Schriftliche Hausarbeit zur Abschlussprüfung der erweiternden Studien für Lehrer im Fach Informatik

VII. Weiterbildungskurs in Zusammenarbeit mit der Fernuniversität Hagen

Eingereicht dem Amt für Lehrerausbildung - Außenstelle Gießen -

JavaScript

Einführung in eine Programmiersprache für die Klasse 11

Verfasser: Jörg Debus

0. Inhaltsverzeichnis

Seite

1. Einleitung

Seit Anfang der 90'er Jahre erobert das Internet immer mehr Bereiche unseres täglichen Lebens. Informationssuche, Homebanking, Chaten und Einkäufe tätigen sind nur einige Beispiele, wie sich Handlungen durch und auf das Internet verlagert haben.

Allein in Deutschland gibt es ca. 2.600.000 Hosts (Januar 2003) und weltweit existieren über 2 Milliarden Webseiten. Jeder, der einen Telefonanschluss, ein Modem und einen Computer besitzt, ist in der Lage, sich jede dieser Webseiten, sofern sie nicht zugangsgeschützt sind, anzusehen. Damit eröffnet sich für jeden ein schier unerschöpfliches Reservoir an Informationen und Möglichkeiten.

Mittlerweile verlagern sich sehr viele Arbeitsplätze aus den herkömmlichen Berufssparten in den Bereich Informationstechnologien, speziell in den Bereich Internet. Sehr viele Firmen drängen in dieses Medium. Sie nutzen aber ihre Webseiten nicht nur zur Selbstdarstellung, sondern auch um ihre Geschäfte dort zu tätigen und ihre Existenz zu sichern.

Der Umgang mit dem Internet birgt aber auch einige Gefahren. Hacker bemühen sich, immer neue Schwachstellen des Mediums aufzudecken und für sich zu nutzen. Nicht nur Viren und Sniffler ermöglichen es dem geschickten User, geheime Daten von fremden Rechnern einzusehen.

Betrachtet man die rasante Entwicklung des Internets seit seiner Kommerzialisierung vor ca. 12 Jahren und mutmaßt, welche rasanten Fortschritte in diesem Bereich in Zukunft auf sich warten lassen, dann muss man zu dem Schluss kommen, dass die Gesellschaft, im besonderen die jüngere Generation auf das Medium Internet vorbereitet werden muss.

Obwohl die Schüler ein hohes Maß an Motivation mitbringen, nutzen sie lediglich das Internet als Informations- und Kommunikationsquelle. Der Umgang mit diesem Medium fällt ihnen im Allgemeinen aufgrund des meist sehr benutzerfreundlichen Aufbaus der einzelnen Webseiten recht leicht, wobei es aber in der Regel leider auch bleibt. Damit nutzen die Schüler nur ein Teil des Internets. Es dient auf der anderen Seite aber auch zur Verbreitung von Informationen. Um Sie mit dieser Seite des Mediums bekannt zu machen, ist die im folgenden beschriebene Unterrichtsreihe gedacht.

2. Vorkenntnisse

Der Einstieg in das Thema Internet wurde mit einer Referat-Reihe begonnen. Nach einer Sammlung von Themen, die uns in diesem Bereich als wichtig erschienen, konnte sich jeder Schüler nach seinem Interessenschwerpunkt für ein Referat-Thema entscheiden. Mit den Referaten wurde so eine breite Grundlage geschaffen, die einen tieferen Einblick und sinnvollen Umgang mit dem Medium Internet ermöglichen sollte.
Themenbeispiele:

-Entstehung des Internets
-Provider
-Suchmaschinen
-Schutzmechanismen
-ICQ
-TCP/IP
-HTML
-Skriptsprachen

Weiterhin wurde die Textformatierungssprache HTML (Hypertext Markup Language) eingehend behandelt, die die Grundlage für die Programmierung von Webseiten und JavaScript bildet. Hierbei handelt es sich aber lediglich um eine statische Gestaltungsmöglichkeit von Webseiten.
Im Verlauf der Unterrichtsreihe wurden folgende Grundlagen zur Gestaltung von Webseiten gelegt:

-Grundgerüst einer HTML-Seite
-Formatierungsmöglichkeiten von Texten
-Listenerstellung
-Tabellenentwurf
-Aufbau einer Navigation zwischen Webseiten (Hyperlinks)
-Verwendung von Grafiken
-Formulargestaltung
-Cascading Stylesheets (CSS)
-Frames

Mit diesen Hilfsmitteln sind Schüler in der Lage, eine eigene Website aufzubauen.

Abschluss dieser Unterrichtssequenz war die Erstellung einer eigenen Website, in der sich die Schüler vorstellen sollten. So hatte ich bei der Bewertung dieser Arbeiten zusätzlich die Möglichkeit, meine Schüler und ihre Vorlieben besser kennen zu lernen.

Während dieser Abschlussarbeit zu HTML wurde seitens der Schüler schnell der Wunsch nach weiterführenden Techniken geäußert, mit denen man dynamische Komponenten und Benutzeraktivitäten in die statischen Seiten einbauen kann. Hierzu gibt es verschiedene Möglichkeiten: JavaScript, PHP, Perl, CGI- Scripte, Java-Applets um nur einige zu nennen.

Insgesamt ist anzumerken, dass eine sehr hohe Leistungsbereitschaft bei den Schülern festzustellen war. Das Thema Internet begeistert Schüler und aufgrund dessen, dass man mit sehr einfachen Mitteln und in sehr kurzer Zeit anspruchsvolle Webseiten selbst erstellen kann, beflügelt sie um so mehr.

Alles in allem waren das die besten Voraussetzungen für eine Unterrichtsreihe zu JavaScript.

3. JavaScript

Der reine Umgang mit HTML lässt sehr schnell den Wunsch erwachsen, aktive Inhalte in eine Webseite einzubinden, bei der der Nutzer mit der Seite interagieren kann. Hierzu sind verschiedene Skript-Sprachen entwickelt worden, die in den HTML-Code eingebettet werden können. Skriptsprachen sind Programmiersprachen, die in einer bestimmten Umgebung, dem Browser, ablaufen und dort die bereitgestellte Funktionalität nutzt.

Dies ist einer der Hauptunterschiede zu Programmiersprachen wie C++ und Java, die diese Funktionalität selbst erzeugen bzw. bereitstellen müssen. Man unterscheidet zwischen clientseitigen und serverseitigen Skriptsprachen. Der Unterschied besteht in der Abarbeitung des Programmcodes, der einmal auf dem Client und zum anderen auch auf dem Server interpretiert werden kann. Zu den clientseitigen Techniken gehören: JavaScript, VB-Script, Java-Applets und Flash. Server Side Includes (SSI), Common Gateway Interface (CGI), Active Server Pages (ASP) und PHP Hypertext Preprocessor (PHP) gehören zu den serverseitigen Scriptsprachen.

JavaScript-Anweisungen werden in den HTML-Code eingebettet. Dies birgt in sich schon ein Problem. Der Klartext des Programms ist für jeden ersichtlich und unterliegt keinem Schutz. Für geschützte Seiten ist JavaScript also ungeeignet.

Der Programmcode wird auf dem im Browser integrierten Interpreter im Gegensatz zu einer Programmiersprache wie C++ abgearbeitet, welcher diesen durch einen Compiler in Maschinensprache übersetzt.

Die Leistungsfähigkeit von JavaScript besteht darin, dass man mit deren Hilfe mit den einzelnen Komponenten einer HTML-Seite interagieren kann. Man kann Benutzereingaben überprüfen, HTML-Komponenten animieren, eigene Funktionalitäten definieren, auf Benutzter direkt ohne Umwege über einen Server reagieren, usw..

In JavaScript sind weiterhin prozedurale und objektorientiert Komponenten enthalten. Beim Syntax-Vergleich erkennt man starke Ähnlichkeiten zu C, Java und PHP. Durch diese Vorteile eignet sich JavaScript sehr gut als Einstiegsprogrammiersprache, da wesentliche Komponenten einer „echten Programmiersprache“ wiederzufinden sind, jedoch zunächst überflüssiger Ballast ausgelassen wurde.

Die Anfänge von JavaScript (ursprünglich LiveScript genannt) gehen auf die Firmen Sun und Netscape zurück, die den ersten JavaScript- fähigen Browser Ende 1995 veröffentlichten. Zu dieser Zeit nutzten etwa 85% der Surfer die Browser von Netscape. Mit jeder neuen Version des Netscape-Navigators wurden die Funktionalitäten der Programmiersprache weiter

ausgebaut. Parallel dazu entwickelte Microsoft aus firmentaktischen Gründen die Scriptsprache Jscript (um eigene Standards zu entwickeln), die aber mit JavaScript zu 80% kompatibel ist. Diese zwar recht große Schnittmenge macht es aber sehr schwer, einheitliche Seiten für beide Browser zu entwerfen, da gerade die fehlenden 20% sehr oft Verwendung finden (einer der wesentlichen Unterschiede ist die Darstellung von Ebenen). Um „Cross-browserfähige“ Webseiten zu gestalten, muss man meist für jeden Browser unterschiedliche Seiten programmieren.

Das ECMA-Konsortium hat sich mit der Standardisierung von Skriptsprachen genauso wie das W3-Konsortium um die Standardisierung von HTML bemüht, jedoch erfüllen JavaScript und JScript diesen Standard. Dieses Problem scheint aber von der Zeit gelöst zu werden, da der Marktanteil des Microsoft Browsers immer mehr zunimmt und Netscape mittlerweile immer unbedeutender wird.

4. Zielsetzungen und Begründung der Arbeit

Die vorliegende Arbeit entstand aus einem Unterrichtskonzept für die Jahrgangsstufe 11/2, welches für einen Orientierungskurs entwickelt wurde. Der Zeitrahmen für die Unterrichtsdurchführung entspricht einem Halbjahr mit jeweils 3 Wochenstunden (51 Stunden).

Ziel dieser Arbeit ist es nicht, den Leser in die Sprache JavaScript einzuführen bzw. eine Referenz zu erstellen (von denen schon sehr viele gute und weniger gute im Internet oder in den Printmedien veröffentlicht worden sind), sondern ein Unterrichtskonzept anzubieten, nach welchem man den Unterricht der Jahrgangsstufe 11/2 gestalten kann.

Die Arbeit richtet sich an Informatiklehrer, die in dieser Jahrgangsstufe unterrichten. Die Arbeit soll aber keine Vorgabe sein, sondern ist als Orientierungshilfe zu verstehen, in der jeder seine eigenen Schwerpunkte setzen kann.

Nachdem in der 11/1 die Grundlagen des Internets und die Textformatierungssprache HTML Unterrichtsgegenstand war, sollen laut Lehrplan Informatik in der 11/2 die Grundlagen der Programmierung gelegt werden.

Da die Schüler in der Jahrgangsstufe 11/1 sehr begeistert im Umgang mit dem Medium Internet gearbeitet haben und ich deren Motivation für den Kurs 11/2 weiter ausnutzen wollte, habe ich mich in Absprache mit den Schülern für eine Programmiersprache entschieden, die „webtauglich“ ist.

Für die Wahl „JavaScript“ sprachen und sprechen einige gewichtige Punkte:

Eine Entwicklungsumgebung ist auf jedem handelsüblichen PC bereits installiert, da ein solches Script (eingebettet in eine HTML-Seite) über einen Browser angezeigt werden kann. Weiterhin ist auf jedem Rechner ein Texteditor vorhanden, mit dem sich die Programmzeilen des Scripts eingeben lassen. Damit entstehen für den Schüler keine zusätzlichen Kosten für Software und keine Probleme bei der Installation der sonst nötigen Software. Jeder Schüler, der zuhause die Möglichkeit besitzt, an einem Computer zu arbeiten, ist somit in der Lage, die im Unterricht erlernten Programmiertechniken weiter zu vertiefen und auszubauen.

Weiterhin spricht für JavaScript, dass die Syntax denen anderer Programmiersprachen ähnelt. Die Schreibweise der einzelnen Befehle entspricht denen in PHP und Java und ist ebenfalls angelehnt an C++, sodass der Einstieg in eine dieser Programmiersprachen den Schülern zu einem späteren Zeitpunkt nicht allzu schwer fallen sollte.

Ein weiterer Vorteil ist der Umgang mit Variabelentypen. In sehr vielen Programmiersprachen muss man bei der Deklaration von Variablen den Datentyp mit angegeben. Da der Umgang mit Variablen den Schülern im allgemeinen sehr schwer fällt,

eignet sich JavaScript besonders, weil die Typzuweisung von dem Interpreter automatisch vorgenommen wird und deshalb anfangs kein zusätzliches Problem schafft.
So eignet sich also JavaScript zur Einführung in die Programmierung besonders, da einige Probleme im Vergleich zu anderen Programmiersprachen nicht vorhanden sind, es kostenlos für jeden zur Verfügung steht, fast alle Eigenschaften einer Programmiersprache besitzt, Grundzüge der Objektorientierung beinhaltet, sehr einfach zu erlernen ist und von der Syntax an einige der wichtigsten Programmiersprachen angelehnt ist.
Selbst für die Jahrgänge 12, 13 ist dieser Weg der Einführung einer Programmiersprache eine Überlegung Wert. Das Halbjahr 12/I „Objektorientierte Modellierung“ könnte man in Anlehnung an JavaScript mit Hilfe der Programmiersprache Java durchführen. Da in JavaScript schon objektorientierte Ansätze zu finden sind (diese jedoch nicht konsequent verfolgt werden) und eine in den Grundbefehlen nahezu identische Syntax im Vergleich zu Java besitzt, sollte der Übergang von der 11 in die 12/I den Schülern nicht schwer fallen.
Im Bereich des Themas „Datenbanken“ der 12/II hätte man die Möglichkeit, über Webdatenbanken das gelernte umzusetzen und zu üben. Hierzu bietet sich PHP in Verbindung mit MySQL an, die kostenlos zur Verfügung stehen und einfach zu installieren sind. Auch hier wäre aufgrund der ähnlichen Syntax der Übergang nicht zu groß für die Schüler. Weiterhin könnte man die Motivation der Schüler in diesem eher theoretisch, abstrakt anmutenden Thema „Datenbanken“ ausnutzen, da sie sich für diese Techniken besonders interessieren.
Für eine Einführung in die Programmierung ist aber meiner Meinung nach die Wahl der Programmiersprache weniger von Bedeutung, sondern eher die Wahl von geeigneten Beispielen und eine zielorientierte und strukturierte Vorgehensweise an Probleme wichtig, denn im Laufe der Zeit werden weitere neue und leistungsfähigere Programmiersprachen entwickelt, bzw. der Funktionsumfang einer bestehenden erweitert. Die Schüler sollten deshalb nach ihrem Informatikunterricht in der Lage sein, sich selbständig in eine neue Programmiersprache einzuarbeiten und diese zu erlernen.
Selbst die Verwendung von Delphi in der 12/I, welches sich an vielen Schulen als Programmiersprache durchgesetzt hat, stellt meines Erachtens bei dem Übergang der 11 in die 12 kein größeres Problem dar, da die wesentlichen Grundstrukturen in beiden Sprachen gleich sind, sie sich in der Syntax aber stark unterscheiden. Man muss sich also hauptsächlich im Bereich der Syntax umstellen, vorausgesetzt man hat das Wesen der einzelnen Befehle begriffen.
Die Kernkompetenzen eines Informatikers bestehen darin, Probleme zu analysieren, zu modellieren und Algorithmen zu entwickeln, um die gestellten Probleme mit Hilfe des Computers zu lösen. Hierbei spielt die Programmiersprache eher eine untergeordnete Rolle,

vielmehr stellt sie bei diesen Arbeitsgängen lediglich das zu beherrschende grundlegende Handwerkszeug dar.
Während des Unterrichts sollten ebenfalls soziale Kompetenzen wie Teamfähigkeit gefördert werden, die gerade in diesen Berufssparten notwendig sind, da größere Anwendungen nur in einem Team in akzeptablen Zeiten entwickelt werden können.
Dazu habe ich versucht, in sehr vielen Unterrichtssequenzen kleinere und größere Projekte einzubauen, bei denen die Schüler zunächst unter Anleitung und später eigenständig in kleineren bzw. größeren Gruppen arbeiten konnten. Zu Anfang der Unterrichtsreihe waren dies erwartungsgemäß kleinere Projekte, die nicht mehr als 2-3 Schulstunden mit Heimarbeit in Anspruch nehmen. Nachdem die Schüler schon einige Erfahrungen in JavaScript gesammelt hatten, wurden auch 3 Großprojektprojekte in Angriff genommen.
Bei den Projektarbeitsphasen sollten die Schüler lernen, dass sie sich individuell nach ihren Fähigkeiten und Stärken für die Gruppe einzusetzen hatten. Somit existieren in den einzelnen Gruppen Experten, die ihr Wissen in Form von Kurzvorträgen oder Einzelgesprächen an andere weitergeben können. Dabei ist es nicht unbedingt erforderlich, dass jeder einzelne Schüler in der Lage ist, jeden Teil des Projekts selbst zu programmieren, sondern das die Konzepte verstanden worden sind. Diese Grundgedanken findet man auch in der Expertenmethode wieder, welche sich gerade im Fach Informatik sehr schön umsetzen lassen.
Es gibt mehrere Möglichkeiten, eine Programmiersprache einzuführen. Man kann nacheinander die einzelnen Konstrukte und Operatoren abhandeln und dazu einige Beispiele nachvollziehen und programmieren lassen. Vorteil dieser Methode ist, dass man eine umfassende, strukturierte und vollständige Einführung in eine Programmiersprache bieten kann. In der Literatur findet man häufig ein solches Vorgehen. Diese Methode ist aber meiner Meinung nach für eine Schulklasse nicht sehr vorteilhaft, da die Abfolge der Einzelthemen sehr stringent und wenig motiviert sind. Die Schüler würden bei einer solch stringenten Abhandlung nach einer gewissen Zeit aus Langeweile dem Unterricht nicht mehr folgen. Weiterhin ist es wenig sinnvoll, Schülern ohne Motivation einen Sachverhalt klarzumachen.
Aus diesem Grund habe ich mich dazu entschlossen, möglichst von Problemstellungen auszugehen, bei denen die Schüler die Motivation verspüren, sich mit der Lösung zu beschäftigen bzw. Ziele zu definieren, auf die man gemeinsam hinarbeiten kann. Solche Ziele können kleine und größere Projekte sein, die man im Moment noch nicht angehen kann, da einem die Grundlagen zur Lösung fehlen, und es so einen Sinn macht, einige Befehle, Funktionen oder Sprachkomponenten einer Programmiersprache zu erlernen. Wichtig hierbei sollte aber nicht die vollständige Abarbeitung der gesamten Referenz einer Programmiersprache sein, sondern die Vorgehensweise, wie man sich das benötigte Wissen aneignet. Bei dieser exemplarischen Aneignung und einigen Wiederholungen sind die Schüler

sehr schnell in der Lage, die Ausdrucksweisen in einer Referenz [3] zu interpretieren und umzusetzen, was für spätere Problemfälle die Selbstständigkeit der Schüler fördert.
Bei der Suche nach möglichen Algorithmen kommt man sehr schnell auf die wesentlichen Merkmale einer Sprache, mit der man den Lösungsweg eines Problems formulieren kann.
Weiterhin sollte man zwecks Übung immer wieder die gelernte Sprachkomponenten einer Programmiersprache im Sinne eines Spiralcurriculum wiederholt anwenden lassen, um so einen vertrauten und eingängigen Umgang mit diesen zu erreichen. Es ist aber darauf zu achten, dass man Probleme nicht immer aus ein und demselben Bereich verwendet, sondern die Möglichkeiten einer Programmiersprache durch eine facettenreiche Auswahl der Problemstellungen aufzeigt.
Um einen roten Faden im ersten Teil des Halbjahres zu haben, baut sich der vorliegende Kurs um das Thema Taschenrechner auf, da man an diesem alle grundlegenden Sprachkomponenten einführen kann.
Es wurden zwei größere Projekte umgesetzt, die sich mit der Programmierung von den bekannten Glückspielen Würfeln und Roulette beschäftigten. Als Abschluss der Unterrichtsreihe sollte eine Website für den Kurs erstellt werden, bei dem die gesamte Gruppe als Team zusammenarbeiten musste.
Mit diesen Vorgaben habe ich ein Konzept entwickelt, welches im folgenden vorgestellt werden soll.

5. Einführung in JavaScript

Der wesentliche Unterschied zwischen JavaScript-basierten und statischen Webseiten ist der, dass die erstgenannten auf Eingaben des Users reagieren können. Die Funktionalität dieser Seiten werden in der Regel über Ereignisse mit dem JavaScript-Code verbunden. Tritt also ein bestimmtes Ereignis ein, so wird eine bestimmte Aktion ausgelöst. Basis dieser Auslöser sind meist Formulare. Da sich die Schüler im Verlauf des letzten Halbjahres mit HTML, insbesondere auch mit der Erstellung von Formularen und deren Versendung per Email eingehend beschäftigt haben, erscheint es mir sinnvoll, auf diesen Grundlagen aufzubauen. Im Sinne eines Spiralcurriculums werden so bekannte Sachverhalte wiederholt und erweitert.

Die Schwierigkeit bei der Einführung einer Programmiersprache besteht darin, dass in den meisten Büchern eine sehr ausführliche Darlegung der einzelnen Grundlagen stattfindet. Gleichbedeutend damit ist, dass das erste akzeptable und motivierende Beispiel in „Kapitel x“ (mit x >> 1) geschrieben wird, also wenn man die ersten 100 Seiten eines Buches abgearbeitet hat. In der Regel hat man aber meist schon den Inhalt der ersten 50 Seiten vergessen. Eine solche Vorgehensweise erscheint mir also für den Informatik-Unterricht nicht sinnvoll, da es wenig motivierend und sehr uneffektiv ist.

Mit der in dieser Arbeit beschriebenen Vorgehensweise möchte ich Schüler möglichst schnell in die Programmiersprache JavaScript einführen, um in recht kurzer Zeit aufwendigere Programme entwickeln lassen zu können und sich den Hauptaufgaben des Informatikunterrichtes, der Analyse, der Beschreibung und Modellierung komplexer Systeme, zuwenden zu können[1].

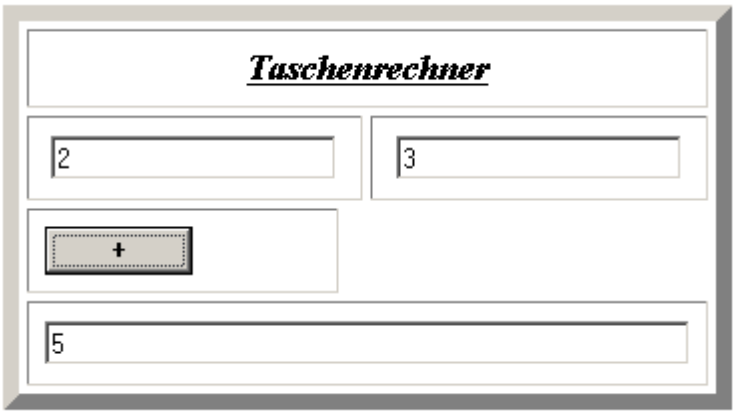
Aus den oben angegebenen Gründen habe ich mich dazu entschieden, mit einem „anspruchsvolleren Beispiel“ in JavaScript einzusteigen. Hierzu habe ich den Schülern den Programmcode eines Taschenrechners gegeben, der lediglich die Grundrechenart Addition beherrscht.

```
<html>
<head>
    <title>Taschenrechner</title>
</head>
<style type="text/css">
<!--
.button {  width:74px; text-align:center;
        font-family:System,sans-serif;
        font-size:100%; }
-->
</style>
```

[1] Lehrplan Informatik (Gymnasialer Bildungsgang) Seite 3

```
<script language="JavaScript">

    function Addition()
      {
        var a,zahl1,zahl2;
        zahl1 =Number(window.document.Taschenrechner.Zahl1.value);
        zahl2 =Number(window.document.Taschenrechner.Zahl2.value);
        a = zahl1+zahl2;
        window.document.Taschenrechner.Ergebnis.value = a;
      }
</script>
<body>
    <form name="Taschenrechner">
      <table border="8" cellspacing="4" cellpadding="8" align="center">
        <tr background="blue">
          <th colspan="4" align="center"><i><b><font
size="4"><u>Taschenrechner</u></font></b></i> </th>
        </tr>
        <tr>
          <td colspan="2" align="center"><input type="Text" name="Zahl1"></td>
          <td colspan="2" align="center"><input type="Text" name="Zahl2"></td>
        </tr>
        <tr>
          <td><input type="Button" value="+" class="button" onClick="Addition()"></td>
        </tr>
        <tr>
          <td colspan="4"><input type="Text" name="Ergebnis" size="50"></td>
        </tr>
      </table>
    </form>
</body>
</html>
```

Abbildung 1: Erscheinungsbild der HTML-Seite Taschenrechner



Einen Taschenrechner findet man im Zubehör jedes Betriebssystems wieder. Ein Programm entwickeln zu können, das die gleiche Funktionalität besitzt wie dieser, ohne eine Programmiersprache zu beherrschen bzw. mit einem solchen Problem in eine

Programmiersprache einzusteigen sollte eine ausreichende Motivation für Schüler sein, sich diesem Problem anzunehmen. Der spielerisch, experimentelle Zugang sollte die Motivation noch weiter fördern.

Die Aufgabe der Schüler bestand nämlich darin, die Funktionalität des Taschenrechners auf die 4 Grundrechenarten auszuweiten. Dazu müssen Variablen deklariert, initialisiert und umgewandelt, Ereignisse hinterlegt, Funktionen entwickelt und Komponenten Werte zugewiesen werden, ohne genau zu wissen, was man da eigentlich macht. Zu Anfang eine große Herausforderung und ein enormes Pensum. Für dieses große Pensum spricht aber die Intuitivität des gegebenen Programm-Codes.

Ein mir bewusstes Problem ist, dass Schüler den Sinn des Programms im ganzen zwar erschließen können, jedoch nicht den jeder einzelnen Programmzeile.

In einer anschließenden Besprechung des Programmcodes sollten folgende Dinge angesprochen werden:

- Einbindung des JavaScript-Code über folgendes Tag
  - **<script language="JavaScript">**
    Programmcode
    </script>
- Ort der Einbindung im HTML-Dokument
  - Die unterschiedlichen Browser akzeptieren den Skriptbereich an jeder Stelle eines HTML-Dokuments.
  - Aus Gründen der Übersichtlichkeit sollte der JavaScript-Code möglichst separat eingebunden werden. Eine mögliche, sinnvolle und standardisierte Stelle liegt zwischen dem Head und Body-Bereich einer HTML-Seite.
  - Bei größeren Programmen kann man den JavaScript-Code auch in eine weitere Datei auslagern. Der Sinn dieser Auslagerungsdatei ist die Übersichtlichkeit. Man kann aber auch eine Art Modularisierung erreichen, und so ein Projekt gut zu strukturieren. Im Gegensatz zu anderen Programmiersprachen ist die Auslagerungsdatei einzusehen. Man kann also den Quellcode nicht verstecken, sondern nur den Weg zu diesem verschleiern.

  <script language = „JavaScript“ type=“text/javascript“ src=“url/meinSkript.js“></script>
- Sinn und Zweck von Variablen

  Variablen dienen zur Speicherung von Daten. Bei der Deklaration von Variablen wird dieser vom Betriebssystem ein Speicherplatz zugewiesen, unter der der Wert der Variablen abgespeichert wird. Der Name einer Variablen kann nun synonym für deren Wert verwendet werden. In JavaScript existieren 4

grundlegende Datentypen: Ganzzahlen, Gleitkommazahlen, Strings (Texte) und boolsche Werte.

Deklaration: **var zahl1;** Der Variable i wird vom Betriebssystem ein Speicherplatz zugewiesen

Wertzuweisung **var zahl1 = 5;** Die Variable wird deklariert und gleichzeitig der Wert 5 zugeordnet (Datentyp Ganzzahl)

var zahl1; Die Deklaration kann auch von der Wertzuweisung getrennt

zahl1 = 5; werden. Hierzu müssen diese nicht direkt untereinander stehen

Hierbei erkennt der Interpreter im Vergleich zu anderen Programmiersprachen, um welchen Datentypen es sich handelt. Dieser muss nicht explizit bei der Deklaration angegeben werden

- Objekthierarchie von JavaScript

Während des ersten Programms muss man nicht gleich auf die Objekthierarchie bzw. auf Objekte eingehen. Vielmehr geht es darum, den Schülern den Zugriff auf die einzelnen Komponenten einer Webseite, eines Dokuments eingebettet in eine Webseite oder ein Formular eingebettet in ein Dokument, welches wiederum in eine Webseite eingebettet ist usw.. Der Zugriff auf die einzelnen Komponenten erfolgt über die folgende Objekthierarchie:

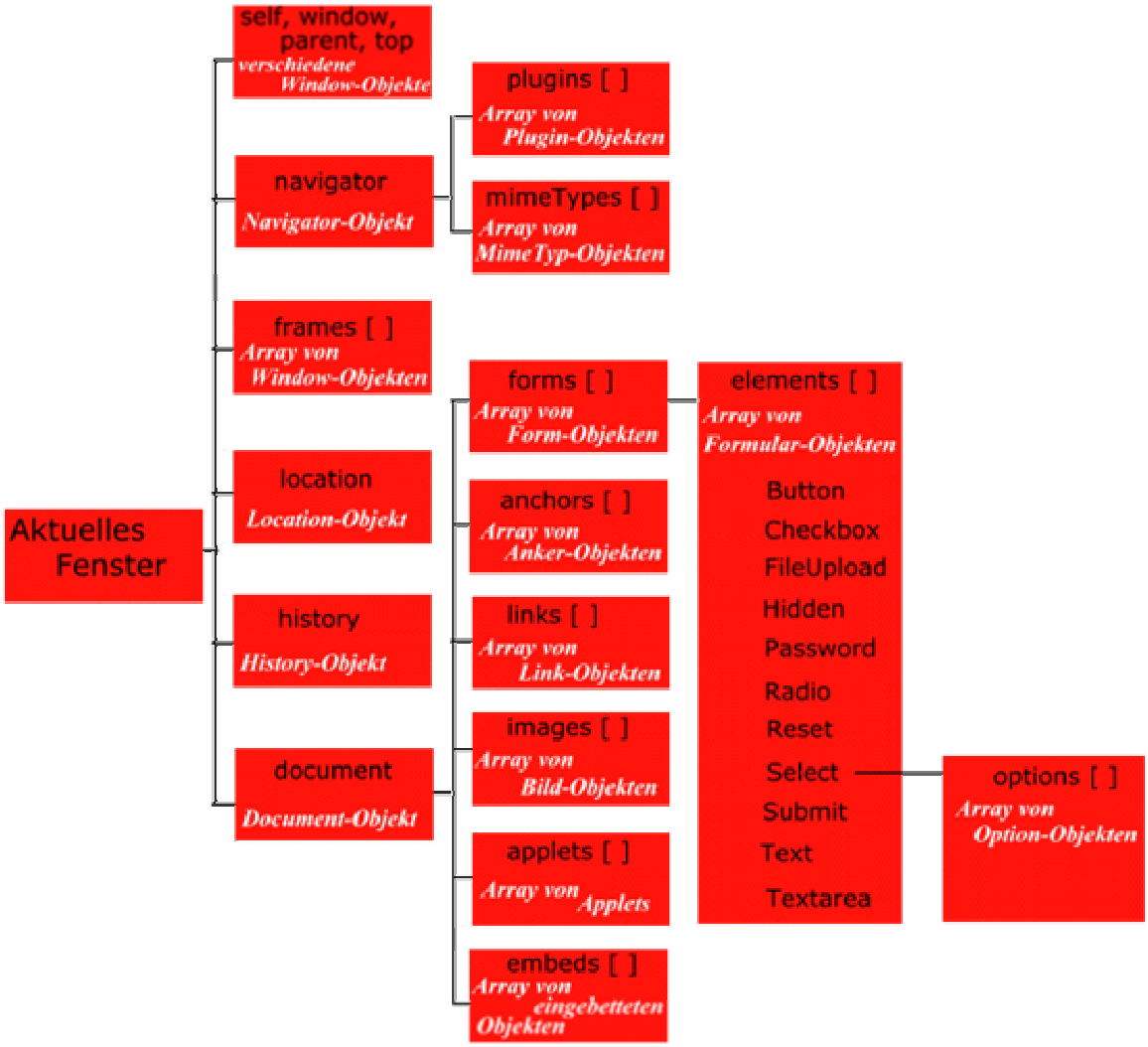


Abbildung 2: Objekthierarchie des Window-Objektes in Javascript (http://www.ik.fh-hannover.de/ik/person/becher/edvhist/sprachen/jscript.html)

Durch die Angabe der Punktnotation kann man auf jede beliebige Seitenkomponente und deren Attribute und Methoden zugreifen. Attribute kennen die Schüler schon von den einzelnen Tagattributen.

Beispiel: **window.document.Taschenrechner.Ergebnis.value**

Mit dieser Angabe kann man auf den Wert des Textfeldes Ergebnis im Formular Taschenrechner zugreifen, welches in dem aktuellen Dokument zu finden ist. Bei gegebener Eindeutigkeit muss nicht die komplette Punktnotation angegeben werden, sondern nur ab der Ebene der letzten Differenz von ähnlichen Ausdrücken.

- Die wichtigste Möglichkeit, JavaScript-Code in einer HTML-Seite ausführen zu lassen, ist die über das Eintreten von Ereignissen. Zu jedem Tag gibt es verschiedene Ereignisse, die Eintreten können. Beispiele:

  - bei Änderung eines Feldes (onChange)

- beim Aktivieren von Komponenten (onFocus)
- beim Anklicken von Komponenten (onClick)
- usw. (siehe Referenzen)

Diese Ereignisse werden wie ein Attribut in einen Tag eingebunden und anschließend mit der JavaScript-Funktion belegt, die beim Eintritt dieses Ereignisses ausgeführt werden soll.

Beispiel:

<input type="Button" value="+" class="button" onClick="Addition()">

Wird der Button mit der Maus „angeklickt“, dann wird die im JavaScript-Bereich definierte Funktion Addition() ausgeführt

- Funktionen

An dieser Stelle möchte ich nicht auf die verschiedenen Möglichkeiten von Funktionsdeklarationen eingehen, da man an dieser Stelle die Schüler damit überfordert.

Zu diesem Zeitpunkt sollten die Schüler wissen, das es zunächst zwei verschiedene Arten von Funktionen existieren, zum Einen die vordefinierten Funktionen, die seitens der Sprache

JavaScript als Funktionalität bereitgestellt werden:

Beispiel: die Funktion **Number()**

Number(window.document.Taschenrechner.Zahl2.value);

Die Funktion Number() wandelt den Wert des Textfeldes um in den Datentyp Ganzzahl bzw. Gleitkommazahl, da die Angabe in einem Textfeld automatisch als String (Text) angesehen wird. Diese Funktionalität wird von JavaScript mitgeliefert.

Auf der anderen Seite findet man die eigens für ein gestelltes Problem entwickelten Funktionen, die man sehr oft selbst programmieren muss, weil sie im Sprachumfang von JavaScript noch nicht enthalten ist. In dem Programmcode des Taschenrechners wird ein solche Funktionalität durch die Funktion **Addition()** gegeben, in der die Inhalte zweier bestimmter Textfelder des Formulars in Zahlen umgewandelt und addiert werden, um anschließend in einem weiteren Textfeld als Ergebnis der Addition ausgegeben zu werden. Hier tritt allerdings automatisch eine Umwandlung der Zahl a in einen String auf, was auf ein nicht ganz schlüssiges Konzept in der Sprache hinweist. Aber an dieser Stelle erleichtert das System die Arbeit, weil man die Umwandlung nicht selbst vornehmen muss.

a = zahl1+zahl2;

window.document.Taschenrechner.Ergebnis.value = a;

- Syntax einer Funktion:

function Name_der Funktion()

{

Programmcode

}

Auch hier sei erwähnt, dass es noch unterschiedlich Arten von Funktionen gibt, die sich durch ihre Eingabeparameter und Ausgabeparameter unterscheiden. Diese Betrachtungen soll zu einer späteren Zeit durchgeführt werden.

Wie oben angedeutet, sollte die Besprechung des Quelltextes aber nicht eine alles erschöpfende Darlegung bieten, sondern es soll vielmehr mit den Schülern analysiert werden, wozu die einzelnen Zeilen notwendig sind. Die Variabelenbenennung ist aus diesem Grunde bezeichnend für die Verwendung gewählt worden, womit die Lesbarkeit des Programmtextes erhöht wird. Die Lesbarkeit des Programmcodes erhöht sich ebenfalls durch konsequente Einrückung, woran die Schüler ebenfalls gewöhnt werden sollen. Vorteile bietet die Einrückung ebenfalls für den Lehrer, da man die Strukturen und mögliche Fehler schneller entdecken kann. Die Erhöhung der Lesbarkeit des Quelltextes stellt somit ein weiteres Lernziel der Unterrichtssequenz dar.

Haben die Schüler erst einmal verstanden, wie man JavaScript-Code in einen HTML-Quelltext einbindet, sind sie in der Lage, sehr viele Programme selbst zu entwickeln, bei denen die Benutzereingaben in einer einfachen Rechnung weiterverarbeitet werden. Man kann aber an dieser Stelle nicht davon ausgehen, dass die Schüler das Besprochene vollständig begriffen haben. Aber anhand des Quellcodes vom „Taschenrechner“ sind Schüler in der Lage, durch „abgucken“ die einzelnen Bereiche auf die folgenden Probleme zu übertragen und einfache Algorithmen zu entwickeln.

Denkbare Problemstellungen:

- Zinsrechner für Jahres-, Monats- und Tageszinsen
- Währungsrechner
- Zwei-Punkte-Form, Punkt-Steigungsform für lineare Funktionen u.v.m.

Mit diesen weiteren Problemstellungen sollen die Schüler das Erlernte umsetzen. Bei der Auswahl kann man den Schülern aber freie Wahl lassen. Die Erfahrung hat gezeigt, dass die Schüler im Entwickeln von eigenen Problemstellungen sehr erfinderisch sind. Die Beschäftigung mit eigenen Problemstellungen und deren Lösung erhöht zusätzlich noch einmal die Motivation der Schüler, sich mit der Programmiersprache auseinander zusetzen. Dabei sollte man aber darauf achten, dass man hier die Art der Problemstellungen sehr genau

eingrenzt, da die bisherigen Möglichkeiten der Umsetzung mit JavaScript noch durch den bisher bekannten Sprachumfang beschränkt wird.
Im weiteren Verlauf kann man nun das unvollständige Wissen der Schüler im Bezug auf die oben angeführten Sachverhalte weiter verfeinern. Vorteil dieser Methode ist es, dass man behandelte Programmkomponenten, bzw. Sprachkonstrukte des öfteren wiederholt, was erfahrungsgemäß zur besseren Verinnerlichung führt.

6. Bedingte Anweisungen

6.1. If- Anweisung

Bei der Entwicklung von Algorithmen kommt man sehr häufig an Stellen, bei denen Entscheidungen getroffen werden müssen, deren Ausgang vom aktuellen Zustand des Systems abhängen. In diesem Kapitel geht es darum, den Schülern zu vermitteln, wie der Computer diese Entscheidungen treffen kann.
Aufhänger für diese Einführung soll die Erweiterung unseres Taschenrechners sein. Die Betragsfunktion bietet sich hier an, bei der eine Entscheidung getroffen werden muss, ob die eingegebene Zahl negativ oder positiv ist. Ist die eingegebene Zahl negativ, so soll der positive Wert dieser Zahl zurückgegeben werden. Ansonsten soll die Zahl selbst zurückgegeben werden.
Die Syntax der ersten bedingten Anweisung, die die Schüler kennen lernen sollen, lehnt sich an den letzten Satz an, der direkt aus dem Erfahrungsbereich der Schüler für das Treffen von Entscheidungen stammt:

Wenn (Bedingung = wahr) **dann**
führe folgende Anweisungen aus,
ansonsten
führe diese Anweisungen aus

<u>Syntax:</u>

```
if (Bedingung)
{
  Anweisungen, die ausgeführt werden müssen, wenn die Bedingung wahr ist
}
else
{
   Anweisungen, die ausgeführt werden müssen, wenn die Bedingung falsch ist
}
```

Notwendig für eine solche Entscheidung ist die Formulierung der Bedingungen, die man mit den Schülern üben muss, weil diese recht ungewohnt für sie sind.

Hierzu ist es erforderlich, Vergleichsoperatoren einzuführen:

Vergleichsoperator	*Wirkung*	*Beispiel*	*Ergebnis*
„= =“	Werte sind gleich	3 = = 3 2+4 = = 7	True False
„! =“ „< >“	Werte sind ungleich	2 + 2 ! = 7 2 ! = 2	True False
„<“	Linker Wert ist kleiner	4 < 7 „abc“ < „abd“ 7 < 4	True True False
„>“	Linker Wert ist größer	7 > 2 „X“ > 10 „qrs“ > „qrst“	True True False
„< =“	Linker Wert ist kleiner oder gleich	5 < = 10 11 < = 10	True False
„> =“	Linker Wert ist größer oder gleich	„buch“ > = „Buch“ „Abc“ > = „abc“	True False

Tabelle 1: Vergleichsoperatoren in JavaScript

Hierbei ist bei Zeichenketten zu beachten, das Ziffern niedriger eingestuft sind als Buchstaben und Kleinbuchstaben höher als Großbuchstaben.

Zu Beginn des Unterrichts kann man die Problemstellung der Betragsfunktion für den schon entwickelten Taschenrechner in den Raum stellen.
Um die Definition der Betragsfunktion den Schülern ins Gedächtnis zu rufen, sollte man diese anhand einiger Beispiele besprechen. Ganz unbekannt sollte diese Funktion aber nicht sein, da sie Gegenstand des Mathematikunterrichts der 11/I war. Mit Hilfe des Zahlenstrahls kann man die 2 verschiedenen Fälle, die bei diesem Problem auftreten, gut diskutieren. Der Zahlenstrahl teilt sich in positive und negative Zahlen ein, wobei die 0 zu den positiven Werten zu zählen ist.
Die Eigenschaft einer Zahl a, positiv zu sein, lässt sich durch folgende Bedingung abfragen
Bedingung (a > = 0):

- Ist die Bedingung wahr, so ändert die Betragsfunktion nichts

- Ist die Bedingung falsch, dann muss die Zahl kleiner als 0 und damit negativ sein. Die Betragsfunktion ändert das Vorzeichen. Mathematisch gesehen kann dies realisiert werden, indem man die Zahl mit -1 multipliziert

Bespricht man an dieser Stelle die oben angegebenen Vergleichsoperatoren (diese Tabelle kann man den Schülern als Informationsblatt übermitteln), so könnte eine mögliche Aufgabe für die Schüler daraus entstehen, alternative Bedingungen für die Betragsfunktion zu formulieren.

Hat man sich diesen Sachverhalt mit den Schülern erarbeitet, ist die Implementierung des Problems in JavaScript ein einfaches. Durch die Angabe und Besprechung der Syntax einer bedingten Anweisung kann man mit den Schülern die Funktion Betrag entwickeln.

Aufgabe der Schüler: Umsetzung der Betragsfunktion in JavaScript und Integration dieser Funktion in den Taschenrechner.

Quellcode der Betragsfunktion

```
<html>

 <head>
     <title>Betrag</title>
 </head>

 <script language="JavaScript">
       function Betragfunktion(a)
       {
         if (a >= 0)
         {
           window.document.Formular.Betrag.value = a;
         }
         else
         {
           window.document.Formular.Betrag.value = -a;
         }
       }
 </script>

 <body>
   <form name="Formular">
     <table>
       <tr>
          <td>Bitte geben Sie eine Zahl ein: </td>
          <td><input type="Text" name="Zahl"></td>
       </tr>
       <tr>
          <td>Der Betrag der Zahl ist: </td>
          <td><input type="Text" name="Betrag"> </td>
       </tr>
```

```
      <tr>
        <td><input type="Button" value ="Betrag" onClick =
"Betragfunktion(Number(window.document.Formular.Zahl.value))"> </td>
        <td><input type="reset"> </td>
      </tr>
    </table>
  </form>
 </body>

</html>
```

Haben die Schüler diese Aufgabe erledigt, sollten komplexere Anwendungen erstellt werden.
In der Mathematik spielt die Bestimmung von Nullstellen eine zentrale Rolle. Da sich die Schüler in der 11/II im Mathematikunterricht mit der Kurvendiskussion von ganzrationalen Funktionen beschäftigt haben und die Bestimmung von Nullstellen mit Hilfe der p-q-Formel eine häufige Fehlerquelle darstellt, leuchtet es den Schülern ein, diesen Algorithmus zu automatisieren. Hat man dies getan, so treten die üblichen Vorzeichen nicht mehr auf. Der Computer dient lediglich als Rechenknecht.
Diese Überlegungen können den Schülern schon in diesem Stadium der Unterrichtsreihe vermitteln, wozu das Formulieren und Programmieren nötig und sinnvoll ist, nämlich zur Abarbeitung von Problemen, deren Lösung „mit der Hand“ sehr fehleranfällig ist und immer nur „stupide“ nach dem selben Schema zu lösen sind.
Die Abstraktion dieses Problems sollte mit den Schülern gemeinsam geschehen, da die Formulierung der einzelnen Bedingungen für sie in diesem Stadium eine Überforderung darstellen würde. Hat man diese Bedingungen jedoch mit den Schülern erarbeitet, sollten diese aber selbständig in JavaScript übertragen werden

-Fall Diskriminante > 0 => Es existieren 2 Lösungen (x_1, x_2 nach p-q-Formel)

-Fall Diskriminante = 0 => Es existiert 1 Lösung (x_1 nach p-q-Formel)

-Fall Diskriminante < 0 => Es existiert keine Lösung

Problematisch für die Schüler bei diesem Algorithmus ist, dass man hier eine geschachtelte If-Abfrage verwenden muss. Eine sehr einfache aber erlaubte und erwünschte Lösung wäre, wenn der Schüler 3 If-Abfragen zur Diskriminanten umsetzen würde. Hier können die Schüler erfahren, dass es nicht nur eine Lösung für ein Problem gibt.
Eine mögliche Lösung:

```
function Nullstellenbestimmung()
        {
        var p,q,d,x1,x2;
        p = Number(window.document.Eingabe.p.value);
        q = Number(window.document.Eingabe.q.value);
        d = Math.sqrt(p*p/4-q);
```

```
if (d < 0)
{
  alert("Es gibt keine Nullstellen der quadratischen Funktion");
}
else
  if (d == 0)
  {
    x1 = -p/2;
    alert("Die quadratische Funktion besitzt eine Nullstelle bei x = "+x1);
  }
  else
    {
    x1 = -p/2+d;
    x2 = -p/2-d;
    alert("Die quadratische Funktion besitzt Nullstellen bei x1 = "+x1+" und x2 ="+x2);
    }
};
```

Schnelle Schüler könnten an dieser Stelle den Algorithmus von normierten ganzrationalen Funktionen 2. Grades auf unnormierte erweitern, welches kein größeres Problem darstellen sollte.

Bei der soeben behandelten Problemstellung ist in großem Maße die Verwendung von Variablen gefordert. Die Schüler müssen selbständig die Anzahl der Variablen, die Deklaration und Initialsierung vornehmen, wobei sich diese Kompetenzen weiterentwickeln.

6.2. Case-Anweisung

Die einfache if-Abfrage bietet nur zwei mögliche Ausgänge einer Entscheidung. Wie wir eben bei dem Algorithmus zur p-q-Formel gesehen haben, muss man bei meheren Entscheidungsmöglichkeiten eine geschachtelte if-Abfrage verwenden. Bei n möglichen Ausgängen einer Abfrage müsste man theoretisch (n-1) if-Abfragen ineinander schachteln. Dies stellt sich aber als ein mühsames Projekt dar.

JavaScript stellt in einem solchen Fall, wie auch sehr viele andere Programmiersprachen, ein weiteres Sprachkonstrukt bereit, die case-Anweisung. Ein Variable kann bei dieser auf mehrere Alternativen getestet werden. Zu jeder Alternative kann man einen Anweisungs-Block anschließen.

```
switch (Variable)
{
case Wert1:    Anweisungsteil, falls Variable den Wert1 besitzt
        break;
case Wert2:    Anweisungsteil, falls Variable den Wert2 besitzt
        break;
```

:

:

case Wertn: Anweisungsteil, falls Variable den Wertn besitzt

break;

default: Anweisungsteil, falls Variable keinen der oben angegebenen Werte annimmt;

Der default-Teil ist fakultativ.

Ein Beispiel aus dem Erfahrungsbereich der Schüler, der dieses Problem beinhaltet, ist eine einfache Anwendung von üblichen Quizsendungen, bzw. Multiple-Choice-Fragen.
Wer wird Millionär:
In der Auswertung einer solchen Frage sollte möglichst die oben diskutierte case-Anweisung verwendet werden.
Aufgabe der Schüler ist es, ein solches Quiz mit Auswertung zu programmieren.

Bei der Umsetzung der Aufgabe sind die Schüler möglichst frei bei der Gestaltung der Fragen und des Designs. Für Schüler, die bei dieser Aufgabe Schwierigkeiten haben, kann man auf Self-HTML verweisen, die für jedes Sprachkonstrukt ein Beispiel vorhält, anhand dem man sich noch einmal den Hintergrund verdeutlichen kann. Dies fördert die Selbstständigkeit.
Bei der Umsetzung dieses Spiels ist es erforderlich, dass für jede Frage eine Auswerte-Funktion geschrieben werden muss. Dies ist zugegebenermaßen eine recht unelegante Variante, aber zum jetzigen Zeitpunkt nicht anders möglich. Um einer eleganteren Variante den Weg zu ebnen, könnte man im Anschluss daran den folgenden Quelltext besprechen.

```
<html>
 <head>
     <title>Wer wird Millionär</title>
 </head>
 <script language="JavaScript">

     function Auswerten(Auswahl)
     {
      switch(Auswahl)
      {
        case "a": alert("Budapest ist die Hauptstadt von Ungarn");
        break;
        case "b": alert("Sofia ist die Hauptstadt von Bulgarien");
        break;
        case "c": alert("Warschau ist die Hauptstadt von Polen");
        break;
        case "d": alert("richtig");
        break;
```

```
    }
    };
  </script>
  <body>
      <form name="Fragenkatalog" >
      <table border="1">
        <tr>
          <td colspan="3">Frage 1: Wie heißt die Hauptstadt von Rumänien? </td>
        </tr>
        <tr>
          <td><input type="Radio" name="f1" value="a" onclick="Auswerten('a')"> </td>
          <td>a)</td>
          <td>Budapest </td>
        </tr>
        <tr>
          <td><input type="Radio" name="f1" value="b" onclick="Auswerten('b')"> </td>
          <td>b) </td>
          <td>Sofia </td>
        </tr>
        <tr>
          <td><input type="Radio" name="f1" value="c" onclick="Auswerten('c')"> </td>
          <td>c) </td>
          <td>Warschau </td>
        </tr>
        <tr>
          <td><input type="Radio" name="f1" value="d" onclick="Auswerten('d')"> </td>
          <td>d) </td>
          <td>Bukarest </td>
        </tr>
      </table>
      </form>
  </body>
</html>
```

Abbildung 3: Oberfläche der entwickelten Webseite zu „Wer wird Millionär“

Die Erweiterung stellt die Art der Funktion dar. Diese Funktion hat einen Eingabeparameter. Ein Vorteil dieser Variante der Funktion ist, dass man in der Funktion auf keinerlei

Frage 1: Wie heißt die Hauptstadt von Rumänien?		
○	a)	Budapest
○	b)	Sofia
○	c)	Warschau
○	d)	Bukarest

Komponente des Formulars zurückgreift. Diese Funktion könnte deshalb auch so in einem

anderen Programm verwendet werden. Dies erhöht die Wiederverwendbarkeit der entwickelten Funktionen.
Erinnern wir uns noch einmal an die Funktion zur p-q-Formel. Da dieses Problem sicherlich auf mehreren Web-Seiten Verwendung finden kann, könnte man die Funktion ebenfalls losgelöst von den Seitenkomponenten einer speziellen Seite schreiben:
Funktionsdeklaration: function p_q_Formel(p,q)
Bei einem Aufruf dieser Funktion könnte man nun die einzelnen Inhalte von Textfeldern als p und q an die Funktion übergeben. (Umsetzung als mögliche Hausaufgabe)

Einschub zum Weiterarbeiten:
Aber warum könnte dieser Aufruf mit Eingabeparameter an dieser Stelle nützlich sein? Schreibt man eine Auswerte-Funktion für das Spiel „Wer wird Millionär“, die unabhängig von der Frage ist, so könnte man die Frage und die vom User gewählte Antwort mit dem Aufruf übergeben. Ziel dieser Überlegung ist die Nutzung nur einer Auswertefunktion.
An dieser Stelle entsteht quasi eine Baustelle, die später noch einmal aufgegriffen werden kann. Denkbar an dieser Stelle wäre, dass man nach der Einführung von Arrays die Fragennummer und die richtige Antwort in einem Mehrdimensionalen Array abspeichert. Wird also die Fragenummer und die vom User gewählte Antwort an die Funktion Auswerten() übergeben, so könnte sich die Funktion die richtige Antwort aus dem Array besorgen und mit der User-Antwort vergleichen.
=> mögliches Ergebnis: Eine Auswerte-Funktion für beliebig viele Fragen
Einsparung von sehr vielen Zeilen im Quelltext

Die Schüler lernen an diesem Beispiel eine weitere Variante der Funktion kennen, ohne dass das Sprachkonstrukt der Funktion bis in Detail besprochen wurde.
JavaScript ist eine Sprache, mit der man Interaktivität aber auch Bewegung in HTML-Seiten einbinden kann. Da als Endprodukt dieses Kurses eine klasseneigene Homepage gestaltet werden soll, habe ich mich dazu entschieden, die If-Anweisung unter Bewegungsaspekten einzubinden.
Im folgenden wird nur das Ebenen- Konzept erörtert, das vom Microsoft Internet Explorer verstanden wird. Gleichzeitig möchte ich darauf hinweisen, das in Netscape ein anderes Konzept gefordert ist. Dies macht aber die Schwierigkeit bei der Webseitenprogrammierung aus, Seiten Cross-Browser-fähig (d.h. in den unterschiedlichen Browsern gleichzeitig funktionsfähig zu programmieren). In der Regel muss man bei der Ebenenprogrammierung zunächst den Browsertyp im JavaScript-Teil abfragen und für die unterschiedlichen Konzepte jeweils eine eigene Seite vorhalten (was sich als sehr mühsam erweist, recht unelegant wirkt,

aber meist nicht anders umsetzbar ist). Für weitere Informationen zu diesem Thema verweise ich den Leser auf Kapitel 14. Grafikprogrammierung

In dem folgenden Programm wird die Ebene Buchstabe als Objekt des Dokuments definiert, wobei die Eigenschaft des Styles left und top manipuliert werden.

```
<html>
  <head>
        <title>Bewegter Buchstabe</title>
  </head>

  <script language="JavaScript">

  var i,hoehe;
  breite=window.screen.width;
  i=0;

  function HinBewegen()
  {
   if(i <= breite)
   {
     Buchstabe.style.left=i;
     Buchstabe.style.top=100;
     i+=10;
     window.setTimeout('HinBewegen()',20);
   }
   else
   {
     window.setTimeout('ZurueckBewegen()',20);
   }
  }

  function ZurueckBewegen()
  {
   if(i > 0)
   {
     Buchstabe.style.left=i;
     Buchstabe.style.top=100;
     i-=10;
     window.setTimeout('ZurueckBewegen()',20);
   }
   else
   {
     window.setTimeout('HinBewegen()',20);
   }
  }
  </script>
  <body onload="HinBewegen()">
       <div id="Buchstabe" style="position:absolute; left:0px;top:0px">A</div>
  </body>
</html>
```

Ein solches Programm könnte man den Schülern zur Analyse als Hausaufgabe geben und als Erweiterung nicht nur horizontale, sondern auch vertikale Bewegungen fordern.

Im o.a. Beispiel werden einige Objekte und deren Eigenschaften bzw. Methoden angesprochen. Auf die Objektorientierung von JavaScript bin ich aber an dieser Stelle mit den Schülern nicht tiefer eingestiegen, da JavaScript in letzter Konsequenz nicht voll objektorientiert ist. Die Schüler sollen hier spielerisch den Umgang mit den Objekten lernen und sich schon einmal damit vertraut machen. Davon ausgehend hat man für die konsequente Einführung der objektorientierten Programmierung in der 12/I eine gemeinsame Ausgangsbasis.

Weiterhin findet man schon einen rekursiven Aufruf in den Funktionen HinBewegen() und ZurueckBewegen(), da in der jeweilige Funktion die Funktion bis zu einer Abbruchbedingung immer wieder aufgerufen wird. Das dies im Gegensatz zur iterativen Abarbeitung von Anweisungen ein weiteres wichtiges Konzept der Programmierung darstellt, kann man den Schülern zur Information geben. Dieses Thema wird in Kapitel 11. Rekursion noch einmal aufgegriffen.

7. Projekt Würfelspiel

Bisher haben die Schüler meistens kleinere Projekte im Informatikunterricht realisiert, die binnen einer Doppelstunde bzw. 3 Unterrichtsstunden (Wochenunterrichtszeit) umgesetzt werden konnten. Im Lehrplan Informatik für den gymnasialen Bildungsgang[2] werden kommunikative und kooperative Arbeitsformen, schöpferisches Denken und die Analyse, Beschreibung und Modellierung komplexer Systeme besonders hervorgehoben. In der Projektarbeit wird man diesen Forderungen in großem Maße gerecht.

Die Schüler haben den Umgang mit Variablen, Formularen und bedingten Abfragen geübt. Mit diesen Voraussetzungen sind sie in der Lage, etwas größere Projekte in Angriff zu nehmen.

Bei der Wahl von Projektthemen muss man darauf achten, nicht zu komplexe motivierende Beispiele zu finden, die die Schüler mit ihrem Wissenstand umsetzen können. Da aber das Leistungsspektrum der Schüler im Informatik-Unterricht meist sehr breit ist, die Leistungsbereitschaft ebenfalls sehr unterschiedlich ist, sollte die Aufgabe für jeden Schüler umsetzbar sein.

Um diesen Vorgaben gerecht zu werden, sollten den Schülern die Anforderungen an das entstehende Programm sehr detailliert vorgegeben werden. Da Vorgaben des Lehrers meist sehr demotivierend auf Schüler wirken, kann man den Anforderungskatalog von den Schülern erarbeiten lassen. Anhand des Anforderungskataloges kann man dann ebenfalls mit den Schülern erarbeiten, welche Kriterien man für welchen Notenbereich erfüllen muss. Ziel dieses Anforderungsprofils ist es, den Schülern seitens der Bewertung eindeutige Vorgaben zu machen. Dadurch, dass er von den Schülern erstellt worden ist, findet eine Veränderung der Lehrerrolle statt. Er ist nicht derjenige, der die Vorgaben macht, sondern sie selbst sind es. Diese Vorgehensweise bezieht den Schüler in den Entscheidungsprozess in gewissem Maß mit ein und lässt die Arbeit selbstbestimmter wirken. Die Bewertungskriterien wurden von den Schülern selbst erarbeitet, und damit sind die Schüler maßgeblich dafür verantwortlich, welche Note sie auf ihr Projekt bekommen.

Im Verlauf der Unterrichtsreihe wurden sehr viele mathematische Probleme bearbeitet. Aus diesem Grund habe ich mich dazu entschieden, ein Spiel von den Schülern umsetzen zu lassen.

Da die Programmierkenntnisse der Schüler sich auf HTML, Variablen und bedingte Abfragen in JavaScript beschränken, habe ich mich aus oben genannten Gründen dazu entschieden, dass Spiel selbst vorzugeben.

[2] Lehrplan Informatik (gymnasialer Bildungsgang) Seite 3

Bei dem gewählten Spiel handelt es sich um ein fiktives Würfelspiel mit 3 Würfeln. Die Spielregeln lauten wie folgt:

Spieleinsatz: 20 €

Gewinn beim Erscheinen einer 6: 20 €

Gewinn beim Erscheinen von zwei 6: 50 €

Gewinn beim Erscheinen von drei 6: 100 €

Der Einstieg in das Projekt könnte spielerisch erfolgen, indem der Lehrer der Klasse dieses Glückspiel natürlich fiktiv anbietet. Oder man gibt die Spielregeln vor und lässt die Schüler in Gruppen diese Spiel mit fiktivem Einsatz spielen. So können die Schüler das Spiel zunächst spielerisch kennen lernen. In dieser Phase ergeben sich schon gewisse Fragen an das Spiel.

Je nachdem, in wie weit die Wahrscheinlichkeitsrechnung in der Sekundarstufe I eingeführt worden ist, kann man das Spiel (nicht zu ausgedehnt) theoretisch beleuchten, damit der nötige Hintergrund den Schülern nicht verborgen bleibt.

Ein solches Spiel eröffnet sehr viele Möglichkeiten der Umsetzung:

- grafische vs. numerische Ausgabe der Spielausgänge
- Programmiertechnische Umsetzung der Spielregeln
- Auswertung des Spielausganges
- rel. Häufigkeiten vs. Wahrscheinlichkeiten
- Kontoführung für mehrmaliges Spiel
- usw.

Es liegt an der Ausarbeitung der Klasse, welche weiteren Features die Schüler für die Umsetzung der Problemstellung anstreben.

Für eine ausreichende Leistung sollte aber mindestens ein funktionsfähiges Programm abgeliefert werden, welches das Spiel in einem Durchgang simuliert. D. h. die Umsetzung der Spielregeln, Durchführung eines Wurfes und die Auswertung eines Spielausgangs sollte jeder gelöst haben.

Wie die Umsetzung eines solchen Spiels aussehen könnte, zeigt der nachfolgende Bildschirmausdruck. Das dazugehörige Programm ist während der Durchführung dieser Reihe entstanden.

Die Schwierigkeit für die Schüler bei der Umsetzung dieses Spiels besteht in der Modellierung der Spielsituation, die Wahl der Variablen, die Verarbeitung der Spielregeln, die Einbindung der Zufallsfunktion, die Kontoführung, die Berechnung der Werte für die statistische Auswertung und die Umsetzung einer grafischen Anzeige des Würfelergebnisses.

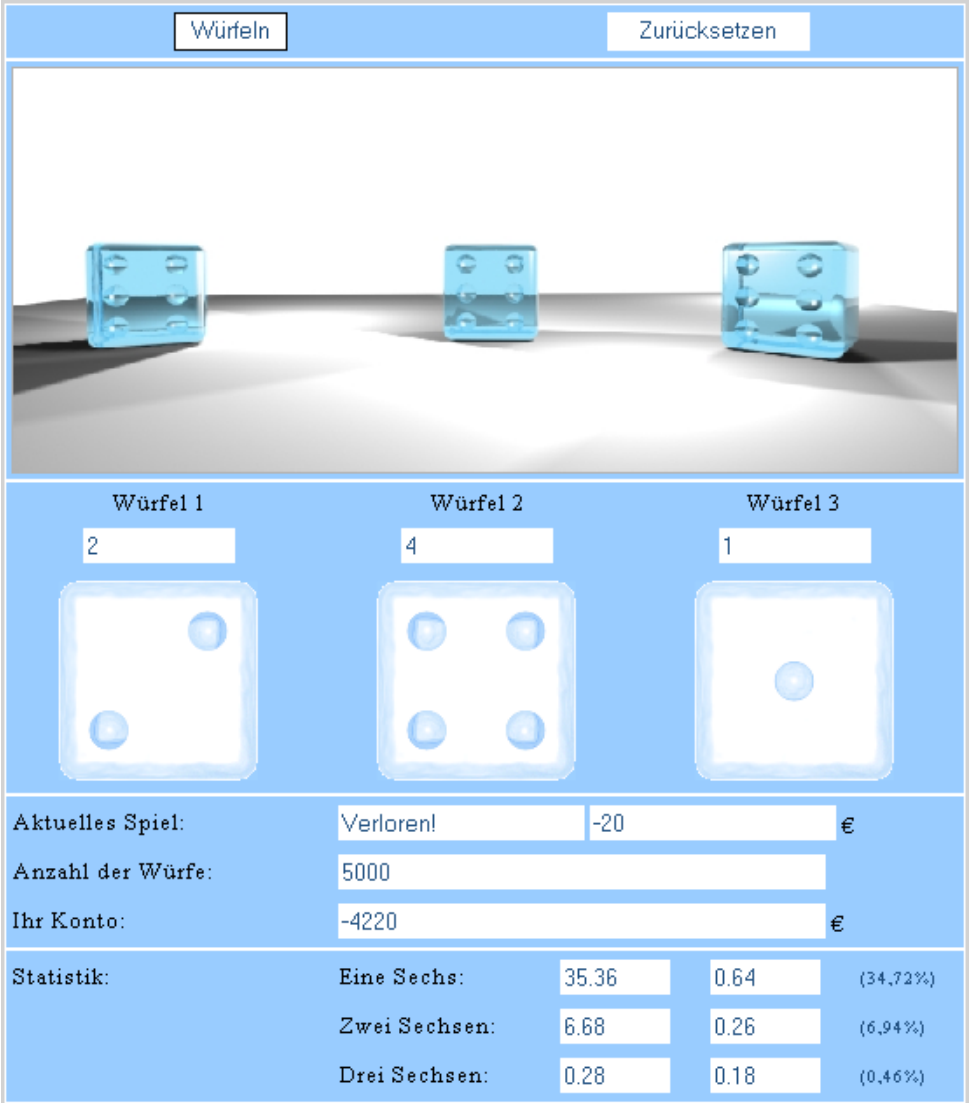
Das hier gezeigte Ergebnis erfüllt alle oben genannten Kriterien und gehört damit zu den sehr guten Umsetzungen.

Es zählt aber nicht nur das äußere Erscheinungsbild dieses Programms, sondern auch der Programmierstil des Quellcodes, der ein Teil der Note ausmacht.

In Anbetracht der Tatsache, dass das Thema Funktionen und strukturierte Programmierung noch nicht auf der Tagesordnung stand, hat sich der Autor dieses Programms bei der Strukturierung einige Mühe gegeben. Die Variablenbezeichnungen hätten aber an dieser Stelle ausführlicher sein können. Durch Kommentare bzw. bessere Namensgebung hätte ein Bezug zu deren Funktionalität hergestellt werden können.

Abbildung 4: Oberfläche eines Würfelspiels (Der Quellcode befindet sich auf der CD)

Das Programm befindet sich auf der CD unter „Beispiel“ im Bereich Würfelspiel



8. Schleifen

8.1. Vorüberlegungen

Bei der Formulierung von Algorithmen müssen sehr oft Anweisungen in der gleichen Weise wiederholt werden. Ein einfaches Beispiel wäre hier die Ausgabe der natürlichen Zahlen von 1 bis 1000. Hierbei könnte man diese 1000 Ziffern alle einzeln auf die jeweilige Webseite schreiben, d.h. Ziffer für Ziffer. Da aber alle natürlichen Zahlen bis auf die Null ein gemeinsames Merkmal besitzen, nämlich das eine Zahl sich um eins vom Vorgänger als auch von ihrem Nachfolger unterscheidet, kann man dies für das oben gestellte Problem ausnutzen. Man schreibt zunächst die erste Zahl, erhöht diese Zahl um eins (eine solche Funktionalität existiert schon in JavaScript) und schreibt die so entstandene Zahl wiederum auf die Webseite, erhöht diese Zahl um eins usw.. Diese Abfolge der Anweisungen: Zahl schreiben, Zahl um eins erhöhen müsste nun 1000 Mal wiederholt werden. Für solche Wiederholungen stellt eine Programmiersprache wie auch JavaScript Schleifen bereit.

In JavaScript werden 3 verschieden Arten von Schleifen bereitgestellt:

- For-Schleife
- While-Schleife
- Do-While-Schleife

Jede dieser 3 Varianten hat in bestimmten Fällen einen Vorteil gegenüber den anderen beiden Typen.

Eine mögliche Vorgehensweise wäre nun, dass man die Syntax dieser 3 Varianten einer Schleife angibt, und die Unterschiede zwischen diesen herausarbeiten lässt. Dies könnte in Gruppenarbeit geschehen, bei der sich die Schüler durch Recherche dieses Wissen aneignen. Dies würde die Selbständigkeit der Schüler fördern und die Kompetenz vermitteln, sich aus den gängigen Informationsquellen Wissen anzueignen.

Aber warum haben sich die Entwickler von Programmiersprachen die Mühe gemacht, 3 Schleifentypen bereitzustellen? Wie sind sie darauf gekommen, eine müsste doch eigentlich genügen?

An dieser Stelle erscheint es mir wichtig, dass Schüler den gleichen Prozess wie die Entwickler mitmachen sollten.

Ich möchte zunächst Problemstellungen bearbeiten, bei denen eine vorgegebene Anzahl von Schleifendurchläufen erforderlich ist. Hierfür bietet sich die For-Schleife an. Sie ist zugleich die Schleifenart, die nur eine beschränkte Art von Problemstellungen lösen kann. Weiterhin scheint mir diese Art von Schleife mit einer vorgegebenen Zahl von Durchläufen die

intuitivste für Schüler zu sein. Nachdem die For-Schleife bei einigen Beispielen verwendet worden ist, kann man die Schüler an die Grenze der Schleifenart führen. Die Schüler sollen selbst erfahren und begreifen, welche Kategorie von Problemen sich nicht mit diesem Schleifentyp lösen lassen. Ein bewusster Umgang mit den 3 Schleifentypen kann nur dann nachhaltig erreicht werden, wenn man diese Grenzen selbst erfahren hat. Eignet man sich die 3 Schleifentypen nacheinander an, lernt deren Syntax, so ist man noch nicht in der Lage, diese gegeneinander abzugrenzen. Dies ist nur durch Programmiererfahrung möglich.
Um mit den Schülern Zeit zu sparen, sollte der Lehrer zunächst Einführungsbeispiele für die For-Schleife vorgegeben, anhand derer die Syntax eingeübt werden kann. Sobald man den Eindruck hat, dass die Schüler diesen Schleifentyp programmieren können, sollte man Problemstellungen vorgeben, die sich eben nicht mit diesem Typ lösen lassen. So begreift der Schüler zunächst, dass ein neuer Schleifentyp notwendig ist, und kann abgrenzen, welche Probleme bei der For-Schleife entstehen können, bzw. wozu diese nicht in der Lage ist.

8.2. Einführung der Schleifenarten

Als Einstieg wähle ich wiederum ein Beispiel, bei dem die Funktionalität unseres Taschenrechners erweitert wird. Hierbei möchte ich an das oben angegebene Beispiel der Darstellung der ersten 1000 natürlichen Zahlen anknüpfen.
Ein schönes Beispiel ist die Summenformel von Gauss, bei der die ersten natürlichen Zahlen von 1 bis n aufsummiert werden. Bevor man mit der Programmierung beginnt, sollte man erst einige Beispiele an der Tafel mit den Schülern besprechen. Bei niedriger oberer Grenze sehen es Schüler nicht ein, ein Programm für diesen Sachverhalt zu schreiben. Erhöht man die obere Grenze, so versuchen schwache Schüler die Summe immer noch sukzessive durch Addition zu ermitteln. Begabtere Schüler sollten an dieser Stelle die unelegante Methode des Aufsummierens verwerfen und nach einer anderen suchen. Zum einen gibt es die Gauss-Formel, die die Schüler ermitteln könnten. Diese Lösung führt allerdings nicht auf Schleifen, sondern auf die allgemein bekannte Summenformel, die Gauss als Schüler bereits entdeckt haben soll.
Auf die Lösung mit Hilfe von Schleifen sollte der Lehrer die Schüler dahingehend lenken, dass sie versuchen, die Summation zu automatisieren und die Gesetzmäßigkeiten der natürlichen Zahlen ausnutzen. Haben die Schüler erkannt, dass immer wieder die gleichen Arbeitsschritte ausgeführt werden müssen, so kann man ihnen die Syntax und den Sinn der For-Schleife angeben, der das Wiederholen von Arbeitsschritten in JavaScript und anderen Programmiersprachen erledigt. An dieser Stelle war es mir aus oben genannten Gründen wichtig, dass die Schüler selbst darauf kommen, dass es für eine Programmiersprache

vorteilhaft wäre, wenn es ein Konstrukt gäbe, der wiederholt die gleichen Anweisungen durchführt.

Die Schüler haben im Verlauf des Halbjahres gelernt, gegebene Syntax in einer Programmiersprache umzusetzen. Aus diesem Grund kann man ihnen die Syntax der For-Schleife angeben und die Programmierung der Summenfunktion für den Taschenrechner überlassen.

Syntax der For-Schleife

```
for( var Laufvariable = Startwert;          // Laufvariable wird Deklariert und Initialisiert
     Abbruchbedingung;                      // Bedingung, die bei einem Schleifendurchlauf
                                            // erfüllt sein
     Laufvariable++ bzw. Laufvariable--)    // Änderung der Laufvariable n. einmaligem
                                            // Schleifendurchlauf
{
    Anweisungen;
}
```

An dieser Stelle verzichte ich aus Übersichtlichkeitsgründen auf die Angabe des ganzen Programmscodes und beschränke mich auf die in JavaScript gegebene Funktionalität (siehe Programm „Summe“ auf der CD). Wie man diese in ein Formular bzw. HTML einbindet, ist den Schülern mittlerweile bekannt.

```
function Summenfunktion(n)
    {
     var sigma = 0;
     for(var i =1; i<=n;i++)
     {
      sigma = sigma + i;
     };
     window.document.Formular.Summe.value = sigma;
    }
```

Hat man die Summenfunktion programmiert, kann man den Taschenrechner der Schüler sehr schnell noch mit der Funkionalität Fakultät erweitern lassen, bei der lediglich der Startwert mit eins initialisiert werden und in der Schleife anstelle des + ein * ersetzt werden muss.

Der verständige Umgang mit der Syntax der For-Schleife sollte an dieser Stelle an einem weiteren Beispiel vertieft werden, da die beiden Beispiele gleichgelagert waren. Hierzu bietet sich unter anderem die Bestimmung der Teiler einer eingegebenen Zahl:

Hierzu werden alle Zahlen, die kleiner als die eingegebene Zahl sind, auf Teilbarkeit mit der eingegebenen Zahl geprüft. Sind sie ohne Rest teilbar, handelt es sich um einen Teiler. Hat

die betrachtete Zahl nur einfache Teiler (1 und die Zahl selbst), so handelt es sich um eine Primzahl.

Hat man diesen Algorithmus verbal mit den Schülern erarbeitet, sollten die Schüler in der Lage sein, dies mit einer For-Schleife und einer If-Abfrage umzusetzen.

Hierzu sollte allerdings die Rechenart mod (%) mit den Schülern besprochen worden sein. Eine Optimierungsmöglichkeit besteht zusätzlich noch, indem man die Schleife bis zur Wurzel der eingegebenen Zahl laufen läst, wobei einige Schleifendurchläufe eingespart werden. An dieser Stelle könnte man kurz auf die Komplexität dieses Programms eingehen. Da man aber an dieser Stelle nicht den direkten Vergleich zu anderen Programmen mit derselben Funktionalität hat, würde ich diese Diskussion nicht zu weit ausdehnen. Effektiver ist diese Diskussion bei den Sortieralgorithmen in Klasse 12, da man hier direkt die Komplexität von zwei verschiedenen Algorithmen vergleichen kann.

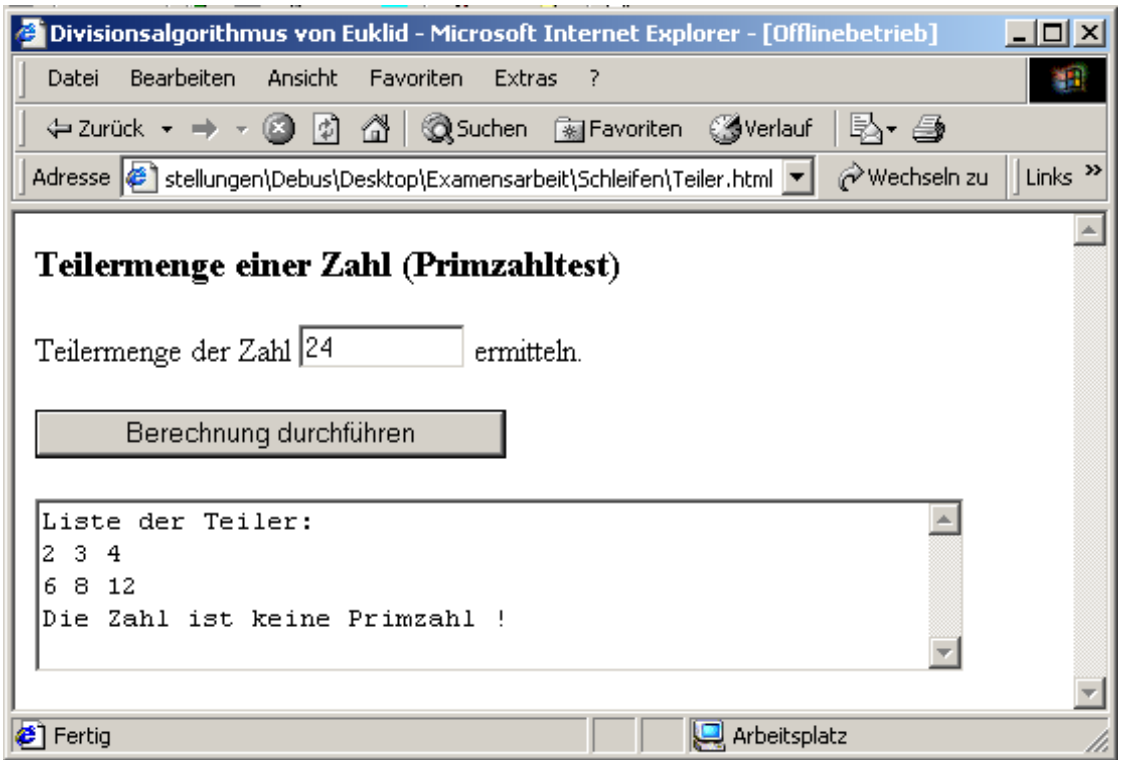
Ebenfalls sollte der Einsatz der von JavaScript bereitgestellten Funktionen angesprochen werden. Diese kann man erreichen, indem man über das Math-Objekt auf die einzelnen Funktionen über die Punktnotation zugreifen kann (z.B.: Math.floor, Math.sqrt usw.). In SelfHTML findet man eine Referenz der bereitgestellten mathematischen Funktionen und eine Beschreibung der einzelnen Funktionalitäten.

```
<html>
 <head>
  <title>Teilermenge von vorgegebenen Zahlen</title>
 </head>
 <script language="JavaScript">
 function berechne(f)
 {
  var aus1="";
  var aus2="";
  var zahl = f.EingegebeneZahl.value
  var limit = Math.floor(Math.sqrt(zahl)+.01)
  for (i=2;i<=limit;i++)
  {
   if (zahl%i == 0)
   {
    aus1 = aus1 + i + " "
    p2 = zahl / i
    aus2 = p2 + " " + aus2
   }
  }
  ausgabe = aus1 + aus2
  if (ausgabe == "")
  {
   ausgabe = "Die Zahl besitzt nur die Teiler 1 und " + zahl + ". \n"
   ausgabe += "Sie ist daher eine Primzahl !"
  }
  else
  {
```

```
    ausgabe = "Liste der Teiler: \n"
    ausgabe = ausgabe + aus1 + "\n" + aus2 + "\n"
    ausgabe += "Die Zahl ist keine Primzahl !"
   }

   f.ergebnis.value = ausgabe
  }
  </script>
  <body bgcolor="#ccccff">
   <h3>Teilermenge einer Zahl (Primzahltest)</h3>
   <form>
    <p>
     Teilermenge der Zahl <input type="text" name="EingegebeneZahl" size="10">
ermitteln.<br><br>
<input type="button" value="  Berechnung durchführen  " name="Berechnung"
onClick="berechne(this.form)">  <br><br><textarea rows="5" cols="55"
name="ergebnis"></textarea>
    </p>
   </form>
  </body>
</html>
```

Abbildung 5: Webseite zur Teilermengenbestimmung

Wichtig erscheint mir die Übergabe des Formulars als Parameter der Funktion berechne(this.form). Nachdem die Schüler schon gelernt haben, dass eine Funktion



Eingabeparameter haben kann, sollten sie ebenfalls wissen, dass auch Objekte bzw. ganze Seiteninhalte übergeben werden können. So erreicht man einen einfacheren Zugriff auf die Inhalte des Formulars, was im weiteren Verlauf von Vorteil sein wird.

Haben die Schüler die Umsetzung der Teilbarkeitsalgorithmus geschafft, so kann man davon ausgehen, dass sie in der Lage sind, die For-Schleife bei späteren Beispielen anwenden zu können.

Nun kann man einen Schritt weitergehen. Problemstellungen, die eine fest vorgegebene Anzahl von Wiederholungen erfordern, können mit der For-Schleife gelöst werden.

Wie sieht es aber aus, wenn die Anzahl der Wiederholungen nicht von vornherein bekannt sind. Ein solches Problem ist durch das Heron-Verfahren bekannt. Dieses Verfahren stellt eine schöne iterative Methode zur Bestimmung der Wurzel einer Zahl bis zu einer gegebenen Genauigkeit dar. Die Iterationsschritte sind von der eingegebenen Zahl und Genauigkeit der Wurzel abhängig.

Das Heron-Verfahren sollte bei der Einführung der Wurzelfunktion in Klasse 9 im Fach Mathematik besprochen worden sein. Der Algorithmus ist den Schüler bedingt geläufig.

Nach der Vorstellung dieses Algorithmus kann man zum Einstieg mit den Schülern einige Wurzeln „per Hand“ nach diesem Verfahren von gegebenen Zahlen bestimmen lassen. Die Umsetzung des Algorithmus in ein Programm sollten die Schüler zunächst eigenständig übernehmen. Da die Schüler nur die For-Schleife anwenden können, und das gegebene Problem nicht mit dieser Schleifenart zu lösen ist, werden sie sehr schnell an ihre Grenze geführt. Die Frage nach einer neuen Schleifenart ist an dieser Stelle vorprogrammiert. Doch bevor die Schüler mit der Do-While- bzw. While-Schleife vertraut gemacht werden, sollte der Unterschied in den Problemstellungen genau herausgearbeitet werden, damit die Schüler in folgenden Problemen selbst und sicher entscheiden können, welcher Schleifentyp zu wählen ist.

```
<html>
  <head>
    <title>Heronverfahren</title>
  </head>
  <script language="JavaScript">
    function Heron(a,b)
    {
      var Zahl,Genauigkeit,Wurzel,xn,yn;
      Zahl = Number(a);
      Genauigkeit = Number(b);
      xn = 1;
      do
      {
        yn = Zahl / xn;
        xn = (yn+xn)/2;
      }
      while (Math.abs(xn-yn) > Genauigkeit);
      alert("Die Wurzel von "+Zahl+" mit einer Genauigkeit von "+Genauigkeit+" ist "+xn);
    }
```

```
  </script>
  <body>
    <form name="Formular">
        Geben Sie eine Zahl ein, von der die Wurzel gezogen wird: <input type="Text" name="Zahle"></p>
        Geben Sie die Genauigkeit an: <input type="Text" name="Genauigkeite"></p>
        <input type="Button" value="Wurzelziehen" onClick = "Heron(window.document.Formular.Zahle.value,window.document.Formular.Genauigkeite.value)">
    </form>
  </body>
</html>
```

Die Anweisung muss bei diesem Programm mindestens einmal durchlaufen werden und am Ende wird die Genauigkeit überprüft. Falls die Bedingung am Ende der Do-While-Schleife erfüllt ist, wird der Anweisungsblock in der Schleife wiederholt.

Dabei ist die Anzahl der Schleifendurchläufe keine fest vorgegebene Zahl, sondern wird von der Abarbeitung des Anweisungsteils abhängig gemacht. Dies ist ein wesentlicher Unterschied zur For-Schleife und stellt deshalb ein erweitertes Konzept einer Schleife dar. Jede For-Schleife lässt sich mit Hilfe einer Do-While-Schleife und wie wir sehen werden mit einer While-Schleife umsetzen, aber nicht umgekehrt. Man hätte sich bei diesem Beispiel auch für eine While-Schleife entscheiden können, jedoch müsste man an dieser Stelle der Variablen yn eine Startwert schon vor der Schleife zuweisen.

Die Syntax der Do-While und der While-Schleife werden im folgenden angegebenen.

Do-While-Schleife:

```
do
{
  Anweisungen werden ausgeführt, solange die Bedingung true ist;
}
while(Bedingung)
```

While-Schleife:

```
while (Bedingung)
{
  Anweisungen werden ausgeführt, solange die Bedingung true ist;
}
```

Der Unterschied dieser beiden Schleifenarten ist der, dass im Vergleich zur While-Schleife bei der Do-While-Schleife der Anweisungsteil mindestens einmal ausgeführt wird.
Da die beiden Schleifenarten sehr verwandt miteinander sind, kann man mit den Schülern diese beiden Schleifenarten gemeinsam besprechen, und sie bei anschließenden Beispielen einüben.

Ein schönes aber anspruchsvolles Beispiel für eine While-Schleife ist die Erstellung eines Kredit-Finanzierungsplan.
Zum Einstieg kann man die Schüler im Internet nach einem solchen Finanzierungsplan recherchieren lassen. Dort werden die Schüler sehr schnell fündig werden. Solche Finanzierungsrechner werden von Banken auf den jeweiligen Seiten zur Verfügung gestellt. Mit ihnen kann man sich die monatlichen Raten und die Anzahl der Monate berechnen lassen, die man für die Rückzahlung eines Kredits benötigt. Bei der Recherche werden die Schüler einige Begriffe, z.B. Nominalzins, Effektivzins usw. finden, die erst einmal geklärt werden müssen.
Weiterhin sind einige Wiederholungen im mathematischen Bereich nötig. Eine Wiederholung der Zinseszins-Rechnung, die natürliche Logarithmusfunktion und die Exponentialfunktion sind nötig, um das gewünschte Programm umzusetzen.
Wichtig ist hierbei die Erarbeitung und Modellierung des Algorithmus, nicht so sehr die Programmierung, die an dieser Stelle nur geübt und angewendet werden soll. Die Auswahl meiner Beispiele für die Durchführung dieser Reihe lehnt sich sehr stark an die Mathematik an. Dies ist auch so beabsichtigt, da die Schüler eines Orientierungskurses begreifen müssen, dass bei der Programmierung von Algorithmen die Mathematik sehr oft als Hilfswissenschaft zu Rate gezogen werden muss. Für einen Grundkurs könnte man verstärkt Beispiele mit einem nichtmathematischen Kontext verwenden.

```
<html>
  <head>
    <title>Finanzierungsplan</title>
  </head>

  <script language="JavaScript">

    function Berechne(f)
    {
      var Kreditsumme1,Zinssatz1,Abtrag1;
      var Ausgabe="";
      var Monatsangabe = 0;
      Kreditsumme1 = Number(f.Kreditsumme.value);
      Zinssatz1 = Number(f.Zinssatz.value)/100+1;
      Zinssatz1 = Math.exp(Math.log(Zinssatz1)/12);
      Abtrag1 = Number(f.Abtrag.value);
```

```
    do
    {
       Ausgabe = Ausgabe + Monatsangabe + ".  " + Kreditsumme1 + ". \n";
       Kreditsumme1 =Kreditsumme1*Zinssatz1 - Abtrag1;
       Monatsangabe++;
    }
    while(Kreditsumme1 >= 0)
    Ausgabe = Ausgabe + Monatsangabe + ".  " + Kreditsumme1 + ". \n";
    Ausgabe = Ausgabe +"Der Kredit ist nach "+ Monatsangabe +" getilgt";
    f.Finanzierungsplan.value = Ausgabe;
  }
 </script>

 <body>
  <form>
     <table >
       <tr>
          <td>Kreditsumme: </td>
          <td> <input type="Text" name="Kreditsumme" ></td>
       </tr>
       <tr>
          <td>Zinssatz:</td>
          <td> <input type="Text" name="Zinssatz" ></td>
       </tr>
       <tr>
          <td>Abtrag: </td>
          <td><input type="Text" name="Abtrag" > </td>
       </tr>
       <tr>
          <td><input type="reset"></td>
          <td><input    type="Button"    onClick="Berechne(this.form)"    value="Berechnen"
width="100" ></td>
       </tr>
     </table>
     <textarea name="Finanzierungsplan" cols="50" rows="10"></textarea>
  </form>
 </body>
</html>
```

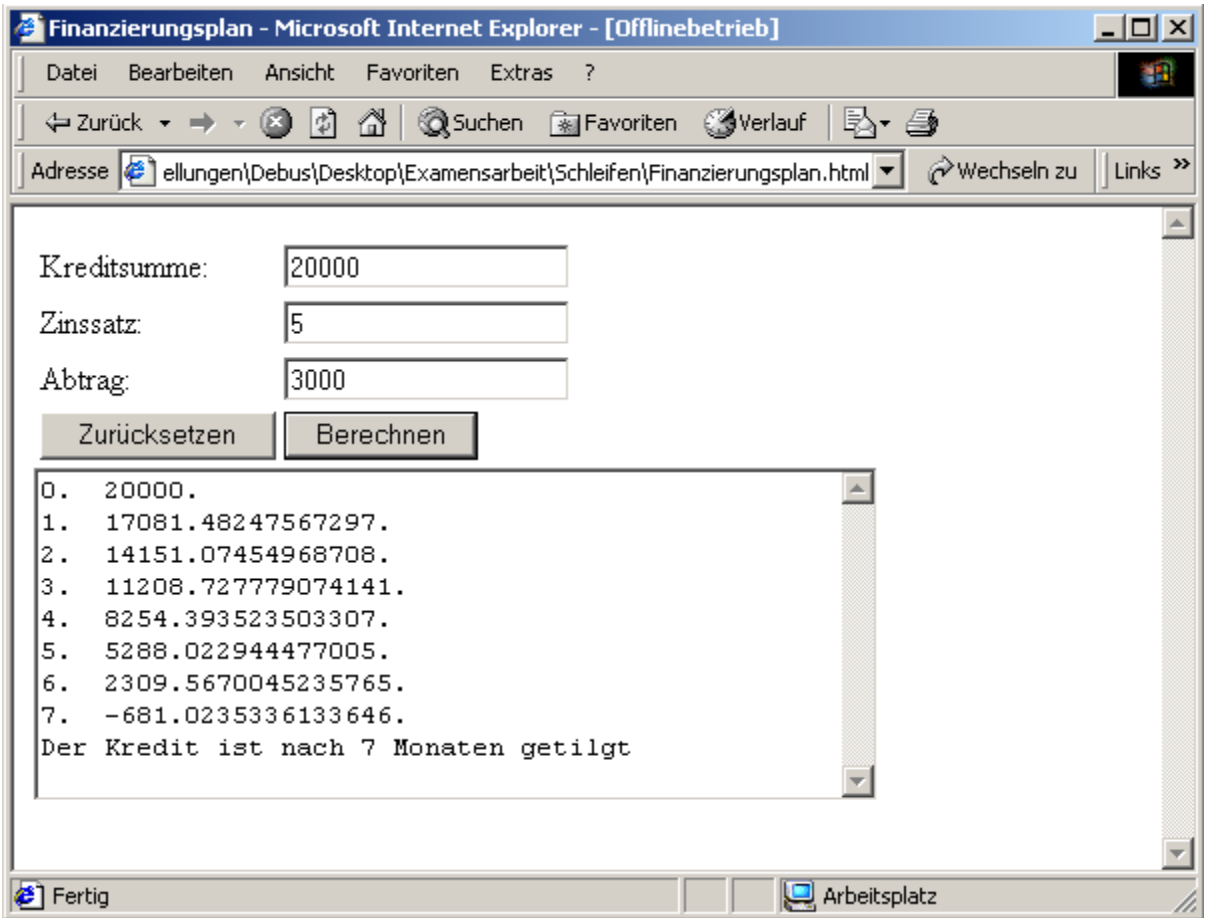


Abbildung 6: Webseite des Finanzierungsplaners

Die Erfahrung bei der Anwendung zeigt, dass die Schüler bei der Programmierung dieses Algorithmus wenige Probleme haben, sofern man das Vorwissen abklärt und die Modellierung ausführlich mit den Schülern durchspricht.

9. Variabelenspezifische Probleme

9.1. Lokale und globale Variablen

In diesem Kapitel möchte ich mit den Schülern den Unterschied zwischen lokalen und globalen Variablen besprechen.
Bei der Deklaration einer Variablen in einer beliebigen imperativen Programmiersprache ist der Ort wichtig, an dem man dies durchführt. Der Einsatz dieser beiden Variablentypen muss wohlüberlegt sein, denn dies stellt eine der häufigsten Fehlerquellen neben Konzeptionsfehlern in der Programmierung dar. Der Nachteil ist meist, dass diese Fehler sehr schwer auszumachen sind, wenn sie erst einmal durchgeführt wurden. Variablen nehmen Werte an, die man sich nur sehr schwer erklären kann.
Globale Variablen werden am Anfang eines Programms deklariert und sind für den gesamten folgenden Programmcode gültig. Lokale Variablen kann man in JavaScript innerhalb von Funktionen deklarieren und gelten ausschließlich in dieser Funktion. Auf die Sonderfälle, dass Funktionen weitere Funktionen in ihrem Rumpf enthalten, muss man ebenfalls mit Schülern eingehen. Da es aber wenige solcher Beispiele innerhalb dieser Unterrichtsreihe gibt, möchte ich dies nicht zu ausführlich besprechen. Es genügt also an dieser Stelle, dass man die Schüler auf diese Problematik hinweist.

9.1.1. Eingabeparameter einer Funktion als lokale Variablen

Eine sinnvolle Vorgehensweise zur Einführung dieser Problematik wäre eine Diskussion über die Variableninhalte während des Ablaufs eines Programms, in der globale und lokale Variablen enthalten sind.
Beispiel:

```
<html>
  <head>
    <title>globale und lokale Variablen</title>
  </head>

  <script language="JavaScript">
  var Zahl = 4;
  function meineFunktion(ZahlInFunktion)
  {
    document.write("Zahl in der Funktion: "+ ZahlInFunktion +"<p>");
     ZahlInFunktion  += 2;
    document.write("Zahl in der Funktion2: "+ ZahlInFunktion +"<p>");
  }
  document.write("Zahl direkt nach seiner Initialisierung: "+Zahl+"<p>");
```

```
  meineFunktion(Zahl);
  document.write("Zahl nach dem Zurückkehren aus der Funktion: "+Zahl+"<p>");
  </script>
</html>
```

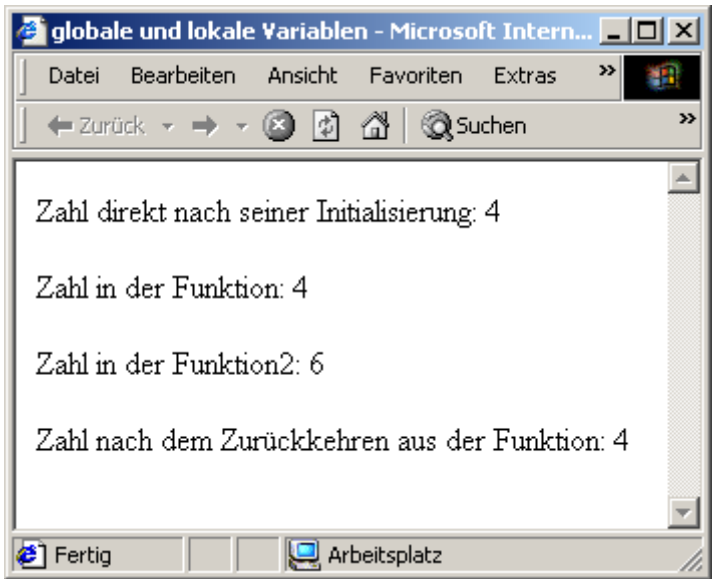


Abbildung 7: Eingabeparameter einer Funktion als lokale Variablen

An diesem einfachen Beispiel kann man sehen, das die Veränderungen, die an der Variable in der Funktion vorgenommen werden, nur in der Funktion gelten. Außerhalb der Funktion hat die Variable Zahl immer noch den Wert 4. Das heißt, dass die Veränderungen in der Funktion nicht nach Außen wirken, und somit lokal sind. Der Eingabeparameter einer Funktion wird nur innerhalb der Funktion benötigt bzw. ist veränderbar und außerhalb dieser nicht existent.

9.1.2. Globale Variablen innerhalb von Funktionen

Modifiziert man das Programm nun leicht, indem man die Variable Zahl zusätzlich direkt und nicht als Übergabeparameter in der Funktion verändert, so wird man erkennen, dass bei einer solchen Veränderung der Inhalt der Variable global, also außerhalb der Funktion ebenfalls noch von Bedeutung ist.

```
<html>
  <head>
    <title>globale und lokale Variablen</title>
  </head>
  <script language="JavaScript">
  var Zahl = 4;
  function meineFunktion(ZahlInFunktion)
  {
    document.write("Zahl in der Funktion: "+ZahlInFunktion+"<p>");
    ZahlInFunktion+=2;
```

```
  Zahl += 5;                    // Veränderung der globalen Variable Zahl
                                // innerhalb der Funktion
document.write("Zahl in der Funktion2: "+ZahlInFunktion+"<p>");
 }
 document.write("Zahl direkt nach seiner Initialisierung: "+Zahl+"<p>");
 meineFunktion(Zahl);
 document.write("Zahl nach dem Zurückkehren aus der Funktion: "+Zahl+"<p>");
 </script>
</html>
```

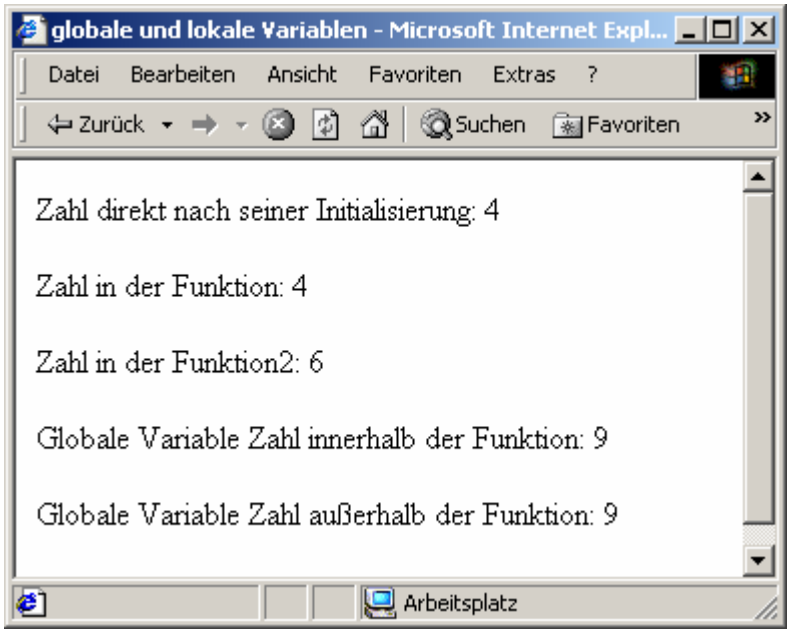


Abbildung 8: Globale Variablen innerhalb von Funktionen

Nach dem Verlassen der Funktion behält die globale Variable den Wert bei.

9.1.3. Lokale Variablen außerhalb des definierten Bereichs

Verwendet man eine in einer Funktion definierte Variable außerhalb der Funktion, so wird diese dort als undefiniert angesehen, weil nach der Beendigung des Funktionsdurchlaufs die Variable gelöscht wird. Aus diesem Grund wird vom Browser der unten eingeblendete Fehler angezeigt. Innerhalb der Funktion kann man diese Variable aber ganz normal verwenden und ausgeben.

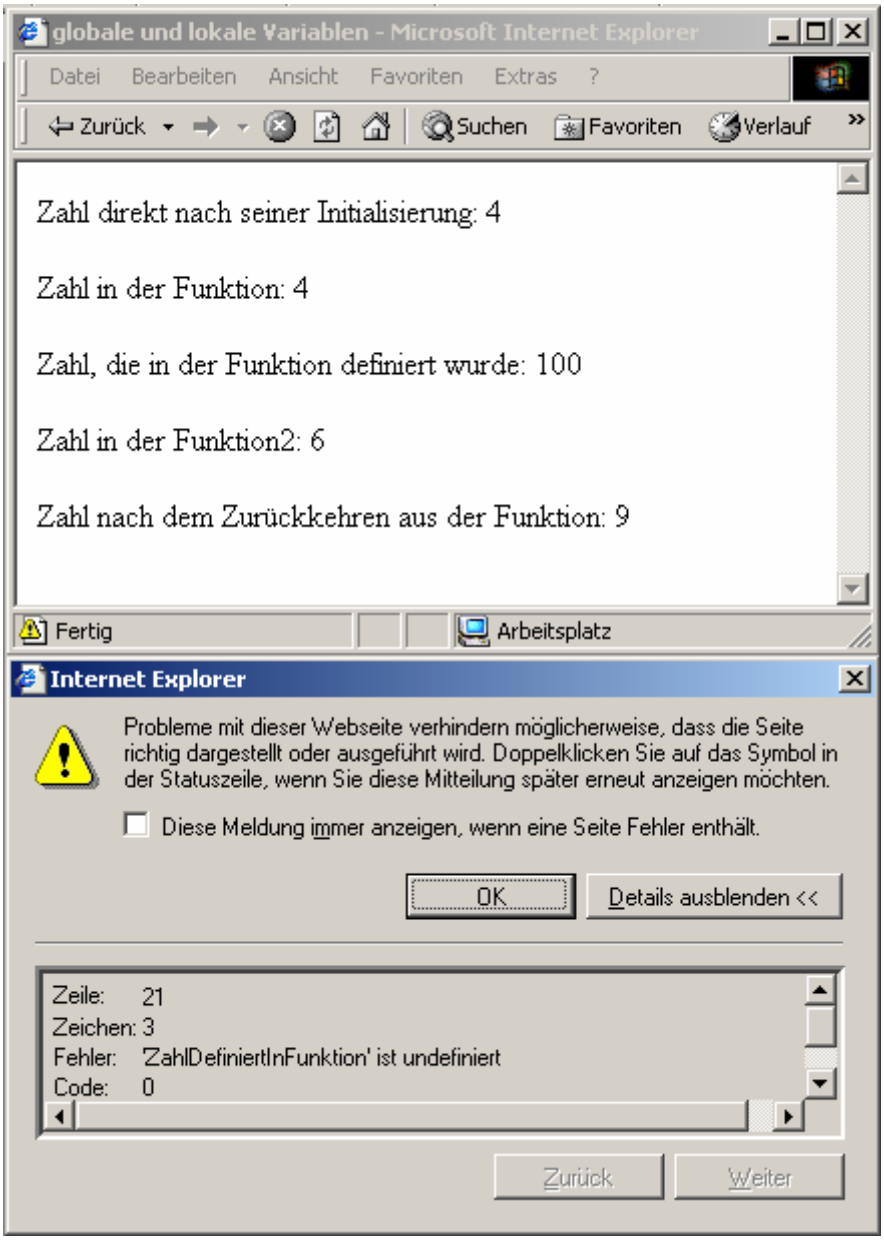


Abbildung 9: Lokale Variablen außerhalb des definierten Bereichs

```
<html>
  <head>
    <title>globale und lokale Variablen</title>
  </head>

  <script language="JavaScript">
  var Zahl = 4;
  function meineFunktion(ZahlInFunktion)
  {
    var ZahlDefiniertInFunktion = 100;     // Veränderung des letzten Quelltextes
    document.write("Zahl in der Funktion: "+ZahlInFunktion+"<p>");
    document.write("Zahl, die in der Funktion definiert wurde:
"+ZahlDefiniertInFunktion+"<p>");
                                           // Veränderung des letzten Quelltextes
    ZahlInFunktion+=2;
    Zahl += 5;
```

```
  document.write("Zahl in der Funktion2: "+ZahlInFunktion+"<p>");
 }
 document.write("Zahl direkt nach seiner Initialisierung: "+Zahl+"<p>");
 meineFunktion(Zahl);
 document.write("Zahl nach dem Zurückkehren aus der Funktion: "+Zahl+"<p>");
 document.write("Zahl, die in der Funktion definiert wurde: "+ZahlDefiniertInFunktion+"<p>");
                                        // Veränderung des letzten Quelltextes
                                        //  In dieser Zeile tritt der Fehler auf, die
                                        //   Variable ZahlDefiniertInFunktion verliert
                                        //   außerhalb der Funktion ihre Gültigkeit

 </script>
</html>
```

Zu diesen 3 verschiedenen Fehlerquellen sollte man nach jeder einzelnen Betrachtung den Schülern einfache Übungsaufgaben geben, damit sie den in diesen Moment noch einfach erscheinenden Sachverhalt in diesen Beispielen noch einmal selbst nachvollziehen können (siehe Anhang Seite 93).

9.2. Überladen einer Variable

9.2.1. Überladen einer Variable durch einen gleichnamigen Eingabeparameter der Funktion

```
<html>
 <head>
  <title>globale und lokale Variablen</title>
 </head>

 <script language="JavaScript">

 var Zahl = 4;
 function meineFunktion(Zahl)            // Variable Zahl wird überladen
 {
  document.write("Zahl in der Funktion: "+Zahl+"<p>");
  Zahl += 2;
  document.write("Zahl in der Funktion2: "+Zahl+"<p>");
  }                                      // Überladene Variable verlässt
                                         // Gültigkeitsbereich und nimmt
                                         // ursprünglichen Wert an
 document.write("Zahl direkt nach seiner Initialisierung: "+Zahl+"<p>");
 meineFunktion(Zahl);
 document.write("Zahl nach dem Zurückkehren aus der Funktion: "+Zahl+"<p>");
 </script>
</html>
```

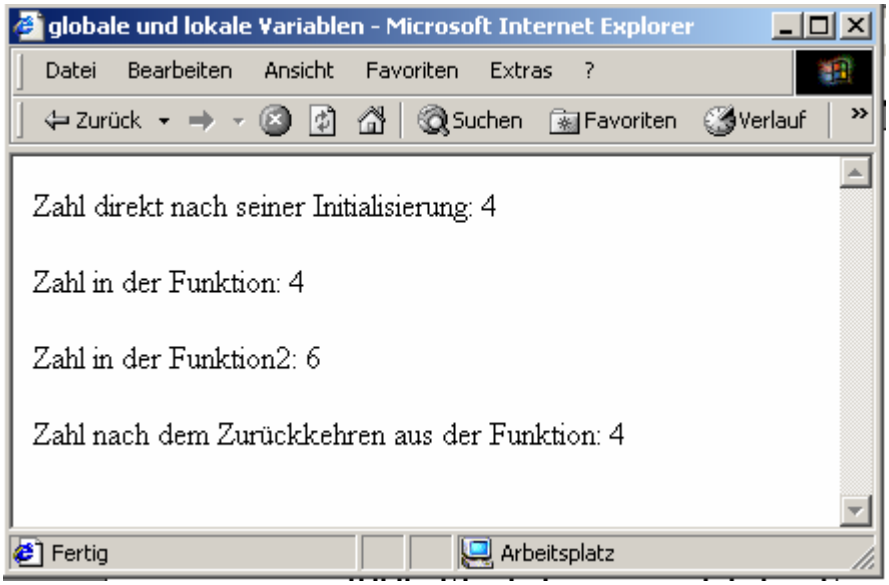


Abbildung 10: Überladen einer Variable

9.2.2. Überladen einer globalen Variable durch gleichnamige Variabelendeklaration innerhalb der Funktion

```
<html>
  <head>
    <title>globale und lokale Variablen</title>
  </head>

  <script language="JavaScript">
  var Zahl = 4;
  function meineFunktion()                    // Kein Eingabeparameter innerhalb der
                                              // Funktion
  {
    var Zahl = 4;                             // Deklaration und Überladung der Variable
                                              // Zahl der Funktion Zahl
    document.write("Zahl in der Funktion: "+Zahl+"<p>");
    Zahl += 2;
    document.write("Zahl in der Funktion2: "+Zahl+"<p>");
  }
  document.write("Zahl direkt nach seiner Initialisierung: "+Zahl+"<p>");
  meineFunktion(Zahl);
  document.write("Zahl nach dem Zurückkehren aus der Funktion: "+Zahl+"<p>");
  </script>
</html>
```

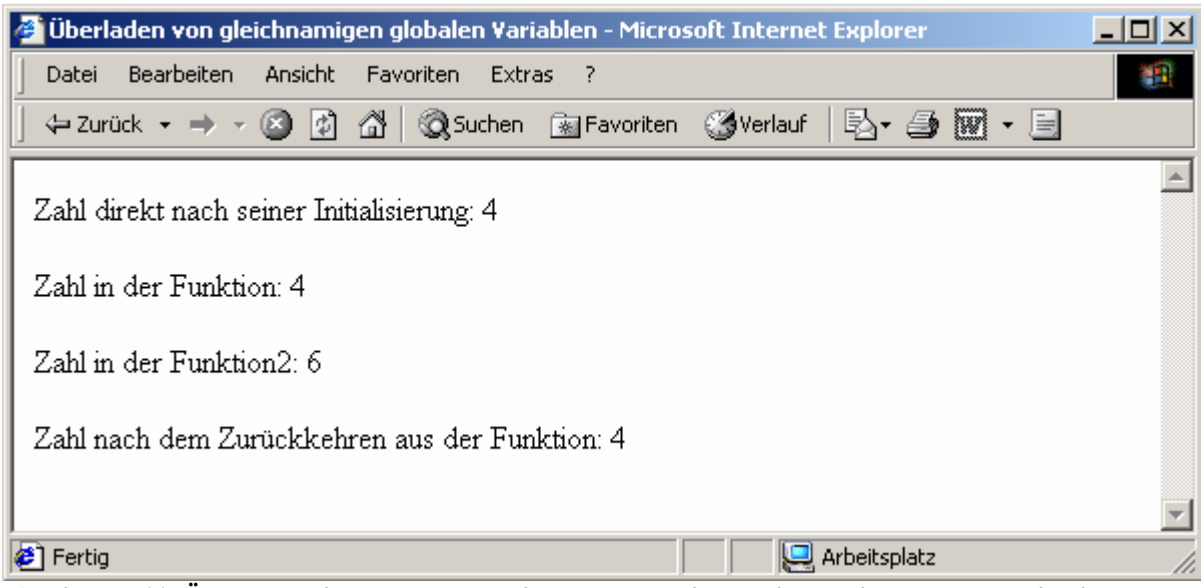


Abbildung 11: Überladen einer globalen Variable durch gleichnamige Variabelendeklaration innerhalb der Funktion

In dieser Darstellung existiert kein Unterschied zum vorherigen Beispiel. Die Variable wird bei der Neudeklaration innerhalb der Funktion überladen, d.h. überschrieben (sie wird zu einer lokalen Variable mit deren Gültigkeitsbereich, nimmt aber nach Beendigung der Funktion ihren ursprünglichen Wert an).

An Hand der 5 unterschiedlichen Beispiele kann man erahnen, worauf man bei der Konzeption und Variabelendeklaration alles achten muss. Dieses Feld bietet vielfältige Möglichkeiten, fehlerhafte Funktionalitäten im Quelltext zu programmieren, die sehr schwer zu orten und zu eliminieren sind.

Das Überladen von Variablen sollte man den Schülern aber generell untersagen, da es die Lesbarkeit des Programmcodes sehr stark beeinflusst, da sich die Fehlersuche bei falschen Variabelenwerten als überaus schwierig erweißt.

10. Taschenrechnererweiterung

Im Kapitel 5 haben die Schüler den Taschenrechner mit Hilfe von 2 Eingabefeldern programmiert. Im weiteren Verlauf der Unterrichtsreihe haben die Schüler weitere Funktionen für den Taschenrechner entwickelt. Ein handelsüblicher Taschenrechner besitzt aber nur ein Anzeigefeld.

Ziel dieses Projekts ist es, den Taschenrechner mit seinen Funktionalitäten auf einen Taschenrechner umzustellen, der nur ein Ein- bzw. Ausgabefeld besitzt.

Die Probleme, die dabei entstehen, sind mit einer geschickten Zwischenspeicherung der Eingaben in Hilfsvariablen zu lösen. Gibt man bei einer Rechnung ein Rechenzeichen ein, so müssen die beiden zuletzt eingegebenen Zahlen mit dem letzten Rechenzeichen ausgewertet werden und als erste Zahl für die mit dem aktuellen Rechenzeichen eingeleitete Rechnung abgespeichert werden.

Diese Funktionalität erfordert ein überlegtes Einsetzen von lokalen und globalen Variablen. Aus diesem Grund eignet sich dieses Projekt zu einer Wiederholung der betrachteten Komponenten der Programmiersprache JavaScript. Durch die Einbindung der schon erarbeiteten Rechenfunktionen in diesen Taschenrechner muss man diese entsprechend des neuen Konzeptes nachvollziehen können und eventuell überarbeiten. Dies erfordert in einem hohen Maße das Wissen der vorherigen Themen.

Zur Erläuterung führe ich die folgenden globalen Variablen und zwei Beispiel-Funktionen an.

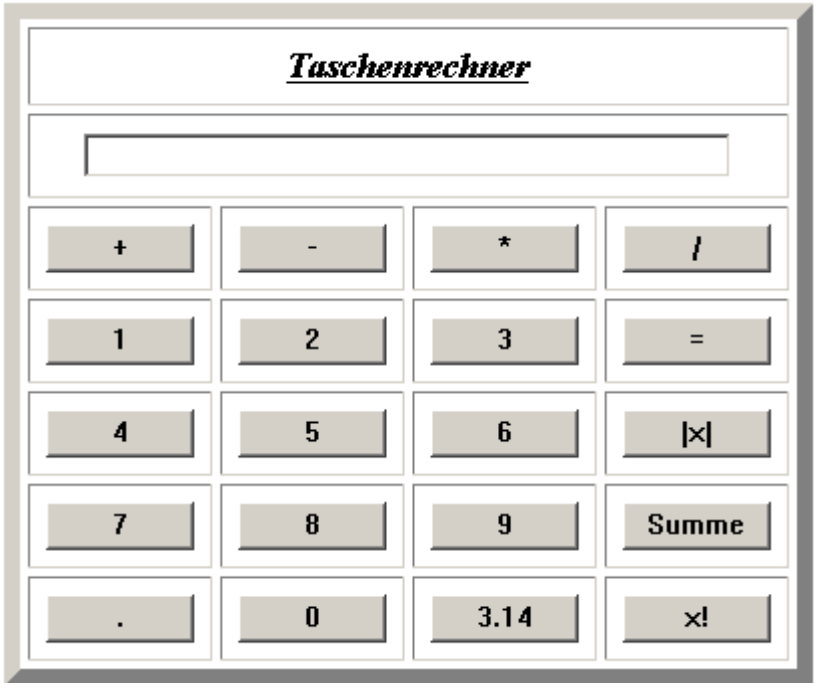


Abbildung 12: Oberfläche des entwickelten Taschenrechners mit einem Eingabefeld

GlobaleVariablen

```
var zahl1,rechenzeichen;
    zahl1 = 0;
    rechenzeichen ="";
```

Erläuterung:

Die Variable Zahl1 wird dazu verwendet, die vorletzte eingegebene Zahl zu speichern. Der Startwert dieser Variablen ist 0, da es keine Zahl gibt, die schon eingegeben wurde.

Die Variable Rechenzeichen speichert die letzte ausgewählte Rechenoperation. Der Startwert dieser Variablen ist „“, da noch keine Rechenoperation zuvor ausgewählt wurde.

```
function Addition()
{
  Auswerten();
  rechenzeichen = "+";
  window.document.Taschenrechner.Display.value ="";
  window.document.Taschenrechner.Display.focus();
}
```

```
function Auswerten()
{
  if (rechenzeichen != "")
  {
   switch(rechenzeichen)
   {
    case "+": zahl1 = zahl1+ Number(window.document.Taschenrechner.Display.value);
    break;
    case "-": zahl1 = zahl1- Number(window.document.Taschenrechner.Display.value);
    break;
    case "*": zahl1 = zahl1* Number(window.document.Taschenrechner.Display.value);
    break;
    case "/": zahl1 = zahl1/ Number(window.document.Taschenrechner.Display.value);
    break;
   }
  }
  else
  {
   zahl1 = Number(window.document.Taschenrechner.Display.value);
  }
}
```

Erläuterung:

Wird der Button Addition gedrückt, wird zunächst die Funktion Auswerten ausgelöst. Diese fragt den aktuellen Wert der Variablen Rechenzeichen ab. Dem entsprechend wird die Rechnung mit der vorletzten Zahl (gespeichert in der Variablen Zahl1) und der im Display angezeigten Zahl ausgewertet und als neue Zahl1 gespeichert. Damit hat die Funktion Auswerten() ihre Schuldigkeit getan. Im weiteren Verlauf der Funktion Addition wird für die neue Rechnung das Rechenzeichen neu gespeichert und die Displayanzeige gelöscht, um sie für die nächste einzugebende Zahl frei zu machen. Zur anwenderfreundlichen Gestaltung wird lediglich der Fokus auf das Display gelegt, um den Weg für die nächste Zahleneingabe zu ebenen.

```
function Gleich()
{
 Auswerten();
 window.document.Taschenrechner.Display.value = zahl1;
 zahl1 = 0;
 rechenzeichen ="";
}
```

Erläuterung:

In der Funktion Gleich() wird die Funktionalität der Gleichtaste definiert. Die letzte Rechnung wird wie oben beschrieben ausgewertet. Anschließend wird das Ergebnis im Display angezeigt und die beteiligten Variablen zahl1 = 0 und rechenzeichen auf „“ gesetzt, da mit dem Gleich eine Art Abschluss der Rechnung erfolgt.

Die Modellierung der Änderung des Taschenrechner und der Einsatz von globalen und lokalen Variablen erweist sich für Schüler als äußerst schwierig. An dieser Stelle ist es aber sinnvoll, den Schülern die Aufgabe zu übergeben, damit sie in der Wiederholungsphase ihre eigenen Erfahrungen machen können. Der Lehrer sollte hier nur als Ratgeber fungieren, der die Schülerideen in durch geschicktes Nachfragen in die richtige Bahn lenkt. Es ist durchaus wünschenswert, wenn die Schüler in kleinen Projektteams arbeiten, da sie dabei lernen, ihre Ideen zu formulieren und den anderen Gruppenmitgliedern zu erläutern. Die mangelnde Kommunikation zwischen Entwicklern ist häufig Ursache für Fehlerquellen in Programmen. Die Absprache und der Informationsaustausch sollte aus diesem Grund geübt werden. Auf die Komponenten Kommunikations- und Teamfähigkeit wird deshalb im Lehrplan ausdrücklich hingewiesen[3].

Ein weiter Grund spricht an dieser Stelle für eine Gruppenarbeit. Es wäre möglich, dass aus den unterschiedlichen Gruppen mehrere Konzepte entstehen, die an dieser Stelle miteinander

[3] Lehrplan Informatik (gymnasialer Bildungsgang) Seite 3

in einer Plenumphase erörtert und analysiert werden könnten (Das gesamte Programm befindet sich auf der CD unter „Taschenrechner 3“).

11. Rekursion

Betrachtet man sich die letzten beiden Beispiele, die im Bereich Schleifen behandelt wurden, dass Heron-Verfahren und der Finanzierungsplan, so erkennt man, dass es sich hier um rein iterative Algorithmen handelt. Iterative Algorithmen erkennt man daran, dass einzelne Schritte solange wiederholt werden, bis eine bestimmt Bedingung erfüllt ist. Man erkennt sie auch meist daran, dass Schleifen bei der Abarbeitung in Algorithmen verwendet werden.
Im Gegensatz dazu stehen rekursive Verfahren. Bei diesem Verfahren wird der Algorithmus immer solange von sich selbst aufgerufen, d.h. selbst verwendet, bis eine Abbruchbedingung erreicht ist.

<u>Zitat:</u> Informatik für die Sekundarstufe II Bd.2
(Rüdeger Baumann, Klett-Verlag 1993)
Rekursion ist Wiederholung durch Schachtelung, Iteration ist Wiederholung durch Aneinanderreihung. Eine rekursive Prozedur (bzw. Funktion) übergibt die Steuerung des Programmablaufs bei rekursivem Aufruf an die gerufene Prozedur. Nach Erledigung ihrer Aufgabe gibt diese – zusammen mit ihren Ergebnissen – die Ablaufsteuerung an die rufende Prozedur zurück, die sodann mit der Ausführung der hinter dem Aufruf kommenden Anweisungen fortfährt.

Im Verlauf der 11. Klasse könnte man sich eigentlich auf die iterativen Algorithmen beschränken, da keine sich selbst reproduzierenden Datenstrukturen behandelt werden. In dem ersten Halbjahr der 12. Klasse werden Datenstrukturen besprochen, die sich immer wieder selbst reproduzieren. Dies sind Listen, Bäume, Stack und Schlangen. Dort hat man immer gleichartige Elemente, die an die gegebene Datenstruktur angehängt werden. Bei diesen Datenstrukturen stellt die Rekursion meist die eleganteste Programmiertechnik dar, mit der man auf ihnen arbeiten kann bzw. überhaupt eine Bearbeitung möglich ist.
Ich halte es aber durchaus für sinnvoll, die beiden unterschiedlichen Programmiertechniken den Schülern schon in der 11. Klasse vorzustellen, um den Weg für die 12. Klasse zu ebnen. Bei der Rekursion handelt es sich um eine Technik, die einem bei der ersten Betrachtung zunächst einmal sehr fremd erscheint. Die iterative Vorgehensweise ist der menschlichen Denkweise etwas verwandter.
Im Vergleich zur Iteration ist die Rekursion meist sehr zeit- und speicheraufwändig. Aber bei manchen Datentypen eben eleganter bzw. die einzig mögliche Variante, eine bestimmte Funktionalität zu programmieren.

Ein Einstiegsbeispiel in die Rekursion ist die Summe der Zahlen von 1 bis n, bzw. die Berechnung der Fakultät einer Zahl. Diese Beispiele sind im Verlauf der Unterrichtsreihe schon einmal besprochen und iterativ programmiert worden.

Im folgenden soll es darum gehen, die beiden Algorithmen zu vergleichen und zu bewerten.

Summenberechnung iterativ:

```
function SummeIterativ(n)
{
  var sigma = 0;
  for(var i =1; i<=n;i++)
  {
    sigma = sigma + i;
  };
  window.document.Formular.Summe.value = sigma;
}
```

Summenberechnung rekursiv:

```
function SummeRekursiv(n)
{
 if (n > 0)
 {
   return (n + SummeRekursiv(n-1));
 }
 else
 {
   return 0;
 };
};

function berechne(n)
{
 window.document.anzeige.Loesung.value = SummeRekursiv(n);
}
```

Um die beiden Algorithmen zu vergleichen, erläutere ich die Unterschiede am Beispiel Summe(4):

Iterativ:

	sigma = 0	
-Schleifendurchlauf:	sigma = 0 + 1 = 1	
-Schleifendurchlauf:	sigma = 1 + 2 = 3	
-Schleifendurchlauf:	sigma = 3 + 3 = 6	
-Schleifendurchlauf:	sigma = 6 + 4 = 10	i = 5 > n = 4 => Schleifenabbruch

Ergebnisausgabe: sigma = 10

Rekursiv:

Funktionsaufruf:	SummeRekursiv(4)
1. Funktionsdurchlauf	SummeRekursiv(4) = 4 + SummeRekursiv(3)
2. Funktionsdurchlauf	SummeRekursiv(3) = 3 + SummeRekursiv(2)
3. Funktionsdurchlauf	SummeRekursiv(2) = 2 + SummeRekursiv(1)
4. Funktionsdurchlauf	SummeRekursiv(1) = 1 + SummeRekursiv(0)
5. Funktionsdurchlauf	SummeRekursiv(0) = 0
6. Ersetzung 1	SummeRekursiv(1) = 1 + 0 = 1
7. Ersetzung 2	SummeRekursiv(2) = 2 + 1 = 3
8. Ersetzung 3	SummeRekursiv(3) = 3 + 3 = 6
9. Ersetzung 4	SummeRekursiv(4) = 4 + 6 = 10

Ergebnisausgabe: SummeRekursiv(4) = 10

Vergleicht man die beiden Varianten, so erkennt man, dass das rekursive Verfahren wesentlich mehr Speicherplatz benötigt, da die Ergebnisse der einzelnen Aufrufe immer zwischengespeichert und weitergeben werden müssen. Bei der iterativen Methode benötigt man nur den einen Speicherplatz für die Variable Sigma, die aber 4 mal umdefiniert wird, bevor das Ergebnis angezeigt werden kann.

Den Schülern sollte an dieser Stelle das Verfahren Rekursion lediglich vorgestellt und erläutert werden. Der geschachtelte Aufruf und das Ersetzen auf dem Rückweg sollte an einem Beispiel (wie oben detailliert an SummeRekursiv(4)) besprochen werden, damit den Schülern der Ablauf plausible wird.

Man sollte ebenfalls sehr deutlich machen, dass dieses Verfahren für spätere Zwecke unbedingt erforderlich ist und aus diesem Grund besprochen wird, obwohl es im Moment noch an eleganten einsichtigen Beispielen mangelt.

Diese Vorgehensweise empfehle ich aus Spiralcurriculumsgründen, damit den Schülern in der 12. Klasse die ohnehin schwierige Hürde der Rekursion etwas genommen wird.

Unterschwellig kann man den Schüler hierbei noch zusätzlich mitgeben, dass es bei einer Funktion einen Rückgabewert geben kann, der mit Befehl return zurückgegeben wird. Diese können jedoch nicht direkt an eine Webseite zurückgegeben und eingebunden werden, sondern müssen über eine Zusatzfunktion oder auf anderen Umwegen dorthin weitergegeben werden.

Weitere Beispiel wären an dieser Stelle trotz allem nötig. Die Schüler könnten an dieser Stelle die Fakultät rekursiv programmieren und einzelne Aufrufe als Beispiel theoretisch nachspielen, um die Schachtelung noch einmal selbst nachzuvollziehen.

Als Übung und kleineres Projekt kann man den euklidischen Algorithmus zur Bestimmung des größten gemeinsamen Teiler (ggT) besprechen und von den Schülern umsetzen lassen.

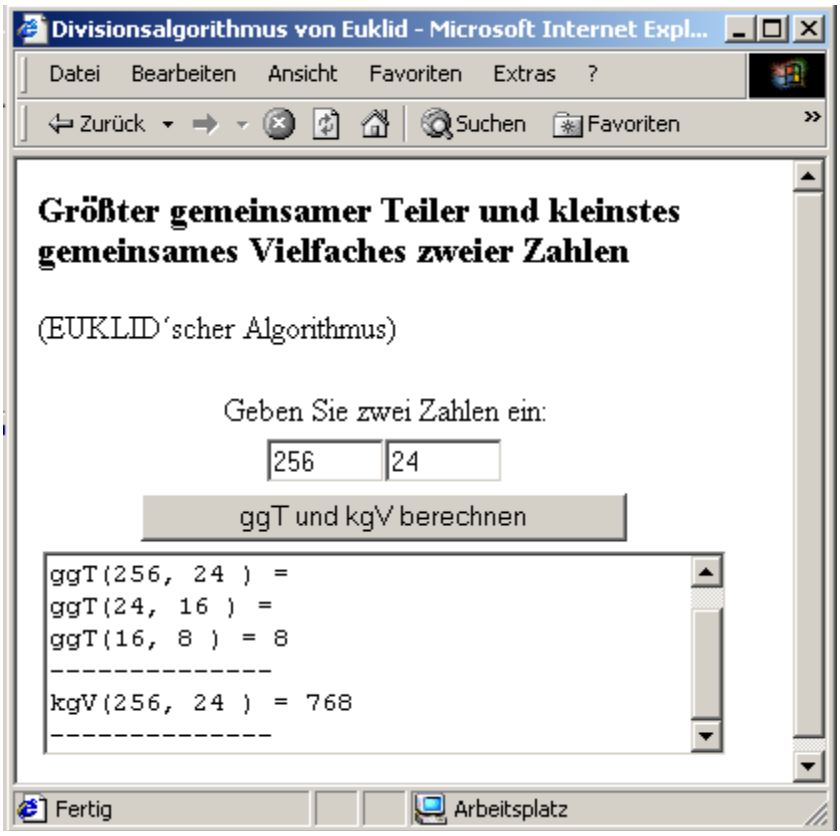
```
<html>
<head>
<title>Divisionsalgorithmus von Euklid</title>
```

```
<script language="JavaScript">

 var text="";

 function euklid(a,b)
 {
  var quotient;
  var rest;
  var h;
  text = text + "\n" + "ggT(" + a +", " + b + " ) = ";
  quotient = Math.floor(a/b);
  rest = a - b*quotient;
  if (rest == 0)
  {
   return(b);
  }
  else
  {
   h = euklid(b,rest);
   return(h);
  }
 }

 function berechne(f)
 {
  text ="";
  a = f.zahl1.value;
  b = f.zahl2.value;
  erg = euklid(a,b);
  text += erg;
  text += "\n" + "--------------";
  text = text + "\n" + "kgV(" + a +", " + b + " ) = " + a*b/erg + "\n";
  text += "--------------";
  f.feld.value = text;
 }
 </script>
 </head>

 <body>
    <h3>Größter gemeinsamer Teiler und kleinstes gemeinsames Vielfaches zweier
Zahlen</h3>
       (EUKLID´scher Algorithmus)
    <form>
      <table border="0">
       <tr>
          <td align="center">Geben Sie zwei Zahlen ein:</td>
       </tr>
       <tr>
          <td align="center"><input type="text" name="zahl1" size="6"><input type="text"
name="zahl2" size="6"></td>
       </tr>
       <tr>
```

```
        <td align="center"><input type="button" value="   ggT und kgV berechnen   "
name="button1" onClick="berechne(this.form)"></td>
      </tr>
      <tr>
        <td align="center"><textarea name="feld" rows="6" cols="40"></textarea></td>
      </tr>
    </table>
  </form>
 </body>
</html>
```

Abbildung 13: Webseite zum Euklidischen Algorithmus

Während dieser kleinen Projekte wiederholen und vertiefen die Schüler permanent die schon



bekannt Programmiertechniken.

12. Arrays

12.1. Allgemeines

In einer Variablen kann man einen beliebigen Wert abspeichern und abfragen. Möchte man aber ein Problem bearbeiten, dass es erfordert, mehrere Variablen des gleichen Typs anzulegen, so ist eine Variable eine recht ungeeignete Methode, dieses Problem zu lösen. Man müsste eine beliebige Anzahl von Variablen deklarieren. Hierzu stellen Programmiersprachen Arrays bereit. Arrays kann man sich als einen Container vorstellen, welcher eine beliebige Anzahl von Variablen gleichen Typs bereitstellt. Um diese Variablen zu deklarieren, muss man nur einen Array anlegen und gegebenenfalls die Länge angeben.
Ein Array kann man sich auch als eine Art Vektor vorstellen. Man ist über einen aufsteigenden Index in der Lage, auf die einzelnen Komponenten des Arrays zuzugreifen und den Inhalt zu erfragen.
Zur Einführung des Arrays ist es erforderlich, bei den Schülern eine Situation zu erzeugen, die ihnen die Notwendigkeit eines solchen Datentyps suggeriert.
Weiterhin spielen Arrays für den Aufbau auf und den Zugriff von Webseiten eine sehr große Rolle. Die einzelnen Komponenten einer Seite werden ebenfalls in Arrays gespeichert.
Der Zugriff auf viele gleichartige Felder kann einmal so gelöst werden, indem man einzeln auf diese mittels Objekthierarchie und Punktnotation zugreift und deren Inhalte komponentenweiße einem Array zuordnet. Ab einer bestimmten Anzahl von Komponenten erweißt sich diese Vorgehensweise aber als sehr Zeilen- intensiv.
Greift man jedoch über einen vom Betriebssystem automatisch angelegten Array auf diese gleichartigen Komponenten zu, so kann man diese mit Hilfe einer For-Schleife abarbeiten.
Ziel dieses Abschnittes ist es, ein Konzept vorzustellen, in dem den Schülern die neue Datenstruktur Array näher gebracht wird und ein vertrautes Arbeiten mit dieser erreicht wird. Weiterhin sollten die Schüler erkennen, dass intern Seitenkomponenten ebenfalls in Arrays abgespeichert werden und so der Zugriff auf Komponenten automatisiert werden kann.
Als Hinweis sei es mir gestattet, dass die Art der Speicherung der einzelnen Seiten-komponenten in objektorientierten Programmiersprachen ebenfalls in Arrays geschieht. Dieses Konzept ist also nicht JavaScript- spezifisch und damit eine Programmiersprachen-übergreifende Methode, die den Schülern auch von Nutzen sein kann, wenn sie später einmal eine andere Programmiersprache erlernen.
In JavaScript wird ein Array als Objekt instantiert. Mit dem New- Operator wird ein neues Array-Objekt ins Leben gerufen.

Zum Array-Objekt gibt es einige Methoden, die von JavaScript bereitgestellt werden und die Arbeit mit diesen erleichtern. An dieser Stelle kann man der objektorientierten Programmierung den Weg ebenen. Es geht aber nicht darum, in der 11. Klasse die Objektorientierung der 12. Klasse vorwegzunehmen, sondern die Schüler an die Instantierung und den Zugriff auf Methoden des Objektes weiter vertraut zu machen.

12.2. Einführung

Zur Einführung des Arrays sollen die Schüler ein Programm schreiben, welches 10 Zahlen der Reihe nach aus Textfeldern einliest und zur weiteren Verarbeitung bereitstellt. Diese Zahlenreihe können die Schüler in umgehrten Reihenfolge ausgegeben lassen.
Die Vorgehensweise sollte zunächst mit den Schülern besprochen werden. Die von den Schülern vorgeschlagene Lösungsvariante wird 10 unabhängige Variablen enthalten, in die der Inhalt der Textfelder eingelesen wird. Die Ausgabe wird ebenfalls 10 Programmzeilen benötigen, in der die Inhalte in umgekehrter Reihenfolge im Textfeld ausgegeben werden.
Zu diesem Zeitpunkt sollte man die Probleme einer Erweiterung des Programms auf die Eingabe von 100 bzw. 1000 Zahlen diskutieren. Den Schülern wird sehr schnell klar werden, dass ihr Lösungsweg für die Erweiterung nur bedingt geeignet ist. Das finden von 100 bzw. 1000 Variabelennamen, dass Einlesen und Ausgeben der Variabeleninhalte würde sich als reichlich unelegant erweisen.
An dieser Stelle erscheint ein Datentyp notwendig, mit der man eine vorgegebene Anzahl von gleichartigen Variablen behandeln kann. Nachdem die Schüler im Laufe der Diskussion die Notwendigkeit eines Arrays erfasst haben, kann man nun die Syntax für die Instantierung eines Arrays angeben und seine Eigenschaften diskutieren.
Die wichtigsten Informationen zu einem Array sollten den Schüler vorgegeben werden:
Deklaration eines Arrays:

var BeispielArray = new Array(Längenangabe)

Die Längenangabe ist nur erforderlich, wenn man im voraus schon genau weiß, wie viele Variablen benötigt werden. Ansonsten wird ein Array mit dynamischer Länge erzeugt

Aufbau eines Array:

BeispielArray

0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***..***	***..***
Inhalt 0	Inhalt 1	Inhalt 2	Inhalt 3	Inhalt 4	Inhalt 5	...	....	....			

Ein Array ist Null-basierend, d.h. das die erste Speicherstelle den Index 0 besitzt

Zugriff auf den Inhalt eines Array:

BeispielArray[i] i steht für den Indexwert des benötigten Feldes

BeispielArray[4] Hiermit erhält man als Inhalt dieser Variablen „Inhalt 4“

Umgang mit den Array-Feldern:

Der Umgang mit den Array-Feldern entspricht dem der Variablen

Array-Objekt:

Die Instantierung bzw. Deklaration eines Arrays wird im Gegensatz zur Deklaration einer Variablen mit dem zusätzlichen new-Operator durchgeführt. Auf den Unterschied, dass es sich um ein Objekt und nicht um mehrere Variablen handelt, sollten die Schüler hingewiesen werden. Den Unterschied kann man an dieser Stelle durch die Verwendung der Array-Objekte- eigenen Methoden erläutern. Bei Variablen kann man mit Hilfe der Punktnotation keine weiteren Funktionen ansprechen.

Für interessierte Schüler kann man noch einen Verweis auf die objektorientierte Programmierung geben. Im Internet stehen genügend gute Hinweis zur kurzen Einführung in die Thematik.

(http://www.oszhdl.be.schule.de/gymnasium/faecher/informatik/oop/index.htm)

Angabe von relevanten Funktionen:

BeispielArray.sort() Sortierung des Array-Inhalts nach lexikographischer Ordnung

BeispielArray.reverse() Umkehrung der Reihenfolge

Auf die Funktionalität dieser und anderer für das Array bereitgestellter Funktionen kann man auf Self-HTML verweisen.

Um den Datentyp Stapel für die 12. Klasse vorzubereiten, könnte man die Funktionen pop() und push() besprechen.

Nach dieser Einführung sollten die Schüler in der Lage sein, die ihnen o.a. Aufgabe zu lösen. Die Schüler werden hierzu die vom Array bereitgestellt Funktion reverse() verwenden.

Als Erweiterungsvorschläge: - Schreiben einer eigenen Funktion reverse()

- Sortierung der Array-Elemente nach der Größe
- Schreiben einer eigenen Funktion sort()

Hier ein Beispielprogramm, welches die Erweiterungsvorschläge beinhaltet:

```
<html>
 <head>
  <title>Array Sortieren</title>
 </head>
 <script language="JavaScript">
 var zahlenfeld = new Array(10);
 var text="";
 function Einlesen(f)
 {
  for(var i=0;i<10;i++)
  {
   zahlenfeld[i]=Number(f.elements[i].value);
  }

  return zahlenfeld;
 }

 function Numsort(a,b)
 {
  return b-a;
 }

 function Sortieren(f)
 {
  var text;
  var Hilfsfeld = new Array(10);
  Hilfsfeld = Einlesen(f);
  Hilfsfeld.sort(Numsort);
  text = AuslesenArray(Hilfsfeld);
  f.Anzeige.value = text;
 }

 function Umkehren(f)
 {
  var text;
  var Hilfsfeld = new Array(10);
  Hilfsfeld = Einlesen(f);
  Hilfsfeld.reverse();
  text = AuslesenArray(Hilfsfeld);
  f.Anzeige.value = text;
 }

 function AuslesenArray(a)
 {
  var text="";
  for(var i= 0;i<=a.length-1;i++)
  {
   text = text + a[i]+ "  ";
  }
```

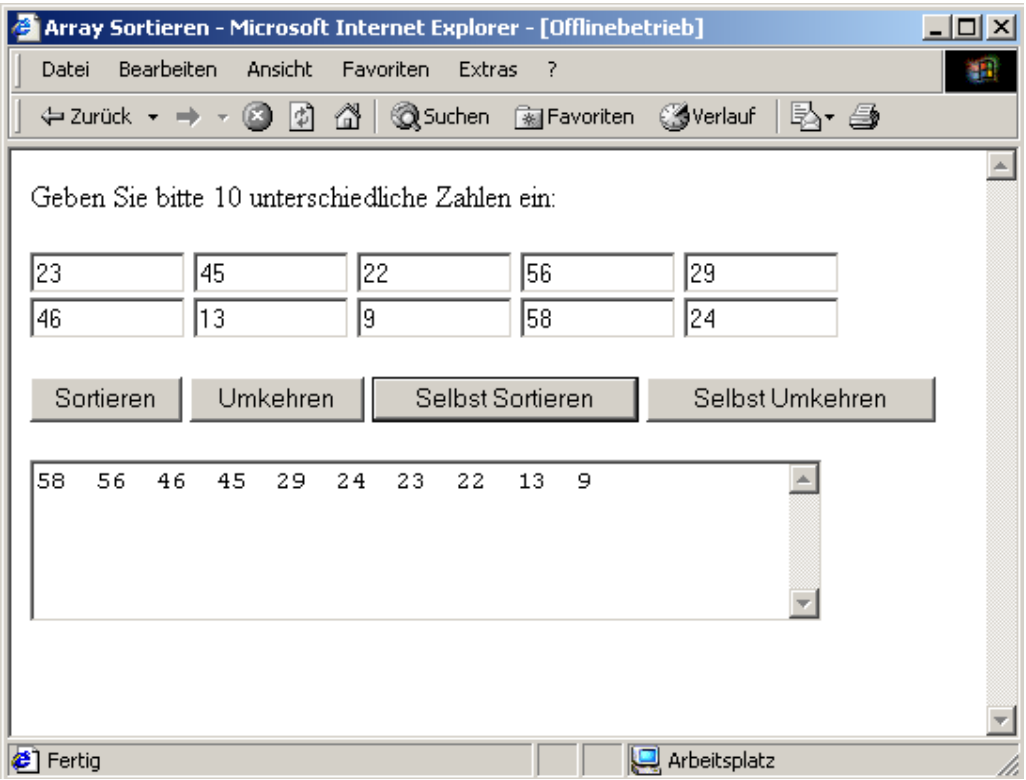
```
  return text;
}

function SelbstSortieren(f)
{
  function MaximumSuche(a)
  {
    var Maxi = 0;
    var Hilfsvariable;
    for(var i = 0; i<= a.length-1;i++)
    {
      if (a[i] > Maxi)
      {
        Maxi = a[i];
        Hilfsvariable = i;
      }
    }
    a[Hilfsvariable]= 0;
    return Maxi;
  }
  var text;
  var Maximum;
  var j = 0;
  var Hilfsfeld = new Array(10);
  var Hilfsfeld1 = new Array(10);
  Hilfsfeld = Einlesen(f);
  for(var i =0; i <= Hilfsfeld.length-1;i++)
  {
    Maximum = MaximumSuche(Hilfsfeld);
    Hilfsfeld1[j] = Maximum;
    j++;
  }
  text = AuslesenArray(Hilfsfeld1);
  f.Anzeige.value = text;
}

function SelbstUmkehren(f)
{
  var text;
  var Hilfsfeld = new Array(10);
  var Hilfsfeld1 = new Array(10);
  Hilfsfeld = Einlesen(f);
  for(var i = 0;i<= Hilfsfeld.length-1;i++)
  {
    Hilfsfeld1[i]=Hilfsfeld[9-i];
  }
  text = AuslesenArray(Hilfsfeld1);
  f.Anzeige.value = text;
}
</script>
<body>
  Geben Sie bitte 10 unterschiedliche Zahlen ein:<p>
  <form Name="Eingabe">
```

```
    <input type="Text" name="Feld0" size="10">
    <input type="Text" name="Feld1" size="10">
    <input type="Text" name="Feld2" size="10">
    <input type="Text" name="Feld3" size="10">
    <input type="Text" name="Feld4" size="10">
    <input type="Text" name="Feld5" size="10">
    <input type="Text" name="Feld6" size="10">
    <input type="Text" name="Feld7" size="10">
    <input type="Text" name="Feld8" size="10">
    <input type="Text" name="Feld9" size="10"><p>
    <input type="Button" value="Sortieren" onClick="Sortieren(this.form)">
    <input type="Button" value="Umkehren" onClick="Umkehren(this.form)">
    <input type="Button" value="Selbst Sortieren" onClick="SelbstSortieren(this.form)">
    <input type="Button" value="Selbst Umkehren" onClick="SelbstUmkehren(this.form)">
    <p><textarea name="Anzeige" cols="50" rows="5"></textarea>
  </form>
 </body>
</html>
```

Abbildung 14: Webseite zum Thema Arrays

Die Funktion Einlesen sollte man im Kontext Arrays noch einmal getrennt besprechen, da dort die Verwaltung der Seitenkomponenten in Arrays abzulesen ist (siehe S. 14 Abb. 2).

Eine Webseite kann aus mehreren Formularen, Formularelementen usw. bestehen. Die einzelnen gleichartigen Elemente werden in Arrays verwaltet und können darüber auch



angesprochen werden.

Das übergebene Formular f enthält in diesem Beispiel 10 Textfelder, 4 Buttons und einen Textbereich. Der Inhalt der ersten 10 definierten Elemente wird dem Array Zahlenfeld mit der For-Schleife zugewiesen. Wäre die Seitenverwaltung nicht so angelegt, könnte man keine

Schleife verwenden, sondern müsste die Felder einzeln „per Hand“ auslesen. Dies wäre auf der Suche nach einer eleganten Lösung ein enormer Rückschritt gewesen.

```
function Einlesen(f)
 {
  for(var i=0;i<10;i++)
  {
   zahlenfeld[i]=Number(f.elements[i].value);
  }

  return zahlenfeld;
 }
```

Die Funktion SelbstSortieren() sollte mit den Schülern im Vorfeld besprochen werden, da die Vorgehensweise nicht unbedingt eindeutig ist. Es gibt sehr viele Sortierverfahren, die sich durch ihre Komplexität aber auch durch den Komplexitätsgrad in der Programmierung unterscheiden. Dies ist aber Gegenstand des Unterrichts der 12. Klasse, dem ich an dieser Stelle nicht vorgreifen möchte. Der einfachste Sortieralgorithmus, den die Schüler mit ihrem Wissensstand ohne Probleme umsetzen können ist eine Art von SelectionSort. Man sucht das Array nach dem Maximum ab, entnimmt das Maximum und schreibt es in einen Hilfsarray. Nun setzt man den Inhalt des Array-Feldes, in dem das Maximum befand, auf 0. Führt man nun wiederum die Maximumsuche erneut durch, so erhält man die zweitgrößte Zahl usw.. Auf diese Weise kann man nun das vorgegebene Array im Hilfsfeld der Größe nach einfügen und anschließend sortiert ausgeben.
In diesem Zuge sollte man noch einmal auf die Modularisierung des Programms hinweisen. Schreibt man für jede Funktionalität eine eigene Funktion, so wird das Programm übersichtlicher, lesbarer und damit wartbarer.

Ein weiterer Anwendungsbereich für Arrays sind die Vektoren. Die Schüler haben Vektoren schon im Physik-Unterricht der 9. bzw. 10. Klasse im Bereich der Mechanik kennen gelernt. Die vektorielle Darstellung der Kraft kann auf die Angabe von Weg, Geschwindigkeit und Beschleunigung übertragen werden. Dies ist ebenfalls Thema der Physik der Klasse 11.
Aufgrund dieser Vorgaben kann man das für die Schüler aktuelle Thema der Physik in einem Programm aufgreifen, indem die Grundrechenarten der Vektorrechnung behandelt werden. Die Eingabe von zwei Vektoren kann mit Hilfe von 6 Variablen erledigt werden. Die Addition und Subtraktion von Vektoren im dreidimensionalen könnten noch mit zwei zusätzlichen Vektoren á 3 Komponenten für die Ergebnisvektoren gelöst werden. Insgesamt sind also 12 Variablen nötig, bzw. 13 wenn man das Skalarprodukt hinzunimmt. Verwendet man aber für einen Vektor jeweils einen Array, so wird die Handhabung einfacher gestaltet.

Ein Programm, mit dem man die Grundrechenarten der Vektorrechnung automatisieren kann, könnte man den Schülern als Hausaufgabe zur Festigung geben (Siehe CD Programm „Vektoren“).

13. DHTML und Ebenenerzeugung

Ein weit verbreitetes Problem bei der Programmierung von dynamischen Webseiten sind die unterschiedlichen Versionen von JavaScript in den verschiedenen Browsern. Um Cross-Browser-fähige Seiten (dynamische Webseiten, die in den verschiedenen Browsern funktionsfähig sind) zu programmieren, sollte man die unterschiedlichen zugrundeliegenden Objektmodelle von Jscript (IE) und JavaScript (Netscape) im Bezug auf Layer kennen.
Unter Layern kann man sich Ebenen vorstellen, die man beliebig auf einer Webseite positionieren kann. Hierbei hat man die Möglichkeit während der Laufzeit des Programmes, diese Position zu verändern und somit eine Bewegung dieser Ebene zu erzeugen. Diese Art der Programmierung wird auch unter dem Begriff DHTML (Dynamic Hypertext Markup Language) zusammengefasst.
Eine Ebene wird in den Tag *** *** Ebenenbeschreibung*** *** eingebettet. Die Positionierung kann über das Attribut style vorgenommen werden. Das Ansprechen der verschiedenen Eigenschaften des Layer-Objektes kann aus der nachfolgenden Aufstellung der Objektmodelle und aus den entsprechenden Referenzen [3] entnommen werden.

Objektmodell der Browser Internet Explorer und Netscape Navigator im Bezug auf Layer

Netscape Navigator ab Version 4

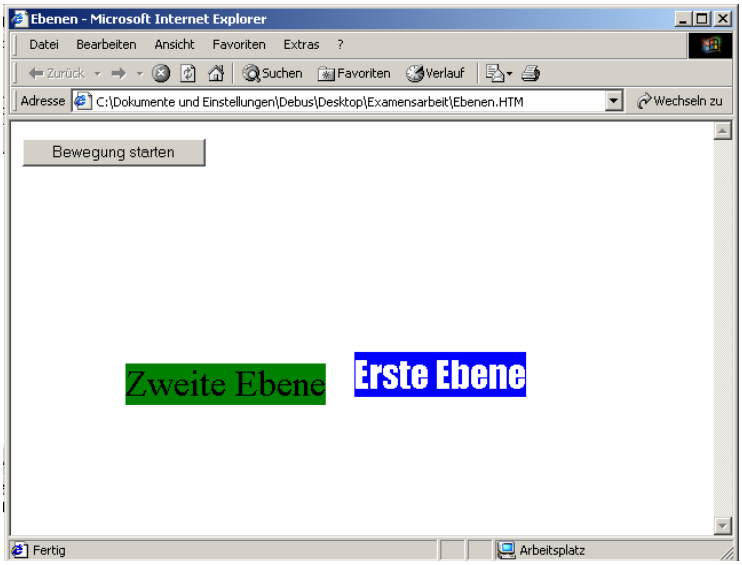
- document.
  - layer.
    - moveTo(x,y)
    - left
    - top
    - .....

Internet Explorer ab Version 4

- document.
  - All.
    - elementID.
      - style.
      - left
      - top
      - .....

Ein Beispiel zu bewegten Ebenen:
<html>

```
<head>
  <title>Ebenen</title>
</head>
<script language="JavaScript">
var Links=100;
var Differenz=10;
function move()
{
  if (document.layers)
  {
    document.ebene2.left=Links+Differenz;
  }
  else
  {
    ebene2.style.left=Links+Differenz;
  }
    Links=Links+Differenz;

  if(Links>600) Links =100;
  setTimeout ("move()",50);
}
</script>
<body>
 <div id="ebene1" style="background-color: blue; position:absolute; font-family:Impact;left:300; top:200; font-size:24pt; color:white">
   Erste Ebene
 </div>
 <div id="ebene2" style="background-color: green; position:absolute; left:100; top:210;width:200pt; font-size:24pt; color:black">
   Zweite Ebene
 </div>
 <input type="Button" value="Bewegung starten" onclick="move()">
</body>
</html>
```

Abbildung 15: Webseite 
 zu Ebenen

In der Funktion move() wird während der ersten if-Abfrage in der Bedingung abgefragt, ob document.layers den Wert

true liefert. Dies wird nur von dem Netscape Navigator mit true beantwortet. So kann man indirekt abfragen, mit welchem Browser die Seite angeschaut wird.
Die Ebenen werden im Body der Seite definiert und über die Funktion move() die Eigenschaft document.ebene2.left bzw. ebene2.style.left bei jedem weiteren Aufruf um den in der Variable Differenz festgelegten Wert nach rechts verschoben. Erreicht die Eigenschaft left einen Wert von über 600, so springt der Wert für diese automatisch auf 100 und die Bewegung beginnt wieder aufs neue.
In den Layer können auch Bilder bzw. Links eingebunden werden, so dass man mit diesem Konzept bis auf die unterschiedlichen Objektmodelle der Browser sehr flexibel ist.
Im Prinzip muss man bei DHTML-Seiten für jeden Browser eine Webseite programmieren, da diese in den verschiedenen Browsern unterschiedlich interpretiert werden. Dies hört sich nach einer sehr ineffizienten Lösung für dynamische Webseiten an. Leider gibt es keine andere Möglichkeit, als durch eine Browser-Abfrage den Browsertyp zu erfragen. Mit einer bedingten Abfrage wird entsprechend dem Browsertyp die für den diesen richtige Seite angezeigt.
Diese umständliche Variante kann man umgehen, indem man Webseiten so programmiert, dass sie nur die Sprachkonstrukte aus der Schnittmenge der unterschiedlichen JavaScript- bzw. JScript-Typen verwenden. Diese Schnittmenge ist aber zu klein, um eine anspruchsvolle dynamische Webseite zu entwickeln, also keine wirkliche Alternative. Weiterhin würden diese Seiten keine Ebenen enthalten.
Über das Objekt Navigator kann man über die Eigenschaft navigator.appName den Browsertyp ebenfalls abfragen. So ist man in der Lage, den erweiterten Funktionsumfang der einzelnen Versionsnummern von Browsertypen gezielt zu nutzen.

Objektmodell des Navigator- Objekts

navigator.

- appName
- appVersion
- language
- platform

Da zum Abschluss der Unterrichtsreihe ein gemeinsames Projekt Kurs-Homepage durchgeführt werden soll, kann man mit den Schülern an dieser Stelle weitere Komponenten einer Homepage entwickeln. Für eine Website ist es wünschenswert, eine benutzerfreundliche und intuitive Navigation zu erstellen. Hierzu eignen sich Menüs, die in mehreren Ebenen

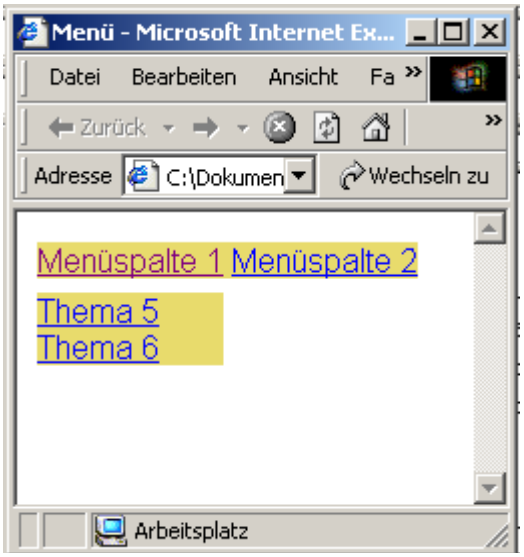
gestaffelt sind. Dabei ist es wichtig, nur den Teil einzublenden, den der User sehen möchte. Die überflüssigen Teile des Menüs sollten dabei ausgeblendet sein.

Pull-Down- Menüs haben den Vorteil, dass man mit ihnen den Inhalt einer Website sehr gut gliedern und zugänglich machen kann. Aus diesem Grund habe ich mich entschieden, eine einfache Variante mit den Schülern zu besprechen, welche im weiteren Verlauf des Unterrichts weiterentwickelt werden soll.

```
<html>
 <head>
  <title>Menü</title>
 </head>

 <script language="JavaScript">

 var openmenu="menu1";
 if(document.layers)
 {
  window.captureEvents(Event.MOUSEDOWN);
  window.onmousedown=ausblenden;
 }
 else
 {
  document.onmousedown=ausblenden;
 }

 function einblenden(param)
 {
  openmenu=param;
  if (document.layers)
  {
   document.layers[openmenu].visibility ="show";
  }
  else
  {
   document.all[openmenu].style.visibility ="visible";
  }
 }

 function ausblenden()
 {
  if (document.layers)
  {
   document.layers[openmenu].visibility="hide";
  }
  else
  {
   document.all[openmenu].style.visibility="hidden";
  }
 }
```

```
  </script>
  <body>
    <div id="menu1" style="background-color:rgb(232,219,108); font-family:Arial;  width:93
px; position: absolute; visibility: hidden; left: 10px; top: 40px">
      <a href="Seite5.htm">Thema 5</a> <br>
      <a href="Seite6.htm">Thema 6</a> <br>
    </div>

    <div id="menu2" style="background-color:rgb(232,219,108); font-family:Arial; width:93
px; position: absolute; visibility: hidden; left: 105px; top: 40px">
      <a href="Seite1.htm">Thema 1</a> <br>
      <a href="Seite2.htm">Thema 2</a> <br>
      <a href="Seite3.htm">Thema 3</a> <br>
      <a href="Seite4.htm">Thema 4</a> <br>
    </div>

    <div style="background-color:rgb(232,219,108); font-family:Arial; width:190 px">
      <a href="javascript:einblenden('menu1')">Menüspalte 1</a>
      <a href="javascript:einblenden('menu2')">Menüspalte 2</a>
    </div>
  </body>
</html>
```

Abbildung 16: Pulldownmenü



14. Projekt Roulette

Nach diesen eher theoretischen Themen sollte eine ausgedehntere Praxisphase folgen, in der die gelernten Sachverhalte Verwendung finden.
Um die Vergleichbarkeit von Programmen zu gewährleisten, ist es von Vorteil, wenn alle Schüler die gleiche Aufgabe erledigen. Auf der anderen Seite steigert es natürlich die Motivation der Schüler, wenn sie sich mit eigenen Projekten beschäftigen.
Ich habe mich an dieser Stelle aus folgenden Gründen für eine gemeinsame Fragestellung zum folgenden Projekt entschieden:

- Vergleichbarkeit
- Gemeinsame von Schülern entwickelte Bewertungskriterien
- Förderung der Teamarbeit
- Expertenbildung

Da die ersten beiden Punkte schon im Kapitel bei dem Projekt Würfelspiel ausgeführt wurden, möchte ich kurz etwas zu den letzten beiden Punkten anmerken. Teilt man den Kurs in kleinere Projektteams, so fördert dies die Kompetenzen im Bereich der allseits geforderten Teamfähigkeit. Diese würde man aber auch erreichen, wenn die einzelnen Projektgruppen unterschiedliche Arbeitsaufträge hätten. Meine Beobachtungen haben gezeigt, dass Schüler bzw. die einzelnen Projektgruppen gerne ihre bisherigen Ergebnisse vor den anderen in Gruppengesprächen demonstrieren, um Selbstbestätigung von den anderen zu bekommen. Sie geizen auch nicht unbedingt mit ihrem Wissen und geben dies gerne an andere weiter. So profitiert der gesamte Kurs von dem „Expertenwissen“ einzelner. Der Experte sollte sein Wissen den anderen also ruhig preisgeben dürfen. Es geht hier nicht darum, den Quellcode des anderen zu übernehmen, sondern um den Anreiz, Ansporn und den Impuls, der von solchen Demonstrationen ausgeht. Nach dem Motto „dass möchten wir auch in unserem Projekt einbringen“, „wenn der das kann, dann können wir das auch“, „ich wusste gar nicht, dass so etwas möglich ist“ usw.. Anhand des abgegebenen Programms kann man bei der Bewertung schon erkennen, ob jemand Teile des Quellcodes eines anderen bei sich eingefügt hat. Hat jemand einen veränderten Quellcode in seinem eigenen Programm eingebaut, setzt dies meist ein tieferes Verständnis des Sachverhaltes voraus, welches ja das eigentliche Ziel des Unterrichts sein sollte.
Der direkte Vergleich zwischen den einzelnen Projekten initiiert meist kleinere Konkurrenzkämpfe im Bezug auf das „beste“ Programm. Dies steigert die Verbundenheit und den Stolz auf das eigene Programm und damit wirkt sich dies direkt positiv auf die Motivation

der Schüler aus. Es sollte sich aber auch die aufgewendete Zeit neben der Qualität des Programms in der Benotung positiv widerspiegeln.

Ziel dieses Projekts sollte die Umsetzung eines Roulettes sein. Die Einführung in diese Projektarbeit habe ich mit einem realen Roulettespiel durchgeführt. Während des Umgangs mit dem Spiel haben die Schüler die Spielregeln besprochen, Spielsysteme erfunden und ausprobiert. Dazu habe ich den Schülern eine kleine Einführung in die Wahrscheinlichkeitstheorie gegeben, welche aber nicht unbedingt zur Umsetzung notwendig war.

Dieses Projekt bietet folgende Möglichkeiten der Umsetzung:

- Modellierung und Umsetzung den Spielregeln
- Modellierung und Umsetzung eines Spiels
- Kontoführung (mehrfacher Durchlauf des Spiels)
- Statistik
- Graphische Ausgabe des Spiels
- Setzen per Maus
- usw.

Für eine ausreichende Leistung sollte mindestens die Simulation eines Spieldurchlaufs mit Setzen, Zufallszahl und Auswertung unter Angabe der Gewinnsumme erwartet werden.

Eine Maximallösung könnte wie folgt aussehen:

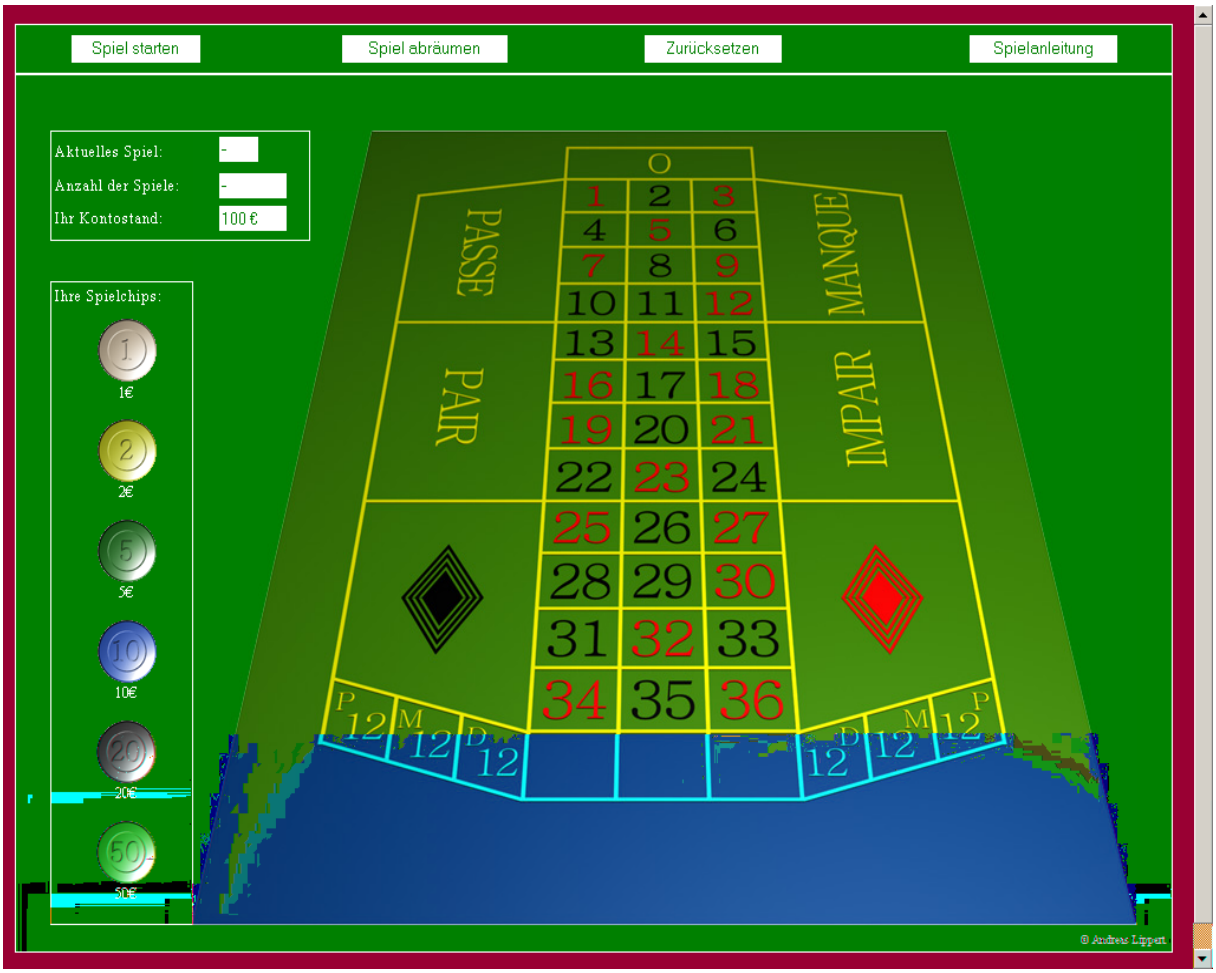


Abbildung 17: Oberfläche des Roulette-Spiels

Diese Lösung beinhaltet die Durchführung eines kompletten Roulette-Spiels mit dem Setzen per Drag & Drop auf die entsprechenden Felder, einem Intro, die Darstellung der Spielregeln, eine Kontoführung mit Limitüberschreitungssperrung, das Abräumen der Spielfläche und das „Stehenlassen“ des Einsatzes.

Das Programm ist auf der CD unter dem Titel „Beispiel“ zum Thema Roulette zu finden.

15. Homepage-Projekt

Es wäre wünschenswert, zum Abschluss des Informatik-Kurses eine Wiederholung des gelernten Stoffes des Schuljahres zu erreichen. Eine solche Intention darf man natürlich nicht so offensichtlicht bei den Schülern ohne Motivationsverlust äußern.
Schließt man aber das Schuljahr mit einem Homepage-Projekt, in dem sich der gesamte Kurs darstellen kann, so erreicht man das angestrebte Ziel auf einem unkonventionellen Weg. Darüber hinaus schult man bei den Schülern zusätzlich die Kompetenz Teamfähigkeit, Selbständigkeit und Organisation.
Das erste Halbjahr der 11. Klasse habe ich in Informatik ebenfalls mit einem Homepage-Projekt abgeschlossen. Hier sollten aber nur die Kompetenzen in HTML abgetestet werden. Außerdem sollte jeder einzelne Schüler sein eigenes Projekt gestalten.
Die Erweiterung des neuen Homepages-Projektes steckt in der Programmierung der dynamischen Seiten und der Projektarbeit.
Die Projektarbeit sollte im Informatik-Unterricht einen sehr hohen Stellenwert einnehmen, da größere Software-Projekt nur in größeren Team realisierbar sind. Aus diesem Grund ist es sehr wichtig, dass Informatik-Schüler eines Orientierungskurses und späteren Leistungskurses in der Lage sind, eine solches Projekt vorzubereiten, zu koordinieren, zu konzipieren und durchzuführen. Dies sind wesentliche Kompetenzen, um ein Softwareprojekt erfolgreich abzuschließen. Wenn man davon ausgeht, dass in alleine im Staat Kalifornien 40% der Management Informationssysteme Entwicklungsprojekte vor Abschluss abgebrochen werden [K.Ewusi-Menach: Critical Issues in Abandoned Information Systems Development Projects; Communications of the ACM, Vol.40, No9.9.1997], dann ist es mehr als notwendig, diese Kompetenzen schon in der Schule zu vermitteln.
Bei der Planung der Homepage sollte man den Schülern möglichst viel Freiraum einräumen, um die Verbundenheit mit dem Projekt zu erhöhen und die Motivation zu steigern.
Obwohl das Projekt aus Zeitmangel nicht fertiggestellt wurde, kann man schon erkennen, dass die gelernten Methoden und ganzen Programmierkenntnisse des Schuljahres und darüber hinaus eingeflossen sind.

Einstiegsseite der Homepage:

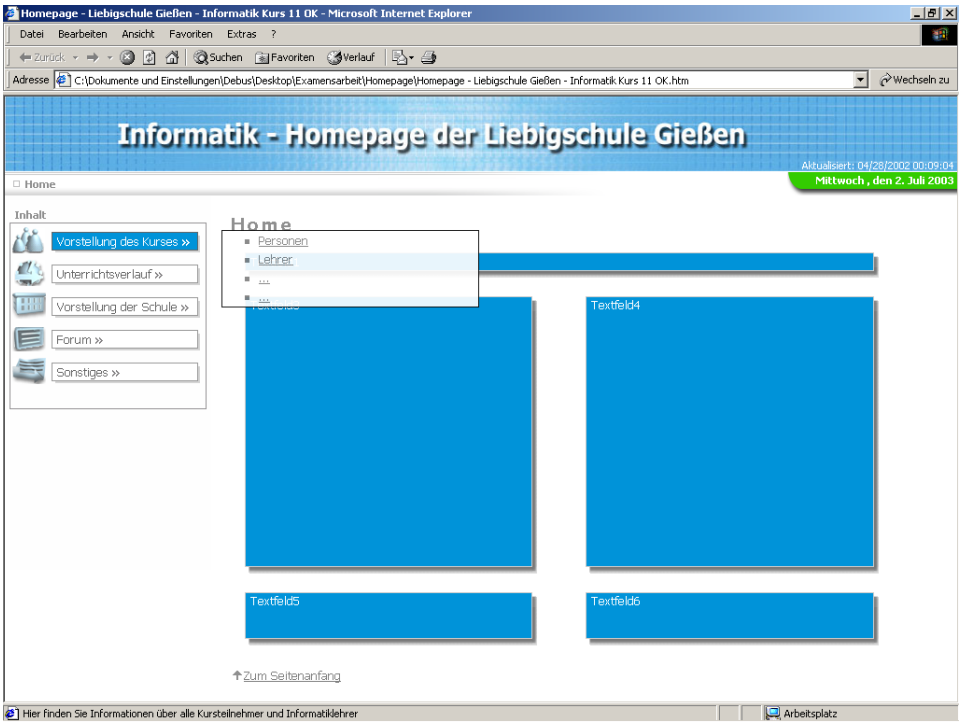


Abbildung18: Startseite der entwickelten Homepage

Sitemap:

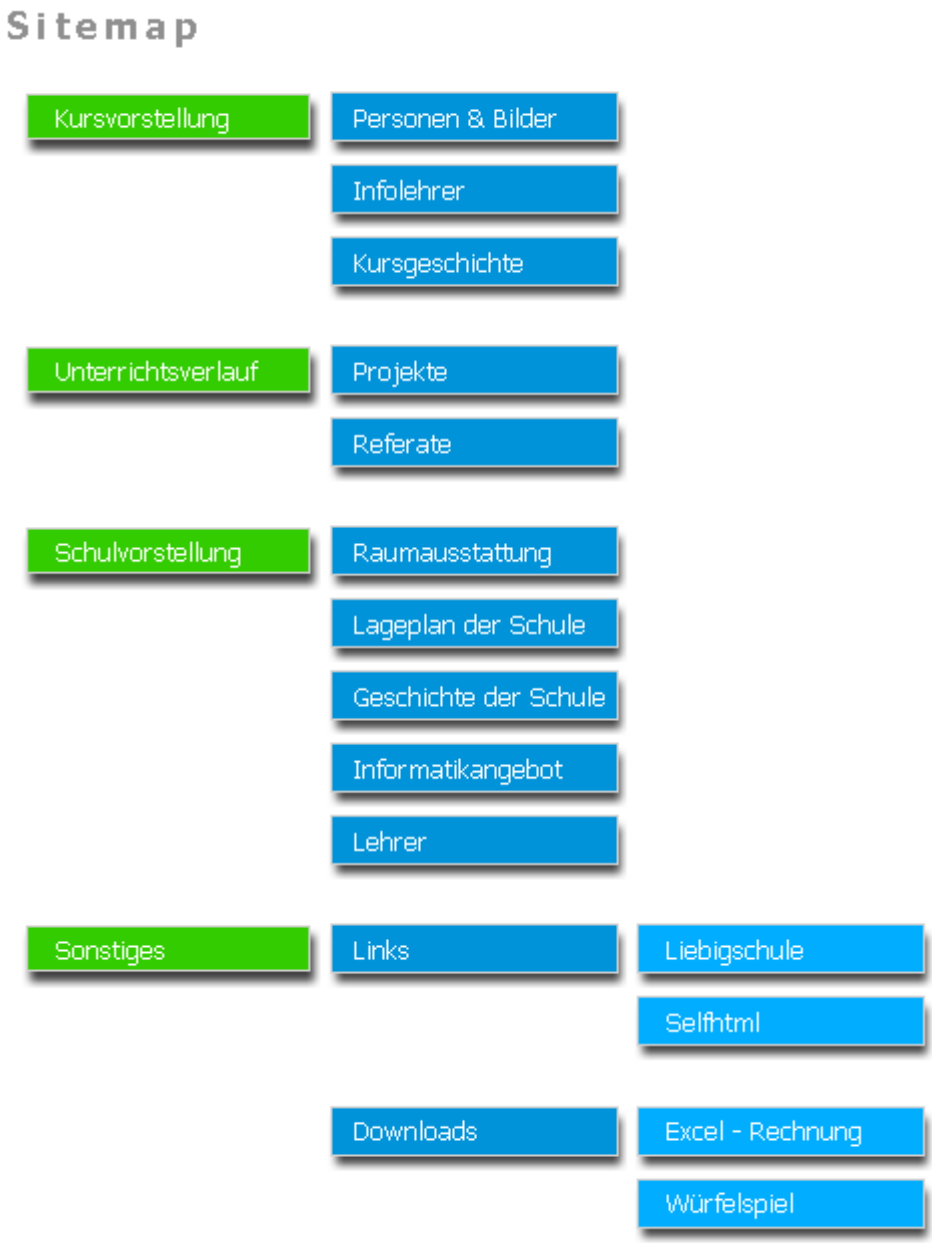


Abbildung 19: Sitemap der entwickelten Homepage

An dieser Stelle möchte ich ihnen nicht zu viel verraten. Schauen sie doch einfach in die Homepage, es lohnt sich. Die Homepage wird mit der folgenden Seite: _index.html gestartet (Auf der CD findet man die Start-Datei im Menü-Punkt Homepage).

Es handelt sich hier um eine Flashanimation, zu dem man ein Plugin der Firma Shockwave benötigt.

16. Literaturverzeichnis

[1] Lehrplan Informatik - Gymnasialer Bildungsgang (Hessisches Kultusministerium 2003)

[2] JavaScript – Eine Einführung (RRZN 2. veränderte Auflage, September 2001)

[3] JavaScript (BHV, Michael Seeboerger-Weichselbaum, 2. Auflage, 2000)

[4] HTML 4 (BHV, Uwe Hess, Günther Karl, 2. Auflage, 2000)

[5] JavaScript – Praxisbuch (Franzis, Gamperl/Nefzger, 1999)

[6] Software Engineering I (Vorlesungsscript zu Software Engineering der Universität Hagen, Prof. Dr. H.-W. Six, 2002)

[7] Informatik für die Sekundarstufe II, Band 2 (Klett, Rüdiger Baumann, 1. Auflage, 1994)

[8] Informatik – Vom Problem zum Algorithmus (Diesterweg, Wolf-Dieter Haaß, 1986)

[9] Informatik heute – Algorithmen und Datenstrukturen, Band 1 (Schroedel, Schöningh, 1987)

[10] Informatik heute – Algorithmen und Datenstrukturen, Band 2 (Schroedel, Schöningh, 1988)

[11] Unzählige Webseiten, die immer wieder aktualisiert werden und die neuesten Entwicklungen der Sprache aufzeigen. Ich gebe aus Aktualitätsgründen keine konkrete Adresse an.

Anhang:

Inhaltsverzeichnis zum Anhang

Anhang:
1. Handout für Schüler zur Einführung in JavaScript

Einführung in Javascript

1. Warum verwendet man eine Skriptsprache im HTML-Code

JavaScript ist eine kleine, einfach zu erlernende Skriptsprache. Sie wurde entwickelt, um Web-Dokumente interaktiv zu gestalten.
Vorteile:
- Man benötigt für JavaScript keine Programmierumgebung, ein normaler Texteditor reicht im Normalfall
- Es reichen meist wenige Programmierzeilen, um dass gewünschte Ergebnis zu erzielen
- Man ist nicht an ein bestimmtes Betriebssystem gebunden
- Relativ leistungsschwach => relativ sicher, da keine tiefergehenden Eingriffe in ein System möglich sind

Nachteile:
- Der Code ist von jedermann einsehbar
- Eigene geniale Ideen können von anderen übernommen werden

2. Integration von Javascript in dem HTML-Code

Die Integration von JavaScript im HTML-Code wird über den Tag
<script langugage="JavaScript"> « Scriptcode »**</Script>** ermöglicht.
Man kann diesen Code an einer beliebigen Stelle im HTML-Code einbinden. Wegen der Übersichtlichkeit sollte man aber darauf achten, den Script-Code außerhalb des Bodies, dem eigentlichen Programmier-Bereich der HTML-Seite, abzulegen. Der gängige Bereich dafür liegt zwischen Head und Body. Eine weitere Möglichkeit ist die Ablage des Script-Codes in einer externen Textdatei (z.B. Scriptdateiname.js)

3. Einführung in Objektorientierung von Programmiersprachen

JavaScript besitzt als einfache Skriptsprache natürlich nicht die Komplexität und Vielfalt „ausgewachsener Programmiersprachen". Trotzdem gibt es neben den unzähligen Besonderheiten auch Gemeinsamkeiten. Wie alle modernen Programmiersprachen arbeitet JavaScript „objektorientiert". In JavaScript können die einzelnen Bestandteile des Browserfensters als Objekt angesprochen werden.

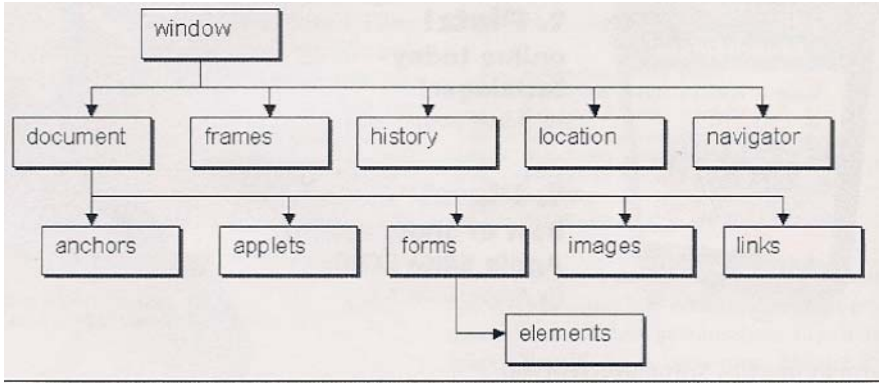
a. Objekte

Als Objekt bezeichnet man also die einzelnen Bestandteile des Browserfensters. Hier gibt es zum Beispiel das Browserfenster selbst. Es handelt sich hier um das window-Objekt. Formulare wiederum werden als form-Objekt bezeichnet.

b. Objekthierarchie

Diese Objekte können nicht unabhängig voneinander betrachtet werden. Die einzelnen Objekte hängen voneinander ab und können aus diesem Grund

hierarchisch angeordnet werden. In der folgenden Abbildung wird diese Hierarchie für das window-Objekt angedeutet.
Um nun mit dem JavaScript-Code ein Formular direkt bearbeiteten zu können, muss man es über diese Struktur ansprechen. Dies geschieht mit dem folgenden Ausdruck: window.document.forms
Indem man also von der Wurzel des Hierarchiebaumes alle Knoten aufzählt (natürlich nur die, die auf dem Weg bis zum gewünschten Objekt liegen) und sie durch einen Punkt trennt, ist man in der Lage, jeden beliebigen Zweig des Baumes anzusprechen, um ihn anschließend zu bearbeiten.



c. Methoden

Hat man nun die einzelnen Objekte durch die Folge im Hierarchiebaum angesprochen, ist man nun in der Lage, mit vordefinierten und selbstentwickelten Funktionen die einzelnen Objekte zu verändern.
Eine solche Funktion ist zum Beispiel die Alert-Funktion. Die Alert-Funktion ruft im window-Objekt ein Fenster auf, in dem der Text angezeigt wird, den man im Argument der Funktion eingegeben hat.
window.alert(„Dieser Text wird nun in einer Dialogbox angezeigt“).

d. Eigenschaften

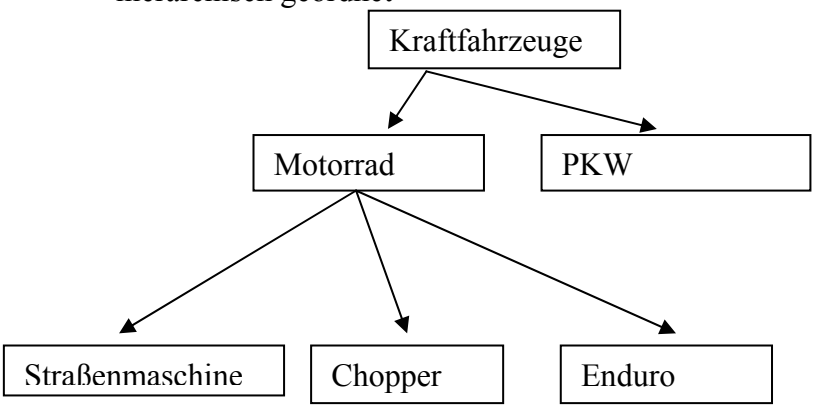
Ein beliebiges Objekt besitzt Eigenschaften. So besitzt das window.document-Objekt die Eigenschaften Hintergrundfarbe, Schrift-Farbe, -Größe, -Art usw..
Diese Eigenschaften können wiederum über die Folge im Hierarchiebaum verändert werden.
window.document.bgcolor=“red“ ändert die aktuelle Dokument-Hintergrundfarbe in rot.

Beispiel zur Objektorientierung:

Die Welt besteht aus einzelnen Objekten. Diese stehen zueinander in Beziehung, kommunizieren miteinander und bilden dadurch eine größere Einheit, die wiederum mit anderen Einheiten in Beziehung stehen usw. ....
Programmiert man objektorientiert, abstrahiert man einfache Daten und arbeitet wie in der realen Welt mit Objekten. Diese Vorgehensweise entspricht viel stärker dem intuitiven Denken und Problemlösen der menschlichen Natur, auch wenn es für Programmierer strukturierter Software als viel komplizierter erscheint.
Objekte sind Elemente, die in einem Anwendungsbereich von Bedeutung sind. Sie haben bestimmte Eigenschaften, enthalten Informationen und wissen, wie

diese verwendet werden. Objekte mit gleichem oder ähnlichen Verhalten können zusammengefasst werden in sogenannte Klassen.

Klassen beschreiben eine Menge nahezu gleicher Objekte. Sie sind hierarchisch geordnet



4. Variablen

a. Sinn von Variablen

Wie alle Programmiersprachen arbeitet auch JavaScript mit Variablen. Variablen kann man sich wie Schubladen eines Schrankes vorstellen. In diesen Schubladen können verschiedene Informationen abgelegt werden. Benötigt man die Informationen aus der ersten Schublade in der zweiten Reihe von unten (mit dieser Ortsangabe weiß jeder, der vor dem Schrank steht, welche Schublade gemeint ist), dann öffnet man diese und schaut hinein und liest die dort abgelegten Informationen.
Bei einem Computer hat man sehr viele Speicherplätze frei. Diese Speicherplätze entsprechen den Schubladen unseres Beispiels. Diesen Speicherplätzen kann man beliebige Namen geben.
Für die Vergabe von Speicherplatznamen gibt es verschiedene Regeln:

- Es dürfen keine Sonderzeichen, Punkte oder Leerzeichen verwendet werden
- Es dürfen Buchstaben, Unterstriche und Zahlen Verwendung finden, das erste Zeichen darf aber keine Zahl sein
- Variabelennamen sind keysensitiv, man muss also auf Groß- und Kleinschreibung achten. Die Variable _zahl1 unterscheidet sich also von der Variablen _Zahl1

Um nun einer Variablen eine solche Schublade bzw. Speicherbereich zuzuordnen, muss man zunächst dem Computer angeben, wie diese heißen soll, damit man ihn später über diesen Namen ansprechen kann (siehe Ortsangabe der Schublade). Dies nennt man Deklaration einer Variablen. Der Computer merkt sich dann, wo dieser Speicherplatz zu finden ist und unter welchem Namen man ihn ansprechen kann.
Die Deklaration einer Variablen wird mit dem Befehl „var“ eingeleitet. Es gibt nun zwei verschiedene Methoden:

- Man legt eine Variable fest, ohne ihr einen Wert zuzuordnen
- Man legt eine Variable fest und weißt ihr direkt einen Wert zu

Beispiel:	var i	var name	var wahrheitswert
	var i = 5	var name =”Müller”	var wahrheitswert=true

Anhang:

Wichtig: Zur strukturierten übersichtlichen Programmierung gehört es, dass man die Variabelenbezeichnungen nach ihrer späteren Verwendung auswählt. Möchte man in der Webseite zwei Zahlen eingeben, die addiert werden sollen, so ist es sinnvoll, folgenden Variablen und Namen zu deklarieren:

```
var Zahl1, Zahl2, Ergebnis;
Ergebnis = Zahl1 + Zahl2;
```

Vorsicht: Ergebnis = zahl1 + zahl2 würde nicht das gewünschte Ergebnis liefern, da es sich bei Zahl1 und zahl1 um zwei verschiedene Variablen handelt.

b. Datentypen

In höheren Programmiersprachen gibt es eine große Bandbreite von Datentypen. Man muss dort schon bei der Deklaration einer Variablen festlegen, ob es sich um eine Zahl, String (Buchstabenfolge) oder einen Wahrheitswert handelt. Diese einzelnen 3 Arten von Datentypen sind je nach Programmiersprache noch einmal untergliedert. In JavaScript wird einem die Deklaration in diesem Bezug sehr stark vereinfacht. Man muss bei der Deklaration nicht angeben, ob es sich um eine Zahl oder einen String handelt. Dies wird automatisch erkannt.
var i = „5“ und var i = 5 sind unterschiedlich Deklarationen. Wird der Wert der Variablen in Anführungszeichen angegeben, so erkennt der Interpreter einen Text, welches einem String entspricht. Fehlen die Anführungszeichen, so handelt es sich um eine Zahl (die man für spätere Berechnungen verwenden kann). Weiterhin gibt es die Wahrheitswerte true/false.
Verwendungsmöglichkeiten:
Zahlen: Berechnungen
Text: Verarbeitung von Benutzereingaben
Wahrheitswerte: Überprüfung von Programmabläufen

Beispiel:

```
<html>
  <head>
        <title>Inhalte Ändern</title>
  </head>



  <script language="JavaScript">
    var test = 5;
    document.write(test+"<br>");
    test=test+3
    document.write(test+"<br>");
    test="Müller"
```

Anhang:

```
    document.write(test+"<br>");
    test=true;
    document.write(test+"<br>");
    test=test+2
    document.write(test+"<br>");
    test="12345"
    document.write(test+"<br>");
    test=test+2
    document.write(test+"<br>");
  </script>
</html>
```

Aufgabe:

1.) Überlege Dir, was wird durch die einzelnen Zeilen window.document.write(test+“
“) auf dem Bildschirm ausgegeben wird

2.) Notiere Dir mit eigenen Worten die Ergebnisse aus diesem Beispiel

3.) Überprüfe Deine Vermutungen, indem Du das Programm umsetzt

5. Einfache interaktive Elemente von Javascript

a. window.alert()

window.alert(„Hallo Welt“)

Bei dem Fenster handelt es sich um ein Ausgabefenster (Fenster mit Text und OK-Button) mit dem Text „Hallo Welt“

b. window.prompt()

name = window.prompt(„Geben Sie Ihren Namen ein“)

Bei dem Fenster handelt es sich um ein Eingabefenster, in der man sowohl Ein- als auch Ausgaben darstellen kann

Text:	Geben Sie Ihren Namen ein
Eingabefeld:	Hier soll der Name eingetragen werden, der dann anschließend in der Variable name gespeichert wird und zur Weiterverarbeitung genutzt werden kann

c. window.confirm()

Anhang:

antwort = window.cofirm(„Wollen Sie das Programm wirklich beenden?“)

Bei diesem Fenster handelt es sich um ein Bestätigungsfenster, in der man eine Text und zwei Buttons (OK- und Abbrechen-Taste) unterbringen kann

d. window.open()

window.open(seite1.htm)

Mit dieser Anweisung wird ein neues Browserfenster namens seite1.htm geöffnet

e. window.document.write()

window.document.write(“Dieser Text erscheint in Deinem HTML-Dokument
“)

Er wird so ausgewertet, dass in dem Fenster letztlich der Text erscheint und anschließend ein Zeilenumbruch durch das
-Tag ausgelöst wird

Aufgabe:
Programmiere ein HTML-Dokument, in dem du die Funktionen der einzelnen Objekte ausprobierst

6. Felder (Arrays)

In einer Variablen kann man jederzeit einen beliebigen Wert speichern und diesen wieder abrufen. Nimmt man an, man müsste mehrere Werte speichern, z.B. mehrere Namen. Für diesen Fall steht ein Array zur Verfügung. Unter einem Array versteht man Variablen, welche aus mehreren verketteten Speicherbereichen zum Ablegen von Werten besteht. Statt nun z.B. beim Speichern von 15 Werten ebenso viele Variablen zu deklarieren, kann man auch ein Array zur Aufnahme von 15 Werten verwenden.
“Kundenliste = new Array(14)” definiert einen Array mit 15 zusammenhängenden Variablen.
Die Besonderheit bei einem Array ist, das die erste Variable den Index 0 bekommt. Dadurch verschiebt sich der Index insgesamt um 1.
„Einkaufsliste = new Array(„Milch“, „Zucker“, „Mehl“, „Äpfel“)“ definiert einen Array mit 4 Einträgen.
Mit dem folgenden Code erreicht man die gleiche Wirkung (vielleicht etwas umständlicher)

```
Einkaufsliste = new Array(3);
Einkaufsliste[0]=“Milch“;
Einkaufsliste[1]=“Zucker“;
Einkaufsliste[2]=“Mehl“;
Einkaufsliste[3]=“Äpfel“;
```

Beispiel:

Anhang:

```
<html>
 <head>
        <title>Array ausgeben</title>
 </head>
 <script language="JavaScript">

  Kundenliste=new Array (2);

  function Aufzaehlen()
  {
    for (i=0;i<Kundenliste.length;++i)
    document.write(Kundenliste[i]+"<br>")
  }

  Aufzaehlen();
  Kundenliste[0]="Meier";
  Kundenliste[1]="Otto";
  Kundenliste[2]="Müller";
  Kundenliste[3]="Schmidt";

  Aufzaehlen();

  document.write (Kundenliste.length);

  </script>
</html>
```

Aufgabe:

1.) Was wird durch das Programm auf dem Bildschirm angezeigt?
2.) Was kann man über die bei der Deklaration von Arrays definierten Länge aussagen?
3.) Notiere Dir die Ergebnisse!
4.) Überprüfe Deine Vermutungen.

7. Funktionen

Du hast schon einige der von JavaScript zur Verfügung stehenden Funktionen kennen gelernt. Unter anderem war es die write-Funktion, die in der Lage ist, ihr Argument (meist ein Text) in das aktuelle Objekt zu schreiben.
Man ist aber auch in der Lage, eigene Funktionen zu definieren.

Der prinzipielle Aufbau einer Funktion ist immer gleich:

```
function Funktionsname(Variable1, Variable2,…….. )
{
 Anweisungen
}
```

Durch die Parameter werden Zahlen an die Funktion übergeben, die durch die Funktion weiterverarbeitet werden.

Beispiel aus der Schulmathematik

Anhang:

```
function f_von_x(x)
{
        f_x=3*x+4;
        window.document.write("f("+x+")= "+f_x);
}
```

Funktionsaufruf: f_von_x(4)

Im folgenden wird der Code für zwei Funktionen dargestellt.

Funktion 1

```
<html>
  <head>
        <title>Funktion1</title>
  </head>
  <script language="JavaScript">
  var Ergebnis;

  function Multipliziere (Faktor)
  {
    var Ergebnis=0;
    Ergebnis=Faktor*5;
    document.write (Ergebnis);
  }

  input= window.prompt("Welche Zahl soll mit 5 multipliziert werden?","");

  Multipliziere (input);

  </script>

</html>
```

Funktion 2

```
<html>
  <head>
        <title>Funktion2</title>
  </head>
  <script language="JavaScript">

  var Ergebnis;
  function Multipliziere (Faktor)
  {
     var Ergebnis=0;
     Ergebnis=Faktor*5;
     return Ergebnis;
  }
  Ergebnis=window.prompt("Welche Zahl soll mit 5 multipliziert werden?","");
  Ergebnis=Multipliziere(Ergebnis);
```

Anhang:

```
    document.write(Ergebnis);

  </script>
</html>
```

Aufgabe:

1.) Was machen die einzelnen Programme
2.) Welche Aufgabe übernehmen dabei die Funktionen
3.) Erkläre den Unterschied der beiden Funktionen

8. Bedingte Anweisungen

a. Operatoren

i. Vergleichoperatoren

==	Wert1 gleich Wert2
!=	Wert1 ungleich Wert2
>=	Wert1 größer oder gleich Wert2
<=	Wert kleiner oder gleich Wert2
>	Wert1 größer Wert2
<	Wert1 kleiner Wert2

ii. Logische Operatoren

&&	UND: beide Bedingungen müssen zutreffen
||	ODER: eine der beiden Bedingung müssen zutreffen

iii. Arithmetische Operatoren

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo-Division
++	x++ entspricht x=x+1
--	x-- entspricht x=x-1

b. If-Anweisung

Um Ausführbedingungen für Anweisungen festzulegen benötigt man Verzweigungen. Eine if-else Verzweigung sieht folgendermaßen aus :

```
if (var==10)
{
Anweisung1;
Anweisung2;
```

```
}
else
{
Anweisung3;
Anweisung4;
}
```

In dem Beispiel wird geprüft ob die Variable var den Wert 10 besitzt. Ist dies der Fall wird die Anweisung 1 und 2 ausgeführt, andernfalls nur Anweisung 3 und 4.

Um die Abfragebedingungen miteinander zu verknüpfen benötigt man Operatoren. Würde man zum Beispiel den Operator && (AND) nutzen : if ((var<10)&&(var>0)). Dabei wird geprüft ob die Variable kleiner als 10 und größer als 0 ist und nur dann wird die dahinterliegende Anweisung ausgeführt.

Beispiel:

```
<html>
  <head>
        <title>If</title>
  </head>
 <script language="JavaScript">

  var input
  input = window.prompt("Bitte geben Sie eine Zahl kleiner als 5 ein","");
  if(input  <= 5)
  {
     alert("Die Zahl ist " + input);
  }
  else
  {
   document.write ("Ihre Eingabe ist falsch");
   document.bgColor="ff0000";
  }

</script>
</html>
```

Aufgabe:
1.) Schreibe ein Programm, in dem Du über ein Formular das Alter des Users abfragst und ermittle mit Hilfe einer If-Anweisung, ob er älter als 20 ist. Ist er älter, dann soll ein Ausgabefenster mit dem Text „alter Sack“ erscheinen, wenn nicht, erscheint der Text „junger Sack“.

c. Switch-Anweisung

Eine Switch-Verzweigung wird genutzt, falls eine Variable mehrere Werte annehmen und man keine umständliche If-Verzweigungen nutzen möchte. Diese Art der Verzweigung sieht grundsätzlich so aus:

Anhang:

```
switch (var)
{
 case "1":
 Anweisung1;
 Anweisung2;
 break;
 case "2":
 Anweisung3;
 Anweisung4;
 break;
 case "3":
 Anweisung5;
 Anweisung6;
 default:
 Standardanweisung;
}
```

In dem Beispiel wird überprüft ob die Variable var den Wert 1,2 oder 3 annimmt und je nachdem werden den die Anweisungszweige ausgeführt. Falls keine Übereinstimmungen eintreten wird der Anweisungsbereich ausgeführt der nach "default" steht.

Beispiel:

```
<html>
 <head>
        <title>Switch</title>
 </head>
 <script language="JavaScript">

 input= window.prompt("Geben Sie eine Zahl zwischen 1 und 3 ein","");
 switch(input)
 {
        case "1":
        alert("Ihre Eingabe war die Zahl "+input);
        break;
        case "2":
        alert("Immer schön in der Mitte bleiben");
        break;
        case "3":
        alert("Mehr geht leider nicht");
        break;
         default:
        alert ("Sie halten sich nicht an die Regeln");
 }

 </script>
</html>
```

Aufgabe:

Anhang:

Überlege Dir eine Anwendungsmöglichkeit für eine Switch-Anweisung und setze sie in einem Webseite um!

9. Schleifen

a. While-Schleife

Diese Art der Schleife wird solange durchlaufen bis die Ausstiegsbedingung erfüllt ist.

```
while (var < 10)
{
  Anweisung1;
  Anweisung2;
}
```

Die Schleife wird solange durchlaufen, bis die Variable var einen Wert größer gleich 10 aufweist.

Beispiel:

```
<html>
  <head>
        <title>While</title>
  </head>
  <script language="JavaScript">

  var i=0;
  while (i<=5)
  {
    document.writeln ("<br>i="+i);
    i++;
  }
  document.write ("<br>Die Schleife wurde beendet");

  </script>
</html>
```

b. Do-While-Schleife

Diese Schleife unterscheidet sich von der while-Schleife lediglich durch den Zeitpunkt der Überprüfung der Bedingung. Diesmal erfolgt die Überprüfung am Ende eines jeden Durchlaufs. Das bedeutet, dass die Schliefe mindestens einmal ausgeführt wird. Damit können Sie z.B. den Benutzer der Seite so lange eine Abfrage durchlaufen lassen, bis ein gewünschtes Ergebnis erreicht wurde.

```
Do
{
  Anweisung, solange die Bedingung erfüllt wird
}
While (Bedingung)
```

Anhang:

Beispiel:

```
<html>
  <head>
          <title>Do</title>
  </head>
  <script language="JavaScript">
  do
  {
    input=window.prompt("Schreiben Sie y für yes","");
  }
  while(input!="y")
  document.write ("<br>Die Schleife wurde beendet");
  </script>
</html>
```

c. For-Schleife

Teilweise ist es notwendig , ein Programm nicht nur einmal ablaufen zu lasen, sondern mehrmals. Aus diesem Grund gibt es Schleifen.

Die For-Schleife wird beispielsweise genutzt, falls die Anzahl der Wiederholungen bereits bekannt ist.

```
for (var v=1;v <= 10;v++)
{
  Anweisung1;
  Anweisung2;
}
```

Innerhalb der Klammern wird erstens die Laufvariable deklariert und zusätzlich der Startwert gesetzt (dieser liegt im Beispiel bei 1), danach wird die Bedingung für die Ausführung definiert (im Beispiel werden die Anweisungen solange ausgeführt bis die Variable v einen Wert größer als 10 annimmt, und als dritte Vereinbarung wird die Wertänderung pro Schleifendurchlauf für die Laufvariable angegeben (im Beispiel wird zu der aktuellen Zahl immer 1 addiert). Insgesamt wird die Schleife also 10 mal durchlaufen.

Beispiel:

```
<html>
  <head>
    <title>For</title>
  </head>
  <script language="JavaScript">

  for (i=10;i>=5;i=i-2)
  {
    document.writeln ("<br>i="+i);
  }
    document.write ("<br>Die Schleife wurde beendet");
```

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

Aufgabe:

1.) In der Wahrscheinlichkeitsrechnung wird sehr häufig die Funktion Fakultät verwendet. Z.B.: 5! = 5*4*3*2*1
n! = n*(n-1)* …….*3*2*1 wobei n aus den natürlichen Zahlen sein soll

Schreibe eine Funktion Fakultät(x), in die Du als Parameter eine natürliche Zahl eingeben kannst, für die dann die Fakultät von x durch die Funktion berechnet wird.
Setze dies um mit der for, while und mit der do-while-Schleife.
2.) Notiere Dir die Unterschiede zwischen den einzelnen Varianten.

d. lokale und globale Variable

Deklariert man eine Variable innerhalb einer Funktion, so ist diese Variable außerhalb der Funktion nicht sichtbar, sie kann also dort nicht aufgerufen werden. Ein solche wird auch lokale Variable genannt, da sie nur lokal (also nur in der Funktion) ihre Gültigkeit besitzt.
Im Gegensatz dazu gibt es globale Variablen, die außerhalb einer Funktion liegen (jedoch innerhalb des Scriptbereichs). Diese können von allen Stellen des Skriptbereichs aufgerufen bzw. verändert werden.

Beispiel:

```
<html>
  <head>
    <title>Globale und lokale Variablen</title>
  </head>
  <script language="JavaScript">

  var Wert_globale_Variable = 5;
  window.document.write("Der Wert der globalen Variable vor dem
Durchlaufen der Funktion ist: "+Wert_globale_Variable+"<br>");

  window.document.write("Hier beginnt die Funktion<br>");

  function Veraenderung()
  {
        var Wert_lokale_Variable = 7;
        window.document.write("Der Wert der lokalen Variable beim
        Durchlaufen der Funktion ist: "+Wert_lokale_Variable+"<br>");
        Wert_globale_Variable = Wert_globale_Variable +
        Wert_lokale_Variable;
        window.document.write("Der Wert der globalen Variable beim
        Durchlaufen der Funktion ist: "+Wert_globale_Variable+"<br>");
  }
  Veraenderung();
  window.document.write("Hier ist die Funktion beendet worden<br>");
  window.document.write("Der Wert der globalen Variable nach dem
Durchlaufen der Funktion ist: "+Wert_globale_Variable+"<br>");
```

Anhang:

```
  window.document.write("Der Wert der lokalen Variable nach dem
Durchlaufen der Funktion ist: "+Wert_lokale_Variable+"<br>");

</script>
</html>
```

Aufgabe:
Gebe die Werte für die globale und lokale Variable vor, während und nach dem durchlaufen der Funktion an und beschreibe, wie diese Werte zustande kommen.

10. Ereignisse (Events)

Mit Event-Handlern hat man die Möglichkeit Funktionen oder Befehle ereignisgesteuert ablaufen zu lassen. Zum Beispiel kann eine Funktion/eine Befehl ausgeführt werden sobald die Seite komplett geladen ist, mit der Maus über einen Link gegangen wird oder wenn ein Element angeklickt wird.

Daran sieht man das Event-Handler äußerst wichtige Javaskript-Elemente sind, da diese die Steuerung übernehmen.

Als Beispiel wäre eine Linkbeschreibung zu nennen, dabei wird bei Überfahren des Links eine kleine Beschreibung in der Statuszeile des Browsers eingeblendet.

```
<a href="link.htm" onMouseOver="window.status='Linkbeschreibung'; return true">
Link</a>
```

Übersicht Event-Handler

In der folgenden Liste findet ihr die wichtigsten Event-Handler um Funktion ereignisgesteuert ablaufen zu lassen.

Name	Beschreibung	Erlaubt in...
onAbort	Abbruch	<img>
OnBlur	Verlassen eines aktiven Bereichs	<body>,<frameset>, <input>,<layer>,<select>, <textarea>
onChange	Änderungen in Formularfeld	<input>,<select>,<textarea>
onClick	Klick	<a>,<area>,<input>,<textarea>
onDblClick	Doppelklick	<a>,<area>,<input>,<textarea>
onError	Fehler	<img>
onFocus	Aktivierung eines Elements	<body>,<frame>,<input>, <layer>, <select>,<textarea>
onLoad	Laden abgeschlossen	<frameset> und <body>
onMouseOver	Überfahren eines Elements (Maus)	<a>, <img>, <area>
onMouseOut	Verlassen des Elements (Maus)	<a>, <img>, <area>

Anhang:

onReset	Zurücksetzen eines Formulars	<form>
onSubmit	Absenden eines Formulars	<form>
onSelect	Selektieren von Text	<input> und <textarea>
onUnLoad	Verlassen der Datei	<frameset> und <body>

2. Übung zum Thema globale und lokale Variablen

Diese Webseite dient ausschließlich zur Verdeutlichung des Kontextes von lokalen und globalen Variablen. Sie ergibt keinen weiteren Sinn

```
<html>
  <head>
    <title>Globale und lokale Variablen</title>
  </head>
  <script language="JavaScript">

  var Wert_globale_Variable = 5;
  window.document.write("Der Wert der globalen Variable vor dem
  Durchlaufen der Funktion ist: "+Wert_globale_Variable+"<br>");
  window.document.write("Hier beginnt die Funktion<br>");

  function Veraenderung()
  {
        var Wert_lokale_Variable = 7;
        window.document.write("Der Wert der lokalen Variable beim
        Durchlaufen der Funktion ist: "+Wert_lokale_Variable+"<br>");
        Wert_globale_Variable = Wert_globale_Variable +
        Wert_lokale_Variable;
        window.document.write("Der Wert der globalen Variable beim
        Durchlaufen der Funktion ist: "+Wert_globale_Variable+"<br>");
  }
  Veraenderung();
  window.document.write("Hier ist die Funktion beendet worden<br>");
  window.document.write("Der Wert der globalen Variable nach dem
  Durchlaufen der Funktion ist: "+Wert_globale_Variable+"<br>");
  window.document.write("Der Wert der lokalen Variable nach dem
  Durchlaufen der Funktion ist: "+Wert_lokale_Variable+"<br>");

</script>

</html>
```

Anhang:

3. Klausur

OK Informatik 11 **Klausur Nr. 1** **16.05.2002**

-Name: _____

Aufgabe 1:

a) Beschreibe die HTML-Seite, indem du den Bildschirm aufzeichnest.
b) Beschreibe die Wirkungsweise der Funktion loese, indem du jede Zeile auskommentierst.
c) Für welches reale Problem stellt dieses Programm eine Lösung dar?

```
<html>
 <script language="JavaScript">
  function loese(f) {
   var d
   d=(f.a.value*f.b.value/100)*(f.c.value/360)
   f.loesung.value=Math.round(d*100)/100 }
 </script>

<body>
 <font size="4">Geben Sie die Zahlen ein:</font size="4"></p>
 <form>
  <table>
   <tr>
     <td>a:<input type="text" size="10" name="a" value></td>
   </tr>
   <tr>
     <td>b:<input type="text" size="10" name="b" value></td>
   </tr>
   <tr>
     <td>c:<input type="text" size="10" name="c" value></td>
   </tr>
  </table>
  <input type="button" name="berechne" value="Berechne" onclick="loese(this.form)"></p>
  <p>
    <font size="4">Das Ergebnis:
      <input type="text" size="10" name="loesung">
    </font size="4">
  </p>
 </form>
</body>
</html>
```

Aufgabe 2: Mit Sieb des Eratosthenes kann man alle Primzahlen aus einem vorgegeben Interval ermitteln. Schreibe ein Programm, mit dem du **nach dem Algorithmus des Eratosthenes** die Primzahlen von 2 bis 20 ermitteln kannst.

Anhang:

Aufgabe 3: Schreibe eine Funktion mit n als Eingabevariable, mit der man folgende Summe bestimmen kann:

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + ... + n^2$$

Verwende in dieser Funktion ein do-while-Schleife für die Berechnung der Summe!

Aufgabe 4: Welche Zahlen werden von dem Programm auf der HTML-Seite ausgegeben? Kommentiere den Programmtext im Bezug auf die Art der Variablen und beschreibe, wie du die Werte im einzelnen bestimmt hast.

```
<html>
  <head>
    <title>Aufgabe 4</title>
    <script language="JavaScript">
<!--
    var a,b,c;
    function funktion1(d)
    {
      var b=5;
      function funktion2(e)
      {
        var h=5;
           b=3+e*h;
        window.document.write('b = '+b+'</p>');
      }

      funktion2(b);
      a=b+d;
      c=d*a;
      window.document.write('a = '+a+'</p>'+'b = '+b+'</p>'+'c = '+c+'</p>');
    }
    a = 2;
    b = 5;
    c = 1;
    funktion1(b);
    window.document.write('a = '+a+'</p>'+'b = '+b+'</p>'+'c = '+c+'</p>');

//-->
    </script>

</head>
<body text="#000000" bgcolor="#FFFFFF" link="#FF0000" alink="#FF0000"
vlink="#FF0000">

</body></html>
```