

Einführung

in die

WEBPROGRAMMIERUNG



[SITEMAP](#) [KONTAKT](#) [IMPRESSUM](#) [ARCHIV](#)



[SCHULE](#) [SCHULLEBEN](#) [FÄCHER](#) [ANGEBOTE](#) [DOWNLOADS](#)



Tradition trifft Moderne!

Wir begrüßen Sie auf der Homepage unserer Schule. Hier erhalten Sie alle Informationen rund um das Hermann-Josef-Kolleg, einen Ort des Glaubens, des Miteinanders und des Lernens, der traditionelle Werte mit den Anforderungen einer modernen Welt verknüpft.

[Startseite](#)

26.01.2015

Bewerbung? So geht's! - Bewerbungstraining für die Q1

Am 23.01.2015 fand das Bewerbungstraining für die Jahrgangsstufe Q1 statt. In verschiedenen Gruppen und unter Anleitung erfahrener Mitarbeiter der AOK, der Barmer GEK und der VR-Bank Nordeifel erarbeiteten die Schülerinnen und Schüler die wichtigsten Aspekte eines Bewerbungsgesprächs. Die drei Unterrichtsstunden begannen mit einer Vorstellungsrunde sowie einleitenden Erfahrungsberichten der freundlichen [...]

[» weiterlesen](#)



[TERMINKALENDER](#)

Fr, 30.01.2015 - 15:00 Uhr
**Anmeldungen für
Grundschüler**



In den nächsten Tagen freuen wir uns auf zahlreiche Grundschülerinnen und Grundschüler, die sich dazu

Teil 2

Clientseitige Programmierung mit JavaScript

HJK 2016

(Version von 30.05.2016)

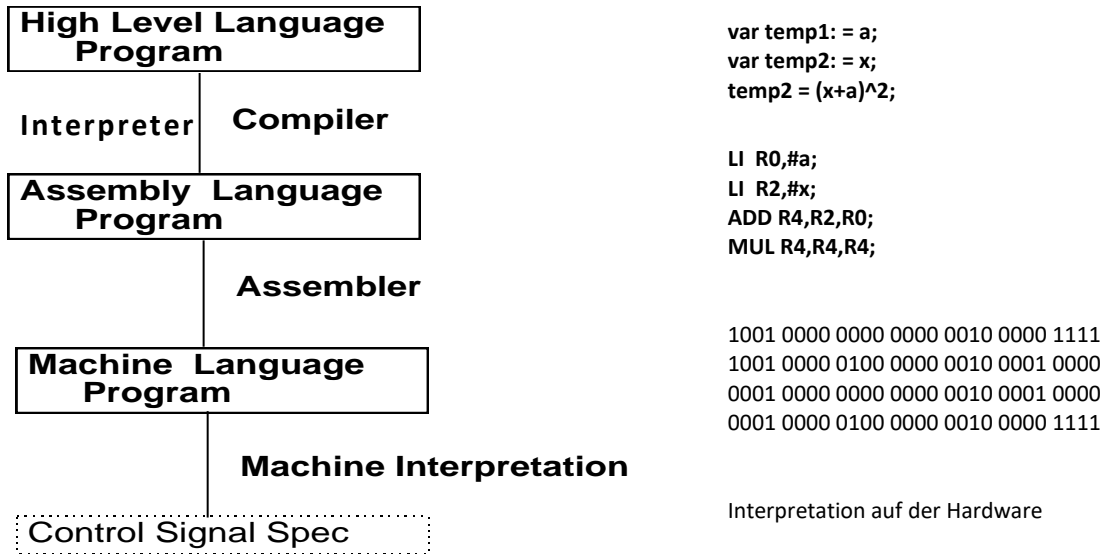
1 JavaScript als eine Programmiersprache

1.1 Programme

Ein Computerprogramm ist eine Folge von Instruktionen (Anweisungen), die vom Computer ausgeführt werden können. Die Programme müssen so geschrieben werden, dass man sie im Hauptspeicher des Computers oder auf einem externen Medium speichern kann. Computer muss diese Anweisungen verstehen. (Problem: Computersprache \neq Menschensprache)

1.2 Programmiersprachen

Sprachen zur Formulierung von Anweisungen, die von einem Computer ausgeführt werden können. Programmiersprachen bilden die wichtigste Schnittstelle zwischen Benutzern und Computern.



Höhere Programmiersprachen

HP beschreiben in einer für Menschen relativ einfach verständlichen Weise, was der Computer tun soll. Die Programmanweisung beschreiben hardwareferne, abstrakte Vorgänge und sind unabhängig von einer bestimmten Maschinenfamilie.

- **Imperative Programmiersprachen** – (lat. imperare = befehlen) Bei diesen Sprachen besteht ein Programm aus einer Folge von Befehlen an den Computer, wie z.B. „Schreibe in die Variable a den Wert 3“, „Springe an die Stelle x im Programm“, „Wiederhole den letzten Schritt zehn mal“... Wesentliche an diesen Sprachen ist das Variablenkonzept. (Variablen sind Behälter, in denen Werte aufbewahrt werden).

Beispiele: Basic, Pascal, C,...

- **Deklarative Programmiersprachen** – Ein „Gegenpol“ zu den imperativen Sprachen. In den deklarativen Sprachen beschreibt man nicht den Weg, wie man ein Problem lösen sollte, sondern man beschreibt genau, was für ein Problem gelöst werden soll. Aus einer nach bestimmten Regeln gebildeten mathematischen Formulierung des Problems wird dann automatisch - für den Anwender oft unsichtbar - eine Lösungsstrategie erzeugt.

Beispiele: SQL, Haskell, Prolog,...

- **Objektorientierte Programmiersprachen** – Bei diesen Sprachen werden alle zum Lösen eines Programms notwendige Informationen (Daten und Anweisungen bzw. Regeln) als Objekte aufgefasst (Man programmiert z.B. einen Taschenrechner). Jedes Objekt kann unterschiedliche Eigenschaften haben (z.B. unterschiedliche Farben, unterschiedliche Größe des Taschenrechners).

auf dem Bildschirm, ...). Jedes Objekt kann über einigen, ihm spezifischen Methoden verfügen (Taschenrechner kann z.B. mit den Zahlen rechnen,...). Objekte können durch Senden von „Nachrichten“ (Mitteilung an andere Objekte) miteinander kommunizieren. (z.B. der programmierte Taschenrechner kann im Worddokument die verschiedenen Berechnungen durchführen,...)

Beispiele: Java, C++, Delphi, Visual Basic,...

Assemblersprachen

(engl. assemble = montieren) – **maschinenorientierte** Programmiersprachen. Ein Programm in einer Assemblersprache besteht aus den computerspezifischen symbolischen Befehlen. Diese Sprache ist **maschinenspezifisch**, d.h. die in dieser Sprache geschriebenen Programme müssen für die Übertragung auf eine andere Maschinenfamilie neu geschrieben werden.

Maschinensprache

Ein Programm in dieser Sprache besteht aus den für die Hardware verständlichen Bitmustern.

Übersetzung einer höheren PS in eine maschinen-orientierte PS

Interpretieren durch einen Interpreter

- Interpreter übersetzt ein Programm zeilenweise. Jede Zeile wird getrennt ausgeführt.
- Die Programme können während des Ablaufs unterbrochen und die Befehle geändert werden.
- Geschwindigkeitsoptimierung ist nicht möglich, Programme laufen langsamer ab.

Compilieren durch einen Compiler

- Ein Compiler übersetzt vollständig ein Programm in eine maschinenorientierte Sprache. Das gesamte Programm wird als ein Packet ausgeführt.
- Das Programm kann während des Ablaufs nicht zwecks einer Korrektur unterbrochen werden.
- Während der Kompilierung optimiert der Compiler die Programmgröße und die Geschwindigkeit. Die Programme laufen schneller ab.

Programmierungsumgebung

Um etwas in einer höheren Programmiersprache zu programmieren, braucht man eine Programmierungsumgebung, die unterschiedliche Werkzeuge zu Verfügung stellt.

- Notwendige Werkzeuge:
 - Ein Editor zu Erstellen und Ändern eines Programmtextes.
 - Ein Interpreter bzw. Compiler zum Übersetzen und auszuführen von Programmen.
 - Ein Debugger für die Fehlersuche in der Testphase eines Programms
- Weitere hilfreiche Werkzeuge:
 - Werkzeugsammlung der graphischen Komponenten
 - Navigator
 - Projektfenster...
- Integrierte Entwicklungsumgebung (Abkürzung IDE, von engl. *integrated development environment*) ist ein Softwarepaket, das auf die komfortable Entwicklung von Anwendungen spezialisiert ist. Manche IDE unterstützen dabei die Programmierung in mehreren Programmiersprachen. Beispiele: Netbeans, Eclipse, VisualStudio, ...

1.3 Was ist JavaScript?

Während HTML lediglich eine Möglichkeit zur Strukturierung von Informationen darstellt, ist JavaScript ein **Werkzeug zur Verarbeitung von Informationen**. Die Seiten, die nur mit HTML und CSS erstellt wurden sind statisch und passiv. JavaScript gibt dem Homepage-Besucher eine Möglichkeit selbst aktiv zu werden. Man spricht: Die Seiten mit JavaScript sind dynamisch und interaktiv. Im Gegenteil zu PHP wird JavaScript im Internet nicht serverseitig sondern **clientseitig ausgeführt**.

JavaScript ist eine **Skriptsprache**.

- Skriptsprachen sind Programmiersprachen, die (mindestens ursprünglich) vor allem für kleine, überschaubare Programmieraufgaben gedacht sind.
- Man spricht davon, dass **Skriptsprachen interpretiert werden**. Programme, die in einer Skriptsprache entwickelt sind, müssen also grundsätzlich immer im Zusammenhang mit einem Interpreter ausgeführt werden.
- Skriptsprachen werden nicht direkt von der Hardware, also vom Prozessor, ausgeführt, sondern von einem anderen Programm. **Sie benötigen einen „Sandbox“**. Typischerweise wird JavaScript in einem Browser ausgeführt.

JavaScript wurde Mitte der 90er Jahre von der Firma Netscape erfunden. Im Laufe des Browserkriegs führte Microsoft als Alternative eigene Skriptsprache - JScript ein. Lange Zeit haben subtile Unterschiede zwischen JScript und JavaScript dem Programmieren viele Kopfschmerzen bereitet. Erst Google hat 2005 mit Google-Maps-Projekt JavaScript ins professionelle Rampenlicht geholt und für die Einhaltung der **ECMAScript – Standard** gesorgt. Inzwischen hat sich JavaScript zu einer mächtigen und ausdrucksstarken Sprache entwickelt.

1.4 Die JavaScript-Hitlist

Für welche Aufgaben wird Javascript bevorzugt eingesetzt?

- Zum Öffnen von weiteren Browserfenstern mit individuellen Vorgaben
- Zum Prüfen von Formulareingaben, bevor das Formular zum Server geschickt wird
- Um dynamisch Inhalt in Seiten zu setzen, je nach zuvor festgelegten Bedingungen
- Um Formularseiten komfortabler für Benutzer zu gestalten
- Um Tabellen ähnlich wie Excel bei einem Klick auf die Kopfzeile zu sortieren oder die Spalten einer Tabelle auszutauschen
- Für Animationen
- Für Spiele
- ...
- Aber auch für umfangreiche Projekte wie Google Maps.

1.5 Kontrollaufgaben

1. Was versteht man unter den Begriffen: Programmieren, Programmiersprache, Programmierumgebung.
2. Welche Typen der Programmiersprachen kennst du?
3. Welche Aufgabe erfüllen Interpreter und Compiler? Wie unterscheiden sich voneinander die Arbeitsweisen des Interpreters und Compilers?
4. JavaScript ist eine Skriptsprache. Was bedeutet das genau?

2 Erste Schritte in JavaScript

2.1 Einbindung von JavaScript in HTML – Dokument

JavaScript direkt in Webseiten eingebettet muss immer in einem `<script>` -Tag stehen.

```
<script> ... </script>
```

Es gibt grundsätzlich zwei Möglichkeiten, Javascript in Webseiten einzubauen. Entweder fügt man den Quellcode in die gleiche Datei ein, die auch den HTML-Code enthält, oder legt man den Quellcode in eine eigene Datei und verweist auf diese von der Webseite.

Wenn man z.B. beim Laden eines HTML-Dokumentes ein neues Fenster mit der Begrüßung „Hallo Welt!“ ausgeben will, kann man das auf folgende Weise tun:

a) Direkt der ganze Quelltext im HTML-Dokument:

```
<script>
    window.alert("Hallo, Welt!");
</script>
```

b) Indirekt als Verweis auf eine JavaScript Datei z.B. `library.js` in der z.B. nur die Anweisung `window.alert("Hallo, Welt!");` steht

```
<script src="library.js"></script>
```

2.2 Grundlegende Eigenschaften der Syntax von JavaScript

- Man muss Groß- und Kleinschreibung beachten. (Mit Zahl und zahl werden zwei verschiedene Elemente bezeichnet!).
- Die Schreibweise bereits existierender Objekte und Methoden ist von JavaScript vorgegeben.
- Zur Eingrenzung von Befehlsblöcken werden geschweifte Klammern `{ }` verwendet.
- Zeichenketten (Text) müssen Sie in Anführungszeichen setzen (so "11" oder so ,11').
- Kommentarzeilen beginnen mit einem doppelten Schrägstrich `//`.
- Statt jede einzelne Kommentarzeile so zu beginnen, können Sie mehrzeilige Kommentare auch mit `/*` beginnen und mit `*/` beenden.
- Eine Befehlszeile muss mit einem Semikolon (`;`) beendet werden, wenn man direkt dahinter (also auf gleicher Höhe) eine weitere Befehlszeile anhängen will.
- Schreibt man Befehlszeilen untereinander, kann man auf das Semikolon verzichten (Man muss aber nicht! Es ist sogar eher üblich, das Semikolon zu setzen).

2.3 Elementare Ein- und Ausgabe-JavaScript-Befehle

Die Computer arbeiten typischerweise nach dem EVA- Prinzip. (Eingabe Verarbeitung Ausgabe) Um nach diesem Prinzip einfache Programme zu schreiben brauchen wir zunächst die einfachsten Befehle um unsere Daten dem Computer zu übermitteln. Dazu werden wir zunächst den Befehl:

- `window.prompt("Meldung", "Vorgabe")` –verwenden.

Für die Ausgabe werden uns zwei Befehle zur Verfügung stehen:

- `window.alert("Meldung")` – Für die Ausgabe in einem neuen Fenster und
- `document.write("zu schreibender Text")` – Für die Ausgabe in dem aktuellen HTML-Dokument
- `console.log("zu schreibender Text")` – Für die Ausgabe in Entwicklungskonsole.

2.4 Variable: Begriff, Deklaration, Initialisierung, (Neu-)Belegung

Bei der Programmierung müssen immer wieder Daten zwischengespeichert werden. Hierzu bedient man sich so genannter Variablen. **Variable** ist nichts anderes als ein logischer Speicherplatz mit dessen Wert. Der gespeicherte Wert einer Variablen kann während der Programmlaufzeit unbegrenzt oft verändert werden. *(Man kann sich eine Variable als ein Gefäß vorstellen, in dem die Daten gespeichert sind. Jedes Gefäß hat seine Bezeichnung (Name). Nach Bedarf kann man in dieses Gefäß einen Wert einlegen oder einen Wert aus diesem Gefäß holen.)*

Der Vorgang der Festlegung einer Variablen wird als Deklaration bezeichnet. Dabei wird Name und der Variablen festgelegt. **Deklaration** = Reservierung von Speicherplatz. Der Computer muss sich den Inhalt einer Variablen merken können. Dazu reserviert er für jede Variable einen bestimmten Speicherbereich. *(In unserem Bild entspricht die Variablendeklaration der Erzeugung (Bereitstellung) und Benennung eines neuen Gefäßes.)*

Syntax der Variablen - Deklaration in JavaScript:

	Deklarationsschlüsselwort	Varname
z.B.	var	zahl ;

Nach der Deklaration der Variable ist der festgelegte Speicherplatz *(das bereitgestellte Gefäß)* zunächst nicht belegt (leer). Die erste Zuweisung (Belegung) der Variable nennt man **Initialisieren**. Jede neue Belegung einer Variablen erreicht man mittels **Zuweisungsoperators**. In JavaScript ist das das einfache Gleichzeichen „=“:

z.B.	zahl = 10
------	------------------

2.5 Datentypen in JavaScript

Programme führen Berechnungen durch, sortieren Schulfreunde nach den Vornamen ihrer Frauen, prüfen Formulare, animieren Elemente einer Webseite und noch vieles mehr. Um derartige Aufgaben durchzuführen, arbeiten die Programme mit unterschiedlichen Datentypen. In JavaScript sind folgende Datentypen von großer Bedeutung:

- **Integer** - Ganze Zahlen
- **Fließkommazahlen** (Float) - enthalten einen Dezimalpunkt (kein Komma!)
- **Bool'sche Werte** - Wahrheitswerte **true** oder **false**
- **String** - Zeichenketten
- **Objekte** - zusammengesetzte Datentypen, die über spezielle Eigenschaften und Methoden verfügen

JavaScript ist im Gegenteil zur Java eine **dynamisch typisierte Programmiersprache**. Die Zuteilung des Typs einer Variablen geschieht hier erst zur Laufzeit eines Programms. Das kann zu unerwarteten Effekten führen. Der Problematik der fehlerhaften dynamischen Typzuweisung kann man aus dem Weg gehen, indem man vor dem Anwenden des Wertes einer Variablen zunächst ihren erwünschten Typ erzwingt. Dazu dienen folgende JavaScript Befehle:

<code>parseFloat()</code>	(in Kommazahl umwandeln)
<code>parseInt()</code>	(in Ganzzahl umwandeln)
<code>String()</code>	(in Zeichenkette umwandeln)

Interessant ist darüber hinaus der JavaScript Befehl **eval()** der ein als Zeichenkette übergebene Term mathematisch korrekt auswertet z.B. `eval („(3+4)*2“)` wird als 7 ausgewertet.

2.6 Operatoren

Jede Programmiersprache, so auch JavaScript, bietet eine ganze Reihe an Operatoren an, um Informationen verarbeiten zu können. Die meisten Operatoren kann man dabei sinnvoll nur auf bestimmte Datentypen anwenden.

Berechnungs-Operatoren

Operator	Beschreibung	Beispiel	Resultat
+	Addition	x=2 und x+2	4
-	Subtraktion	x=2 und 5-x	3
*	Multiplikation	x=4 und x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulo (Restwert bei einer Division)	5%2 10%2	1 0
++	Inkrement	x=5 x++	x=6
--	Dekrement	x=5 x--	x=4

Zuweisungs-Operatoren

Operator	Beschreibung	Beispiel	Resultat
=	Wertzuweisung	x=5	x enthält 5
+=	Addition und Zuweisung	x+=5 oder x=x+5	x enthält 10
-=	Subtraktion und Zuweisung	x-=2 oder x=x-2	x enthält 8
=	Multiplikation und Zuweisung	x=2 oder x=x*2	x enthält 16
/=	Division und Zuweisung	x/=4 oder x=x/4	x enthält 4
%=	Modulo-Operation und Zuweisung	x%=3 oder x=x%3	x enthält 1

Vergleichs-Operatoren

Operator	Beschreibung	Beispiel
==	Werte sind gleich	5==8 ergibt <i>false</i>
!=	Werte sind ungleich	5!=8 ergibt <i>true</i>
>	linker Wert ist grösser	5>8 ergibt <i>false</i>
<	linker Wert ist kleiner	5<8 ergibt <i>true</i>
>=	ist grösser oder gleich	5>=8 ergibt <i>false</i>
<=	ist kleiner oder gleich	5<=8 ergibt <i>true</i>

Logische Operatoren

Operator	Beschreibung	Beispiel
&&	und (Ergibt true nur wenn beide Teilausdrücke true sind.)	x=6 und y=3 (x < 10 && y > 1) ergibt <i>true</i>
	oder (Ergibt true wenn einer oder beide Teilausdrücke true sind.)	x=6 und y=3 (x==5 y==5) ergibt <i>false</i>
!	nicht Kehrt den Wert um!	!true ergibt <i>false</i> !false ergibt <i>true</i>

Anfüge-Operator (+) angewandt auf eine Zeichenkette

Operator	Beschreibung	Beispiel	Resultat
+	Braucht man, um zwei Variablen zusammenzusetzen	a="Hans" b="Muster" c = a + b	c enthält den Wert "HansMuster"

2.7 Mathematische Berechnungen mit Math- Objekt

Im Vergleich zu den nicht objektorientierten Programmiersprachen verfügt JavaScript über eine relativ kleine Menge von Operatoren. Für eine aufwendigere Informationsverarbeitung benutzt JavaScript stattdessen speziell dafür bereitgestellte Objekte. Was die Objekte sind und wie sie genau funktionieren werden wir im weiteren Verlauf des Kurses kennenlernen. An dieser Stelle lernen wir nur, was man eintippen muss um bestimmte mathematische Konstanten und Funktionen nutzen zu können

Math.E	Eulersche Konstante
Math.PI	Konstanter Wert für Kreiszahl PI
Math.abs(x)	Absolutwert (positiver Wert)
Math.acos(x)	Arcus-Cosinus
Math.asin(x)	Arcus-Sinus
Math.atan(x)	Arcus-Tangens
Math.cos(x)	Cosinus
Math.exp(x)	Exponentialwert (e^x)
Math.log(x)	Anwendung des natürlichen Logarithmus, d.h. zur Basis e (Euler'sche Zahl)
Math.max(x,y)	Liefert die größere Zahl von x und y.
Math.min(x,y)	Liefert die kleinere Zahl von x und y
Math.pow(x,y)	Potenzfunktion: x hoch y
Math.random()	Zufällige Gleitkommazahl zwischen 0 und 1
Math.round(x)	Rundet die Zahl x auf die nächste ganze Zahl
Math.sin(x)	Sinus
Math.sqrt(x)	Quadratwurzel
Math.tan(x)	Tangens

2.8 Ausdrücke

Um sich in JavaScript richtig auszudrücken, benötigen Sie JavaScript-**Ausdrücke**. Ausdrücke bestehen normalerweise aus den Werten, Variablen und Operatoren und werden zu Werten evaluiert (ausgewertet). Beispiele für die folgende Variablenbelegung

```
var a = 10;
var b = 20;
var c = „abc“;
a + b   wird evaluiert zu → 30
a + c   wird evaluiert zu → „10abc“
a > b   wird evaluiert zu → false
```


2.9 Übungen:

1. Schreibe ein Programm, das zunächst in einem „Prompt-Fenster“ Deinen Namen erfragt und anschließend im aktuellen HTML-Dokument eine Begrüßung nach der Form :

„Herzlich willkommen Paul !!!“ ausgibt.

2. Schreibe ein Programm zur Berechnung der Summe von zwei Zahlen. Die Werte diesen Zahlen sollten dabei mit einem „Prompt-Fenster“ abgefragt werden. Der Ausgabeformat (in einem „Alert-Fenster“) sollte folgendes Format haben: z.B. $5 + 7 = 12$

In allen folgenden Programmen sollen die Eingaben über „Prompt-Fenster“ erfolgen und die Ergebnisse sollen in dem aktuellen HTML-Dokument ausgegeben werden:

3. Schreibe ein Programm zur Berechnung der Stromstärke, wenn Spannung und Widerstand gegeben sind. Der Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

$$\begin{aligned}U &= 12 \\R &= 3 \\I &= 4\end{aligned}$$

4. Schreibe ein Programm zur Berechnung der Hypotenuse. Der Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

$$\begin{aligned}a &= 3 \\b &= 4 \\c &= 5\end{aligned}$$

5. Schreibe ein Programm, das die Nullstellen einer quadratischen Funktion berechnet. Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

$$\begin{aligned}x^2 + 2x - 8 &= 0 \\x_1 &= -4 \\x_2 &= 2\end{aligned}$$

6. Schreibe ein Programm, das den Ersatzwiderstand berechnet:

- a. In einer Reihenschaltung ($R=R_1+R_2$)
- b. In einer Parallelschaltung: $R=1/(1/R_1+1/R_2)$

Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

Reihenschaltung:

$$\begin{aligned}R_1 &= 50 \, \Omega \\R_2 &= 150 \, \Omega \\R_{\text{ges}} &= 200 \, \Omega\end{aligned}$$

7. Schreibe ein Programm, das die vorgegebene Sekundenanzahl in Stunden, Minuten und Sekunden umrechnet. Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

8000 Sekunden = 2 Stunden 13 Minuten 20 Sekunden:

8. Schreibe ein Programm, das die Dezimalzahlen (bis 256) in Binärzahlen umrechnet. Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

$$100 \text{ (Dec)} = 01100100 \text{ (Bin)}$$

9. Schreibe ein Programm, das eine Zahl auf das nächstgelegene Vielfache von 100 auf- bzw. abrundet. Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

1259 ist gleich ca. 1300

10. Schreibe ein Programm, das die Eurobetrag in die polnische Zloty umrechnet. Es gilt dabei: 1€ = 4,1342 zł. Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

52.92 € = 218.78 zł

Nun folgen einige Aufgaben zu der Rechnung mit booleschen Werten.

11. Schreibe ein Programm, das aus dem vorgegebenen Alter (durch **true** oder **false** Ausgabe) entscheidet, ob man volljährig ist.

Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

*Alter: 15
Volljährig: false*

12. Schreibe ein Programm, das aus dem vorgegebenen Alter (durch **true** oder **false** Ausgabe) entscheidet, ob es sich um einen Jugendlichen handelt (zwischen 13 und 17)

Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

*Alter: 15
Jugendlicher: true*

13. Schreibe ein Programm, das durch **true** oder **false** Ausgabe entscheidet, ob eine vorgegebene Zahl dreistellig ist. Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

*Zahl: 1105
Dreistellig: false*

14. Schreibe ein Programm, das aus den vorgegebenen p und q Werten durch **true** oder **false** Ausgabe entscheidet, ob die entsprechende quadratische Gleichung lösbar ist oder nicht.

Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

*p : 16
 q : 200
Quadratische Gleichung lösbar: false*

15. Weil die astronomische Dauer eines Jahres (wenn die Erde die Sonne einmal umrundet hat) etwas länger ist als 365 Tage, wurden Schaltjahre zum Ausgleich eingefügt. Ein Schaltjahr ist ein Jahr, welches eine Jahreszahl hat, die durch 4 teilbar ist. Jahreszahlen, die durch 100 teilbar sind, sind allerdings keine Schaltjahre. Es sei denn, die Jahreszahl ist durch 400 teilbar. **Schreibe ein Programm, das aus dem vorgegebenen Jahr durch **true** oder **false** Ausgabe entscheidet, ob es sich um ein Schaltjahr handelt.**

Das Ausgabeformat sollte dabei folgendem Beispiel entsprechen:

*Jahr: 2015
Schaltjahr: false*

3 Algorithmen

3.1 Definition und Eigenschaften Algorithmen

Der Mensch hatte von jeher den Wunsch, Handlungen in knapper, verständlicher Form aufzuschreiben. So findet man für alle Lebensbereiche entsprechende Vorschriften (Algorithmen):

- Gebrauchsanweisungen,
- Bastelanleitungen,
- Spielregeln,
- Kochrezepte,
- mathematische Lösungsverfahren,
- Strickmustervorlagen usw.

Definition:

Algorithmus ist eine Verarbeitungsvorschrift, die aus einer endlichen Folge von eindeutig ausführbaren Anweisungen besteht und zur Lösung einer Klasse von Problemen führt.

In dieser Definition wurden folgende notwendige Attribute (Eigenschaften) jedes Algorithmus beschrieben:

- **Endlichkeit:**
Ein Algorithmus besteht aus endlich vielen Anweisungen (Befehlen bzw. Regeln) endlicher Länge.
- **Eindeutigkeit:**
Mit jeder Anweisung ist auch die nächstfolgende festgelegt, d.h. die Reihenfolge der Abarbeitung der Anweisungen unterliegt nicht der Willkür des Ausführenden. (Man sagt auch: Algorithmen sind **deterministisch**.)
Gleiche Eingabegrößen führen bei wiederholter Abarbeitung eines Algorithmus zu den gleichen Ausgabegrößen. (Man sagt auch: Algorithmen sind **determiniert**.)
- **Ausführbarkeit:**
Jede einzelne Anweisung muss für den Ausführenden des Algorithmus (den „Prozessor“) verständlich und ausführbar sein.
- **Allgemeingültigkeit**
Ein Algorithmus muss auf **alle Aufgaben gleichen Typs** (Aufgabenklasse) anwendbar sein und (bei richtiger Anwendung) stets zum gesuchten Resultat (zur Lösung) führen.

Diese Eigenschaften reichen meist aus, um zu entscheiden, ob eine Prozessbeschreibung ein Algorithmus ist oder nicht, also ob ein Vorgang automatisiert und von einer Maschine übernommen werden kann oder nicht. Oft wird darüber hinaus gefordert, dass der Prozess, den ein Algorithmus beschreibt, irgendwann einmal zum Ende kommt (**terminiert**). Die Erfüllung von dieser Bedingung ist jedoch nicht der notwendige Bestand des Algorithmus Begriffes.

Übungen:

Welche der folgenden Prozesse können durch einen Algorithmus beschrieben werden?

- a) Wechseln eines Autoreifens (Radwechsel)
- b) Konstruieren einer zur Geraden g parallelen Geraden h durch einen Punkt P, der nicht auf g liegt
- c) Addition zweier Brüche
- d) Benoten eines Aufsatzes
- e) Schießen eines Tores beim Handball
- f) Aufschreiben aller geraden natürlichen Zahlen

3.2 Darstellungsmöglichkeiten für Algorithmen

A. Verbale Beschreibung der Algorithmen

a) Beschreibung mithilfe der Umgangssprache:

Als Beispiel für die umgangssprachliche Beschreibung eines Algorithmus seien die Rundungsregeln für natürliche Zahlen angegeben: „Beim Runden natürlicher Zahlen werden alle auf eine bestimmte Ziffer folgenden Ziffern durch Nullen ersetzt. Die Stelle, auf die zu runden ist, bleibt unverändert, wenn ihr eine 0, 1, 2, 3 oder 4 folgt. Die Stelle, auf die zu runden ist, wird um 1 erhöht, wenn ihr eine 5, 6, 7, 8 oder 9 folgt.“ Eine solche Darstellung ist wenig sinnvoll, wenn man an den Prozessor Computer denkt.

b) Verbale formalisierte Beschreibung:

Bestimmte, oft wiederkehrende Elemente eines Algorithmus werden immer mit den gleichen Worten beschrieben. Obiges Beispiel könnte dann so aussehen:

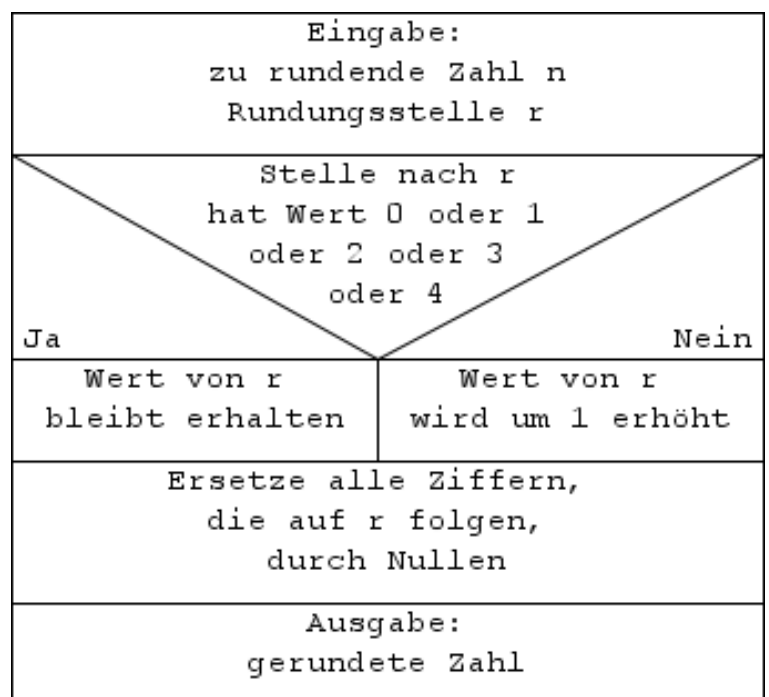
1. **Eingabe:** zu rundende natürliche Zahl, Rundungsstelle
2. **Wenn** auf die Rundungsstelle eine 0, 1, 2, 3 oder 4 folgt,
dann bleibt diese Stelle unverändert,
sonst wird sie um 1 erhöht.
3. Alle Ziffern, die auf die Rundungsstelle folgen, werden durch Nullen ersetzt.
4. **Ausgabe:** gerundete Zahl

a) Programme

Ein Programm ist die Formulierung eines Algorithmus in einer Programmiersprache.

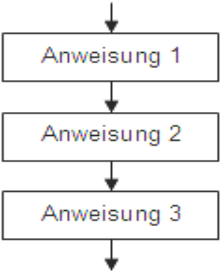
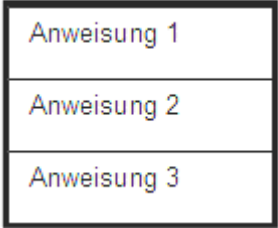
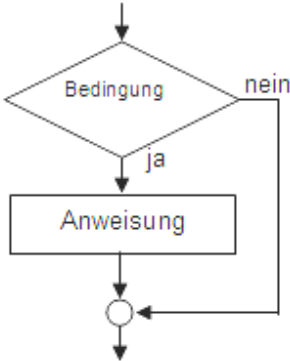
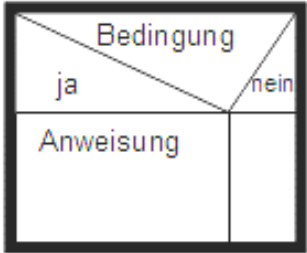
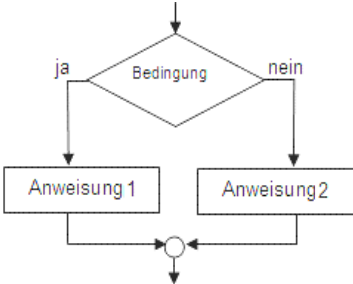
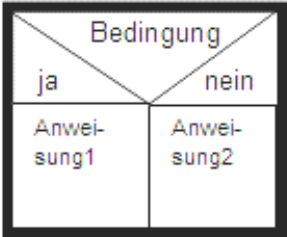
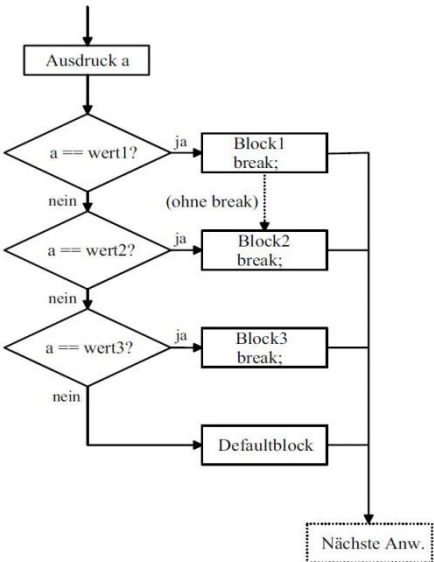
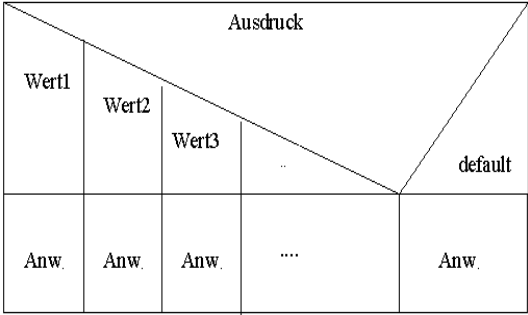
B. Grafische Darstellung:

Bei der grafischen Darstellung von Algorithmen haben sich zwei Darstellungsmöglichkeiten durchgesetzt: der Programmablaufplan (Flussdiagramm) und das Struktogramm. Hier zum Vergleich ein Beispiel.



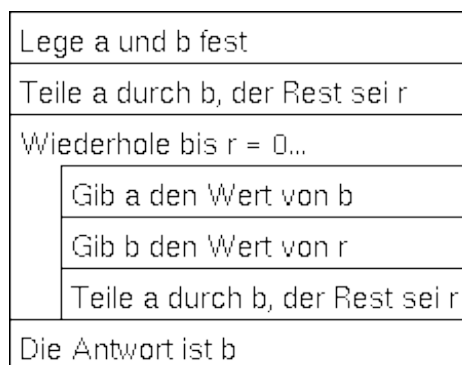
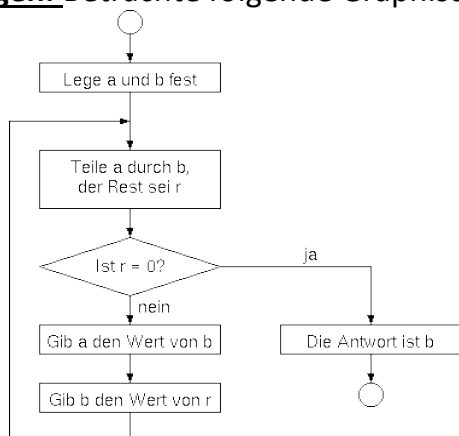
3.3 Kontrollstrukturen von Algorithmen

Um Ablauf des Algorithmus zu steuern, benutzt man s.g. Kontrollstrukturen. Eine **Kontrollstruktur** definiert, wie die einzelnen **Anweisungen** und deren **Blöcke** (eine Zusammenfassung von Anweisungen, die in JavaScript in geschweiften Klammern eingeschlossen sind) ausgeführt werden: sequenziell, bedingt oder wiederholt.

Bezeichnung	Sprachliche Formulierung	Programmablaufplan	Struktogramm
Sequenz	Anweisung 1 Anweisung 2 Anweisung 3		
einseitige Verzweigung	Wenn Bedingung erfüllt ist dann Anweisung		
zweiseitige Verzweigung	Wenn Bedingung erfüllt ist dann Anweisung 1 sonst Anweisung 2		
Fallauswahl	Wenn Ausdruck (a) gleich wert1 dann Block1 Wenn a gleich wert2 dann Block2 ... sonst Defaultblock		

Bezeichnung	Sprachliche Formulierung	Programmablaufplan	Struktogramm
Kopf-Bedingungs-gesteuerte Schleife (while-Schleife)	Solange Eingangsbedingung erfüllt ist Führe aus Anweisung		
Kopf-Zähler-gesteuerte Schleife (for-Schleife)	Starte i mit 0 Solange i < n Führe aus Anweisung erhöhe i um 1		
Fuß-gesteuerte Schleife (do-while-Schleife)	Führe aus Anweisung solange Wiederholungsbedingung erfüllt ist		

Übungen: Betrachte folgende Graphische Darstellungen von einem Algorithmus



- Lege am Anfang die Variable $a = 18$ und die Variable $b = 14$ fest und untersuche, wie sich die Belegung der Variablen a , b und r im Laufe des Algorithmus verändert.
- Notiere die Veränderung der Variablen im Laufe des Algorithmus für eine andere von dir gewählte Anfangsbelegung von a und b .
- Was errechnet der Algorithmus, der hier einmal als Flussdiagramm, einmal als Struktogramm dargestellt wird.

3.4 Struktogramme lesen, verstehen und entwickeln

Die Darstellung der Algorithmen mittels Flussdiagrammen hat auch einige Nachteile:

- Für grundlegende Kontrollstrukturen existieren keine Symbole,
- Schachtelstrukturen sind im PAP nicht gut zu erkennen.

Deswegen werden wir vor allen Struktogramme benutzen. Mit den Struktogrammen kann man für jeden Algorithmus die Abläufe strukturieren, so dass man sie später in jede Programmiersprache umsetzen kann.

3.4.1 Lineare Strukturen

Beispiel 1

Ein Programm soll den Radius eines Kreises über die Tastatur einlesen, die Fläche berechnen und anschließend den Wert der Fläche ausgeben.

Eingabe Radius
$F = \text{Radius} * \text{Radius} * 3.14$
Ausgabe F

Bei diesem Programm handelt es sich um die klassische E-V-A (Eingabe-Verarbeitung-Ausgabe) – Situation. Alle Anweisungen werden in zeitlicher Reihenfolge ausgeführt. Es ist zu beachten, dass die Bezeichnungen der Variablen in allen Anweisungen korrekt verwendet werden müssen.

Beispiel 2

In diesem Beispiel wird die Variable A zu Anfang auf den Wert 5 gesetzt, die Variable B auf den Wert 3. Der Wert der Variable C berechnet sich aus dem Produkt aus A und B. Der Wert von C soll ausgegeben werden.

Hier wird die Zahl 15 ausgegeben.

$A = 5$
$B = 3$
$C = A \times B$
Ausgabe C

Hier werden mehrere Variable verwendet. Die Deklarationen der Variablen wurden hier nicht als zusätzliche explizite Anweisungen in das Struktogramm übernommen. Ein Struktogramm sollte zunächst verständlich eine Grundidee des Algorithmus darstellen und sich nicht (zu) sehr um die spätere Implementierung in eine konkrete Programmiersprache kümmern.

Übungen:

a) Erstellen Sie das Struktogramm für folgende logische Anweisungen:

Variable x = 2
Variable y = 6
Variable z = y – x
Variable y = 4
Variable z = z + y
Ausgabe z

Welcher Wert wird ausgegeben?

b) In den Übungen auf den Seiten 9 und 10 handelte sich ausschließlich um lineare Programmstrukturen.

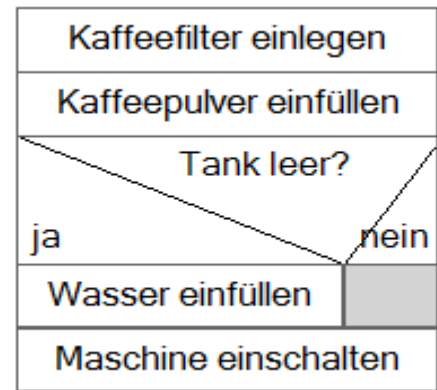
- Erstelle einen passenden Struktogramm zu der Aufgabe 3.
- Erstelle einen passenden Struktogramm zu der Aufgabe 7.

3.4.2 Verzweigungen

Beispiel 1

Beim Kaffeekochen in einer herkömmlichen Maschine fallen folgende Tätigkeiten an:

Kaffeefilter einlegen
 Kaffeepulver einfüllen
 Prüfen, ob noch Wasser im Tank ist.
 Wenn nein,
 dann Tank auffüllen
 Maschine einschalten



Bei einer Verzweigung gibt es immer zwei mögliche Fälle (Ja – Nein). In beiden Fällen können weitere Aktionen erfolgen. Ein Zweig kann aber auch (wie in diesem Beispiel) ohne Aktion bleiben.

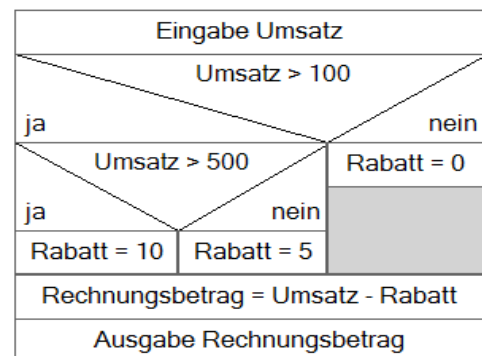
Beispiel 2

Je nach Höhe des Umsatzes wird dem Kunden ein bestimmter Rabatt gewährt.

Wenn der Umsatz höher ist als 100 €, bekommt der Kunde 5 € Rabatt.

Beträgt der Umsatz mehr als 500 € erhält der Kunde 10 € Rabatt.

Ein Programm soll den Rechnungsbetrag des Kunden abzüglich Rabatt berechnen.



In vorliegendem Beispiel wird gezeigt, dass Verzweigungen auch ineinander verschachtelt werden können. D. h. dass innerhalb eines ja – oder nein – Zweiges wieder eine Verzweigung folgen kann.

Übungen:

- Zeichnen Sie ein Struktogramm für folgende Problemstellung: Es wird eine Zahl über die Tastatur eingegeben. Wenn die Zahl gerade ist, wird sie mit 2 multipliziert, wenn sie ungerade ist, wird zu dieser Zahl der Wert 1 addiert. Anschließend wird das Ergebnis ausgegeben.
- Weil die astronomische Dauer eines Jahres (wenn die Erde die Sonne einmal umrundet hat) etwas länger ist als 365 Tage, wurden Schaltjahre zum Ausgleich eingefügt. Ein Schaltjahr ist ein Jahr, welches eine Jahreszahl hat, die durch 4 teilbar ist. Jahreszahlen, die durch 100 teilbar sind, sind allerdings keine Schaltjahre. Es sei denn, die Jahreszahl ist durch 400 teilbar. Erstelle der ein Struktogramm für ein Programm, welches prüft, ob eine eingegebene Jahresziffer ein Schaltjahr ist oder nicht und anschließende eine entsprechende Antwort ausgibt. Anders als bei der Aufgabe 15 auf der Seite 10, benutze Verzweigungen bei der Lösung dieser Aufgabe.

3.4.3 Mehrfachauswahl - Fallauswahl

Beispiel 1

Ein Auswahlmenü bietet die folgenden Optionen:

- 1 - Anzeigen
- 2 - Ausdrucken
- 3 - Neueingabe
- 0 - Programm beenden.

Menu anzeigen			
Auswahlziffer			
1	2	3	0
anzeigen	Drucken	Neueingabe	Beenden

Die einzelnen Fälle (**case**) werden je nach dem aktuellen Wert einer Variablen ausgeführt. Diese Variable nennt man **switch** (Schalter). Der switch muss in jedem Fall eine ganze Zahl (integer) sein. Das dargestellte Konstrukt wird auch **switch-case-Konstrukt** genannt.

Beispiel 2

Für ein Zeugnis wird der Ziffernwert einer Note eingelesen.

Anhand des Wertes soll ein Text gedruckt werden, der den Notenwert wiedergibt (Z. B. Note 1 = sehr gut).

Eingabe note						
note						
1	2	3	4	5	6	sonst
sehr gut	gut	befriedigend	ausreichend	mangelhaft	ungenügend	Fehlermeldung

Hier ein Beispiel für eine Mehrfachauswahl mit einem **default-Zweig**. Dieser wird ausgeführt, wenn der Schalter einen Wert hat, dem kein case zugeordnet ist.

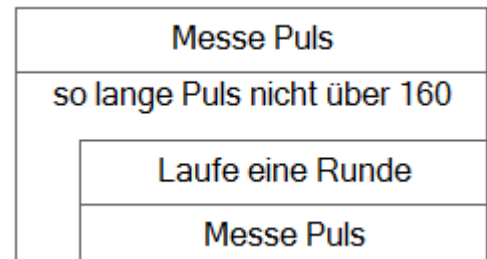
Übungen:

- a) Nach Eingabe einer Monatsziffer (1 – 12) soll der passende Monat am Bildschirm ausgegeben werden (z.B. 3 = März). Bei einer Fehleingabe soll eine Fehlermeldung ausgegeben werden. Stelle die Programmlogik als Struktogramm dar.
- b) Nun sollen die deutschen Wochentags Bezeichnungen ins Englisch übersetzt werden. Bei einer Fehleingabe soll eine Fehlermeldung ausgegeben werden. Stelle die Programmlogik als Struktogramm dar.
- c) Finde drei weitere Aufgabestellungen, die man mittels eines Algorithmus mit Fallauswahl-Kontrollstruktur lösen kann.

3.4.4 Kopf – Bedingungsgesteuerte Schleife

Beispiel 1

Ein Läufer läuft Runden in einem Stadion. Sein Trainingsprogramm besagt, dass er keine weitere Runde laufen soll, wenn sein Puls den Wert von 160 überschreitet

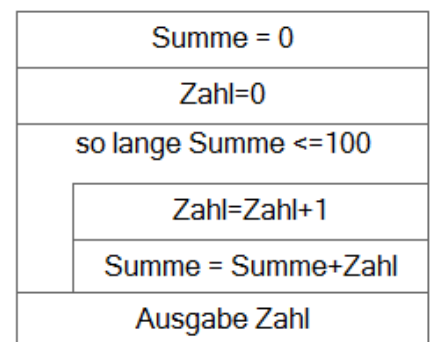


Merkmal einer kopfgesteuerten Schleife ist es, dass die Wiederholungsbedingung zu Anfang der zu wiederholenden Anweisung geprüft wird. Das heißt, dass unter Umständen, die Anweisungen in der Schleife überhaupt nicht ausgeführt werden, wenn die Bedingung gleich zu Beginn schon nicht erfüllt ist. (In obigem Beispiel kann es unter Umständen auch zu einer Endlosschleife kommen, wenn der Puls des Läufers niemals den Wert von 160 überschreitet).

Beispiel 2

Es werden alle Zahlen von 1 bis n fortlaufend addiert, so lange bis die Summe den Wert von 100 überschreitet. Dann wird die letzte Zahl ausgegeben

Wichtig ist es hierbei, den Wert von Summe vor dem Schleifeneintritt auf 0 zu setzen. Da sonst innerhalb der Schleife die Zahl zu einem unbekannten Wert addiert wird. Innerhalb der Schleife muss die Zahl bei jedem Durchlauf um 1 erhöht werden.

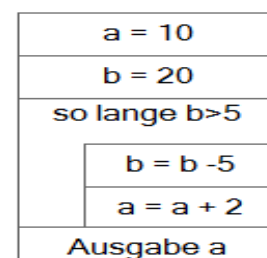


Die Schleifenbedingung wird hier mit dem <= (kleiner gleich) Operanden gebildet. Alternativ hätte man auch schreiben können: „so lange Summe nicht größer als 100“. Zu beachten ist auch, dass die Zahl innerhalb der Schleife vor der Summenbildung erhöht wird. Würde man den Wert der Zahl im Anschluss an die Summenbildung erhöhen, wäre der Wert der Zahl der am Ende ausgegeben wird um 1 zu hoch!

Übungen:

a) Bei einem Würfelspiel wird mit einem Würfel so lange gewürfelt, bis eine 6 fällt. Die Anzahl der Würfe wird gezählt. Wenn eine 6 gefallen ist, wird die Anzahl der Würfe ausgegeben.

b) Gegeben ist folgender Struktogramm.
Notiere die Veränderung der Variablen a und b im Laufe des Algorithmus (vor der Schleife, beim jeden Schleifenrumpf-Durchlauf und nach der Schleife) und ermittle dadurch den Ausgabewert.



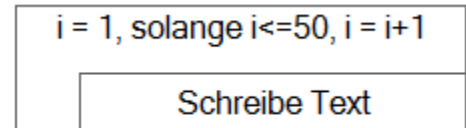
c) Zeichnen Sie ein Struktogramm nach folgenden Anweisungen:
Der Wert von x beträgt 1, der Wert von y beträgt 3.
so lange die Summe von x und y <50 ist sollen folgende Anweisungen ausgeführt werden:
- Es wird die Summe von x und y gebildet,
- Der Wert von x wird um 2 erhöht.
Im Anschluss an die Schleife wird der aktuelle Wert von x ausgegeben. Wie groß ist er

3.4.5 Kopf – Zählergesteuerte Schleife

Zählergesteuerte Schleifen werden immer dann verwendet, wenn die Anzahl der Schleifendurchläufe vorher bekannt ist. Zählschleifen sind der schnellste Schleifentyp, d. h. sie benötigen die wenigste Rechenzeit. Deshalb ist es immer angebracht, zu überprüfen, ob man eine bedingungsgesteuerte Schleife nicht durch eine Zählschleife ersetzen kann.

Beispiel 1

Hans soll 50 Mal schreiben:
„Ich muss immer meine Hausaufgaben machen“.



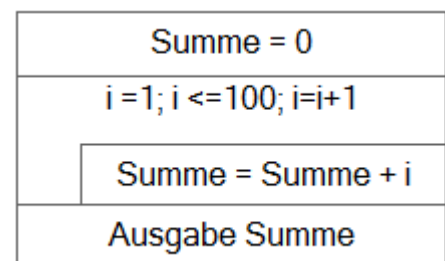
Im Schleifenkopf einer zählergesteuerten Schleife müssen drei Angaben gemacht werden: Der Startwert eines Zählers, die Bedingung mit der die Schleife durchlaufen werden soll und die Veränderung des Zählers nach jedem Durchlauf.

Im Beispiel wird ein Zähler mit der Bezeichnung i verwendet, dessen Wert zu Anfang auf 1 gesetzt wird. Die Bedingung prüft, ob der Wert von i noch kleiner oder gleich 50 ist. Nach jedem Schleifendurchlauf wird der Wert von i um 1 erhöht.

Beispiel 2

Es soll die Summe aller Zahlen von 1 bis 100 berechnet und am Ende ausgegeben werden.

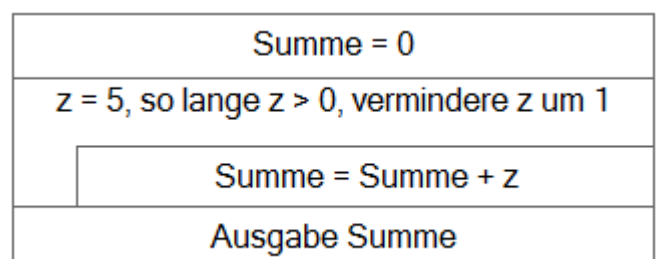
Der Wert des Zählers wird in jedem Durchlauf zum Wert der Summe hinzu addiert. Am Ende wird die Summe mit dem Wert 5050 ausgegeben



Übungen:

- a) Erstellen Sie ein Struktogramm, welches die Summe aller Zahlen zwischen 100 und 1000 berechnet und am Ende ausgibt.
- b) Es soll Produkt von 15 ersten natürlichen Zahlen berechnet werden ($1 \cdot 2 \cdot \dots \cdot 15$)
- c) Es sollte Summe von 200 ersten positiven ungeraden Zahlen berechnet werden.

- d) Gegeben ist folgender Struktogramm
Notiere die Veränderung der Variablen Summe und z im Laufe des Algorithmus (vor der Schleife, beim jeden Schleifenrumpf-Durchlauf und nach der Schleife) und ermittle dadurch den Ausgabewert.

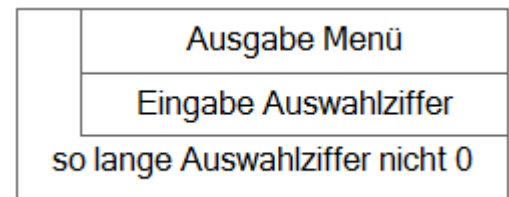


3.4.6 Fuß – (Bedingung) gesteuerte Schleife

Diese Schleife funktioniert sehr ähnlich Kopf- Bedingungsgesteuerte Schleife. Der einzige Unterschied liegt nur darin, dass bei dieser Schleife mindestens ein Durchlauf gesichert ist.

Beispiel 1

Ein Auswahlmenü wird so lange angezeigt, wie der Benutzer nicht den Wert 0 für Beenden eingegeben hat.

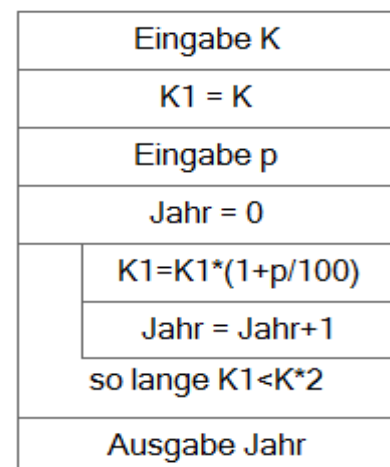


Fußgesteuerte Schleifen eignen sich besonders für Menüführungen, da das Menü auf jeden Fall einmal angezeigt werden soll. Die Bedingung für den Schleifendurchlauf wird erst geprüft, nachdem die Anweisungen der Schleife einmal durchlaufen wurden.

Beispiel 2

Ein Kapital K wird mit einem Zinssatz p verzinst. Es soll berechnet, wie viele Jahre es dauert, bis sich das Kapital verdoppelt hat.

Zu beachten ist hier, dass die Hilfsvariable K1 eingeführt wurde, mit der jeweils der aktuelle Wert des Kapitals berechnet wird. Dieser wird dann in der Bedingung mit dem Anfangswert K verglichen.



Übungen:

- Tim soll 500 Mal schreiben. „Ich muss immer meine Hausaufgaben machen“. Ähnliche Aufgabe mit Hans wurde mit Hilfe einer zählergesteuerten Schleife gelöst. Löse die „Tim-Aufgabe“ mittels eines Struktogramms mit einer fußgesteuerten Schleife.
- Ein Menü hat die folgenden Optionen:
 - 1 – Neuer Datensatz
 - 2 – Daten anzeigen
 - 3 – Daten korrigieren
 - 4 – Daten löschen
 - 0 – BeendenDer Benutzer gibt eine Auswahlziffer ein. Anhand der Auswahlziffer werden weitere Anweisungen ausgeführt. Zeichne Sie ein Struktogramm unter Einbeziehung einer Mehrfachauswahlstruktur ohne default-Zweig.
- Tobias will Party organisieren, wenn er Führerschein bekommt. Um den zu bekommen, muss er jedoch sooft zunächst eine theoretische und dann eine praktische Prüfung ablegen, bis er beide besteht. Zeichne ein Struktogramm, das veranschaulicht, wann uns Tobias zur Party einladen wird.
- In der Eifel gibt es relativ wenige Ampeln für die Fußgänger. Trotzdem ist man immer wieder gezwungen, eine befahrene Straße zu überqueren. Entwickle einen passenden Algorithmus für eine sichere Überquerung einer Straße ohne Fußgängerampel und stelle ihn in Form eines Struktogramms dar.

3.5 Kontrollstrukturen in JavaScript:

3.5.1 Verzweigungen → if-else-Verzweigung

Mit dem if-else-Konstrukt lassen sich einfache Verzweigungen umsetzen. Ist die Bedingung wahr, wird der erste Anweisungsblock ausgeführt; ist die Bedingung unwahr, wird Anweisungsblock 2 ausgeführt. Der else-Teil kann weggelassen werden.

```
if(Bedingung) {  
    Anweisungsblock1;  
} else {  
    Anweisungsblock2;  
}
```

if-else-Verzweigungen lassen sich beliebig oft schachteln und hintereinander hängen: Ist *Bedingung1* wahr, wird *Anweisungsblock1* ausgeführt und die *Bedingung2* nicht mehr geprüft, da sie über den else-Zweig aufgerufen würde. Die Bedingungen 1 und 2 können sich dabei auf unterschiedliche Variablen bzw. Ausdrücke beziehen.

```
if(Bedingung1) {  
    Anweisungsblock1;  
} else if(Bedingung2) {  
    Anweisungsblock2;  
} else {  
    Anweisungsblock3;  
}
```

3.5.2 Mehrfachauswahl - Fallauswahl → switch-Mehrfachverzweigung

Die switch - Mehrfachverzweigung kann nur dann eingesetzt werden, wenn man in Abhängigkeit vom Wert einer einzelnen (integer- oder String-) Variable oder auch eines einzelnen Ausdrucks verschiedene Aktionen ausführen will.

```
switch(Variable/Ausdruck) {  
    case 1 :  
        Anweisungsblock1;  
    case 2 :  
        Anweisungsblock2;  
        break;  
    case 5 :  
        Anweisungsblock3;  
        break;  
    default:  
        Anweisungsblock4;  
}
```

case *Wert*: Mit case wird ein „Einsprungpunkt“ in die Anweisungen des switch definiert. Hat *Variable/Ausdruck* den passenden Wert *Wert*, werden die nachfolgenden Anweisungen ausgeführt bis zum Schlüsselwort break; oder zum Ende des switch-Konstruktes.

Beispiel oben: Hat (*integer*) *Variable* den Wert 1, werden *Anweisungsblock1* und *Anweisungsblock2* ausgeführt.

Hat *Variable/Ausdruck* den Wert 2, wird nur *Anweisungsblock2* ausgeführt.

break;	break definiert einen „Ausstiegspunkt“ innerhalb des switch-Konstruktes. Erreicht das Programm dieses Schlüsselwort, wird die Ausführung des switch Abgebrochen und das Programm nach dem switch fortgesetzt.
default:	Mit default lassen sich „Standard-Anweisungen“ definieren, die ausgeführt werden, wenn <i>Variable</i> keinem der „case-Werte“ entspricht. default ist optional und muss nicht gesetzt werden, damit switch funktioniert. Beispiel oben: Ist <i>Variable</i> nicht vom Wert 1, 2 oder 5, so wird <i>Anweisungsblock4</i> ausgeführt.

3.5.3 Kopf – Bedingungsgesteuerte Schleife → while-Schleife

Die while-Schleife wird nur dann und nur so lange ausgeführt, wie die Eingangsbedingung wahr ist. Sie wird verwendet, wenn erst geprüft werden soll, ob die *Bedingung* erfüllt ist oder die Anzahl der Durchläufe von vornherein nicht genau feststeht.

```
while(Bedingung) {
    Anweisungsblock;
}
```

3.5.4 (Kopf) – Zählergesteuerte Schleife → for-Schleife

Die for-Schleife wird verwendet, wenn eine Zählervariable gebraucht wird und/oder wenn die Anzahl der Durchläufe von vornherein feststeht. Die Schleife wird ausgeführt, wenn die Bedingung wahr ist und so oft ausgeführt, bis die Bedingung unwahr ist (deshalb: Ausgangsbedingung).

```
for(Startwert des Zählers; Wiederholungsbedingung; Zähleränderung) {
    Anweisungsblock;
}
```

<i>Startwert</i>	definiert die Zählervariable und legt den Startwert des Zählers fest: <code>var i = 0</code>
<i>Wiederholungsbedingung</i>	legt fest, wie oft die Schleife durchlaufen wird bzw. wann die Schleife abgebrochen wird: <code>i < 100</code>
<i>Zähleränderung</i>	definiert, wie der Zähler jeden Schleifendurchlauf Verändert werden soll: <code>i++</code> / <code>i--</code> / <code>i+=4</code>

Schleifendurchläufe beeinflussen

Schleifendurchläufe können durch bestimmte Schlüsselwörter beeinflusst werden.

continue;	überspringt den Rest des aktuellen Durchlaufs und beginnt den nächsten Durchlauf.
break;	bricht die Schleife komplett ab und führt das Programm nach der Schleife fort.

Beispiel

Wir wollen alle Zahlen zwischen 1 und 1000, die restlos durch 5 teilbar sind, addieren. Wir wollen die Schleife aber abbrechen, wenn die Summe größer als 400 ist.

```
var summe = 0;
for(var i = 1; i <= 1000; i++) {

    //wenn nicht restlos durch 5 teilbar,
    //Durchlauf überspringen
    if(i % 5 != 0) {
        continue;
    }
    //ab hier nur ausgeführt, wenn Rest 0
    //i wird zu summe addiert
    summe += i;

    if(summe > 400) {
        break; //Schleifenabbruch bei summe > 400
    }
}
```

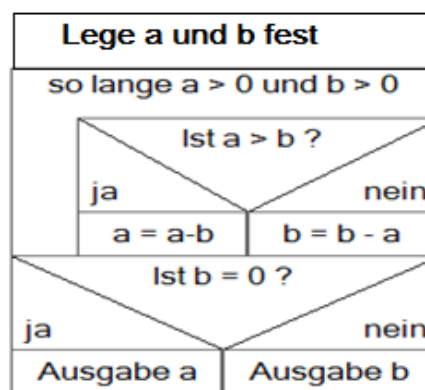
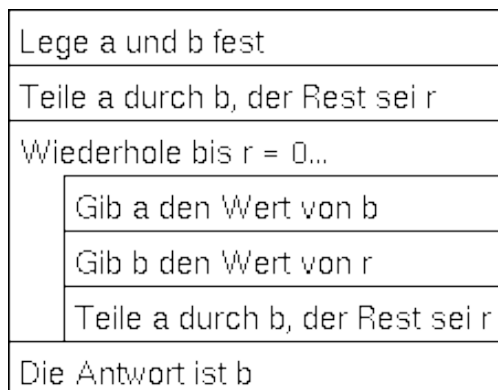
3.5.5 Fuß – (Bedingung) gesteuerte Schleife → do-while-Schleife

Die do-while-Schleife führt den *Anweisungsblock* immer ein Mal aus und prüft danach, ob die *Bedingung* erfüllt ist und die Anweisungen wiederholt ausgeführt werden sollen. Ansonsten verhält sie sich genau wie die while-Schleife.

```
do {
    Anweisungsblock;
} while(Bedingung);
```

Übungen:

- a) Implementiere in JavaScript das Beispiel 2 auf der Seite 16.
- b) Implementiere in JavaScript das Beispiel 2 auf der Seite 17.
- c) Implementiere in JavaScript das Übung b) auf der Seite 17.
- d) Gegeben sind zwei Algorithmen zur ggT-Berechnung



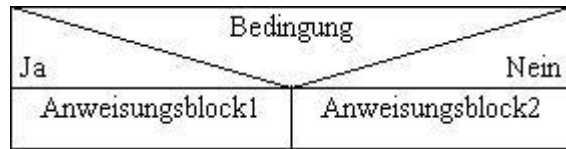
Implementiere beide in JavaScript und entscheide welcher besser ist.

- e) Implementiere in JavaScript das Beispiel 1 auf der Seite 19.
- f) Implementiere in JavaScript das Beispiel 2 auf der Seite 19.
- g) Implementiere in JavaScript das Übung C) auf der Seite 19.
- h) Implementiere in JavaScript das Beispiel 2 auf der Seite 20.

Übersicht: Kontrollstrukturen in JavaScript

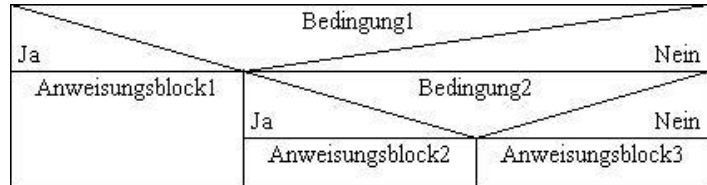
if-else-Verzweigung

```
if(Bedingung) {
    Anweisungsblock1;
} else {
    Anweisungsblock2;
}
```



Mehrfachverzweigung / Schachtelung:

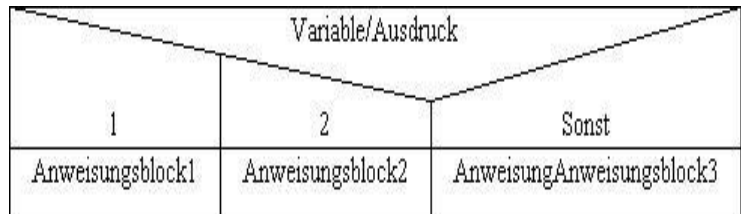
```
if(Bedingung1) {
    Anweisungsblock1;
} else if(Bedingung2) {
    Anweisungsblock2;
} else {
    Anweisungsblock3;
}
```



switch-Mehrfachverzweigung

```
switch(Variable/Ausdruck) {
    case 1 : Anweisungsblock1;
              break;
    case 2 : Anweisungsblock2;
              break;
    default: Anweisungsblock3;
}

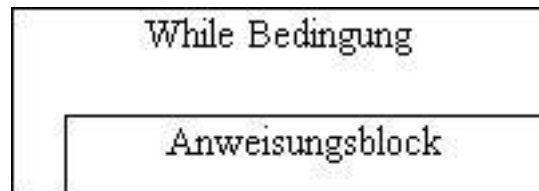
case      Einsprungpunkt
break;    Ausstiegspunkt
default   Standard-Anweisungen
```



while-Schleife

```
while(Bedingung) {
    Anweisungsblock;
}
```

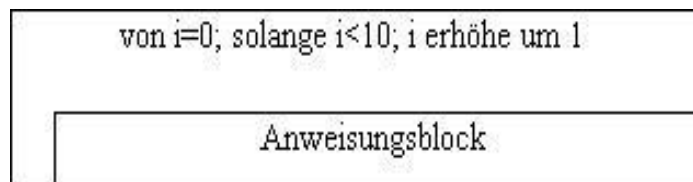
Schleifen beeinflussen
 continue; Durchlauf überspringen
 break; Schleife beenden



for-Schleife

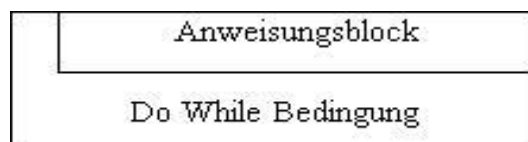
```
for(var i = 0; i < 10; i++) {
    Anweisungsblock;
}
```

var i = 0 Startwert
 i < 10 Ein-/Ausgangsbedingung
 i++ Zähleränderung



do-while-Schleife

```
do {
    Anweisungsblock;
} while(Bedingung);
```



4 Funktionen

JavaScript Funktionen geben dem Programmierer die Möglichkeit, ein Stück Code mit einem Namen zu versehen und ihn beim Bedarf mit diesem Namen immer wieder aufzurufen. Z.B.: Wir haben schon Quellcode für ggT Algorithmus geschrieben. Bei der Bruchrechnung (und auch bei der einigen Verschlüsselungsmethoden) muss diese Berechnung immer wieder durchgeführt werden. Eine vorgefertigten ggT – Funktion kann da sehr hilfreich sein.

4.1 Syntax der Definition einer Funktion

(Die fakultativen Teile einer Funktion werden hier rot und unterstrichen dargestellt)

```
function Funktionsname ( Parameter-1, ... , Parameter-n) {  
    Anweisungen;  
    return Wert oder Ausdruck;  
}
```

- Die Anweisung `function` leitet eine Funktion ein.
- `Funktionsname` ist der Bezeichner der Funktion, mit dem diese im weiteren Verlauf aufgerufen werden kann.
- In den runden Klammern () werden die `Parameter` angegeben, die einen beliebigen Typ besitzen können und deren Werte innerhalb der Funktion verarbeitet werden sollen. Die Klammern müssen in jedem Fall angegeben werden, auch wenn die Funktion keine Parameter erwartet.
- Zwischen den geschweiften Klammern { } befindet sich der Körper der Funktion, der aus dem eigentlichen Algorithmus besteht.
- Durch die Angabe des Schlüsselwortes `return` mit einem optionalen `Rückgabewert` kann eine Funktion an jeder Stelle verlassen werden.

Beispiel:

```
function ggT ( a, b ){  
    var z=a;  
    while(a%z!=0 || b%z!=0){  
        z--;  
    }  
    return z;  
}
```

4.2 Funktionsaufruf

Nach der Funktionsdefinition (normalerweise geschieht das im head-Teil des HTML-Dokumentes oder in einer separaten JavaScript-Datei) kann die Funktion per Funktionsaufruf ausgeführt werden. Funktionen werden mit den Namen aufgerufen, mit dem sie definiert sind. Wenn die Funktion aufgerufen wird, werden die internen Anweisungen abgearbeitet und Ergebnisse über das Schlüsselwort `return` dem Aufrufer übergeben.

4.3 Globale und lokale Variablen

JavaScript kennt, wie die meisten anderen Programmiersprachen, sowohl die globalen als auch lokalen Variablen. Unter globalen Variablen versteht man Variablen, die im ganzen Programm verfügbar sind. Im Gegensatz dazu sind lokale Variablen nur innerhalb eines bestimmten Bereichs definiert.

- Globale Variablen

Generell erzeugt man globale Variable dadurch, dass man eine Variable außerhalb einer Funktion oder eines mit Klammern begrenzten Anweisungsblocks mit dem Schlüsselwort `var` definiert. Damit ist die Variable und deren Inhalt für alle Programmteile verfügbar.

- Lokale Variablen

Definiert (deklariert) man eine Variable mit `var` innerhalb einer Funktion, dann erzeugt man eine lokale Variable, die nur in dieser Funktion Gültigkeit besitzt. Außerhalb dieser Funktion ist dann die Variable nicht bekannt. Nach Beendigung der Funktion kann man demnach nicht auf eine lokale Variable dieser Funktion zugreifen. (Der Speicherplatz wird freigegeben und anders verwendet.) Allgemein kann man sagen, dass eine mit Schlüsselwort `var` deklarierte Variable behält ihre Gültigkeit in dieser Umgebung, in der sie erzeugt (deklariert) wurde.

4.4 Übungen:

Folgende Aufgaben sollten nun mit Hilfe von Funktionen gelöst werden. Die Eingabe erfolgt dabei immer mittels Prompt-Fenster und die Ausgabe im HTML-Dokument.

a) Eingabe der Jahreszahl und Ausgabe in dem Format:

Jahr 2015 ist kein Schaltjahr

b) Eingabe zwei Zahlen und die schnelle ggT-Berechnung im Format:

ggT(18,24)=6

c) Übersetzung der Wochentage ins Englisch im Format:

Mittwoch --> Wednesday

d) Eingabe von Anfangskapital, Zinssatz und Laufzeit und Berechnung vom Endkapital im Format:

Endkapital: 2066 €

In den folgenden Aufgaben wird zunächst die Implementierung der einzelnen Funktionen im HEAD-Teil des HTML Dokumentes erwartet. Testen findet dann durch Funktionsaufruf im BODY-Teil statt.

e) Eine JavaScript Funktion **sterne(anzahl)** soll nach der Eingabe eines positiven int-Wertes *anzahl* (Parameter) *anzahl*-mal ein Sternchen im HTML-Dokument ausgeben.

Beispiel: Eingabe: 4 Ausgabe: ****

f) Eine Funktion **multisterne(anzahl)** sollte einen Sternhaufen erzeugen. Schreibe drei Versionen von diesem Programm **multiterne1**, **multisterne2**, **multisterne3** und zwar nach folgenden Regeln:

I. Eingabe: 4 Ausgabe: ****

II. Eingabe: 4 Ausgabe: *
 **

III. Eingabe: 4 Ausgabe: *****

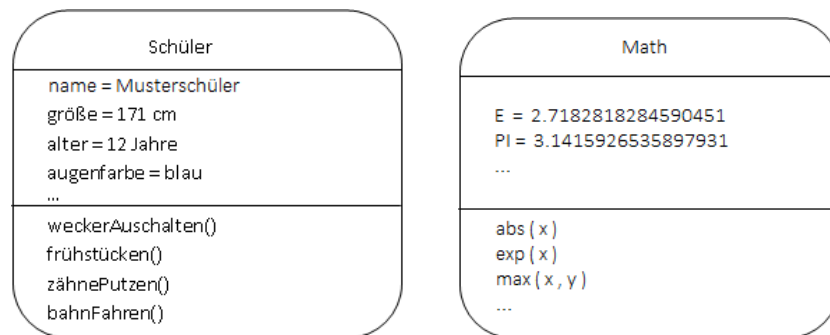
 **
 *

5 JavaScript als eine objektorientierte Programmiersprache

JavaScript ist eine moderne objektorientierte Programmiersprache. Bei einer objektorientierten Sprache redet man von Objekten, Eigenschaften und Methoden.

Objekte

Ein Objekt kann als Ding oder Sache betrachtet werden. Außerhalb der Programmierung ist es z.B. ein Schüler, ein Buch, ein Auto. Im Programm selbst ist es beispielsweise `Math` ein schon bekanntes Objekt oder `window` und `document`. Graphisch kann man Objekte in Form von Objektkarten darstellen. Oben schreibt man da Objektnamen gefolgt von den Eigenschaften (Attributen) mit Attributwerten und Methoden.



Eigenschaften (Attribute)

Objekte besitzen Eigenschaften (Attribute). Ein Schüler hat z.B. Name, Größe, Alter und Augenfarbe. Beim konkreten Schüler (einem Objekt) werden diese Attribute mit konkreten Werten belegt. (Hier Musterschüler, 171 cm, usw.).

Wenn man den Wert eines Attributes abfragen will, muss man schreiben: `Objektname.Attribut`

Wenn man z.B. `Schüler.augenfarbe` abfragt, erhält man `blau`

Wenn man den Wert verändern will, muss man schreiben:

`Objektname.Attribut = wert`

Wenn man die Augenfarbe des Schülers verändern will schreibt man z.B. `Schüler.augenfarbe = green.`

Methoden

Methoden sind schließlich die eigentlichen Aktionen, die ein Objekt ausführen kann. Ein Schüler schaltet den Wecker ab, eine Katze miaut, ein Auto fährt, Math-Objekt berechnet Maximum usw. Die Methoden werden stets durch runde Klammern gekennzeichnet. Wenn man eine Methode aufrufen will, muss man schreiben: `Objektname.methodenname()`

Wenn man z.B. will, dass unser Schüler Zähne putzt, schreibt man `Schüler.zähnePutzen()` .

Wenn man will, dass Math-Objekt uns Maximum von 3 und -7 berechnet, schreibt man `Math.max(3, -7)` .

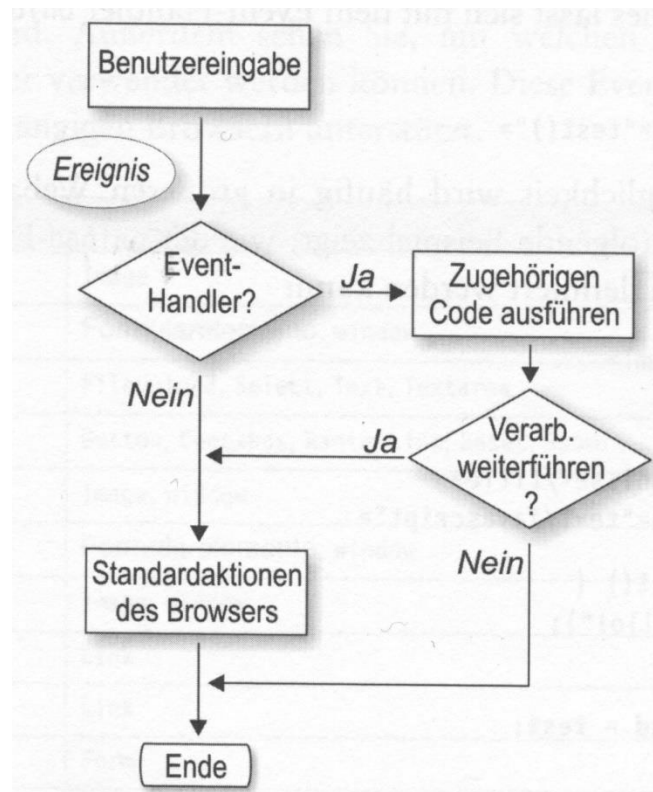
Zugriffsrechte

Bei der Arbeit mit den Objekten ist es manchmal sinnvoll bestimmte Attribute oder Methoden zu schützen. (Bei einem Objekt Bankkonto ist es z.B. nicht ratsam, dass jemand den Wert von Attribut `kontostand` ohne jegliche Berechtigung verändert.) In JavaScript ist es also möglich die Attribute und Methoden öffentlich `public` oder privat `private` zu programmieren.

6 JavaScript als eine ereignisgesteuerte Programmiersprache

JavaScript-Code kann als Reaktion auf **Ereignisse** ausgeführt werden. So ein Ereignis kann etwa ein Mausklick `onclick` des Anwenders oder das erfolgreiche Laden `onload` einer Webseite sein. Wie der Browser mit Ereignissen umgeht, wird durch das Ereignismodell festgelegt. In diesem Script wird nur das grundlegende Ereignismodell vorgestellt, das jeder JavaScript-fähige Browser versteht.

Funktionsweise



Die Funktionsweise von Ereignissen lässt sich am besten an einem Beispiel erklären. Stelle dir vor, der Anwender klickt auf ein Link-Objekt. Der Browser stellt fest, dass ein Link angeklickt wurde, und gibt eine Nachricht an das zuständige Programm mit der Information, was passiert ist. Das Programm entscheidet nun, ob und wie auf diese Nachricht bzw. dieses Ereignis reagiert werden soll. Dafür definiert der Programmierer einen sogenannten **Event-Handler**. In dem Event-Handler wird angegeben, wie das Programm auf bestimmte Ereignisse reagieren soll.

Der Code für dieses Beispiel könnte z.B. so aussehen:

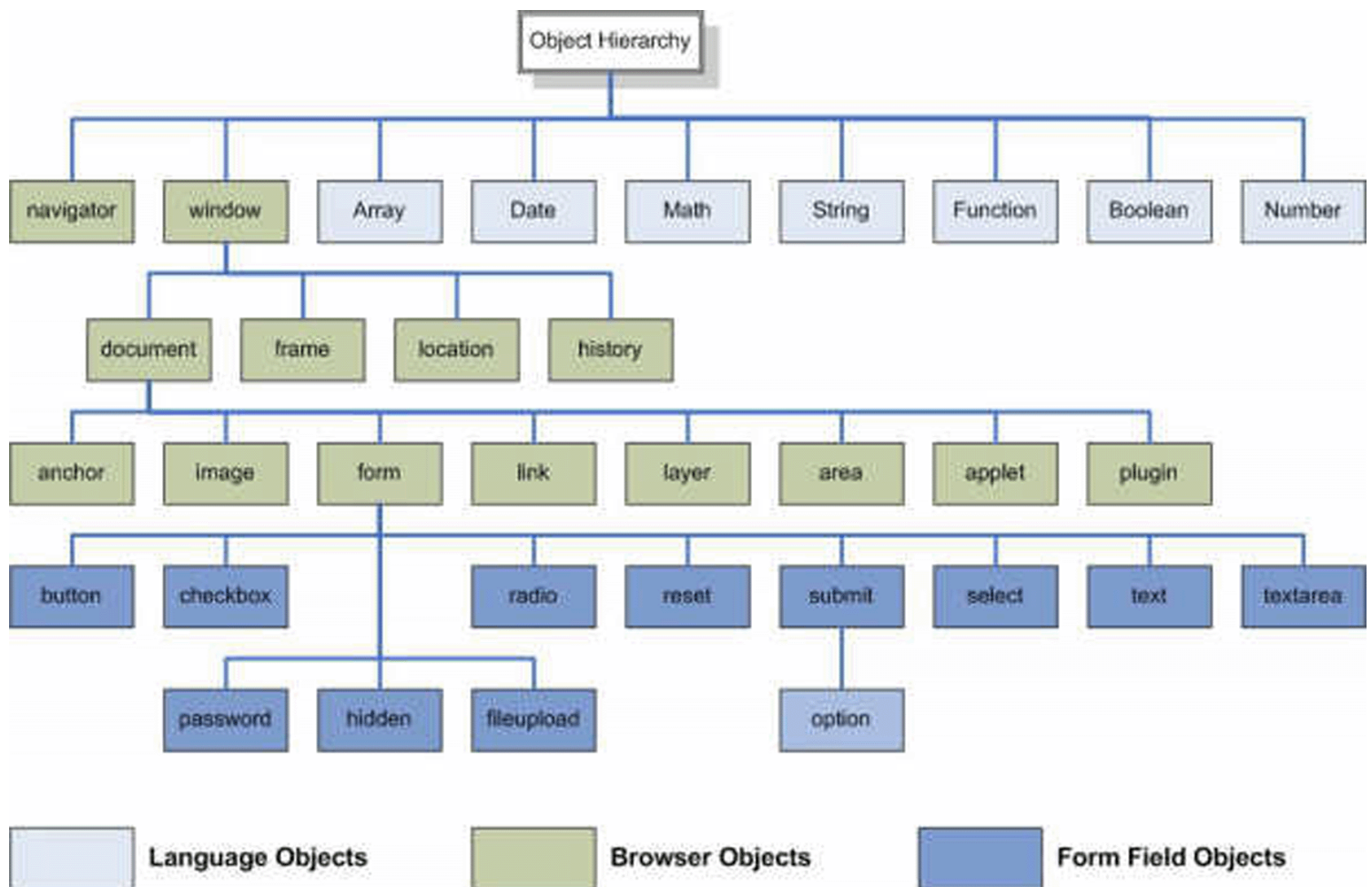
```
<a href="http://www.hjk-stinfeld.de" onclick="alert('Auf Wiedersehen!'); return false">Link<a>
```

Über das `<a>` Tag wird ein Link erzeugt. Mit `onclick` wird der Event-Handler festgelegt. Wenn der Anwender auf den Link klickt, wird das Ereignis `click` ausgelöst. Nun prüft der Browser, ob ein entsprechender Event-Handler definiert ist. Ist das der Fall, wird der dazugehörige JavaScript-Code ausgeführt. In unserem Beispiel wurde dieser Code direkt im HTML-Tag untergebracht. Hier öffnet sich ein `alert`-Fenster, wenn aus den Link geklickt wird. Sollen komplexere Aktionen durchgeführt werden, kann man an dieser Stelle auch eine Funktion aufrufen.

Die zweite JavaScript – Anweisung in unserem Beispiel (`return false`) legt fest, dass die Standardaktion des Browsers unterbunden wird. **Standardaktionen** werden vom Browser selbständig ausgeführt, wenn ein bestimmtes Ereignis eintritt. Klickt der Anwender auf einen Link, reagiert der Browser standardmäßig, indem eine neue HTML-Seite mit der im Link angegebenen Adresse geladen wird. Diese Aktion wird in unserem Beispiel blockiert.

7 Vordefinierte Objekte in JavaScript-Überblick

JavaScript verfügt über eine Menge von vordefinierten Objekten. Die einzelnen Objekte können wiederum mehrere andere Objekte beinhalten (so wie z.B. hat ein Auto einen Motor, mehrere Räder...) Hier ist nur ein kleiner Ausschnitt aus solchen Inklusions-Objekten-Hierarchie in JavaScript.



Die aufgezeichnete Inklusions-Objekten-Hierarchie wird in der Praxis oft bei der Ansteuerung der konkreten Objekten und ihrer Eigenschaften sichtbar. So z.B. wenn man im Dokument eine Eigenschaft eines Elementes innerhalb eines Formulars ansprechen will, kann man dazu folgenden Pfad nutzen:

```
document.Formularname.Elementname.Eigenschaft
```

Manche Objekte werden automatisch erzeugt (z.B. Math) Diese sind immer „ansprechbar“. Andere Objekte muss man zunächst selbst erzeugen. Um ein konkretes Objekt zu erzeugen bedient man sich normalerweise eines Konstruktors von dem man noch das Wort `new` steht. Konstruktor ist eine spezielle Methode, kann also auch noch Parameter besitzen. Hier noch ein paar Beispiele, bei denen unterschiedliche Objekte erzeugt und in einer Variable gespeichert werden.



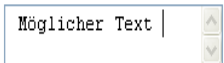

```
var a = new Date();
```

```
var b = new Date(jahr, monat, tag);
```

8 Formulare

Mithilfe der Formulare entsteht eine nützliche Möglichkeit eine Homepage interaktiv zu gestalten. Ein Formular kann man an einer beliebigen Stelle innerhalb einer Webseite definieren. Auf einer Webseite kann man auch mehrere anbieten. Wie man das an der Inklusions-Objekt-Hierarchie erkennen kann, bestehen Formulare aus mehreren Elementen. Formular-Objekte erzeugt man nicht mit Hilfe der Konstruktoren, sondern mittels HTML-Tags. Die Tags `<form>` und `</form>` bieten einen logischen „Rahmen“ eines Formulars.

Hier eine unvollständige Beschreibung der ausgewählten Form-(Teil)-Objekten

Element	Ausgewählten Eigenschaften/Methoden/Event-Handler
Grundaufbau <code><form></form></code>	Eigenschaften: <ul style="list-style-type: none"> name - Mit diesem Attribut legt man den eindeutigen Namen des Formulars fest. length - gib die Anzahl der in dem Formular beinhalten Elemente. Methoden: <ul style="list-style-type: none"> reset() - Setzt die in einem Formular getätigten Eingaben auf die Ausgangswerte zurück. Event-Handler: <ul style="list-style-type: none"> onreset - Gibt an, was passieren soll, wenn ein Formular zurückgesetzt werden soll.
Einzeiliges Eingabefeld <code><input type="text"></code> 	Eigenschaften: <ul style="list-style-type: none"> name - Mit diesem Attribut legt man den eindeutigen Namen des Formulars fest. size - Gib die sichtbare Größe des Objektes maxLength - Gib die maximale Eingabelänge value - Gib den Inhalt des Feldes. readonly - Erlaubt nur das Lesen aber nicht das Schreiben im Eingabefeld. Event-Handler: <ul style="list-style-type: none"> onchange - Gibt an, was passieren soll, wenn der Inhalt des Objektes geändert wurde.
Passwortfeld <code><input type="password"></code> 	Eigenschaften: <ul style="list-style-type: none"> name - Mit diesem Attribut legt man den eindeutigen Namen des Formulars fest. size - Gib die sichtbare Größe des Objektes maxLength - Gib die maximale Eingabelänge value - Gib den Inhalt des Feldes. Event-Handler: <ul style="list-style-type: none"> onchange - Gibt an, was passieren soll, wenn der Inhalt des Objektes geändert wurde.
Mehrzelliges Eingabefeld <code><textarea></code> Möglicher Text <code></textarea></code> 	Eigenschaften: <ul style="list-style-type: none"> name - Mit diesem Attribut legt man den eindeutigen Namen des Formulars fest. rows - Gib die Anzahl der Zeilen cols - Gib die Anzahl der Spalten value - Gib den eingegebenen String. Event-Handler: <ul style="list-style-type: none"> onchange - Gibt an, was passieren soll, wenn der Inhalt des Objektes geändert wurde.
Schaltflächen <code><input type="button"></code> 	Eigenschaften: <ul style="list-style-type: none"> name - Mit diesem Attribut legt man den eindeutigen Namen des Formulars fest. value - Gib die Beschriftung der Schaltfläche. Event-Handler: <ul style="list-style-type: none"> onclick - Gibt an, was passieren soll, wenn man auf die Schaltfläche klickt.

Berechnung Prozentsatz

Eingabe Grundwert in Euro	<input type="text"/>
Eingabe Prozentwert in Euro	<input type="text"/>
Berechnung Prozentsatz	<input type="text"/>
Ausgabe Prozentsatz in %	<input type="text"/>

Lösung der quadratischen Gleichung (p-q-Formel)

Eingabe p:	<input type="text"/>
Eingabe q:	<input type="text"/>
Lösung der Gleichung berechnen	<input type="text"/>
Diskriminante:	<input type="text"/>
Lösung x1:	<input type="text"/>
Lösung x2:	<input type="text"/>

Lineares Gleichungssystem mit zwei Variablen

$$\begin{aligned} a_1 \cdot x + b_1 \cdot y &= c_1 \\ a_2 \cdot x + b_2 \cdot y &= c_2 \end{aligned}$$

Eingabe a ₁ :	Eingabe b ₁ :	Eingabe c ₁ :
<input type="text"/>	<input type="text"/>	<input type="text"/>
Eingabe a ₂ :	Eingabe b ₂ :	Eingabe c ₂ :
<input type="text"/>	<input type="text"/>	<input type="text"/>
	Lösung LGA	<input type="text"/>
Lösung x:	<input type="text"/>	<input type="text"/>
Lösung y:	<input type="text"/>	<input type="text"/>

9 String-Object

Ein String-Objekt repräsentiert eine Zeichenkette.

Konstruktor

String([str])

Erzeugt einen String mit dem Inhalt str. Wird das Argument weggelassen, wird ein leerer String erzeugt.

Kurzschreibweise

Ein String kann mit folgender Kurzschreibweise erzeugt werden:

```
var x = "String";
```

Eigenschaften

length

Gibt die Länge der Zeichenkette an.

Methoden

charAt(x)

Liefert das Zeichen an der Position x des Strings. Die Nummerierung der Buchstaben fängt bei 0 an.

substring(position1 [, position2])

Liefert einen Teil des Strings zurück. Der Teilstring entspricht dem String von der Position position1 bis zu der Position position2. Das Zeichen an der Position position2 wird dabei nicht berücksichtigt. Wird position2 nicht angegeben, enthält der zurückgelieferte Teilstring alle Zeichen von position1 bis zum Ende des String-Objekts.

indexOf(zeichenkette [, startPosition])

Gibt die Position an, an der der String zeichenkette innerhalb des Strings das erste Mal vorkommt. Wird die Zeichenkette in dem String nicht gefunden, wird -1 zurückgeliefert. Als zweites Argument kann eine Zahl angegeben werden, die festlegt, ab welcher Position in dem String nach der Zeichenkette gesucht werden soll.

charCodeAt(x)

Liefert den Unicode-Zeichencode des Zeichens an der Position x des Strings zurück.

fromCharCode(num1 [, num2, ...])

Erzeugt anhand der übergebenen Unicode-Zeichencodes einen String und liefert diesen zurück.

toLowerCase()

Setzt jeden Buchstaben in dem String auf den entsprechenden Kleinbuchstaben.

toUpperCase()

Setzt jeden Buchstaben in dem String auf den entsprechenden Großbuchstaben.

9.1 Übungen:

- a) Probiere zunächst selbst die Funktionsweise der oben genannte Methoden von dem Objekt String. Benutze dazu die Seite http://www.w3schools.com/jsref/jsref_obj_string.asp
- b) Cäsar-Chiffre

Bei der Cäsar-Chiffre wird jedem Buchstaben eines Textes ein anderer eindeutiger Buchstabe zugeordnet. Diese Zuordnung ist allerdings nicht willkürlich, sondern basiert auf der zyklischen Drehung des Alphabets um k Zeichen, dabei folgt auf Z wieder A. Das k ist dann der Schlüssel, mit dem ver- bzw. entschlüsselt wird. Praktisch verschiebt man das Alphabet um k Zeichen (z.B. k=4):

Klartextalphabet:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
	
Geheimtextalphabet:	EFGHIJKLMNOPQRSTUVWXYZABCD

Zur Verschlüsselung wird nun für jeden Buchstaben aus dem Klartext der darunter stehender Buchstabe aus dem Geheimtext eingesetzt. Beim Entschlüsseln geht man umgekehrt vor und schreibt für jeden Buchstaben des Geheimtextes den entsprechenden Buchstaben des Klartextes. (Satz-, Leer- und Sonderzeichen werden nicht berücksichtigt.)

Mathematisch entspricht diese Verschlüsselung einer buchstabenweisen "Addition" der Schlüssel-Zahl k zu jedem Buchstaben des Klartextes. Entsprechend muss für die Entschlüsselung die Schlüssel-Zahl vom Geheimtext abgezogen werden, um wieder den Klartext zu erhalten.

Verschlüsseln:

KLARTEXT
+ 44444444
= OPEVXIBV

Um aus dem Geheimtext den Klartext zu erhalten, muss der Empfänger wissen, mit welchem Schlüssel k verschlüsselt wurde. Durch Umkehrung des Algorithmus - bei Benutzung des richtigen Schlüssels - ergibt sich dann wieder der Klartext.

Entschlüsseln:

OPEVXIBV
- 44444444
= KLARTEXT

Unten ist Quelltext für eine „spezielle Variante“ der Cäsar Verschlüsselung. Analysiere diesen Quelltext, erkläre wie es funktioniert, und nenne die Vorteile, die die spezielle Fassung gegenüber der originalen Fassung aufweist. Schreibe ein dazu passender Quelltext für die Entschlüsselung und nutze den Quelltext für die Erstellung eines kleinen Projektes.

```
function cesar_ver (key, text) {  
    var chiffre = "";  
    for (var i=0; i < text.length; i++) {  
        chiffre = chiffre + String.fromCharCode(text.charCodeAt(i)+key*1);  
    }  
    return chiffre;  
}
```

c) Vigenere-Verschlüsselung

Informiere dich zunächst im Internet, wie die polyalphabetische Verschlüsselung von Vigenere funktioniert. Mithilfe der unten abgebildeten Vigenere-Quadrat Verschlüssele das Wort GYMNASIUM mit dem Schlüssel: HJK.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Unten ist Quelltext für eine zu Vigenere analoge Verschlüsselung. Analysiere diesen Quelltext. Schreibe ein dazu passender Quelltext für die Entschlüsselung und nutze den Quelltext für die Erstellung eines kleinen Projektes.

```
function vigenere_ver (key, text) {
    var chiffre = "";
    for (var i=0; i < text.length; i++) {
        chiffre = chiffre + String.fromCharCode(text.charCodeAt(i)
                                                    + key.charCodeAt(i%key.length));
    }
    return chiffre;
}
```

Palindrom

- d) **Palindrom** sind Zeichenketten, die von vorn und von hinten gelesen dasselbe ergeben. Schreibe ein Programm, dass zunächst die angegebene Zeichenkette umdreht, und dann entscheidet, ob die angegebene Kette ein Palindrom ist, oder nicht

Eingabetext

Ausgabertext

Vergleich