verwerfen will.

Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript">
 function ResetCheck()
 {
   chk = window.confirm("Wollen Sie alle Eingaben loeschen?");
   return chk;
 }
</script>
</head><body>
<form name="Test" onReset="return ResetCheck()">
  Name: <input size=30><br>
  Idee: <input size=30><br>
  <input type=reset>
</form>
</body></html>
```

Erläuterung:

*Das Beispiel enthält ein Formular, das einen Abbrechen-Button (**`Reset`**-Schaltfläche) enthält. Beim Anklicken dieses Buttons werden normalerweise alle Eingaben im Formular gelöscht. Im Beispiel ist jedoch im einleitenden **`<form>`**-Tag der Event-Handler **`onReset=`** notiert. Dieser tritt in Aktion, wenn der Reset-Button angeklickt wird. Im Beispiel wird dann die Funktion **`ResetCheck()`** aufgerufen, die in einem Script-Bereich im Dateikopf steht. Diese Funktion fragt den Anwender in einem Bestätigungsfenster (**`window.confirm()`**), ob er wirklich alle Eingaben in dem Formular löschen will. Bestätigt er den Löschwunsch, gibt das Bestätigungsfenster den Wert **`true`** zurück. Verneint er den Löschwunsch, wird **`false`** zurückgegeben. Der Rückgabewert wird in der Variablen **`chk`** gespeichert und diese wird wiederum von der Funktion **`ResetCheck()`** an den aufrufenden Event-Handler zurückgegeben. Der Effekt ist, dass die Formulareingaben nur gelöscht werden, wenn **`true`** zurückgegeben wird.*

Aufgabe 30:

Erweitern Sie die Aufgabe 27 um eine Reset-Schaltfläche, die das Eingabefeld löscht.

4.2 Grafiken in HTML-Code

4.2.1 Anzeigen von Grafiken mit HTML

Um Grafiken in Ihre HTML-Dateien einzubinden, referenzieren Sie die Grafikdateien an der gewünschten Stelle im HTML-Quelltext. Geeignete Dateiformate für WWW-gerechte Grafiken sind GIF und JPG-Formate

Beispiel:

```
<img src = "datei.gif">
<img src = "datei.jpg">
```

Erläuterung:

Die Grafikreferenz beginnt mit ***<img src=*** *(img = image = Bild, src = source = Quelle). Hinter dem Ist-Gleich-Zeichen geben Sie den Namen der Grafikdatei an, auf die verwiesen wird (in den Beispielen:* ***"datei.gif"*** *bzw.* ***"datei.jpg"****). Der Dateiname muss in Anführungszeichen stehen.*

Beachten Sie:

Die obigen Beispiele setzen voraus, dass sich die Grafikdatei im gleichen Verzeichnis befindet wie die HTML-Datei, in der die Grafik referenziert wird. Sie können auch Grafiken in anderen Verzeichnissen**:**

```
<img src = "verzeichnis/unterverz/datei.gif">
```

und sogar beliebige Grafiken auf anderen WWW-Seiten referenzieren**:**

```
<img src  = "http://www.netzwelt.com/software/xdance1.gif">
```

Um eine Grafik alleine - ohne Text daneben - anzuzeigen, notieren Sie am besten vor und nach der Grafikreferenz eine Absatzschaltung für Textabsätze. Also zum Beispiel:

```
<p>
  <img src = "datei.gif">
</p>
```

Aufgabe 31:

Suchen Sie eine Grafik im Internet, laden Sie diese Grafik auf ihr Home-Verzeichnis und binden Sie diese Grafik in eine HTML-Seite ein.

4.3 Grösse von Grafiken

Wenn Sie HTML-Dateien für's WWW erstellen, sollten Sie darauf achten, dass die darin referenzierten Grafiken nicht zu gross sind, denn aufwendige Grafiken verursachen lange Ladezeiten. Reduzieren Sie in Ihren Grafiken gegebenenfalls die Anzahl der Farben, verringern Sie die Bildgrösse und stopfen Sie nicht zu viele Grafik-Referenzen in eine einzige HTML-Datei. In jedem Fall sollten Sie reite und Höhe mit angeben:

Beispiel:

```
<img src = "datei.gif" width = 250 height = 450>
```

Erläuterung:

*Mit der Angabe **width=** [Pixel] geben Sie die Breite der Grafik an, mit h**eight=** [Pixel] die Höhe (width = Breite, height = Höhe).*

Beachten Sie:

Um die genaue Breite und Höhe einer Grafik zu ermitteln, brauchen Sie entweder ein Grafikprogramm, das diese Werte anzeigt, oder einen HTML-Editor, der beim Einbinden einer Grafik im Dialog auch gleich den Dateikopf der Grafik ausliest und die entsprechenden Angaben in das **<img>-**Tag automatisch einfügt.

Aufgabe 32:

Untersuchen Sie mit dem Bildbetrachter des HTML-Editors die Grafik, die Sie in Aufgabe 31 benutzt haben. Ändern Sie diese Werte und betrachten Sie was mit der Grafik geschieht.

4.4 Grafikbeschriftung oben, mittig, unten

Sie können bestimmen, dass der Text, den Sie im Anschluss an eine Grafikreferenz notieren, als Beschriftungstext für die Grafik interpretiert wird. Bei der Grafikreferenz selbst können Sie bestimmen, wie dieser Text im Verhältnis zur Grafik plaziert werden soll.

Beispiele:

```
<img src = "datei.gif" align = top> Beschriftungstext
<img src = "datei.gif" align = middle> Beschriftungstext
<img src = "datei.gif" align = bottom> Beschriftungstext
```

Erläuterung:

*Mit dem Attribut **align=** erreichen Sie, dass nachfolgender Text als Beschriftungstext der Grafik interpretiert wird, und mit den möglichen Wertzuweisungen richten Sie den Beschriftungstext neben der Grafik aus (align = Ausrichtung).*

*Mit **align=top** wird der folgende Text als Beschriftungstext oben bündig zur Grafik interpretiert (top = oben). Mit **align=middle** wird der folgende Text als Beschriftungstext mittig zur Grafik interpretiert (middle = mittig). Mit **align=bottom** wird der folgende Text als Beschriftungstext unten bündig zur Grafik interpretiert (bottom = unten).*

Aufgabe 33:

Geben Sie dreimal dieselbe Grafik (untereinander!) auf den Bildschirm aus mit je einer Bildbeschreibung. Für die erste Grafik oben bündig, die Zweite mittig und die Letzte unten bündig zur Grafik.

4.5 Verweise

Alle Verweise in HTML haben einen einheitlichen Aufbau, egal ob sie zu einem Verweisziel in der gleichen Datei, zu einer anderen Datei im eigenen Projekt, zu einer beliebigen WWW-Adresse oder zu einer beliebigen Datei eines anderen Dateityps im Internet oder lokal auf dem eigenen Rechner führen.

Schema:

```
<a href="[Verweisziel]">Verweistext</a>
```

Erläuterung:

Das Setzen eines Verweises beginnt mit ***<a href=*** *(a = anchor = Anker, href = hyper reference = Hyper(text)-Referenz). Hinter dem Ist-Gleich-Zeichen geben Sie das Verweisziel an. Das Verweisziel muss in Anführungszeichen stehen. Es folgt der Text, der dem Anwender als Verweis angeboten wird (bei den meisten WWW-Browsern andersfarbig, häufig unterstrichen). Im Beispiel ist das der Text* ***"Verweistext"****.*

Das Verweisziel kann sein:

- Eine Stelle innerhalb der gleichen HTML-Datei
- Eine andere HTML-Datei im eigenen Projekt
- Eine WWW-Adresse
- Eine FTP-, Gopher-, Telnet- oder Newsgroup-Adresse
- Eine e-Mail-Adresse
- Eine Datei im Internet, auch eine Download-Datei
- Eine lokal abgelegte Datei

4.6 Grafiken als Verweise verwenden

Wenn Sie Verweise setzen, müssen Sie immer auch einen Verweistext definieren, also den Text, der dem Anwender als anklickbar dargestellt wird. Anstelle eines Verweistextes können Sie jedoch auch eine beliebige Grafik angeben. Dann ist die gesamte Grafik anklickbar, und beim Anklicken der Grafik wird der Verweis ausgeführt.

Beispiel:

```
<a href = "NextPage.html"> <img src = "datei.gif"> </a>
```

Erläuterung:

"NextPage.html" *ist die Bezeichnung der HTML-Seite, die beim Anklicken des der Grafik* ***"datei.gif"*** *aufgerufen wird.*

Aufgabe 34:

Schreiben Sie eine einfache HTML-Seite mit dem Namen **NextPage.html** und rufen Sie diese Seite über einen Grafik-Verweis aus einer HTML-Datei mit dem Namen **Aufgabe34.html** aus auf.

4.7 Dynamische Grafiken

Auf WWW-Seiten kommen häufig grafische Buttonleisten zum Einsatz, die Verweise zu bestimmten Seiten des Projekts enthalten. Wenn der Anwender mit der Maus über solche Grafiken fährt, erkennt er am veränderten Mauscursor und an den Verweiszielanzeigen in der Statuszeile des WWW-Browsers, dass es sich um Verweise handelt. Mit Hilfe von JavaScript können Sie solche grafischen Verweise jedoch noch deutlicher kenntlich machen und der Benutzerführung gleichzeitig ein wenig mehr Pep verleihen.

Dazu brauchen Sie je zwei gleichartige, farblich unterschiedliche Grafiken für je einen Grafikbutton,

z.B:

Mit Hilfe von JavaScript lässt sich nun eine Grafik durch eine andere ersetzen, zum Beispiel, wenn die Grafik als Verweises dient und der Anwender mit der Maus über die Grafik fährt.

Beispiel:

```
<html><head><title>Test Hoverbutton</title>
<script Language = "Javascript">
  function BilderLaden()
  {
    Image1      = new Image();
    Image1.src  = 'plan.gif';
    Image1A     = new Image();
    Image1A.src = 'plan1.gif';
  }

  function Bildtauschen(BildID, BildObjekt)
  {
    window.document.images[BildID].src = BildObjekt.src;
  }
</script></head>
<body onLoad = "BilderLaden()" >
<a href = "NextPage.html"
   onMouseout  = "Bildtauschen(0, Image1)"
   onMouseover = "Bildtauschen(0, Image1A)">
   <img src = "plan.gif" >
</a>
</body></html>
```

Erläuterung:

Für jede Grafik, die Sie dynamisch anzeigen möchten, müssen Sie eine Instanz des Image-Objekts erzeugen - hier durch das Ereignis ***OnLoad*** *ausgelöst in der Funktion* ***BilderLaden()****. Das gilt sowohl für die Grafiken, die zunächst in der HTML-Datei referenziert werden, als auch für diejenigen, die beim Überfahren mit der Maus dynamisch angezeigt werden sollen.*

Dazu wird im Beispiel mit Anweisungen wie ***Image1 = new Image();*** *eine Objektinstanz erzeugt. Nachdem die Objektinstanz erzeugt ist, sind unter dem gewählten Objektnamen, im ersten Fall* ***Image1****, alle Eigenschaften des Objekts ansprechbar. Zunächst enthält das Objekt noch gar keine Daten. Mit der Anweisung* ***Image1.src = "plan.gif";*** *wird dem neuen Grafikobjekt eine Grafikdatei zugewiesen.*

Wiederholen Sie die beiden beschriebenen Anweisungen für jede Grafikdatei, die von dynamischen Änderungen betroffen sein wird - und zwar sowohl für Grafiken, die im "Normalfall" angezeigt werden, als auch für Grafiken, die beim Darüberfahren mit der Maus dynamisch angezeigt werden. Vergeben Sie dabei jedesmal einen neuen Objektnamen.

Funktion `Bildtauschen1()`zum dynamischen Austauschen von Grafiken

Der bisherige Code im Beispiel wird beim Einlesen der HTML-Datei direkt ausgeführt, da er nicht in einer Funktion gebunden ist. Die Ausführung dieses Codes bewirkt aber keine sichtbaren Ausgaben und wird vom Anwender gar nicht bemerkt. Das, was am Bildschirm mit Hilfe von JavaScript passieren soll, nämlich das dynamische Austauschen eines Bildes, geschieht in der definierten Funktion **`Bildwechsel()`**.

Die Funktion soll aufgerufen werden, wenn der Anwender mit der Maus über einen grafischen Verweis fährt, und wenn er den Verweisbereich mit der Maus wieder verlässt. Dazu benötigt die Funktion zwei Parameter: die wievielte Grafik in der Datei ausgetauscht werden soll (Parameter **`BildID`**), und durch welches zuvor definierte Grafikobjekt das Bild ersetzt werden soll (Parameter **`BildObjekt`**). Die Funktion kommt dann mit einer einzigen Anweisung aus. Diese Anweisung ersetzt das vorhandene Bild durch das neue (**`window.document.images[BildID].src = BildObjekt.src;`**). Beachten Sie hier den wichtigen Zusammenhang: das dynamische Ersetzen einer Grafik ist nur möglich, wenn für die neue Grafik zuvor eine Instanz des Grafikobjekts erzeugt wurde. Im Beispiel geschah dies ja ganz am Anfang des Scripts.

Grafische Verweise mit Ereignissen für Mausbewegungen des Anwenders

Damit der gewünschte Effekt zustande kommt, müssen Sie in der HTML-Datei Grafiken als Verweis definieren. Im einleitenden Verweis-Tag werden die Ereignisse **`onMouseover=`** und **`onMouseout=`** notiert. Bei **`onMouseover=`** (wenn der Anwender mit der Maus über den verweis-sensitiven Bereich, hier eine Grafik fährt) wird die im Script-Bereich definierte Funktion **`Bildwechsel()`** aufgerufen, ebenso bei **`onMouseOut=`** (wenn der Mauszeiger den verweis-sensitiven Bereich wieder verlässt).

Beim Aufruf von **`Bildwechsel()`** werden jeweils die beiden benötigten Parameter übergeben. Zählen Sie dazu die referenzierten Grafikdateien in der HTML-Datei, aber fangen Sie bei **`0`** an zu zählen, d.h. **`0`** für die erste Grafik in der Datei, **`1`** für eine mögliche zweite Grafik usw. Übergeben Sie den Indexwert der Grafik, die in den Verweis eingebettet ist. Beim zweiten Parameter übergeben Sie den Objektnamen für das gewünschte Grafikobjekt. Das ist einer der Namen, die Sie am Anfang des Script-Bereichs vergeben haben. Im Beispiel wird im ersten Fall z.B. **`Image1`** bei **`onMouseover`** übergeben, und **`Image1A`** bei **`onMouseout`**.

Aufgabe 35:

Programmieren Sie diese HTML-Seite. Die Grafiken **`plan.gif`** und **`plan1.gif`** finden Sie auf dem Schüler-Pool. Vergessen Sie die HTML-Seite **`"NextPage.htm"`** nicht!