Informatik 5ai Schuljahr 2023/2024 als PDF exportieren

# Informatik 5. Klasse - Schuljahr 2023/24

## Lehrplan

- Lehrplaninhalte

## Themengebiete

- 1) Zahlensysteme in der Informatik
- 2) Informationseinheiten in der Informatik
- 3) Zeichencodierung
- 4) Schaltalgebra
- 5) Algorithmik und Programmierung
- 6) Tabellenkalkulation
- 7) Web Design Grundlagen (HTML, CSS)

## Leistungsbeurteilung

### 1/3 - Test (SA)

- 2x Tests pro Semester
  - 1. Test – 5ai – Do, 09.11.2023 - Themengebiet 1-4
  - 2. Test – 5ai – Do, 04.12.2023 - Themengebiet 5
  - 3. Test – 5ai – Do, 07.03.2024 - Themengebiet 5.8
  - 4. Test – 5ai – Do, 16.05.2024 - Themengebiet 6-7

### 1/3 - Mitarbeit (MA)

- Aktive Mitarbeit im Unterricht (aMA)
- Mündliche Stundenwiederholungen (mMA)
- Schriftliche Stundenwiederholungen (sMA)

### 1/3 - Praktische Arbeiten (PA)

- 1x praktischer Arbeitsauftrag pro Woche via Google Classroom

## Leistungsstand

Den aktuellen Leistungsstand könnt ihr jederzeit einsehen!

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324**

Last update: **2024/05/06 06:19**

# Themengebiete

## Funktionsweise eines Computers

1. Zahlensysteme
    1. binär
    2. oktal
    3. dezimal
    4. hexadezimal
    5. Anwendungen (Farbcodes, IP-Adresse, Informationseinheiten, MAC-Adresse, Zeichencodierung..)
2. Informationseinheiten
    1. Bit, Byte, …
3. Zeichencodierung
    1. ASCII
    2. Unicode
4. Schaltalgebra
    1. Grundlagen
    2. Grundschaltungen
    3. zusammengesetzte Schaltungen
    4. digitale Rechenschaltungen

## Hardware

1. EVA-Prinzip
2. Von-Neumann-Architektur
3. Komponenten
4. Schnittstellen

## Betriebssysteme

1. Windows
    1. Client
    2. Server
    3. Domänen (DC-Controller)
    4. Active Directory
    5. DNS
    6. CMD
    7. Powershell
2. Linux
    1. Befehle
    2. Prozesse
    3. Rechte
    4. Oberflächen

# Virtualisierung

1. Definition
2. Funktionsweise
3. Arten

# Algorithmen und Datenstrukturen

1. Grundlagen
    1. Begriffsdefinitionen
    2. Programmiersprachen
    3. Compiler und Interpreter
    4. Visualisierung von Algorithmen
2. C++
    1. Grundgerüst
    2. Variablen
    3. Verarbeitung
    4. Ein- und Ausgabe
    5. Ablaufsteuerung
        1. Verzweigungen
        2. Schleifen
        3. Funktionen
            1. Parameter
            2. Überladen von Funktionen
            3. Geltungsbereich von Variablen
            4. Rekursionen
    6. Datentypen und -strukturen
        1. Arrays
        2. Zeiger und Adressen
        3. Variablenverbund struct
        4. Listen
        5. Stack
        6. Baum
    7. Sortieren & Suchen
    8. Klassen
        1. Objekt
        2. Zugriffsrechte
        3. Konstruktor & Destruktor
        4. Überalden
        5. Vererbung

# Datenschutz und -sicherheit

1. Passwörter
2. Urheberrecht
3. E-Commercegesetz
4. Cloud

# Kryptographie

1. Steganographie
2. Ziele
3. Methoden
4. Anwendungsgebiete

# Netzwerke

1. Grundlagen
2. Topologien
3. Übertragungsmedien
4. OSI-Schichtenmodell
5. Komponenten
6. Adressierung
7. Routing
8. Protokolle
9. Netzwerksimulation

# Anwendungssoftware

1. Tabellenkalkulation
2. Textverarbeitung
3. Präsentationen
4. Bildbearbeitung
5. Audiobearbeitung
6. Videobearbeitung

# Webdesign & -programmierung

1. Schichtenmodell
2. DOM
3. HTML
    1. Grundgerüst
    2. Elemente
        1. Überschriften
        2. Paragraphe
        3. Links
        4. Bilder
    3. Attribute
    4. Überschriften
    5. Listen
    6. Tabellen
4. CSS
    1. Grundlagen
    2. Deklarationen

    3. Vererbung und Kaskadierung
    4. Box-Modell
5. Javascript
6. Bootstrap als Framework
7. PHP
8. CMS-Systeme

## Datenbanken

1. Grundlagen
2. Datenmodellierung
    1. ER-Modell
    2. Relationenmodell
    3. Normalformen
3. SQL als Datenbanksprache

## 3D-Modellierung & 3D-Druck

## Geschichte der Informatik & künstliche Intelligenz

## Projektmanagement

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:0_lehrplaninhalte**

Last update: **2023/09/10 12:44**

# 1) Zahlensysteme in der Informatik

- Kurzüberblick - Geschichtliche Entwicklung des Zählens
- Skriptum zur Einführung
- Informationseinheiten in der Informatik

---

- 1.00) Dezimalsystem
- 1.01) Dualsystem
- 1.02) Hexadezimalsystem
- 1.03) Oktalsystem
- 1.04) Umrechnungen

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:01_zahlensysteme**

Last update: **2023/09/10 12:52**

# 1.00) Dezimalsystem

Das dezimale Zahlensystem kennzeichnet die Verwendung von zehn verschiedenen Ziffern innerhalb eines Stellenwertsystems. Eine Stelle kann jeweils einen Wert von 0 bis 9 annehmen. Damit war erstmals ein einfaches und schnelles Rechnen möglich.

Die Grundlagen des Dezimalsystems sind aus der Tatsache hervorgegangen, dass wir Menschen jeweils 10 Finger und 10 Zehen haben. Da ist es naheliegend, dass wir auf der Basis von 10 rechnen. So beruhen viele Zahlensysteme auf einer natürlichen Gliederung, die sich durch die fünf Finger einer Hand, die 10 Finger beider Hände oder die sogar insgesamt 20 aus Finger und Zehen ergeben.

Bei der Analyse der Sprache primitiver Völker entdeckten Forscher, dass deren Zahlworte meist bei Zwanzig enden. Die heutigen Sprachen, wie z. B. Englisch, Deutsch und Französisch kenne für Zahlworte bis Zwanzig eigenständig Bezeichnungen. Zum Beispiel Elf (Eleven), Zwölf (Twelf), Dreizehn (Thirteen), Vierzehn (Fourteen).

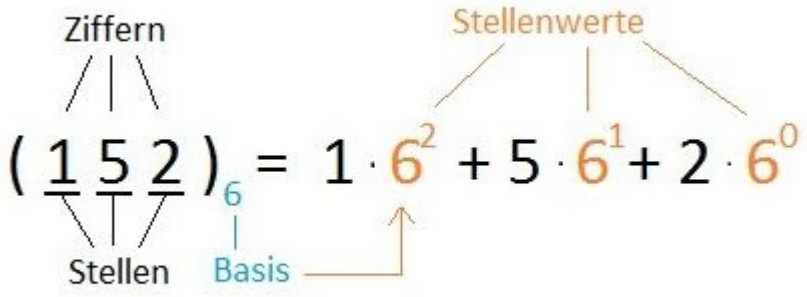
Basis: 10

Nennwerte: 0 1 2 3 4 5 6 7 8 9

Größter Nennwert: 9

Stellenwerte:

```
10^0 = 1
10^1 = 10
10^2 = 100
...
```




Man kann Zahlen auch anhand ihrer Basis darstellen. Im Dezimalsystem haben wir 10 Zahlen zur Verfügung, von 0 bis 9. Mit 2 Stellen können wir also 10 * 10 = 100 Zahlen darstellen. 100 Zahlen? Aber 100 hat doch drei Stellen! Dieser Einwand stimmt. Da wir jedoch mit der Zahl 0 beginnen, ist 0 die 1. Zahl, 1 die 2. Zahl, … 98 die 99. Zahl und 99 die 100. Zahl.

Mit 3 Stellen können wir 10 * 10 * 10 = 1000 Zahlen darstellen. Jede Stelle entspricht einer 10-er Potenz.

An einem einfachen Beispiel versuche ich diesen Sachverhalt zu erklären.

Wir nehmen dazu die Zahl 372 und schreiben sie als kleine Rechnung auf:

372 = 3*100 + 7*10 + 2*1.

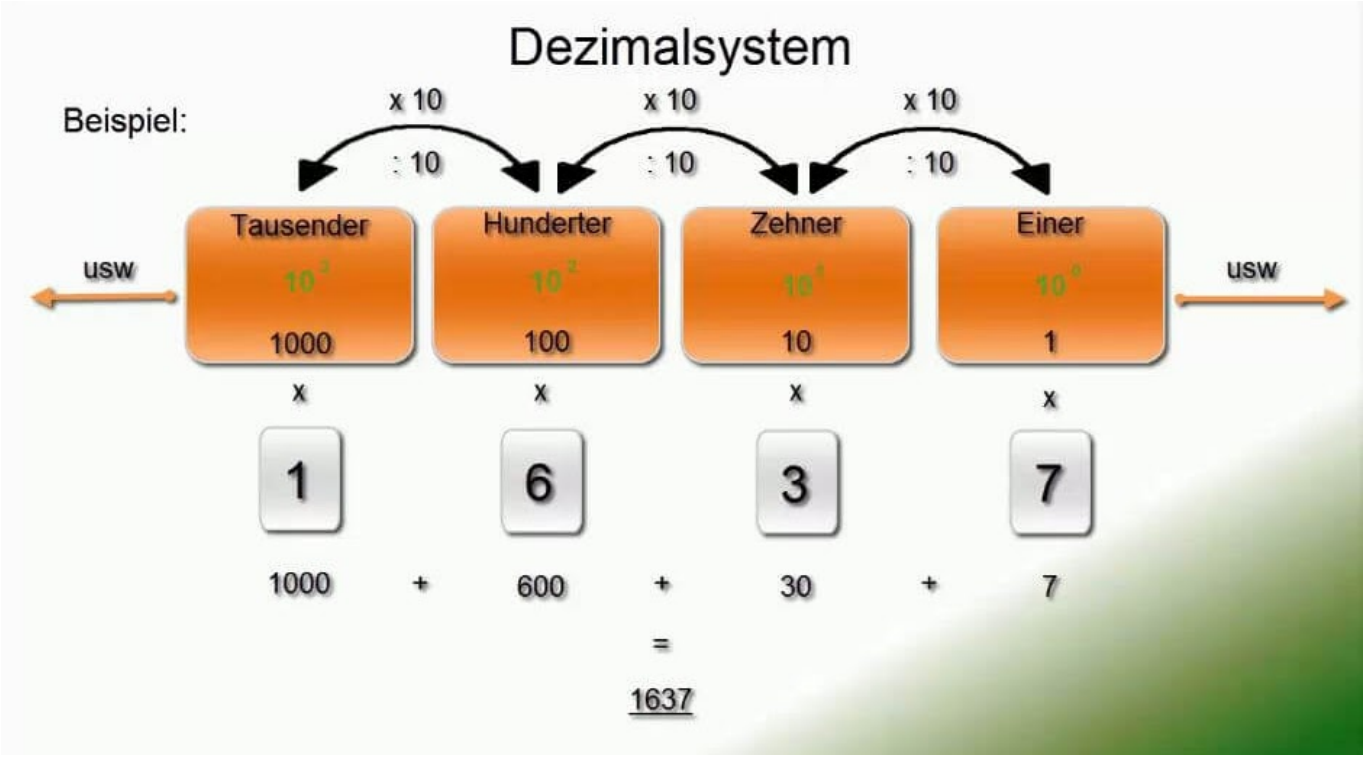
Das kann man jetzt noch in Potenzschreibweise darstellen als:

372 = 3*102 + 7*101 + 2*100.

Auf diese Weise kann man jetzt alle anderen Zahlen auch darstellen:

6574 = 6*103 + 5*102 + 7*101 + 4*100

12032 = 1*104 + 2*103 + 0*102 + 3*101 + 2*100




From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:01_zahlensysteme:1_00**

Last update: **2023/09/10 12:55**

# 1.01) Dualsystem

Das **duale Zahlensystem - auch Dualsystem oder Binärsystem** genannt - besteht aus **2 Ziffern**, gekennzeichnet durch 0 und 1. Man benötigt dieses Zahlensystem in der Informatik, da sich mit technischen Bauteilen sehr leicht die Zustände AN und AUS erzeugen lassen können. Diese Zahlen können entsprechend unserem „normalen“ Dezimalsystem verwendet werden. Man kann sie addieren, subtrahieren, multiplizieren und dividieren. Da sie sich also kaum vom „normalen“ Rechnen unterscheiden, eignen sie sich hervorragend, um in der EDV eingesetzt zu werden.


## Zählen im Dualsystem

Auch hier beginnen wir mit 0 und zählen dann 1. Leider haben wir nur 2 Zahlen, also gehen uns hier die Zahlen schnell aus. Wir machen es jetzt aber genau wie im Dezimalsystem und nehmen eine Stelle dazu. Nach 0 und 1 kommen dann also 10 und 11. Wieder reichen die Stellen nicht! Also noch eine dazu: 100, 101, 110, 111, usw.

## Zählen von 0 bis 15 im Dezimal- und Dualsystem

| Dezimal | Dual |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 10 |
| 3 | 11 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| 8 | 1000 |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |

**Hilfstabelle: Zweierpotenzen**

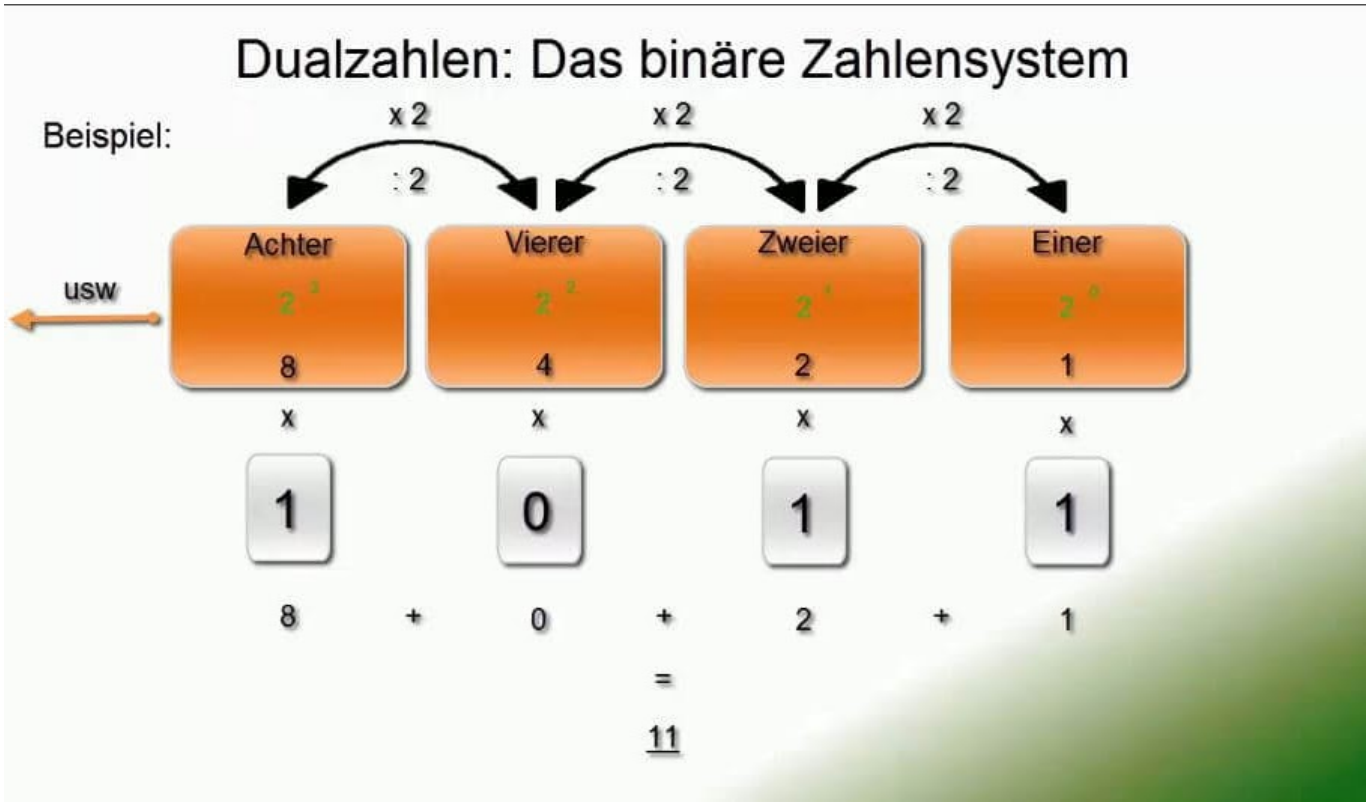
| n | $2^n$ | n | $2^n$ |
|---|---|---|---|
| 0 | 1 | 11 | 2 048 |
| 1 | 2 | 12 | 4 096 |
| 2 | 4 | 13 | 8 192 |
| 3 | 8 | 14 | 16 384 |
| 4 | 16 | 15 | 32 768 |
| 5 | 32 | 16 | 65 536 |
| 6 | 64 | 17 | 131 072 |
| 7 | 128 | 18 | 262 144 |
| 8 | 256 | 20 | 1 048 576 |
| 9 | 512 | 24 | 16 777 216 |
| 10 | 1 024 | 32 | 4 294 967 296 |

## Verwendung der Potenzschreibweise bei binären Zahlen

Wendet man die Potenzschreibweise bei binären Zahlen an, so muss man eine andere Basis wählen. Es gibt ja nur 2 verschiedene Ziffern, 0 und 1. Also nehmen wir als Basis 2.

Die Zahl 1011 schreibt sich dann als

$1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$




## Die Umrechnung von dezimalen in binäre Zahlen

Bei der Umrechnung der Dezimalzahlen verwenden wir die „Division mit Rest“ aus der Grundschule. Wir teilen die Zahl solange durch 2, bis als Ergebnis 0 herauskommt und merken uns dabei den Rest. Als Beispiel sollen die Zahlen 13 und 14 dienen.

13 / 2 = 6 Rest 1
6 / 2 = 3 Rest 0
3 / 2 = 1 Rest 1
1 / 2 = 0 Rest 1

Die Reste von unten nach oben aneinander gereiht ergeben dann die Dualzahl 1101.

14 / 2 = 7 Rest 0
7 / 2 = 3 Rest 1
3 / 2 = 1 Rest 1
1 / 2 = 0 Rest 1

Hieraus ergibt sich dann die Dualzahl 1110.

## Addition von Dualzahlen

Einige erinnern sich vielleicht noch an die Addition von Zahlen wie wir sie in der Grundschule gelernt haben. Wir schreiben die Zahlen untereinander und addieren sie Stelle für Stelle. Dabei beginnen wir mit der letzten Stelle und arbeiten uns langsam nach vorne durch. Die gleiche Vorgehensweise benutzen wir nun auch wenn wir Dualzahlen addieren wollen.

Die Addition funktioniert wie bei der Addition von Dezimalzahlen:

```
 0111
+0100
 ====
 1011
```

Addition mit Überlauf:

```
 1111
+0100
 ====
10011
```

Es gelten die Regeln 0+0=0, 1+0=1, 0+1=1, 1+1=0 Übertrag 1. Im Prinzip also nichts neues. Addiert man im Dezimalsystem 2 Zahlen so kommt bei 5+5 auch 0 Übertrag 1 heraus. Der Übertrag wird bei beiden System jeweils voran gestellt. Also gilt im Dualsystem 1+1=10.

#### Vorsicht Überlauf!

Die Addition ist im Prinzip problemlos. Es gibt allerdings einen Haken an der Sache: Die Addition funktioniert nur innerhalb eines bestimmten Wertebereiches. Woran liegt das? In der Realität können wir beliebig grosse Zahlen darstellen. Das geht leider nicht in der Informatik. Wir haben nur einen begrenzten Raum bzw. Speicherplatz zur Verfügung. Wir müssen aus diesem Grund den Speicherbereich einschränken (siehe was ist ein Byte), in unserem Beispiel nehmen wir als Speicherplatz ein Byte. Normalerweise verwendet man zur Addition von ganzen Zahlen, einen deutlich größeren Wertebereich, aber um einen überschaubaren Rahmen zu haben, begrenzen wir uns absichtlich auf ein Byte. Unsere größte darstellbare Zahl ist die 11111111, also dezimal 255. Was passiert nun, wenn wir eine 00000001, also 1, addieren? Das verrückte ist: es kommt 00000000, also 0 heraus. Da bekanntlich 1+1=0 Übertrag 1 gibt, bekommt man als Ergebnis in Wirklichkeit nicht 00000000 sondern 00000000 Übertrag 1. Dieser letzte Übertrag kann jedoch nicht mehr gespeichert werden und wird deshalb einfach ersatzlos gestrichen. Es ergibt sich daraus jedoch auch eine gewisse Logik. Durch meinen beschränkten Wertebereich komme ich irgendwann an meine obere Grenze. Bei der Addition von 1 fängt dann jedoch der Wertebereich wieder von vorne an, ich bin jetzt an der unteren Grenze, man durchläuft dann wieder den Bereich bis zur oberen Grenze, usw. Wenn wir also immer und immer wieder 1 addieren zählen wir unendlich oft von 0 bis 255, dann wieder 0 bis 255 usw.

## Subktraktion von Dualzahlen

#### Drei Schritte zu Subtraktion

Hier kommen wir mit unserer normalen Schulmathematik nicht mehr weiter. Bevor wir uns mit dem komplizierten „Warum ist das denn so?“ beschäftigen, merken wir uns erst einmal den Mechanismus. Die Subtraktion von binären Zahlen wird durch die **Addition des Zweierkomplementes** durchgeführt. Zur Erklärung beginnen wir im ersten Schritt mit dem **Einerkomplement**, dann schauen wir im zweiten Schritt was das **Zweierkomplement** ist und dann kommen wir im letzten Schritt zur **Subtraktion**.

### Das Einerkomplement

Was ist das Komplement von Dualzahlen? Man bildet das sogenannte Einerkomplement, indem man jede Zahl durch ihr Gegenteil ersetzt, also die 0 durch die 1 und die 1 durch die 0.

01011010 wird zu 10100101 11101101 wird zu 00010010

### Das Zweierkomplement

Das Zweierkomplement entspricht dem Einerkomplement, nur wird zusätzlich noch 00000001 addiert.

01011010 wird im Einerkomplement zu 10100101 im Zweierkomplement zu 10100110 11101101 wird im Einerkomplement zu 00010010 im Zweierkomplement zu 00010011

### Die Subtraktion von Dualzahlen

Der Satz lautet: Die Subtraktion von 2 Zahlen erfolgt durch die Addition des Zweierkomplementes. Als konkretes Beispiel nehmen wir dazu die Rechnung 14-9=5.

**!!WICHTIG:!!**
**Die restlichen Ziffern müssen immer mit 0 aufgefüllt werden.**

9 ist im Dualsystem 00001001.

Das Einerkomplement zu 00001001 ist 11110110.

Das Zweierkomplement 11110111.

Dies addieren wir nun zu 14 also 00001110.

```
 00001110
+11110111
 ========
 00000101
```

Auch hier wäre die richtige Zahl eigentlich 00000101 Übertrag 1, da wir den Übertrag jedoch nicht speichern können, bleiben wir bei 00000101 was ja der Dezimalzahl 5 entspricht.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:01_zahlensysteme:1_01**

Last update: **2023/09/10 12:56**

# 1.02) Hexadezimalsystem

Besonders **wichtig** ist in der **Informatik und Digitaltechnik** neben dem Binärsystem auch das **Hexadezimalsystem (Sedezimalsystem)**. Das Hexadezimalsystem verwendet die **Basis 16**, d.h. es gibt **16 verschiedene Ziffern**, **0 bis 9** und zusätzlich die Buchstaben **A bis F** (sog. Zahlzeichen; können auch als klein geschrieben werden: a-f).

Mit dem Hexadezimalsystem können auf einfachere und kürzere Weise Binärzahlen notiert werden. Mit einer 4-stelligen Binärzahl (auch als Halbbyte oder Nibble bezeichnet) lassen sich 16 (24 = 16) verschiedene Zahlen darstellen, und zwar 0 bis 15 (die Null zählt mit!). Da das Hexadezimalsystem die Basis 16 (= 24) verwendet, reicht eine (!) Hexadezimalzahl aus, um vier Bits (Binärziffern) darzustellen. Mit zwei Hexadezimalzahlen kann ein Byte (8 Bits) angeschrieben werden.

## Gegenüberstellung Hexadezimal-, Binär- und Dezimalsystem

| Hex | Binär | Dezimal |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| A | 1010 | 10 |
| B | 1011 | 11 |
| C | 1100 | 12 |
| D | 1101 | 13 |
| E | 1110 | 14 |
| F | 1111 | 15 |

Um eindeutig darauf hinzuweisen, dass es sich um eine Hexadezimalzahl handelt, kann ebenso wie in anderen Zahlensystemen die Basis tiefgestellt dazu geschrieben werden, z.B. $\mathbf{3F_{16}}$ (= $63_{10}$ dezimal) oder $\mathbf{93_{16}}$ (= $147_{10}$ dezimal). Es sind aber auch andere Schreibweisen üblich:

a) Vorangestelltes **0x (Prefix), z.B. 0x93**. Diese Notation wird in Programmiersprachen mit C-ähnlicher-Syntax verwendet.

b) Nachgestelltes **h** (Postfix), z.B. **93h**. Letztere Schreibweise ist besonders in der Technik gebräuchlich.

# Umrechnung vom Dezimal- ins Hexadezimalsystem

Die Umrechnung funktioniert ähnlich der Umrechnung von Dezimal- zu Binärzahlen (s.o.). Nun muss aber, statt durch 2, durch 16 dividiert werden. Die Reste werden genauso von rechts nach links angeschrieben und geben, wenn das Ergebnis der Ganzzahlendivision 0 ist, das Endergebnis.

Beispiel: Die Dezimalzahl **$304_{10}$** soll in eine Hexadezimalzahl umgewandelt werden.

```
304 / 16 = 19 => 0 Rest
 19 / 16 =  1 => 3 Rest
  1 / 16 =  0 => 1 Rest
```

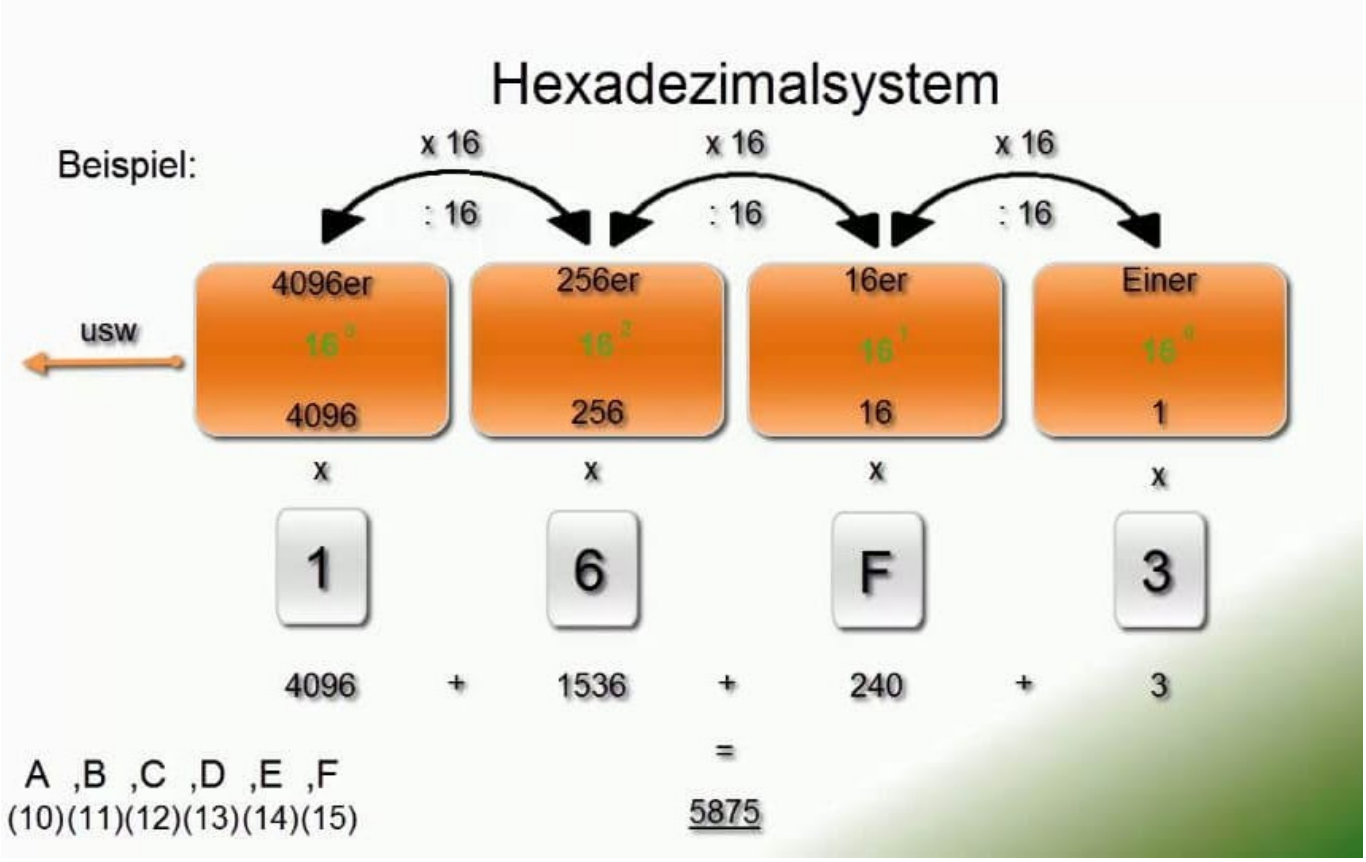
Für das Endergebnis werden jetzt die Reste **von unten nach oben gelesen**.
Somit ergibt sich ein Endergebnis von $130_{16}$, das entspricht der Dezimalzahl **$304_{10}$**.

# Umrechnung vom Hexadezimal- ins Dezimalsystem

Die Umrechnung vom Hexadezimal- ins Dezimalsystem kann genauso wie oben von Binär→Dezimal demonstriert, erfolgen. Die einzelnen Ziffern werden mit dem jeweiligen Stellenwert ($16^n$, wobei n = 0, 1, 2, ...) multipliziert und die jeweiligen Ergebnisse aufsummiert. Das folgende Beispiel demonstriert dies anhand der Hexadezimalzahl $130_{16}$:

```
0 * 16^0  =    0
3 * 16^1  =   48
1 * 16^2  =  256
---------------
          =  304
```

Als Ergebnis erhalten wir 304 dezimal, womit die Probe - zur vorigen Rechnung in die umgekehrte Richtung - erfolgreich war. $304_{10}$ entspricht $130_{16}$. Diese Antwort hätte in der Praxis natürlich auch ein wissenschaftlicher Taschenrechner geliefert.  Es reicht dazu sogar der **Windows-Rechner** (den Sie nur auf die wissenschaftliche Ansicht umstellen müssen) oder unter Linux Programme wie z.B. KCalc.




From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:01_zahlensysteme:1_02**

Last update: **2023/09/10 12:58**

# 1.03) Oktalsystem

Das Oktalsystem, auch Achtersystem genannt, verwendet die **Basis 8 (acht)**. Um Zahlen darzustellen, stehen die **Ziffern 0 bis 7** zur Verfügung. Die Bedeutung in der Informatik/Digitaltechnik ergibt sich dadurch, dass sich mit einer **Oktalzahl drei Bits** darstellen lassen. 23 ist 8, somit lassen sich mit 3 Bits 8 verschiedene Möglichkeiten darstellen. Eine Oktalzahl reicht, um diese Information wiederzugeben.

Das Oktalsystem wird hier insbesondere deshalb erwähnt, weil in vielen Programmiersprachen Zahlen auch in Oktalform angegeben werden können. Meist, z.B. in PHP, wird dazu eine 0 (Null) vorangestellt, z.B. 077 für $77_8$ (= $63_{10}$).

Umrechnungen erfolgen genauso wie beim Hexadezimalsystem gezeigt. Um die Oktalzahl auszurechnen, die einer best. Dezimalzahl entspricht, dividieren Sie die Dezimalzahl fortlaufend durch 8 und schreiben die Reste von rechts nach links an. In umgekehrter Richtung - von Oktal nach Dezimal - multiplizieren Sie die einzelnen Ziffern mit dem Stellenwert ($8^n$ für n = 0, 1, 2, ...) und addieren die Teilergebnisse.

## Umrechnung vom Dezimal- ins Oktalsystem

Die Umrechnung funktioniert ähnlich der Umrechnung von Dezimal- zu Binärzahlen (s.o.). Nun muss aber, statt durch 2, durch 8 dividiert werden. Die Reste werden genauso von rechts nach links angeschrieben und geben, wenn das Ergebnis der Ganzzahlendivision 0 ist, das Endergebnis.

Beispiel: Die Dezimalzahl **$304_{10}$** soll in eine Hexadezimalzahl umgewandelt werden.

```
304 / 8 = 38 => 0 Rest
 38 / 8 =  4 => 6 Rest
  4 / 8 =  0 => 4 Rest
```

Für das Endergebnis werden jetzt die Reste **von unten nach oben gelesen**.
Somit ergibt sich ein Endergebnis von $460_8$, das entspricht der Dezimalzahl **$304_{10}$**.

## Umrechnung vom Oktal- ins Dezimalsystem

Die Umrechnung vom Oktal- ins Dezimalsystem kann genauso wie oben von Binär→Dezimal demonstriert, erfolgen. Die einzelnen Ziffern werden mit dem jeweiligen Stellenwert ($8^n$, wobei n = 0, 1, 2, ...) multipliziert und die jeweiligen Ergebnisse aufsummiert. Das folgende Beispiel demonstriert dies anhand der Oktalzahl $460_8$:

```
0 * 8^0  =    0
6 * 8^1  =   48
4 * 8^2  =  256
---------------
         =  304
```

Als Ergebnis erhalten wir 304 dezimal, womit die Probe - zur vorigen Rechnung in die umgekehrte Richtung - erfolgreich war. $304_{10}$ entspricht $460_{8}$. Diese Antwort hätte in der Praxis natürlich auch ein wissenschaftlicher Taschenrechner geliefert. Es reicht dazu sogar der **Windows-Rechner** (den Sie nur auf die wissenschaftliche Ansicht umstellen müssen) oder unter Linux Programme wie z.B. KCalc.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:01_zahlensysteme:1_03**

Last update: **2023/09/10 12:58**

# Umrechnungen

Prinzipiell kann man jede Zahl von einem Zahlensystem in ein anderes Zahlensystem umrechnen. Dazu muss man normalerweise die Zahl immer zuerst in das Dezimalsystem und dann in das Ziel-Zahlensystem umrechnen. Ausnahmen gibt es bei Umrechnungen vom Binärsystem in ein anderes Zahlensystem, hier braucht man nicht unbedingt das Dezimalsystem.

## Binärsystem ⇔ Oktalsystem

### Binärsystem ⇒ Oktalsystem

Wir wissen ja bereits, dass eine Ziffer im Oktalsystem die Ziffern 0-7 annehmen kann und dass man 3 Bits im Dualsystem braucht um 8 verschiedene Zahlenwerte darzustellen.

**⇒ Immer 3 Stellen im Binärsystem ergeben eine Ziffer/Stelle im Oktalsystem**

#### Beispiel

Umwandlung der Zahl $010110011111_2$ ins Hexadezimalsystem:

```
010 110 011 111
 ⇓   ⇓   ⇓   ⇓
 2   6   3   7
```

Die Zahl $010110011111_2$ entspricht somit der Zahl $2637_8$.

### Oktalsystem ⇒ Binärsystem

Selbiges gilt auch für die Umrechnung vom Oktalsystem in das Binärsystem.

**⇒ Eine Stelle im Oktalsystem entspricht 3 Stellen im Binärsytem**

#### Beispiel

Umwandlung der Zahl $672_8$ ins Binärsystem:

```
 6   7   2
 ⇓   ⇓   ⇓
110 111 010
```

Die Zahl $672_8$ entspricht somit der Zahl $110111010_2$.

# Binärsystem ⇔ Hexadezimalsystem

## Binärsystem ⇒ Hexadezimalsystem

Wir wissen ja bereits, dass eine Ziffer im Hexadezimalsystem die Werte 0-15 annehmen kann und dass man 4 Bits im Dualsystem braucht um 16 verschiedene Zahlenwerte darzustellen.

**⇒ Immer 4 Stellen im Binärsystem ergeben eine Ziffer/Stelle im Oktalsystem**

### Beispiel

Umwandlung der Zahl $010110011111_2$ ins Oktalsystem:

```
0101 1001 1111
 ⇓    ⇓    ⇓
 5    9    F
```

Die Zahl $010110011111_2$ entspricht somit der Zahl $59F_{16}$.

## Hexadezimalsystem ⇒ Binärsystem

Selbiges gilt auch für die Umrechnung vom Hexadezimalsystem in das Binärsystem.

**⇒ Eine Stelle im Hexadezimalsystem entspricht 4 Stellen im Binärsytem**

### Beispiel

Umwandlung der Zahl $672_{16}$ ins Binärsystem:

```
 6    7    2
 ⇓    ⇓    ⇓
0110 0111 0010
```

Die Zahl $672_{16}$ entspricht somit der Zahl $11001110010_2$.

# Oktalsystem ⇒ Hexadezimalsystem

Will man vom Oktalsystem ins Hexadezimalsystem umrechnen, rechnet man am besten zuerst ins Binärsystem und dann ins Hexadezimalsystem um.

# Hexadezimalsystem ⇒ Oktalsystem

Will man vom Hexadezimalsystem ins Oktalsystem umrechnen, rechnet man am besten zuerst ins Binärsystem und dann ins Oktalsystem um.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:01_zahlensysteme:1_04**

Last update: **2023/09/10 12:58**

# 2) Informationseinheiten



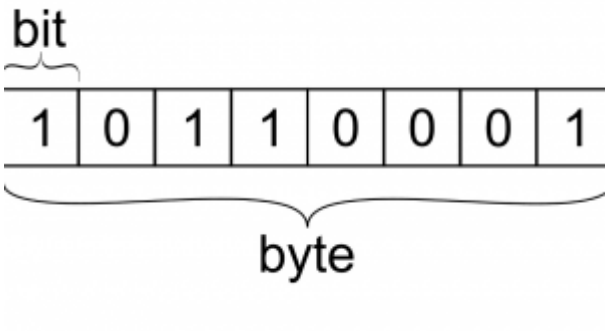
---

Zur **einfacheren Verarbeitung** und **Übertragung von Daten in und zwischen digitalen Systemen**, weisen die **Daten bestimmte Strukturen** auf. Dabei handelt es sich um **einzelne Zustände in Form von „0“ oder „1“** oder eben **Gruppen von solchen Zuständen**. Einzelne **Zustände werden als Bit** bezeichnet. Eine **Gruppe von Zuständen wäre eine Bitfolge**, die man zum Beispiel Byte oder Datenwort nennt.

## Bit

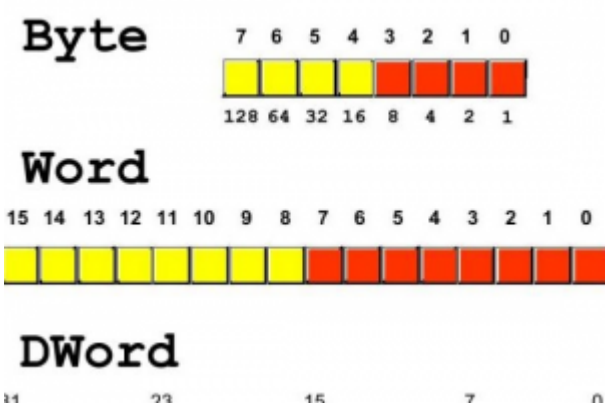
Das **Bit ist die kleinste Informationseinheit** in der Informations- und Kommunikationstechnik. Es kann **zwei Zustände** annehmen und stellt eine Speicherstelle dar.


## Byte



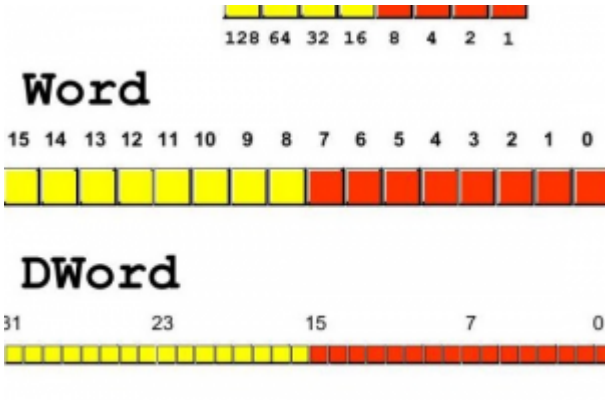

8 Bit werden zu einem Byte zusammengefasst und repräsentiert oft ein Zeichen. Eine weitere Bezeichnung für 8 Bit ist das Oktett.

# Word




Das Word kommt aus dem Englischen womit ein Wort gemeint ist. Ein Wort besteht aus mindestens 2 Buchstaben. Daher besteht ein Wort aus 2 Byte, was insgesamt 16 Bit entspricht.

# DWord




Das DWord ist das Doppelwort, das aus 2 Wörtern besteht. Das sind 32 Bit oder 4 Byte.

# Vorzeichen der Maßeinheiten (SI-Präfix)

Um den Umgang mit großen Zahlen zu erleichtern, existieren im metrischen System Einheitenvorzeichen bzw. SI-Präfixe.

Das Kilo, Mega, Giga, Tera usw. sind jeweils ein Vielfaches von 10^3 (= 1.000).

Bekannte Beispiele.:

1 Kilogramm ⇒ 1000 Gramm

1 Kilometer ⇒ 1000 Meter

Diese Vorzeichen werden für die Maßeinheiten Bit und Byte gleichermaßen verwendet.

Bits:

```
10^3 Bit =             1 000 Bit = 1 Kilobit = 1 kBit = 1Kb

10^6 Bit =     1 000 000 Bit = 1 Megabit = 1 MBit = 1Mb

10^9 Bit =      1 000 000 000 Bit = 1 Gigabit = 1 GBit = 1Gb

10^12 Bit = 1 000 000 000 000 Bit = 1 Terabit = 1 TBit = 1Tb

Bytes:

10^3 Byte  =            1 000 Byte= 1 Kilobyte     = 1 KByte = 1KB

10^6 Byte  =        1 000 000 Byte= 1 Megabyte     = 1 MByte = 1MB

10^9 Byte  =    1 000 000 000 Byte= 1 Gigabyte     = 1 GByte = 1GB

10^12 Byte = 1 000 000 000 000 Byte= 1 Terabyte = 1 TByte = 1TB
```

# 1KBit (1 000 Bits) vs. 1 KiBit (1 024 Bits)

Wenn man sich mit Computertechnik beschäftigt, dann stellt man immer wieder fest, das die üblichen Angaben zu Speicherkapazität und Übertragungsrate nicht immer dem entspricht, wie angegeben ist. So ist ein Kilobit (kBit) nicht unbedingt ein Kilobit (kBit).

Das liegt daran weil die SI-Präfixe für ein Vielfaches von 103 (= 1.000) definiert sind, in der Informatik fälschlicherweise aber auch für ein Vielfaches von 210 (= 1.024) verwendet werden.

Das bedeutet, wir haben ein dezimales Kilobit mit 1.000 Bit und ein binäres Kilobit mit 1.024 Bit.

DIE LÖSUNG:

Um eine eindeutige Unterscheidung zu treffen gibt es seit 1996 zusätzliche Präfixe, die auf dem dualen Zahlensystem beruhen und für binäre Werte festgelegt wurden.

- Kibibit: KiBit, Kib
- Mebibit: MiBit, Mib
- Gibibit: GiBit, Gib
- Tebibit: TiBit, Tib

bzw.

- Kibibyte: KiByte, KiB
- Mebibyte: MiByte, MiB
- Gibibyte: GiByte, GiB
- Tebibyte: TiByte, TiB

Das „bi“ in „Kibibit“ und das „i“ in Kib stehen jeweils für „binär“.

```
2^0 =               1 Bit = 1 Bit

2^10 =          1 024 Bit = 1 Kibibit / KiBit

2^20 =      1 048 576 Bit = 1 Mebibit / MiBit

2^30 =  1 073 741 824 Bit = 1 Gibibit / GiBit
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:02_informationseinheiten**

Last update: **2023/10/09 06:29**

# 03) Information und Daten

Im allgemeinen Sprachgebrauch wird „Information“ mit „Bedeutung“ oder „Wissen“ gleichgesetzt. Im Vergleich dazu sind „Daten“ Angaben zu „Sachverhalten“ und „Vorgängen“. Daten sind also Werte und Inhalte, die eine Information darstellen können.

Informationen und Daten haben keinen Ort. Sie können jederzeit von einem ortsgebundenen materiellen Träger zu einem anderen wandern. Das bedeutet auch, dass sich Informationen nicht zweifelsfrei lokalisieren lassen.

Gesetze und Regeln, die Informationen und Daten an einen Ort binden, scheinen nützlich und sinnvoll zu sein. Sie sind aber kaum umsetzbar und damit sinnlos. Denn jede einzelne Informationseinheit kann jederzeit an jeden Ort der Welt übertragen und gespeichert werden.

Nur dann, wenn man Informationen und Daten durchgehend verschlüsselt, ist es fast egal, wo die Informationen und Daten gespeichert sind.

## Information

- als Beseitigung von Unwissenheit
- als eine Nachricht, welche der Absender dem Empfänger über einen Kanal vermittelt. Die Nachricht wird dann interpretiert/verstanden.
- als darstellbar als Folge von 0 und 1
- als über Raum und Zeit sich ein physikalisches veränderliches Signal

## Daten

- stellen Informationen dar!
- sind Träger von Informationen
- können in Schrift, Ton und Bild auftreten
- können analog oder digital gespeichert werden
- können steuern, nutzen oder adressieren

## EDV (Elektronische Datenverarbeitung)

Die elektronische bzw. digitale Datenverarbeitung kennt im Prinzip nur zwei Zustände. Diese beiden Zustände werden als logisch „High“ und „Low“ bezeichnet und häufig als „1“ und „0“ dargestellt. Etwas, was nur diese zwei Zustände kennt, bezeichnet man als binäres System oder Binärlogik. Alle Daten, die elektronisch verarbeitet werden sollen, müssen in dieses Binärsystem übersetzt werden.

Das bedeutet, Schrift in Form von Buchstaben, Zahlen und Zeichen, und Bilder mit der Darstellung von Personen, Gegenständen oder Landschaften und jegliche andere Daten und Informationen werden als elektronisch lesbare Codierung in Form einer Folge aus „0“ und „1“ verarbeitet und gespeichert.

# Codierung von Zeichen

Eine **Zeichenkodierung (englisch Character encoding, kurz Encoding)** erlaubt die eindeutige Zuordnung von Schriftzeichen (i. A. Buchstaben oder Ziffern) und Symbolen innerhalb eines Zeichensatzes. In der elektronischen Datenverarbeitung werden Zeichen über einen Zahlenwert kodiert, um sie zu übertragen oder zu speichern. Der deutsche Umlaut Ü wird zum Beispiel im ISO-8859-1-Zeichensatz mit dem Dezimalwert 220 kodiert. Im EBCDIC-Zeichensatz kodiert derselbe Wert 220 die geschweifte Klammer }. Zur richtigen Darstellung eines Zeichens muss also die Zeichenkodierung bekannt sein; der Zahlenwert allein reicht nicht aus.

Zahlenwerte aus Zeichenkodierungen lassen sich auf verschiedene Art speichern oder übertragen, z. B. als Morsezeichen, verschieden hohe Töne (Faxgerät), verschieden hohe Spannungen.

# Geschichte des Character Encoding

Mit der Entwicklung des Computers begann die Umsetzung der im Grunde schon seit dem Baudot-Code verwendeten binären Zeichenkodierung in Bit-Folgen, bzw. intern meist in verschiedene elektrische Spannungswerte als Unterscheidungskriterium, ganz analog zu der bisher zur Unterscheidung der Signalwerte genutzten Tonhöhe oder Signaldauer.

Um diesen Bit-Folgen darstellbare Zeichen zuzuordnen, mussten Übersetzungstabellen, sogenannte Zeichensätze, engl. Charsets, festgelegt werden. 1963 wurde eine erste **7-Bit-Version des ASCII-Codes durch die ASA (American Standards Association)** definiert, um eine **Vereinheitlichung der Zeichenkodierung** zu erreichen. Obwohl IBM an der Definition mitgearbeitet hatte, führte man 1964 einen eigenen **8-Bit-Zeichencode EBCDIC** ein. Beide finden bis heute in der Computertechnik Verwendung.

Da für viele Sprachen jeweils unterschiedliche diakritische Zeichen benötigt werden, mit denen Buchstaben des lateinischen Schriftsystems modifiziert werden, gibt es für viele Sprachgruppen jeweils eigene Zeichensätze. Die **ISO** hat mit der **Normenreihe ISO 8859 Zeichenkodierungen für alle europäischen Sprachen** (einschließlich Türkisch) und Arabisch, Hebräisch sowie Thai standardisiert.

Das **Unicode Consortium** schließlich veröffentlichte 1991 eine erste Fassung des gleichnamigen Standards, der es sich zum Ziel gesetzt hat, alle Zeichen aller Sprachen in Codeform zu definieren. **Unicode** ist gleichzeitig die **internationale Norm ISO 10646**.

Bevor ein Text elektronisch verarbeitet wird, muss der verwendete Zeichensatz und die Zeichenkodierung festgelegt werden. Dazu dienen beispielsweise folgende Angaben:

Definition des Zeichensatzes in einer HTML-Seite

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

Definition des Zeichensatzes in den Kopfzeilen (Headern) einer E-Mail oder eines HTTP-Pakets

```
Content-Type: text/plain; charset="ISO-8859-1"
```

# ASCII - American Standard Code for Information Interchange

Der **American Standard Code for Information Interchange (ASCII, deutsch „Amerikanischer Standard-Code für den Informationsaustausch“)** ist eine **7-Bit-Zeichenkodierung**; sie entspricht der US-Variante von ISO 646 und dient als Grundlage für spätere, auf mehr Bits basierende Kodierungen für Zeichensätze.

Der ASCII-Code wurde zuerst am 17. Juni 1963 von der **American Standards Association (ASA) als Standard** ASA X3.4-1963 gebilligt und 1967/1968 wesentlich sowie zuletzt im Jahr 1986 von ihren Nachfolgeinstitutionen aktualisiert. Die Zeichenkodierung definiert **128 Zeichen**, bestehend aus **33 nicht druckbaren** sowie **95 druckbaren Zeichen**.

## ASCII-Code Tabelle

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

www.VirtualUniversity.ch

## Fakten

- sehr verbreiteter Standard, u.a. in PCs (Hardware-Ebene)
- von ISO genormt

- ursprünglich 7-Bit-Code, also 128 Zeichen
- davon einige nach nationalem Bedarf abgewandelt z.B. deutsche Umlaute statt [ ] { } \ |
- unterschiedliche 8-Bit-Erweiterungen mit zusätzlichen Zeichen im Bereich 128 – 255
- Erweiterungen oft problematisch bei älteren Rechnern, im Internet1) etc. → deshalb E-Mail, HTTP etc. auf 7 Bit beschränkt (→ 2. Sem.)

Probiere weitere Zeichen der ASCII-Tabelle in Microsoft Word aus. Drücke die ALT-Taste und tippe einen dezimalen ASCII Code ein! (z.B. ALT + 65)

# UTF8 - UCS Transformation Format

**UTF-8 (Abk. für 8-Bit UCS Transformation Format, wobei UCS wiederum Universal Character Set abkürzt)** ist die am weitesten verbreitete Kodierung für Unicode-Zeichen (Unicode und UCS sind praktisch identisch). Die Kodierung wurde im September 1992 von Ken Thompson und Rob Pike bei Arbeiten am Plan-9-Betriebssystem festgelegt.

UTF-8 ist in den **ersten 128 Zeichen (Indizes 0–127) deckungsgleich mit ASCII** und eignet sich mit in der Regel nur **einem Byte Speicherbedarf** für Zeichen vieler westlicher Sprachen besonders für die Kodierung englischsprachiger Texte, die sich im Regelfall ohne Modifikation daher sogar mit nicht-UTF-8-fähigen Texteditoren ohne Beeinträchtigung bearbeiten lassen, was einen der Gründe für den Status als **De-facto-Standard-Zeichenkodierung des Internets** und damit verbundener Dokumenttypen darstellt. Im Oktober 2017 verwendeten **89,9 % aller Websites UTF-8**

In anderen Sprachen ist der Speicherbedarf in Byte pro Zeichen größer, wenn diese vom ASCII-Zeichensatz abweichen: Bereits die **deutschen Umlaute erfordern zwei Byte**, ebenso griechische oder kyrillische Zeichen. Zeichen fernöstlicher Sprachen und von Sprachen aus dem afrikanischen Raum belegen dagegen bis zu 4 Byte je Zeichen. Da die Verarbeitung von UTF-8 als Multibyte-Zeichenfolge wegen der notwendigen Analyse jedes Bytes im Vergleich zu Zeichenkodierungen mit fester Byteanzahl je Zeichen mehr Rechenaufwand und für bestimmte Sprachen auch mehr Speicherplatz erfordert, werden abhängig vom Einsatzszenario auch andere UTF-Kodierungen zur Abbildung von Unicode-Zeichensätzen verwendet: Microsoft Windows als meistgenutztes Desktop-Betriebssystem verwendet intern als Kompromiss zwischen UTF-8 und UTF-32 etwa UTF-16 Little Endian

## Fakten Unicode (UTF-8, UTF-16, UTF-32)

- neuere Kodierung, ebenfalls ISO-standardisiert 1993 und heutzutage Standard
- Ziel: Berücksichtigung möglichst vieler Sprachen und ihrer Eigenheiten
- Buchstaben-, Silben-, und Ideogrammsprachen
- Schreibrichtungen (links-rechts, rechts-links, oben-unten)
- außerdem diverse Sonderzeichen, mathematisch-technische Symbole, Diakritika, geometrische Formen, Pfeile, Piktogramme u.v.m.
- dafür 16-Bit-Darstellung (erlaubt 65.536 Zeichen)
- Erweiterung auf 32 Bit für künftigen Bedarf
- Standard auf unixoiden Betriebssystemen

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:03_zeichencodierung**

Last update: **2023/09/10 13:15**

# 04) Schaltalgebra



https://elektroniktutor.de/swfdatei/gatter.swf

Quellen: http://elektronik-kurs.net/digitaltechnik/logische-verknupfungen/
https://www.edv-lehrgang.de/digitaltechnik/ https://elektroniktutor.de/digitaltechnik/gatter.html
https://www.elektronik-kompendium.de/sites/dig/0205184.htm

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:04_schaltalgebra**

Last update: **2023/09/10 13:15**

# 4.01) Grundlagen

In der Digitaltechnik sind die Eingangsvariablen so miteinander zu verknüpfen, dass die Ausgangsvariable einen definierten Zustand annimmt, um damit einen Prozess zu steuern. So soll ein Fahrstuhl nur dann fahren, wenn eine Zieletage gewählt wurde, in der er zurzeit nicht steht, seine Tür vollständig geschlossen ist und von ihr nichts eingeklemmt wurde und die Kabine nicht überlastet ist. Man kann durch Kombinieren der beschriebenen digitalen Gatter und Ausprobieren bei einfacheren Aufgaben die eine oder andere Lösung finden. Das eigentliche Ziel ist es, eine wirtschaftliche Digitalschaltung zu entwickeln, die mit einem Minimum gleicher Gattertypen zum Ziel führt.

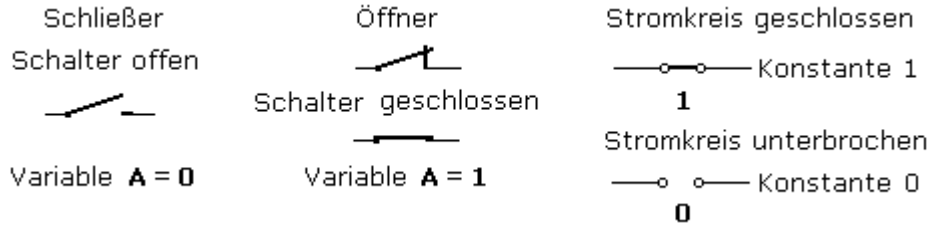
Der englische Mathematiker Georg Boole entwickelte eine Mengenalgebra, die in angepasster Weise als Boolesche Schaltalgebra bei der Problemlösung hilfreich ist. In der Schaltalgebra gibt es Variablen und Konstanten. Die binäre Digitaltechnik kommt mit zwei definierten logischen Zuständen, der 0 und 1 aus.

## Konstante

In der Schaltalgebra gibt es nur die zwei konstanten Größen 0 und 1. In der elektronischen Schaltung entspricht eine dauerhaft geschaltete Leitung der Konstanten mit dem Zustand 1. Die dauerhafte Unterbrechung eines Stromkreises steht für die Konstante mit dem Wert 0.

## Variable

Veränderbare, schaltbare Eingangsgrößen und davon abhängige veränderliche Ausgangswerte werden als Variable bezeichnet. In der binären Digitaltechnik nehmen sie entweder den Zustand 0 oder 1 an. In elektronischen Schaltungen können sie mit einem Schalter verglichen werden. Der geöffnete Schalter entspricht einer Variablen mit dem Wert 0. Bei geschlossenem Schalter nimmt die Variable den Wert 1 an.




Wir wissen, dass alle Grundrechnungsarten auf eine Addition zurückgeführt werden können. Deshalb ist das Ziel dieses Kapitels eine Maschine simulieren zu können, die addieren kann.

## Informationseinheiten

### BIT

Die Ziffern 0 und 1 werden physikalischen Zuständen zugeordnet:

```
0   Strom fließt nicht
1   Strom fließt
```

0 und 1 werden in der Informatik mit Bit bezeichnet. Ein Bit stellt die kleinste Informationseinheit dar (nur 2 Zustände – 0 und 1).

1 Bit ⇒ $2^1$ ⇒ 2 Möglichkeiten
2 Bit ⇒ $2^2$ ⇒ 4 Möglichkeiten
3 Bit ⇒ $2^3$ ⇒ 8 Möglichkeiten
8 Bit = 1 Byte ⇒ $2^8$ ⇒ 256 Möglichkeiten

### BYTES

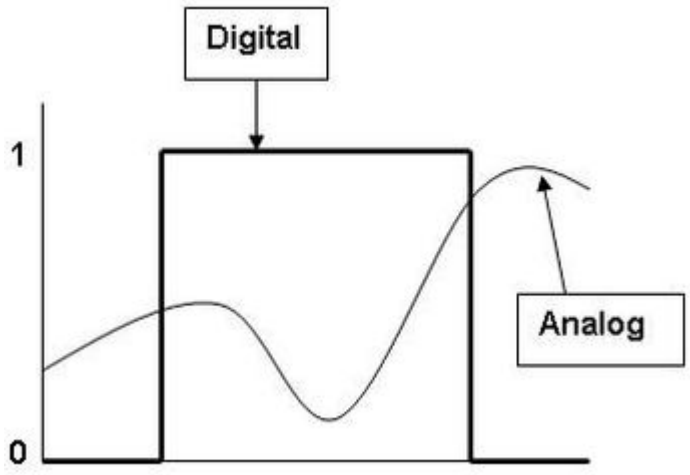
#### 8 Bits sind ein Byte!

1 KB (Kilobyte) = $2^{10}$ = 1024 Byte
1 MB (Megabyte) = $2^{10}$ KB = $2^{10}$ x $2^{10}$ Byte = $2^{20}$ Byte = 1048576 Byte
1 GB (Gigabyte) = $2^{10}$ Byte
1 TB (Terrabyte) = $2^{10}$ Byte

## Darstellung von 0 und 1

Ein Computer verarbeitet systembedingt Digitalsignale, da nur ausgewertet werden kann, ob eine Spannung anliegt oder nicht bzw. innerhalb eines Definitionsbereichs einen Wert über- oder unterschreitet. Diese zwei Zustände werden auch häufig bezeichnet als:

**HIGH** und **LOW**
**TRUE** und **FALSE**
**AN** und **AUS**
**1** und **0**



From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:04_schaltalgebra:4_01**

Last update: **2023/09/10 13:16**

# 4.02) Grundschaltungen

## UND-Verknüpfung (=Serienschaltung)

Bei der UND-Verknüpfung führt der Ausgang das Signal 1, wenn an beiden Eingängen (a und b) das Signal 1 anliegt.

### Zeichen

| Deutsch | Englisch | Fachausdruck | Zeichen (boolsche Algebra) | Zeichen (Mathematik) |
|---|---|---|---|---|
| UND | AND | Konjunktion | X=A∧B | X=A*B |

### Schaltwerttabelle

| a | b | a∧b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

### Schaltfunktion

f(a,b)=a∧b

### Schaltsymbol

### elektronische Schaltung

## ODER-Verknüpfung (=Parallelschaltung)

Bei der ODER-Verknüpfung führt der Ausgang das Signal 1, wenn an einem der beiden Eingänge das Signal 1 anliegt.

### Zeichen

| Deutsch | Englisch | Fachausdruck | Zeichen (boolsche Algebra) | Zeichen (Mathematik) |
|---|---|---|---|---|
| ODER | OR | Disjunktion | X=A∨B | X=A+B |

### Schaltwerttabelle

| a | b | a∧b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

### Schaltfunktion

f(a,b)=a∨b

### Schaltsymbol

### elektronische Schaltung

## NICHT-Verknüpfung (=NOT-Schaltung)

Bei einer NICHT-Verknüpfung wird der Ausgang logisch 1, wenn der Eingang logisch 0 ist, und umgekehrt.

### Zeichen

| Deutsch | Englisch | Fachausdruck | Zeichen (boolsche Algebra) | Zeichen (Mathematik) |
|---|---|---|---|---|
| NICHT | NOT | Negation | X=¬A | X=Ā |

### Schaltwerttabelle

| a | ¬a |
|---|---|
| 0 | 1 |
| 1 | 0 |

### Schaltfunktion

f(a)=¬a

### Schaltsymbol

# elektronische Schaltung

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:04_schaltalgebra:4_02**

Last update: **2023/09/10 13:16**

# 4.03) Zusammengesetzte Schaltungen

Die **Grundverknüpfungen** werden in der Regel miteinander **kombiniert**, sodass die Logik der Schaltung verändert wird. Beispielsweise kann man eine UND- mit einer NICHT-Verknüpfung kombinieren und erhält dadurch eine sogenannte NAND-Verknüpfung (NICHT-UND). Da solch zusammengesetzte Schaltfunktionen sehr häufig verwendet werden, haben sie **eigene Schaltzeichen** bekommen. Mit den folgenden Schaltfunktionen werden die Grundverknüpfungen erweitert.

- NAND
- NOR
- XOR
- XNOR

## NAND-Verknüpfung

Eine Kombination aus einem **UND-Gatter mit nachfolgendem NICHT-Gatter ergibt ein NAND-Gatter**. Die Ausgangsvariable Z des UND-Gatters wird durch das NICHT-Gatter negiert und erzeugt die Ausgangsvariable X des NAND-Glieds. Viele logische Funktionen lassen sich durch den Einsatz von NAND-Gattern lösen. Oft steigt dadurch die Anzahl der notwendigen Gatter, mit dem wirtschaftlichen Vorteil nur einen Gattertyp zu verwenden.

**Der Ausgangszustand eines NAND-Glieds ergibt 1, wenn nicht alle Eingangszustände 1 sind.**

### Zeichen

| Deutsch | Englisch | Fachausdruck | Zeichen (boolsche Algebra) |
|---|---|---|---|
| NEGIERTES UND | NOT AND | Negative Konjunktion | |

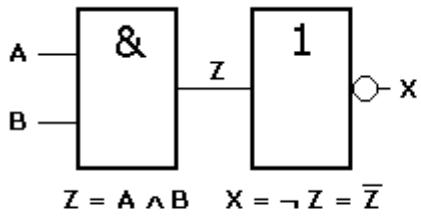
### Schaltwerttabelle

| a | b | $a \barwedge b$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

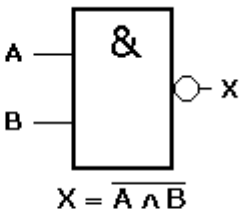
### Schaltfunktion

$f(a,b)=a \barwedge b$

### Schaltsymbol




UND-NICHT Schaltung




NAND genormtes Schaltzeichen

| A | B | Z | X |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Wahrheitstabelle

# NOR-Verknüpfung

Eine Kombination aus einem **ODER-Gatter mit nachfolgendem NICHT-Gatter ergibt ein NOR-Gatter**. Die Ausgangsvariable Z des ODER-Gatters wird durch das NICHT-Gatter negiert und erzeugt die Ausgangsvariable X des NOR-Glieds. NOR-Glieder haben in logischen Schaltungen die gleiche wichtige Bedeutung wie NAND-Glieder.

**Der Ausgangszustand eines NOR-Glieds ergibt 1, wenn alle Eingangszustände 0 sind.**

## Zeichen

| Deutsch | Englisch | Fachausdruck | Zeichen (boolsche Algebra) |
|---|---|---|---|
| NEGIERTES ODER | NOT OR | Negative Disjunktion | |

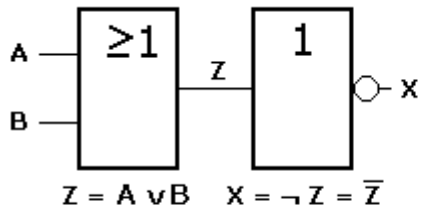
## Schaltwerttabelle

| a | b | a⊽b |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

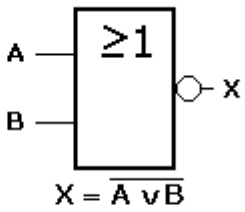
## Schaltfunktion

f(a,b)=a⊽b

## Schaltsymbol



ODER-Nicht Schaltung



NOR genormtes Schaltzeichen

| A | B | Z | X |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

Wahrheitstabelle

# XOR-Verknüpfung

Die logische Verknüpfung des XOR-Gatters mit zwei Eingangsvariablen kann mit einem **'entweder - oder'** umschrieben werden. Die **Ausgangsvariable wird immer dann 'true' liefern, wenn die Eingangsvariablen unterschiedliche Zustände haben**. Die Wahrheitstabelle des XOR-Gatters entspricht dem ODER-Gatter mit dem Ausschluss gleicher Eingangszustände, also exklusiv der Äquivalenz. Dieses Verhalten wird als Antivalenz bezeichnet.

**Der Ausgangszustand eines XOR-Glieds ergibt 1, wenn eine ungerade Zahl der Eingangszustände 1 und alle anderen Eingangszustände 0 sind.**

## Zeichen

| Deutsch | Englisch | Fachausdruck | Zeichen (boolsche Algebra) |
|---|---|---|---|
| Exklusives ODER (Antivalenz) | Exclusiv OR | Exklusive Disjunktion | Z=a⊻b |

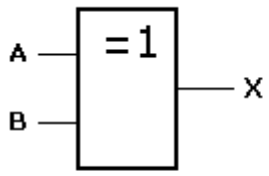
## Schaltwerttabelle

| a | b | a⊻b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## Schaltfunktion

f(a,b)=a⊻b

## Schaltsymbol




Exklusiv ODER
XOR – Antivalenz

$X = (A \wedge \overline{B}) \vee (\overline{A} \wedge B)$

$X = A \underline{\vee} B$

$X = A \oplus B$

Funktionsgleichung

| A | B | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Wahrheitstabelle

# XNOR-Verknüpfung

Die Bezeichnung **Äquivalenz bedeutet Gleichwertigkeit**. Beim XNOR-Gatter mit zwei Eingangsvariablen ist der **Zustand der Ausgangsvariable 1, wenn beide Eingangsvariablen den gleichen Zustand 0 oder 1 haben**. Die Funktion kann durch eine Schaltung mit vier NOR-Gattern erreicht werden. Es sind zwei unterschiedliche Schaltzeichen zu finden.

**Der Ausgangszustand eines XNOR-Glieds ergibt 1, wenn eine gerade Zahl der Eingangzustände 1 und alle anderen Eingangszustände 0 aufweisen oder alle 0 sind.**

## Zeichen

| Deutsch | Englisch | Fachausdruck | Zeichen (boolsche Algebra) |
|---|---|---|---|
| Exklusives NICHT ODER (Äquivalenz) | Exclusiv NOT OR | Biimplikation | Z=a≡b |

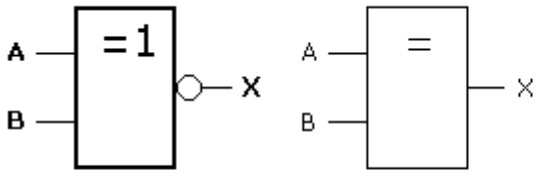
## Schaltwerttabelle

| a | b | a≡b |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## Schaltfunktion

f(a,b)=a≡b

## Schaltsymbol




Exklusiv NICHT ODER
XNOR – Äquivalenz

$X = (A \wedge B) \vee (\overline{A} \wedge \overline{B})$

$X = \overline{A \underline{\vee} B}$

$X = \overline{A \oplus B}$

Funktionsgleichung

| A | B | X |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Wahrheitstabelle

xor_nand.swf

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:04_schaltalgebra:4_03**

Last update: **2023/09/10 13:16**

# 4.04) Gesetze der Schaltalgebra

## Vorrang- und Klammerregel

Wie in der Algebra ist auch in der Digitaltechnik auf eine bestimmte Reihenfolge der Operationen zu beachten. Die Negation sollte vor der Konjunktion (UND) und diese vor der Disjunktion (ODER) ausgeführt werden. Das ist vergleichbar mit der Aussage, dass die Multiplikation Vorrang vor der Addition hat.

Bei mehreren Variablen und unterschiedlichen Verknüpfungen kann eine Klammersetzung notwendig sein. Beachtet man die Vorrangregel, kann man auf viele Klammern verzichten. Ein Setzen zeigt sogleich eindeutig, welche der Variablen wie zu verknüpfen ist. Bei ODER sollte immer geklammert werden, ebenfalls beim Anwenden der De Morganschen Gesetze auf NAND- und NOR-Verknüpfungen.

$X = A \vee B \wedge C$

nach der Vorrangregel gilt

$\boxed{X = A \vee (B \wedge C)}$

festgelegte Abfolge der Verknüpfungen

$\boxed{Z = (A \vee B) \wedge C}$

| A | B | C | B∧C | A∨(B∧C) | A∨B | (A∨B)∧C |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

$A \vee (B \wedge C) \neq (A \vee B) \wedge C$

Bei drei Eingangsvariablen und binären Zuständen gibt es 2^3 = 8 Eingangskombinationen. Die Wahrheitstabelle zeigt bei definierter Klammersetzung oder Beachtung der Vorrangregel UND vor ODER unterschiedliche Ergebnisse.

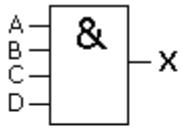
## Kommutativgesetz

Das Kommutativ- oder Vertauschungsgesetz besagt, dass bei der UND sowie ODER Verknüpfung auch bei beliebiger Vertauschung der Reihenfolge der Eingangsvariablen das Ergebnis gleich bleibt.

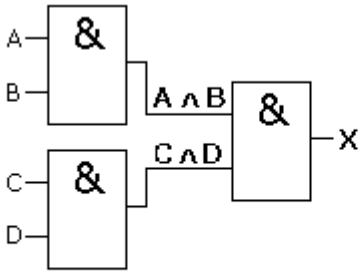
| | |
|---|---|
| UND | $X = A \wedge B \wedge C = C \wedge A \wedge B = B \wedge C \wedge A$ |
| ODER | $X = A \vee B \vee C = C \vee A \vee B = B \vee C \vee A$ |

## Assoziativgesetz

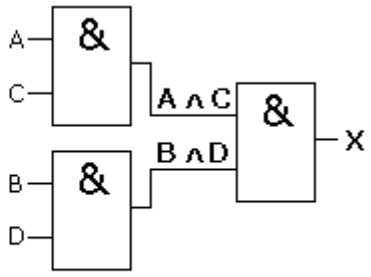
Das Assoziativ- oder Verbindungsgesetz besagt, dass bei einer UND- beziehungsweise ODER-Verknüpfung mit mehr als zwei Schaltvariablen die Verknüpfung auch stufenweise nacheinander in beliebiger Reihenfolge erfolgen kann.




| A | B | C | D | X |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |




| A∧B | C∧D | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




| A∧C | B∧D | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

$$X = A \wedge B \wedge C \wedge D = (A \wedge B) \wedge (C \wedge D) = (A \wedge C) \wedge (B \wedge D)$$

Beispiele zum Assoziativgesetz der UND-Verknüpfung

Anstelle des UND-Gatters mit vier Eingangsvariablen lassen sich zwei UND-Gatter mit zwei Eingängen verwenden. Die Variablen A, B, C, D werden einzeln beliebig mit den Eingängen verbunden. Die Ausgangsvariable der beiden UND-Gatter kann den Wert 0 oder 1 annehmen. Beide Ausgangsvariablen sind nochmals mit einem UND-Gatter zu verknüpfen, wobei sich wieder vier Eingangskombinationen ergeben. Nur wenn alle Eingangsvariablen 1 sind, wird der Ausgangszustand ebenfalls 1.

Das Assoziativgesetz gilt gleichermaßen für die ODER-Verknüpfung. Die Klammersetzung ist nicht notwendig und soll nur verschiedene Verteilungen besser erkennbar machen.

# Distributivgesetz

Das Distributiv- oder Verteilungsgesetz wird zur Vereinfachung von Verknüpfungsgleichung angewendet. Es ist vergleichbar mit dem Ausmultiplizieren und Ausklammern von Variablen der normalen Algebra. Da die logische UND-Verknüpfung der algebraischen Multiplikation entspricht, während die ODER-Verknüpfung mit der Addition vergleichbar ist, gibt es zwei unterschiedliche Distributivgesetze.

## Konjunktives Distributivgesetz

Eine Variable wird mit UND verknüpft und auf den Folgeausdruck verteilt. Die Vorgehensweise entspricht dem aus der Algebra bekannten Ausmultiplizieren eines Klammerausdrucks mit einem Faktor. Im umgekehrten Fall kann eine Variable, die mit mehreren anderen Variablen verknüpft ist, ausgeklammert werden. Das bedeutet für den Schaltungsaufbau eine Einsparung an Gattern.

$$X = A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$$

⇨ Ausmultiplizieren
⇦ Ausklammern

$$X = A \wedge (A \vee B)$$
$$X = (A \wedge A) \vee (A \wedge B)$$
$$X = A \vee (A \wedge B) = A$$
$$X = A$$

| | | UND | ODER | |
|---|---|---|---|---|
| A | B | A∧B | A | X |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Beim Ausklammern wird die Variable mit ihrem Verknüpfungszeichen, hier UND, vor die Klammer gesetzt. Die in der Klammer stehenden Variablen werden mit dem zuvor zwischen den Klammern stehenden ODER verknüpft.

### Disjunktives Distributivgesetz

Eine Variable kann durch ein vergleichbares Ausmultiplizieren auf andere Ausdrücke verteilt werden oder bei mehrfachem Auftreten ausgeklammert werden. Beim Ausklammern wird das ODER Verknüpfungszeichen mit der Variablen vor die Klammer geschrieben. disjunktives Distributivgesetz

$$X = A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$$

⇨ Ausmultiplizieren
⇦ Ausklammern

$$X = A \vee (A \wedge B)$$
$$X = (A \vee A) \wedge (A \vee B)$$
$$X = A \wedge (A \vee B)$$
$$X = A$$

| | | ODER | UND | |
|---|---|---|---|---|
| A | B | A∨B | A | X |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

## De Morgansche Gesetze

Die boolesche Algebra wurde vom englischen Mathematiker De Morgan weiter entwickelt. Für die Schaltalgebra gibt es zwei De Morgansche Gesetze. Sie machen Aussagen zur Negation einer Verknüpfung und der Umkehr von Verknüpfungszeichen. Mit den De Morganschen Gesetzen lassen sich bei der Entwicklung von Digitalschaltungen Gatter gegeneinander austauschen, Schaltungen verkleinern oder mit nur einem Gattertyp verwirklichen. Das erste Gesetz ist für die NAND-Verknüpfung und das zweite Gesetz entsprechend für die NOR-Verknüpfung definiert.

$$X = \overline{A \wedge B} = \overline{A} \vee \overline{B}$$

1. De Morgansche Gesetz

| | | | ODER | | |
|---|---|---|---|---|---|
| A | B | $\overline{A \wedge B}$ | $\overline{A}$ | $\overline{B}$ | X |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |

$$X = \overline{A \vee B} = \overline{A} \wedge \overline{B}$$

2. De Morgansche Gesetz

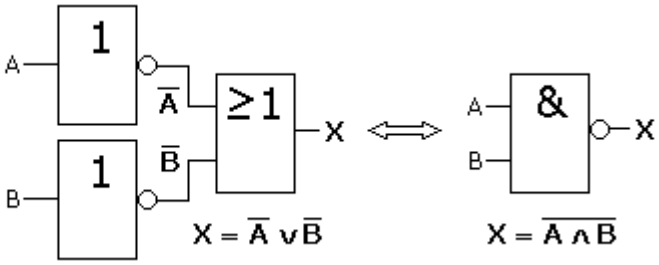
| | | | UND | | |
|---|---|---|---|---|---|
| A | B | $\overline{A \vee B}$ | $\overline{A}$ | $\overline{B}$ | X |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |

Ist eine Verknüpfung insgesamt negiert, so ist ihre Ausgangsvariable negiert. Die Negation kann auf die Einzelglieder aufgeteilt werden, wobei aus der Grundverknüpfung UND ein ODER beziehungsweise aus ODER ein UND wird. Eine doppelte Negation hebt sich auf. Getrennte Negationsstriche über

Variablen bedeuten, dass die Eingangsvariablen negiert sind.

Die oben zum 1. De Morganschen Gesetz gezeigte Verknüpfung kann schaltungstechnisch mit zwei NICHT- und einem ODER-Gatter verwirklicht werden. Wie zu erkennen ist, folgt das gleiche Ergebnis mit nur einem NAND-Gatter. Für das 2. De Morgansche Gesetz gilt die entsprechende Aussage. Zwei NICHT- und ein UND-Gatter reduzieren sich auf den Einsatz eines NOR-Gatters. Mit der doppelten Negation und den De Morganschen Gesetzen kann bei einer gegebenen Funktionsgleichung auf das Bestehen einer derartige Vereinfachung geprüft werden. Treten dabei mehrfache Negationen auf, so lassen sie sich von innen nach außen auflösen.

$$X = \overline{A} \vee \overline{B}$$
$$\overline{\overline{X}} = \overline{\overline{A} \vee \overline{B}} \quad \overline{v} \rightarrow \wedge$$
$$\overline{\overline{X}} = \overline{\overline{A} \wedge \overline{B}} \quad \overline{\overline{A}} = A$$
$$X = \overline{A} \wedge \overline{B}$$
$$\boxed{X = \overline{A} \wedge \overline{B} = \overline{A} \vee \overline{B}}$$




From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:04_schaltalgebra:4_04**

Last update: **2023/09/10 13:16**

# 4.05) Digitale Rechenschaltungen

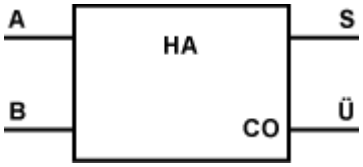
Aus logischen Verknüpfungen lassen sich digitale Schaltungen zusammenbauen, mit denen man Rechenvorgänge durchführen kann. Das heißt, diese Schaltungen haben zwischen ihren Eingängen eine Kombination aus logischen Verknüpfungen, die einem Rechenvorgang entspricht. In der Digitaltechnik kennt man Rechenschaltungen hauptsächlich für das duale Zahlensystem und den BCD-Code. Im Prinzip kann für jedes Zahlensystem eine Rechenschaltung aufgebaut werden.

Einige Rechenschaltungen der Digitaltechnik

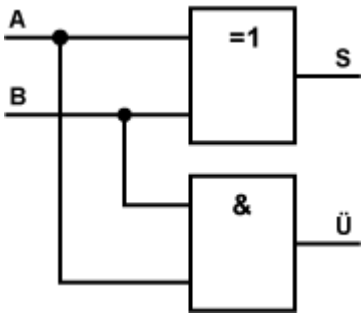
- Addiererschaltungen
- Subtrahiererschaltungen
- Addier-Subtrahier-Werke
- Multiplikationsschaltungen
- Arithmetisch-logische Einheit (ALU)

Stellvertretend für alle Rechenschaltungen dienen die folgenden Ausführungen zum Halbaddierer und dem Volladdierer.

## Halbaddierer




Ein Halbaddierer ist die einfachste Rechenschaltung und kann zwei einstellige Dualziffern addieren. Der Eingang A des Halbaddierers ist der Summand A, der Eingang B ist der Summand B. Die Schaltung hat zwei Ausgänge. Der Ausgang S als Summenausgang (2^0) und der Ausgang Ü als Übertrag (2^1) für die nächsthöhere Stelle im dualen Zahlensystem.



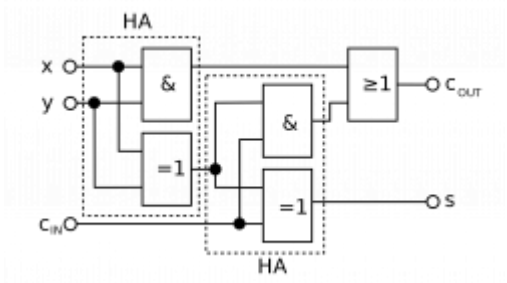

| B | A | Ü | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

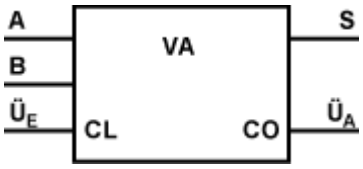
Der Halbaddierer ist eine Schaltung aus den Verknüpfungsgliedern XOR und UND. Das XOR ist die Addier-Verknüpfung. Das UND stellt fest, ob ein Übertrag für die nächsthöhere Stelle vorgenommen

werden muss. Aus der Tabelle ist ersichtlich, dass das Ergebnis aus der Spalte Summe (S) einer Exklusiv-ODER-Verknüpfung (Antivalenz, XOR) entspricht. Das Ergebnis der Spalte Übertrag (Ü) entspricht einer UND-Verknüpfung. Die so entstandene Schaltung wird als Halbaddierer bezeichnet. Sie ist in der Lage zwei 1-Bit Summanden (einstellig) zu addieren.

# Volladdierer

Um **mehrstellige Dualzahlen** addieren zu können benötigt man Schaltungen die auch einen Übertrag einer niederwertigen Stelle berücksichtigen. Man spricht vom **Übertragseingang ÜE**. Die Schaltung bezeichnet man als **Volladdierer (VA)**. Ein Volladdierer besteht aus 2 Halbaddierern und einem Oder-Gatter und kann drei Dualzahlen addieren. Bzw. zwei Dualzahlen addieren und den Übertrag aus einer niederwertigen Stelle berücksichtigen.

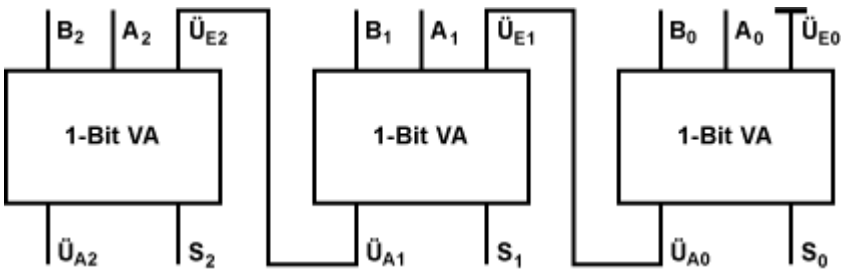



Folgende Wahrheitstabelle ergibt sich:

| $Ü_E$ | B | A | $Ü_A$ | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

## 3-Bit-Volladdierer



Folgende Schaltung zeigt einen 3-Bit-Volladdierer (VA), der aus drei 1-Bit-Volladdierer (VA) realisiert wurde. Damit lassen sich zwei dreistellige Dualzahlen addieren. Der Eingang $Ü_{E0}$ liegt an 0 V (Masse), weil in der niederwertigsten Stelle kein Übertrag berücksichtigt werden muss.

### Beispiel

```
 010 = A
+111 = B
---
1001 = S
```

#### 0. Stelle

| $Ü_{E0}$ | $A_0$ | $B_0$ | $Ü_{A0}$ | $S_0$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |

#### 1. Stelle

| $Ü_{E1}$ | $A_1$ | $B_1$ | $Ü_{A1}$ | $S_1$ |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 |

#### 2. Stelle

| $Ü_{E2}$ | $A_2$ | $B_2$ | $Ü_{A2}$ | $S_2$ |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |

#### Ergebnis

$Ü_{A2}$ $S_2$ $S_1$ $S_0$

```
1  0  0  1
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:04_schaltalgebra:4_05**

Last update: **2023/10/16 07:08**

# 4.06) Vereinfachen von boolschen Ausdrücken

Prinzipiell kann man zur Vereinfachung eines booleschen Ausdrucks alle kennengelernten Gesetze anwenden.

Doch wie kann man dabei **systematisch** vorgehen?

## KV-Diagramm

Das **Karnaugh-Veitch-Diagramm, kurz KV-Diagramm**, dient der übersichtlichen Darstellung und Vereinfachung Boolescher Funktionen – Umwandlung der disjunktiven Normalform in einen minimalen logischen Ausdruck.

Mittels eines KV-Diagramms lässt sich jede beliebige **disjunktive Normalform** (DNF) in **einen minimalen logischen Ausdruck** umwandeln. Der Vorteil gegenüber anderen Verfahren ist, dass der erzeugte Term (meist) minimal ist. Sollte der Term noch nicht minimal sein, ist eine weitere Vereinfachung durch Anwenden des Distributivgesetzes (Ausklammern) möglich. Das Umwandeln beginnt mit dem Erstellen einer Wahrheitstafel, aus der dann die DNF abgeleitet wird, die dann wiederum direkt in ein KV-Diagramm umgewandelt wird.

Da sich benachbarte Felder jeweils in einer Variable nur um ein Bit unterscheiden, ist folgende Regel anwendbar: A oder ¬A = 1. Auf dieser Regel basiert die Reduzierung der Gruppen.

### Disjunktive Normalform

Als disjunktive Normalform (kurz DNF) wird in der Booleschen Algebra eine in besonderer Weise normierte Funktionsdarstellung Boolescher Funktionen bezeichnet.

Bei der disjunktiven Normalform handelt es sich um einen logischen Ausdruck, der aus ODER-Verknüpfungen (Disjunktion – nicht ausschließendes ODER) besteht. Der logische Ausdruck besteht in der obersten Ebene ausschließlich aus ODER-Verknüpfungen.

$$A \vee B \vee C \vee D$$

Dabei können die einzelnen Elemente der ODER-Verknüpfung (A, B, C, D) komplexere Ausdrücke sein, die dann auch eine UND-Verknüpfung (Konjunktion) enthalten können.

$$(A \wedge B) \vee (A \wedge B \wedge C) \vee (B \wedge C) \vee D$$

Hier handelt es sich um eine Disjunktion (ODER-Verknüpfung) von drei Konjunktionen (UND-Verknüpfungen) und der Aussage D – genau das ist die disjunktive Normalform.

Vereinbarungsgemäß werden die Klammern und die Zeichen (Operatoren) für die UND-Verknüpfung nicht mitgeschrieben.

$$AB \vee ABC \vee BC \vee D$$

**Beispiel**

Zuerst wird die Wahrheitstabelle einer Schaltung und daraus die Gleichung erstellt.
Da wir hier nur zwei Eingänge haben, ergibt sich die größe des KV-Diagramms mit 2n also 22= 4 Felder. ( n steht für die Anzahl der Eingangsvariablen )
Nun werden die Werte entsprechend ihrer Bedingungen in die Felder (dort wo sie sich überschneiden) eingetragen.
Nach der Minterm-Methode werden die Zustände für 1 (Z=1) und nach der Maxterm-Methode die Zustände für (Z=0) eingetragen.

Nun werden alle 1 oder alle 0 zusammengefasst.
Wir wenden die Minterm-Methode an und fassen die 1 zusammen. Es können immer nur 2,4,8 usw. benachbarte Felder horizontal oder vertikal zusammengefasst werden.

| | B | A | Z |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | 0 |

$$0 \longrightarrow Z = \bar{A} \wedge \bar{B} = \overline{A \wedge B}$$

$$1 \longrightarrow Z = A \wedge \bar{B}$$

$$2 \longrightarrow Z = \bar{A} \wedge B$$

**Die Gleichung lautet:**

$$Z = \bar{A} \wedge \bar{B} \vee Z = A \wedge \bar{B} \vee Z = \bar{A} \wedge B$$

| $\bar{B}$ | $\bar{A}$ | $A$ |
|---|---|---|
| $\bar{B}$ | 1 | 1 |
| $B$ | 1 | 0 |

| $\bar{B}$ | $\bar{A}$ | $A$ |
|---|---|---|
| $\bar{B}$ | 1 | 1 |
| $B$ | 1 | 0 |

$\bar{B}$

$\bar{A}$

$$\longrightarrow Z = \bar{A} \vee \bar{B}$$

| $\bar{B}$ | $\bar{A}$ | $A$ |
|---|---|---|
| $\bar{B}$ | 1 | 1 |
| $B$ | 1 | 0 |

**Nach der Maxterm-Methode:**

$$\longrightarrow \bar{Z} = A \wedge B \longrightarrow Z = \overline{A \wedge B}$$

## 8-Felder KV-Diagramm

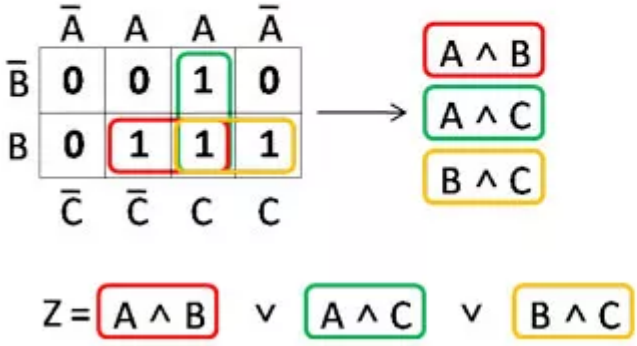
Da wir 3 Eingänge haben, vergrößert sich unser KV-Diagramm auf 23 also 8 Felder.

Wir erstellen auch hier wieder die Wahrheitstabelle und übertragen die Gleichungen in das KV-Diagramm.
Durch Zusammenfassen der 2-er Kombinationen erhalten wir unsere Gleichung.

| | C | B | A | Z | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 1 | 0 | |
| 2 | 0 | 1 | 0 | 0 | |
| 3 | 0 | 1 | 1 | 1 | $\longrightarrow Z = A \wedge B \wedge \overline{C}$ |
| 4 | 1 | 0 | 0 | 0 | |
| 5 | 1 | 0 | 1 | 1 | $\longrightarrow Z = A \wedge \overline{B} \wedge C$ |
| 6 | 1 | 1 | 0 | 1 | $\longrightarrow Z = \overline{A} \wedge B \wedge C$ |
| 7 | 1 | 1 | 1 | 1 | $\longrightarrow Z = A \wedge B \wedge C$ |

**Die Funktionsgleichung lautet:**

$$Z = A \wedge B \wedge \overline{C} \vee Z = A \wedge \overline{B} \wedge C \vee Z = \overline{A} \wedge B \wedge C \vee Z = A \wedge B \wedge C$$




$$Z = A \wedge B \vee A \wedge C \vee B \wedge C$$

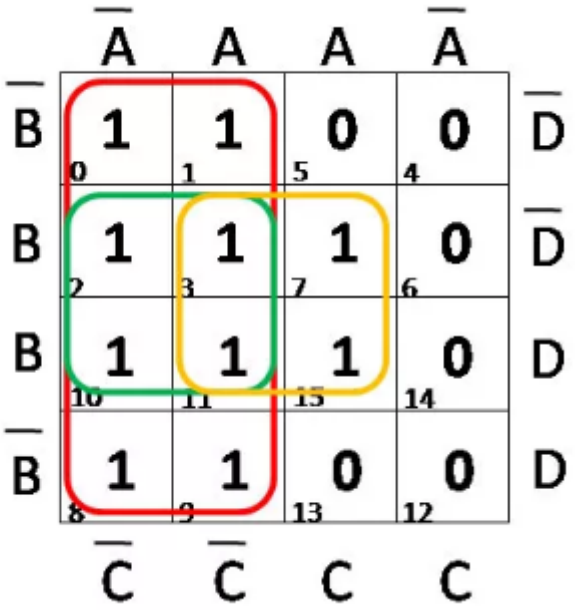
## 16-Felder KV-Diagramm

Da wir 4 Eingänge haben, vergrößert sich unser KV-Diagramm auf 24 also 16 Felder.

Wir erstellen auch hier wieder die Wahrheitstabelle und übertragen die Werte in das KV-Diagramm.

Da wir einen 6er-Block nicht zusammenfassen können, teilen wir ihn in zwei 4er-Blöcke auf.

| | D | C | B | A | Z |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |

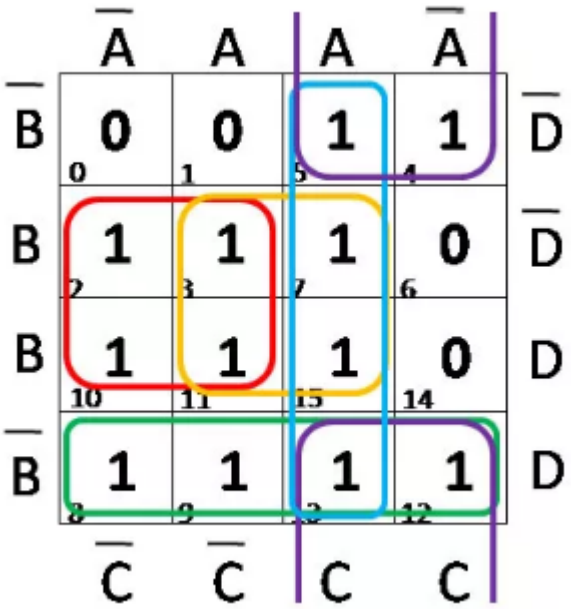



Als Ergebnis für Z=1 erhalten wir die Gleichung

$$\bar{C} \vee B \wedge \bar{C} \vee B \wedge A$$

Möglich sind auch folgende Zusammenfassungen:

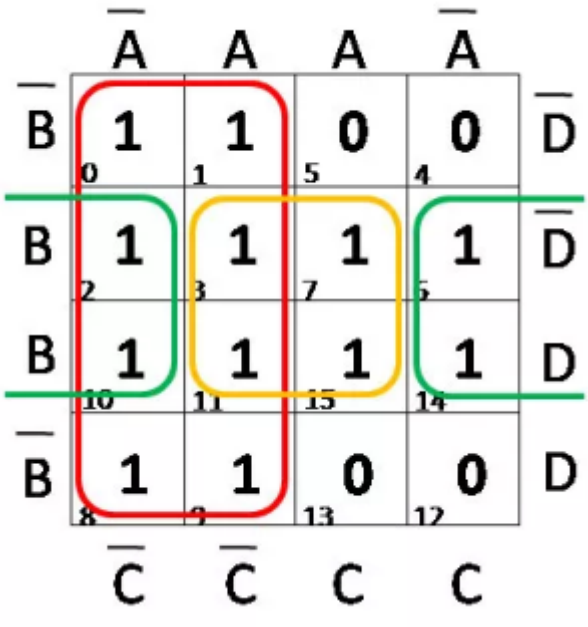
Auch Eckfelder sind benachbarte Felder und können zusammengefasst werden.




Das Ergebnis ist:

$$B \wedge \bar{C} \vee B \wedge A \vee \bar{B} \wedge D \vee A \wedge C \vee \bar{B} \wedge C$$

... oder auch:




Das Ergebnis lautet dann wie folgt:

$\overline{C} \vee B \wedge \overline{A} \vee B \wedge A$

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

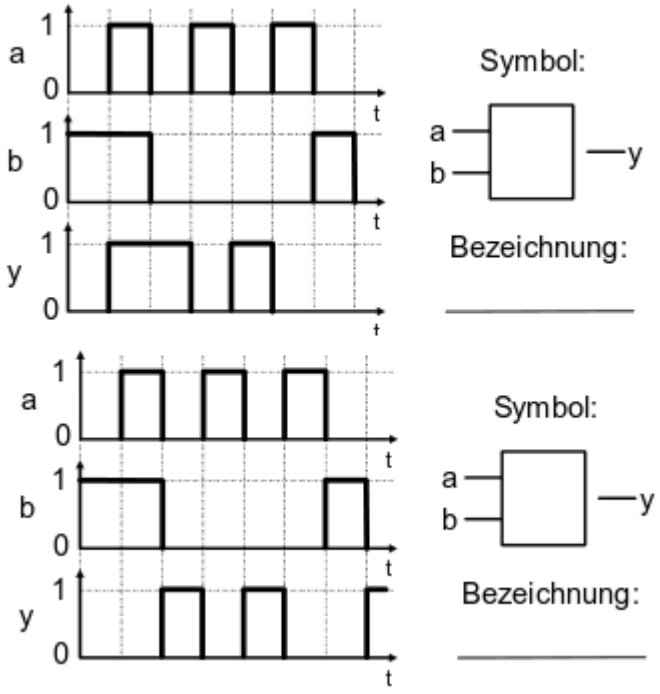
Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:04_schaltalgebra:4_06**

Last update: **2023/09/10 13:17**

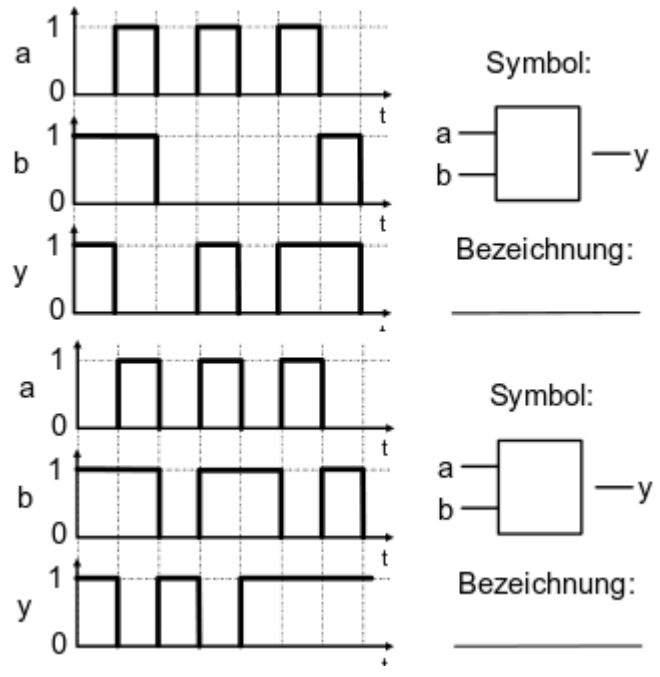
# 4.07) Übungen

1) Um ihren Swimmingpool schneller füllen zu können, hat Fam. Huber zwei Wasserleitungen (beide verfügen über einen Absperrhahn B und C verlegt, die direkt in den Pool führen. Diese Leitungen zweigen beide von einer Hauptleitung ab, die ebenfalls über einen Absperrhahn A verfügt. Demnach kann Wasser nur in den Pool fließen, wenn der Hauptabsperrhahn A und zumindest einer der anderen Absperrhähne offen sind.
Stelle die Situation mittels einer Wahrheitstabelle und einer Schaltfunktion (LogikSim, ...) dar.

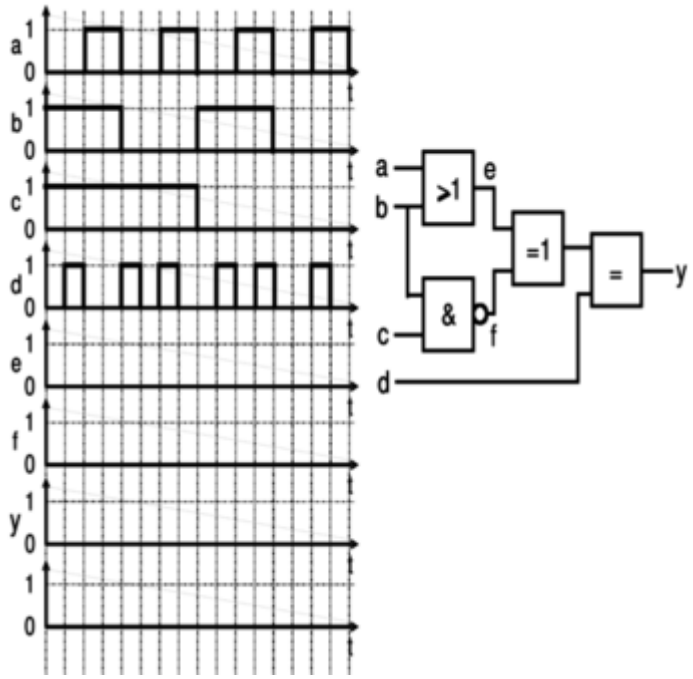
2) Zu welchen logischen Schaltungen gehören die folgenden Zeitablaufdiagramme?



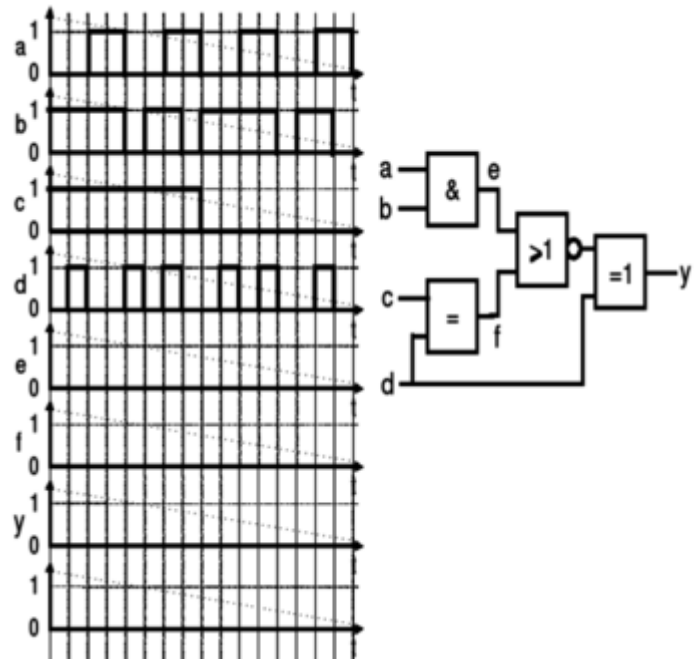

3) Zu welchen logischen Schaltungen gehören die folgenden Zeitablaufdiagramme?



4) Vervollständige das Zeitablaufdiagramm




5) Vervollständige das Zeitablaufdiagramm




6) Zeichne in Logiksim für folgende schaltalgebraischen Ausdrücke die elektronische Schaltung und erstelle die jeweilige Wahrheitstabelle!

- f(A,B) = (¬A ∨ ¬B)
- f(A,B,C) = (¬A ∧ ¬B) ∧ C
- f(A,B) = (¬A ∨ ¬B) ∨ (¬A ∧ B)

7) Überprüfe das Gesetz von De Morgan mit Hilfe einer Wahrheitstabelle und stelle die Schaltung im Digitalsimulator dar.

8) Überprüfe das Distributivgesetz mittels einer Wahrheitstabelle und stelle die Schaltung im

Digitalsimulator dar.

9) Vereinfache die Schaltfunktion f(a,b)= a ∧ (¬a ∨ b) und baue die Schaltung im Digitalsimulator auf.

10) Vereinfache die Schaltfunktion f(a,b)= (a ∨ b) ∧ (a ∨ ¬b) ∧ (¬a ∨ b)

11) Vereinfache die Schaltfunktion f(a,b)= (a ∧ b) ∨ (a ∧ ¬b) ∨ (¬a ∧ b)

12) Vereinfache folgende Terme:
a) (¬A ∨ ¬B) ∧ (¬A ∨ B) ∧ (A ∨ ¬B)
b) (A ∧ B) ∨ (A ∧ C) ∨ (B ∧ ¬C)
c) (A ∧ ¬B) ∨ (A ∧ ¬B ∧ C)
d) (A ∨ ¬(B ∧ A)) ∧ (C ∨ (D ∨ C))
e) (¬(A ∧ B) ∨ ¬C) ∧ (¬A ∨ B ∨ ¬C)
f) ¬(¬(A ∧ B) ∨ C) ∨ (A ∧ C)
g) (A ∨ B) ∧ (¬A ∨ B) ∧ (A ∨ ¬B) ∧ (¬A ∨ ¬B)
h) A ∨ (¬B ∧ ¬(A ∨ ¬B ∨ C))

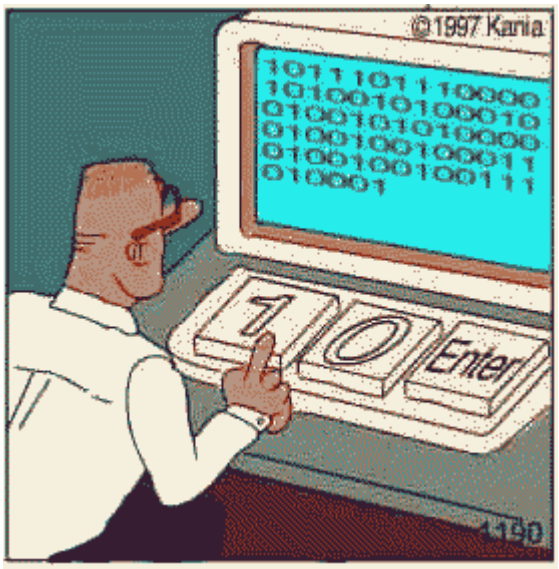
13) Zur automatischen Brandbekämpfung wurden drei Sensoren in einem Raum angebracht. Melden mindestens zwei der drei Sensoren eine Rauchentwicklung, so schaltet sich die Sprinkleranlage ein. Entwirf eine Leitwerttabelle, eine Schaltfunktion und stelle diese mittels Digitalsimulator dar!

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:04_schaltalgebra:4_07**

Last update: **2023/10/27 12:35**

# 5) Algorithmik und Programmierung



Real programmers code in binary.



From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik**

Last update: **2023/10/23 05:09**

# 5.1) Grundlagen zur Programmierung

## Programm

Im Alltag sind uns Programme geläufig, wie z.B.:

- das Programm einer Feier,
- ein Kochrezept,
- oder das Programm einer politischen Partei.

**Diese beschreiben einen Ablauf d.h. was wie zu tun ist.**
**In der Informatik wird der Begriff enger gefasst und man spricht von der Programmierung**.

## Programmierung

**Unter Programmierung versteht man das Schreiben solcher Programme in Form von Quellcode d.h. Befehle in einer Text Datei**.

**Das Verfassen von Quellcode erfolgt in einer Programmiersprache (C++, Java, Python, Fortran, …)**

**Diese Programmiersprachen werden von Softwareprogrammen in Maschinensprache übersetzt, welche ein Computer versteht und ausführen kann.**

Somit kann man mit Programmierung das Verhalten eines Computers steuern.

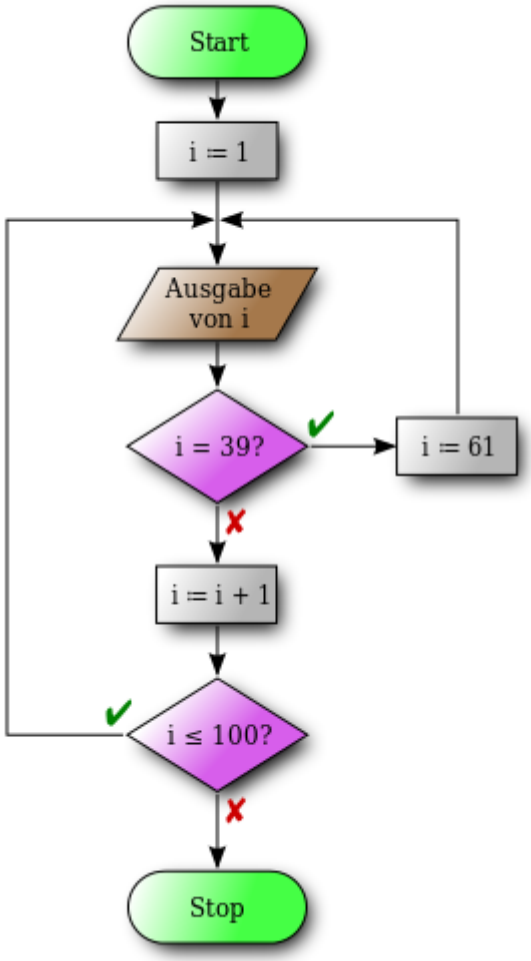
## Algorithmus

**Ein Algorithmus ist eine eindeutige Abfolge von Anweisungen (Befehlen) zur Lösung eines Problems.**

Beispiel eines Algorithmus:

```
ALGORITHMUS mach_was()
   ANFANG mach eine Aufgabenliste
   SOLANGE was zu tun ist
   WENN nicht erledigt
      mach weiter
   SONST
      geh zur nächsten Aufgabe
   WIEDERHOLE
ENDE
```

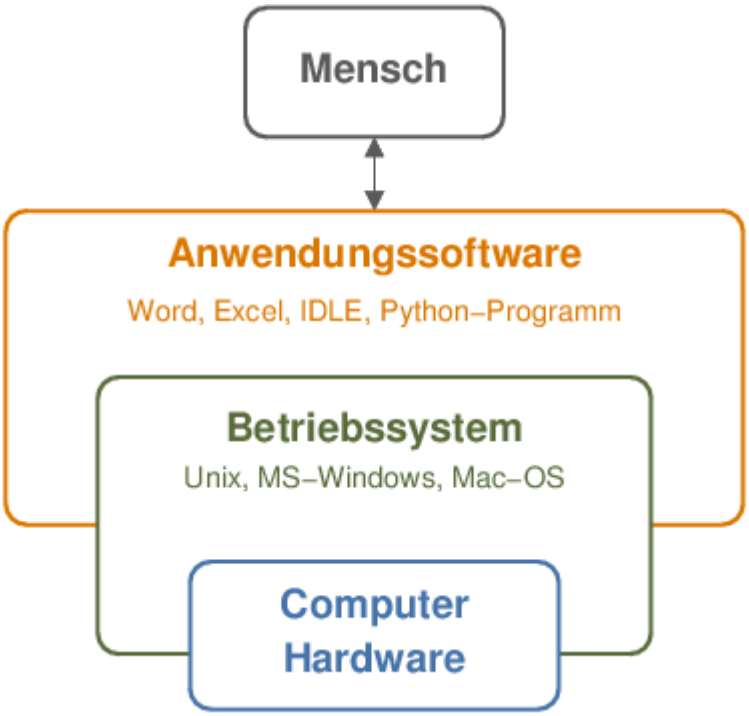
Ein Algorithmus kann auch graphisch in Form eines Flussdiagramms dargestellt werden.




**Ein Programm besteht im Allgemeinen aus vielen Algorithmen!**

# Computersystem

Ein Computersystem kann als Schichtmodell gesehen werden.




From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_01**

Last update: **2023/10/23 05:09**

# 6.2) Programmiersprachen

**Eine Programmiersprache zeichnet sich - wie jede Sprache - durch Syntax und Semantik aus.**

## Syntax

**Legt fest in welcher Folge Zeichen den Programmtext bilden.**

Z.B. Python-Syntax zur Definition einer Variablen:

```
a = 1.843
```

Nicht erlaubt wäre z.B.:

```
a 1.843
```

Bei anderen Programmiersprachen würde dies wie folgt aussehen:

```
real a = 1.843     # Fortran
float a = 1.843;   # C++, Java
$a = 1.843;        # Perl
```

## Semantik

**Beschreibt die Bedeutung eines Programmtextes**

```
print "Alles Python oder was?"
```

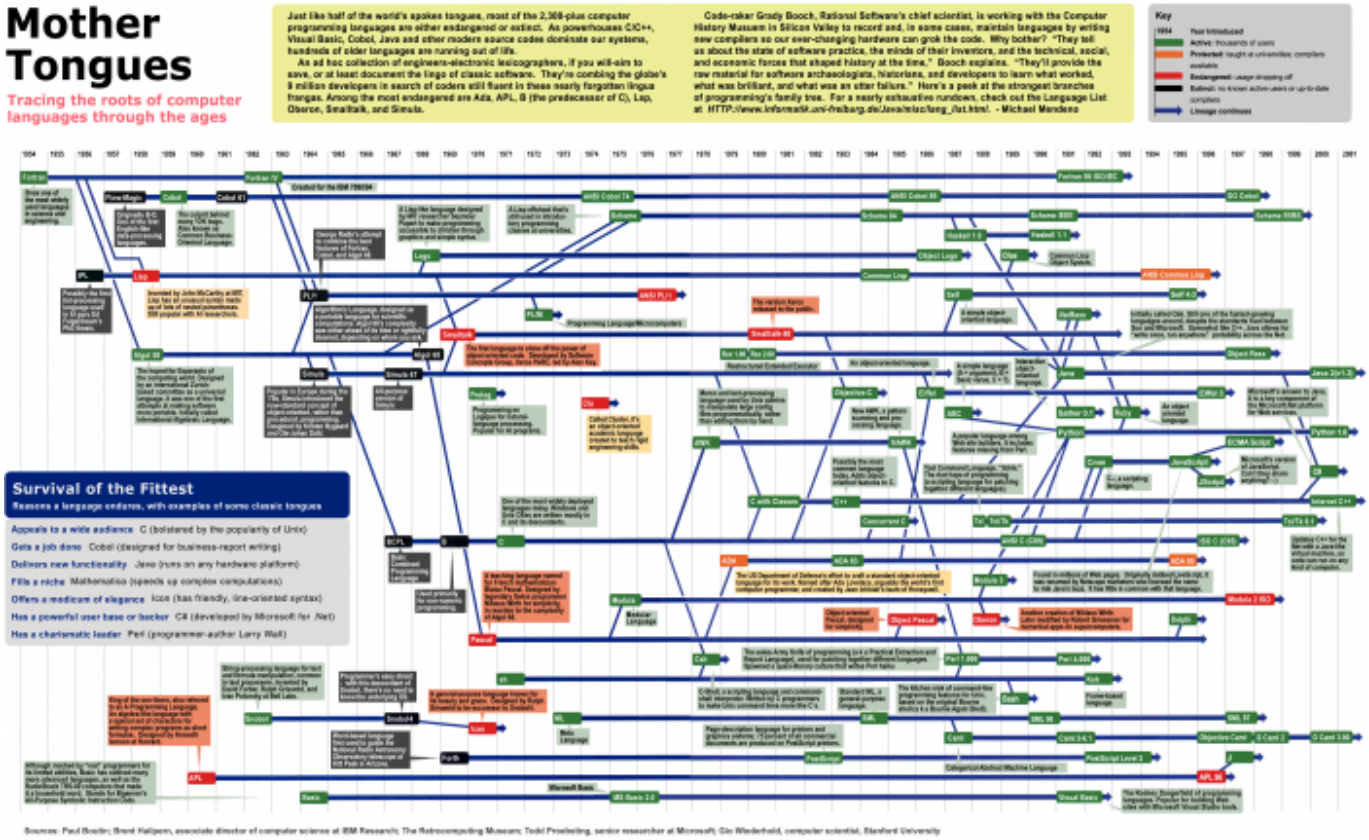
Dieser Programmtext gibt einen Text am Bildschirm aus. Ein weiteres Beispiel :

```
if a < 0 :
  a = a*(-1)
```

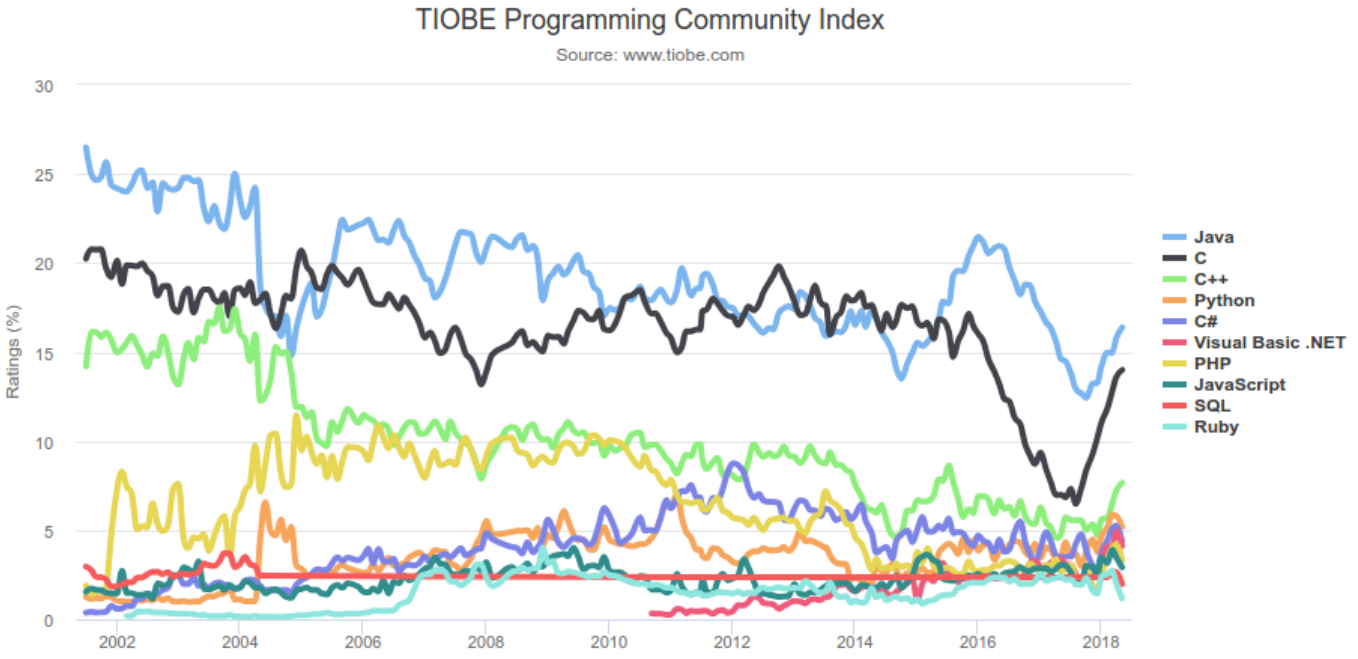
Dieser Code macht aus a eine positive Zahl.

## Wichtige Programmiersprachen

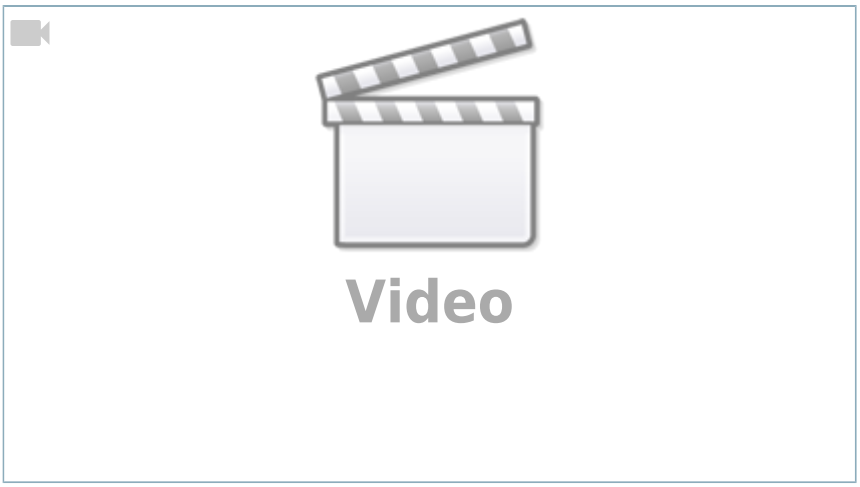
Die Liste der Programmiersprachen ist lang. Anbei eine Abbildung über die Entwicklung wichtiger Programmiersprachen.




Anbei noch eine Abbildung über die Popularität von Programmiersprachen




Aber auch Experten wie **Bjarne Stroustrup** - der **Erfinder von C++** meint dazu im Video zu den 5 wichtigsten Programmiersprachen:




**… Experten kennen mehr als nur eine Sprache, kennt man diesen Cluster, kennt man alle anderen!**

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_02**

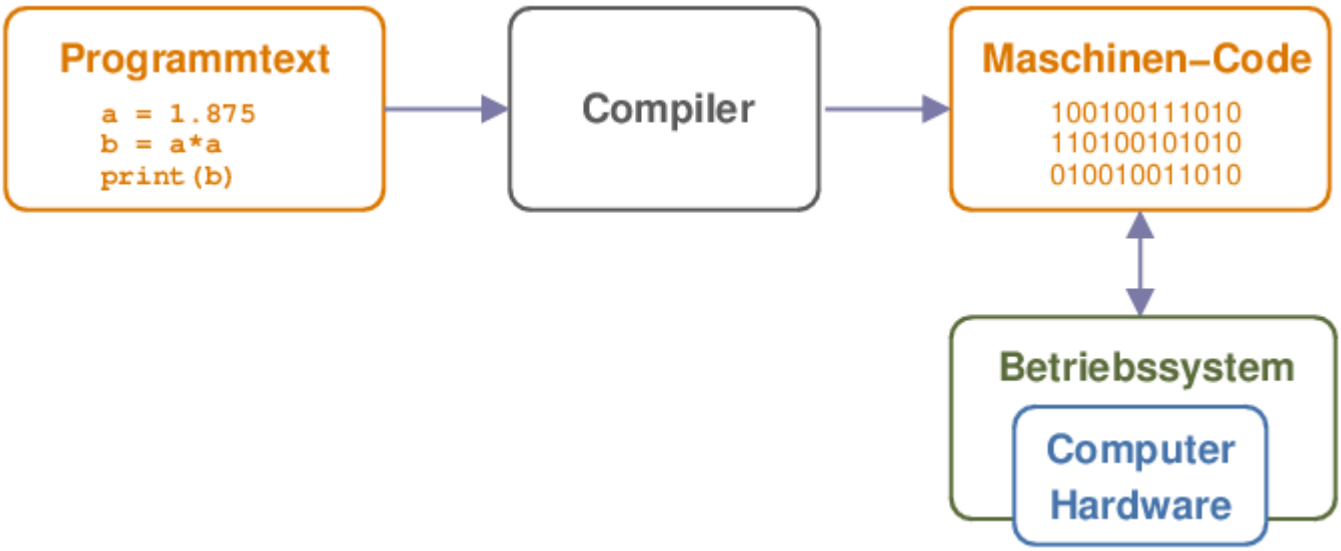
Last update: **2023/10/23 05:10**

# 5.3) Übersetzung von Quellcode in Maschinensprache

Es gibt **2 unterschiedliche Methoden** wie **Programmtext (Quelltext, source code) in Computersprache übersetzt** wird d.h. vom Computer **verstanden** wird.

## Compiler

Der Programmtext wird nach Erstellung komplett in Maschinen-Code (executeable) übersetzt (siehe Arbeitsweise eines Compilers) und ausgeführt.

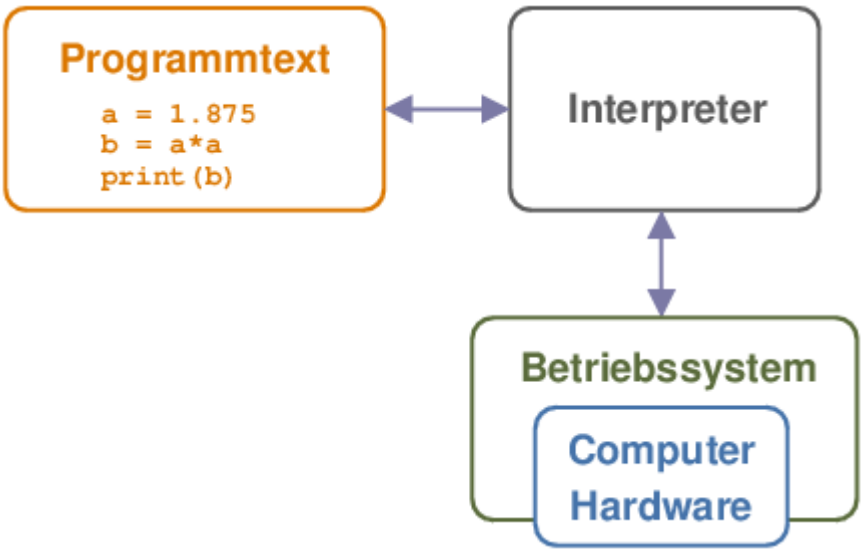



### Vor- und Nachteile

- benötigt Compiler (=eigene Anwendungssoftware)
- läuft nur auf dem Betriebssystem, wo Programm kompiliert wurde
- i.a. sehr schnell (high performance)
- Quelltext nach kompilieren nicht einsehbar
- typische Sprachen: Fortran, C++

## Interpreter

Der Programmtext wird Zeile für Zeile an das Betriebssystem geschickt und ausgeführt (siehe Arbeitsweise eines Interpreters).




## Vor- und Nachteile

- benötigt Interpreter (=eigene Anwendungssoftware)
- unabhängig vom Betriebssystem
- schnell implementiert, langsam exekutiert
- Quelltext einsehbar, schnell änderbar
- typische Sprachen: JavaScript, Python

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_03**

Last update: **2023/10/23 05:10**

# 5.4) Programmierstile

Quellcode kann **imperativ (z.B. prozedural)** oder **objektorientiert** geschrieben werden.

## Imperativ

Kommt aus dem lateinischen imperare = anordnen d.h. einzelnen Befehle werden **sequentiell hintereinander** geschrieben mittels:

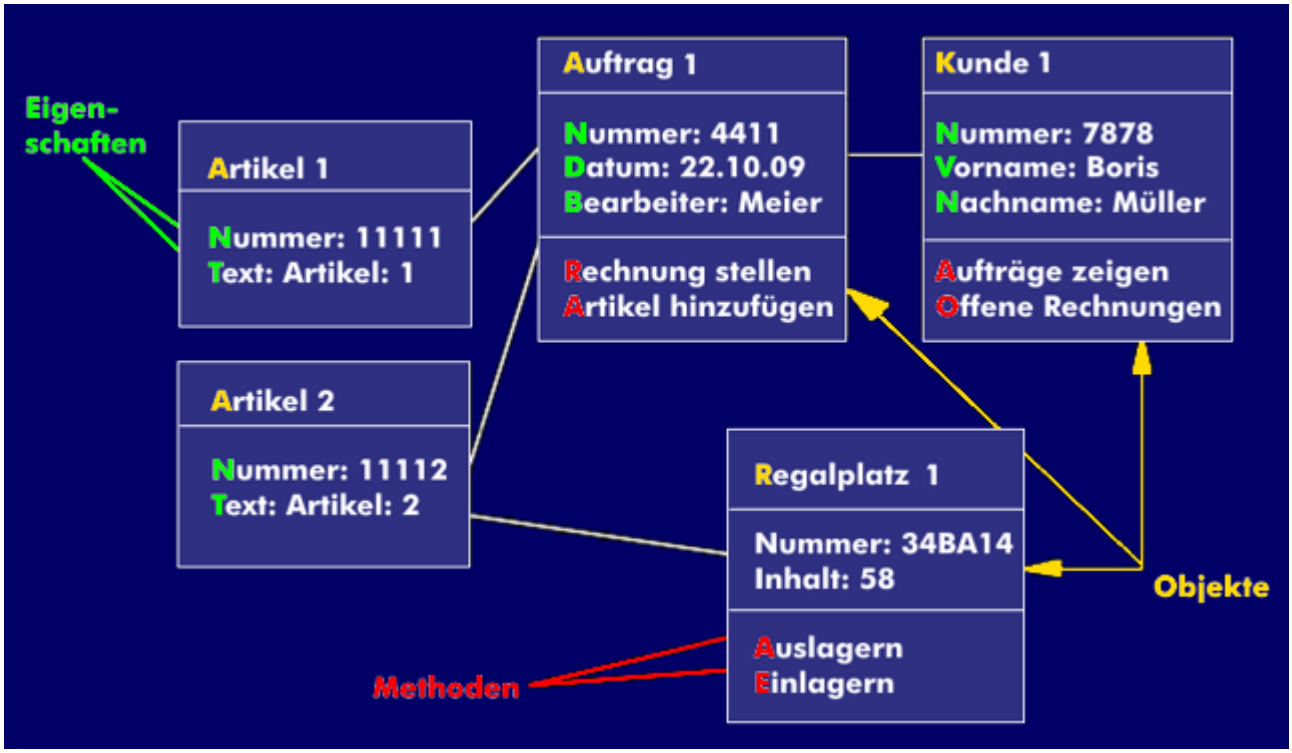
- Kontrollstrukturen = Strukturierte Programmierung
- Überschaubare Unterprogrammen = Prozedurale Programmierung
- Logisch zusammengefassten Unterprogrammen = Modulare Programmierung

### Vor- und Nachteile

- für kleine Programme geeignet
- i.a. sehr schnell, da Compiler den Code gut optimieren kann
- wird schnell unübersichtlich, schwer wartbar, ...

## Objektorientiert

Das Programm wird als **System von Objekten** dargestellt, die **miteinander kommunizieren** (siehe Beispiel objektorientierte Programmierung).




**Erklärungen:**

- Zunächst definiert man **sogenannte Klassen** (z.B. Klasse “Artikel”). Ein Programm kann beliebig viele unterschiedliche Klassen haben (Artikel, Auftrag, Kunde, ...).
- Erzeugt man nun einen bestimmten “Artikel” (z.B. Artikel 1, anders gesagt gibt man diesen spezifische Daten), spricht man von einem **Objekt**.
- Diesen Vorgang bezeichnet man auch als **das Objekt (Artikel 1) wird von der Klasse (Artikel) instantiiert**. Man kann viele Objekte einer Klasse erzeugen (Artikel 1, Artikel 2, ..).
- Jedes Objekt besitzt (anhand der Klassendefinition) **Eigenschaften (Attribute)**, dies sind die Daten (z.B. Nummer, Text, ...) und **Methoden (Algorithmen)**, welche diese Daten ändern, anzeigen, bereitstellen, etc (z.B. auslagern, einlagern, ...).
- Ein Hauptprogramm enthält z.B. alle hier dargestellten Objekte in Form von Listen und steuert den Programmablauf inklusive Daten Ein- und Ausgabe.

## Vor- und Nachteile

- für große Programmierprojekte gut geeignet
- Konzepte wie Vererbung, Datenkapselung, Polymorhismus, etc umsetzbar

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_04**

Last update: **2023/10/23 05:10**

# Darstellung von Algorithmen bzw. von Programmlogiken

Es gibt verschiedene Darstellungsformen, um algorithmische Grundbausteine und Algorithmen darzustellen. An dieser Stelle soll auf die drei wichtigsten eingegangen werden:

- Pseudocode
- Struktogramm
- Programmablaufplan

## Pseudocode

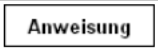
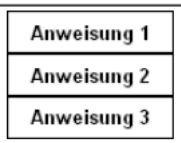
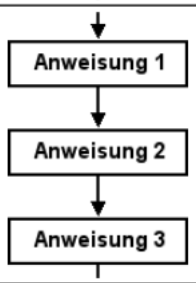
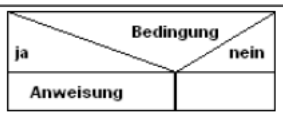
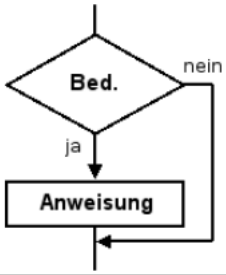
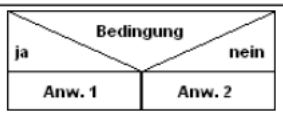
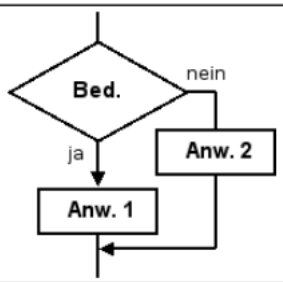
Der Pseudocode stellt einen **Algorithmus in einer Schreibweise** in der Art einer Programmiersprache dar, die aber in der Regel **näher an der natürlichen Sprache ist als am formalen Programmcode**. Es gibt **keine Standardisierung bzw. Vorschrift**, wie ein Pseudocode auszusehen hat; die Ablaufstrukturen werden grob durch Texte und fest vorgeschriebene Schlüsselwörter beschrieben. Guter Pseudocode zeichnet sich durch eine **kurze präzise Beschreibung des Algorithmus** aus.

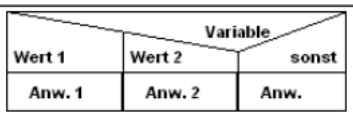
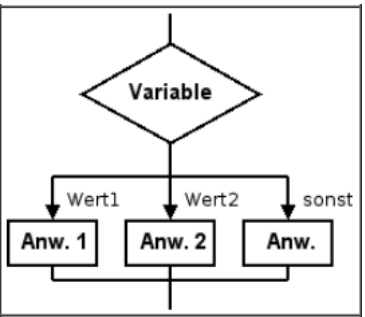
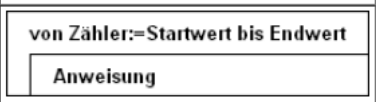
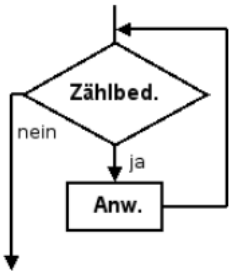
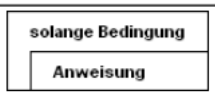
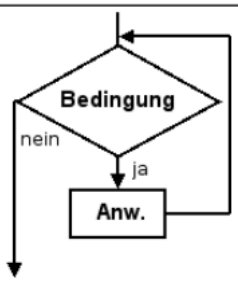
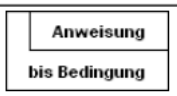
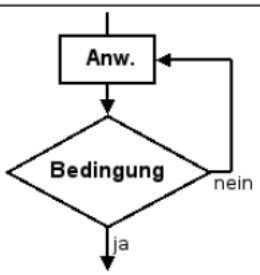
## Struktogramm

Struktogramme (auch nach ihren Entwicklern **Nassi-Shneidermann-Diagramme** genannt) ist eine **genormte (DIN 66261) Entwurfsmethode für Algorithmen**. Die Methode **zerlegt ein Gesamtproblem**, das man mit dem gewünschten Algorithmus lösen will, in immer **kleinere Teilprobleme** bis schließlich nur noch **elementare Grundstrukturen wie Sequenzen und Kontrollstrukturen** zur Lösung des Problems übrig bleiben.

## Programmablaufplan (PAP)

Der **Programmablaufplan (auch Flussdiagramm oder Programmstrukturplan)** ist eine **graphische Darstellung zur Umsetzung eines Algorithmus** in einem Programm und beschreibt die **Folge von Operationen zur Lösung einer Aufgabe**. Die Symbole für Programmablaufpläne sind in der DIN 66001 genormt.

| Strukturelement | Pseudocode | Struktogramm | Programmablaufplan |
|---|---|---|---|
| **Verarbeitung** (Elementarblock) | `Anweisung` |   |   |
| **Reihenfolge** (Sequenz) | `Anweisung1`<br>`Anweisung2`<br>`Anweisung3` |   |   |
| **Verzweigung** (Alternative) | `wenn Bedingung`<br>`  dann Anweisung` |   |   |
| | `wenn Bedingung`<br>`  dann Anweisung1`<br>`  sonst Anweisung2` |   |   |

| | | | |
|---|---|---|---|
| **Fallunterscheidung** | für Variable<br>  Wert 1 : Anweisung1<br>  Wert 2 : Anweisung2<br>  ...<br>  [sonst Anweisung] |  |  |
| **Zählschleife** | von Zähler:=Startwert bis Endwert<br>  Anweisung |  |  |
| **anfangsgeprüfte Schleife** | solange Bedingung<br>  Anweisung |  |  |
| **endgeprüfte Schleife** | wiederhole<br>  Anweisung<br>bis Bedingung |  |  |

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_05**

Last update: **2023/10/23 05:11**

# Code.org

Code.org ist eine 2013 in den USA gegründete Non-Profit-Organisation und gleichnamige Website, die möglichst viele Menschen und insbesondere Kinder für das Thema Informatik (im englischen „Computer Science“) und Programmieren begeistern will.

- Kurs 3
- Kurs 4

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_06**

Last update: **2023/10/23 05:11**

# Scratch

Mit Scratch kannst du deine eigenen interaktiven Geschichten, Spiele und Animationen programmieren und deine Kreationen mit anderen in der Gemeinschaft online teilen.

Scratch hilft jungen Leuten, kreativ zu denken, systematisch zu schließen und miteinander zusammenzuarbeiten — grundlegende Fähigkeiten für das Leben im 21. Jahrhundert.

## Beispielhafte Scratch-Projekte

- Scratch-Games

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_07**

Last update: **2023/10/23 05:14**

# C++ bzw. CPP

C + + (zeh plus plus ausgesprochen) ist eine kompilierte Allzweck Programmiersprache. Ihre Komplexität liegt zwischen Einsteiger und Fortgeschritten, da sie sowohl High-Level als auch Low-Level Sprachelemente vereint. Sie bietet imperative, objekt-orientierte und generische Programmiermerkmale.

C + + ist eine der beliebtesten Programmiersprachen und ist auf vielen Hardwareplattformen und Betriebssystemem verfügbar. Da sie besonders effizient ist, wird es für Systemprogramme, Applikationen, Gerätetreiber, eingebettete Software, Server-Client und Unterhaltungsprogramme, z. B. Videospiele, eingesetzt. Verschiedene Hersteller bieten Open Source und proprietäre C + + Compiler an, bspw. FSF, LLVM, Microsoft und Intel.

## Entwicklungsumgebung DEV C++

- Download DEV C++

## Los gehts!

- C++ Skriptum
- Einstieg in C++
- WikiBook C++
- Learn C++
- C++ spielerisch lernen

## Programmierung in C++

- 5.8.1) Grundgerüst
- 5.8.2) Variablen
- 5.8.3) Datentypen
- 5.8.4) Verarbeitung
- 5.8.5) Ein- und Ausgabe
- 5.8.6) Ablaufsteuerung
- 5.8.7) Funktionen

## Übungen

- 5.8.6.3) Übungen

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08**

Last update: **2024/01/22 06:08**

# 5.8.1) Grundgerüst eines Programms

Ein **Programm** ist eine **Aneinanderreihung von Befehlen**, die dem Computer zur Abarbeitung zugeführt werden können. Wie schon erwähnt, wird ein **C++-Programm in einem Texteditor** geschrieben und dann von einem **Compiler in ausführbaren Code übersetzt**. Danach kann es gestartet werden.

## Hauptprogramm - main()

Ein C++-Programm besteht **mindestens** aus der **Funktion main()**. Wichtig ist, dass Sie in **jedem C++-Programm irgendwo den Namen main** mit einem **Klammerpaar** finden werden. In manchen Fällen stehen **zwischen den Klammern Variablen für die Parameterübergabe**. Aber auch das sollte Sie zu Anfang nicht stören. Ignorieren Sie es von ganzem Herzen, bis Sie wissen, wozu Sie es brauchen.
Die **Hauptfunktion beginnt direkt nach einer öffnenden geschweiften Klammer und endet mit der schließenden Klammer**. **Zwischen den geschweiften Klammern** stehen die **Befehle der Hauptfunktion**. Sie werden nacheinander ausgeführt. Diese Hauptfunktion enthält nichts und ist damit das kleinste denkbare C++ Programm.

```
int main()   //Hauptfunktion main()
{

   return 0;  //Optionale Rückgabe an das Betriebssystem
}
```

Dieses Programm tut gar nichts. Wenn es etwas täte, stünden die Befehle zwischen den geschweiften Klammern.

## Kommentare

Wenn wir schon einmal ein Programm haben, das nichts tut, wollen wir es auch um **Befehle** erweitern, **die nichts tun: KOMMENTARE.**

### Freundlichkeit unter KollegInnen

Sie können in das Programm Kommentare einfügen, die **vom Compiler völlig ignoriert** werden. Der **Zweck** solcher Kommentare ist es zu **dokumentieren**, warum das Programm so und nicht anders geschrieben wurde. Die **Kommentare richten sich an Programmierer**. Dies können **Kollegen** sein, die Ihr **Programm korrigieren** oder **erweitern** sollen. Aber noch öfter helfen Sie sich selbst damit. In der Praxis ist es so, dass Sie vermutlich auch herangezogen werden, wenn eines Ihrer Programme nicht korrekt läuft oder ergänzt werden soll. Da Sie zwischendurch andere Programme geschrieben haben werden, vielleicht geheiratet haben, umgezogen sind und noch drei weitere Programmiersprachen gelernt haben, werden Sie sich **nicht mehr an jedes Detail erinnern**. Sie werden dankbar sein, wenn Sie in den Programmen ausführliche Hinweise finden, wie und warum es

so funktioniert oder zumindest funktionieren soll.

Es gibt **zwei Arten**, einen Text davor zu schützen, dass der Compiler versucht, ihn als Programm zu interpretieren.

## Zeilenweise

Die einfachere Art der Kommentierung ist es, **zwei Schrägstriche** direkt hintereinander zu setzen. Damit gilt der **Rest der Zeile als Kommentar**.

```
int main()
{
    // Hier beginnt der Kommentar.
    // Die naechste Zeile braucht ihr
    // eigenes Kommentarzeichen.

    return 0;
}
```

## Kommentarblock

Daneben gibt es die Möglichkeit, einen **größeren Text in Kommentarklammern einzuschließen** und damit dem Compiler zu entziehen. Der Anfang des Kommentars wird durch den Schrägstrich, gefolgt von einem Stern, festgelegt. Der Kommentar endet mit der umgekehrten Zeichenfolge, also mit einem Stern und dann einem Schrägstrich.

```
int main()
{
  /* Hier beginnt der Kommentar.
     Die naechste Zeile braucht kein
     eigenes Kommentarzeichen.
   */

   return 0;
}
```

**!! ACHTUNG !!** : Eine Verschachtelung von Kommentarblöcken ist nicht möglich!

```
int main()
{
  /* Hier beginnt der Kommentar
     /*
       Die naechste Zeile braucht kein
       eigenes Kommentarzeichen
      */
     Dies wird der Compiler wieder uebersetzen wollen.
  */
```

```
  return 0;
}
```

# Anweisungen

Anweisungen sind die **Befehle, aus denen ein Programm** besteht. Eine Anweisung kann dazu dienen, etwas zu **speichern**, zu **berechnen**, zu **definieren** oder auf dem Bildschirm **auszugeben**. **Anweisungen sind die Grundeinheiten**, aus denen **Programme bestehen**.

# Blöcke

## Zusammenfassen von Anweisungen

**Mehrere Anweisungen** können **zu einem Block zusammengefasst** werden. Ein Block wird durch **geschweifte Klammern** eingefasst. Der **Compiler** wird einen **Block wie eine einzige Anweisung** behandeln. Sicher ist Ihnen schon aufgefallen, dass die Hauptfunktion main() ebenfalls solche **geschweiften Klammern** hat. In der Tat ist dies der Block, in dem die Befehle des Programms zusammengefasst werden.

## Einrückungen

Um die **Übersicht zu erhöhen**, pflegt man **alle Befehle innerhalb eines Blocks einzurücken**. Achten Sie vor allem am Anfang darauf, dass die **zusammengehörigen geschweiften Klammern auf einer Ebene** stehen. Das folgende Beispiel mit Pseudobefehlen zeigt, wie das Einrücken korrekt ist.

```
int main()
{
    Hier sind Pseudo-Anweisungen;
    Diese gehört dazu;
    {
        Neuer Block, also einrücken;
        Wir bleiben auf diesem Niveau;
        Das kann ewig weitergehen;
    }
    Nun ist die Klammer geschlossen;
    Und die Einrückung ist auch zurück;

    return 0;
}
```

## Hello World!

Es ist eine **alte Tradition**, eine **neue Programmiersprache** mit einem **„Hello-World“-Programm**

**einzuweihen**. Auch dieses Buch soll mit der Tradition nicht brechen, hier ist das „Hello-World“-Programm in C++:

```
#include <iostream>                                          // Ein- und
Ausgabebibliothek

int main(){                                                  // Hauptfunktion
    std::cout << "Hallo, du schöne Welt!" << std::endl; // Ausgabe

    return 0;                                                // Optionale
Rückgabe an das Betriebssystem
}
```

## Namensraum - namespace std

Die C++-Standardbibliotheken verwenden für alle Funktionen und Variablen den Namensraum std. Bei Verwendung von Typen, Funktionen oder Variablen aus den Standardbibliotheken müssten Sie jeweils std:: voranstellen. Sie können aber auch mit dem Befehl **using** erklären, dass Sie den Namensraum **std** verwenden wollen, als seien alle darin definierten Variablen und Funktionen ohne Namensraum definiert. Die Anweisung dazu lautet:

```
using namespace std;
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_01**

Last update: **2023/11/15 15:51**

# 5.8.2) Variablen

Die folgenden 4 Aussagen erläutern die Notwendigkeit von Variablen:

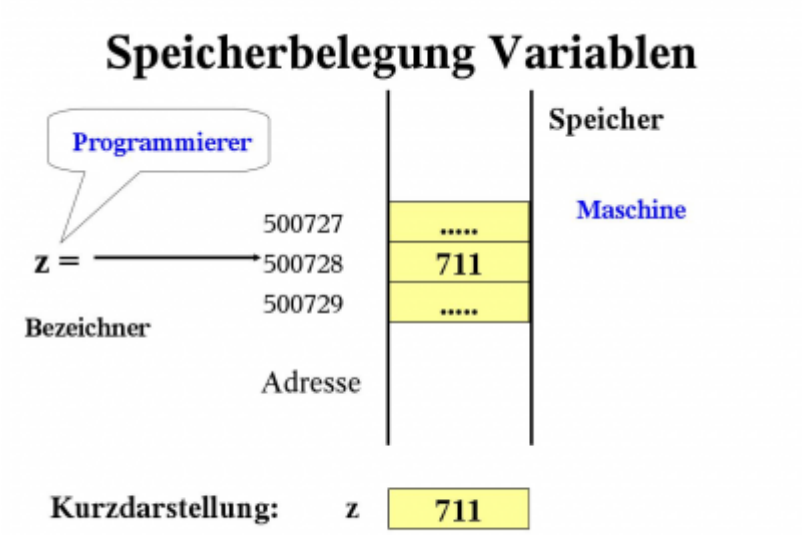
- Jedes Programm verarbeitet Informationen.
- Diese Informationen liegen im Speicher.
- Höhere Programmiersprachen greifen nicht direkt auf den Speicher zu, sondern verwenden Variablen.

⇒ **Variablen sind also die Behälter, in denen das Programm Zahlen und Texte ablegt.**

## Eigenschaften einer Variable

### SPEICHER

Die **Variable benötigt** zum Ablegen der Informationen immer **Speicher**. Der Speicher wird **vom Compiler zur Verfügung gestellt**. Über die **Lage und Größe muss sich der Programmierer NICHT kümmern**. Der Compiler ermittelt die Größe vom Datentyp und reserviert den dafür notwendigen Speicher.




### NAME

Die Variable wird im Programm über einen **weitgehend frei wählbaren Namen** angesprochen. Dieser **Name** identifiziert die Variable **innerhalb eines Blocks eindeutig**. C++ unterscheidet sogar zwischen **Groß- und Kleinschreibung**!

### DATENTYP

Der **Datentyp** einer Variablen **bestimmt, welche Informationen abgelegt werden können**. So kann eine Variable je nach ihrem Typ beispielsweise einen Buchstaben oder eine Zahl speichern. Der **Datentyp bestimmt** natürlich auch die **Speichergröße**, die benötigt wird. Der Typ bestimmt aber auch, **welche Operationen auf die Variable angewendet werden können**. So können Sie beispielsweise Zahlen durch einander dividieren. Mit Texten geht das nicht.

# Variablendefinition

Das folgende Beispiel zeigt eine Variablendefinition innerhalb der Hauptfunktion main():

```
int main()
{
   int Gehalt;
}
```

#### Datentyp

Der Datentyp ist in diesem Fall **int**. Dieser Datentyp steht für ganzzahlige Zahlen.

#### Name

Durch ein Leerzeichen abgesetzt, beginnt der Name der Variablen. Den **Namen** sollten Sie immer so wählen, dass Sie **auf den Inhalt schließen können**. Hier lässt der Name Einkommen bereits auf die Verwendung der Variablen im Programm schließen. **Verwenden Sie ruhig lange Namen**. Abkürzungen ersparen zwar Tipparbeit, Sie werden aber auf lange Sicht mehr Zeit verschwenden, wenn Sie darüber nachdenken müssen, was in welcher Variable abgelegt ist. Das gilt auch dann, wenn Sie extrem langsam tippen. Noch mehr Zeit werden Sie brauchen, wenn Sie einen Fehler suchen müssen, der entstand, weil Sie zwei Variablen verwechselt haben, nur weil der Name unklar war.

#### Semikolon (=Strich-Punkt)

Zu guter Letzt folgt das Semikolon. **Damit wird jede Anweisung abgeschlossen, auch eine Variablendefinition.**

#### Initialisierung

Sie können **Variablen gleich bei ihrer Definition mit einem Wert vorbelegen**. Dazu setzen Sie hinter den Variablennamen ein Gleichheitszeichen. Das **initialisiert die Variable mit dem nachfolgenden Wert**.

```
   int Gehalt=2000;
```

Damit wird bei der Erstellung die Variable Gehalt auf 2000 gesetzt. Normalerweise initialisiert man

einen neue Variable immer mit 0.

```
   int Gehalt=0;
```

#### Mehrfache Definition

Es können mehrere Variablen mit dem gleichen Datentyp hintereinander definiert werden, in dem sie durch Komma getrennt werden.

```
   int zahl, Gehalt=0, Freunde=1, Feinde=5;
```

## Geltungsbereich einer Variable

Eine Variable in C++ kann je nach ihrer Definition unterschiedliche Geltungsbereiche besitzen.

### Blockgrenzen

Während **in C** Variablendefinitionen **nur am Blockanfang vor der ersten Anweisung** erlaubt sind, kann in **C++ eine Variable überall definiert werden**.

**WICHTIG**: Sie **gilt** dann **für den gesamten restlichen Bereich des aktuellen Blocks und aller darin verschachtelten Blöcke**.

Es können sogar **in verschachtelten Blöcken Variablen mit dem gleichen Namen** verwendet werden. Dabei **überdeckt die innen liegende Definition diejenige, die außerhalb des Blocks liegt**. Folgendes Beispiel macht das deutlich:

```
int main()
{
    int a = 5;
    {
    // hier hat a den Inhalt 5
    }
    {
        int a = 3;
        // hier hat a den Inhalt 3
        {
            // a ist immer noch 3
        }
    }
    // hier ist a wieder 5

    return 0;
}
```

## Lokale Variable

Die **innere Variable wird als lokale Variable** bezeichnet. Für ihren Geltungsbereich **überdeckt sie die außen liegende Variable a**. Die **außen liegende Variable existiert durchaus noch**, aber **aufgrund der Namensgleichheit** mit der lokalen Variablen **kann man bis zu deren Auflösung nicht auf sie zugreifen**. Sobald das Programm den Block verlässt, in dem die lokale Variable definiert ist, wird die äußere Variable wieder sichtbar. **Alle Operationen auf der lokalen Variablen berühren die äußere Variable nicht**.

## Globale Variable

Variablen, die **außerhalb jedes Blocks definiert** werden, nennt man globale Variablen. Sie **gelten für alle Blöcke**, die nach der Definition im Quelltext auftreten, **sofern darin nicht eine Variable gleichen Namens lokal definiert wird**.

## Konstante

Eine Konstante ist ein **unveränderlicher Wert**. Alle Zahlen in einem Programm sind Konstanten, da ihr Wert, im Gegensatz zu Variablen, nicht geändert werden kann. In C++ haben **auch Konstanten einen Datentyp**.

### Deklaration einer Konstanten

Zahlenkonstanten die ohne weitere Erklärung im Programm (=Quellcode) stehen, haben keinen Dokumentationswert und können zu Verwechslungen bzw. Verwirrungen führen. Um die Lesbarkeit des Programms zu erhöhen, könnte man anstatt der Zahl eine Konstante definieren.
Nehmen Sie an, sie möchten ein Programm für die Zeiterfassung ihrer Mitarbeiter schreiben. Jeder Mitarbeiter soll jeden Tag 8 Stunden arbeiten. Anstatt die Zahl 8 zu verwenden, könnten Sie eine Konstante namens Regelarbeitszeit verwenden.
Ist diese Konstante einmal definiert so kann sie bei Programmausführung nicht mehr verändert werden.
Notwendig dafür, ist neben dem Datentyp und dem Variablennamen noch das Schlüsselwort **const**.

```
   const int Regelarbeitszeit=8;
```

Manchmal findet man in C++ auch noch die Form aus der Programmiersprache C.

```
  #define Regelarbeitszeit 8
```

### Beispiel für Variablen und ihren Geltungsbereich

```
#include <iostream>         //Bibliothek für Ein- und Ausgabe

#define Wochentage 7      //Alte Definition einer Konstanten namens
Wochentage mit dem Wert 7
```

```
int Arbeitsstunden=8;       //globale Variable

int main()       //Hauptfunktion
{
    const int Arbeitstage=5;       // Deklaration einer Konstanten namens
Arbeitstage initialisiert mit 5
    std::cout << Arbeitsstunden;      //Ausgabe der globalen Variable
Arbeitsstunden => 8
    {
        int Arbeitsstunden=5;         //Deklaration einer lokalen Variable
namens Arbeitsstunden initialisiert mit 5
        std::cout << Arbeitsstunden;        //Ausgabe der lokalen Variable
Arbeitsstunden => 5
    }

    std::cout << Arbeitsstunden;  //Ausgabe der globalen Variable
Arbeitsstunden => 8
    //Arbeitstage=4;                --> NICHT ERLAUBT, DA Arbeitstage EINE
KONSTANTE IST
    std::cout << Arbeitstage;     //Ausgabe der Konstante Arbeitstage => 5
    std::cout << Wochentage;      //Ausgabe der Konstante Wochentage => 7

    return 0;
}
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_02**

Last update: **2023/11/15 15:52**

# 5.8.3) Datentypen

C++ bietet viele eingebaute Datentypen als auch Möglichkeiten Datentypen selbst zu definieren.

In C++ gibt es prinzipiell 7 grundlegende Datentypen:

- bool ⇒ true oder false
- char ⇒ Buchstabe (8 Bit)
- int ⇒ Ganze Zahl (32 Bit)
- float ⇒ Fließkommazahl (32 Bit)
- double ⇒ Fließkommazahl doppelte Genauigkeit (64 Bit)
- void ⇒ nichts
- enum ⇒ Aufzählung

In der Standard Template Library (STL) ⇒ Standardbibliothek sind dann noch weitere Datentypen enthalten, wie z.B.:

- string ⇒ Zeichenketten
- complex ⇒ Komplexe Zahlen
- vector ⇒ Ähnlich wie Arrays
- valarray ⇒ Vektorrechnung
- list ⇒ Listen
- map ⇒ Assoziative Felder (Wörterbuch)

## Wahrheitswerte - bool

Der **Datentyp für Wahrheitswerte heißt in C++ bool**, was eine **Abkürzung für boolean** ist. Er kann nur **zwei Zustände** annehmen: **true (wahr)** oder **false (falsch)**. Obwohl eigentlich 1 Bit ausreichen würde, hat bool mindestens eine Größe von einem Byte (also 8 Bit), denn 1 Byte ist die kleinste adressierbare Einheit und somit die Minimalgröße für jeden Datentyp.

```
bool betrunken=false;
```

## Zeichen - char

**Zeichen sind eigentlich Ganzzahlen**. Sie unterscheiden sich von diesen nur bezüglich der Ein- und Ausgabe. **Jeder Zahl ist ein Zeichen zugeordnet**. Mit den Zahlen lässt sich ganz normal rechnen aber bei der Ausgabe erscheint das zugeordnete Zeichen auf dem Bildschirm. Welches Zeichen welcher Zahl entspricht, wird durch den verwendeten Zeichensatz festgelegt.

Die meisten Zeichensätze beinhalten den **sogenannten ASCII-Code (American Standard Code for Information Interchange)**, welcher die Zeichen 0 – 127 belegt. Er enthält **32 Steuerzeichen (0 – 31) und 96 druckbare Zeichen (32 – 127)**.

**char** ist der Standard-Datentyp für Zeichen. Er ist in der Regel **1 Byte** groß und kann somit **256 verschiedene Zeichen** darstellen. Diese genügen für einen erweiterten ASCII-Code, welcher zum

Beispiel auch deutsche Umlaute definiert. Für Unicode-Zeichen gibt es die Datentypen char16_t mit einer Größe von 2 Byte und char32_t mit einer Größe von 4 Byte. Früher nutzte man für Unicode auch den Datentyp wchar_t, welcher je nach System 2 oder 4 Byte groß war. Dieser Datentyp sollte jedoch nicht mehr eingesetzt werden.

# Ganzzahlen - int

Dieser Datentyp umfasst sowohl **ganzzahlige positive als auch negative Zahlen** aus dem Wertebereich von -2.147.483.648 bis 2.147.483.647. Es werden also vom Compiler für eine Variable des Datentyps int **32 Bit (4 Byte)** reserviert.

```
int x; // Variablendeklaration; die Variable x soll vom Typ int sein
x = 3; //Wertzuweisung, Initialisierung; die Variable x enthält den Wert 3
```

# Fließkommazahlen

In der realen Welt sind **ganzzahlige Werte oft nicht ausreichend**.

Bei Gewichten, Geschwindigkeiten und anderen Werten aus der Physik ist immer mit Nachkommastellen zu rechnen. Dieser Tatsache kann sich auch eine Computersprache nicht entziehen. Eine solche Fließkommazahl besitzt zwei Komponenten.

Die eine ist die **Mantisse**, und die andere ist der **Exponent zur Basis 10**.

Der Exponent wird durch ein kleines oder großes E abgetrennt. Die Zahl **1.5E2** hat den Gegenwert von **150**. Dabei ist **1.5** die **Mantisse**, **2** ist der **Exponent**.

## float

Der **einfachste Typ mit Nachkommastellen** heißt **float**. Die Zahlen dieses Typs sind binär kodiert, damit sie effizient im Computer verarbeitet werden können. Die Zahl lässt **ganzzahlige, aber auch negative Exponenten** zu. Dadurch sind nicht nur extrem große Zahlen darstellbar, sondern auch sehr kleine Brüche. Die Speicheranforderung einer float-Variablen ist nicht sehr hoch, typischerweise liegt sie bei **4 Bytes**.

```
float pi=3.14159265;
```

## double

Reicht die Genauigkeit nicht aus oder werden Größenordnungen benötigt, die über die Kapazität einer float-Variablen hinausgehen, steht der Typ double zur Verfügung. Der Name bedeutet »doppelt« und bezieht sich auf die **Genauigkeit**. Diese ist **doppelt so hoch wie bei float**.

```
double pi=3.14159265;
```

# Nichts - void

Der Datentyp **void** bedeutet **nichts**, ist **kein echter Datentyp** und wird überall dort **verwendet, wo kein Wert benötigt wird** oder vorhanden ist. Bei **Funktionen** wird void verwendet, wenn eine Funktion keinen Wert zurückgibt oder die Funktion keinen Parameter hat.

# Aufzählungen - enum

Mit **enum können Aufzählungstypen** definiert werden. Dazu gehören **Wochentage** oder **Farben**. Den **Elementen** eines Aufzählungstyps werden **Namen zugeordnet**, obwohl sie **intern natürlich als Zahlen codiert** werden

```
enum Farbe {ROT, GELB, GRUEN, BLAU};
```

# Zeichenketten - string

Die C++-Standardbibliothek enthält eine Klasse namens **string**. Um diese Klasse nutzen zu können, müssen Sie die gleichnamige Headerdatei string einbinden.

```
#include <iostream>  //Bibliothek für Ein- und Ausgabe-Befehle
#include <string>  // Bibliothek für Zeichenketten

using namespace std;

int main() {
    string zeichenkette;
    zeichenkette = "Hallo Welt!";
    cout << zeichenkette << endl;
}
```

# Datentyp prüfen

Mit der Bibliothek <typeinfo> und dem Befehl typeid(Variable).name() kannst du dir den Datentyp jeder Variable anzeigen lassen.

```
#include <iostream>
#include <typeinfo>

using namespace std;

int main(int argc, char** argv) {

    float test = 3.3;
```

```
    cout << typeid(test).name();


    return 0;
}
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_03**

Last update: **2023/11/19 09:34**

# 5.8.4) Verarbeitung

## Zuweisung

### Gleichheitszeichen

Bei der **Initialisierung** wurde **bereits das Gleichheitszeichen verwendet**, um eine **Variable mit einem Wert vorzubelegen**. Das Gleichheitszeichen wird auch ansonsten verwendet, wenn eine Variable einen **neuen Wert bekommen soll**. Dabei gibt es einen feinen Unterschied zwischen der Vorbelegung oder **Initalisierung**, die **beim Anlegen der Variablen** erfolgt, und der **Zuweisung**, wie die **Belegung** mit Werten genannt wird, **wenn die Variable bereits existiert**.

**Links vom Gleichheitszeichen steht immer das Ziel** der Zuweisung. Im Allgemeinen ist das eine Variable. Auf der **rechten Seite des Gleichheitszeichens steht die Datenquelle**. Das kann eine andere Variable, ein Zahlenwert oder eine Berechnung sein. Man bezeichnet die **Datenquelle auf der rechten Seite** einer Zuweisung **allgemein als Ausdruck**. Die englische Bezeichnung dafür ist **»expression«**. Ein Ausdruck ist Konstrukt, das einen Wert liefert und so als Datenquelle dienen kann. Das folgende Beispiel zeigt mehrere Zuweisungen. Dabei wird auch schon den Rechenoperationen ein wenig vorgegriffen.

```
int main()
{

    //Initialisierungen
    int MWStSatz=0, Netto=0;
    double MWStBetrag=0.0, Brutto=0.0;

    //Zuweisungen
    MWStSatz = 16;
    Netto = 200;
    MWStBetrag = Netto * MWStSatz / 100;
    Brutto = Netto + MWStBetrag;

    return 0;
}
```

## Kaskadierende Zuweisung

C++ hat die Besonderheit, dass Sie in einer Anweisung mehreren Variablen den gleichen Wert zuweisen können. Sie müssen sich das so vorstellen, dass eine Zuweisung ihren Wert nach links durchreicht. Diese Fähigkeit ermöglicht eine Zeile wie die folgende:

```
int main()
{
    int a,b,c,d;
```

```
    a = b = c = d = 5 + 2;

    return 0;
}
```

# Grundrechenarten

Letztlich sieht es nicht sehr viel anders aus, als wenn Sie sich eine Rechenaufgabe auf einen Zettel schreiben. Etwas ungewohnt ist lediglich, dass auf der linken Seite das Zuweisungsziel und ein Gleichheitszeichen steht. Das **Multiplikationszeichen ist der Stern**, und das **Divisionszeichen der Schrägstrich**. **Plus- und Minuszeichen** sehen so aus, wie man es erwartet. Sie können sogar das **Minuszeichen wie gewohnt als Vorzeichen** verwenden.

## Restrechnung bzw. Modulo

Eine besondere Rechenart ist die **Modulo-Rechnung**. Sie liefert den **Rest einer ganzzahligen Division**. Wenn Sie sich daran erinnern, wie Sie in den ersten Schulklassen dividiert haben, dann fallen Ihnen vielleicht noch Sätze ein wie: »25 geteilt durch 7 sind 3, Rest 4«. Diese Restberechnung gibt es auch unter C++. Man bezeichnet sie als die Modulo-Rechnung. Als **Operatorzeichen wird das Prozentzeichen** verwendet.

```
Rest = 25 % 7; // Rest ist also 4
```

## Punkt vor Strich

Auch in C++ werden die Gesetze der Mathematik geachtet. So wird die alte Regel **»Punktrechnung geht vor Strichrechnung«** auch hier eingehalten. Diese Regel sagt aus, dass die Multiplikation und die Division vor einer Addition oder Subtraktion ausgeführt werden, wenn beide gleichwertig nebeneinander stehen.

## Links nach rechts

Binäre Operatoren, also Rechensymbole, die zwei Ausdrücke miteinander verknüpfen, sind im Allgemeinen **linksbindend**. Das bedeutet, dass sie von links nach rechts ausgeführt werden, wenn die Priorität gleich ist. Das heißt, dass **a*b/c als (a*b)/c** ausgewertet wird. Eine **Ausnahme ist die Zuweisung**. Hier wird **a=b=c als a=(b=c)** ausgewertet. Der **Zuweisungsoperator ist also rechtsbindend**. Auch einstellige Operatoren sind rechtsbindend. Dazu gehört auch der Operator ++, den Sie im Laufe des Abschnitts noch kennen lernen werden.

# Abkürzungen

Der Mensch neigt zur Bequemlichkeit. Nicht anders geht es Programmierern. Sie handeln nach dem Grundgedanken, niemals etwas zu tun, was ein Computer für sie tun kann, und so ist es naheliegend,

dass es Wege gibt, wiederkehrende Aufgaben möglichst kurz zu formulieren.

## Inkrementieren

Um den Wert einer Variablen um 1 zu erhöhen, können Sie die folgende Zeile schreiben:

```
int Zaehler=0;
Zaehler=Zaehler+1;
```

Sind sie schreibfaul, so funktioniert auch folgende Variante:

```
int Zaehler=0;
Zaehler+=1;
```

Hier wird das Pluszeichen mit dem Gleichheitszeichen kombiniert. Damit das Plus nicht fälschlicherweise als Vorzeichen der 1 interpretiert wird, muss es vor dem Gleichheitszeichen stehen. Zwischen Plus- und Gleichheitszeichen darf kein Leerzeichen stehen. Die Bedeutung der Zeile ist also: **»Addiere der Variablen Zaehler den Wert 1 hinzu.«**

Das geht mit allen Rechenarten:

| kurze Schreibweise | lange Schreibweise |
|---|---|
| a+=b | a=a+b |
| a-=b | a=a-b |
| a/=b | a=a/b |
| a*=b | a=a*b |
| a%=b | a=a%b |

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_04**

Last update: **2023/11/15 16:05**

# 5.8.5) Ein- und Ausgabe

## Ausgabestrom nach cout

C++ verwendet für die **Ein- und Ausgabe** das **Datenstrom-Modell**. Der Datenstrom zur Ein- und Ausgabe wird in einer **eigenen Bibliothek (iostream)** behandelt. Um sie zu verwenden, muss vor der ersten Verwendung die folgende Zeile eingegeben werden:

```
#include <iostream.h>
```

In einigen Listings werden Sie auch sehen, dass iostream ohne die Erweiterung .h verwendet wird. Wenn Sie einen halbwegs aktuellen Compiler verwenden, ist beides möglich. Der Unterschied hat mit der Verwendung von Namensräumen zu tun.

Um Daten anzuzeigen, werden sie auf die **Datenausgabe cout** (Das »out« in cout ist englisch und bedeutet »aus«). Zunächst wird das Datenziel genannt. Dann werden zwei **Kleiner-Zeichen** verwendet, um die **Richtung der Daten** in Richtung der Ausgabe anzuzeigen.

```
cout << MeinZahlenWert;
cout << endl;
```

Im Beispiel wird die **Variable MeinZahlenWert ausgegeben**. In der Zeile darunter wird **endl** auf den Datenstrom gesendet. Dies ist keine selbst definierte Variable, sondern eine **vordefinierte Konstante für das Zeilenende**. Die Verwendung von endl sorgt auch dafür, dass alle anzuzeigenden Daten sofort auf dem Bildschirm erscheinen.

### Kaskadierung

Der Übersicht halber oder um sich Tipparbeit zu sparen, können mehrere Ausgabeobjekte direkt hintereinander, durch die doppelten Kleiner-Zeichen getrennt, aufgeführt werden.

```
cout << MeinZahlenWert << endl;
```

### Textausgabe

Sie können nicht nur Variablen, sondern auch **Konstanten (Text)** auf diesem Weg ausgeben. Das ist besonders interessant bei Zeichenketten, mit denen Sie Ihren Programmausgaben ein paar **erläuternde Texte beifügen** können.

```
cout << "Ergebnis: " << MeinZahlenWert << endl;
```

Im Sinne der Benutzerfreundlichkeit eines Programms, sagt der Programmierer dem Benutzer, dass der Wert, der jetzt auf dem Bildschirm erscheint, das Ergebnis des Programms ist.

# Eingabestrom aus cin

Um **Eingaben von der Tastatur zu lesen**, wird die Datenquelle **cin** (Das »in« in cin steht für »ein«. Es ist also das Eingabe-Objekt. Der Eingabeoperator besteht genau **spiegelverkehrt zum Ausgabeoperator** aus **zwei Größer-Zeichen**. Sie weisen quasi vom cin auf die Variable, in der die Eingabe abgelegt werden soll.

```
#include <iostream.h>

int main(void)
{
    int Zahleingabe;
    int Doppel;

    cout << "Bitte geben Sie eine Zahl ein!" << endl;
    cin >> Zahleingabe;
    Doppel = Zahleingabe * 2;
    cout << "Das Doppelte dieser Zahl ist "
        << Doppel << "." << endl;

    return 0;
}
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_05**

Last update: **2023/11/15 16:06**

# 5.8.6) Ablaufsteuerung

Bisher bestanden die Programme nur aus aneinander gehängten Anweisungen. Jedoch ist es oft notwendig, den Ablauf eines Programms aufgrund einer Bedingung bzw. von Abhängigkeit von Zuständen zu verändern. Hier ein Beispielfür ein besseres Verständnis zum Thema Kaffeekochen.

## Ablaufsteuerung - Beispiel "Kaffee kochen"

Folgende Anweisungen werden beim Kaffeekochen nacheinander ausgeführt:

```
Filtertüte in den Filter stecken
6 Löffel Kaffeepulver in die Filtertüte einfüllen
1 Liter Wasser in den Wasserbehälter einfüllen
Maschine einschalten
Warten
Maschine ausschalten
Kaffee entnehmen
```

Das ist allerdings eine sehr unvollständige Beschreibung des Ablaufs. In der Praxis werden Abläufe in Abhängigkeit von Zuständen erfolgen und Vorgänge wiederholt. Eine detailliertere Beschreibung des Kaffeekochens sieht so aus:

```
Wenn noch eine benutzte Filtertüte in der Maschine steckt
{
    Entnimm die Filtertüte
    Wirf sie in den Müll
}

Neue Filtertüte aus der Packung nehmen
Filtertüte in den Filter stecken

Wiederhole 6-mal:
{
    fülle einen Löffel Kaffeepulver in die Filtertüte
}

Wenn nicht genug Wasser im Wasserbehälter
{
    Wasser in den Wasserbehälter einfüllen
}

Maschine einschalten

Wiederhole:
{
    Warte eine Minute
```

```
} bis kein Kaffee mehr aus dem Filter tropft

Maschine ausschalten
Kaffee entnehmen
```

- Verzweigungen
- Schleifen

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_06**

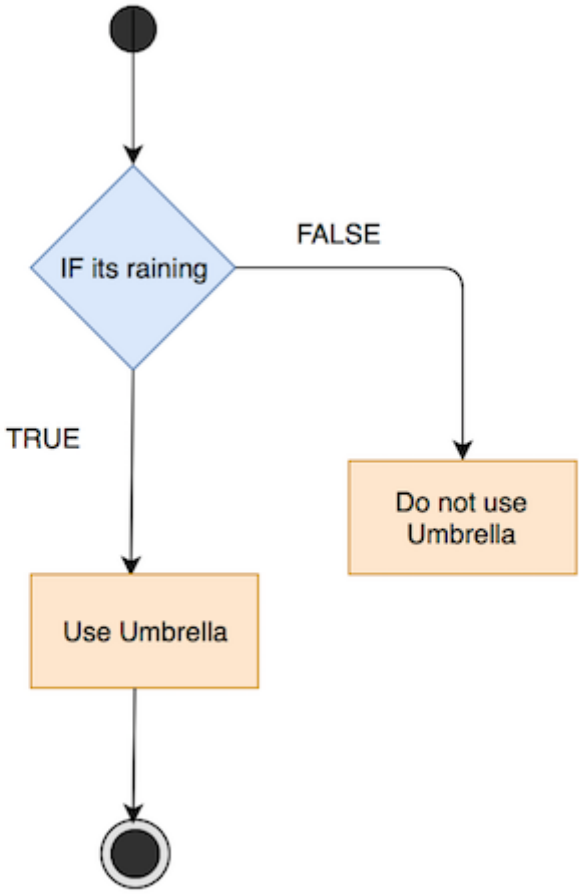
Last update: **2023/12/01 20:34**

# Verzweigungen

Es gibt diverse Gründe, warum ein Programm in Abhängigkeit von Variablen bestimmte Operationen nicht ausführen soll. Hier sind ein paar typische Beispiele zusammengestellt:

- Ganz offensichtlich darf ein Programm nicht dividieren, wenn der Nenner 0 ist.
- Vor dem Ziehen einer Wurzel ist es sinnvoll zu prüfen, ob der Operand negativ ist.
- Das Programm darf nicht mehr auf Aliens schießen, wenn die letzte Rakete des eigenen Raumschiffs verschossen ist.

Das Programm soll in all diesen **Fällen** in **Abhängigkeit von einem Variableninhalt** seinen **Ablauf ändern**. Dazu brauchen wir einen **Befehl, der unter einer Bedingung eine Anweisung oder einen Anweisungsblock** ausführt. Um die Bedingung zu formulieren brauchen wir **Operatoren**, mit denen wir **Werte vergleichen** können.




## Verzweigung - IF (Wenn-Dann)

Folgende Sätze wären Bedingungen, die man

```
Wenn eine Frau ihren Mann aus dem Fenster wirft, liest man das in der
Kronen-Zeitung.
Wenn ein Mann seine Frau aus dem Fenster wirft, liest man das in Schöner
Wohnen.
```

Zitat unbekannter Herkunft.

Mit dem Befehl **»if«** ist es möglich, eine **Befehlssequenz unter einer Bedingung** auszuführen. Das Schlüsselwort if kommt aus dem Englischen und bedeutet **»falls«** oder **»wenn«**. Der Syntax einer einfach if-Verzweigung sieht folgendermaßen aus:

```
if( «Bedingung» ){
    «Anweisungen»;
}
```

Dem **Schlüsselwort if** folgt eine Klammer, in der die Bedingung formuliert wird, unter der der nachfolgende **Anweisungsblock** ausgeführt wird. Ein **Anweisungsblock** ist entweder **eine einzelne Anweisung oder mehrere Anweisungen**, die durch **geschweifte Klammern zusammengefasst** werden.

```
int Gemeindeeinwohner=12000;

if (Gemeindeeinwohner > 10000)
{
    cout << "Diese Gemeinde ist eine Gemeinde mit Stadtrecht!" << endl;
}
```

#### Einrücken

Es hat sich bewährt, **Anweisungen**, die unter Bedingungen oder in Schleifen ausgeführt werden, **einzurücken**. Einige Programmeditoren führen dies automatisch durch. Für das Einrücken können die **Tabulatortaste oder Leerzeichen** verwendet werden.

```
if (Divisor != 0)
{
    Ergebnis = Dividend / Divisor;
}
```

Die Division wird nur durchgeführt, wenn die Bedingung zutrifft, also der Divisor ungleich 0 ist.

#### Verschachtelte IF-Verzweigungen

Die Anweisung, die hinter der if-Bedingung steht, kann selbst wieder eine if-Anweisung sein. Sie wird nur ausgeführt, wenn die erste if-Bedingung eintritt. Eine solche Folge von Abfragen nennt man Verschachtelung.

```
int a=0, b=0;
int c=2;
if (a==0)
    if (b==0)
        c=0;
cout << c << endl;
```

Die Abfrage, ob die Variable b 0 enthält, wird nur gestellt, wenn die Variable a den Inhalt 0 hat. Nur wenn beide Variablen 0 sind, wird die Variable c auf 0 gesetzt. Ansonsten enthält sie weiterhin 2.

## Andernfalls: ELSE (Wenn-Dann-Sonst)

Kehren wir zum Divisionsbeispiel zurück. Es wäre schön, wenn das Programm nicht nur prüft, ob der Divisor ungleich 0 ist, sondern den Anwender darüber informieren würde, dass es die Division gar nicht durchführt. Diese Meldung sollte genau dann erscheinen, wenn die if-Bedingung nicht zutrifft. Das könnten Sie erreichen, indem Sie zwei gegensätzliche if-Konstruktionen hintereinander setzen:

```
if (Divisor != 0)
    Ergebnis = Dividend / Divisor;
if (Divisor == 0)
    cout << "Divisor ist 0! Keine Berechnung!";
```

Da solche Konstruktionen immer wieder gebraucht werden, bietet C++ einen eigenen Befehl an, um den gegenteiligen Fall der Bedingung abzudecken. Das Schlüsselwort dazu heißt **else**.

```
if (Divisor != 0)
    Ergebnis = Dividend / Divisor;
else
    cout << "Divisor ist 0! Keine Berechnung!";
```

## Mehrfachverzweigung: switch case

Die Abfrage mit if ermöglicht die Unterscheidung zweier Fälle. Wenn auf Grund verschiedener Werte unterschiedliche Anweisungen ausgeführt werden sollen, können **mehrere verschachtelte if-Anweisungen** hintereinander gesetzt werden oder es kann eine **spezielle Fallunterscheidung** verwendet werden. Ein schönes Beispiel ist ein Fahrstuhl in einem Kaufhaus, in dem eine Vielzahl von Stockwerken zur Auswahl stehen und in jedem Stockwerk andere Waren angeboten werden. Zunächst wird die Aufgabe durch **kaskadierende if-Anweisungen** gelöst:

```
if (Stockwerk == 1)
{
     cout << "Süssigkeiten, Bücher" << endl;
}
else if (Stockwerk == 2)
{
    cout << "Bekleidung" << endl;
}
else if (Stockwerk == 3)
{
    cout << "Bekleidung" << endl;
}
else if (Stockwerk == 4)
{
    cout << "Spielzeug" << endl;
}
```

```
else if (Stockwerk == 5)
{
    cout << "Unterhaltungselektronik" << endl;
}
else
{
    cout << "Garage" << endl;
}
```

Diese Konstruktion lässt sich durch die Mehrfachverzweigung vereinfachen.

```
int Stockwerk;

cin >> Stockwerk;

switch (Stockwerk)
{
    case 1:
        cout << "Süssigkeiten, Bücher" << endl;
        break;
    case 2:
        cout << "Bekleidung" << endl;
        break;
    case 3:
        cout << "Bekleidung" << endl;
        break;
    case 4:
        cout << "Spielzeug" << endl;
        break;
    case 5:
        cout << "Unterhaltungselektronik" << endl;
        break;
    default:
        cout << "Garage" << endl;
        break;
}
```

Die Fallunterscheidung beginnt mit dem **Schlüsselwort switch**. In der darauf folgenden Klammer steht der **ganzzahlige Ausdruck**, dessen **Ergebnis die Verzweigung steuert**. Ein Ausdruck kann eine Variable sein, wie im Beispiel die Variable Stockwerk. Hier könnte aber auch eine Berechnung stehen, die zu einem ganzzahligen Ergebnis führt. Es können **auch Buchstaben** verwendet werden, da Buchstaben aus Sicht von C++ letztlich nichts anderes als getarnte Zahlen sind. Im Beispiel wird die **Variable Stockwerk ausgewertet**, die offensichtlich eine ganze Zahl aufnehmen kann.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_06:5_08_06_01**

Last update: **2024/01/22 08:49**

# Schleifen

Es gibt viele Aufgaben, in denen Befehle wiederholt werden sollen, bis ein bestimmtes Ergebnis erzielt worden ist. Eine solche Wiederholung nennt man in Programmen eine Schleife. Jede Form des Zählens wird im Programm in einer Schleife realisiert. In Computerspielen dürfen Sie so lange ballern, bis Ihr letztes Raumschiff abgeschossen wurde. Eine Schleife hat den Wiederholungscharakter, enthält aber auch immer eine Bedingung, unter der die Schleife verlassen wird, oder positiv formuliert: Die Schleife wird so lange durchlaufen, wie die Schleifenbedingung gilt.

## Kopfgesteuerte Schleife : for-loop

Die beiden bisher gezeigten Schleifen benötigten drei Elemente:

- Eine Initialisierung vor Beginn der Schleife
- Eine Bedingung, unter der die Schleife durchlaufen wird
- Eine Aktion, die die Bedingung verändern kann

Fehlt eines dieser Elemente oder ist es nicht korrekt, dann ist das Risiko einer Endlosschleife groß. Die for-Schleife enthält alle drei Elemente in ihrem Kopf. So sind sie an einer Stelle zusammengefasst und können leichter unter Kontrolle gehalten werden.

Eine sehr typische Anwendung für Schleifen ist das Zählen; und gerade hier zeigt sich besonders die Übersichtlichkeit der for-Schleife. Das folgende Beispiel zählt von 0 bis 9.

```
for (i=0; i<10; i++)
{
    cout << i << endl;
}
```

### Zusammengefasst

In der Klammer, die dem Schlüsselwort for folgt, findet sich als Erstes eine Startanweisung, die die Variable i auf 0 setzt. Bei den while-Schleifen wurde diese Initialisierung vor der Schleife durchgeführt. Als Zweites ist die Schleifenbedingung zu erkennen. Die dritte Anweisung ist das Inkrementieren der Zählervariablen, die üblicherweise am Ende der Schleife steht. Diese drei Elemente werden je durch ein Semikolon getrennt.

Die for-Schleife ist der while-Schleife sehr ähnlich. Sie verpackt ihre Fähigkeiten nur eleganter und bewirkt, dass der Programmierer nicht so leicht wichtige Elemente vergisst. Sie könnten das obige Schema leicht auch als while-Schleife darstellen.

```
Startanweisung;
while ( Bedingung)
{
    Schleifenkörper;
    Schlussanweisung;
}
```

Entsprechend unterscheidet sich das Struktogramm nicht von dem der while-Schleife. Es wird lediglich inhaltlich der Kopf der Schleife in den Abfragebereich geschrieben.

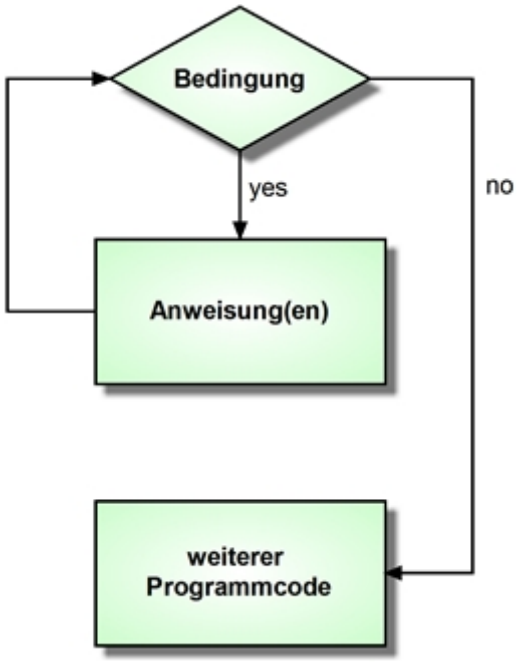
## Kopfgesteuerte Schleife : while-loop

### Solange

Die **einfachste Schleife** wird durch das **Schlüsselwort while** eingeleitet. Die deutsche Übersetzung lautet **»solange«**.

Sie **wiederholt Vorgänge, solange eine bestimmte Bedingung erfüllt ist**.

Auch im Alltag verwenden wir solche Schleifen. Solange die Suppe noch nicht salzig genug ist, schüttet man noch einen Teelöffel Salz hinein. Und wie in diesem Beispiel zuerst der Salzgehalt geprüft wird, so prüft auch die while-Schleife zuerst die Bedingung und führt dann erst die Anweisung aus.




### Analogie zu if

Der Syntax ähnelt sehr einer if-Anweisung. Und tatsächlich bestehen die Unterschiede nur darin, dass die **Schleife die Anweisung wiederholt**. Ist die **Bedingung von vornherein nicht erfüllt**, verhält sich die Schleife exakt wie die if-Anweisung - **Sie führt die Anweisung nicht aus**.

Die Schleife beginnt mit dem **Schlüsselwort while**. In Klammern folgt die Bedingung, unter der die direkt anschließende Anweisung ausgeführt wird. Auch hier können mehrere Anweisungen zu einem Anweisungsblock zusammengefasst werden, wenn sie in geschweiften Klammern stehen.

### Zählen für Anfänger

Als erstes Beispiel bringen wir dem Computer das Zählen bei. Denken Sie darüber nach, was Sie tun, wenn Sie zählen. Zunächst beginnen Sie bei einer bestimmten Zahl, typischerweise bei eins. Dann folgt die Wiederholung, in der Sie immer den bisherigen Wert um eins erhöhen. Da Sie auch irgendwann einmal etwas anderes tun wollen als zählen, werden Sie bei einem bestimmten Wert die Wiederholung abbrechen.

### Zählen für Computer

Im Programm verwenden Sie eine Variable, die Sie mit dem Startwert vorbesetzen. Dieser Startwert muss vor der Schleife gesetzt werden, sonst würde die Variable in jeder Runde auf ihren Ausgangspunkt zurück gesetzt, und die Schleife würde ewig rotieren. Danach soll das Programm in die Schleife eintreten, in der die Variable immer um eins erhöht wird. Die Schleife beginnt mit dem Schlüsselwort while. Direkt dahinter wird in Klammern die Bedingung formuliert. Solange diese Bedingung gilt, bleibt das Programm innerhalb der Schleife. Wenn Sie bis zehn zählen wollen, müssen Sie hier prüfen, ob die Variable kleiner oder gleich 10 ist. Direkt anschließend an die Bedingung können Sie eine Anweisung angeben, die in der Schleife wiederholt werden soll. Da Sie ja einerseits die Variable um eins erhöhen wollen und andererseits den aktuellen Stand auf dem Bildschirm ausgeben wollen, brauchen Sie aber zwei Anweisungen, die in der Schleife ausgeführt werden sollen. Das kann aber leicht mit einem Block, also geschweiften Klammern gelöst werden. In den Block wird also die Ausgabe der Variablen und die anschließende Erhöhung der Variablen gesetzt.

```
#include <iostream>
using namespace std;
int main()
{
    i = 1;
    while (i <= 10) // Schleifenbedingung
    {
        cout << i << endl;  // Aktion
        i++; // Ohne diese Erhöhung wird es eine Endlosschleife
    }

    return 0;
}
```

### Grenzfragen

Leicht entstehen Fehler bei der Festlegung der Schleifengrenzen. Würden Sie statt ⇐ nur das Kleinerzeichen verwenden, würde das Programm nur bis neun zählen. Der Grund ist, dass sobald der Inhalt von i den Wert 10 erreicht, die Bedingung nicht mehr gilt und damit das Schleifeninnere nicht mehr betreten wird. Sie können auch eine Schleife bilden, die zehnmal durchläuft, indem Sie bei 0 beginnen und abfragen, ob die Variable kleiner als zehn ist. Dann würde die Variable i von 0 bis 9 laufen. Wenn Sie i allerdings vor der Ausgabe erhöhen, erhalten Sie wieder die Zahlen von 1 bis 10.

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int i = 0;
    while (i < 10)
    {
        i++;
        cout << i << endl;
    }
    return 0;
}
```

#### Test am Kopf

Die **while-Schleife** prüft die Bedingung am Schleifenkopf. Das bedeutet, dass die Anweisungen in der Schleife gar nicht ausgeführt werden, wenn die Bedingung vor dem Betreten der Schleife unwahr ist.

#### Endlosschleife

Achten Sie besonders auf die Formulierung der Schleifenbedingung. Nicht nur Anfängern passiert es, dass die Schleife niemals endet, weil die Schleifenbedingung niemals unwahr wird. Eine solche Schleife wird Endlosschleife genannt. Eine andere Ursache für eine Endlosschleife kann darin bestehen, dass die Bedingung zwar richtig formuliert ist, aber dass der Programmierer vergisst, die Daten, die in der Bedingung abgefragt werden, innerhalb der Schleife zu ändern.

```
while (true)
{

}
```

### Fußgesteuerte Schleife : do....while-loop

In manchen Situationen ist die Prüfung am Kopf der Schleife ungünstig. Manchmal wollen Sie eine Aktion durchführen und werden erst an ihrem Ende erkennen können, ob die Aktion wiederholt werden soll. Ein Kind wird beispielsweise seinen Opa so lange um Süßigkeiten anbetteln, bis dieser keine mehr hat. (Erfahrene Eltern werden zu Recht einwerfen, dass die Abschlussbedingung von dem System Opa nicht eingehalten wird.) Das Kind wird in jedem Fall erst einmal betteln, ansonsten kann es die Bedingung gar nicht erst prüfen. In diesem Fall findet also die Prüfung erst nach dem Durchlauf des Schleifenkörpers statt.

#### Eingabeprüfung

In Programmen wird eine fußgesteuerte Schleife besonders bei Eingabeprüfungen eine typische Anwendung. Innerhalb der Schleife fordert das Programm vom Anwender eine Eingabe an. Erst dann wird geprüft, ob die Eingabe korrekt ist. Ist die Eingabe nicht zufriedenstellend, wird die Schleife wiederholt. Ein weiteres Beispiel ist das Spiel Minesweeper. Der Benutzer soll auf ein Feld klicken. Solange unter dem Feld keine Mine liegt, darf er weiterspielen. Sie können erst am Ende der Schleife

feststellen, wohin der Benutzer geklickt hat.

Auch hier werden Sie in den meisten Fällen einen Block aus geschweiften Klammern verwenden, da eine Schleife meist mehrere Anweisungen umfasst.

Im folgenden Beispiel wird rückwärts gezählt. Betrachten Sie das Listing, und überlegen Sie, ob die Zahlen 10, 1 und 0 ausgegeben werden! Prüfen Sie, ob Ihre Vermutungen stimmen!

```
#include <iostream>
using namespace std;

int main()
{
    int i = 10;
    do
    {
        cout << i << endl;
        i--;
    }
    while(i>0);

    return 0;
}
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_06:5_08_06_02**

Last update: **2023/11/15 17:11**

# Übungen

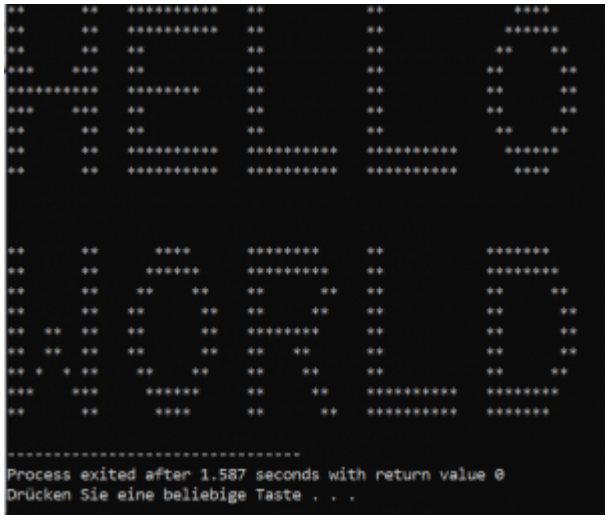
## 01 - Hello World

Schreibe ein Programm, das den Text „Hello World“ am Bildschirm ausgibt.

```
Hello World!
--------------------------------
Process exited after 0.01695 seconds with return value 0
Drücken Sie eine beliebige Taste . . .
```

## 02 - Hello World grafisch

Schreibe ein Programm, das mithilfe von * folgende Ausgabe erzeugt.




## 03 - Summe

Entwickle ein Programm, welches zwei Zahlen vom Benutzer einliest und anschließend addiert. Das Ergebnis wird anschließend ausgegeben.

```
Geben Sie die 1. Zahl ein:10
Geben Sie die 2. Zahl ein:20
10+20=30
--------------------------------
Process exited after 2.44 seconds with return value 0
Drücken Sie eine beliebige Taste . . .
```

## 04 - Produktbestellung

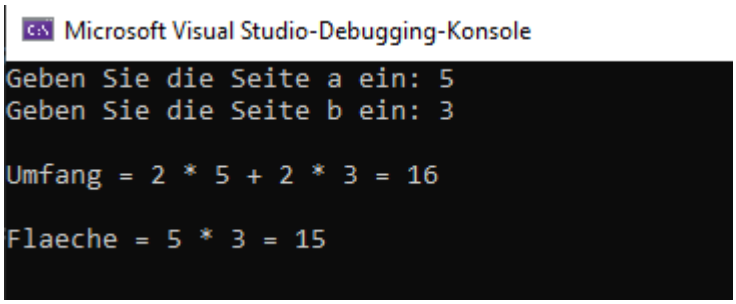
Entwickle ein Programm, welches vom Benutzer zwei Zahlen (Anzahl & Preis) einliest. Berechne

anschließend den Gesamtpreis (Anzahl * Preis) und gib diesen aus. Z.B:
Anzahl: 5
Preis (€): 1.5

Gesamtpreis: 7.5 €

## 05 - Rechteck

Entwickle ein Programm, welches 2 Zahlen (Seite a & Seite b) für die Berechnung der Fläche und des Umfangs eines Rechteckes vom Benutzer einliest und anschließend die Ergebnisse ausgibt.




## 06 - Kreisumfang und -flächeninhalt

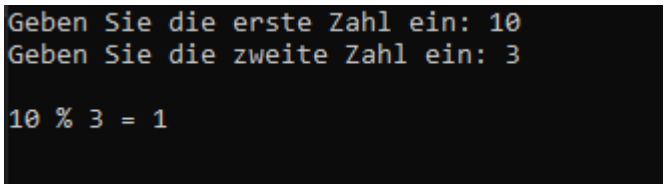
Nach Eingabe von Radius wird der Umfang und Flächeninhalt eines Kreises ausgegeben.

## 07 - Division

Entwickle ein Programm, dass nach Eingabe von Dividend und Divisor den Quotient berechnet und ausgibt. Auch Gleitkommazahlen sollen möglich sein!

## 08 - Restberechnung

Entwickle ein Programm, dass nach Eingabe von ganzzahligen Dividend und Divisor den Rest einer Division (%) berechnet.




## 09 - Fahrenheit

Ein Fahrenheit-Wert wird eingegeben und in Grad Celsius umgerechnet. Weiters wird ausgegeben, ob Glatteisgefahr besteht oder nicht.

```
Geben Sie die Temperatur als Fahrenheit-Wert ein:25

Grad Celsius = (25 - 32 ) * 5 / 9 = -3.88889
Achtung!! Es herrscht Glatteisgefahr!
```

## 10 - Geschwindigkeit

Je nach Höhe der Geschwindigkeit gibt es entweder keine Strafe oder gestaffelte Strafzahlungen. Die maximal zulässige Höchstgeschwindigkeit war 50 km/h.

> 55 bis 60 km/h: 35€
>
> 60 bis 65 km/h: 55€
>
> 65 bis 70 km/h: 80€
>
> 70 bis 75 km/h: 100€
>
> 75 bis 80 km/h: 150€
>
> 80 km/h: Führerscheinentzug

## 11 - Wochentag

Das Programm wandelt die Zahlen 0 bis 6 in den entsprechenden Wochentag um.

z.B.:
Wochentag: 1

Montag

## 12 - Noten

Das Programm gibt die eingegebene Schulnote verbal aus.
z.B.:
Note: 1

Sehr gut

## 13 - Maximum finden

Entwickle ein Programm, welches drei unterschiedliche große Zahlen einliest und anschließend die größte Zahl herausfindet und ausgibt.
z.B:
1. Zahl: 3

2. Zahl: 5
3. Zahl: 4

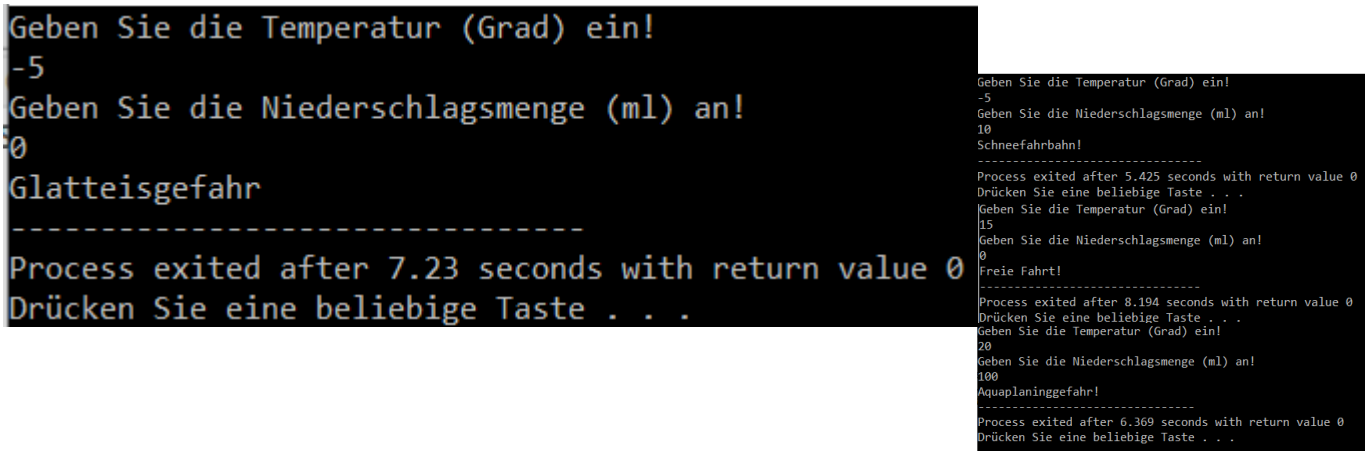
Maximum: 5

## 14 - Temperatur

Mit diesem Programm wird festgestellt, welche Umstände bei entsprechenden Temperaturen und Niederschlag zu beachten sind. Die Temperatur soll eingegeben werden. Weiters wird eingegeben, ob es Niederschlag (0 ⇒ kein Niederschlag, >0 ⇒ Niederschlag) gibt.

Das Programm gibt je nach Eingaben folgende Werte zurück:

- Glatteisgefahr, Schneefahrbahn, Freie Fahrt, Aquaplaninggefahr.




## 15 - Body Mass Index

Im Programm wird eingegeben, ob man eine Frau oder ein Mann ist. Weiters gibt man die Körpergröße in cm sowie das Gewicht in kg an. Mittels folgender Formel wird der Body Mass Index berechnet und ausgegeben:

`bmi=(kg)/m^2`

Weiters soll nach folgender Tabelle ermittelt werden, ob Unter-, Normal- oder Übergewicht vorliegt.

| | Frauen | Männer |
|---|---|---|
| Untergewicht | <19.1 | <20.7 |
| Normalgewicht | 19.1-25.8 | 20.7-26.4 |
| Übergewicht | >25.8 | >26.4 |

## 16 - Dreieck

Die drei Seitenlängen a, b und c eines Dreiecks eingegeben. Es wird ausgegeben, ob es sich um ein Dreieck handelt. Weiters wird ermittelt, ob ein gleichschenkeliges, gleichseitiges, rechtwinkeliges oder allgemeines Dreieck vorliegt.

## 17 - Textbasierter Rechner

Der Benutzer kann zwei Zahlen und einen Operator eingeben.

Zumindest die vier folgende Operatoren + (addieren), - (subtrahieren), * (multiplizieren) und / (dividieren) sollen möglich. Das Ergebnis wird anschließend ausgegeben.
Wird eine andere Zahl eingegeben, bricht das Programm ab.
z.B.:
1. Zahl: 3
Operator: -
2. Zahl: 2

Ergebnis: 1

## 18 - Uhrzeit

Das Programm wandelt die digitale Uhrzeit (z.B. 12:45) in die gesprochene Uhrzeit (Dreiviertel eins) um!

## 19 - Gaußsche Wochentagsformel

Das Programm berechnet nach eingegebenem Datum den entsprechenden Wochentag.

Formel siehe: Gaußsche Wochentagsformel

## 20 - Zahlensubtraktion

Entwickle ein Programm welches zwei Zahlen vom Benutzer einliest und anschließend die kleinere Zahl von der größeren Zahl abzieht solange die Zahl größer 0 ist.

```
1. Zahl:4
2. Zahl:18


18-4=14
14-4=10
10-4=6
6-4=2
2-4=-2
--------------------------------
Process exited after 3.737 seconds with return value 0
Drücken Sie eine beliebige Taste . . .
```

```
1. Zahl:9
2. Zahl:5


9-5=4
4-5=-1
--------------------------------
Process exited after 10.71 seconds with return value 0
Drücken Sie eine beliebige Taste . . .
```

```
1. Zahl:8
2. Zahl:2


8-2=6
6-2=4
4-2=2
2-2=0
--------------------------------
Process exited after 5.298 seconds with return value 0
Drücken Sie eine beliebige Taste . . .
```

## 21 - Euroscheinüberprüfung

**Schleifen und char nötig!!!**

Jeder Euroschein hat eine Nummer, zum Beispiel N27175276569.

Der erste Buchstabe (hier N) ist ein Ländercode für das Herstellungsland, z.B. N für Österreich.

Darauf folgt eine zehnstellige, fortlaufende Seriennummer (hier: 2717527656). Dann folgt am Ende eine einstellige Prüfziffer, im Beispiel ist dies die 9.

Um zu prüfen, ob die Seriennummer bzw. der Schein echt ist, ist folgendes Verfahren anzuwenden:

1. Buchstaben durch die Position des Buchstaben im Alphabet ersetzen: A=1, B=2, ..., Y=25, Z=26.
2. Die Summe aus der Buchstabenzahl, den zehn Ziffern der Seriennummer und der Prüfziffer bilden.
3. Von dieser Summe die Ziffernsumme (und wieder die Ziffernsumme ...) bilden, bis die Zahl einstellig wird.

Beispiel: 14+2+7+1+7+5+2+7+6+5+6+9=71

7+1=8

Das Ergebnis muss 8 sein, damit die Seriennummer echt ist!

Welcher Buchstabe für welches Land steht, findest du z.B. auf der Seite der Europäischen Zentralbank: http://www.ecb.int/euro/banknotes/html/index.de.html

## 22 - Umwandlung Dualzahl in Dezimalzahl

Schreibe ein Programm, welches eine Dualzahl in eine Dezimalzahl umwandelt.
Bei der Eingabe dürfen nur die Ziffern 0 und 1 oder e eingegeben werden. Sobald e eingegeben wird, ist die Zahl zu Ende.
Die entsprechende Dezimalzahl wird ausgegeben.

## 23 - Umwandlung Hexadezimalzahl in Dezimalzahl

Schreibe ein Programm, welches eine Hexadezimalzahl in eine Dezimalzahl umwandelt.

Bei der Eingabe dürfen nur die Ziffern 0 bis 9 bzw. A bis F oder e eingegeben werden. Sobald e eingegeben wird, ist die Zahl zu Ende. Die entsprechende Dezimalzahl wird ausgegeben.

## 24 - "Tempo, kleine Fische"

Simuliere das Spiel „Tempo, kleine Fische“, indem du den Spielverlauf in geeigneter Form in ein Programm umsetzt.

Spielanleitung

a) Via Zufallswürfel soll der Verlauf einer Spielrunde simuliert werden.
b) Alternativ dazu soll der Benutzer bestimmen können, wie oft das Spiel durchgeführt wird (z.B. 100 mal). Danach gibt das Programm aus, wie oft die verschiedenen Möglichkeiten des Spielendes (Alle Fische erreichen das Meer, etc.) aufgetreten sind.

## 25 - Malreihen

Schreibe ein Programm, dass alle 1*1-Reihen (1-10) auf einmal ausgegeben werden.

*Hinweise: Schleifen können auch ineinander geschachtelt werden!*

Ausgabebeispiel:

```
Das folgende Programm gibt alle Einmaleins-Reihen aus:

1er-Reihe:
1*1=1
2*1=2
...

2er-Reihe:
1*2=2
2*2=4
...

Wollen Sie das Programm nochmals ausfuehren (j/n)?
```

## 26 - Quadratzahlen

Schreibe ein Programm, das eine Quadratzahltabelle von 1 bis zu einem eingegebenen Wert *n* ausgibt. (Quadratzahl.cpp)

```
Eingabe n=9

1 -> 1
2 -> 4
3 -> 9
```

```
4 -> 16
5 -> 25
6 -> 36
7 -> 49
8 -> 64
9 -> 81

Wollen Sie das Programm nochmals ausfuehren (j/n)?
```

## 27 - Fibonacci

Schreibe ein Programm, das die ersten *n* Fibonacci-Zahlen erzeugt und ausgibt! Eine Fibonacci-Zahl berechnet sich aus der Summe der beiden vorhergehenden Zahlen. Die ersten beiden Fibonacci-Zahlen sind gleich 1! (Fibonacci.cpp)

z.B.: 1, 1, 2, 3, 5, 8, 13, 21,...

## 28 - Dezimal-Dual

Schreibe ein Programm, das die eingegebene Dezimalzahl in eine Dualzahl umwandelt.
Auf welches Problem stößt du dabei? (DeziDual.cpp)

## 29 - Teiler

a) Schreibe ein Programm, das die Teiler einer eingegebenen natürlichen Zahl n auflistet. (Teilerliste.cpp)
b) Schreibe ein Programm, das die Anzahl der Teiler eingegebenen natürlichen Zahl n ausgibt. (Teileranzahl.cpp)

## 30 - Staffelrechnung/Rechenturm

Erstelle ein Programm, das eine Staffelrechnung / einen Rechenturm durchführt. Der Benutzer gibt ein, mit welcher Zahl er die Staffelrechnung beginnen will und bis zu welcher Multiplikation die Rechnungen durchgeführt werden:

Z.B.
Anfangszahl: 385
Multiplikationen bis: 6

Ausgabe:

```
385*2=770
770*3=2310
2310*4=9240
9240*5=46200
46200*6=277200
```

```
277200/2=138600
138600/3=46200
46200/4=11550
11550/5=2310
2310/6=385
```

## 31 - verschachtelte Schleifen

### Rechteck

Schreibe ein Programm, das mit Sternchen ein Rechteck ausgibt! Das Programm soll Breite, Höhe und einen Versatz einlesen. Der Versatz bestimmt, um viele Zeichenbreiten das gesamte Rechteck nach rechts verschoben wird.

```
Ein Rechteck

    Breite?
    17

    Hoehe?
    4

    Versatz?
    2

      ******************
      ******************
      ******************
      ******************
```

### Parallelogramm

Schreibe ein Programm, das mit Sternchen ein Parallelogramm ausgibt! Das Programm soll die Breite, Höhe und die Versatzänderung (die Verschiebung nach rechts pro Zeile) einlesen.

```
Ein Parallelogramm

    Breite?
    11

    Hoehe?
    5

    Versatzaenderung?
    1

    ***********
```

```
      **********
       **********
        **********
         **********
```

### Dreieck

Schreibe ein Programm, das mit Sternchen ein Dreieck ausgibt! Das Programm soll die Höhe und die Verbreiterung pro Zeile einlesen.

```
Ein Dreieck

    Hoehe?
    6

    Verbreiterung?
    1

    *
    **
    ***
    ****
    *****
    ******
```

## 32 - Regelmäßige Zahlungen

Schreibe ein Programm, das den Betrag eines regelmäßigen jährlichen Einzahlungsbetrages, die Verzinsung und den gewünschten Endbetrag einliest und ausgibt, nach wie vielen Jahren der gewünschte Endbetrag erreicht (oder erstmals überschritten) ist.

Am Anfang des ersten Jahres wird zunächst ein Einzahlungsbetrages, z.B. 1000 Euro, eingezahlt. Am Ende des ersten Jahres (nach einem Jahr) kommt der Zins, bei 5 % z.B. 50 Euro, und ein weiterer Einzahlungsbetrag, wieder 1000 Euro, hinzu. Nach einem Jahr ist der Kontostand dann also beispielsweise 1000 + 50 + 1000 Euro = 2050 Euro. Sollten also beispielsweise 2000 Euro angespart werden, so wäre dies unter diesen Umständen nach einem Jahr erreicht.

Man kann also in einer Schleife die Jahre mitzählen und den Kontostand fortschreiben. Dann kann man jedesmal prüfen, ob der gewünschte Endbetrag schon erreicht oder überschritten wurde und gegebenenfalls die Zahl der abgelaufenen Jahre und den Kontostand ausgeben. In der Endfassung darf das Programm aber nichts anderes ausgeben als das Endergebnis (keine Zwischenstände).

Das Programm muß auch dann eine richtige (sinnvolle) Antwort geben, wenn der gewünschte Endbetrag kleiner oder gleich dem Einzahlungsbetrag ist.

Die Lösung der Übungsaufgabe sollte mindestens mit den beiden folgenden Testsituationen geprüft werden und muß dann identische Ergebisse liefern.

```
Beispiel 1

    jaehrlicher Zahlbetrag?
    1000

    Verzinsung, in Prozent?
    5

    gewuenschter Endbetrag?
    2000

    Es wird 1 Jahr benoetigt.
    Der erreichte Endbetrag ist dann 2050.


Beispiel 2

    jaehrlicher Zahlbetrag?
    100

    Verzinsung, in Prozent?
    5

    gewuenschter Endbetrag?
    10000

    Es werden 36 Jahre benoetigt.
    Der erreichte Endbetrag ist dann 10162.8.
```

## 33 - Statistische Kenndaten

Schreibe Sie ein Programm, das in einer Schleife solange natürliche Zahlen einliest, bis -1 eingegeben wird und dann Maximum, Minimum und das arithmetische Mittel der Zahlen ausgibt. Da die Zahl -1 nur als Stoppwert verwendet wird, soll sie bei der Statistik nicht berücksichtigt werden.

Hinweis: Wenn die Variablen summe und n beide vom Typ integer sind, dann wird bei einer Division das Ergebnis (arithmetisches Mittel am) auch ganzzahlig berechnet. Um dies zu verhindern konvertiert man den Zähler mittels `(float)`.

```
am=summe/(float)n;
```

## 34 - Harshad-Zahlen

Eine natürliche Zahl, die durch ihre eigene Ziffernsumme teilbar ist, heißt Harshad- oder Niven-Zahl. Zum Beispiel:
777 ist durch 7 + 7 + 7 = 21 teilbar und damit eine Harshad-Zahl.

Schreibe ein Programm, das alle Zahlen bis n testet, ob sie Harshad-Zahlen sind. Das Programm soll

alle Harshad-Zahlen ausgeben.

## 35 - Werfen von 2 Würfeln

Stochastische Feststellung: Kommt die 7 wirklich am häufigsten vor?

Wenn man mit 2 Würfeln würfelt, so tritt die 7 am häufigsten auf.

Wie oft tritt welche Zahl auf, wenn man n mal würfelt?

```
Werfen von 2 Würfeln...
Gib an, wie oft gewürfelt werden soll: 100
Gib an, welche Summe gezählt werden soll: 6

Insgesamt kam die Augensumme 6 genau 12 mal vor.
```

## 36 - Prüfen der Sozialversicherungsnummer

Schreibe ein Programm, welche die Gültigkeit einer Sozialversicherungsnummer prüft.
Der Benutzer gibt die Sozialversicherungsnummer ein, das Programm gibt aus, ob es sich um eine gültige Sozialversicherungsnummer handelt.

Informationen zur Sozialversicherungsnummer:

http://www.pruefziffernberechnung.de/V/VSNR-AT.shtml

## 37 - Zählen

Schreibe ein Programm, das in einer Schleife eine Variable von 0 beginnend solange hochzählt, bis der Benutzer ein 'X' oder 'x' eingibt. Gib anschließend den Wert der Variable über den Standard-Ausgabekanal (stdout) aus.

Erweiterung: Der Benutzer gibt zuerst eine Zahl ein, die er anschließend versucht zu erreichen, in dem er das Programm rechtzeitig stoppt.

## 38 - Buchstabieren

Schreibe ein Programm, das in einer Schleife eine Variable von 'a' beginnend solange hochzählt, bis der Benutzer ein 'X' oder 'x' eingibt. Implementiere das Buchstabieren bzw. Hochzählen der Variable so, dass nach einem 'z' wieder von vorne mit einem 'a' begonnen wird. Gib anschließend den Wert der Variable über den Standard-Ausgabekanal (stdout) aus.

Erweiterung: Der Benutzer gibt zuerst einen Buchstaben ein, den er anschließend versucht zu erreichen, in dem er das Programm rechtzeitig stoppt.

## 39 - Zahlen raten

Schreiben Sie mithilfe einer Schleife ein kleines Spiel. Der Spielablauf ist wie folgt:

- Am Anfang gibt der Spieler einen Zahlenbereich ein. (Zum Beispiel: 1-100)
- Der Spieler muss sich innerhalb dieses Bereiches eine Zahl merken (eingegebene Grenzzahlen sind nicht zulässig).
- Das Programm soll dann die Zahl erraten. Der Benutzer teilt dem Programm mit, ob die Zahl an die er denkt kleiner, größer oder gleich der vom Programm geratenen Zahl ist.
- Dies kann zum Beispiel über die Eingabe von <, > und = erfolgen.

## 40 - Getränkeautomat

Stelle dir vor, du hast die Aufgabe eine Software für einen Getränke-Automaten zu schreiben. Der Automat spuckt gegen Geldmünzen Getränkeflaschen aus.

- Der Automat besitzt mehrere Sorten von Getränken.
- Nach der Wahl der Sorte kann der Benutzer auch eine Menge eingeben.
- Der zu zahlende Betrag errechnet sich dann aus dem Produkt von Menge und Preis pro Flasche.
- Die Ausgabe der einzelnen Flaschen erfolgt mit einer Bildschirmausgabe, siehe unten.
- Die Bezahlung soll jetzt auch mit verschiedenen Geldstücken gemacht werden können.
- Der Benutzer wird solange aufgefordert Geld einzuwerfen, bis der zu zahlende Betrag erreicht ist.

```
Waehlen sie ihr Getraenk aus:
1) Wasser (0,50 Euro)
2) Limonade (1,00 Euro)
3) Bier (2,00 Euro)

Geben sie 1, 2 oder 3 ein: 3

Geben sie die gewuenschte Menge ein: 2

--- Bezahlvorgang ---

Es fehlen noch 4.00 Euro.
Bitte werfen sie ein Geldstueck ein: 2

Es fehlen noch 2.00 Euro.
Bitte werfen sie ein Geldstueck ein: 1

Es fehlen noch 1.00 Euro.
Bitte werfen sie ein Geldstueck ein: 0.5

Es fehlen noch 0.50 Euro.
Bitte werfen sie ein Geldstueck ein: 0.5

--- Getraenkeausgabe ---

Flasche 1 von 2 wurde ausgegeben.
Flasche 2 von 2 wurde ausgegeben.

Vielen Dank, bitte entnehmen sie ihre Getraenke.
```

## 41 - Sternfunktion

Schreibe ein Programm, welches folgende Ausgabe liefert:

```
**********
Einleitung:
**********
Hauptteil:
**********
Schluss:
**********
```

Die Sterne sollen jeweils mit Hilfe eines Unterprogramms/einer Funktion ausgegeben werden.

## 42 - Sternfunktion erweitert

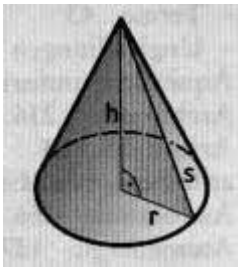
Modifiziere das bereits erstellt Programm „Sterne“ so, dass der Benutzer festlegen kann, wie viele Sterne an beliebiger Stelle ausgegeben werden. Verwende dafür eine Funktion mit Parameter.

## 43 - Gerade/Ungerade Zahl

Schreibe ein Programm, welches mit Hilfe einer Funktion überprüft, ob die vom Benutzer eingegebene Zahl gerade ist.

## 44 - Drehkegel mit Funktion und Übergabe per Wert

Schreibe ein Programm (Drehkegel.cpp), das die Oberfläche und das Volumen eines Drehkegels




ausgibt, wenn der Benutzer den Radius und die Höhe eingibt!

Die Berechnung der Oberfläche soll in einer eigenen Funktion erfolgen.

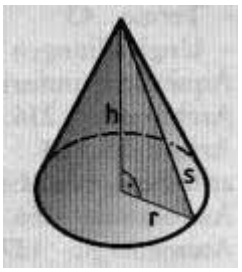
```
void oberflaeche(float r, float h){
....
}
void volumen(float r, float h){
....
}
```

Hinweis:

1. $O = r^2\cdot \pi + r \cdot s \cdot \pi = r \cdot \pi \cdot (r+s)$ wobei $s$ die Längelinie ist $s = \sqrt{r^2+h^2} $
2. Die Quadratwurzel wird mit der in der Mathematikbibliothek ( `#include <math.h>` ) enthaltenen Funktion `sqrt()` berechnet!

## 45 - Drehkegel mit Funktion und Übergabe per Referenz

Schreibe ein Programm (Drehkegel.cpp), das die Oberfläche und das Volumen eines Drehkegels




ausgibt, wenn der Benutzer den Radius und die Höhe eingibt!

Die Berechnung der Oberfläche und des Volumens soll in einer Funktion erfolgen. Wähle die Parameter so, dass man die beiden Ergebnisse anschließend im Hauptprogramm ausgeben kann.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

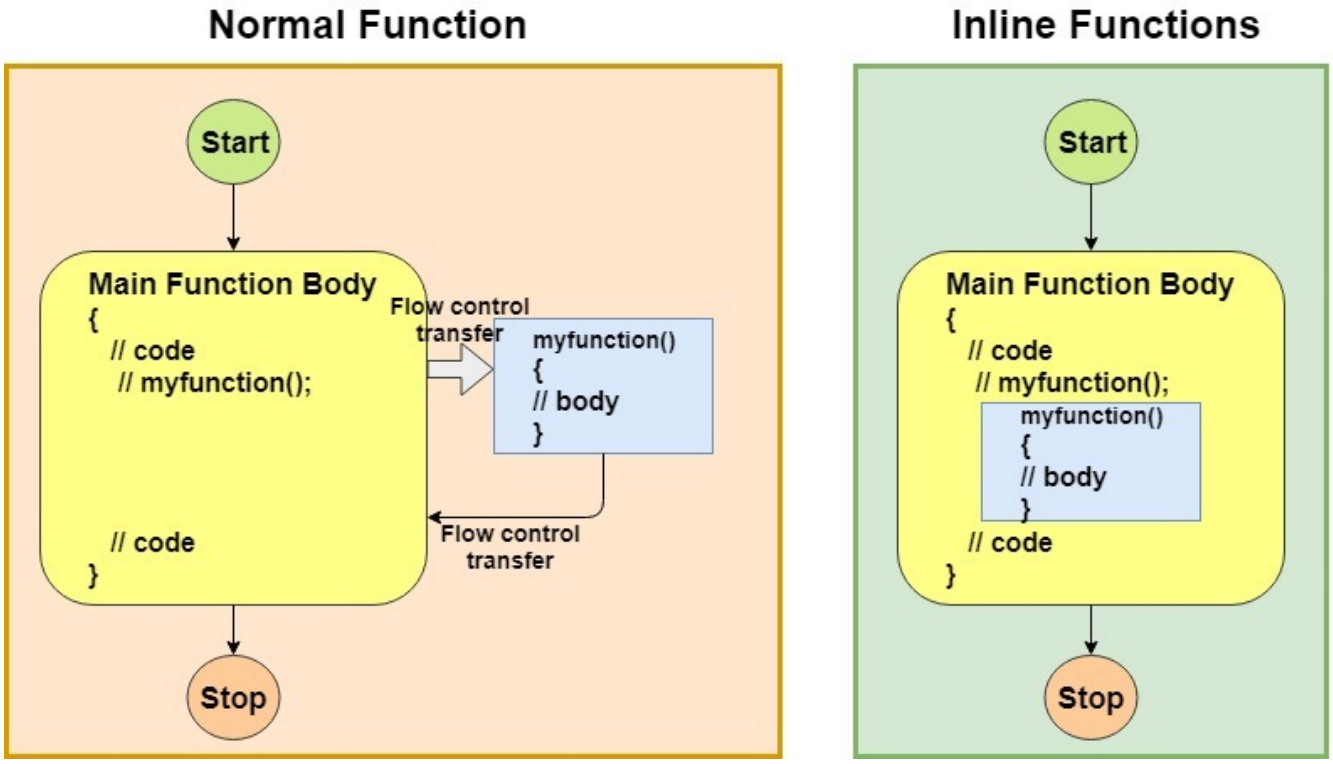
Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_06:5_08_06_03**

Last update: **2024/02/19 08:40**

# Funktionen

Unter einer **Funktion** (function, in anderen Programmiersprachen auch **Prozedur** oder **Subroutine** genannt) versteht man ein **Unterprogramm**, das eine **bestimmte Aufgabe** erfüllt.

Funktionen sind unter anderem sinnvoll, um sich oft **wiederholende Befehle zu kapseln**, so dass diese nicht jedes Mal neu geschrieben werden müssen. Zudem verbessert es die **Übersichtlichkeit**.




## Parameter und Rückgabewert

Die spezielle Funktion main() ist uns schon mehrfach begegnet. In C+ + lassen sich Funktionen nach folgenden Kriterien unterscheiden:

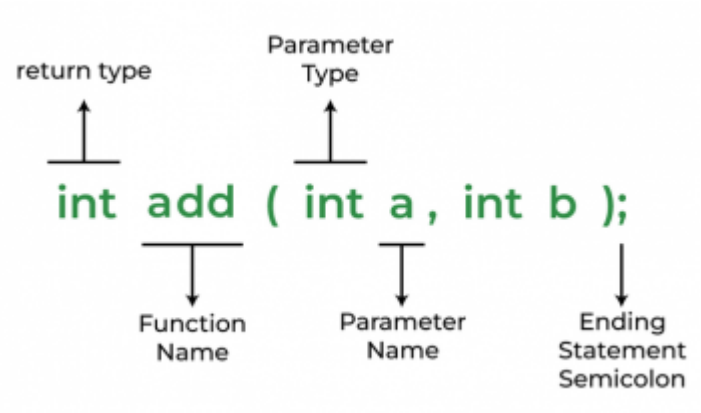
- Eine Funktion kann Parameter besitzen, oder nicht.
- Eine Funktion kann einen Wert zurückgeben, oder nicht.

```
# include <iostream>
using namespace std;

int add(int, int);

int main() {
    ... .. ...
    sum = add(num1, num2);  // Actual parameters: num1 and num2
   ... .. ...
}

int add(int a, int b) {     // Formal parameters: a and b
    ... .. ...
    add = a+b;
   ... .. ...
}
```

Dem Funktionsbegriff der Mathematik entsprechen diejenigen C+ + -Funktionen, die sowohl Parameter haben als auch einen Wert zurückgeben. Dieser Wert kann im Programm weiter genutzt werden, um ihn z.B. einer Variablen zuzuweisen.

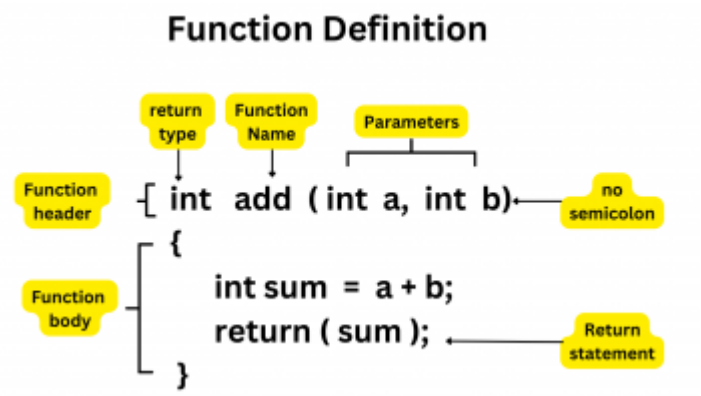
```
int sum = add(3,5); // Aufruf einer Funktion
```

Damit diese Anweisung fehlerfrei kompiliert wird, muss vorher die **Funktion add() deklariert** worden sein. Bei einer Funktion bedeutet **Deklaration** die **Angabe des Funktionsprototyps**. Das heißt, der **Typ von Parametern und Rückgabewert** muss angegeben werden. Das folgende Beispiel deklariert bspw. eine Funktion, die einen **Parameter vom Typ int** besitzt und als **Rückgabewert einen int-Wert** zurückgibt.




```
int add (int a, int b); // Funktionsprototyp == Deklaration
```

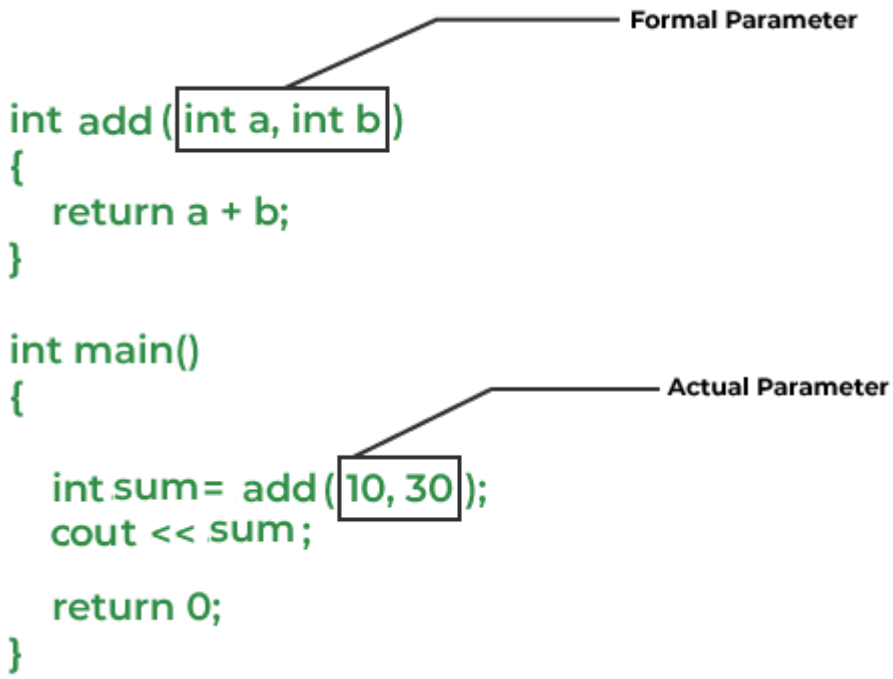
Soll eine Funktion keinen Wert zurückliefern, lautet der **Rückgabetyp** formal **void**.
Nach dem Compilieren ist das Linken der entstanden Objektdateien zu einem ausführbaren Programm nötig. Der Linker benötigt die Definition der aufzurufenden Funktion. Eine **Funktionsdefinition** umfasst auch die **Implementation der Funktion**, d.h. den Code, der beim Aufruf der Funktion ausgeführt werden soll. In unserem Fall wäre das:




```
int add(int a, int b); // Funktionsdeklaration

int main(){

   int sum = add(3, 6); // Funktionsaufruf
      // a hat jetzt den Wert 9

   return 0;
```

```
}

int add(int a, int b){ // Funktionsdefinition
   return a+b;
}
```

Der Compiler muss die Deklaration kennen, um eventuelle Typ-Unverträglichkeiten abzufangen. Würden Sie die obige Funktion z.B. als int a = add(2.5, 3); aufrufen, käme die Warnung, dass add() ein ganzzahliges Argument erwartet und keine Fließkommazahl. Eine Definition ist für den Compiler auch immer eine Deklaration, das heißt Sie müssen nicht explizit eine Deklaration einfügen um eine Funktion aufzurufen, die zuvor definiert wurde.




Die **Trennung von Deklaration und Definition** kann zu **übersichtlicher Code-Strukturierung** bei größeren Projekten genutzt werden. Insbesondere ist es sinnvoll, Deklarationen und Definitionen in verschiedene Dateien zu schreiben. Oft will man, wenn man fremden oder alten Code benutzt, nicht die Details der Implementierung einer Funktion sehen, sondern nur das Format der Parameter o.ä. und kann so in der Deklarationsdatei (header file, üblicherweise mit der Endung .hpp, z.T. auch .h oder .hh) nachsehen, ohne durch die Funktionsrümpfe abgelenkt zu werden. (Bei proprietärem Fremdcode bekommt man die Implementation in der Regel gar nicht zu Gesicht!)

Bei einer Deklaration ist es nicht nötig, die Parameternamen mit anzugeben, denn diese sind für den Aufruf der Funktion nicht relevant. Es ist allerdings üblich die Namen dennoch mit anzugeben um zu verdeutlichen was der Parameter darstellt. Der Compiler ignoriert die Namen in diesem Fall einfach, weshalb es auch möglich ist den Parametern in der Deklaration und der Definition unterschiedliche Namen zu geben. Allerdings wird davon aus Gründen der Übersichtlichkeit abgeraten.

Mit der Anweisung **return** gibt die **Funktion einen Wert zurück**, in unserem Beispiel x * x, wobei die Variable x als Parameter bezeichnet wird. Als Argument bezeichnet man eine Variable oder einen Wert, mit denen eine Funktion aufgerufen wird. Bei Funktionen mit dem Rückgabetyp void schreiben Sie einfach return; oder lassen die return-Anweisung ganz weg.

Nach einem **return** wird die **Funktion sofort verlassen**, d.h. alle nachfolgenden Anweisungen des

Funktionsrumpfs werden ignoriert. Erwartet eine Funktion mehrere Argumente, so werden die Parameter durch Kommas getrennt. Eine mit

```
int g(int x, double y);
</cpp>

deklarierte Funktion könnte z.B. so aufgerufen werden:

<code cpp>
int a = g(123, -4.44);
```

Für eine leere Parameterliste schreiben Sie hinter dem Funktionsnamen einfach ().

```
int h(){ // Deklaration von h()

// Quellcode ...

}

int a = h(); // Aufruf von h()
```

# Übergabe der Parameter (=Argumente)

In C+ + gibt es mehrere Varianten, wie einer Funktion die Argumente übergeben werden können:

1. call-by-value
2. call-by-reference
3. call-by-pointer (siehe Kapitel Zeiger)

## call-by-value (Übergabe per Wert)

Bei **call-by-value (Wertübergabe)** wird der **Wert des Arguments in einen Speicherbereich kopiert**, auf den die Funktion **mittels Parametername zugreifen** kann. Ein **Werteparameter** verhält sich wie eine **lokale Variable**, die „automatisch“ mit dem richtigen Wert initialisiert wird. Der Kopiervorgang kann bei Klassen (Thema eines späteren Kapitels) einen erheblichen Zeit- und Speicheraufwand bedeuten!

```
#include <iostream>

void f1(int x) {
   x = 3 * x;
   std::cout << x << std::endl;
}

int main()
{
   int a = 7;
   f1(a); // Ausgabe: 21
```

```
    f1(5); // Ausgabe: 15

    std::cout << x; // Fehler! x ist hier nicht definiert
    std::cout << a; // a hat immer noch den Wert 7

    return 0;
}
```

### Weiteres Beispiel

```
#include <iostream>
#include <conio.h>

using namespace std;

//Prototyp, Deklaration
int sumW(int x,int y);    //Parameterübergabe per Wert (Value)

int main(int argc, char** argv) {
    cout << "Hello World" << endl;

    int a=5,b=6,erg=0;
    cout << "Parameteruebergabe per Wert" << endl;
    erg=sumW(a, b);
    cout << "a: " << a <<endl;
    cout << "b: " << b <<endl;


    return 0;
}

//Funktionendefinition - Parameterübergabe per Wert
int sumW(int x,int y){
    x+=1; //x=x+1; x++; ++x;
    y+=1;

    cout << "x: " << x <<endl;
    cout << "y: " << y <<endl;

    return x+y;
}
```

### Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

| Variable | Adresse | Wert |
|---|---|---|
| a | 0x0001 | 5 |
| b | 0x0005 | 6 |
| ... | . | . |

| Variable | Adresse | Wert |
|---|---|---|
| ... | . | . |
| x | 0x00a1 | 5 → 6 |
| y | 0x00a5 | 6 → 7 |

**Die Ausgabe des Programms ist**

```
Hello World
Parameteruebergabe per Wert
x: 6
y: 7
a: 5
b: 6
```

## call-by-reference (Übergabe per Referenz)

Die Sprache C kennt nur call-by-value. Sollen die von einer Funktion vorgenommen Änderungen auch für das Hauptprogramm sichtbar sein, müssen sogenannte Zeiger verwendet werden.

C+ + hingegen stellt Prozeduren und Funktionen ebenfalls **Zeiger (pointer)** zur Verfügung. C+ + gibt Ihnen aber auch die Möglichkeit, diese Zeiger mittels **Referenzen** zu umgehen. Beide sind jedoch noch Thema eines späteren Kapitels.

Im Gegensatz zu call-by-value wird bei **call-by-reference** die **Speicheradresse des Arguments** übergeben, also **der Wert nicht kopiert**. **Änderungen der (Referenz-)Variable betreffen** zwangsläufig auch die **übergebene Variable vom Hauptprogramm selbst** und bleiben nach dem Funktionsaufruf erhalten.

Um call-by-reference anzuzeigen, wird der **Operator &** verwendet, wie Sie gleich im Beispiel sehen werden. Wird keine Änderung des Inhalts gewünscht, sollten Sie den Referenzparameter als const deklarieren um so den Speicherbereich vor Änderungen zu schützen.

Fehler, die sich aus der ungewollten Änderung des Inhaltes einer übergebenen Referenz ergeben, sind in der Regel schwer zu finden.

Die im folgenden Beispiel definierte Funktion swap() vertauscht ihre beiden Argumente. Weil diese als Referenzen übergeben werden, überträgt sich das auf die Variablen mit denen die Funktion aufgerufen wurde:

```
#include <iostream>

void swap(int &a, int &b)
{
   int tmp = a; // "temporärer" Variable den Wert von Variable a zuweisen
   a = b; // a mit dem Wert von Variable b überschreiben
   b = tmp; // b den Wert der "temporären" Variable zuweisen (Anfangswert
von Variable a)
}
```

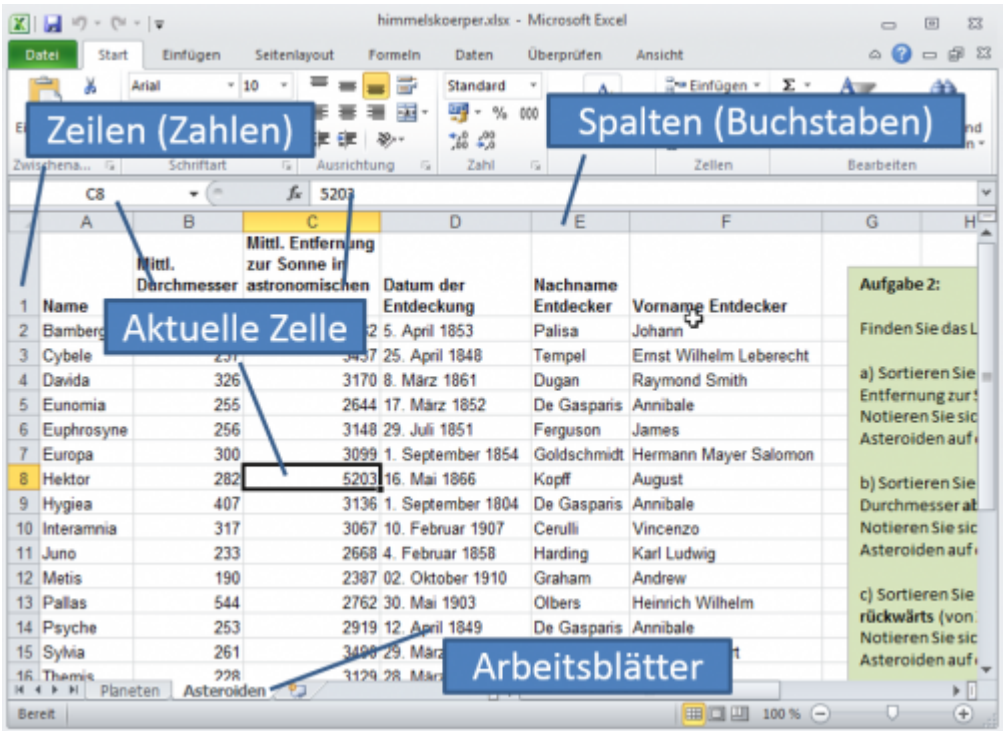
```
int main()
{
   int x = 5, y = 10;
   swap(x, y);
   std::cout << "x=" << x << " y=" << y << std::endl;

   return 0;
}

**Ausgabe**
x=10 y=5
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:05_algorithmik:5_08:5_08_07**

Last update: **2024/01/23 14:01**

# 6) Tabellenkalkulation




## Theorie

Eine Tabellenkalkulation ist eine Software für die interaktive Eingabe und Verarbeitung von numerischen und alphanumerischen Daten in Form einer Tabelle. Vielfach erlaubt sie zusätzlich die grafische Darstellung der Ergebnisse in verschiedenen Anzeigeformen.

Das Bildschirmfenster der Software ist dabei in Zeilen und Spalten eingeteilt. Je nach Programm bzw. Bedienungskonzept heißt dieser Bereich zum Beispiel Arbeitsblatt, Worksheet oder Spreadsheet. Jede Zelle der Tabelle kann eine Konstante (Zahl, Text, Datum, Uhrzeit …) oder eine Formel enthalten. Für die Formeln stehen meist zahlreiche Bibliotheksfunktionen zur Verfügung. Die Formeln können Werte aus anderen Zellen benutzen.

Bei Änderung der referenzierten Zellen einer Formel aktualisiert die Software den erst angezeigten Wert der Formelzelle normalerweise automatisch, ggf. aber auch nur auf Anforderung. Werden Formeln eines Tabellenfeldes an andere Stellen kopiert, muss zwischen absolutem und relativem Zellbezug unterschieden werden. Formelzellen können auf andere Formelzellen verweisen. Mit diesem Prinzip können komplizierte Rechengänge mit vielen verknüpften Teil-Ergebnissen übersichtlich dargestellt werden.

### Wozu?

Die Erfindung der elektronischen Tabellen zusammen mit Textverarbeitungs-Software und Datenbanken war zweifellos ein wesentlicher Faktor im Menschen den Wert von Mikrocomputern in den Anfangsjahren nach Apple und IBM begann Vermarktung Personalcomputer zu überzeugen. Seit dieser Zeit haben die stetig wachsende Vielseitigkeit und weiteren Anwendungen der

Tabellenkalkulations-Software es in ein Produkt gemacht, die scheint fast unentbehrlich für Unternehmen und Privatanwender. Tabellen sind jetzt ein Standardbestandteil der Office-Suite-Pakete.

## Funktionen

Kalkulationstabellen können mehr als einfache arithmetische Berechnungen durchführen. Eine Tabellenkalkulation übersetzen komplizierte Daten und Berichte in eine Kombination von Zahlen und Grafiken. Moderne Versionen enthalten eine umfangreiche Liste von finanzielle Rechner, z. B. Interesse Berechnungen, Kreditberechnung, auch Berechnungen für Schatzwechsel Preise. Statistische Funktionen von üblichen Berechnungen (Chi-Quadrat, Pearson-Koeffizient der Korrelation und Standardabweichung) auf abstruse Funktionen wie die hypergeometrische Verteilung und die Poisson-Verteilung Rückgabewerte benötigten mit keinen Schmerz. Es gibt mindestens 100 dieser Formeln in zeitgenössischen Tabellen enthalten.

Tabellen können als einfachen Datenbanken funktionieren. Durch das Einfügen von Daten und Anzahl in verschiedenen Spalten, können die Ergebnisse sortieren, gesucht oder gefiltert. Was-wäre-wenn-Analyse läuft, wenn Daten aus Zellen in einem Arbeitsblatt. Es gibt auch mathematische und trigonometrische Funktionen. Ein Benutzer kann eine vereinfachte Mailing-Liste in einer Kalkulationstabelle erstellen, durch die Eingabe von Namen und Adressen in einzelnen Spalten. Es gibt sogar Funktionen der Textverarbeitung bietet Kontrolle über Schriftarten, Fett oder kursiv-Schrift, Größe, Farbe und Seitenformatierung.

Bei Änderung der referenzierten Zellen einer Formel aktualisiert die Software den erst angezeigten Wert der Formelzelle normalerweise automatisch, ggf. aber auch nur auf Anforderung. Werden Formeln eines Tabellenfeldes an andere Stellen kopiert, muss zwischen absolutem und relativem Zellbezug unterschieden werden. Formelzellen können auf andere Formelzellen verweisen. Mit diesem Prinzip können komplizierte Rechengänge mit vielen verknüpften Teil-Ergebnissen übersichtlich dargestellt werden.

Verschiedene Arten des Einsatzes von Tabellenkalkulationsprogrammen können sein:

- Automatisches Ausführen und Aktualisieren von Berechnungen
- Datenspeicherung & Datenverwaltung
- Automatisches erstellen und aktualisieren von Diagrammen
- Modellierung
- Programmierung (Makros, Visual Basic)

## Einsatzmöglichkeiten

Vor allem aus dem kaufmännischen Bereich ist sie nicht mehr wegzudenken, wie ein Auszug aus den Anwendungsfeldern in der mittleren Spalte zeigt. Aber auch im wissenschaftlichen Bereich hat Tabellenkalkulation ihren Platz, insbesondere dann, wenn ein spezifischer Funktionsvorrat für Standardaufgaben aus den jeweiligen Bereichen angeboten wird (zB Statistikfunktionen, finanzmathematische Funktionen)

| **privat** | **betrieblich** | **wissenschaftlich** |
|---|---|---|
| Haushaltsbudget | Kalkulationen | Statistik |
| Angebotsvergleich | einfache Kostenrechnung | Charts |

| **privat** | **betrieblich** | **wissenschaftlich** |
|---|---|---|
| Kostenvergleich | Reisekostenabrechnung | mathematische Modelle |
| Kreditkostenberechnung | Statisitk | kaufmännische Anwendungen |
| uvm… | Charts | volkswirtschaftliche Modelle |
| | Investitutionsrechnung | …uvm |
| | einfache Fakturierung | |
| | ABC-Analyse | |
| | Budgetierung | |

### Einschränkungen

Jedoch ist auch ein Tabellenkalkulationsprogramm keine „Eierlegende Wollmichsau“ und man stößt auch hier an Grenzen:

- Datenverwaltung wird schnell aufwendig ⇒ Lösung: Datenbanksystem verwenden
- Beschränkt in der Grösse ⇒ Lösung: Datenbanksystem verwenden
- Zusammenhänge sind nicht sichtbar ⇒ Lösung: Gute Dokumentation
- Gefahr von Nebenwirkungen (side effects) ist gross ⇒ Lösung: Sorgfältig arbeiten

## Praxis

Folgende Themen solltest du beherrschen:

### Themen

1. Navigation & Layout & Ansicht (Zelle, Spalte, Zeile, Arbeitsblatt, Formeleditor, Druckansicht mit Druckbereich)
2. Markieren, Einfügen, Ausschneiden, Kopieren und Löschen von Zellen/Formeln und Inhalten
3. Formatierung & Layout (Spalten- & Zeilenbreite, Inhaltsbezogene Spaltenbreite, Zellen verbinden, Rahmen, Tabellenformatierungen, Bedingte Formatierung,..)
4. Formeln erstellen und anwenden
5. Sperren von Zeilen/Spalten und Zellen
6. Diagramme
7. Datenimport (CSV) inkl. Anpassungen (Zahlenformat)
8. Pivottabellen inkl. Datenanalyse

https://support.office.com/de-de/excel

Folgende Befehle/Funktionen sollte dir in MS Excel geläufig sein! Falls nicht erkunde die Hilfe von MS Excel.

- Öffnen/Speichern
- Navigieren mit Pfeiltasten & Tabulator
- Seite einrichten (Ränder, Ausrichtung, Format, Gitternetzlinien und Überschriften anzeigen/drucken)
- Arbeitsmappe
- Arbeitsblatt

- Zelle
- Namenfeld
- Spalte
- Zeile
- Bearbeitungsleiste/Formelzeile
- Automatisches Ausfüllen von Zellen
    - Sperren von Spalten und Zeilen
- Mehrere Spalten/Zeilen/Zellen markieren
- Breite/Höhe von mehreren Spalten/Zeilen ändern/automatisch an den Text anpassen
- Textausrichtung horizontal/vertikal
- Zellen verbinden
- Formatieren
- Bedingte Formatierung
- Sortieren und Filtern
- Grafiken einfügen
- Diagramme einfügen
- Daten einfügen
- Formeln
    - Grundrechnungsarten
    - Summe
    - Zählen
    - Anzahl
    - Anzahl2
    - Mittelwert
    - Maximum
    - Minimum
    - Median
    - Wenn
    - Oder
    - Und
    - Nicht
    - IstZahl
    - IstText
    - Summewenn
    - Zählenwenn
    - Verketten
    - Links
    - Rechts
    - Teil
    - SVerweis
    - Mittelwertwenn
    - …
- Ansichten (Umbruchvorschau)
- Pivot-Tabellen und -Charts
- Fenster fixieren (Zeile, Spalte)
- Spalten ein- und ausblenden
- Duplikate entfernen
- Namen definieren
- Sekundärachse Diagramm
- Szenario Manager
- Datenüberprüfung

- Transponiertes Einfügen
- 3D-Summenfunktion
- Pivot Tabelle Gruppieren
- Erweitertes Filtern
- Teilergebnisse

Folgende Shortcuts sind sehr hilfreich:

| **Shortcut** | **Funktion** |
|---|---|
| F2 | Zelle editieren |
| STRG+Bild rauf/runter | Tabellenblatt wechseln |
| STRG+Z | Rückgängig |
| STRG+A | Alles markieren |
| STRG+C | Zelle kopieren |
| STRG+V | Zelle einfügen |
| STRG+X | Zelle ausschneiden |
| STRG+F | Suchen & Ersetzen |
| STRG+Mausklick | mehrere einzelne Zellen markieren |
| STRG+Pfeiltaste rechts | gehe zur ganz rechten Spalte |
| STRG+Pfeiltaste links | gehe zur ganz linken Spalte |
| STRG+Pfeiltaste unten | gehe zur ganz letzten Zeile |
| STRG+Pfeiltaste oben | gehe zur ganz ersten Zeile |
| SHIFT+Mausklick | mehrere Zellen neben- bzw. untereinander markieren |
| STRG+SHIFT+* | Ganze Tabelle markieren |

## Übungen

- Arbeitsauftrag 1 - Excel
- Arbeitsauftrag 2 - Excel

siehe auch

ECDL Excel advanced

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:06_tabellenkalkulation**

Last update: **2024/03/04 07:27**

# 7) Web Design - Grundlagen



From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

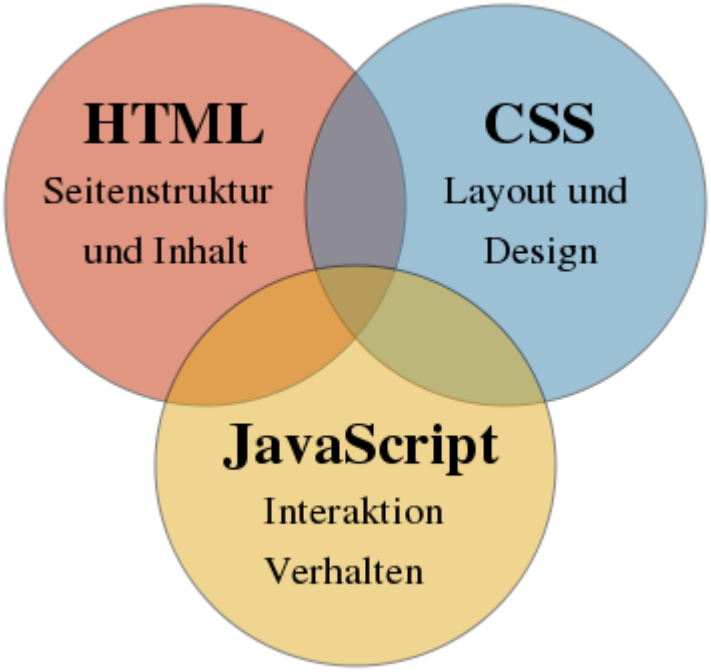
Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung**

Last update: **2024/04/27 20:34**

# HTML und CSS und Scripts

Im modernen Webdesign kommt den Webtechniken **HTML, CSS** und **JavaScript** jeweils eine bestimmte Rolle zu.

- **HTML** legt fest, was auf der Seite stehen soll **(struktureller Aufbau einer Webseite)**
- **CSS** legt fest, wie es dargestellt werden soll **(Formatierung & Gestaltung)**
- **JavaScript** legt fest, was passieren soll. **(interaktive Elemente)**




Dies entspricht auch dem in der Informatik und anderen Wissenschaftsbereichen anerkannten **Prinzip** der **Trennung von Zuständigkeiten (Separation of Concerns)**, die besagt, dass verschiedene Elemente der Aufgabe möglichst in verschiedenen Elementen der Lösung repräsentiert werden sollten. Um Webseiten möglichst effizient und einfach zu entwickeln sowie sie nachträglich mit geringem Aufwand pflegen zu können, sollten diese Aufgaben strikt voneinander getrennt werden:

- Im **Markup** werden keine Angaben zur Präsentation oder Interaktion gemacht.
- Im **Stylesheet** befinden sich demnach alle Angaben zur Präsentation in möglichst effizienter Weise.
- **JavaScript** wird nicht mit HTML-Attributen aufgerufen, sondern dynamisch an HTML-Elemente angehängt.

## HTML - Struktur für den Inhalt

Der Inhalt von Webseiten besteht grundsätzlich aus einem Text-Dokument. HTML soll diese Texte sinn- und bedeutungsvoll strukturieren, indem z.B. Überschriften, Listen, Absätze, Datentabellen, zusammenhängende Bereiche sowie wichtige Abschnitte, Zitate usw. als solche ausgezeichnet werden.

**Empfehlung**

Verwenden Sie möglichst bedeutungsvolle HTML-Elemente, die sich am Inhalt der Webseite orientieren. Vergeben Sie für die Feinstrukturierung sparsam gesetzte und aussagekräftige Klassen und IDs. Der Klassenbezeichner hinweis etwa ist semantischer als rot-unterstrichen. Klassen sollten den Grund ihrer Existenz in ihrem Namen tragen, nicht ihr visuelles Erscheinungsbild. Vermeiden Sie zusätzliche Elemente wie , die allein für die Gestaltung verwendet werden. Eine Webseite mit weniger Elementen verbessert die Ladezeit und damit die Benutzerfreundlichkeit.

# CSS - Formatierung und Gestaltung

CSS ist dafür da, die Regeln für Darstellung dieser Inhalte vorzugeben, sei es auf einem Smartphone, einem Desktop-Bildschirm, ausgedruckt auf Papier oder anders. Optimalerweise wird es in einem ausgelagerten Stylesheet notiert, das dann alle Seiten Ihres Webauftritts zentral formatiert. Alternativ können Sie so das Design Ihrer Webseite schnell und unkompliziert ändern. Das klassische Beispiel dafür ist der CSS ZEN GARDEN, in dem eine Webseite mit verschiedenen Stylesheets auf Knopfdruck immer wieder ein völlig anderes Layout bekommt.

**Empfehlung**

Setzen Sie Angriffspunkte für CSS-Selektoren wie Klassen und IDs nur sparsam ein - alternativ können Sie Elemente mit strukturelle Pseudoklassen wie :first-child etc. ansprechen, allerdings sind strukturelle Pseudoklassen bei Markup-Änderungen weniger robust als andere Selektoren.

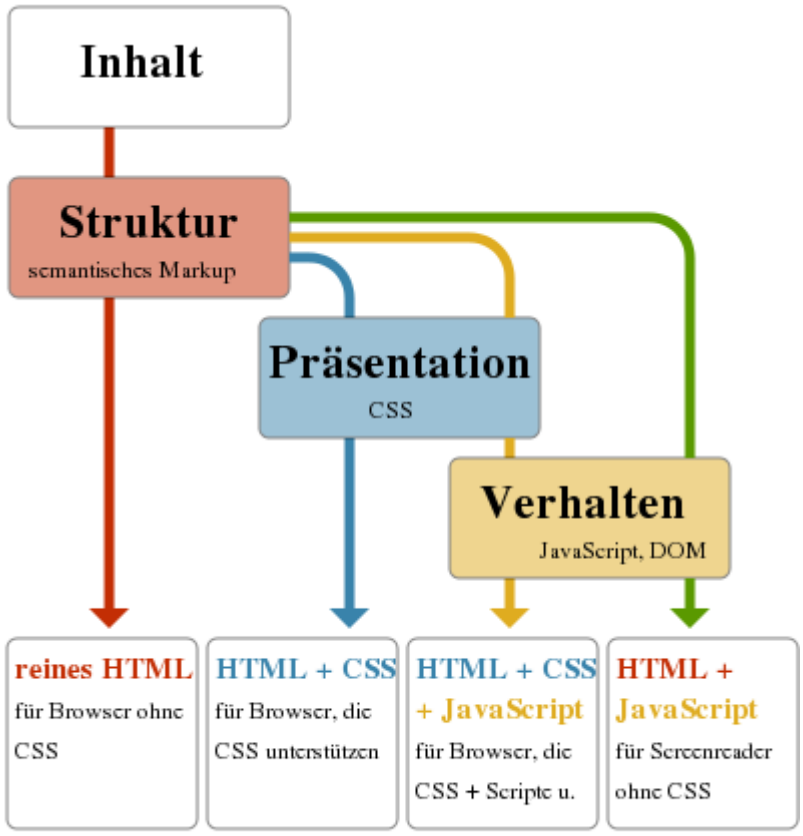
# JavaScript - Interaktivität/Verhalten

Wenn nun die dritte Technik – JavaScript – hinzutritt, sollten Sie dieses Modell konsequent fortführen. JavaScript kommt in diesem Konzept die Aufgabe zu, dem Dokument Verhalten (behaviour) hinzuzufügen. Damit ist gemeint, dass das Dokument auf gewisse Anwenderereignisse reagiert und z.B. Änderungen im Dokument vornimmt (DOM-Scripting).

## Das Schichtenmodell

Da diese drei Aspekte aufeinander aufbauen (mit der Struktur als Basis), kann man sie auch als Schichtenniodell betrachten (siehe Abbildung). Hier wird deutlich, dass die HTML-Struktur die Basis bildet, auf der die darüberliegenden Schichten aufsetzen.

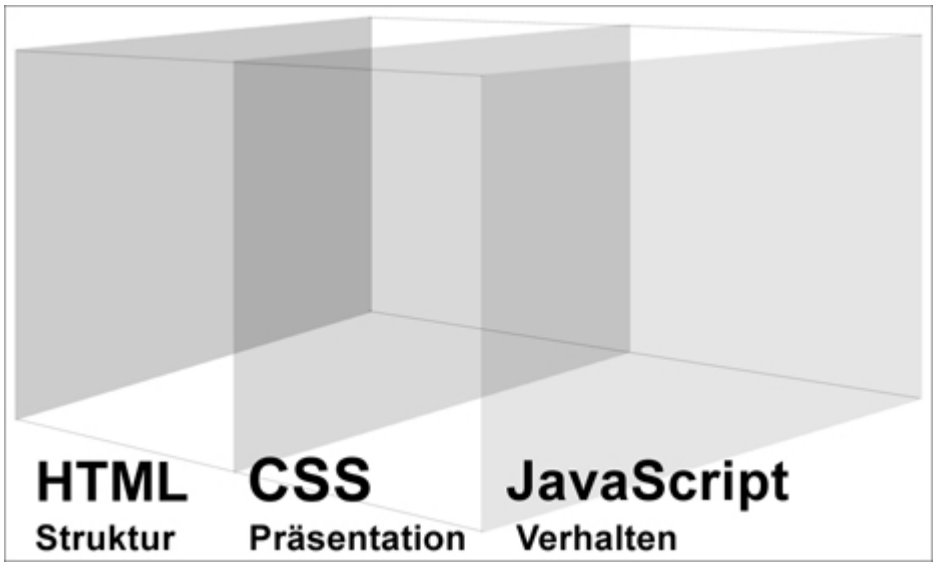
Tatsächlich liest der Browser auch zunächst diese Struktur ein, die er anschließend gemäß den CSS-Anweisungen »dekoriert;' und schließlich mit dem in Javascript beschriebenen Verhalten ausstattet. Setzen wir uns kurz mit den Grundlagen von HTML auseinander.




Die wichtigste Schicht eines Webdokuments ist das Markup. Dessen Inhalte müssen auch ohne Stylesheets oder JavaScripte sicht- und benutzbar sein.

Durch Stylesheets kann eine weitere Schicht über das Dokument gelegt werden, die ein Layout zuschaltet, das die Lesbarkeit verbessert und durch die Gestaltung die Webseite ansprechender und interessanter macht. Falls das Stylesheet nicht geladen werden kann, bleibt die Webseite aber weiterhin benutzbar.

JavaScript bildet die dritte Schicht, die dem Dokument Interaktivität und Verhalten gibt. Bei abgeschaltetem oder nicht verfügbaren JavaScript (z.B. bei Screenreadern) bleibt die Seite aber weiterhin benutzbar.



Die drei Ebenen - Struktur, Präsentation und Verhalten

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_01**

Last update: **2024/04/22 06:06**

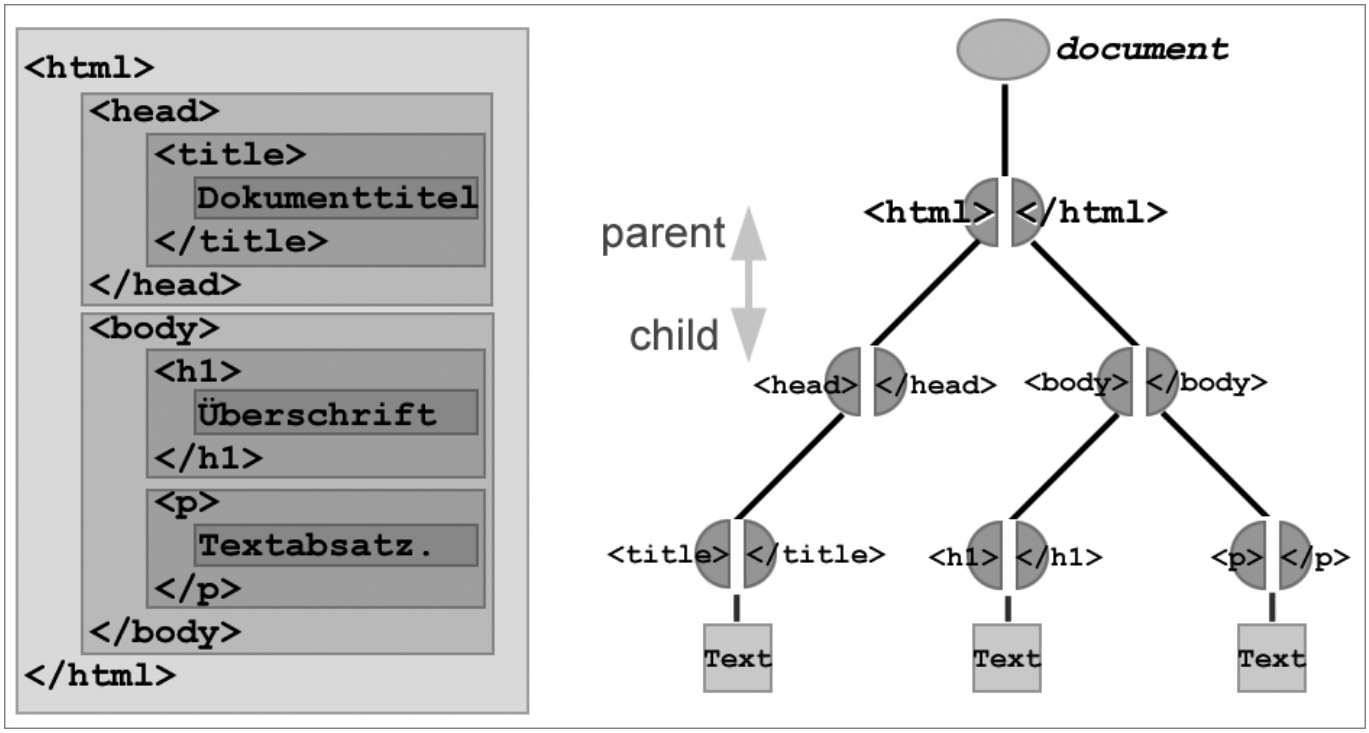
# Das Document Object Model

Ein wichtiges Konzept im Zusammenhang mit dem Zugriff auf Webseiten ist das Document Object Model (.DOM). Das DOM stellt die Schnittstelle zwischen der Dokumentstruktur und dem Rest dar und dient Ihnen einerseits zur Verknüpfung von HTML und CSS, andererseits ist es auch das Interface, das Ihnen erlaubt, per Script auf Struktur und Präsentation des Dokuments einzuwirken.

## Das Erstellen des DOM-Baums

Das DOM stellt eine Abstraktion des Dokuments dar, die der Browser beim Einlesen des HTML-Quelltexts im Arbeitsspeicher erzeugt. Eigentlich existiert es also rein virtuell. Stellen Sie es sich als Baumstruktur vor. und zwar in Form eines nach unten hängenden Baums, der an einem Punkt aufgehängt ist und sich ab dort verzweigt. Bezeichnen wir den Punkt, an dem der Baum ansetzt, als Dokumentknoten (`documentNode`). Dieser Dokumentknoten besitzt keine Entsprechung im Dokument, sondern stellt das »Dokument« an sich dar.

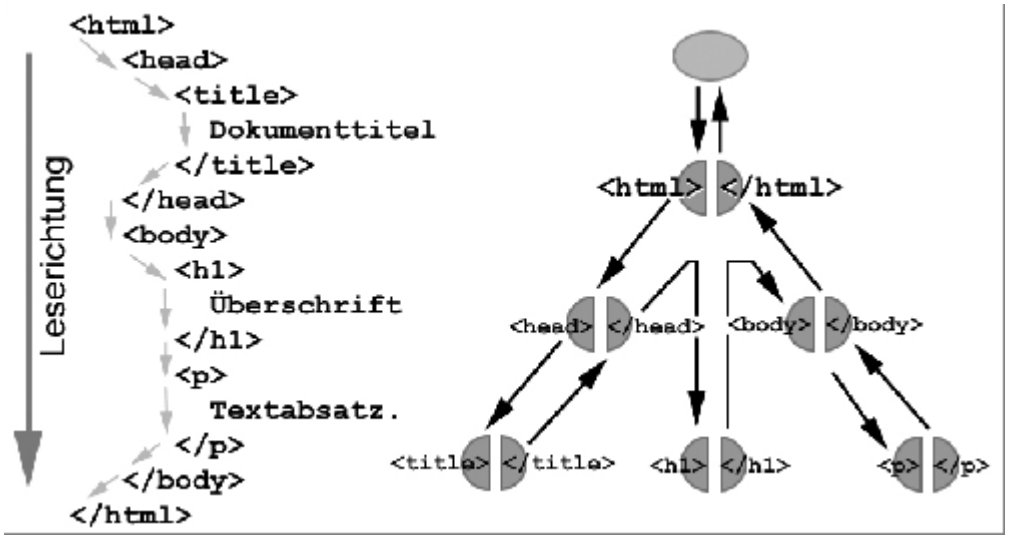
Einen solchen Dokumentknoten erzeugt der Browser beim Einlesen (Parsing) des Dokuments. Er hängt dann für jedes Element, das er in Quelltextreihenfolge antrifft, einen weiteren Knoten unten an diesen Dokumentknoten an. Das erste Element, das er antrifft, ist stets das Root-Element HTML (allgemein als `document`-Element bezeichnet). Ab hier spaltet sich der Baum in zwei Zweige, die den <head> und den <body> repräsentieren (siehe Abbildung).



Der HTML-Quelltext und seine DOM-Repräsentanz

Der Baum besteht also aus Verbindungen und Knoten. Im Falle eines Elementknotens fallen in diesem Start- und Endmarke des Quelltexts zusammen. Textknoten bilden stets das Ende eines Zweiges (»leaf nodes«). Enthält ein Element einen Inhalt (gehen wir also hierarchisch in dessen Inneres), fächert sich der Baum weiter nach unten auf. Hierdurch entsteht für jeden Bestandteil dieses Inhalts ein weiterer Ast mit daran hängenden Knoten.

Hierbei gilt der oben liegende Knoten als »Elternknoten« (»parent«), die von ihm unmittelbar abstammenden Knoten als »Kindknoten« (»children«). Sowohl Elemente als auch Textknoten stehen stets in einer Eltern-Kind-Beziehung, wobei jeder Knoten genau einen Elternknoten besitzt (niemals mehrere). Ein Element-knoten besitzt darüber hinaus weitere Eigenschaften, zu denen (salopp gesprochen) auch seine Attribute gehören. Der Browser baut auf diese Weise sukzessive ein komplettes Abbild der hierarchischen Struktur des Dokuments auf.




Das DOM ist also eine Abstraktion folgender Information:

- Wie ist der hierarchische Zusammenhang in der Dokumentstruktur?
- Welches Element ist an welcher Position der Hierarchie?
- Welche Eigenschaften hat es (z. B. Attribute, Inhalte, Nachbarelemente)?

Hier soll kurz ein Vorteil des DOM-Konzepts gegenüber »seriellem« Quelltext erwähnt werden, der darauf beruht, dass ein Knoten gleichzeitig Start- und Endmarke eines HTML-Tags repräsentiert: Bei einer Manipulation des Baums (wir werden gleich sehen wie das geht), also der Entnahme oder dem Hinzufügen einzelner Knoten oder ganzer Zweige, wird stets mit vollständigen Strukturen gearbeitet. Ein Dokument behält so stets seine Wohtgeformtheit, also die Art von regelmäßiger Struktur, die in der XMT.-Datenverarbeitung gefordert ist.

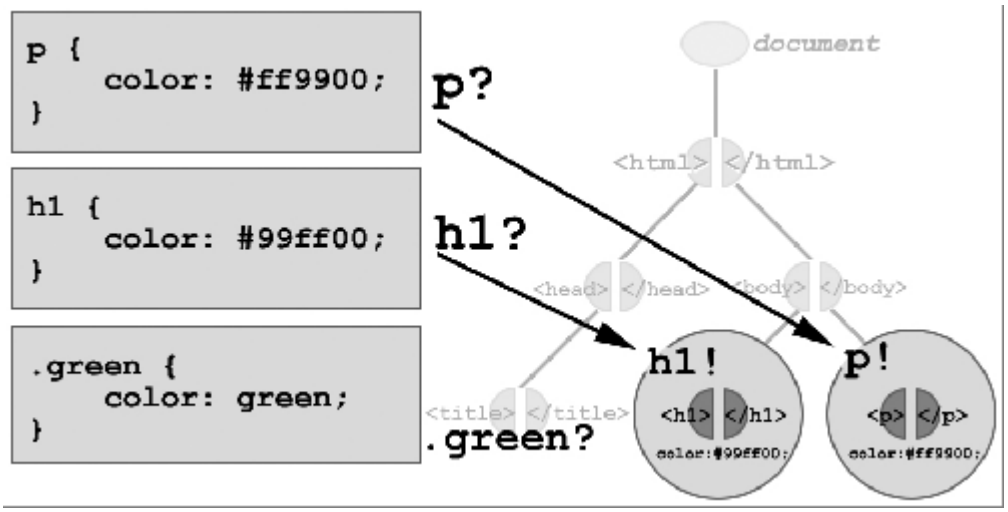
## Das »Schmücken« des DOM-Baums

Dieses Modell, in dem alle Elemente des Dokuments repräsentiert sind, ist noch eine Abbildung der »nackten« Struktur - durchaus wörtlich zu nehmen, denn bis jetzt hat der Browser die CSS-Informationen noch nicht eingebracht. Die nun folgende Phase wird als »decorating the tree« bezeichnet: Der Browser liest alle CSS-Informationen ein und löst dabei auftretende Konflikte und Unstimmigkeiten auf. Anschließend liegt das sogenannie Stylesheet vor: die vollständigen Präsentationsvorschriften, die auf das Dokument angewendet werden sollen.

Nun werden die Selektoren aller CSS-Regeln registriert und der Dokumentbaum anhand dieser Vergleichsmuster durchsucht. Jeder Elementknoten, auf den das Muster zutrifft, bekommt den Rcgclsatz zugewiesen. Auf diesem Weg werden alle Regeln durchgearbeitet. Regeln, für die keine Übereinstimmung mit dem Dokument gefunden wird, liegen brach (siehe folgende Abbildung).

Bekommt ein Element hierbei mehrfach Regeln zugewiesen, addieren sich diese. Ein dabei

auftretender Konflikt wird nach Rang der Regel beigelegt. Sobald der Vorgang beendet ist, ist der Baum »dekorierte und der Browser geht daran, ihn im Viewport darzustellen.



Der DOM-Baum wird per Vergleichsmuster durchsucht

Im dekorierten Baum sind folgende Informationen zusätzlich in den einzelnen Elementknoten gespeichert:

- Welche Präsentationsvorschriften existieren für dieses Element?

Die Stilevorschriften können wir uns durchaus als »Eigenschaften« des Elements vorstellen, die beim Elementknoten gelagert werden. Wir werden gleich sehen, dass dies uns hilft, die CSS-Regeln eines Elements auszulesen und auch bei Bedarf zu ändern. In diesem Augenblick kommt wieder JavaScript ins Spiel.

## Manipulation von DOM und CSS per JavaScript

Der dekorierte Baum stellt, wie wir jetzt wissen, ein Abbild des Dokuments im Arbeitsspeicher des Browsers dar, das der Darstellung des Dokuments im Viewport entspricht. Da es sich um eine rein virtuelle Sache handelt, ist dieses Abbild per Programmierung beliebig manipulierbar. Genau dies ist überhaupt die Aufgabe des DOM- eine Schnittstelle (API) zu bieten, die es ermöglicht, mittels einer Programmiersprache auf das Dokument einzuwirken.

In Zusammenhang mit JavaScript bietet das DOM eine Reihe von Schnittstellenfunktionen, die eine Brücke schlagen zwischen der Scripting-Umgebung und dem Dokumentbaum. Einige dieser Methoden sind dem Dokumentknoten (den wir hierfür praktisch erweise einlach als JavaScript-Objekt betrachten! unterstellt. Andere Methoden stehen auch direkt den Elementknoten zur Verfügung.

Diese DOM-Methoden sind nicht ausgesprochen zahlreich und zum Teil umständlich anzuwenden - ihre Anzahl hängt zudem von der Implementierung des DOM ab (»DOM-Level«), die der ausführende Browser jeweils unterstützt.

Dies sind ihre Aufgaben:

- Selektieren von Knoten
- Traversieren (Bewegung) innerhalb des Baums
- Wert (Inhalt) eines Knotens auslesen oder schreiben

- Attribute eines Elementknotens lesen oder schreiben
- Erzeugen von Element- und Attributknoten
- Einhängen von Knoten in den Baum
- Löschen von Knoten im Baum

| DOM-Methode | Erläuterung |
|---|---|
| addEventListener() | Bindet Event Handler an DOM-Element. |
| removeEventListener() | Löst Event Handler an DOM-Element. |
| createAttribute() | Erzeugt einen Attributknoten. |
| createTextNode() | Erzeugt einen Textknoten. |
| createElement() | Erzeugt einen Elementknoten. |
| getAttribute() | Liest einen Attributwert. |
| setAttribute() | Schreibt einen Attributwert. |
| removeAttribute() | Entfernt einen Attributknoten. |
| appendChild() | Hängt Elementknoten am Ende des Inhalts des aktuellen Knotens ein. |
| insertBefore() | Hängt Elementknoten vor dem aktuellen Knoten ein. |
| removeChild() | Entfernt Kindknoten des aktuellen Knotens. |
| replaceChild() | Ersetzt Kindknoten des aktuellen Knotens. |
| getElementById() | Referenziert einen Elementknoten per ID. |
| getElementsByClassName() | Erstellt NodeList aus Elementknoten nach CSS-Klasse. |
| getElementsByTagName() | Erstellt NodeList aus Elementknoten nach Tag-Bezeichner. |

Die wichtigsten DOM-Methoden zur DOM-Manipulation

---

Links:

- Was ist DOM ?
- Der DOM-Baum
- Eine umfassende Onlinereferenz zu DOM (in englischer Sprache) findet man bei der Mozilla Foundation unter dieser Adresse https://developer.mozilla.org/en/DOM

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_02**

Last update: **2024/04/22 06:06**

# Grundlagen in HTML



---

Links:

- SELFHTML, die HTML-Referenz
- W3-Schools (engl.)

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_03**

Last update: **2024/04/24 12:13**

# Grundbegriffe des Internets

## Browser

Browser sind Anwendungen, die zum Anzeigen einer Webseite dienen und das Navigieren im World Wide Web erlauben. Die derzeit am häufigsten verwendeten Browser sind:

- Internet Explorer
- Mozilla Firefox
- Netscape Navigator (veraltet)
- Opera
- Apple Safari
- Google Chrome
- Edge

Alle diese genannten Browser sind frei verfügbar.

## Domain

Das Internet ist in viele Gebiete, die sogenannten **Domains**, eingeteilt. Maßgeblich für die Einteilung ist das Land, in dem ein Internet-Server steht, aber auch die Organisation des Betreibers (etwa eine Universität oder eine Firma), dem es gehört.

Ein Domainname ist beispielsweise *www.microsoft.de* . Daraus ist ersichtlich, dass dies die deutsche Internetadresse der Firma Microsoft ist.

Eine kleine Übersicht über mögliche Länderkennungen (auch Top Level Domain):

| | |
|---|---|
| .at | Österreich |
| .de | Deutschland |
| .ch | Schweiz |
| .com | Commercial (kommerzieller Anbieter) |
| .net | Firmen, die einen umfangreichen Internet- oder Telefonservice bieten |
| .eu | Europa |
| .org | Organisationen |

## Download

Ist die Bezeichnung für das Überspielen einer Datei von einem Internet-Server auf die Festplatte des eigenen Computers.

## Upload

Ist die Bezeichnung für das Überspielen einer Datei vom eigenen Rechner auf einen Internet-Server. Bei den meisten Internet-Service-Provider (ISP) ist die Upload-Geschwindigkeit deutlich niedriger als die Download-Geschwindigkeit.

## FTP

Das File Tranfer Protocol (Dateiübertragungsprotokoll) dient speziell zur Übertragung von Dateien zwischen unterschiedlichen Rechnern.

Viele Internet-Provider bieten einen FTP-Zugang zur Übertragung von Daten auf eine Homepage.

## HTTP

Über das HyperText Transfer Protocol werden die Daten einer Webseite vom Internet-Server zum Browser übertragen.

## Server

Der Server ist ein Computer, der andere Computer, den Clients, eine Dienstleistung zur Verfügung stellt. Ein Internet-Server, auch Webserver genannt, stellt z.B. Daten bereit, die man sich im Browser anzeigen lassen oder herunterladen kann. Zur Kommunikation müssen Client und Server, wie z. B. über das Internet, vernetzt sein.

## URL

Der URL ist eine Art Adresse, die alle Angaben enthält, die für das Auffinden einer bestimmten Information im World Wide Web notwendig sind.

## WWW

Ist die Abkürzung für World Wide Web. Das WWW wird im allgemeinen Sprachgebrauch oft mit dem Internet gleichgesetzt, obwohl es jünger ist und nur eine mögliche Nutzung des Internets darstellt. Es gibt durchaus Internet-Dienste, die nicht in das WWW integriert sind (z.B. E-Mail).

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

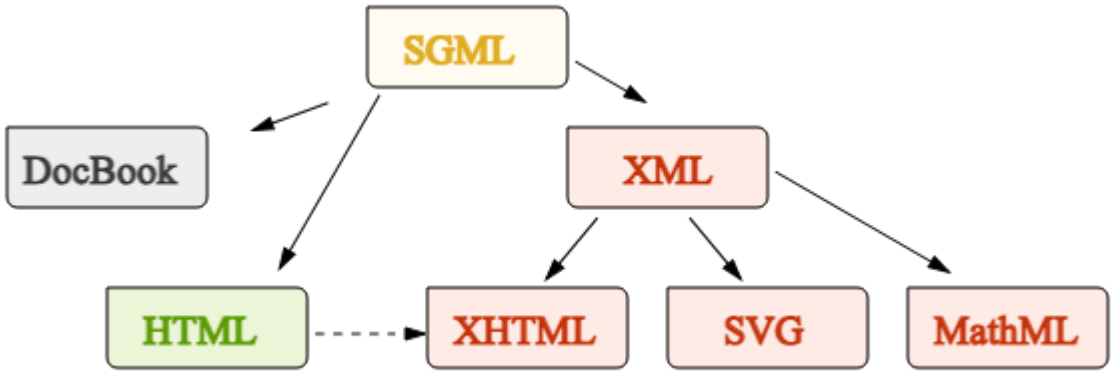
Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_03:7_03_01**

Last update: **2024/04/23 12:58**

# HTML - Einführung

HTML (Hyper Text Markup Language) ist die Standard Markup Sprache für das Erstellen für Webseiten.

Es beschreibt die Struktur und beinhaltet den Inhalt einer Webseite. Mithilfe von verschiedenen Tags (z.B.: <h1>) wird der Inhalt beschrieben (Überschrift, Paragraph, Link, Bild,...) und der Browser weiß, wie er diesen anzeigen soll.




## Grundgerüst

Das Grundgerüst besteht aus einigen wenigen Elementen (Tags) und sieht wie folgt aus:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Page Title</title>
  </head>
  <body>

    <h1>My First Heading</h1>
    <p>My first paragraph.</p>

  </body>
</html>
```

### Erklärung der Elemente

- <!DOCTYPE html> deklariert dieses Dokument als HTML-Dokument
- <html> ist das Wurzelelement einer Webseite
- <head> beinhaltet alle Meta-Informationen der Webseite
- <meta charset=„utf-8“> definiert, dass der geschriebene Inhalt im Dokument den Zeichensatz utf-8 verwendet

- <title> beinhaltet den Titel der Seite (Text im Tab)
- <body> definiert den Inhaltsbereich der Webseite und ist gleichzeitig der Container für alle sichtbaren Inhalte (Überschrift, Paragraphen, Titel, Bilder, Links, Tabellen, Listen,...)
- <h1> definiert die größte Überschrift
- <p> definiert einen Paragraphen

## Was ist ein HTML-Element?

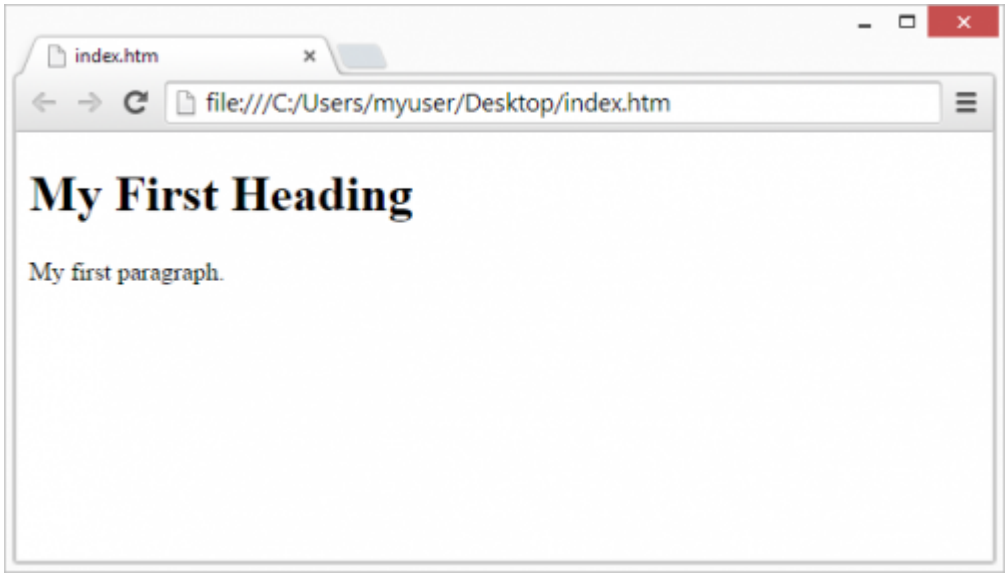
Ein HTMl-Element beinhaltet meist einen öffnenden Tag, den Inhalt und ein schließendes Tag.

z.B.:

```
<h1> My First Heading </h1>

<p>My first paragraph.</p>
```

Dennoch gibt es immer wieder Ausnahmen, die kein schließendes Tag besitzen. z.B.:

```
<br>
```




## HTML Seitenstruktur

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_03:7_03_02**

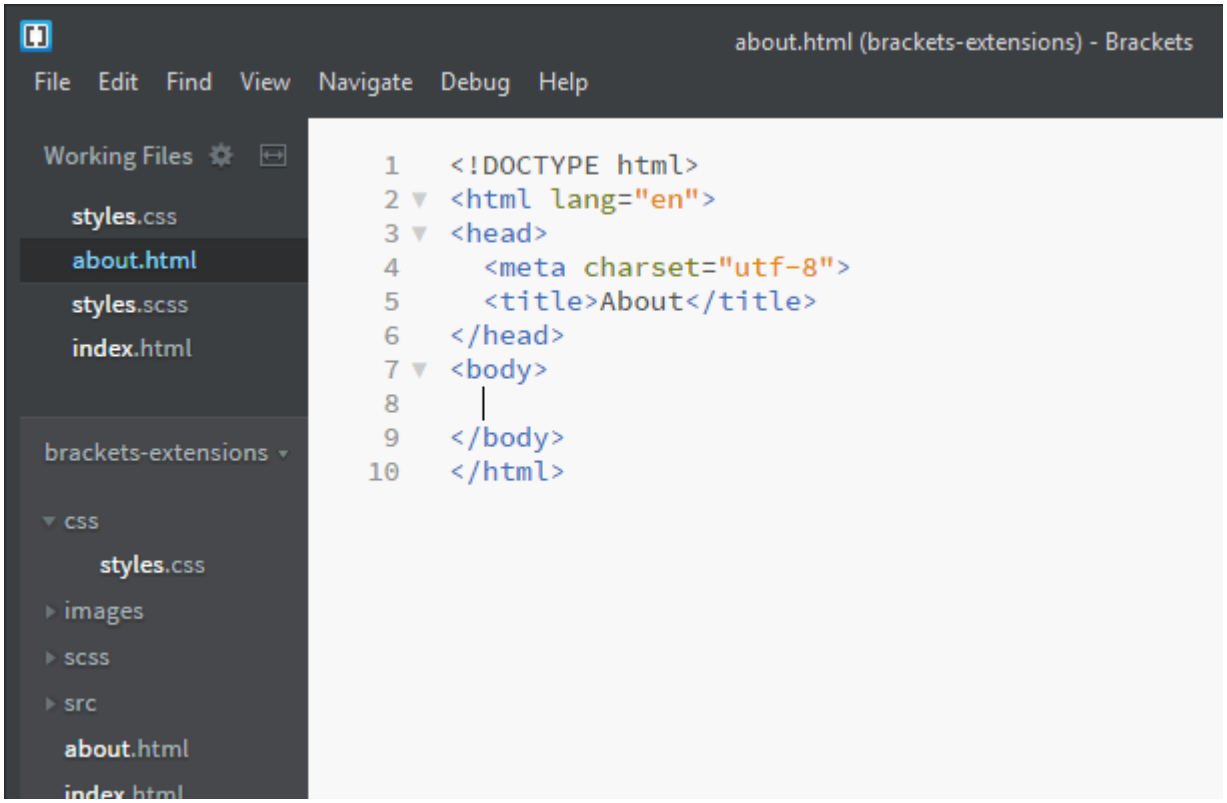
Last update: **2024/04/23 14:24**

# HTML Editor

Als HTML-Editor ist jeder Texteditor brauchbar. Die jeweilige HTML-Datei ist eine normale Textdatei und wird zumeist als .html oder .htm gespeichert.

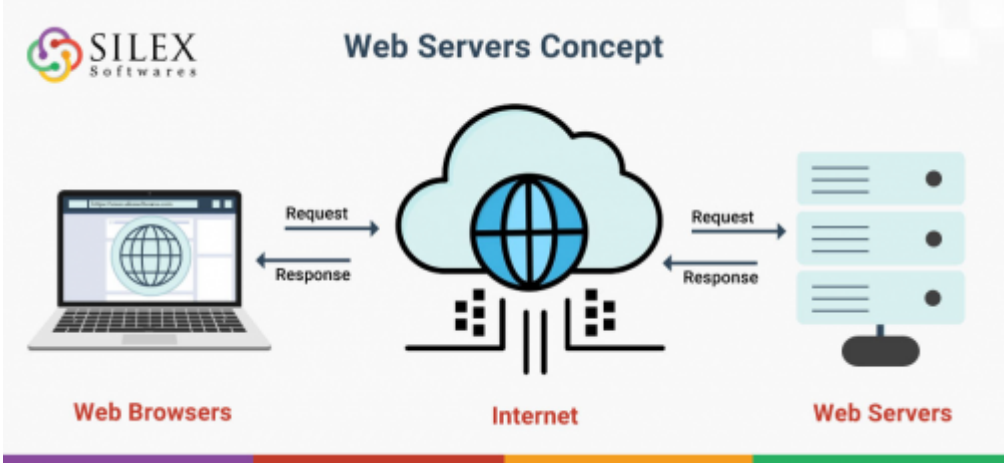
Dennoch gibt es verschiedene HTML-Editoren, die das Schreiben von HTML-Code bzw. das Entwickeln von Webseiten vereinfachen.

- Brackets
- Visual Studio
- Atom
- Nova
- Sublime Text 3
- WebStorm
- Notepad++




## Anzeigen einer Webseite

Für das lokale Anzeigen von HTML-Webseiten ist lediglich ein Webbrowser notwendig. Damit die Webseiten anschließend auch im Netzwerk bzw. Internet erreichbar sind, benötigt man zusätzlich einen Webserver inkl. Internetverbindung die den Zugriff auf den Server von außen zulässt.




From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_03:7_03_03**

Last update: **2024/04/23 14:23**

# Basiselemente

## Dokumenttyp

Jedes HTML-Dokument sollte mit einer Document Type Declaration beginnen:

```
<!DOCTYPE html>
```

Damit erkennt der Browser, dass diese Datei ein HTML-Dokument ist und versucht dieses optimal darzustellen. Dieses Tag darf außerdem nur einmal und zwar ganz oben im ganzen Dokument vorkommen!

## Grundgerüst

Ein valides HTML5-Grundgerüst beinhält mindestens die folgenden Elemente.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
  </body>
</html>
```

Der HTML-Tag beinhält anschließend das eigentliche HTML-Dokument. Wobei der Head-Bereich nur Metainformationen und der Body-Bereich alle sichtbaren Elemente des Dokuments beinhält.

Noch besser wäre die Angabe einer Sprache (lang) und weiteren Meta-Informationen im Head-Bereich:

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Titel</title>
  </head>
  <body>

  </body>
</html>
```

### Zeichencodierung

Man sollte immer die Zeichencodierung festlegen:

```
<meta charset="utf-8">
```

Beachte: Wenn man für HTML-Dokumente die Zeichencodierung durch <meta charset=„utf-8“> festlegt, muss man darauf achten, dass sich diese Angabe vollständig innerhalb der ersten 1024 Bytes des Dokuments befindet. Des Weiteren muss das Dokument auch mit dieser Zeichencodierung vom Server ausgeliefert werden, dafür einfacherweise auch in derselben Zeichencodierung abgespeichert sein.

### Viewportangabe

Um zu verhindern, dass Mobilgeräte die gesamte Web-Seite in einer sehr kleinen Ansicht darstellen, kann man eine Viewportangabe einstellen. Ohne Angabe setzen Browser die Breite einer Webseite auf 960px und verkleinern, sodass es in den Viewport passt. So erhält der Nutzer zwar einen Überblick, Text wird aber erst nach dem hineinzoomen lesbar.

Mit folgender Einstellung erreicht man, dass die Seite sich an den Viewport anpasst, aber dennoch ein Skalieren ermöglicht:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=yes">
```

Folgende Angaben sind möglich:

- **initial-scale**: (Zoom) (initial-scale=1.0 Inhalte werden 1:1 dargestellt)
- **user-scalable**: Mit diesem Attribut kann man definieren, ob der Nutzer auf der Seite zoomen kann (yes) oder nicht (no), der default-Wert ist yes.
- **minimum-scale** bzw. **maximum-scale**: ermöglichen es, den Zoomgrad einzuschränken. (z. B.: maximal scale=2.0 Inhalt kann maximal 2x-fach vergrößert werden.)

Da hiermit etwas eingestellt wird, das mit der Darstellung zu tun hat, ist eine Metaangabe natürlich nicht ideal. Langfristig wird diese Metaangabe auch entfallen können und statt dessen in CSS erfolgen. Die entsprechende Regel @viewport {} ist aber auch schon wieder zurückgezogen worden.

Man kann dieses meta-Element ohne jegliche Nebenwirkungen für jede Webseite einsetzen, um die Darstellung auf Mobilgeräten zu verbessern. Die Benutzbarkeit und der Komfort werden auf keinen Fall leiden.

Empfehlung: Unterbinde das Zoomen auf Mobilgeräten nicht. Verwende

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

# Überschriften

In HTML sind insgesamt 6 verschiedene (*<h1> bis <h6>*) Überschriften definiert

```
<h1>Wichtig</h1>
```

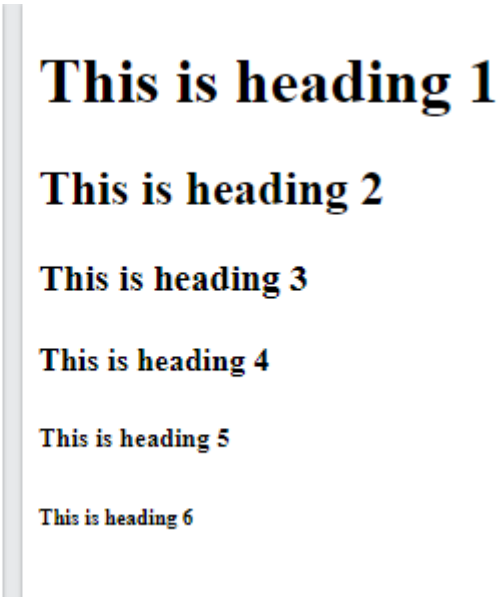
definiert die wichtigste und größte Überschrift

```
<h6>Nicht ganz so wichtig</h6>
```

definiert die unwichtigste und kleinste Überschrift

Bsp.: Code vs. Browseransicht

```
<!DOCTYPE html>
<html>
<body>

<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>

</body>
</html>
```




## Paragraphen

HTML Paragraphen werden wie folgt mit * Inhalt * definiert:

```
<!DOCTYPE html>
<html>
<body>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>

</body>
</html>
```

This is a paragraph.

This is another paragraph.

## Links

HTML Links werden wie folgt mit *<a href=„URL“>Text</a>* definiert und verkörpern die Grundidee des Internets.

```
<!DOCTYPE html>
<html>
<body>

<h2>HTML Links</h2>
<p>HTML links are defined with the a tag:</p>

<a href="https://www.w3schools.com">This is a link</a>

</body>
</html>
```

**HTML Links**

HTML links are defined with the a tag:

This is a link

Dabei ist das Ziel des Links im Attribut **href** definiert. Attribute beinhalten zusätzliche Informationen für den jeweiligen Tag.

## Bilder

Bilder werden wie folgt mit *<img src=„“>* definiert:

```
<!DOCTYPE html>
<html>
<body>

<h2>HTML Images</h2>
<p>HTML images are defined with the img tag:</p>

<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">

</body>
</html>
```

**HTML Images**

HTML images are defined with the img tag:


Das Attribut *src* beinhaltet wiederum den Pfad der Bildquelle. Das Bild muss dabei im Webverzeichnis des Webservers liegen, damit es angezeigt werden kann.

Das Attribut *alt* definiert einen alternativen Text, der dann angezeigt wird, wenn das Bild nicht eingeblendet werden kann.

## Verschachtelte Elemente

Elemente können auch verschachtelt werden (d.h. ein Element beinhält ein anderes Element).

Z.B.: Das Body-Element beinhaltet viele andere Tags wie *<h1>*, *<p>*,.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_03:7_03_04**

Last update: **2024/04/25 07:13**

# HTML Attribute

HTML Attribute beinhalten zusätzliche Informationen über das jeweilige HTML-Element.

- Alle HTML Elemente können Attribute besitzen.
- Attribute beinhalten Zusatzinformationen
- Attribute sind immer nur im Start-Tag
- Attribute sind normalerweise immer eine Kombination aus Name und Wert - z.B.: src=„Pfad“

## href

Das href-Attribut spezifiziert eine URL eines Links.

```
<!DOCTYPE html>
<html>
<body>

<h2>The href Attribute</h2>

<p>HTML links are defined with the a tag. The link
address is specified in the href attribute:</p>

<a href="https://www.w3schools.com">Visit W3Schools</a>

</body>
</html>
```

**The href Attribute**

HTML links are defined with the a tag. The link address is specified in the href attribute:

Visit W3Schools

## src

Das src-Attribut spezifiziert beim img-Tag den Pfad zum Bild, welches angezeigt werden soll.

```
<!DOCTYPE html>
<html>
<body>

<h2>The src Attribute</h2>
<p>HTML images are defined with the img tag, and
the filename of the image source is specified in
the src attribute:</p>

<img src="img_girl.jpg" width="200" height="300">

</body>
</html>
```

**The src Attribute**

HTML images are defined with the img tag, and the filename of the image source is specified in the src attribute:

Dabei gibt es zwei Methoden wie man Pfad angeben kann:

#### Absolute URL

Links zu externen Bildern, welche auf anderen Webserver gehosted liegen, müssen natürlich absolut

angegeben werden:

```
src="https://www.w3schools.com/images/img_girl.jpg".
```

Achtung: Externe Bilder sind womöglich geschützt (Copyright). Falls man dafür nicht die notwendigen Berechtigungen erhält, begeht man womöglich eine Datenschutzverletzung. Außerdem können externe Bilder jederzeit gelöscht bzw. geändert werden.

#### Relative URL

Links zu Bildern, welche am selben Server liegen wie die Webseite selbst. Hier benötigt man nicht die ganze URL, sondern man betrachtet den Pfad ausgehend von der jeweiligen HTML-Datei. Gibt man zum Beispiel als Pfad nur bild.jpg an, so muss die Datei im selben Verzeichnis wie die HTML-Datei selbst liegen.

Tipp: Verwende wenn möglich immer relative Pfadangaben, da diese auch bei einem etwaigen Server- bzw. Domainumzug noch weiter funktionieren.

## width und height

Mithilfe von width and height - Attributen kann man die Breite und die Höhe eines Bildes bereits in HTML ohne CSS spezifizieren.

## alt

Mithilfe des alt-Attributes kann man für Bilder <img> einen alternativen Text spezifizieren, der dann angezeigt wird, falls das Bild nicht angezeigt werden kann bzw. bei Screenreader vorgelesen wird.

```
<!DOCTYPE html>
<html>
<body>

<img src="img_typo.jpg" alt="Girl with a jacket">

<p>If we try to display an image that does not
exist, the value of the alt attribute will be
displayed instead. </p>

</body>
</html>
```

Girl with a jacket

If we try to display an image that does not exist, the value of the alt attribute will be displayed instead.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_03:7_03_05**

Last update: **2024/04/23 13:59**

# Seitenstrukturierung

Im ersten Schritt sollten Texte in Absätze geteilt und mit Überschriften versehen werden. Webseiten enthalten neben dem eigentlichen Inhalt zusätzlich noch weitere Elemente.

## Absätze

Absätze dienen der Gliederung eines Textes. Sie werden in HTML mit dem p-Element (engl: paragraph) ausgezeichnet.

```
<h1>Textabsätze definieren</h1>
<p>Hier beginnt ein Absatz, und hier ist er zu Ende.</p>
<p>Hier beginnt ein neuer Absatz, und hier ist er zu Ende.</p>
```

Das End-Tag *</p>* ist in HTML optional, d. h., es muss nicht zwingend angegeben werden. Es ist aber auch in HTML guter Stil das schließende *</p>*-Tag anzugeben. Das verdeutlicht zugleich auch die Bedeutung dieses Tags: Es umfasst einen Textblock – es steht nicht für den Zwischenraum zwischen zwei Absätzen.

### Abstände

Beim Erstellen von HTML-Dateien genügt es nicht, im Editor einen harten Zeilenumbruch einzufügen. Browser ignorieren solche Umbrüche (siehe auch Leerzeichen, Tabulatoren und Zeilenumbrüche).

Abstände zwischen Absätzen werden in HTML oft fälschlicherweise durch einen leeren Absatz oder durch eine Aneinanderreihung mehrerer Zeilenumbrüche ( br) realisiert.

falsch

```
<p>Ein Absatz mit Inhalt</p>
<p> </p>
<p>Ein Absatz mit Abstand</p>

<p>Ein Absatz mit Inhalt</p>
<br>
<br>
<p>Ein Absatz mit Abstand</p>
```

richtig

```
<p style="margin-bottom:2em;">Ein Absatz mit
Inhalt</p>
<p>Ein Absatz mit Abstand, der durch CSS
festgelegt wurde.</p>
```

**Wichtig:** Definiere stattdessen den gewünschten Abstand mithilfe der CSS-Eigenschaft margin.

### Trennlinien

#### Thematischer Bruch

Das Element *hr* kennzeichnet einen thematischen Bruch. Ursprünglich wurden solche Trenner als waagerechte Linien dargestellt, daher die Bezeichnung des Elements horizontal ruler, horizontales Lineal. In HTML5 bekommt das hr-Element die Bedeutung eines Themenwechsels auf Absatz-Ebene (paragraph-level thematic break).

Sichtbare Trennlinien dienen auch visuell der Abgrenzung von nicht unmittelbar zusammengehörigen Textabschnitten.

```
<p>Hier ist ein Abschnitt zu Ende.</p>
<hr>
<p>Und hier beginnt etwas Neues ohne eigene Überschrift.</p>
```

Hier ist ein Abschnitt zu Ende.

---

Und hier beginnt etwas Neues ohne eigene Überschrift.

Ein hr-Element hat keinen Inhalt.

#### Trennlinie mit CSS

Zur grafischen Gliederung zwischen einzelnen Teilen einer Adresse kann man Trennlinien einfügen, indem man ein Element mit CSS einen unteren Rand (border) gibt.

# Überschriften

Überschriften dienen zur Strukturierung eines Dokuments und bringen die folgenden Inhalte auf einen Punkt. Sie sind das wichtigste Merkmal für den Leser, sich im Dokument zurechtzufinden.

**Empfehlung**: Auf einer Seite sollte maximal eine Überschrift der ersten Ebene sein und dann den Abschnitten entsprechend der Hierarchie eine Überschrift einer tieferen Ebene.

Typischerweise reichen in einem Dokument Überschriften erster und zweiter Ordnung (h1 und h2) Sie bilden dort den Titel des Dokuments (h1) und eventuell nötige Zwischenüberschriften (h2) ab. HTML soll aber auch anderes als die „Allerweltswebsite“ können, zum Beispiel auch technische Dokumentationen oder wissenschaftliche Abhandlungen. Dort sind auch tiefere Schachtelungen des Inhalts üblich, die nur mit mehr als nur zwei Überschriftenebenen abzubilden sind. HTML hält dafür sechs Ebenen – eben h1 bis h6 – vor, bei Microsoft Word beispielsweise gibt es 9 Gliederungsebenen.

Weiters sollte keine Hierarchieebene ausgelassen werden - z.B.: <h1> und dann gleich <h3>. Falls die Überschrift zu groß ist, sollte man daher lieber das CSS anpassen.

Die Überschrift ist oft nicht nur das Erste, sondern auch das Einzige, was Besucher einer Webseite lesen. Im Webdesign ist die Überschrift noch wichtiger als im Print-Bereich, da Suchmaschinen den Inhalt der Überschriften für die Bewertung der Webseite heranziehen. Deshalb sollten Überschriften immer inhaltlich sinn- und wertvoll sein.

#### Die Überschrift soll auch alleine wirken

Bsp.: Neue Zeiten brauchen neue Entscheidungen

⇒ Es ist nicht erkennbar, um was es geht.

#### Kurz, knapp und prägnant: Vermeide unnötige Wörter und Nebensätze

Bsp.: Treffen Sie schnellere und bessere Entscheidungen mit Daten-Visualisierungen Daten-

Visualisierungen entdecken und verhindern Datenklau

⇒ Die untere Überschrift ist kürzer und inhaltlich genauer.

**Stellen Sie wichtige Schlüsselwörter an den Anfang.**

Neue Wege in der Daten-Analyse durch Daten-Visualisierungen Daten-Visualisierungen erleichtern Daten-Analyse

Überschriften sollten **kurz und aussagekräftig** sein. Sie können auch über zwei Zeilen gehen, dabei ist jedoch zu beachten, dass auf kleineren Viewports mehr Zeilen entstehen.

### Untertitel

Nicht immer kann man von einer Überschrift auf den Inhalt des eigentlichen Textes schließen, in solchen Fällen ist es sehr nützlich, im Untertitel (engl: subtitle, auch Unterüberschrift (engl: subline zu headline) noch zusätzliche Infos unterzubringen.

Verwende hier keine Überschrift einer tieferen Kategorie, sondern einen normalen Textabsatz:

```
<hgroup>
  <h1>Schreinerei Meier</h1>
  <p role="doc-subtitle">Ihr Partner für alles</p>
</hgroup>
```

Innerhalb eines *hgroup*-Elements befindet sich die *h1*-Überschrift und ein Untertitel. Dieser soll kleiner dargestellt werden. Da er aber kein Teil der Überschrift an sich ist, wird er in einem Absatz ausgezeichnet. Das hgroup-Element dient als reiner Container.

Damit der Absatz für Assistenzsysteme als Teil der Überschrift erkennbar ist, erhält er noch das ARIA-Attribute *role*.

## Listen

Es gibt zwei Arten von Aufzählungslisten, sortierte und unsortierte. Sie unterscheiden sich nur durch den Namen des Elternelements und können miteinander kombiniert werden:

### ul

Das Element ul, unordered list (englisch für ungeordnete, unsortierte Liste), beschreibt eine Liste, bei der die Reihenfolge der Elemente nur eine untergeordnete oder keine Rolle spielt.

Aufzählungslisten sind z. B. von Bedeutung, um Produkteigenschaften oder Argumente für eine These übersichtlich darzustellen. Bei einer Aufzählungsliste werden die Listeneinträge von grafischen Browsern standardmäßig mit einem Aufzählungszeichen (Bullet) versehen. Aufzählungslisten eignen sich sehr gut, um Navigationsleisten in Webseiten zu strukturieren.

```
<ul>
    <li>Probieren geht über Studieren</li>
    <li>Liebe geht über Triebe</li>
    <li>Tante fällt über Kante</li>
</ul>
```

- Probieren geht über Studieren
- Liebe geht über Triebe
- Tante fällt über Kante

<*ul*> leitet eine Aufzählungsliste ein.

Mit <*li*> beginnt ein neuer Punkt innerhalb der Liste (li = list item = Listeneintrag).

</*li*> beendet den Listeneintrag, </*ul*> die Liste.

Wie das Aufzählungszeichen (Bullet) dargestellt wird, bestimmt dabei der Browser. Dessen Darstellung kann jedoch mit list-style-type beeinflusst werden.

### Listen verschachteln

Das Verschachteln von Listen ist ebenfalls möglich. Zwischen <li> und </li> darf eine komplette weitere Liste stehen. Auch andere Listentypen sind dabei erlaubt.

```
    <ul>
        <li>Suchmaschinen
            <ul>
                <li>Google</li>
                <li>Bing</li>
                <li>Ask.com</li>
            </ul>
        </li>
        <li>Verzeichnisse
            <ul>
                <li>Yahoo</li>
                <li>Web.de</li>
                <li>DMOZ</li>
            </ul>
        </li>
        <li>Was anderes</li>
        <li>Noch was anderes</li>
    </ul>
```

- Suchmaschinen
  - Google
  - Bing
  - Ask.com
- Verzeichnisse
  - Yahoo
  - Web.de
- Was anderes
- Noch was anderes

Nach dem Text „Suchmaschinen“ sowie „Verzeichnisse“ wird das begonnene <li> zunächst nicht geschlossen. Stattdessen folgen unmittelbar die inneren Listen (grün dargestellt). Erst danach kommt das schließende </li> .

### ol

Das Element ol, ordered list (englisch für geordnete, sortierte Liste), bezeichnet eine Liste, bei der die Reihenfolge der Elemente von Bedeutung ist.

```
<ol>
  <li>bei Anette vorbeischauen</li>
  <li>bei Bert vorbeischauen</li>
  <li>bei Christine vorbeischauen</li>
  <li>bei Dieter vorbeischauen</li>
</ol>
```

1. bei Anette vorbeischauen
2. bei Bert vorbeischauen
3. bei Christine vorbeischauen
4. bei Dieter vorbeischauen

## HTML5 - Seitenstruktur

Früher wurde die Seitenstrukturierung durch div-Elemente, denen man spezielle ids oder Klassen gegeben hat, dargestellt. Auch wenn dies immer noch valider Code ist, gibt es gute Gründe, die vielen Vorteile von HTML5 zu nutzen.

Bei einer Vielzahl von div-Elementen (der so genannten Div-Suppe) fehlt gerade bei schließenden Tags die Übersicht.

**Suchmaschinen erkennen die semantische Bedeutung der HTML5-Elemente und werten den Inhalt besser aus.**

Screenreader erkennen die Bedeutung der HTML5-Elemente und lesen den Inhalt flüssiger vor.

Viele Nutzer jagen ihre Webseiten durch den Validator, ob ihre Webseite „richtig“ ist. „Valides HTML“ sagt, dass Sie die HTML-Elemente und -Attribute richtig verwendet haben. Das sagt nichts darüber, ob Sie die richtigen, semantisch passenden HTML-Elemente und -Attribute verwendet haben. Verwende die **neuen HTML5-Elemente** und verzichte auf divs, wo es nur geht. So sparen Sie sich unnötige Klassen und id-Selektoren und sowohl ihr HTML-Markup als auch ihr CSS werden übersichtlicher!

## Seitenkopf und -fuß

Der *body* ist der sichtbare Bereich unserer Webseite. Die meisten Webpräsenzen haben oben einen Seitenkopf, den sogenannten *header*, der Logo, Titel und die Navigationselemente enthält.

```
<!doctype html>
<html lang="de">
<head>
 ...
</head>

<body>
  <header>
    <img src="logo.svg" alt="logo">
    <p>Herzlich willkommen!</p>
  </header>

  <h1>Überschrift der Seite</h1>

  <footer>
     <a href="kontakt.html">Kontakt</a>
     <p>© 2014 by selfHTML</p>
  </footer>
</body>
</html>
```


Das **header**-Element gruppiert das Logo und eine Begrüßung. Zur besseren Veranschaulichung erhält das header-Element einen hellroten Hintergrund. Der Textabsatz wird größer dargestellt und erhält eine rote Textfarbe.

**Beachte**: Ein header mit nur einer Überschrift ist semantisch sinnlos. Er bläht nur das Markup unnötig auf und sollte weggelassen werden.

Weiters gibt es den **footer**, der Kontakt, Impressum, Copyright und evtl. die Sitemap enthält. Während allein schon der Name eine Position unterhalb des Inhalts suggeriert, ist es in Blogs und Forenpostings üblich, den footer rechts zu positionieren.

## Navigation

Die Haupt-Navigation wird durch ein nav-Element umschlossen, das aber auch für Unter-Navigationen verwendet werden kann.

```
<header> <img src="https://src.selfhtml.org/designs/design05/img/logo.svg"
alt="logo">
    <p>Herzlich willkommen!</p>
</header>
<nav>
    <ul>
        <li><a href="#link_1.html">Wiki</a>
        </li>
        <li><a href="#link_2.html">Blog</a>
        </li>
        <li><a href="#link_3.html">Forum</a>
        </li>
    </ul>
</nav>
<footer> <a href="kontakt.html">Kontakt</a>
```

```
    <p>© 2014 by selfHTML</p>
</footer>
```


Oben oder unten?

Während es früher oft empfohlen wurde, das Navigationsmenü unterhalb des Inhalts am Ende der Seite zu notieren und erst mit CSS oben zu positionieren, ist es heute üblich die Navigation unterhalb des header zu platzieren und Benutzern von Screenreadern bei umfangreichen Menüs einen Skip-Link zum Überspringen anzubieten.

## Hauptinhalt

Bisher wurde der Inhalt von HTML-Dokumenten vor allem über die verschiedenenen Überschriften h1 - h6 organisiert. In HTML5 gibt es für verschiedene Bereiche eigene Elemente mit eigener semantischer Bedeutung.

Das main-Element enthält den (Haupt-)Inhalt einer Webseite. Dieser wurde bisher oft mit  oder  gekennzeichnet.

```
<!doctype html>
<html lang="de">
  <head>
    …
  </head>
  <body>
    <header>
      <p>Name unserer Seite</p>
    </header>
    <nav>
      …
    </nav>
```

```
    <main>
      <h1>Überschrift des Artikels</h1>
      <p>weiterer Inhalt</p>
    </main>
  </body>
</html>
```


**article vs. section**

Bis jetzt waren die neuen HTML5-Elemente eine große Vereinfachung, aber für die Auszeichnung des Inhaltsbereichs gibt es nun gleich 3 Elemente:

- **main**: der Hauptinhalt der Seite (das role-Attribut ist nicht mehr erforderlich)
- **article**: ein in sich geschlossener Artikel, der Überschrift (und evtl. einige sections, einen eigenen header und footer) enthält.
- **section**: Abschnitte wie ein Kapitel, ein …

Besonders der Unterschied zwischen article und section ist oft Gegenstand heißer Diskussionen.

Verwende article, wenn der Inhalt in sich abgeschlossen ist Verwende section, wenn es mehrere ähnliche Blöcke gibt

```
<header>
</header>

<main>
<article>
  <h1>Überschrift</h1>
  <p>Dies ist meine erste HTML5-Seite1</p>
  ... mehr Inhalt
</article>

<aside>
  <section>
    <h2>Kontakt</h2>
    <ul>
      <li><a href="link_1.html">Wiki</a></li>
      <li><a href="link_2.html">Blog</a></li>
      <li><a href="link_3.html">Forum</a></li>
    </ul>
  </section>

  <section>
  <h2>Weiterführende Links</h2>
  <ul>
    <li><a href="link_1.html">Wiki</a></li>
    <li><a href="link_2.html">Blog</a></li>
    <li><a href="link_3.html">Forum</a></li>
  </ul>
  </section>
</aside>
</main>

<footer>
</footer>
```

Und sonst ist es auch völlig legitim, gelegentlich ein div-Element zur Strukturierung ohne weitere semantische Bedeutung zu verwenden.

### aside

Schon im Buchdruck gab es Seitenleisten und Randnotizen, die in HTML5 mit dem aside-Element dargestellt werden. Dabei ist es uns egal, ob diese Seitenleiste an der (rechten oder linken) Seite oder gar unten platziert ist, da dies später durch die CSS-Eigenschaften bestimmt wird. Wichtig ist nur, dass im aside-Block Informationen zum Inhalt der Webseite stehen, die aber nicht Teil des Inhalts der Webseite sind.

```
<header>
  <img src="logo.gif" alt="logo">
  <h1>Titel</h1>
</header>

<h1>Überschrift</h1>
<p>Dies ist meine erste HTML5-Seite1</p>
... mehr Inhalt

<aside>
  <h2>Weiterführende Links</h2>
  <ul>
    <li><a href="link_1.html">Wiki</a></li>
    <li><a href="link_2.html">Blog</a></li>
    <li><a href="link_3.html">Forum</a></li>
  </ul>
</aside>

<footer>
</footer>
```

### Fertiges Beispiel

```
<!doctype html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>HTML5-Seite mit Grundstruktur</title>
</head>

<body>
  <header>
    <img src="logo.gif" alt="logo">
    <h1>Titel</h1>
    <nav>
      <ul>
        <li><a href="#link_1.html">Startseite</a></li>
        <li><a href="#link_2.html">Unterseite 1</a></li>
        <li><a href="#link_3.html">Unterseite 2</a></li>
        <li><a href="#link_4.html">Kontakt</a></li>
      </ul>
    </nav>
  </header>

  <main>
```

```
    <article>
      <h2>CSS-basierte Layouts</h2>
      <h3>HTML-Struktur ohne CSS</h3>
      <p>Diese Seite enthält zunächst nur die Struktur und den Inhalt.
         Das Aussehen wird erst in folgenden Beispielen über CSS definiert.
      </p>
      <p>Dennoch ist diese Seite schon nutzbar und wird vom Browser
         entsprechend dessen Voreinstellungen bereits sinnvoll angezeigt.
      </p>
    </article>

    <aside>
      <h2>Weiterführende Links</h2>
      <ul>
        <li><a href="#link_1.html">Wiki</a></li>
        <li><a href="#link_2.html">Blog</a></li>
        <li><a href="#link_3.html">Forum</a></li>
      </ul>
    </aside>
  </main>

  <footer>
    <a href="#kontakt.html">Kontakt</a>
    <p>© 2014 by selfHTML</p>
  </footer>
</body>
</html>
```

Probiere es selbst aus!

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_03:7_03_06**

Last update: **2024/05/06 05:42**

# Textauszeichnung

Innerhalb einer Textpassage lassen sich besonders wichtige Informationen hervorheben – zum Beispiel durch Zentrieren, Einrücken, unterschiedliche Schriftgrößen und -farben oder Schriftschnitte in kursiv und bold. Während eine Textverarbeitung wie z. B. Word nur Wert auf das Aussehen legt, können Sie den Text von Webseiten nach seiner inhaltlichen Bedeutung auszeichnen, da Webseiten nicht nur am Bildschirm, sondern auch von Screenreadern gelesen und von Suchmaschinen ausgewertet werden.

**Beachte**: Bei den Elementen zur Textauszeichnung handelt es sich um sogenannte Inline-Elemente. Deshalb bleibt beispielsweise eine Zuweisung von Länge oder Breite zunächst einmal wirkungslos.

## Schreib- und Gestaltungsregeln

Es ist empfehlenswert, Texten eine moderne Seitenstruktur mit den neuen HTML5-Elementen wie article, aside und footer zu geben. Verwende diese Elemente und auch eigenschaftslose div-Elemente nur sparsam.

Jeder Textblock sollte seine eigene Überschrift haben. Durch die Verwendung der verschiedenen Überschriftenebenen können Sie dem Text eine auch für Screenreader und Suchmaschinen interpretierbare Struktur geben.

Abweichend von Textdokumenten haben sich im Webdesign einige Grundsätze herausgebildet.

Während DIN A4-Seiten oft im Blocksatz gestaltet sind, entstehen dadurch gerade auf kleineren Viewports zu große Lücken zwischen längeren Wörtern.

**Empfehlung**: Verwende im Webdesign Flattersatz und verzichte auf *text-align: justify;*.

## Hervorhebungen

#### span

Das span-Element (engl span für überspannen) ist ein Element, das Text und andere Inline-Elemente enthalten kann, selbst aber keine semantische Bedeutung hat und nichts bewirkt. Es ist dazu gedacht, um mit Hilfe von CSS formatiert zu werden.

#### HTML-Code

```
<h1>Auch <span>grüne</span> Weihnachtsbäume werden <span>braun</span>.</h1>
```

#### Browseransicht

**Auch grüne Weihnachtsbäume werden braun.**

### fett <b> oder kursiv <i>

**b** (für bold engl. für fett) zeichnet einen Teil eines Fließtextes aus, der sich vom übrigen Text abheben soll, keine besondere Bedeutung hat und üblicherweise in Fettschrift dargestellt wird. Das trifft etwa auf Schlüsselwörter oder Produktnamen zu.

*i* (von italic, kursiv) zeichnet einen Teil eines Fließtextes aus, der sich vom übrigen Text abheben soll, keine besondere Bedeutung hat und üblicherweise kursiv dargestellt wird. Das trifft etwa auf Fachbegriffe, Namen oder Textpassagen, die in einem anderen sprachlichen Zusammenhang stehen, zu.

#### HTML-Code

```
<p>Sie können sich auf der Kundenseite
  <b>http://example.org/kunden</b>
  einloggen und Ihre persönlichen Daten einsehen.
</p>
<p>
  Ihre Login-Daten lauten:<br>
  Name: <b>rumpel.stilz</b><br>
  Passwort: <b>q39a30qPPZ</b>
</p>
```

#### Browseransicht

Sie können sich auf der Kundenseite **https://example.org/kunden** einloggen und ihre persönlichen Daten einsehen.

Ihre Login-Daten lauten:
Name: **rumpel.stilz**
Passwort: **q39a30qPPZ**

-  - um eine andere Betonung auszudrücken,
-  - um eine starke Wichtigkeit auszudrücken oder
- <mark> - um Textabschnitte hervorzuheben.
- <dfn> - um eine Definition eines Begriffes auszuzeichnen.

### hoch- <sup> und tiefgestellt <sub>

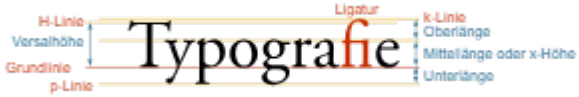
Man kann ein Textzeichen etwas oberhalb (hochgestellt) oder unterhalb (tiefgestellt) der normalen Basislinie des Typs festlegen:

$H_2O$ findet sich überall auf der Erde - alleine die Ozeane umfassen 1.332.000.000 $km^3$ (1.332 × 1018 $m^3$) Wasser. Der jährliche $CO_2$-Ausstoß wird nicht in Millionen $m^3$ , sondern in Millionen Tonnen gemessen.

Hier gibt es mehrere Möglichkeiten der Umsetzung:

Für die meisten Exponentionalzahlen gibt es passende Unicodezeichen. Die HTML-Elemente *sub* und *sup* können beliebigen Text hoch- bzw. tiefstellen. Dies wird in der Regel auch von Screenreadern vorgelesen. Mit MathML kann man auch komplexere Terme von der Grundlinie verschieben.

Eine Tiefstellung mit sub verschiebt einen Teil des Textes vertikal (Browserdefault: *sub { vertical-align: sub }*), sodass seine Zeilenbox aus der des übrigen Textes herausragt.




### gelöscht <del> und eingefügt <ins>

Gerade in immer wieder aktualisierten Web-Dokumenten ist es vorteilhaft, Änderungen (edits) am Dokument sichtbar zu machen.

*del* kennzeichnet einen Teil eines Textes als nicht länger gültig, deleted Text, gelöschter Text. *ins* kennzeichnet einen Teil eines Textes als eingefügt, inserted Text, eingefügter Text.

Es geht dabei weniger um richtig oder falsch als vielmehr darum, Änderungen nachvollziehbar zu dokumentieren. Wichtig ist das etwa für Gesetzestexte, die sich im Verlauf des Gesetzgebungsverfahrens ändern.

#### HTML-Code

```
 <p>Hänsel und <del>Bärbel</del><ins>Gretel</ins> verliefen sich im
Wald.</p>

    <del datetime="2013-03-25"
cite="https://de.wikipedia.org/wiki/Sonnensystem/">
      <p>In unserem Sonnensystem gibt es 9 Planeten.</p>
    </del>
```

#### Browseransicht

Hänsel und ~~Bärbel~~Gretel verliefen sich im Wald.

~~In unserem Sonnensystem gibt es 9 Planeten.~~

### Keyboard-Eingabe <kbd>

Das Element *kbd* zeichnet einen Teil eines Fließtextes als Benutzereingabe, etwa über die Tastatur, aus. kbd steht dabei für keyboard, eng. Tastatur.

Es wird in den browsereigenen Einstellungen meist nur in einer monospace-Schriftart dargestellt. Im CSS unseres Wikis werden Textauszeichnungen mit *kbd* mit einem typischen Tastatur-Look versehen:

#### HTML-Code

```
<h1>Tastenkombinationen mit kbd sichtbar machen</h1>
<h2>Tastaturen, Mäuse und Touchgeräte</h2>
<p>Webseiten im <a
href="https://wiki.selfhtml.org/wiki/JavaScript/Fullscreen">Fullscreen</a>-
```

```
Modus
    können mit <kbd>ESC</kbd> wieder in die klassische Ansicht zurückgesetzt
werden,
    nur nicht auf Touch-Geräten, da es dort keine Tasten gibt. Auch der
Klammergriff
    ( <kbd>strg</kbd> + <kbd>alt</kbd> + <kbd>entf</kbd> ) ist auf
Touchgeräten nicht
    möglich. </p>
```

#### CSS-Code

```
kbd {
    background: #f9f9f9 linear-gradient(to bottom, #eee, #f9f9f9, #eee)
repeat scroll 0 0;
    border: thin solid #aaa;
    border-radius: 2px;
    box-shadow: 1px 2px 2px #ddd;
    font-family: inherit;
    font-size: 0.9em;
    padding: 0 0.5em;
}
```

#### Browser-Ansicht

Tastenkombinationen mit kbd sichtbar machen

Tastaturen, Mäuse und Touchgeräte

Webseiten im Fullscreen-Modus können mit ESC wieder in die klassische Ansicht zurückgesetzt werden, nur nicht auf Touch-Geräten, da es dort keine Tasten gibt. Auch der Klammergriff ( strg + alt + entf ) ist auf Touchgeräten nicht möglich.

### Adresse <address>

Ein *address*-Element enthält Informationen, wie der Autor oder inhaltlich Verantwortliche einer Seite oder eines Abschnittes zu erreichen ist. Dies muss nicht zwangsläufig eine E-Mail- oder Postadresse sein, sondern kann auch einfach nur ein Link zu einer weiteren Seite mit Kontaktinformationen sein.

#### HTML-Code

```
      <address>
        Autor: <a href="../autoren/mustermann.html">Max Mustermann</a>
      </address>
```

Telefonnummern müssen kein Text sein. Wenn man sie als Link auszeichnet, können Benutzer auf mobilen Geräten mit einem Klick den Wählvorgang einleiten.

#### HTML-Code

```
<address>
    <dl>
```

```
      <dt>Tel.</dt>
      <dd><a href="tel:+49-89-00000000">089 0000 0000</a></dd>
     </dl>
</address>
```

Früher war es üblich Links auf eine E-Mail-Adresse mit mailto einzuleiten. Bei einem Klick auf den Link wird dann das Standard-Mailprogramm des Nutzers geöffnet.

**HTML-Code**

```
<p>
  Kontaktieren Sie uns per <a href="mailto:kontakt@example.com">Mail an
kontakt@example.com</a>
  oder mit nachfolgendem Formular:
</p>
```

Eine Alternative wäre ein Kontaktformular.

Fertiges Beispiel:

**HTML-Code**

```
<h2>Kontakt</h2>
  <address>
    <dl>
      <dt>Tel.</dt>
      <dd><a href="tel:+49-89-00000000">089 0000 0000</a></dd>

      <dt>Fax</dt>
      <dd>089 0000 0001</dd>

      <dt>Öffnungszeiten</dt>
      <dd><span>Mo–Fr:</span> <span>7:00–21:00 Uhr</span></dd>
      <dd><span>Sa:</span> <span>9:00–15:00 Uhr</span></dd>

      <dt>E-Mail Adresse</dt>
      <dd><a href="mailto:kontakt@example.com">kontakt@example.com</a></dd>

      <dt>Geschäftsadresse</dt>
      <dd>Fiktives Unternehmen<br/>
        Musterstraße 93<br/>
        80331 München
      </dd>

      <dt>Rücksendungen an</dt>
      <dd>Fiktives Unternehmen<br/>
        Musterstraße 93<br/>
        80331 München
      </dd>
    </dl>
  </address>
```

### CSS-Code

```css
dl.grid {
    display: grid;
    grid-template-columns: 1fr 100%;
}

dd {
    margin: 0;
    padding-left: 1em;
}

dl.grid dd {
    margin-bottom: 1em;
}

a[href^="tel"] {
    white-space: nowrap;
}
```

### Browser-Ansicht

# Kontaktadressen im Grid Layout

## Kontakt

*Öffnungszeiten*
*Mo–Fr: 7:00–21:00 Uhr*
*Sa: 9:00–15:00 Uhr*

| | |
|---|---|
| *Tel.* | *089 0000 0000* |
| *Fax* | *089 0000 0001* |
| *E-Mail Adresse* | *kontakt@example.com* |
| *Geschäftsadresse* | *Fiktives Unternehmen*<br>*Musterstraße 93*<br>*80331 München* |
| *Rücksendungen an* | *Fiktives Unternehmen*<br>*Musterstraße 93*<br>*80331 München* |

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_03:7_03_07**

Last update: **2024/04/24 08:48**

# Block- vs. Inline-Elemente

Bis HTML 4.01 wurden HTML-Elemente grundsätzlich den beiden Gruppen Block- und Inline-Elemente zugeordnet.

## Block-Elemente

Block-Elemente (engl. block-level elements) sind dadurch gekennzeichnet, dass sie standardmäßig einen Zeilenumbruch vor und nach sich erzwingen. Sie stehen dann sozusagen als einzelner Block in einer Zeile und nehmen die gesamte Breite des Viewports ein.. Block-Elemente können nur Nachfahre des body-Elements oder eines anderen Block-Elements sein. Sie entsprechen in HTML5 in weiten Teilen der Kategorie flow content.

#### Beispiele

- h1
- div
- p
- …

## Inline Elemente

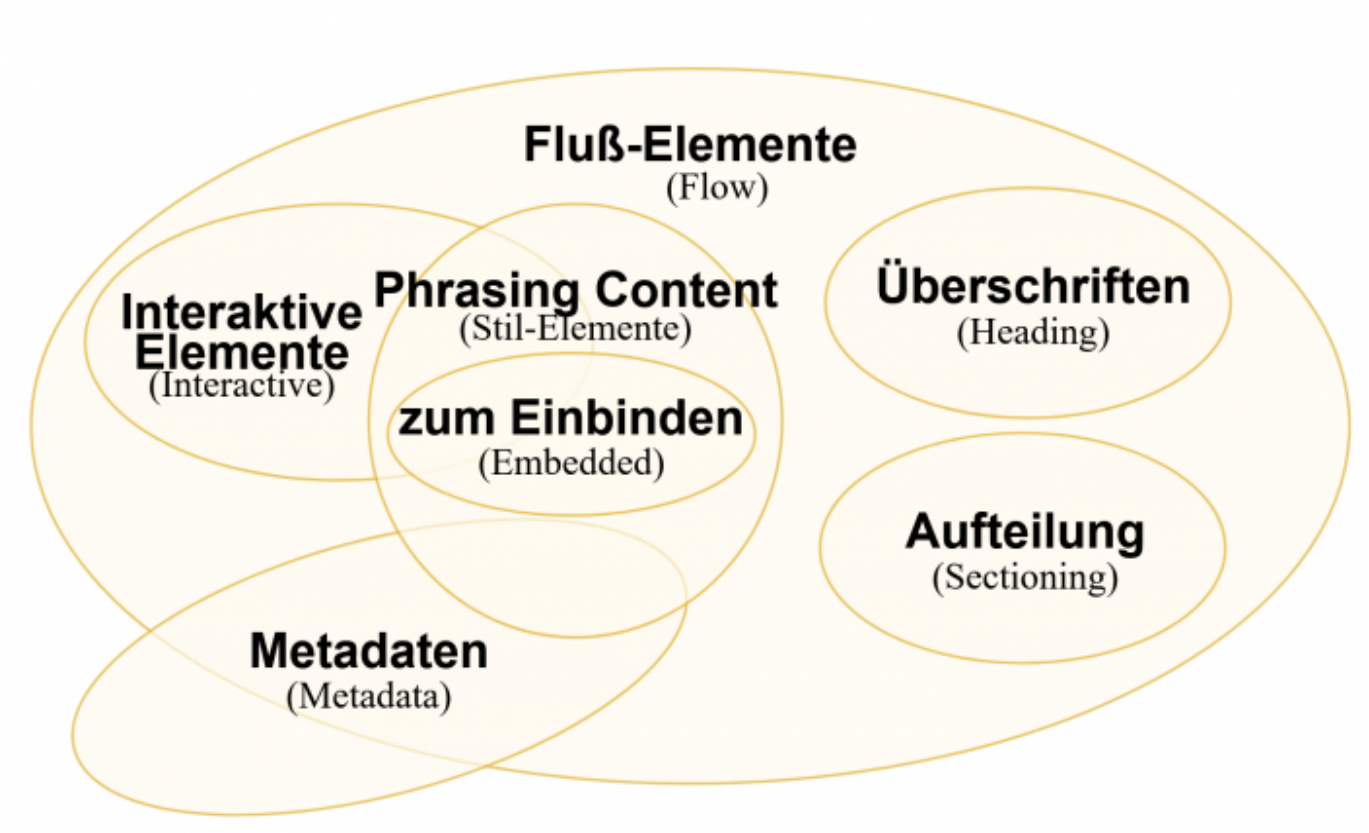
Inline-Elemente (engl. inline elements) sind dadurch gekennzeichnet, dass sie standardmäßig keinen Zeilenumbruch erzwingen. Sie befinden sich somit innerhalb des Text-Flusses. Inline-Elemente können Nachfahre eines Block-Elements oder eines anderen Inline-Elements sein. Sie entsprechen in HTML5 in weiten Teilen der Kategorie phrasing content.

#### Beispiele

- span
- a
- em
- …

# Kategorien von HTML-Elementen

HTML-Elemente können hinsichtlich des Inhalts, den sie enthalten dürfen, kategorisiert werden:




siehe Auflistung mittels interaktive Kategorien

Die Kategorie-Einteilung der Elemente erfolgt so, dass man die Kategorie so genau wie möglich angibt. Wenn es für ein Element nur heißt, dass es zum flow content gehört, dann bedeutet das, dass eine genauere Angabe nicht möglich ist und dass es damit nicht in einer der Teilmengen zu finden ist. Zum Beispiel heißt es bei fieldset: Darf vorkommen in allen Elementen, die flow content als Inhalt erlauben. Diese Formulierung schließt Elemente wie p als Elternelement für fieldset aus, die nur eine Teilmenge von flow content, nämlich phrasing content, als Inhalt gestatten.

## Metadaten

Metadaten (engl. metadata) beeinflussen das gesamte Dokument („die gesamte HTML-Seite“). Dazu zählt beispielsweise das Einbinden externer CSS-Dateien.

Elemente dieser Kategorie: *base, link, meta, noscript, script, style, template und title*

## flow content

flow content (engl. flow) enthält in der Regel Text oder phrasing content.

Elemente dieser Kategorie: *a, abbr, address, article, aside, audio, b, bdo, bdi, blockquote, br, button, canvas, cite, code, command, data, datalist, del, details, dfn, div, dl, em, embed, fieldset, figure, footer, form, h1, h2, h3, h4, h5, h6, header, hr, i, iframe, img, input, ins, kbd, keygen, label, main, map, mark, math, menu, meter, nav, noscript, object, ol, output, p, pre, progress, q, ruby, s, samp, script, section, select, small, span, strong, sub, sup, svg, table, template, textarea, time, ul, var,*

*video, wbr und einfacher Text.*

# Überschriften

Überschriften-Elemente (engl. headings) beschreiben einen Abschnitt. Der Abschnitt kann entweder explizit über Elemente zur Aufteilung beschrieben oder durch die Überschrift selbst begonnen werden.

Elemente dieser Kategorie: *h1, h2, h3, h4, h5 und h6*

## sectioning content

Elemente zur Aufteilung (engl. sectioning):

Elemente dieser Kategorie: *article, aside, nav und section*

## phrasing content

phrasing content differenziert Formulierung und Darstellungsebene ihres Inhalts.

Elemente dieser Kategorie: *abbr, audio, b, bdo, br, button, canvas, cite, code, command, datalist, dfn, em, embed, i, iframe, img, input, kbd, keygen, label, mark, math, meter, noscript, object, output, progress, q, ruby, samp, script, select, small, span, strong, sub, sup, svg, textarea, time, var, video, wbr und einfacher Text, wobei der Text nicht ausschließlich aus Whitespace bestehen sollte*

## embedded content

embedded content (engl. für eingebette Inhalte) sind Multimedia-Dateien, die nicht direkt Teil der Webseite sind. So können Inhalte aus anderen Quellen in das aktuelle Dokument geladen werden.

Elemente dieser Kategorie: *audio, canvas, embed, iframe, img, math, object, svg und video*

## Interaktive Elemente

nteraktive Elemente (engl. interactive) dienen der Nutzerinteraktion und haben ein browsereigenes Standardverhalten, das eine bestimmte Funktionalität zur Verfügung stellt.

Elemente dieser Kategorie: *a (mit href-Attribut), button, details, embed, iframe, keygen, label, select und textarea*

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_03:7_03_08**

Last update: **2024/05/06 05:46**

# 7.4) CSS



From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04**

Last update: **2024/09/11 05:08**

# CSS Grundlagen

CSS (Cascading Style Sheets, zu deutsch „Mehrstufige Formatvorlagen“) ist eine Formatierungssprache für HTML-, SVG- und XML-Dokumente. Warum Sie Layouts mit CSS einsetzen sollten, wird in diesem Artikel genauer erläutert.

## Gründe für CSS

Der „große Sinn von CSS“ besteht in der **Trennung von Inhalt und Design**. Das hört man oft, stellt sich nichts drunter vor und bastelt dann doch eine besondere Überschrift mit einem inline-style, einen neuen div-Container mit einer ganz speziellen ID oder Klasse, die später nirgends wieder auftaucht. Das funktioniert auch mehr oder weniger – man wird ohne viel Mühe eine statische Seite zusammenstellen, die ihren Zweck erfüllt – und vergisst das ganze.

Es ist aber beinahe unmöglich, einen so über die Zeit gewucherten Internetauftritt umzugestalten – man müsste dutzende oder mehr Einzeldateien umschreiben und im Gewusel der Klassen und Elemente wird man sich schnell verlieren. Letztlich dauert die Änderung beinahe länger als eine Neuerstellung.

Und genau darin liegt die Stärke von CSS: uneingeschränkte Flexibilität, wenn z. B. das Layout nicht mehr zeitgemäß ist oder neue Strukturen, vor allem bei dynamischen Seiten, Änderungen erfordern. Dabei wäre man ohne CSS buchstäblich „verloren im Quelltext“.

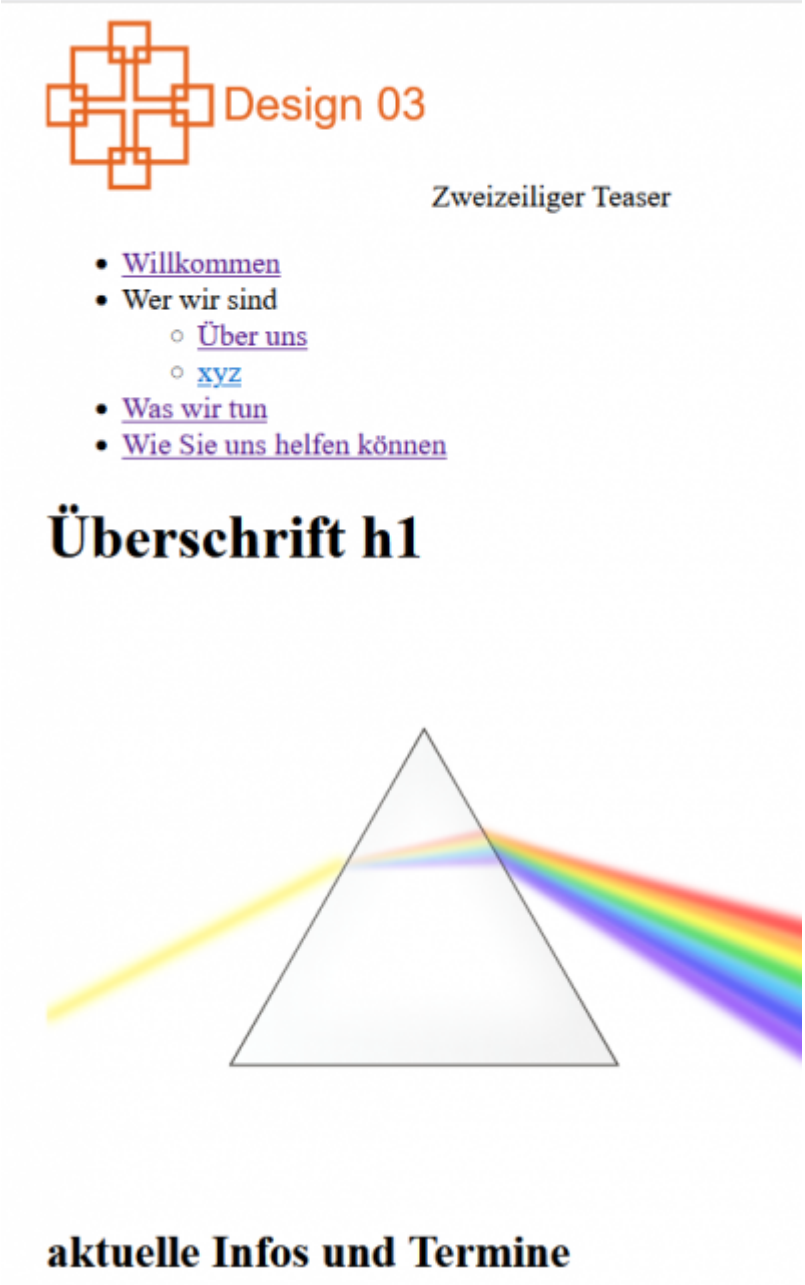
## Layout - am Anfang ohne CSS!

Das erste Gebot von CSS lautet: „am Anfang ohne.“

Soll heißen – man gestaltet die Seite erst komplett ohne CSS, nur die Seitenstrukturierung in Kapitel und Absätze, bei diesen dann die Textauszeichnung von Überschriften und Ähnlichem. Es stellt auch die „allgemeine Gültigkeit“ des Layouts sicher – ist die Grundlage an sich stimmig, kann man die Seite beliebig umgestalten, jeder Fehler schränkt dies wieder ein.

Eine solche Webseite sieht nicht gut aus, ist aber uneingeschränkt nutzbar. Die einzelnen Elemente werden vom Browser einfach untereinander auf dem Bildschirm angeordnet, und zwar in der Reihenfolge, in der sie in der HTML-Datei aufgeschrieben sind. Die Elemente selbst werden gemäß den Voreinstellungen des Browsers dargestellt; so ist z. B. festgelegt, dass der Hintergrund des Dokuments weiß oder grau darzustellen ist, dass Überschriften in fett und einer Größe von soundsovielen Punkten darzustellen sind.

Als Beispiel eine Webseite, die wir zu Demonstrationszwecken ihrer CSS-Datei beraubt haben:




## CSS als Sitechanger

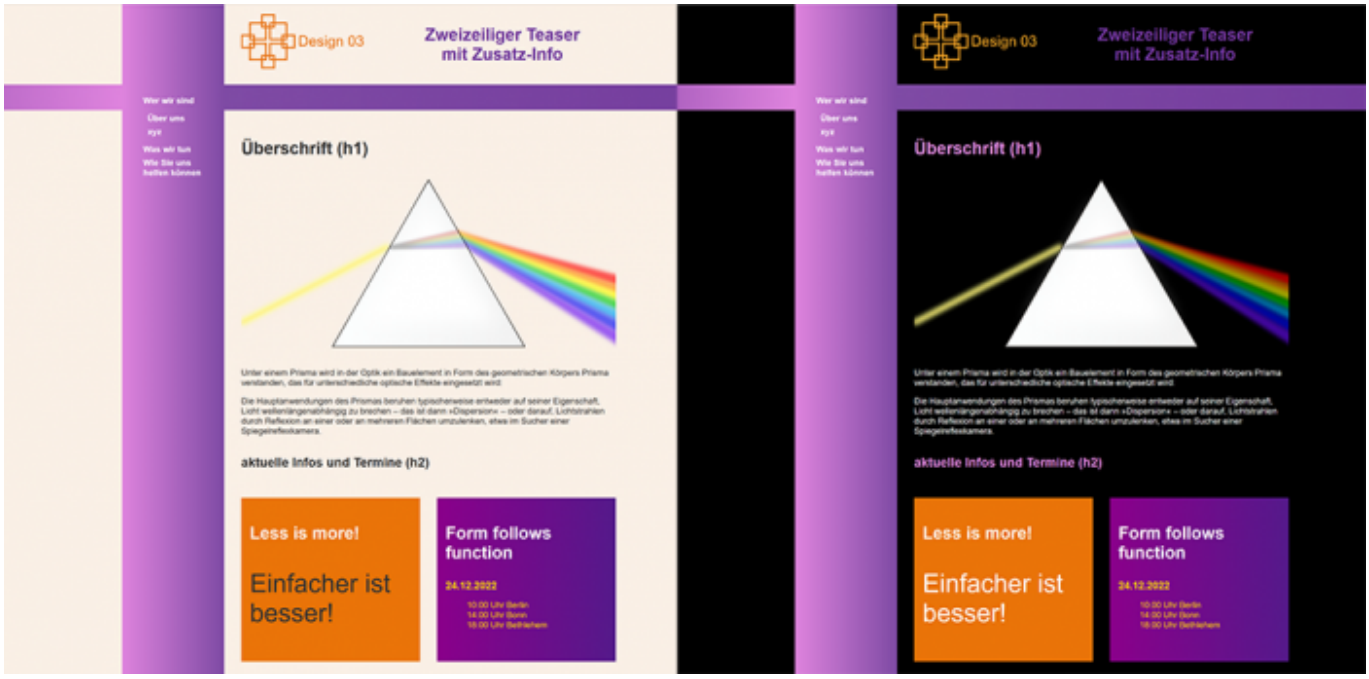
An diesem Punkt setzen die Cascading Stylesheets ein. Es handelt sich dabei um eine unmittelbare Ergänzungssprache, die vorwiegend für HTML (aber auch für SVG) entwickelt wurde. Sie klinkt sich als progressive enhancement (englisch für fortlaufende Verbesserung) nahtlos in HTML ein und erlaubt das beliebige Formatieren einzelner HTML-Elemente. Mit Hilfe von Stylesheets können Sie beispielsweise festlegen, dass alle Überschriften 24 Punkt groß sind und mit einem Nachabstand von 16 Punkt und mit einer grünen doppelten Rahmenlinie oberhalb dargestellt werden. Schematisch würde dies etwa so aussehen:

```
Überschrift {
Schriftgröße: 24 Punkt;
Abstand-unten: 16 Punkt;
Rahmenfarbe-oben: grün;
Rahmenart-oben: doppelt
```

```
}
```

Diese und andere Definitionen können Sie aber genauso gut auch für einen beliebigen Text festlegen.

Die nächste Abbildung zeigt, wie die eben gezeigte HTML-Seite mit CSS aussehen könnte:




From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_01**

Last update: **2024/04/24 12:41**

# Einbindung von CSS-Styles

CSS-Eigenschaften können im wesentlichen auf 3 Arten mit HTML verwendet werden:

## Inline Styles

Mit dem Universalattribut style können Sie Eigenschaften einem Element direkt zuweisen. Dabei sind nur Deklarationen, also Eigenschaft-Wert-Zuweisungen möglich.

```
<p style="text-align: center; color: green;">
  Dieser Absatz wird über ein style-Attribut formatiert.
</p>
```

Dieses Beispiel zeigt, wie einem p-Element per style-Attribut verschiedene Formate zugewiesen werden können. Diese Festlegung gilt aber nicht für weitere Textabsätze.

**Achtung:** Durch das direkte Festlegen von Formaten, umgangsprachlich auch „Inline-Style“ genannt, gehen viele Vorteile verloren. Der Wartungsaufwand steigt während sich die Flexibilität verringert. Inline-Styles sind an ein Element gebunden und können nicht an zentraler Stelle bearbeitet werden.

## Interne Styles

Das HTML-Element style erlaubt es, Formate zentral im head des HTML-Dokuments festzulegen.

```
<!doctype html>
<html lang="de">
  <head>
    <style>

    h1 {
       background-color: green;
       color: white;
    }
    p {
        text-align: center;
        color: green;
    }

    </style>
  </head>
  <body>
  </body>
</html>
```

In diesem Beispiel wird das Format für das Überschriftenelement h1 und alle Textabsätze des

Dokuments innerhalb eines style-Elements festgelegt. Das style-Element ist ein Kindelement des head-Elements.

## Externe Styles

Im Normalfall besteht eine Webpräsenz aus mehreren (oft hunderten) von Seiten, die alle gleich formatiert werden sollen. Hier hat es sich durchgesetzt, das CSS in einem eigenen externen Stylesheet mit der Endung .css abzuspeichern und mit dem HTML-Element link direkt im head einzubinden.

```
Einbinden eines externen Stylesheets
<!doctype html>
<html lang="de">
  <head>
    <link rel="stylesheet" href="stylesheet.css">
    ...
```

Das link-Element ist ein Kindelement des head-Elements.

Folgende Attribute gibt es:

**rel** definiert den logischen Beziehungstyp des Elements. „stylesheet“ bedeutet, dass ein Stylesheet eingebunden werden soll. **href**: referenziert die einzubindende Stylesheetdatei

Diese beiden Attribute sind erforderlich und ausreichend, um ein Stylesheet einzubinden. Das link-Element kann aber noch weitere Attribute besitzen, die auf Stylesheets eine Wirkung haben.

## Ein Layout oder doch mehrere?

Früher wurden Webseiten oft für eine bestimmte Browserversion und eine festgelegte Bildschirmauflösung entwickelt. Da es zwischen den Browsern gravierende Darstellungsunterschiede gab, wurden mittels einer Browserweiche verschiedene Stylesheets geladen.

Heutzutage geht es eher darum, Webseiten auf allen verfügbaren Ausgabegeräten vom Smartphone bis zum Riesen-Monitor gleichermaßen lesbar zu machen. Wenn die Seitenbreite zur Darstellung der Inhalte nicht mehr ausreicht, werden sie mittels media queries automatisch in ein neues, passendes Design verschoben.

**Empfehlung**: Das Arbeiten mit CSS erlaubt es, mit Medienabfragen (Media Queries) für Ausgabemedien mit bestimmten Eigenschaften, etwa Tablets, Smartphones aber auch Drucker abweichende Darstellungen innerhalb eines Stylesheets festzulegen. Passen Sie Ihr Layout flexibel an.

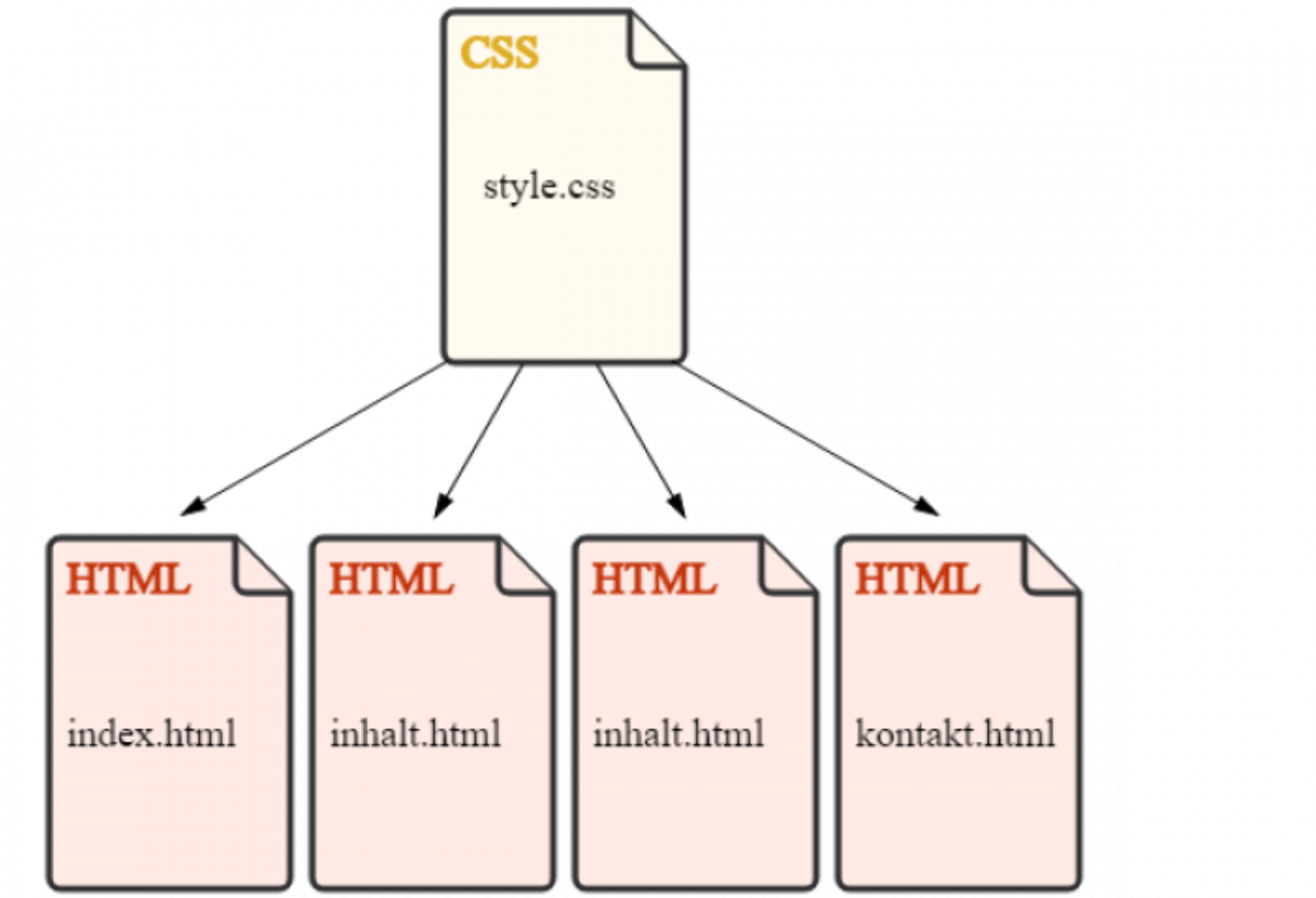
## zentrale Formate für mehrere HTML-Dateien

CSS erlaubt es, Stile, Farben und Formen zu definieren, beispielsweise für alle Überschriften, oder für alle Textabsätze mit einem bestimmten Klassennamen, oder für kursiv ausgezeichneten Text, der

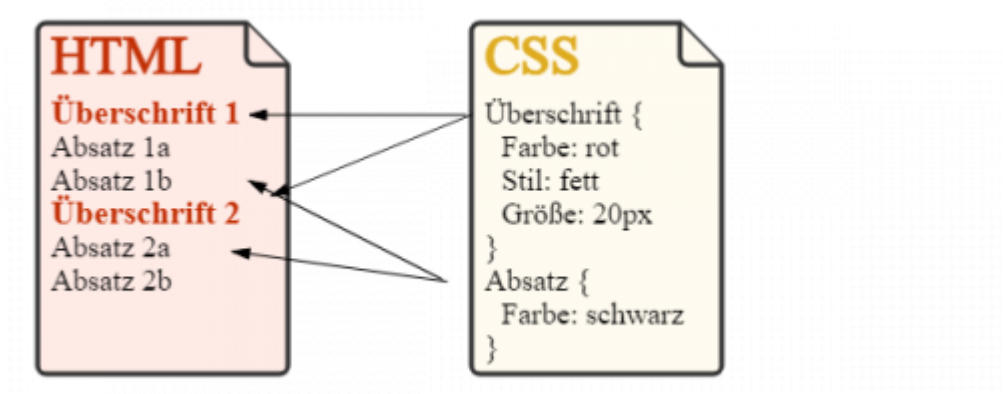
innerhalb einer Tabellenzelle vorkommt.

Die zentralen Formate können sich auf eine HTML-Datei beziehen, aber auch in eine externe Style-Datei ausgelagert werden, die man in beliebig viele Seiten einbinden kann. So werden einheitliche Formatvorgaben möglich, und der HTML-Code wird von unnötigem Ballast befreit. Spätere Änderungen am Design können so leicht durchgeführt werden.

In der folgenden Abbildung dient eine einzige CSS-Datei beispielsweise vier HTML-Dateien als Formatvorlage:




Die folgende Abbildung zeigt den Zusammenhang zwischen HTML und CSS:




Während in HTML der Inhalt semantisch ausgezeichnet wird, wird das Aussehen im Stylesheet festgelegt.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_02**

Last update: **2024/04/24 12:53**

# Syntax

CSS ist zwar keine Programmiersprache, sondern nur eine **Auszeichnungs- oder Formatsprache** für HTML und SVG. Trotzdem gibt es eine eigene Syntax - also ein festes Regelwerk, wie was notiert werden soll!

## Regelsatz und Deklaration

In CSS sind Eigenschaften innerhalb von Regelsätzen (auch Regeln) organisiert. Ein Regelsatz besteht aus:

- **Selektor**
- **geschweiften Klammer {}**
- **eine oder mehreren Deklarationen**, in denen **Eigenschaften ein Wert zugewiesen** wird

Eine Deklaration benötigt zwischen Eigenschaft und Wert einen Doppelpunkt. Zwischen zwei Deklarationen steht ein Semikolon.

Die Werte, die man zuweisen kann, sind sehr unterschiedlicher Art. Es gibt Eigenschaften, die einen einzelnen Wert erhalten können, und Sammeleigenschaften (shorthand), denen man eine Liste von Werten gibt, die dann auf bestimmte andere Eigenschaften verteilt werden. Werte können Zahlen sein, Längenangaben, Zeichenketten oder unterschiedlichste Schlüsselbegriffe. Das wird bei den Beschreibungen der Eigenschaften im Detail erklärt.


**Beachte:** Genau genommen benötigt die letzte Deklaration eines Regelsatzes kein Semikolon. Sie sollten es aber trotzdem setzen. Wenn Sie den Regelsatz später erweitern und das Semikolon vergessen, entsteht ein Syntaxfehler, der beide Deklarationen unwirksam macht.

Beispiel:

```
h1 {
  color: red;
}
h1, h2 {
  background-color: #eee;
  border-radius: .5em;
```

```
}
```

Der erste Regelsatz besteht aus einem Selektor h1, dem als Deklaration die Eigenschaft color mit dem (Farb)-Wert rot zugewiesen wird.

Farben können entweder mit Farbnamen (z.B. red) oder mit Zahlenwerten (RGB-Modell) angegeben werden (z.B.: #33a3aa).

Der zweite Regelsatz besteht aus einer Selektor-Liste mit zwei voneinander unabhängigen Selektoren h1 und h2, über die den Überschriften mit background-color eine Hintergrundfarbe und mit border-radius abgerundete Ecken zugewiesen werden.

Selektoren können HTML-Elemente, aber auch ganz spezielle Kombinationen sein.

## shorthand Eigenschaft

Werden einer **Eigenschaft mehrere Werte** zugewiesen, handelt es sich üblicherweise um eine so genannte „shorthand“-Eigenschaft. Dabei handelt es sich um Eigenschaften, die mehrere andere Eigenschaften zusammenfassen. Die Werte für die Einzeleigenschaften werden dabei durch Leerzeichen voneinander getrennt.

```
h1 {
  color: red;
  font: 300% cursive;
}

h1, h2 {
  background-color: #eee;
  border-radius: .5em;
  border: thin solid red;
}
```

Die Deklaration für die zusammenfassende Eigenschaft font hat einen Wert für die Schriftgröße (font-size) und durch ein Leerzeichen getrennt einen für den Schrift-Stil (die Einzeleigenschaft font-style).

Der zweite Regelsatz enthält eine Deklaration für die zusammenfassende Eigenschaft border, die den Selektoren eine Randlinie zuweist. Hier werden mehrere Werte für Breite (border-width), Stil (border-style) und Farbe (border-color) durch Leerzeichen voneinander getrennt.

## Mehrfaches Vorkommen einer Eigenschaft

Der Browser arbeitet die verschiedenen Deklarationen des Stylesheets nacheinander ab. Wenn eine Eigenschaft mehrfach definiert wird, so gilt nur der zuletzt akzeptierte gültige Wert. Werden in einer Regel Eigenschaften aufgeführt, die das verarbeitende Programm nicht kennt, so wird diese Eigenschaft ignoriert. Das gilt auch für unbekannte oder ungültige Werte.

```
h1 {
  color: red;
```

```
  color: verygreen;
  color: gren;
}
```

In diesem Beispiel wird für ein Element die Schriftfarbe rot definiert. Die nachfolgenden Wertzuweisungen werden ignoriert, denn der Farbname „verygreen“ ist keine in CSS gültige Farbangabe und auch der Wert „gren“ (statt „green“) wird nicht akzeptiert.

## Kommentare

Um die Übersicht zu behalten oder bestimmte Regelsätze zu erläutern, ist es möglich, innerhalb von Stylesheets Kommentare zu verfassen, die vom Browser bei der Anzeige des Dokuments ignoriert werden. Ein Kommentar beginnt mit den Zeichen. Das bedeutet, dass Kommentare nicht ineinander verschachtelt werden können.

```
/* Überschrift */
h1
{
  font-size: 18pt;
}
```

# CSS-Stil

HTML ist schon zu einem großen Teil barrierearm. Block-Elemente nehmen den verfügbaren Raum ein und brechen bei schmalen Viewports in die nächste Zeile um. In den Browser-Stylesheets gibt es schwarze Schrift auf weißem Grund. Man sollte darauf achten, keine zusätzlichen Barrieren durch benutzerunfreundliche CSS-Festlegungen zu errichten.

CSS ermöglicht eine individuelle Gestaltung Ihrer Webseite. Achten Sie dabei aber darauf, dass

- sie für alle Besucher benutzbar bleibt.
- Ihre Stylesheets übersichtlich organisiert sind und helfen, Fehler zu vermeiden, bzw. später einfach zu warten

**Beachte:** Trennung von Inhalt (HTML), Präsentation (CSS) und Verhalten (Javascript)! Auch wenn es oft einfacher scheint, einem Element ein style-Attribut zu geben, wird HTML-Markup so auf Dauer unübersichtlich und schwierig zu warten.

## Formatierung

Die grundsätzliche Formatierung ist bei CSS weitgehend frei, man kann also zwischen den einzelnen Bestandteilen Selektor, geschweiften Klammern, Eigenschaften, Doppelpunkt, Werten und Semikola beliebig Leerzeichen, Tabulatorzeichen und Zeilenumbrüche einfügen, z. B. um das Stylesheet übersichtlicher zu gestalten. So sind die zwei folgenden – inhaltlich identischen – Definitionen beide erlaubt:

```
/* einzeilig */
h1{font-size:2.5em;margin-left: 2em;text-align:center;border-bottom:medium solid red; margin-right:2em;}
```

versus

```
/* mehrzeilig*/
h1 {
  font-size: 2.5em;
  border-bottom: medium solid red;
  text-align: center;
  margin: 0 2em;
}
```

Der mehrzeilige Regelsatz wirkt viel übersichtlicher:

- Der Selektor ist gleich erkennbar
- Deklarationen sind eingerückt
- Eigenschaft und Wertzuweisung sind durch ein zusätzliches Leerzeichen getrennt.
- die beiden Deklarationen margin-left und margin-right wurden zu margin zusammengefasst.

Er erscheint zwar länger, moderne Code-Editoren können solche Regelsätze mittels Code-Folding jedoch ein- und ausklappen.

#### Selektor-Listen

```
a.backlink:hover,
a.backlink:focus {
  color: purple;
  background: #fffbf0 url("data:image/svg+xml,...") no-repeat right;
}
```

Hier werden im zweiten Regelsatz mehrere Selektoren in einer durch Komma separierten Selektor-Liste angesprochen. Der Regelsatz wird lesbarer, wenn die einzelnen Selektoren untereinander dargestellt werden.

#### Empfehlung:

- zwischen 2 Regelsätzen sollte eine Leerzeile stehen
- Ein Umbruch sollte
  - nach der öffnenden geschweiften Klammer
  - hinter jedem Semikolon
  - und bei Selektor-Listen erfolgen
- nach dem Doppelpunkt zwischen Eigenschaft und Wert sollte ein Leerzeichen sein

## Organisation

Man sollte sich eine Reihenfolge für die Deklarationen überlegen:

1. Farbpalette
2. Schrift
3. Positionierung
4. Display
5. Box-Modell
6. Farben, Hintergründe und weiteres Styling

Die Deklarationen werden im Seiteninspektor in der festgelegten Reihenfolge aufgeführt, was auch zum Debuggen praktischer ist. Dabei ist anzumerken, dass bei modernen Mobile First-Layouts oft keine Positionierung mehr nötig ist.

Für größere Viewports kann man dann mit Medienabfrage weitere Festlegungen treffen.

# Benutzereinstellungen

Während media queries in den letzten Jahren vorwiegend für die Ermittlung der Breakpoints für die optimierte Darstellung auf unterschiedlich großen Bildschirmen verwendet wurden, kamen in den letzten Jahren neue Medienabfragen hinzu, die vorher getroffene Benutzereinstellungen (White or Black-Design?) abfragen.

### Schriftgröße

Viele Nutzer haben in den Einstellungen des Betriebssystems und Ihres Browsers bestimmte Schriftgrößen festgelegt.

- Nutzen Sie relative Schriftgrößen in em oder rem!
- Verzichten Sie auf feste Schriftgrößen in px oder womöglich sogar pt!

**Empfehlung:**

- keine festen Werte für Breite
- keine festen Werte für die Höhe eines Elements, sondern lasse dem Inhalt den nötigen Platz. Abstände mithilfe von padding!
- relative Längenmaße wie em, die sich an geänderte Schriftgrößen anpassen

### Kontraste

Verwenden Sie eine ausreichende Schriftgröße und hohe Kontraste.

Geben Sie zur verwendeten Hintergrundfarbe jeweils immer auch eine kontrastreiche Textfarbe an. Wenn ein background-image definiert wird, sollte auch die background-color definiert werden, denn ein Bild könnte fehlen.

Nutzen Sie die Möglichkeit der Medienabfragen:

- prefers-color-scheme fragt ab, ob der Nutzer ein Dark Mode oder helles Farbschema bevorzugt.
- prefers-contrast emittelt, ob Inhalte mit höherem (oder niedrigeren) Kontrast dargestellt werden soll.

```
@media (prefers-color-scheme: dark) {
  /* dunkles Farbschema für die Nacht */
  body {
    color: white;
    background: black;
  }

  a {
    color: skyblue;
  }
}

@media (prefers-color-scheme: light) {
  /* helles Farbschema für den Tag */
}
```

#### Animationen

Bei Animationen sollte der Nutzer immer die volle Kontrolle über seinen Bildschirm haben. Entweder will er keine Animationen sehen oder eine bestimmte Animation erneut abspielen. Obwohl so etwas nur mit JavaScript erreicht werden kann, können zumnindest die Voreinstellungen abgefragt werden:

- prefers-reduced-motion (bevorzugt weniger Bewegung) erkennt, ob der Nutzer eine Seite mit allen Animationen und Bewegungen sehen oder eine ruhigere Darstellung wünscht.

## Fokus

Wann immer Sie die Pseudoklasse :hover einsetzen, verwenden Sie auch die Pseudoklasse :focus um Tastaturbenutzern beim Durchtabben dasselbe Feedback zu geben.

```
a:hover,
a:focus {
  background: skyblue;
  text-decoration: underline double navy;
}
```

Entfernen Sie nicht den gepunkteten Rahmen, den die Browser um fokussierte Links oder Buttons zeichnen.

Zeigen Sie für Sehende versteckte aber fokussierbare Elemente an, wenn sie den Fokus erhalten.

## Normalisierung

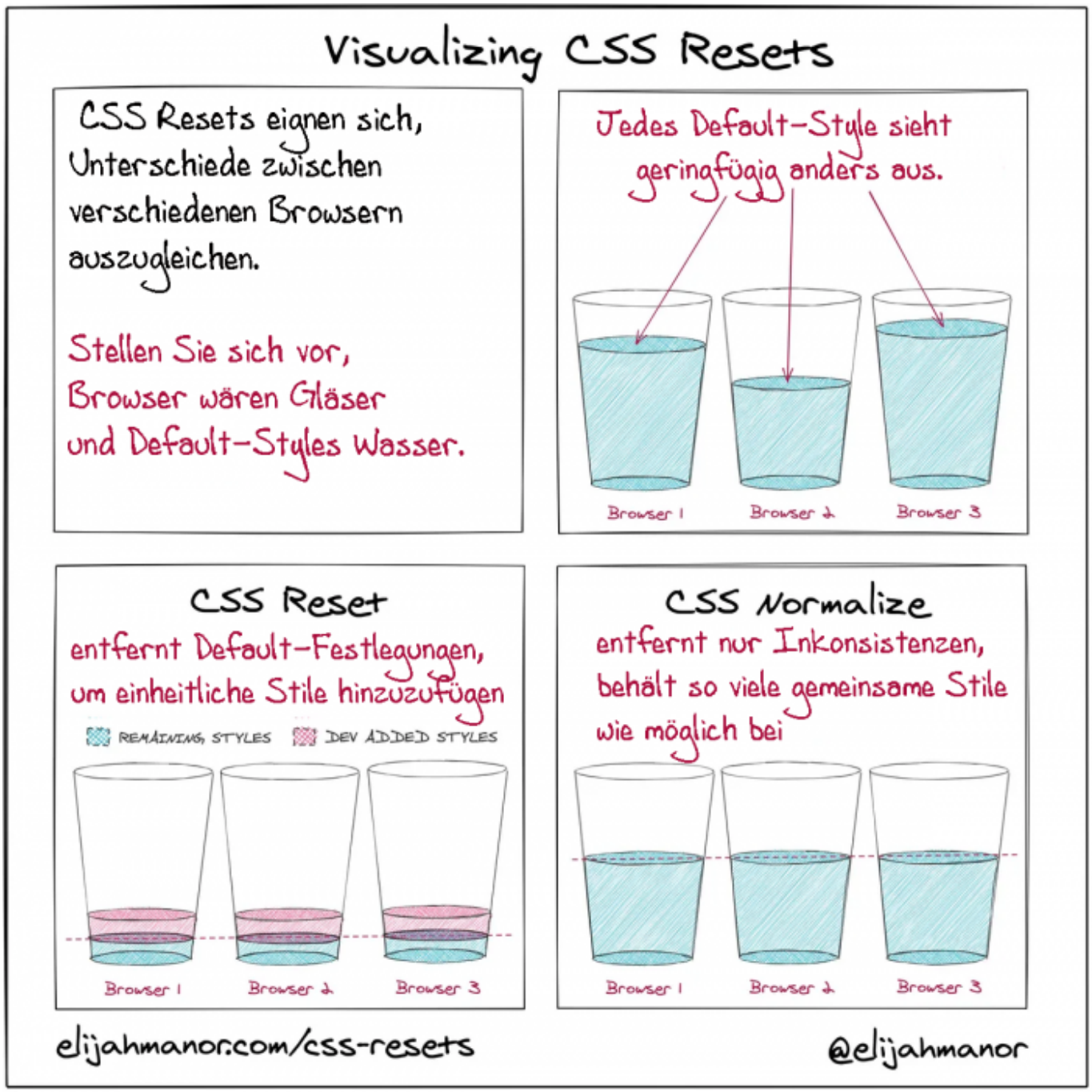
Browser wenden ihr eigenes Browser-Stylesheet an, bevor sie das Autoren-CSS anwenden. Die **Default-Regeln** können aber von **Browser zu Browser abweichen**.

Beispielsweise wird die Einrückung von Listen durch die Browser unterschiedlich gehandhabt. Die einen realisieren dies über Innenabstände des ul- bzw. ol-Elementes, die anderen über Außenabstände der li.

Deshalb empfehlen manche Webdesigner eine Normalisierung, bei der die CSS-Einstellungen der Browser, vor allem Abstände, Rahmen oder Schrifteinstellungen, zurückgesetzt und somit vereinheitlicht werden. Sie ist vor allem für folgende Elemente verbreitet: ul, ol, li, dl, dt, dd.

```
Beispiel
ul, ol, li, dl, dt, dd {
  display: block;
  padding: 0;
  margin: 0;
}

li {
  display: list-item;
  margin-left: 2em;
}
```

**Empfehlung:** Vermeiden Sie weitreichende Normalisierungen, alle Default-Angaben sind sinnvoll und die meisten sind browserübergreifend sehr ähnlich.




# Vorlagen und Frameworks

Grundsätzlich sollte man vermeiden, mit jedem Projekt wieder bei null anzufangen. Vieles benötigt man in jedem Projekt. Das beginnt bei so profanen Dingen wie Ordnern für Bilder oder CSS, geht über das HTML-Grundgerüst und die Verknüpfung der HTML-Datei mit einer Formatvorlage und kann je nach Arbeitsumgebung bis hin zu Serverkonfigurationen per htaccess, der Einbindung von JavaScript-Bibliotheken und ganzen Frameworks gehen.

Um das alles nicht bei jedem Projekt neu anlegen zu müssen, arbeiten viele Entwickler mit selbst erstellten Vorlagen, die sie für neue Projekte nutzen.

Es gibt aber auch Projekte im Web, die solche Vorlagen kostenlos bereitstellen. Das wohl bekannteste ist die HTML5-Boilerplate von Paul Irish. Der Gedanke hinter so einer Boilerplate (englisch für

Kochplatte) ist es, nicht jedes Mal einen Herd bauen zu müssen, wenn man sich ein Spiegelei braten möchte.

Die HTML5-Boilerplate bringt all die genannten Dinge mit. Darüberhinaus sind sinnvolle und meist nötige CSS-Styles bereits in der Datei main.css vorhanden, die leicht an eigene Bedürfnisse angepasst werden können.

Für Anfänger lohnt sich ein Blick auf die Boilerplate auch weil man daran eine typische und sinnvolle Struktur nachvollziehen kann.

## CSS Frameworks

Noch weiter gehen CSS-Frameworks zu deren bekanntesten Vertretern Bootstrap, Skeleton und Foundation zählen. Diese bringen fertige Styles für die meisten Komponenten einer Webseite mit. Dazu gehören auch komplexere Konstrukte wie Menüs, Slider oder Tabs (Karteikartenreiter). Alles responsiv und sofort nutzbar. Insbesondere bei Bootstrap sind zudem gute Grundlagen für die Entwicklung zugänglicher Webseiten vorhanden.

**Empfehlung:** Empfehlung: Binden Sie Bootstrap vor Ihren eigenen CSS-Festlegungen ein. So werden Ihre Einstellungen nicht wieder überschrieben!

#### Kritik an Frameworks

Hiermit erstellte Webseiten sehen naturgemäß den vielen Millionen Seiten, die ebenfalls auf den Frameworks aufbauen, sehr ähnlich. Wenn man die Formatvorlagen weitgehend an ein vorgegebenes Layout anpassen möchte, beschreiben viele Entwickler den nötigen Aufwand als genauso hoch oder höher wie bei einer kompletten Eigenentwicklung.

Darüberhinaus wird oft die mangelnde Trennung von Inhalt und Layout kritisiert, die aufgrund der massenhaft eingesetzten präsentationsbezogenen Klassen nicht aufrechtzuhalten ist. Wie in den Zeiten vor CSS muss an alle Elemente geschrieben werden, wie diese aussehen sollen – insbesondere Änderungen an statischen Seiten werden so extrem aufwändig und sind fehleranfällig.

Nicht zuletzt steht mit CSS-Grid eine Technik zur Verfügung, die leicht zu nutzen ist und Teile eines solchen Frameworks (insbesondere die zum Seitenlayout) überflüssig macht.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_03**

Last update: **2024/04/27 20:05**

# Responsive Webdesign

Unter responsivem Webdesign versteht man, dass das Layout einer Webseite für das Ausgabemedium angepasst wird. Übersetzt wird das englische Adjektiv „responsive“ etwa mit „ansprechbar“, „empfänglich“, „zugänglich“ oder „antwortend“, was auf den ersten Blick nicht sehr viel mit dem Internet zu tun hat.

Brad Frost zeigt in Responsive Web Design: Missing the Point[1], dass es nicht darum geht, neben einer Desktop-Version noch eine oder mehrere mobile Layoutvarianten zu erstellen, sondern Seiten zu erstellen, die möglichst geräteunabhängig schnell laden und überall gut aussehen.


Webdesign bedeutet

- eine Fläche zu gestalten ohne deren Format zu kennen
- Text zu layouten, ohne wirklichen Einfluss auf Schriftart, Größe und Zeilenfluss zu besitzen
- Farben einzusetzen, die sich kaum genauer als ungefähr definieren lassen.

Webdesign schafft eine flexible Konstruktion, eine dehnbare Architektur, die sich in unterschiedliche Räume einfügt, technische Besonderheiten berücksichtigt und Spielraum lässt für individuelle Gegebenheiten. Webdesign ist immer Gast auf fremden Bildschirmen.

## Mobile first

Da die meisten Webseiten schon in irgendeiner Form vorhanden sind, liegt es nahe, dieses Layout an

bestimmten Breakpoints mittels Media Queries an kleinere Viewports anzupassen. Dabei stand man vor dem Problem, dass Dekorationen und Grafiken nicht auf die kleineren Monitore passen. Trotz der Bandbreitenbegrenzungen vieler Mobilfunk-Verträge wurden diese aber mitübertragen und geladen, dann aber ausgeblendet.

Luke Wroblewski hatte in seinem Buch „**Mobile First!**“ die Idee, den Prozess umzudrehen und mit einer abgespeckten, mobilen Version zu beginnen. Dies sollte aber kein Selbstzweck sein, sondern den Fokus wieder auf die ältere, aber immer noch gültige Maxime „**Inhalt über Design**“ und andere Usability-Grundsätze legen.

Mobile First baut auf folgenden Tatsachen auf:

- Inhalte im Layout für kleine Bildschirme werden meist untereinander dargestellt.
- Dies ist auch das Verhalten eines jeden Browsers bei einem HTML-Dokument ohne jegliches CSS.
- Die Inhalte für einen großen Viewport werden oft nebeneinander angeordnet.

Es ist also zweckmäßig, als erstes die allgemein gültigen Definitionen und die für kleine Bildschirme ins CSS zu schreiben und dann erst in einer Media Query das CSS für große Bildschirme. Ansonsten bräuchte man entweder zwei komplett unterschiedliche Stylesheets oder würde erst alles für große Bildschirme in Spalten anordnen und dies dann für Geräte mit kleinem Bildschirm alles wieder rückgängig machen.

Mobile First sollte folglich nicht bedeuten, dass mobile Endgeräte bevorzugt werden, sondern, dass das Layout für mobile Geräte aus praktischen Gründen vor den Anpassungen für den Desktop in der CSS-Datei liegt.

## Device-agnostic

Agnostizismus (altgriechisch ἀγνοεῖν a-gnoein „Unwissen“) ist die philosophische Ansicht, dass Annahmen – insbesondere theologische, die die Existenz oder Nichtexistenz einer höheren Instanz, beispielsweise eines Gottes, betreffen – entweder ungeklärt oder grundsätzlich nicht klärbar sind.

Trent Walton bevorzugt anstelle des Begriffs Responsive Webdesign den des Device-Agnostic. Damit will er betonen, dass Webdesigner nicht ahnen können, welche Geräte mit welchen Fähigkeiten Benutzer einsetzen werden und daher möglichst geräteunabhängig gearbeitet werden soll. Dabei geht seiner Meinung nach die Betonung weg von der Responsiviät mittels Media queries und hin zu Techniken, die geringe Übertragungsraten in mobilen Netzen und Besonderheiten der Toucheingabe genauso berücksichtigen.

# Tastaturen, Mäuse und Touchgeräte

Viele JavaScript-Interaktionen und CSS-Hovereffekte funktionieren auf Touchgeräten nicht, da es kein mouseover und kein :hover gibt. Zum Teil versuchen die Browser dieses Verhalten nachzuahmen, besser ist es jedoch bereits bei der Programmierung Touch-Events zu implementieren.

Webseiten im Fullscreen-Modus können mit ESC wieder in die klassische Ansicht zurückgesetzt

werden, nur nicht auf Touch-Geräten, da es dort keine Tasten gibt. Auch der Klammergriff ( strg + alt + entf ) ist auf Touchgeräten nicht möglich.

# Performance Optimierung

Scott Jehl legte in seinem Artikel „**Planning for Performance**“ das Augenmerk auf die sich immer weiter steigernde Datenmengen, die heutige Webseiten ausliefern. Während die durchschnittliche Webseite 1,7 MB groß ist, haben manche responsive Seiten wie disney.com, die für jeden Browser die passenden Frameworks, Grafiken und Skripte ausliefern, über 5 MB, teilweise sogar über 100 MB!

Gerade angesichts der Tatsache, dass Benutzer mobiler Geräte nicht nur über WLAN, sondern auch über 3G surfen, sollten Webseiten nicht nur responsiv, sondern auch schlank sein. Das Surfen im Mobilnetz bewirkt eine zeitliche Verzögerung von 2 Sekunden beim Seitenaufbau, da der Datenstrom vom Server zusätzlich noch den Funksender des Mobilfunknetzes passieren muss.

Auch wenn LTE und 4G diese Antwortzeiten verringert haben, ist mobiles Surfen an sich langsamer als Surfen mit WLAN oder in Kabelnetzen. Deshalb sollten Sie versuchen, die Anzahl der http-Requests so gering wie möglich zu halten. Normalerweise blocken externe CSS-Dateien und JS-Skripte im Unterschied zu Grafiken das Rendering einer Seite. Durch das Nachladen von Stylesheets mit Hilfe von Skripten und nachträglichem Einfügen mittels document.write wird der schon gerenderte Seiteninhalt wieder verändert.

**Empfehlung**: Vermeiden Sie ein Laden der Seite, bevor die Stylesheets geladen sind. Dieses **FOUC (Flash of unstyled Content)** ist extrem ärgerlich und verzögert den Seitenaufbau.

# HTML

Das Grundgerüst sollte auf jeden Fall die Sprache, den Zeichensatz und auch den Viewport beinhalten:

```
<!--responsives HTML-Grundgerüst-->

<!doctype html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Titel</title>
  </head>
  <body>
  </body>
</html>
```

Dabei ist besonders die letzte Meta-Angabe wichtig, damit sich Webseiten einerseits an die verfügbare Breite anpassen, aber auch skalieren lassen.

Wenn Sie alle Seiten in Ihrem Code-Editor wie z. B. Notepad++ öffnen, können Sie die ersten fünf Zeilen durch „Suchen und Ersetzen“ in allen Dateien einfügen. Eine Angabe von utf-8 ist aber nur sinnvoll, wenn die Seiten dann auch in diesem Format gespeichert werden.

# CSS

Häufig reichen einige wichtige Änderungen am CSS, um eine Webseite zumindest großteils responsiv zu gestalten. Webseiten aus den 90ern mit einem Liquid Layout passten sich auch damals schon an verfügbare Breiten an, ließen in Tabellen und mehrspaltigen Layouts längere Wörter aber über den sichtbaren Rand überstehen.

## Pixel ist ein No-Go!

Das Haupthindernis herkömmlicher Seiten ist der Versuch eines pixelgenauen Designs. Durch unterschiedliche Darstellungen auf verschiedenen Plattformen, sowie durch Retina-Displays ist dieser Ansatz von vornherein zum Scheitern verurteilt.

*Diese Website ist optimiert für eine Auflösung von 800×600 und den Internet Explorer 6.*

Kennen Sie diesen Hinweis aus der Anfangszeit des Webs noch? Machen Sie nicht den Fehler und versuchen Sie nun neben Ihrer Desktop-Version eine oder mehrere Versionen für bestimmte Viewport-Größen zu erstellen. Brad Frost zeigt, dass es mittlerweile unmöglich ist vorherzusagen, mit welchen Endgeräten Nutzer heute oder gar morgen Ihre Webseite besuchen werden.

Unser Ausgangsbeispiel ist eine aside-Box innerhalb einer kleinen Webseite, wie sie auch im HTML-Einstieg vorgestellt wird:

```
#kontakt {
  border: solid 1px red;
  float: right;
  margin-right: 10px;
  height: 125px;
  width: 145px;
}
<aside id="kontakt">
  <h2>Kontakt</h2>
  <p>Telefon: 01234 4711</p>
  <p>7 - 17 Uhr, sonst AB</p>
</aside>
```

Die aside-Box ist pixelgenau abgemessen, sodass Überschrift und zwei Zeilen Text exakt hineinpassen. (Im Normalfall hätte man den Absätzen noch etwas padding gegeben.) Sie wird mit float:right; rechts positioniert (Eine bessere Alternative zeigen wir Ihnen im nächsten Abschnitt.)

Bei einer Änderung der Schriftgröße durch den Entwickler müssten die Breite und Höhe angepasst werden. Bei der Änderung der Schriftgröße durch den Nutzer im Browser oder durch ein User-Stylesheet würden die Textzeilen umbrechen und aus dem Rahmen - bei overflow:hidden; sogar aus dem sichtbaren Bereich wandern.

## Schriftgröße

*Was soll ich denn dann anstelle von Pixel nehmen?*

Ein guter Ausgangspunkt für responsive Webseiten ist die aktuelle Schriftgröße als Basisgröße. 1em repräsentiert die vom Browser berechnete Schriftgröße des Elements; es geht dabei kaskadierend von der Schriftgröße des Elternelements aus. Sofern vom Benutzer nicht anders eingestellt, beträgt die eingestellte Schriftgröße in den meisten Browsern 16px, der genaue Wert ist aber egal, da wir ja flexibel und responsiv werden wollen.

### feste Schriftgrößen

In vielen älteren Webseiten wurde die Schriftgröße bewusst auf einen kleinen Wert festgelegt, damit mehr Text ins Layout passt. Auf mobilen Geräten ist dies jedoch viel zu klein.

```
html {
  font-size: 9px;
}
```

Verzichten Sie im Normalfall auf die Festlegung einer festen Schriftgröße. Verwenden Sie statt dessen em (bezieht sich auf die Schriftgröße des Elternelements) oder, falls sich die Größe auf das Root-Element beziehen soll, rem.

## relative Maßeinheiten

Diese Schriftgröße kann (und sollte) auch als Basis für Breitenangaben von Elementen verwendet werden:

```
#kontakt {
  border: solid thin red;
  float: right;
  margin-right: 1em;
  width: 9em;
}
```

Die Breite ist in der relativen Maßeinheit em angegeben. Bei einer Änderung der Schriftgröße wächst die Breite der aside-Box einfach mit.

Eine Angabe einer Höhe ist im Allgemeinen nicht notwendig. Bei der Verwendung von flexiblen und prozentualen Breiten ist es sogar kontraproduktiv: Wenn der Inhalt auf schmalen Viewports zusammengeschoben wird, muss er nach unten ausweichen können und nicht über den Rand herausragen oder gar verschluckt werden.

Um bei vielen verschachtelten Elementen mit verschiedenen Festlegungen nicht durcheinanderzukommen, können Sie auch die Einheit rem (root em = Wurzel-em) verwenden, die sich auf die Schriftgröße des root-Elements html bezieht.

### empfohlene Vorgehensweise

Um mit wenigen Handgriffen schon sichtbare Ergebnisse zu erzielen, durchforsten Sie ihr Stylesheet:

Verwende:

- Schriftgrößen (und Maße) in relativen Größen wie em, damit eine größere Schriftgröße nicht das Design zerschießt
- prozentuale Breiten oder relative Angaben in em oder rem, aber keine festen Pixelwerte
- keine Höhenangaben für Elemente

Verzichte:

- Positionierung mit position: absolute oder fixed, da diese sich bei unterschiedlichen Viewports zwangsläufig verschieben.
- mit float, außer für Bilder innerhalb von Fließtext.

## Festlegungen für unterschiedliche Viewports

CSS bietet mehrere Möglichkeiten durch eine browserinterne Eigenschaftsabfrage unterschiedliche Darstellungen zu erreichen.

### media queries

Die heutzutage üblichste Variante ist der Einsatz von media queries (engl. für Medienabfragen), mit denen dann unterschiedliche Festlegungen für unterschiedliche Breiten, aber auch für andere Kriterien, getroffen werden können.

```
/* Kompaktes Layout für mobile Geräte */
#kontakt {
  border: solid thin red;
  float: right;
  margin-right: 1em;
}

@media (min-width: 30em) {
  /* mehrspaltiges Layout für breitere Bildschirme */
  #kontakt {
```

```
    float: right;
    width: 9em;
  }
}
```

Die aside-Box nimmt auf kleinen Bildschirmen die volle Breite des Viewports ein. Folgende Inhalte werden darunter dargestellt. Als Breakpoint wird eine Breite von 30em genommen.

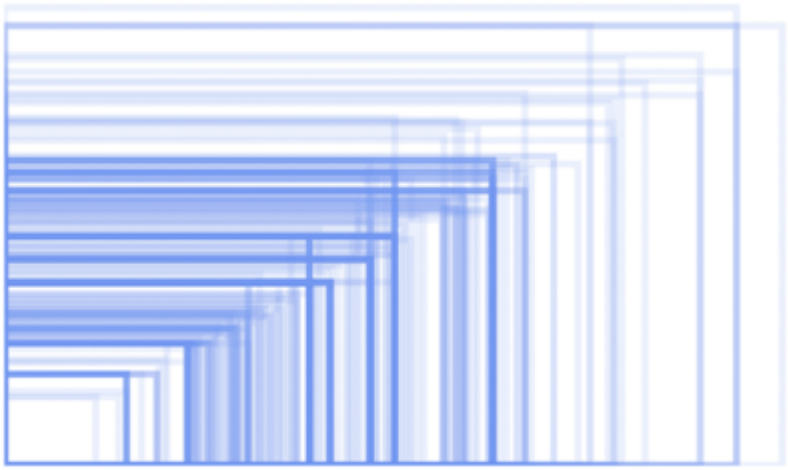
*Wo soll ich denn meine Breakpoints setzen? Welche Maße haben die meistverwendeten Smartphones?*

Erkennen Sie den schon oben angesprochenen Denkfehler?

Es ist heutzutage unmöglich, genaue Werte für Geräte-Abmessungen zu ermitteln, da eine Vielzahl dieser Geräte am Markt ist. SmartPhones werben mit 2000 (Geräte)Pixeln, Phablets lassen die Grenzen zwischen Phones und Tablets immer weiter verschwimmen. Hochauflösende 4K-Monitore sollen nicht für die mögliche Anzeige von mehreren DIN A4 Seiten in Mini-Schrift verwendet werden.

Die Breakpoints sollen sich nicht auf einige Endgeräte, sondern **am Inhalt ausrichten**.

- Eine **Box** von 9em Breite ist auf einer breiten Seite ok - wenn der Hauptinhaltsbereich daneben aber nur 5em breit ist, sieht die Seite komisch aus. Deshalb soll der Hauptinhalt mindestens 20em Breite betragen. So wird der Breakpoint bei 30em gesetzt.
- Bei einer **Navigation** wäre ein Breakpoint sinnvoll, bei dem alle Navigationselemente neben- anstatt untereinander dargestellt werden können. Dies kann bei 30em, aber auch bei einer anderen Breite passieren.



## grid-Layout

Mit Grid-Layout (siehe 7.4.8) können Sie Elemente beliebig in einem responsiven Raster anordnen.

Das Geniale dabei ist der mögliche Verzicht auf jegliche Breitenangaben:

```
body {
    display: grid;
        gap: 0.5em;
}
```

```
@media (min-width: 30em) {
    /* Breite beträgt mindestens 30em */
    body {
        grid-template-columns: 1fr 3fr;
        grid-template-rows: auto 1fr 100px;
    }
}
```

Unsere Beispielseite enthält im Body ein Grid (=Raster), das aber noch nicht weiter definiert ist. Erst ab einer Breite von 30em entstehen mit grid-template-columns nun zwei Spalten, die sich im Verhältnis 1 zu 3 aufteilen (fr steht für fraction, engl. für Bruchteil.

## Große Bildschirme

Ein Aspekt wird bei responsiven Webdesign oft vergessen: Neben vielen kleinen Viewports auf mobilen Geräten werden die Monitore auf dem Schreibtisch immer größer.

Textzeilen, die sich über 2550 oder sogar 4000 Pixel Breite erstrecken, sind nur noch schwer lesbar. Als Empfehlung hat sich eine Reduzierung der maximalen Seitenbreite auf ca. 60em durchgesetzt:

```
body {
  max-width: 60em;
  margin: 1em auto;
}
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_04**

Last update: **2024/05/06 06:03**

# Vom Entwurf zum Layout

Dieses Tutorial erklärt, wie du Webseiten so gestaltest, dass sie flexibel und responsiv auf jedem Gerät passend dargestellt werden.

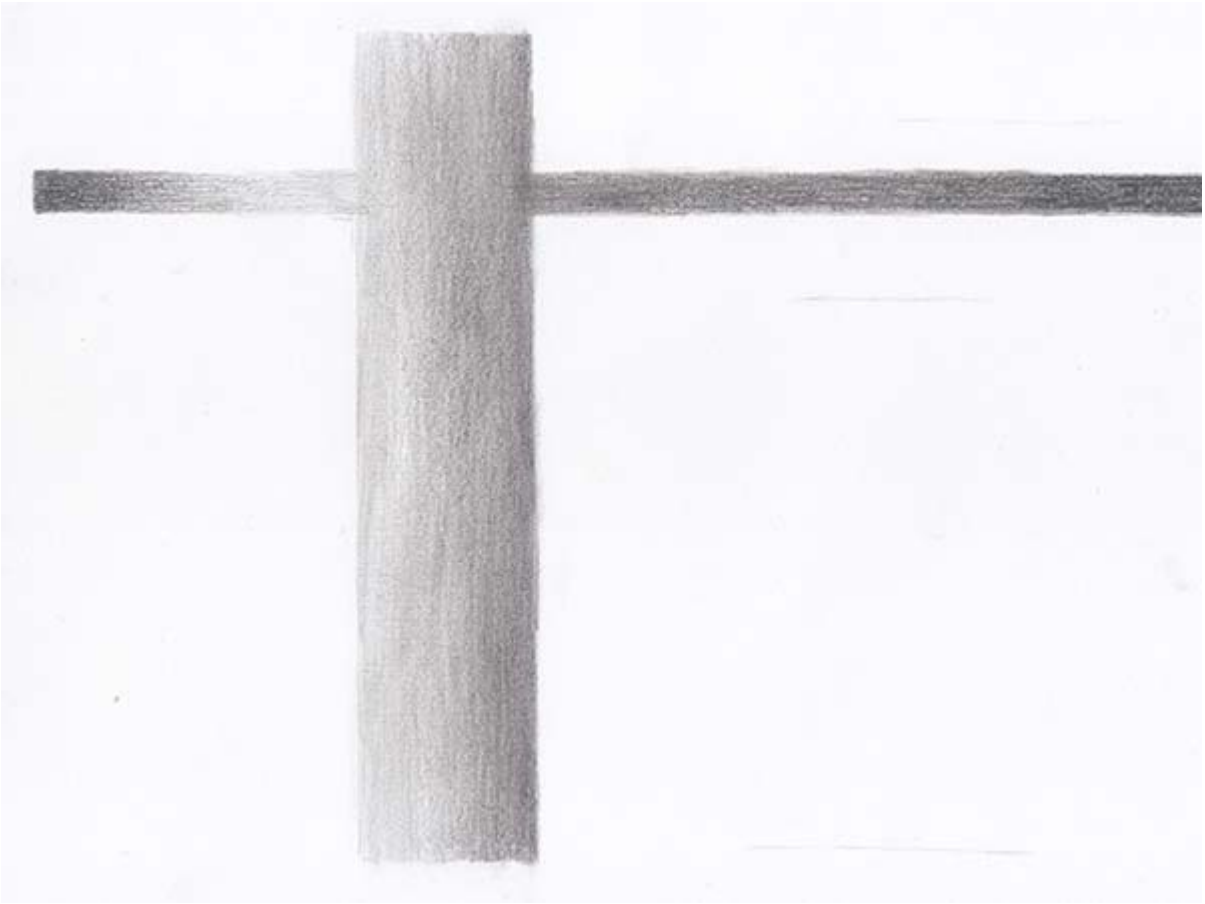
Dabei wird Wert darauf gelegt, dass CSS-Festlegungen nach den Regeln der „best practice“ so organisiert und notiert werden, dass sie von dir und anderen leichter verstanden und dann später auch aktualisiert werden können.

**Beachte:** Nicht ohne Grund gibt es die Schlagworte **„Content is King!“** und **„Form follows function“**. Eine Webseite können Sie erst gestalten, wenn Sie Inhalte und Struktur haben. Ein wireframe ist eine Hilfe, aber kein vollwertiger Ersatz!

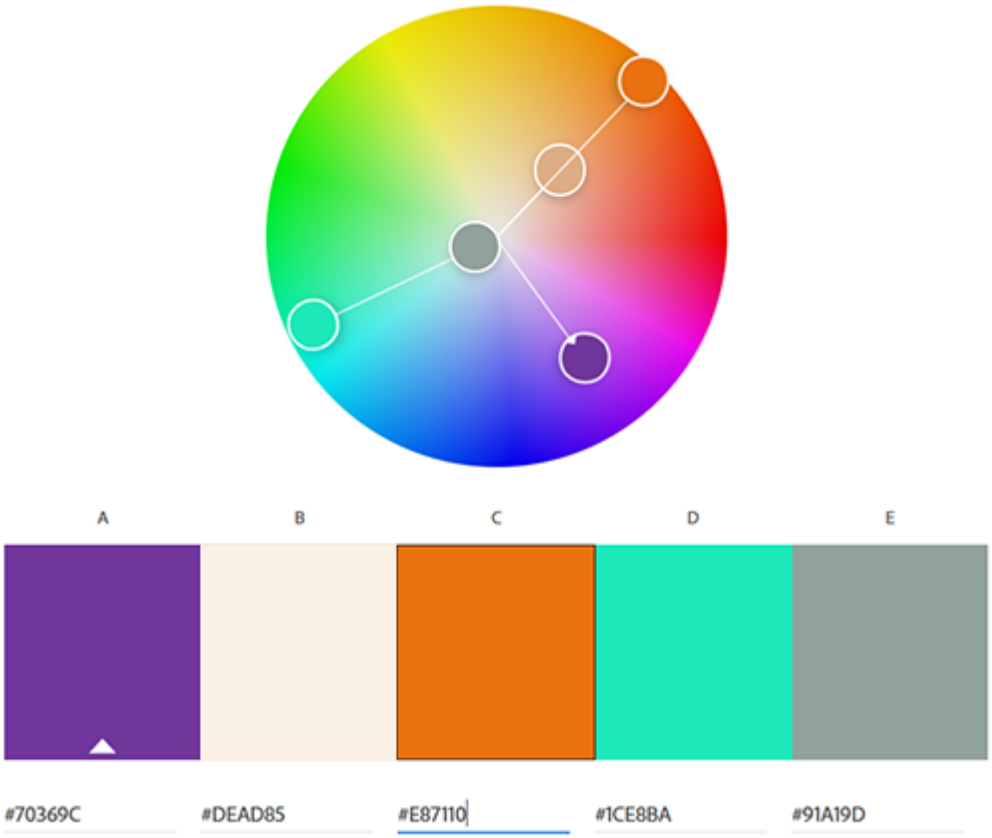
In den ersten Kapiteln wurde CSS eher theoretisch beleuchtet- nun wollen wir aus einem Entwurf mit CSS ein modernes responsives Layout geben.

## Wireframe

Wireframe ist ein konzeptioneller Entwurf ohne Gestaltung und Funktion



Die Farbpalette richtet sich an einer lila Grundfarbe aus, die durch die Komplementärfarben orange und grün als Akzente ergänzt werden.




## Mobile first

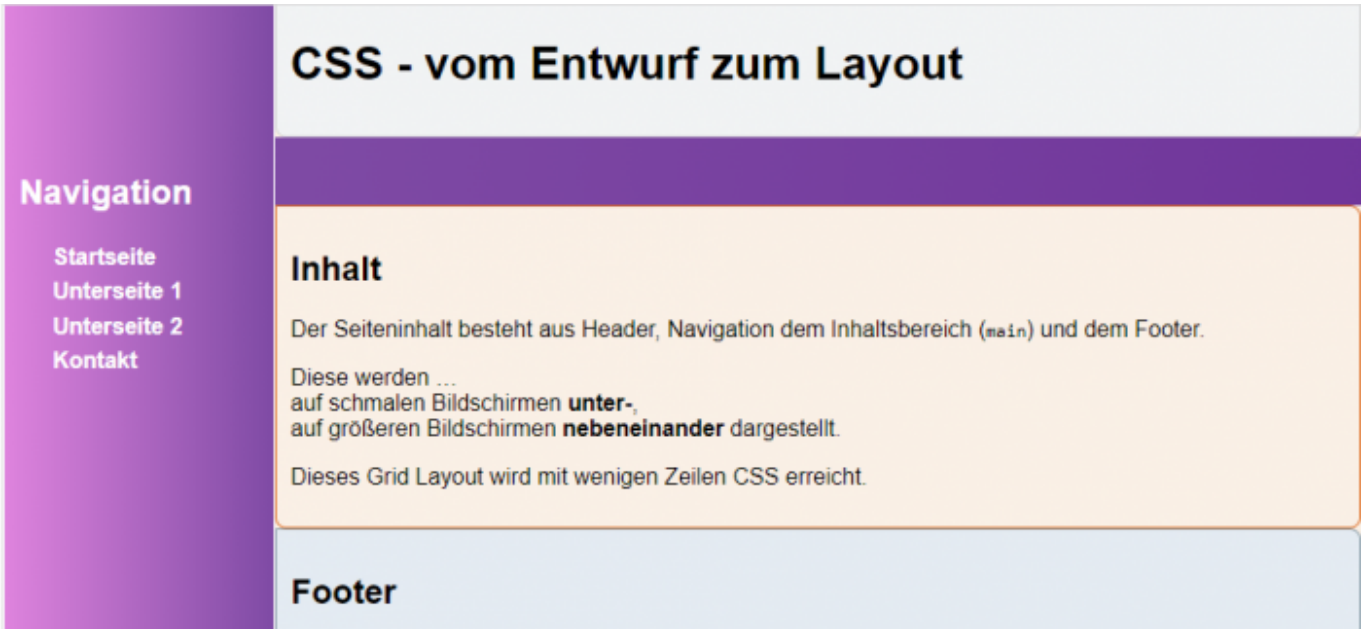
Webseiten sollen responsiv sein und sich flexibel an den verfügbaren Platz anpassen. Dabei müssen die Inhalte nicht gleich aussehen, sondern gut lesbar sein - welcher Nutzer vergleicht eine Seite gleichzeitig auf mehreren Geräten?

Was vielen nicht bewusst ist: HTML an sich ist **responsiv** - Elemente wie Überschriften und Absätze nehmen die gesamte Breite des Viewports ein und brechen überstehenden Text um, so dass er nach unten in einer neuen Zeile dargestellt wird. Erst eine starre Mehrspaltigkeit verhindert die passende Darstellung auf mobilen Geräten.

Deshalb werden in einer ersten Version nur die Farben und Schriftarten festgelegt.

```
:root {
    --accent1: #70369b;
    --accentmedium1: #7f4ba5;
    --accentlighter1: #df84de;
    --accent2: #e87110;
    --accentlighter2: orange;
    --accent3: #1ee7b9;
    --backgroundColor: linen;
    --textColor: #333;
    --navColor: #fff;

    --navWidth: 12em;
```

```
    --mainWidth: min(48em, calc(100% - var(--navWidth)));
    --headerLineHeight: 3em;
}

body {
    background: var(--backgroundColor);
    color:      var(--textColor);
    font-family: Avenir, Arial, sans-serif;
    min-height: 100vh;
    margin: 0;
    padding: 0;
}
header {
    padding: 2em;
}
nav {
    background: linear-gradient(90deg, var(--accentlighter1), var(--
accentmedium1)) ;
    color: var(--navColor);
    padding: 1em;
}
...
```




Dieses CSS besteht aus zwei Abschnitten:

1. Die in der Entwurfsphase ausgewählte Farbpalette wird mit **custom properties** für den **root-Selektor** festgelegt.
    1. *custom properties* - oft auch als CSS-Variablen bezeichnet, sind selbstdeklarierte Eigenschaften, die man am vorangestellten –... erkennt. Diesem Eigenschaftsnamen wird dann ein Wert zugewiesen:
        1. die Farbwerte der oben erwähnten Farbpalette
        2. –navWidth, –mainWidth und –headerLineHeight, die zentral Abstände festlegen
2. body und einige Seitenstrukturierungselemente erhalten über den jeweiligen Typselektor
    1. Text- und Hintergrundfarbe - Dabei werden die oben deklarierten custom properties über

    die var()-Funktion wieder aufgerufen.
    2. sowie unterschiedliche Innenabstände.
3. Auf Breiten- und Höhenangaben wird verzichtet!
    1. Alle Elemente werden in der vollen Breite untereinander angeordnet
    2. Die Höhe richtet sich nach dem vorhandenen Inhalt.

Dieser Ansatz erleichtert auch das (spätere) Anlegen eines Print-Stylesheets für eine Druckausgabe.

## Breakpoints

Eine Stärke von CSS liegt darin, die Darstellung eines Dokuments mit Hilfe von Medienabfragen (media queries) je nach Ausgabemedium verschieden festlegen zu können. So können Sie im Druckbereich von der normalen Bildschirmansicht abweichende Formatierungen festlegen.

Ihre volle Stärke entfalten Media Queries mit Hilfe von Medienmerkmalen, mit denen Sie z.B. unterschiedliche Viewport-Breiten berücksichtigen können.

Für welche Geräte muss ich wann Breakpoints setzen?

So oder ähnlich lauten viele Fragen im Forum. Die Antwort darauf lautet: „Gar nicht!“

erzeit aktuelle Geräteauflösungen können morgen schon überholt sein.

- Richte die Breakpoints am Inhalt aus.
- Überlege, ab wann dieser bei einer bestimmten Viewportbreite nicht mehr gut aussieht.
    - Bei Text ist es eigentlich egal, höchstens ein *max-width: 60em;*, damit die Zeilen nicht zu lang werden.
    - Bei Grafiken und bestimmten Seitenelementen sollten Sie flexible, prozentuale Breiten verwenden, damit Bilder
        - auf schmalen Bildschirmen 100% breit dargestellt werden
        - auf Tablets in landscape-Modus nebeneinander oder
        - auf breiten Bildschirmen drei oder vier nebeneinander dargestellt werden.

```
body {
    display: grid;
    gap: 1em;
}
@media (min-width: 30em) {
  body {
    grid-template:  "nav head" min-content
                    "nav main" auto
                    "nav foot" min-content /
                    12em 1fr;
    }
}
```

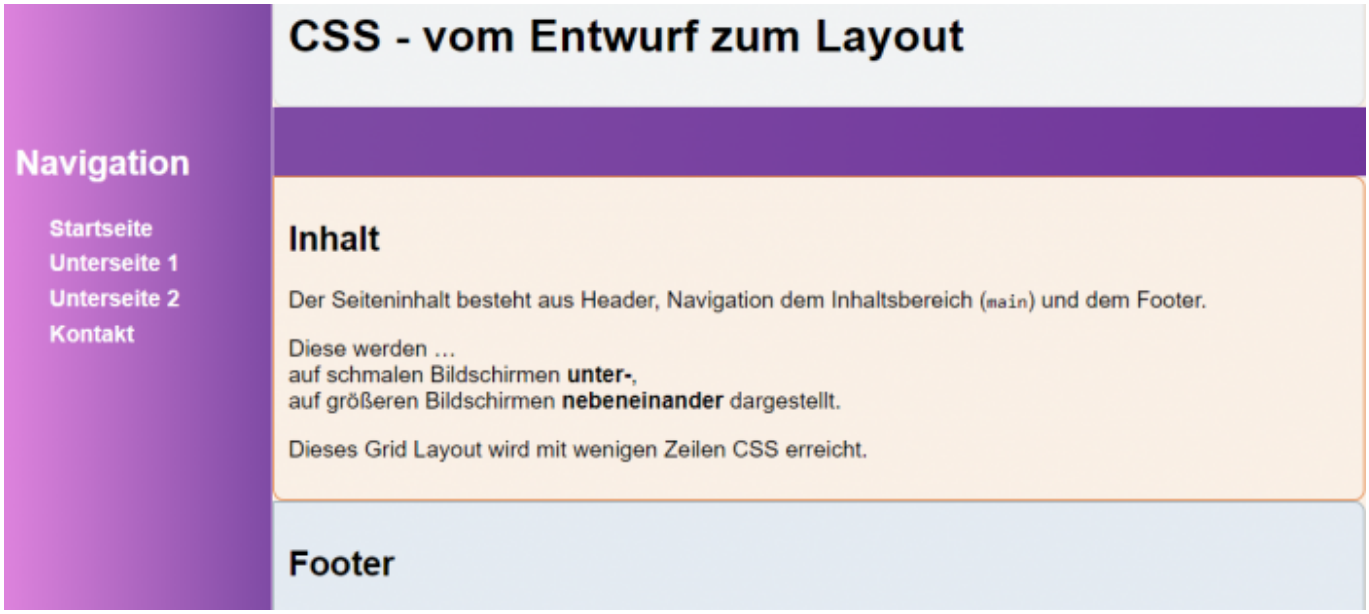
Im Grundzustand Mobile First! werden alle Seitenelemente untereinander angeordnet. Bei einer Viewportbreite von mindestens 30em soll die Navigation links angeordnet werden. Dies erreichen wir, indem wir grid-template-columns eine ASCII-Map unseres Seitenaufbaus geben. Rechts stehen die jeweiligen Höhenangaben, die nicht in festen Pixelwerten sondern mit min-content angegeben

werden. So wird sichergestellt, dass alle Inhalte immer angezeigt werden!

Die Breitenangaben unten sind mit 12em Breite für die Navigation vorher abgemessen, der Inhaltsbereich verteilt sich auf den Rest.

In unserer Zeichnung sollte sich die Navigation aber kreuzartig über die Seite zu erstrecken. Ein solcher Querbalken kann eingefügt werden - da er nur zur Dekoration dient, aber nicht im HTML, sondern nur im CSS:

```
:root {
    --navWidth: 12em;
    --mainWidth: min(48em, calc(100% - var(--navWidth)));
    --headerLineHeight: 3em;
}

@media (min-width: 30em) {
  body {
    display: grid;
    grid-template:  ". nav head ." min-content
                    ". nav rule ." var(--headerLineHeight)
                    ". nav main ." auto
                    ". nav foot ." min-content /
                1fr var(--navWidth) var(--mainWidth) 1fr;
    }
    body::before {
        content: '';
        grid-row: rule;
        grid-column: 1 / 1;
        background: linear-gradient(to right, var(--accent1), var(--
accentlighter1));
    }
    body::after {
        content: '';
        grid-row: rule;
        grid-column: rule-start / -1;
        background-image: linear-gradient(to right, var(--accentmedium1),
var(--accent1));
    }
}
@media (min-width: 40em) {
  header {
    display: grid;
    grid-template-columns: 1fr 2fr;
    padding: 1em 1em 1em 2em;
    gap: 2em;
    justify-items: center;
  }
}
```




Dieses Beispiel folgt dem Grundsatz „Mobile first!“ und enthält nach den oben beschriebenen Farbfestlegungen zwei media queries:

- Auf kleinen Viewports nehmen alle Blockelemente ohne weitere Festlegungen die volle Breite ein.
- Ab einer Mindestbreite von 30em trifft die Medienabfrage @media (min-width: 30em) zu und es wird ein Raster angelegt:
  - Mit grid-template werden innerhalb der Anführungszeichen 2×4 Rasterbereiche festgelegt, denen als zusätzlicher Wert die Höhe beigefügt wird.
    - Das nav-Element nimmt alle (drei) Zellen der ersten Spalte ein; header, main und footer kommen in die rechte Spalte
    - Allerdings ist das Template vierspaltig: 1fr var(–navWidth) var(–mainWidth) 1fr; links und rechts der beiden Inhaltsspalten werden jeweils 1fr notiert, um den Inhaltsbereich zu zentrieren.
  - Die zweite Reihe „. nav rule .“ var(–headerLineHeight) legt nun einen Rasterbereich rule (engl. für Maß) fest. Dieser findet sich nicht im HTML.
    - Im Regelsatz body::before{…} wird nun ein Pseudoelement erzeugt, dass diesen rule-Bereich mit der Dekorationsgrafik füllt. Anstelle einer Rastergrafik im jpeg-Format wird der Verlauf allein mit CSS erzeugt.
- Ab 40em Breite ist die zweite Spalte so breit, dass Logo und Teaser nebeneinander stehen können. Deshalb erhält der header ein weiteres Grid, das seine Kindelemente img und span in Rasterzellen verwandelt.

## Dark Mode

Mit Media Queries können Sie nicht nur Geräteabmessungen, sondern auch Benutzereinstellungen abfragen und ihr Layout entsprechend anpassen. So wünschen viele Benutzer einen Dark Mode, der sich mit wenigen Zeilen CSS realisieren lässt:

```
@media (prefers-color-scheme: dark) {
  /* dunkles Farbschema für die Nacht */
  body {
    --textColor: white;
```

```
    --backgroundColor: black;
  }

  a {
    color: skyblue;
  }
  h1,h2,h3 {
    color: var(--accentlighter1);
  }
}
```

Mit prefers-color-scheme wird abgefragt, ob ein dunkles Schema gewünscht wird. Falls dies der Fall ist, werden die entsprechenden custom properties neu gesetzt.


HTML-Dokument mit CSS-Stylesheet für Light und Dark Mode

# Fine-Tuning

## Überschriften

Eines der besten Mittel einer Webseite ihren „eigenen“ Look zu geben ist die Verwendung einer speziellen Schriftart.

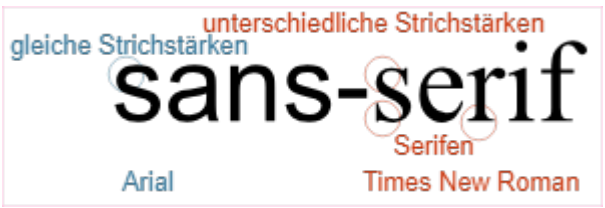
```
<link
href="https://fonts.googleapis.com/css2?family=Gloria+Hallelujah&display=swap" rel="stylesheet">
Über das link-Element wird die gewünschte Schriftart eingebunden.

Alternativ kann man das auch zentral im Stylesheet mit einer @import-Regel erledigen:
```

```
@import
url('https://fonts.googleapis.com/css2?family=Gloria+Hallelujah&display=swap
');

h1,h2,h3 {
    color: var(--accent1);
    font-family: 'Gloria Hallelujah', cursive;
    text-align:center;
}

// Besser: fonts selbst hosten:

@font-face {
    font-family: 'Cantarell';
    font-style: normal;
    font-weight: 700;
    src: local('Cantarell Bold'), local('Cantarell-Bold'),
url(../font/Cantarell-Bold.ttf) format('truetype');
}
```


Allen Überschriften von h1 bis h3 wird mit font-family die Schriftart Gloria Hallelujah zugewiesen. Falls die nicht geladen werden kann, wird als Fallback die Schriftfamilie cursive angegeben.

Über @font-face können Sie die URL des selbst gehosteten Fonts angegeben.

**Beachte:** So verlockend es scheint, Schriften mit einer Zeile Code von einem CDN zu laden, sollten Sie Fonts immer selbst hosten.




Dabei muss aber stets die Lesbarkeit der Texte im Auge behalten werden. Viele Zierschriften mit

Serifen benötigen eine gewisse Schriftgröße, um sie zufriedenstellend anzuzeigen. Deshalb empfiehlt man oft, für Fließtext die Standardschrift des Browsers zu nehmen und die eigene Schriftart nur für Überschriften und wichtige Zitate.

# Größen und Abstände

Wie oben bereits erwähnt, ist es empfehlenswert, sich generell auf die Grundeinstellungen des Browser-Stylesheet zu verlassen. Allerdings muss man bei besonderen Schriften darauf achten, dass Unterlängen nicht in den nachfolgenden (Text)-Inhalt ragen.


Hier können Sie mit line-height und margin-bottom entgegensteuern.

## Einheiten

### Absolute Einheiten

Ein Pixel (px) bezeichnet man hauptsächlich deshalb als absolute Einheit, weil es festgelegt ist und sich nicht abhängig von den Abmessungen eines anderen Elements verändert. Wenn Sie einen Wert Ihres Stylesheets ändern, müssen Sie entsprechend alle anderen (absoluten) Werte ebenfalls manuell anpassen.

### Relative Einheiten

Relative Einheiten wie %, em oder rem sind für ein responsives Design besser geeignet. Hauptsächlich deshalb, weil sie im Stande sind, mit unterschiedlichen Bildschirmgrößen zu skalieren.

- **em** - Relativ zur font-size des Elements selbst (Dezimalwert)
- **rem** - Relativ zur Schrifthöhe (font-size) des html-Elements (Dezimalwert)
- **%** - Relativ zu einer Längenangabe Elternelement.
- **vw** - Relativ zur Breite des Viewports (Prozentwert)
- **vh** - Relativ zur Höhe des Viewports (Prozentwert)

Jeder Browser legt eine Defaultgröße für Text fest, zumeist font-size: 16px. Die Einheit rem verwendet diesen Wert als Rechenbasis. Wenn ein Anwender die Standardtextgröße in seinem Browser verändert, skaliert alles auf der Seite passend zur Basisgröße. Maße, die Sie in rem angeben, werden

mit dieser Größe multipliziert.

Zum Beispiel:

```
.8rem = 12.8px  (0,8 * 16)
1rem  = 16px    (1 * 16)
2rem  = 32px    (2 * 16)
```

Was ist, wenn man die Defaultschriftgröße ändert? Wie schon gesagt, dies sind relative Einheiten und die endgültigen Größenwerte werden dann aus der neuen Basisgröße ermittelt. Das ist in Media Queries nützlich; man muss nur die Schriftgröße verändern und die ganze Seite skaliert dementsprechend hoch oder herunter.

```
html {
  font-size: 10px;
}
```

```
 .8rem =  8px (0,8 * 10)
  1rem = 10px (1   * 10)
  2rem = 20px (2   * 10)
```

Mit Prozentangaben lässt sich der Wert einer Eigenschaft basierend auf dem Wert einer Elterneigenschaft festlegen. Welcher das ist, ist nicht immer ganz logisch, und muss im Zweifelsfall bei der festzulegenden Eigenschaft nachgeschlagen werden.

**Beachte**: Alle genannten Einheiten, außer Prozent, sind Längeneinheiten und lassen sich überall dort verwenden, wo eine CSS-Eigenschaft eine Länge erwartet. Die Angabe eines Prozentwertes an Stelle einer Länge ist an vielen Stellen zulässig, aber nicht überall. Beispielsweise ist die Angabe einer Randdicke (border-width) nicht in Prozent möglich.

```
html {
  font-size: 16px;
}
p {
   font-size: 200%;    /* Bezieht sich auf die font-size des Elternelements
*/
   width: 50%;         /* Bezieht sich auf die Breite des Elternelements */
   padding-top: 5%;    /* Bezieht sich ebenfalls auf die Breite des
Elternelements! */
}
```

Diese CSS-Regel stellt p-Elemente mit einer Schrift dar, die doppelt so groß wie die Schrift im Elternelement ist. Wenn sonst nirgends Schriften festgelegt wurden, wären das im Beispiel 32px. Die Breite der p-Elemente wäre die Hälfte der Breite des Elternelements (z. B. des body).

Und was ist der Unterschied zwischen den Einheiten **rem** und **em**?

Er besteht darin, was die Einheit als Basiselement verwendet. **rem** berechet Werte basierend auf der **font-size** des **<html>**-Elements, während ein Element, das **em**-Werte spezifiziert, seine eigene **font-size** verwendet. Ein Sonderfall ist dabei die Angabe der eigenen **font-size** in **em**, dieser Wert bezieht sich auf vom Elternelement geerbte **font-size**. Wenn sich die **font-size** des Elternelements von der

im **<html>** Element unterscheidet, ermitteln **rem** und **em** unterschiedliche Werte. Damit erhalten wir eine feinere Kontrolle darüber, wie sich unsere Elemente in unterschiedlichen responsiven Kontexten verhalten.

**vh** ist eine Abkürzung für *viewport height*. Der Viewport ist der Teil des Browserfensters, der die Webseite darstellt. **100vh** ist die vollständige Höhe des darstellbaren Bereichs (also des Browserfensters abzüglich Rändern, Titelleiste, Adresszeile, Lesezeichenzeile und ggf. weiterer Gimmicks). Dementsprechend ist **100vw** (viewport width) die vollständige Breite dieses Bereichs.

Anmerkung des Übersetzers: Leider ist die CSS-Spezifikation hier nicht ganz exakt und es gibt Unklarheiten, wie Scrollbars zu behandeln sind und was passiert, wenn ein Mobilgerät-Browser beim Scrollen die Adresszeile wegblendet oder eine Tastatur einblendet. Ursprünglich hat sich **vh** an solche Ein- oder Ausblendungen angepasst, mit unangenehmen Folgen für die Darstellung der Seite. Deswegen wurde das in Safari für iOS und Chrome für Android geändert, **vh** bleibt dort nun konstant. In Firefox für Android ist das (Stand September 2020) ein sechs Jahre altes, offenes Ticket.

## Minimum und Maximum

Die Vergleichsfunktionen **min()** und **max()** dienen dazu, den kleinsten beziehungsweise größten Wert aus einer Liste von Werten auszuwählen. Damit kann man erreichen, dass der berechnete Wert für eine Eigenschaft einen bestimmten Wert nicht überschreitet oder unterschreitet. Es ist allerdings nicht ganz intuitiv: um für eine Eigenschaft einen Maximalwert festzulegen, muss man die min() Funktion verwenden – weil ja in dem Moment, wo der berechnete Wert das gewünschte Maximum überschreitet, dieses Maximum der kleinere der beiden Werte ist.

```
img {
    width: min(100%, 400px);
  }
```

Die Grafik ist responsiv und nimmt 100% der Containerbreite ein, bis zu einem Maximalwert von 400px. Um dies zu veranschaulichen wird die Breite des body-Element durch eine CSS-Animation verändert.

## clamp ()

Es kommt auch vor, dass man eine Unter- und eine Obergrenze festlegen möchte. Dazu kann man min und max kombinieren, z. B. so: width: max(300px, min(50%, 800px) );. Die Breite des Elements wird auf 50% des Elternelements festgelegt, aber mindestens 300px und nicht mehr als 800px. Um eine solche Angabe lesbarer zu gestalten, gibt es die Funktion clamp():

```
.box {
   width: clamp(400px, 50%, 800px);   /* 50% Breite, mindestens 400px und
höchstens 800px*/
}
```

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_05**

Last update: **2024/05/09 06:31**

# Kaskade

Ein besonderes Merkmal von **CSS** ist die **Kaskade**, das **C** in **CSS** steht für **cascading** (englisch für kaskadierend). Ein Dokument kann nicht nur von einem einzelnen Stylesheet formatiert werden, sondern von einer Vielzahl von Stylesheets, die aus verschiedenen Quellen stammen können und aufeinander aufbauen.

Gleichzeitig entsteht aber das Problem, dass Eigenschaften für ein Element mehrfach und mit widersprüchlichen Werten festgelegt werden können. Dieses Problem wird umgangen, indem die Kaskade für Regeln und Eigenschaften eine Gewichtung oder Spezifität errechnet, anhand derer die tatsächlich für ein Element geltenden Formate bestimmt werden.

## Kriterien

### Herkunft

Ein Stylesheet kann aus drei Quellen stammen:

- Browser-Stylesheet: Browser besitzen vorgegebene Formatierungen für HTML-Dokumente. Dieses Stylesheet hat die niedrigste Priorität und bildet die Basis für jedes HTML Dokument.
- Benutzer-Stylesheet: Benutzer können in den Einstellungen ihres Browser bevorzugte Schriftarten oder Farbmischungen festlegen. Benutzer-Stylesheets haben mittlere Priorität und überschreiben widersprechende Eigenschaften im Browser-Stylesheet.
- Autoren-Stylesheet: Diese sind vom Dokument selbst eingebunden oder mit dem Dokument mitgeliefert. Autoren-Stylesheets haben hohe Priorität und überschreiben Browser- und Benutzer-Stylesheets.

**Beachte**: Formate in Browser- und Benutzer-Stylesheets können von den Werten abweichen, die man üblicherweise vorfindet oder die in Webstandards genannt werden. Spezifische Einstellungen des Betriebssystems wie „hoher Kontrast“ oder „dunkles Thema“ können hier Auswirkungen zeigen. Gehe also nicht davon aus, dass die Hintergrundfarbe immer weiß und die Schriftfarbe immer schwarz ist, oder dass 1em gleich 16px ist.

### Reihenfolge

Wird eine Eigenschaft oder ein Regelsatz mehrfach deklariert, so wird die zuletzt deklarierte Eigenschaft verwendet.

```
h1 { color: red; }
h1 {
 color: orange;
 color: green;
}
```

In diesem Beispiel wird die **color**-Eigenschaft drei mal innerhalb von zwei Regelsätzen deklariert. Der Wert orange im zweiten Regelsatz überschreibt dabei den Wert red aus dem ersten Regelsatz. orange

wird jedoch schon in der nächsten Zeile mit dem Wert green überschrieben, da diese Deklaration als letztes notiert wurde.

## Spezifität

In CSS werden Regelsätze nach ihrer Spezifität gewichtet. Allgemeine Regelsätze besitzen dabei eine geringere Gewichtung als spezifische Regelsätze.

Die Spezifität wird durch die Betrachtung der Bestandteile eines Selektors ermittelt. Für jeden Selektor werden drei Zähler (A, B und C) mit dem Startwert Null festgelegt. Jeder Bestandteil eines Selektors wirkt sich auf diese Zähler aus:

- Der Zähler A wird durch jedes Vorkommen eines ID-Selektors um eins erhöht.
- Der Zähler B wird durch jedes Vorkommen eines Attribut- oder Klassenselektors bzw. einer Pseudoklasse um eins erhöht.
- Der Zähler C wird durch jedes Vorkommen eines Typselektors oder eines Pseudoelements um eins erhöht.
- Der Universalselektor verhält sich neutral, er wird ignoriert.
- Die Pseudoklasse :not() selbst wird ignoriert, die Selektoren innerhalb der Klasse werden jedoch wie vorbeschrieben gewertet.

```
*              /* A=0, B=0, C=0, Spezifität 0 0 0 */
h1             /* A=0, B=0, C=1, Spezifität 0 0 1 */
ul li          /* A=0, B=0, C=2, Spezifität 0 0 2 */
a::after       /* A=0, B=0, C=2, Spezifität 0 0 2 */
p:first-child  /* A=0, B=1, C=1, Spezifität 0 1 1 */
a:not([href])  /* A=0, B=1, C=1, Spezifität 0 1 1 */
ul.nav [href]  /* A=0, B=2, C=1, Spezifität 0 2 1 */
#author        /* A=1, B=0, C=0, Spezifität 1 0 0 */
#editor p      /* A=1, B=0, C=1, Spezifität 1 0 1 */
```

**Beachte:** Die Zähler der Spezifität sind voneinander unabhängig. Ein hoher Wert des Zählers C wirkt sich nicht auf den Zähler B aus (z. B. ist 0 0 15 weniger spezifisch als 0 1 5).

### Spezifität des style-Attributs

Werden Eigenschaften in einem style-Attribut festgelegt, so ist diese Eigenschaft spezifischer als jeder Regelsatz in einem Stylesheet (Regeln innerhalb von style-Attributen können nur durch die Verwendung von !important überschrieben werden). Die Errechnung der Spezifität erfolgt mit Hilfe eines vierten Zählers (hier A), der sich um eins erhöht, wenn ein style-Attribut gesetzt wurde.

```
*              /* A=0, B=0, C=0, D=0, Spezifität 0 0 0 0 */
h1             /* A=0, B=0, C=0, D=1, Spezifität 0 0 0 1 */
ul li          /* A=0, B=0, C=0, D=2, Spezifität 0 0 0 2 */
a::after       /* A=0, B=0, C=0, D=2, Spezifität 0 0 0 2 */
p:first-child  /* A=0, B=0, C=1, D=1, Spezifität 0 0 1 1 */
a:not([href])  /* A=0, B=0, C=1, D=1, Spezifität 0 0 1 1 */
ul.nav [href]  /* A=0, B=0, C=2, D=1, Spezifität 0 0 2 1 */
#author        /* A=0, B=1, C=0, D=0, Spezifität 0 1 0 0 */
```

```
#editor p       /* A=0, B=1, C=0, D=1, Spezifität 0 1 0 1 */
style=""        /* A=1, B=0, C=0, D=0, Spezifität 1 0 0 0 */
```

**!important**

Wird eine Eigenschaft mehrfach deklariert, so gilt in der Regel der zuletzt festgelegte Wert. Da dies auch unabsichtlich geschehen kann, existiert die Möglichkeit, einen bestimmten Wert als wichtig zu kennzeichnen. In diesem Fall wird der Wert einer Eigenschaft nicht durch eine nachfolgende Deklaration mit identischer Gewichtung überschrieben.

Ein Wert wird als wichtig gekennzeichnet, indem nach dem Wert selbst optional ein Leerzeichen, ein Ausrufezeichen (!) sowie das Schlüsselwort important notiert werden.

```
h1 { color: green !important; }
h1.main { color: red; }
```

In diesem Beispiel wurde der Wert green als wichtig gekennzeichnet. Aus diesem Grund zeigt die nachfolgende Deklaration, die aufgrund des Klassenselektors eigentlich eine höhere Spezifität hätte, keine Auswirkung. Das bedeutet, dass die Schriftfarbe grün dargestellt wird.

Wird eine shorthand-Eigenschaft als wichtig gekennzeichnet, so gilt die Kennzeichnung für alle Untereigenschaften der Kurzschreibweise

```
border-left: medium double black !important;
/* ... ist identisch zu: */
border-left-width: medium !important;
border-left-style: double !important;
border-left-color: black !important;
```

**Beachte:** Verwende !important nur in Ausnahmefällen. Auch wenn man nur wenige !important einsetzt und damit die normale Reihenfolge von Regeln und Deklarationen aussetzt, erschwert es das Zurechtfinden im Stylesheet.

# Ablauf der Kaskade

Der Ablauf der Kaskade besteht aus mehreren Schritten, in denen die einzelnen Kriterien angewandt werden und zur Aussortierung von Deklarationen führen.

Im ersten Schritt werden alle Deklarationen gesucht, die unter Berücksichtigung des aktuellen Ausgabemediums auf ein Element angewendet werden. Anschließend werden diese Deklarationen nach Herkunft und Wichtigkeit sortiert:

- Deklarationen im Browser-Stylesheet
- Deklarationen des Benutzers
- Deklarationen des Autors
- animations
- !important-Deklarationen des Autors
- !important-Deklarationen des Benutzers

- !important-Deklarationen im Browser-Stylesheet
- transitions

Diese Sortierung soll sicherstellen, dass Autoren größtmöglichen Gestaltungspielraum haben, aber für den Benutzer wichtige Formate, z. B. die Mindestschriftgröße immer mit Vorrang behandelt werden. Die nun übrigen Regelsätze werden nach ihrer Spezifität geordnet. Regelsätze mit hoher Spezifität überschreiben die mit geringerer Spezifität. Sind am Ende noch widersprüchliche Deklarationen vorhanden, wird die Reihenfolge beachtet. Später deklarierte Eigenschaften überschreiben früher deklarierte.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_06**

Last update: **2024/05/09 06:44**

# Vererbung

Unter **Vererbung** versteht man die Weitergabe von Eigenschaften eines Elternelements an seine Kindelemente. In CSS steuert die Vererbung, was passiert, wenn für eine Eigenschaft eines Elements kein Wert angegeben wird.

CSS-Eigenschaften können in zwei Typen eingeteilt werden:

- **vererbte Eigenschaften**, die standardmäßig auf den berechneten Wert des übergeordneten Elements gesetzt werden
- **nicht vererbte Eigenschaften**, die standardmäßig auf den Standardwert der Eigenschaft gesetzt werden

In jeder CSS-Eigenschafts-Referenz kann man nachlesen, ob eine bestimmte Eigenschaft standardmäßig vererbt wird („Vererbt: ja“) oder nicht („Vererbt: nein“).

## Vererbte Eigenschaften

Wenn für ein Element kein Wert für eine geerbte Eigenschaft angegeben wurde, erhält das Element den berechneten Wert dieser Eigenschaft von seinem übergeordneten Element. Nur das Wurzelelement des Dokuments erhält den in der Zusammenfassung der Eigenschaft angegebenen Anfangswert.

Ein typisches Beispiel für eine geerbte Eigenschaft ist die color-Eigenschaft:

```
h1 {
  color: green;
  border-left: medium solid green;
}
```

```
<h1>Ein <abbr>CSS</abbr>-Beispiel</h1>
```

In diesem Beispiel werden die Eigenschaften *color* und *border-left* für das h1-Element deklariert. Die Farbangabe wird auch von dem *abbr*-Element verwendet, da *color* eine Eigenschaft ist, die vererbt wird. Der tatsächliche Standardwert der color-Eigenschaft (der entweder vom Benutzer oder vom Browser im Default-Stylesheet für das Wurzelelement festgesetzt wird), wird hier nicht angewendet.

## Nicht vererbte Eigenschaften

Wenn für ein Element kein Wert für eine nicht vererbte Eigenschaft angegeben wurde, erhält das Element den Standardwert dieser Eigenschaft.

Ein typisches Beispiel für eine nicht vererbte Eigenschaft ist die border-Eigenschaft:

```
h1 {
  color: green;
```

```
  border-left: medium solid green;
}
```

```
<h1>Ein <abbr>CSS</abbr>-Beispiel</h1>
```

Die Eigenschaft *border-left* wird nicht vererbt, daher besitzt das abbr-Element im Gegensatz zum h1-Element keinen seitlichen Rahmen. Da *border-left* eine shorthand-Eigenschaft ist, werden die Standardwerte aller zusammengefassten Eigenschaften verwendet. Ausschlaggebend ist hier also, dass der Standardwert von border-left-style dem Wert none entspricht.

# Wertvarianten

Ein Wert, der einer Eigenschaft zugewiesen wird, ist nicht unveränderlich. Tatsächlich werden während der Verarbeitung eines Stylesheets verschiedene spezialisierte Werte, sogenannte Wertvarianten aus dem festgelegten Wert gebildet.

Ausgangspunkt ist der …

## Standardwert

Eine CSS-Eigenschaft kann ohne einen gültigen Wert nicht existieren, sie kann nicht unbestimmt oder undefiniert sein. Es ist klar, dass ein Text in einer bestimmten Schriftfarbe dargestellt werden muss, auch ohne dass der Entwickler dazu eine Angabe macht. Dieser Wert wird als **Standardwert**, auch default-Wert, Ausgangswert, voreingestellter Wert oder initialer Wert bezeichnet.

Dieser wird teilweise vom W3C, teilweise auch von Browsern (mehr oder weniger gleich; vgl. margins bei li) festgelegt und findet sich in den Default-Stylesheets der Browser, die Sie in eigenen Stylesheets überschreiben können.

## festgelegte Wert

Als „festgelegt“ bzw. „spezifiziert“ wird ein Wert bezeichnet, wenn er einer Eigenschaft zugeordnet wurde. Die Zuordnung kann während der Kaskade, durch das Prinzip der Vererbung oder durch die Eigenschaft selbst erfolgen. Der Ablauf stellt sich wie folgt dar:

1. Wird während der Kaskade ein Wert für die Eigenschaft gefunden, so wird dieser Wert verwendet. Das bedeutet, dass der verwendete Wert im Browser-, Autoren- oder Benutzer-Stylesheet definiert wurde.
2. Anderenfalls wird, wenn die Eigenschaft vererbt wird und es sich bei dem Element nicht um das Wurzelelement handelt, der für das Elternelement berechnete Wert derselben Eigenschaft verwendet.
3. Ansonsten gilt der Standardwert der Eigenschaft als festgelegter Wert.

```
font-size: 1.2em;
```

In diesem Beispiel enthält die Deklaration der Eigenschaft font-size den Wert 1.2em. Der festgelegte Wert lautet also „1.2em“.

## Der berechnete Wert

Für jeden Wert wird während der Kaskade der „berechnete Wert“ gebildet. Jede Eigenschaft besitzt eigene Regeln, wie dieser berechnete Wert bestimmt wird (die unser Wiki leider noch nicht enthält, bitte folgen Sie im Zweifelsfall den Links zur Mozilla-Dokumentation).

Relative Werte wie bestimmte Längenangaben (außer Prozentwerte) und Pfadangaben werden dabei in absolute Werte umgewandelt.

```
background-image: url('bilder/hintergrund.png'); }
```

In diesem Beispiel wird die Eigenschaft background-image deklariert. Der berechnete Wert dieser Eingenschaft kann z.B. url('http://www.example.org/bilder/hintergrund.png') lauten.

Bei einzelnen Eigenschaften sind spezielle Regeln für die Findung des berechneten Wertes definiert.

Anhand der font-size-Eigenschaft wird deutlich, warum berechnete anstatt festgelegter Werte vererbt werden.

```
<body>
 <h1>Ein <abbr>CSS</abbr>-Beispiel</h1>
</body>
```

```
body {
  font-size: 20px;
}
h1 {
  font-size: 1.5em;
}
```

Für das h1-Element wird die Schriftgröße auf 30 Pixel berechnet (20 Pixel mal 1,5 ergibt 30 Pixel). font-size ist eine Eigenschaft, die vererbt wird, daher ist die Angabe font-size: inherit; beim abbr-Element nicht erforderlich.

Angenommen, es würde der festgelegte Wert vererbt. Dann betrüge die Schriftgröße des abbr-Elements 45 Pixel (30 Pixel mal 1,5 ergibt 45 Pixel). Dies wäre jedoch unerwünscht, da das Element der strukturellen Auszeichnung und nicht der zusätzlichen Gestaltung dient.

Tatsächlich wird daher der berechnete Wert des h1-Elements (30 Pixel) geerbt.

## benutzte Wert

Die berechneten Werte entstehen während der Kaskade, bevor die Darstellung erfolgt. Dies hat zur Folge, dass sich nicht aus allen festgelegten Werten ein spezialisierter berechneter Wert bilden lässt, z. B. aus Prozentangaben, da die Basis dafür erst während der Darstellung berechnet wird. Während der Darstellung werden daher „benutzte“ bzw. „verwendete“ Werte ermittelt.

```
p {
   width: 80%;
```

```
}

In diesem Beispiel wird dem p-Element eine Breite von 80% zugewiesen.
Während der Kaskade ist es nicht möglich, die Breite des Elternelements von
p herauszufinden. Der berechnete Wert von width beträgt also 80%. Erst nach
der Kaskade, wenn die Darstellung erfolgt, kann der benutze Wert ermittelt
werden. Besitzt das Elternelement von p beispielsweise eine Breite von 220
Pixel, beträgt der benutze Wert für die Breite des Absatzes 176 Pixel (220
Pixel mal 0,8 ergibt 176 Pixel).
```

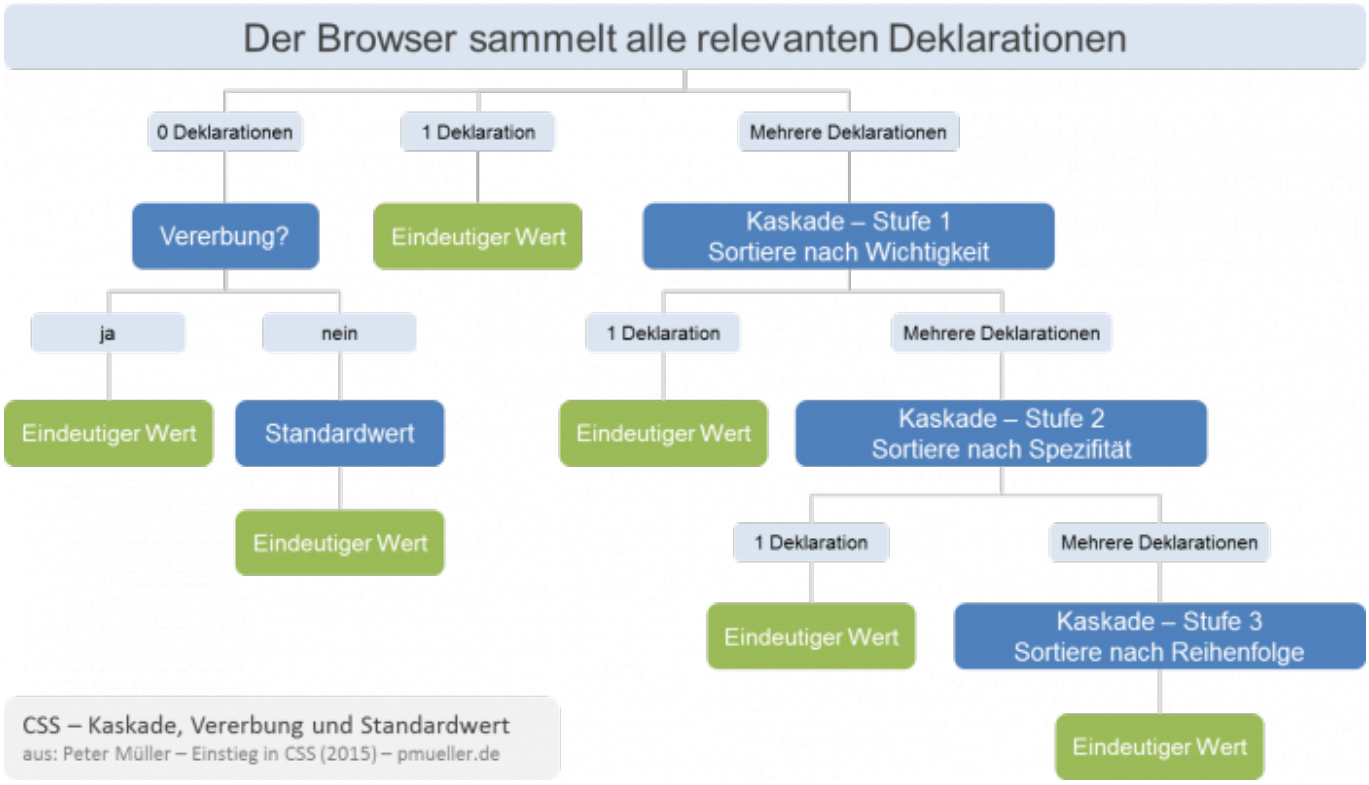
### tatsächliche Wert

Durch die Einschränkungen, die einem Ausgabemedium innewohnen können, ist es manchmal nicht möglich, dass der benutzte Wert verwendet wird. Beispielsweise können auf einem Bildschirm keine Pixelwerte mit Dezimalstellen dargestellt werden. Der Browser muss hier einen Annäherungswert an den benutzten Wert ermitteln. Dieser Wert wird als „tatsächlicher“ Wert bezeichnet.

```
<div id="information">
  <p class="kleingedrucktes">Kleine Information.</p>
</div>
```

```
#information     { font-size: 16px; }
.kleingedrucktes { font-size: .8em; }
```

In diesem Beispiel wird die Schriftgröße des div-Elements auf 16 Pixel festgesetzt. Bei Elementen mit der Klasse kleingedrucktes soll die Schriftgröße nur 80% der normalen Schriftgröße betragen. Der Browser errechnet daher den berechneten und benutzten Wert 12,8 Pixel (16 Pixel mal 0,8 ergibt 12,8 Pixel). Auf einem Ausgabemedium auf dem nur ganzzahlige Pixelwerte dargestellt werden können, rundet der Browser den berechneten Wert auf und bildet so den tatsächlichen Wert: 13 Pixel.



From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_07**

Last update: **2024/05/13 06:12**

# Grid Layout

Ein grid (englisch für Gitter, Gestaltungsraster)) ist ein Ordnungssystem in der visuellen Kommunikation, das als Hilfskonstruktion die Organisation von grafischen Elementen auf einer Fläche oder in einem Raum erleichtert.

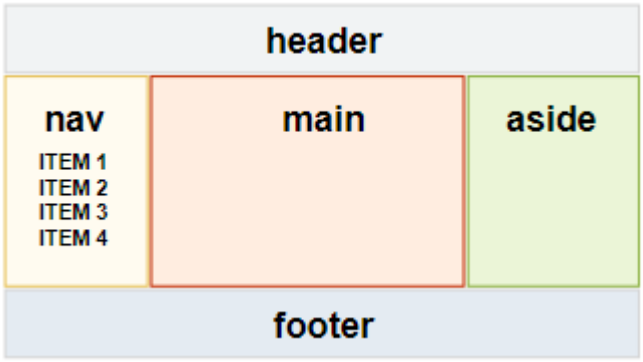
Im Webdesign waren das früher Tabellen-Layout der 90er und die Entwicklung von „CSS-Frameworks“ wie bootstrap Versuche, Webseiten übersichtlich mit Rastern zu gestalten. Allerdings benötigten diese Ansätze eine Vielzahl von zusätzlichen HTML-Elementen, die das Markup an sich unübersichtlich werden ließen und das Prinzip der Trennung von Inhalt, Präsentation und Verhalten missachteten.

Mit dem CSS Grid Layout Modul ist nun endlich eine Lösung gefunden worden. Teilweise ist es sogar möglich, alle CSS-Festlegungen im Grid-Container vorzunehmen, was responsive Flexibilität und spätere Änderungen sehr erleichtert.

Dieses Kapitel stellt mehrspaltige Allround-Layouts vor, die Seitenelemente nach dem Grundsatz „Mobile first!“ auf kleinen Viewports untereinander, bei breiteren Bildschirmen in mehreren Spalten nebeneinander angeordnet werden.

## Holy-Grail-Layout

Das Holy-Grail-Layout ist ein Webseiten-Layout, das mehrere, trotz unterschiedlichen Inhalts gleich hohe Spalten hat. Es wird häufig gewünscht und implementiert, konnte aber viele Jahre lang mit damals verfügbarem CSS nur mit vielen Hacks, die alle Nachteile hatten, realisiert werden. Aus diesem Grund wurde das Finden einer optimalen Implementierung mit der Suche nach dem schwer fassbaren Heiligen Gral verglichen.



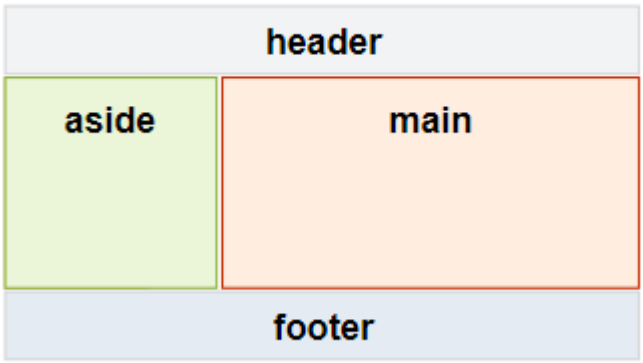

```
body {
   display: grid;
   gap: 0.5em;
}

@media (min-width: 45em) {
   /* Breite beträgt mindestens 45em */
   body {
      grid-template-columns: 1fr 3fr 1fr;
      grid-template-rows: auto 1fr 100px;
```

```
    }
}

header,footer {
     grid-column: 1 / -1;
}
```

Das Grid Layout erstellt für den body ein Raster mit 3 Spalten und 3 Reihen. Diese erhalten zum Teil feste, zum Teil flexible Höhen und Breiten. So bestehen die Werte für grid-template-columns nur aus Bruchteilen (fr - der verfügbare Platz verteilt sich automatisch. Die Höhe richtet sich in der ersten Reihe nach dem Inhalt (auto), in der letzten beträgt sie 100px Höhe - die mittlere Reihe nimmt mit der Festlegung 1fr wieder den verfügbaren Rest ein.

Nur header und footer werden mit grid-column von der ersten (1) bis zur letzen Randlinie (-1) positioniert. Im 3-Spalten-Layout bei breiten Viewports erstrecken Sie sich über die gesamte Breite.

## Zweispaltiges Layout




```
body {
   display: grid;
   gap: 0.5em;
}

@media (min-width: 30em) {
   /* Breite beträgt mindestens 30em */
   body {
      grid-template-columns: 1fr 3fr;
      grid-template-rows: auto 1fr 100px;
   }
}

header,footer {
   grid-column: 1 / -1;
}
```

# Medien im Grid-Layout

Das Media-Objekt ist ein Entwurfsmuster mit einer Grafik links und heading sowie Text rechts daneben.


```
<ul class="media">
  <li>
    <img src="Frühling.png" alt="Bild einer Narzisse, Quelle Wikipedia">
    <h3>Frühling</h3>
    <p>Endlich wird es wieder ein bisschen sonniger und wärmer.
       Wer jetzt spazieren geht, sieht schon die ersten grünen Knospen
sprießen.
     </p>
     <footer>
    optionaler Footer
     </footer>
  </li>
</ul>
Das Media-Objekt besteht aus einem Listenelement, das ein Bild, eine
Überschrift sowie einen Textabsatz und optional einen Footer enthält.
.media li {
  display: grid;
  grid-template-columns: 7em 1fr;
  grid-template-rows: 2em 3em;
  gap: 1em;
  margin-bottom: 1em;
}

.media li>img {
  grid-row: 1 / 3;
  width: 6em;
}
```

# Längenangabe

Viele Tutorials über Grid Layout verwenden für die Spaltenbreiten Längenangaben in festen Pixelwerten. Dies ist möglich; verhindert aber einen entscheidenen Vorteil des Grid Layout Moduls: anstelle des Entwicklers legt der Browser des Benutzers die Maße anhand des verfügbaren Platzes fest.

Anstelle des Werts auto, der dem Inhalt den minimal benötigten Platz zuweist, können Sie auch eigene Zuweisungen in Form von Längenangaben oder Bruchteilen angeben. Schon bei Flexbox konnte man auf die Berechnung von Prozent-Werten verzichten und der flex-Eigenschaft eine Zahl als Anteil an der verfügbaren Breite zuordnen, aus deren Gesamtwerten dann die verfügbare Breite vom Browser berechnet wurde. Im Grid Layout gibt es hierfür die neue Einheit fr (fraction: engl. für Bruchteil).

Ändern Sie im obigen Beispiel die Werte für die Spalten auf:

```
 grid-template-columns: 1fr 1fr 1fr ;
```

Jede Spalte bekommt nun ein 1/3 der gesamten Breite.

## repeat ()

Das „CSS-Framework“ Bootstrap besteht aus einem Raster mit 12 Spalten. Hier könnte man mit der repeat()-Funktion auf eine Angabe der Einzelwerte verzichten:

```
grid-template-columns: repeat(12, 1fr);
```

Die Kindelemente des Rasters verteilen sich auf 12 Spalten. Für unser Beispiel ist das aber noch nicht praktikabel, da die sechs direkten Kindelemente des Rasters jetzt alle nebeneinander und sehr schmal dargestellt werden

## gap

Das Grid-Layout platziert die Grid-Streifen normalerweise direkt nebeneinander. Die frühere Methode, für Abstand zu sorgen, bestand aus dem Setzen von margin-Angaben, was aber zu störenden Rändern am Rand des Grids führen kann.

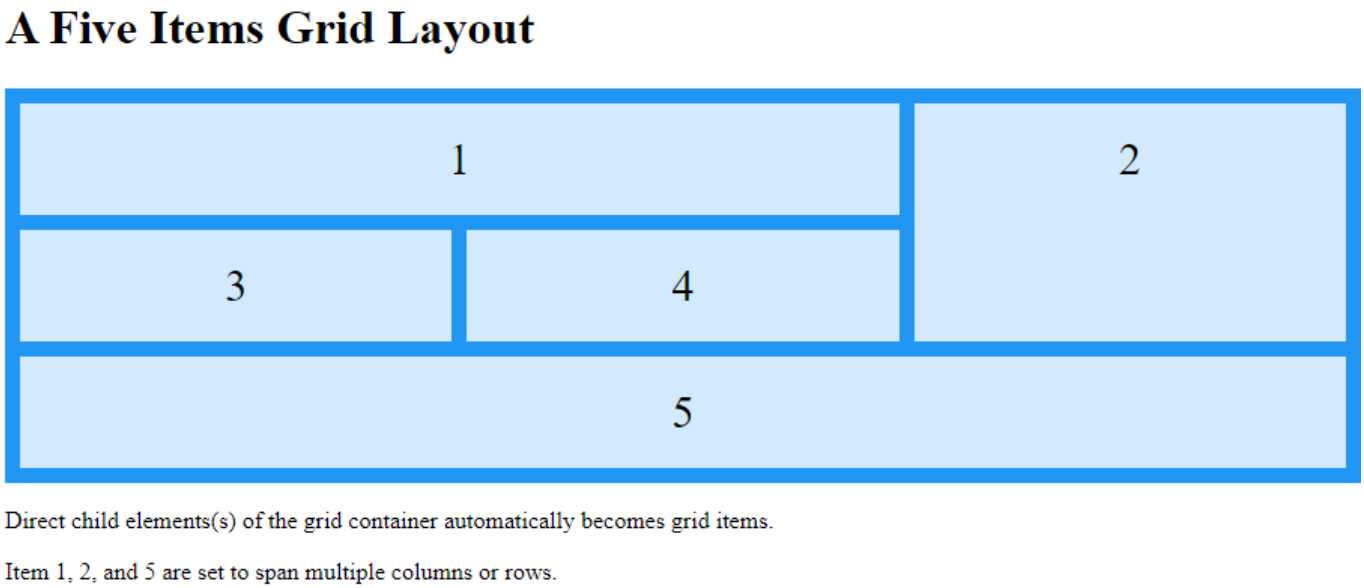
Die Eigenschaften column-gap und row-gap ermöglichen das Setzen von Spalten- und Zeilenabständen direkt im Grid. Sie können eine Längen- oder Prozentangabe verwenden, oder mit calc() berechnen. Um Spalten- und Zeilenabstand gleichzeitig zu setzen, gibt es die Shorthand-Eigenschaft gap.

Beispiel

# Grid Items

Ein Grid Cointainer enthält Grid Items. Jeder Container hat zumindest ein Grid Item für jede Spalte. Für jede Zeile und Spalte kann man mehrere Grid Items zusammenfassen.

Beispiel:




```
<html>
<head></head>
<body>

<h1>A Five Items Grid Layout</h1>

<div class="grid-container">
  <div class="grid-item item1">1</div>
  <div class="grid-item item2">2</div>
  <div class="grid-item item3">3</div>
  <div class="grid-item item4">4</div>
  <div class="grid-item item5">5</div>
</div>

<p>Direct child elements(s) of the grid container automatically becomes grid items.</p>

<p>Item 1, 2, and 5 are set to span multiple columns or rows.</p>

</body>
</html>
```

```
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr ;
  grid-template-rows: 1fr 1fr 1fr;
  gap: 10px;
```

```
  background-color: #2196F3;
  padding: 10px;
}

.grid-item {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px;
  font-size: 30px;
}

.item1 {
  grid-column: 1 / span 2;
  grid-row: 1;
}

.item2 {
  grid-column: 3;
  grid-row: 1 / span 2;
}

.item5 {
  grid-column: 1 / span 3;
  grid-row: 3;
}
```

# Responsive Grid-Layouts

Blockelemente wie Überschriften oder Absätze nehmen im Standardverhalten der Browser stets die verfügbare Breite ein. Mit der Einführung von CSS wurde es möglich, Positionierungen und Größenangaben auch außerhalb des Elementflusses festzulegen.

Dies ermöglichte einerseits ein pixelgenaues Layout, wurde andererseits bei einer Änderung des Viewports zum Bumerang, da sich Inhalte außerhalb des sichtbaren Bereichs oder unter anderen Elementen befanden oder durch overflow: hidden; abgeschnitten wurden. Media queries bieten hier eine Lösung, doch die immer wiederkehrende Frage nach den richtigen Breakpoints zeigt, dass aber auch dies nicht die einfachste Lösung ist.

Grid Layout beginnt wieder von vorne: Der verfügbare Raum wird aufgeteilt, weitere Inhalte werden innerhalb des Rasters in weiteren Reihen dargestellt und wandern so nach unten, wo man sie durch Scrollen erreichen kann.

## Responsives Raster mit Media-Queries

```
body{
  display: grid;
  grid-template-columns: repeat(2, 1fr);
}
```

```
@media (min-width: 30em) {
  body{
    grid-template-columns: repeat(3, 1fr);
}

@media (min-width: 50em) {
  body{
    grid-template-columns: repeat(4, 1fr);
  }
}
```

## Responsivität ohne Media queries

Im Folgenden soll gezeigt werden, wie Raster mit flexibler Spaltenanzahl angelegt werden, die sich auch ohne Media Queries responsiv an den Viewport anpassen

### auto-fill

Bequemer als die Lösung mit media queries ist es allerdings, mittels des Schlüsselworts auto-fill den Browser die Anzahl der Spalten automatisch festlegen zu lassen:

```
body{
  display: grid;
  grid-template-columns: repeat(auto-fill, 20em);
}

article {
  grid-column: 1 / -1;
  grid-row: span 2;
}
```

In diesem Beispiel wurde die repeat()-Funktion verwendet, die nun über den ersten Parameter auto-fill so viele Spalten wie möglich erzeugt. Der zweite Parameter gibt die Breite der Spalten (20em) vor.

Das article-Element soll, wenn der Platz ausreicht, breiter werden.

- grid-column: 1 / -1; legt fest, dass der article an der ersten Gridlinie beginnt und an der letzten endet (negative Werte beginnen die Zählung rechts)
- grid-row: span 2; das Element soll sich in der Höhe über 2 Reihen (Rasterfelder) erstrecken

### minmax()

Damit der autoplacement-Algorithmus seine volle Wirkung entfalten kann, muss die Breite der Spalten flexibel sein. Bisher wurde dies bei fester Spaltenanzahl mit der Einheit fr erledigt. Nun sollen die Spalten eine feste Mindestgröße, aber auch keinen überschüssigen Raum in Form eines leeren Rands haben.

Voilà: Die minmax()-Funktion erlaubt es anstelle von festen oder relativen Längenangaben einen Mindest- und einen Maximalwert anzugeben:

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

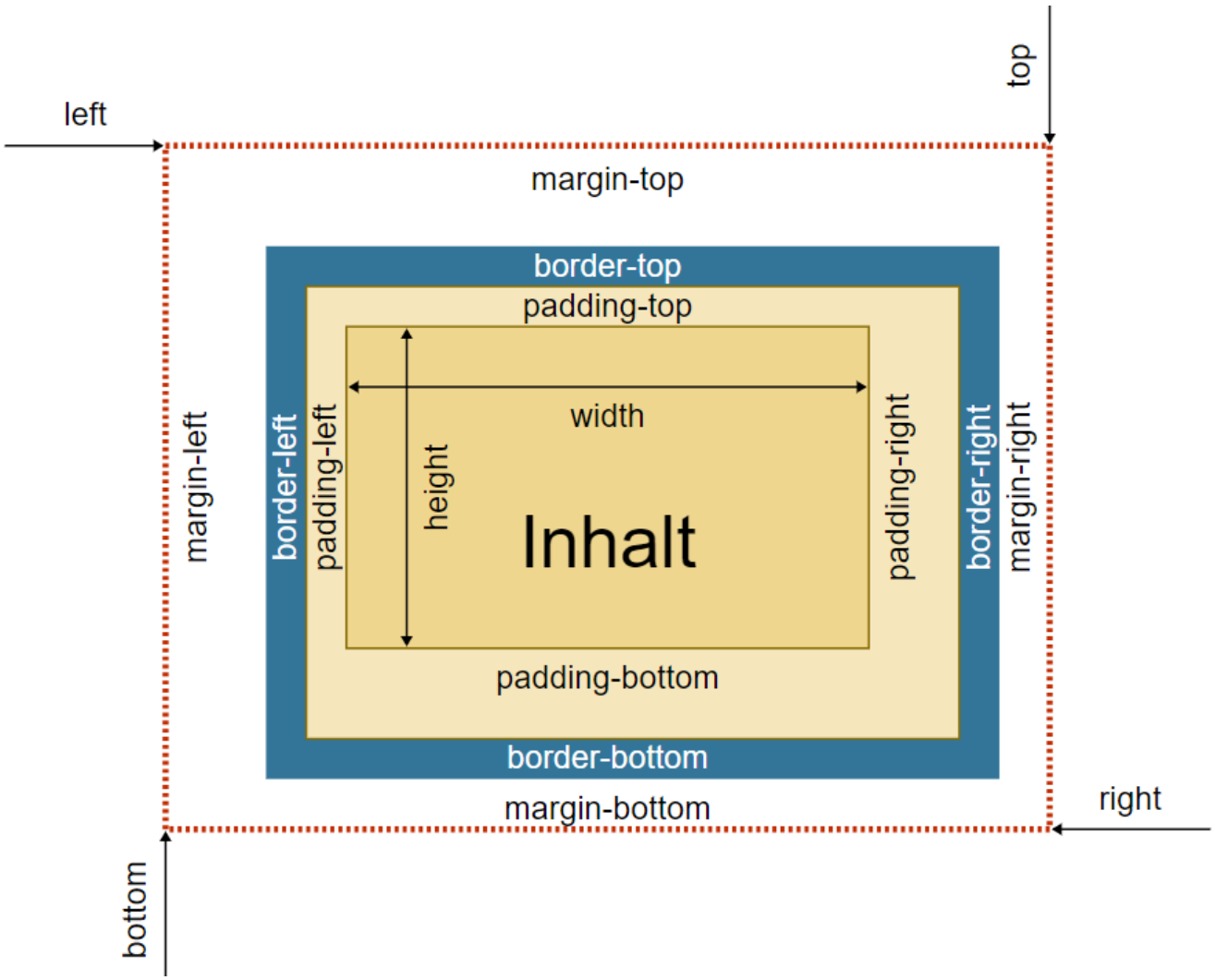
Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_08**

Last update: **2024/09/11 06:54**

# Boxmodell

Was vielen nicht bewusst ist: HTML ist von Haus aus responsiv - Block-Elemente wie Überschriften oder Absätze nehmen die gesamte Breite des Viewports ein und brechen überstehenden Text um, sodass er nach unten in einer neuen Zeile dargestellt wird.

## Das „klassische“ Boxmodell



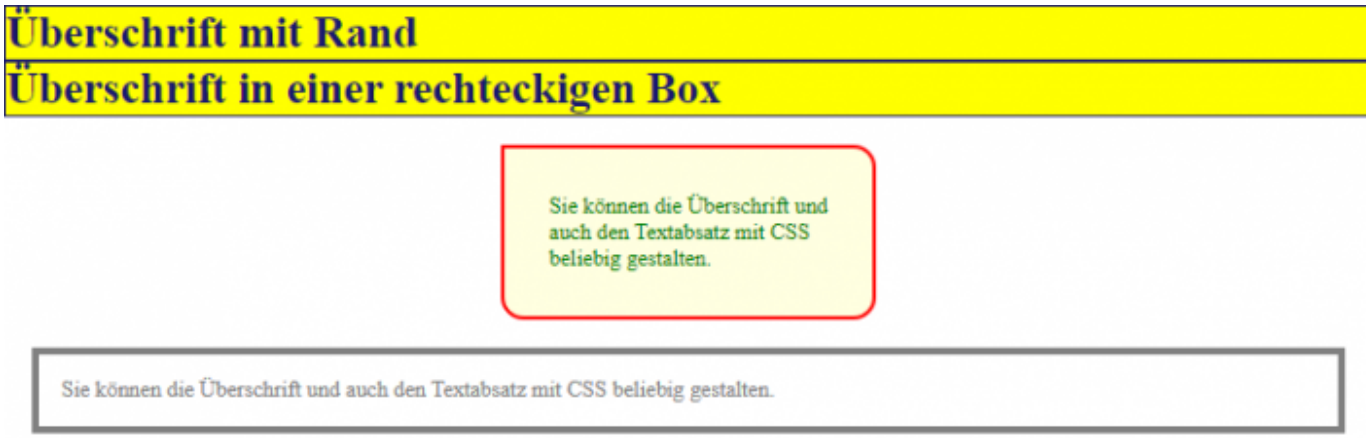

Mit CSS können Sie Elementen (feste) Breiten, Höhen und Abstände geben und diese nebeneinander positionieren. Die rechteckigen Blöcke, die im Elementbaum erzeugt und auf dem Bildschirm dargestellt werden, folgen einem Schema, dem „Box-Modell“. Es ist somit Grundlage jeden Layouts.

Jedes Block-Element wie Textabsätze (p), -Abschnitte (div, article, main, aside, …) und Überschriften (h1, h2, …) bildet eine rechteckige Box, die Sie frei formatieren können.

Im CSS-Einstieg haben wir Überschriften oft eine Text-und Hintergrundfarbe zugewiesen. Im folgenden Beispiel wollen wir die um Randlinien und Abstände erweitern:

```
h1 {
  color: midnightblue;
```

```
  background-color: yellow;
  border: 1px solid;
  margin: 0;
  padding: 0;
}

.box {
  width: 200px;
  color: green;
  background-color: lightyellow;
  border:  2px solid red;
  border-radius: 0 1em 1em 1em;
  margin:  20px auto;
  padding: 2em;
}
```




Im Beispiel erhalten Überschrift *h1* und Textabsatz *p* zusätzlich mit *border* noch eine Randlinie. Die Eigenschaft ist eine zusammenfassende „shorthand“-Eigenschaft der drei Werte für die Randstärke *border-width*, die Form *border-style* und den optionalen Wert für die Randfarbe *border-color*. Wird dort kein Wert zugewiesen, wird die Textfarbe des Elements verwendet.

Mit *margin* können Sie einen Abstand zum Elternelement und benachbarten Elementen festlegen. Die Überschrift „klebt“ durch margin: 0; am oberen Rand der Seite. Der Textabsatz erhält mit margin: 20px auto; einen Abstand von 20px nach oben (und unten), sowie mit auto einen gleichmäßig verteilten Abstand nach rechts und links. In Verbindung mit der Festlegung für die Breite width: 200px; wird der Textabsatz zentriert. Die Überschrift stößt fast gegen die Randlinie; der Textabsatz hat mehr Platz. Das erreichen Sie durch eine Angabe eines Innenabstands padding (engl. für Polsterung)

Alle diese Eigenschaften verwenden Längenangaben. Dabei können Sie absolute Werte in Pixel, aber auch relative Werte wie Prozentangaben und die Längeneinheit em, die die Größe der verwendeten Schriftart als Basisschriftgröße verwendet, verwenden.

## padding oder margin

Es wird oft gefragt, welche der beiden Eigenschaften man verwenden soll – padding oder margin?

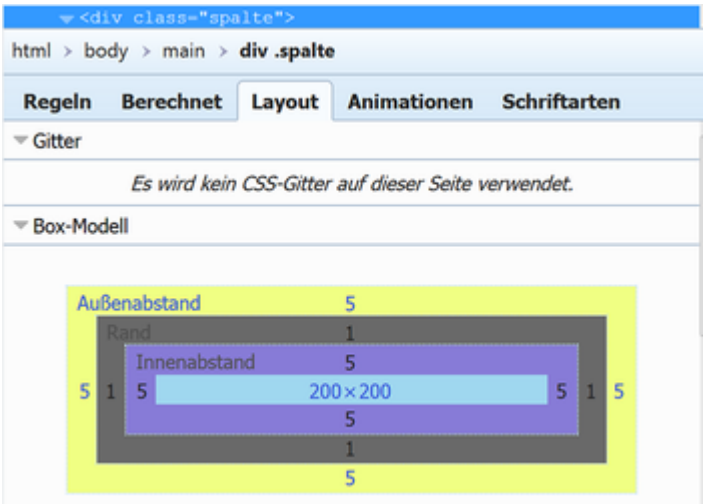
- Wenn die Box eine eigene Hintergrundfarbe hat, soll der Rand um den Inhalt mit gefärbt sein (padding) oder nicht (margin)? Oder gar beides?
- margins können überlappen (→ Collapsing margins). D.h. wenn ich zwei Boxen übereinander habe, und beide haben margin: 2em, dann ist der Abstand zwischen den Boxen 2em, nicht 4em.

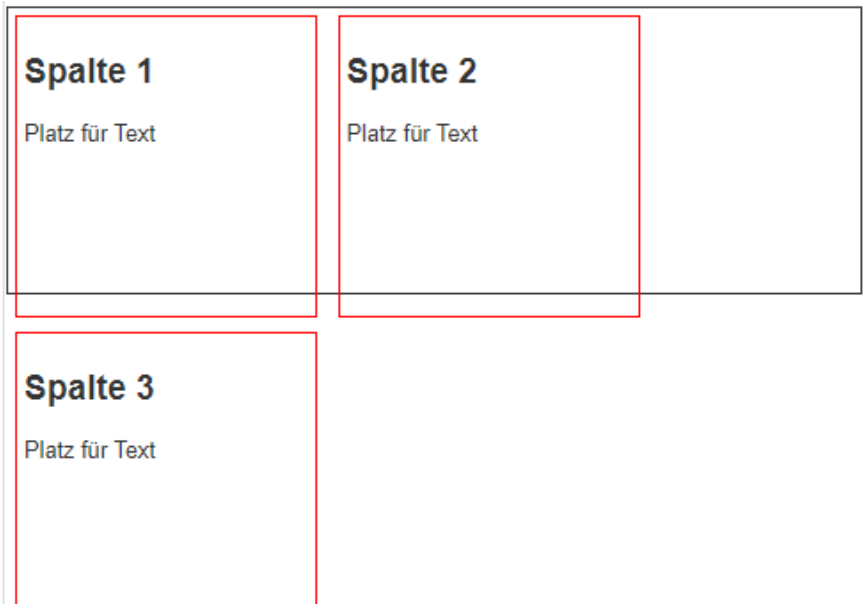
## Beispiel

Damit Text auf unserer Webseite gut lesbar ist, wollen wir ein Drei-Spaltenlayout entwerfen. Die Seite erhält eine feste Breite von 600px, die drei Spalten von je 200px.

```
.container {
  width: 600px;
  height: 400px;
  border: 1px solid;
  padding: 0;

}

.spalte  {
  width:  200px;
  height: 200px;
  border:  1px solid red;
  margin:  5px;
  padding: 5px;
  display: inline-block;
}
```

Das Beispiel besteht aus dem Container-Element, das eine feste Breite von 600px und eine feste Höhe von 200px besitzt. Als Spalten dienen drei section-Elemente, die eine feste Breite und Höhe von je 200px (ein Drittel von 600px) besitzen. Damit die Divs nebeneinander platziert werden, erhalten sie ein display: inline-block;.






Das Beispiel zeigt unerwünschtes Verhalten:

Warum werden die drei Boxen nicht nebeneinander dargestellt? Ein Blick in den Seiteninspektor (erreichbar mit F12) zeigt uns, dass die Spalten aus Boxen mit je 200px Breite, einer Randlinie von 1px und einem Innen- und Außenrand von je 5px, also ingesamt einer Gesamtbreite von 222px bestehen:

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_09**

Last update: **2024/09/11 05:18**

# Selektoren

Damit Formateigenschaften auf ein Element angewendet werden können, muss definiert werden, welche Elemente angesprochen werden. Dies geschieht über Selektoren.

Ein CSS-Selektor nennt die Bedingungen, die auf ein Element zutreffen müssen, damit der nachfolgende Regelsatz an CSS-Deklarationen auf das Element angewendet wird. Der Selektor ist somit der Teil vor den geschweiften Klammern.

Beispiel

```
Selektor {
 /* CSS-Deklarationen
   z.B: font-color: blue;
 */
}
```

## Typselektor

Der Element- bzw. Typselektor besteht aus dem Namen des Elements, das angesprochen werden soll. Mit diesem Selektor werden alle Elemente eines Typs angesprochen.

```
h1 {
  color: red;
}
h1, h2 {
  background-color: #ccc;
  border-radius: .5em;
}
```

Der erste Regelsatz besteht aus einem Selektor h1, der allen h1-Elementen die Textfarbe rot zuweist. Der zweite Regelsatz besteht aus einer Selektor-Liste mit zwei voneinander unabhängigen Selektoren h1 und h2, die alle Überschriften vom Typ h1 und h2 anspricht.

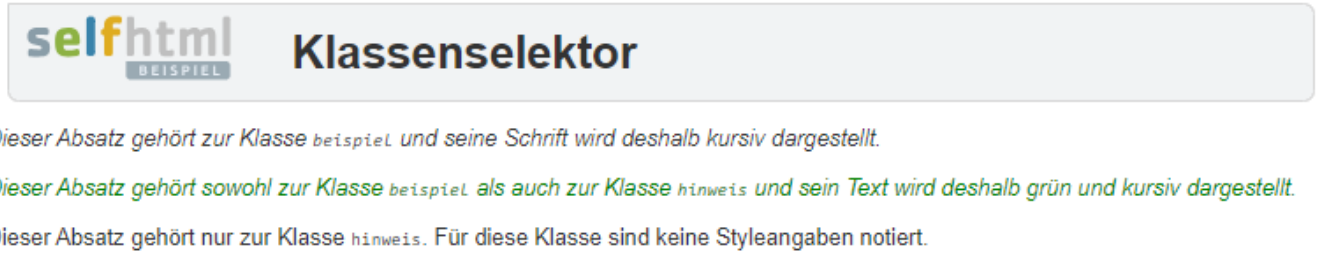

## Klassenselektor

Ein Klassenselektor spricht Elemente an, die einer bestimmten Klasse zugehörend sind. Welche Elemente das sind, hängt von der verwendeten Auszeichnungssprache ab. In HTML-Dokumenten

werden Klassen über das class-Attribut vergeben.

Ein Klassenselektor wird gebildet, wenn vor dem Klassennamen ein Punkt notiert wird.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS-Beispiel: Klassenselektor</title>
    <style>
      .beispiel {font-style: italic;}
      .beispiel.hinweis {color: green;}
    </style>
  </head>
  <body>
    <p class="beispiel">Dieser Absatz gehört zur Klasse
<strong>beispiel</strong> und seine
                        Schrift wird deshalb kursiv dargestellt.</p>
    <p class="beispiel hinweis">Dieser Absatz gehört sowohl zur Klasse
<strong>beispiel</strong>
                                als auch zur Klasse <strong>hinweis</strong>
und sein Text wird deshalb grün und
                                kursiv dargestellt.</p>
    <p class="hinweis">Dieser Absatz gehört nur zur Klasse
<strong>hinweis</strong>.
                       Für diese Klasse sind keine Styleangaben notiert.</p>
  </body>
</html>
```




In diesem Beispiel wird der Text des ersten und zweiten Absatzes kursiv dargestellt, da beide Absätze zur Klasse „beispiel“ gehören. Klassenselektoren können mit anderen Selektoren verbunden werden, um genauer festzulegen, welche Elemente angesprochen werden sollen. Im Beispiel erhält der zweite Absatz eine grüne Schriftfarbe, da das Absatzelement sowohl der Klasse „beispiel“ als auch der Klasse „hinweis“ zugeordnet ist.

Sie können den Klassenselektor auch mit dem Typselektor verbinden, die Schreibweise lautet dann „elementname.klassenname“. Auf diese Weise sprechen Sie nur die Elemente an, bei denen Element- und Klassenname mit den Angaben des Selektors übereinstimmen. Die Kombination Universal- und Klassenselektor („*.klassenname“) ist ebenfalls möglich, aber identisch zur einfachen Schreibweise.

### Beachte:

- .beispiel.hinweis selektiert Elemente, die der Klasse „beispiel“ und „hinweis“ angehören;
- .beispiel, .hinweis selektiert Elemente, die der Klasse „beispiel“ oder „hinweis“ angehören (siehe von einander unabhängige Selektoren)
- .beispiel .hinweis selektiert Elemente, die der Klasse „hinweis“ innerhalb der Klasse „beispiel“ angehören. (siehe Nachfahrenselektor).

Klassennamen sollten nach ihrer Funktion (z.B. „Hinweis“) benannt werden und nicht nach den Formatierungen, die sie beinhalten. So stellen sie sicher, dass die Klasse auch nach einer Überarbeitung der Gestaltung noch nachvollziehbar den Bestandteilen eines Dokuments zugeordnet werden kann. Viele informelle Eigenschaften eines Elements können durch elementspezifische Attribute abgedeckt werden. CSS erlaubt das Ansprechen von Elementen anhand ihrer Attribute. Klassen und Klassenselektoren sollten Sie daher nur dann einsetzen, wenn keine andere Zuordnungsmöglichkeit besteht. </code>

## ID-Selektor

Mit dem ID-Selektor kann ein Element angesprochen werden, dem eine ID mit dem id-Attribut zugeordnet wurde. Ein ID-Selektor wird gebildet, indem das Gatterzeichen „#“ vor den ID-Namen gestellt wird.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS-Beispiel: ID-Selektor</title>
    <style>
      #beispiel { border: medium solid green; }
    </style>
  </head>
  <body>
    <div id="beispiel">
      <h1>ID-Selektoren</h1>
      <p>Dieses Beispiel demonstriert die Wirkung des ID-Selektors.</p>
    </div>
  </body>
</html>
```


In diesem Beispiel erhält das gruppierende div-Element die ID „beispiel“. Die CSS-Regel bewirkt, dass um dieses Element ein grüner Rahmen dargestellt wird.

ID-Selektoren können mit anderen Selektoren verbunden werden:

- Mit dem Universalselektor: *#id – diese Schreibweise ist zu der Schreibweise ohne Universalselektor identisch.
- Mit Elementselektoren: elementname#id
- Mit Klassenselektoren: .klassenname#id bzw. #id.klassenname

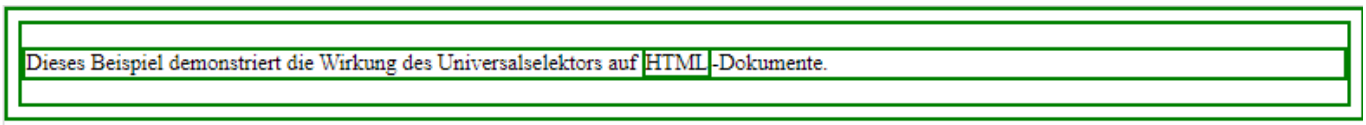
### Beachte:

Eine ID darf in einem Dokument nur ein einziges Mal vorkommen, die Kombination des ID-Selektors mit anderen Selektoren ergibt nur dann Sinn, wenn Sie die entsprechende ID in mehreren Dokumenten und in unterschiedlichen Situationen einsetzen. Welches Attribut als ID erkannt wird, hängt bei XML-Dokumenten von der DTD ab. Da ein XML-Parser eine DTD nicht verarbeiten muss, kann bei dem Browser unbekannten XML-Formaten nicht garantiert werden, dass die CSS-Eigenschaften angewendet werden.

## Universalselektor

Das Sternzeichen „*“ (der Asterisk) ist der Universalselektor. Mit ihm werden alle Elemente in einem Dokument angesprochen.

```
<!doctype html>
<html>
  <head>
    <title>CSS-Beispiel: Universalselektor</title>
    <style>
      * {border: 3px solid green;}
    </style>
  </head>
  <body>
    <p>Dieses Beispiel demonstriert die Wirkung des Universalselektors
      auf <abbr>HTML</abbr>-Dokumente.</p>
  </body>
</html>
```




Zeigt ein Browser obiges Dokument an, so wird um die Elemente html, body, p und abbr ein grüner Rahmen gezeichnet. Das Element head und seine Kindelemente werden aufgrund ihrer speziellen Eigenschaften in HTML überhaupt nicht angezeigt.

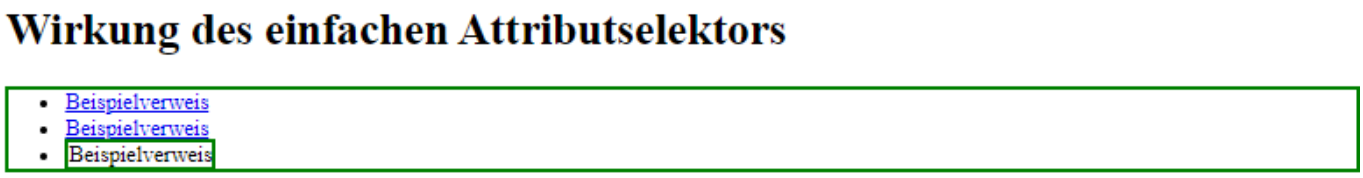
## Attributselektor

Ein Element, das ein bestimmtes Attribut besitzt, kann durch den einfachen Attributselektor angesprochen werden. Ein Attributselektor wird gebildet, indem eckige Klammern („[“ und „]“) um den Namen des Attributs gesetzt werden.

### Beachte:

Ob die Groß- und Kleinschreibung des Attributnamens beachtet wird, hängt davon ab, ob Sie CSS für HTML oder für XML verwenden. Bei HTML wird sie nicht beachtet, bei XML (z. B. in SVG-Dokumenten) hingegen schon.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS-Beispiel: Attributselektor</title>
    <style>
      [title] {border: 3px solid green;}
    </style>
  </head>
  <body>
    <h1>Wirkung des einfachen Attributselektors</h1>
    <ul title="Linkliste">
      <li><a href="http://www.example.org/">Beispielverweis</a></li>
      <li><a href="http://www.example.net/">Beispielverweis</a></li>
      <li><a title="Beispiel">Beispielverweis</a></li>
    </ul>
  </body>
</html>
```




In diesem Beispiel erhalten nur das letzte Element der Liste und die Liste selbst einen grünen Rahmen, da beide über ein title-Attribut verfügen. Der Attributwert spielt für den einfachen Attributselektor keine Rolle.

Attributselektoren können mit anderen Selektoren verbunden werden:

- Mit dem Universalselektor: *[attributname] – diese Schreibweise ist zu der Schreibweise ohne Universalselektor identisch.
- Mit dem Elementselektor: elementname[attributname]
- Mit Klassenselektoren: .klassenname[attributname]
- Mit dem ID-Selektor: #id[attributname]

## Selektorliste

Eine Selektor-Liste ist eine komma-separierte Liste von beliebigen Selektoren. Die Selektorliste trifft auf ein Element zu, wenn mindestens einer der darin enthaltenen Selektoren auf das Element zutrifft.

```
h1 { font-family: sans-serif }
```

```
h2 { font-family: sans-serif }
h3 { font-family: sans-serif }
h1, h2, h3 { font-family: sans-serif }
```

Beide Beispiele setzen für die Überschriften 1., 2. und 3. Ordnung die Eigenschaft font-family auf den Wert sans-serif. Bei dem zweiten Beispiel handelt es sich um eine Selektor-Liste.

# Kombinatoren

Kombinatoren sind Zeichen, die zwei Selektoren miteinander verketten. Durch diese Verkettung bildet der erste Teilselektor eine Bedingung und der zweite Teilselektor das Ziel, das angesprochen werden soll, wenn die Bedingung erfüllt wurde. Der Kombinator gibt dabei an, in welchem Verhältnis die Teilselektoren vor und nach ihm zueinander stehen müssen.

Zwischen zwei Selektoren kann nur ein Kombinator stehen, jedoch kann an diese Kette ein weiterer Kombinator zusammen mit einem weiteren Teilselektor angehängt werden.

Die Bedeutung von kombinierten Selektoren ähnelt der von Pseudoklassen, ist jedoch allgemeiner.

## Kindselektor

Werden zwei Selektoren durch den Kombinator > (schließende spitze Klammer, Größer-Als-Zeichen) verbunden, z. B. E > F, so wird das Element F nur dann angesprochen, wenn es ein Kindelement eines E-Elements ist.

```
p > em {
color: green;
}
```

In diesem Beispiel werden alle em-Kindelemente von p in grüner Schriftfarbe dargestellt. Dies betrifft nur das Wort „Dieses“, da es sich um das einzige em-Element handelt, das direkt innerhalb des Absatzelements vorkommt.

```
<h1>
  Der
  <em>Kind</em>
  selektor
</h1>
<p>
  <em>Dieses</em>
  Beispiel demonstriert die Wirkung des
  <del>
    <em>Nachbar</em>
  </del>
  Kindselektors.
</p>
```

Das em-Element in der Überschrift ist überhaupt kein Kind des p-Elements. Und das em-Element

innerhalb des del-Elements ist kein direktes Kind des p-Elements, sondern ein Kind des del-Elements; also sozusagen nur ein Kindeskind des p-Elements. Deshalb wird es in der normalen Schriftfarbe dargestellt.


## Nachfahrenselektor

Werden zwei Selektoren durch den Kombinator (Leerzeichen) verbunden, z. B. E F, so wird das Element F nur dann angesprochen, wenn es ein Nachfahrenelement eines E-Elements ist. Dabei muss es kein direktes Kindelement sein, sondern kann auch weiter unten im Elementbaum notiert sein.

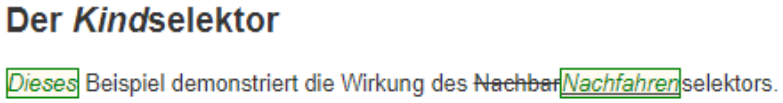
Das Leerzeichen (bzw. ein anderes Whitespace-Zeichen) zwischen zwei Selektoren dient nur dann als Kombinator, wenn kein anderer Kombinator vorhanden ist.

```
p em {
   color: green;
   padding: border: 1px solid;
   background-color: #fffbf0;
   font-style: italic;
}
```

Stilfestlegung für em-Elemente, die Nachfahrenelemente von p sind.

```
<h2>Der <em>Nachfahren</em>selektor</h2>
<p>
  <em>Dieses</em> Beispiel demonstriert die Wirkung des
  <del>Nachbar</del><ins><em>Nachfahren</em></ins>selektors.
</p>
```

In diesem Beispiel werden die beiden em-Elemente innerhalb des Absatzes mit grüner Schriftfarbe dargestellt, da es sich bei beiden um Nachfahrenelemente des p-Elements handelt. Das em-Element innerhalb der Überschrift ist kein Nachfahre des p-Elements, es ist deshalb nicht betroffen.




## Nachbarselektor

Werden zwei Selektoren durch den Kombinator + (Pluszeichen) verbunden, z. B. E + F, so wird das Element F nur dann angesprochen, wenn es im Elementbaum direkt auf ein E-Element folgt, also der direkte Nachbar ist.

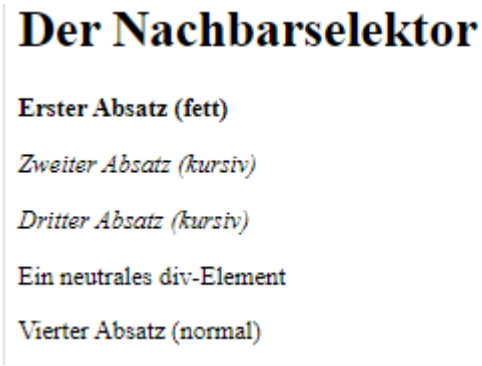
```
h1 + p { font-weight: bold }
```

```
p + p { font-style: italic }
```

```
<h1>Nachbarkombinator</h1>
<p>Erster Absatz.</p>
<p>Zweiter Absatz.</p>
<p>Dritter Absatz.</p>
<div>neutrales Element</div>
<p>Vierter Absatz</p>
```

In diesem Beispiel wird der erste Absatz mit fett formatierter Schrift dargestellt, denn es ist das einzige unmittelbare Nachbar-p-Element von h1. Für den zweiten und dritten Absatz gilt: Sie sind Nachbar-p-Elemente je eines p-Elements. Sie werden also kursiv dargestellt.

Der vierte Absatz ist ein Nachbar eines neutralen div-Elements und wird nicht gestylt.




## Geschwisterselektor

Werden zwei Selektoren durch den Kombinator ~ (Tilde) verbunden, z. B. E ~ F, so werden alle F-Elemente angesprochen, die im Elementbaum in derselben Ebene auf ein E-Element folgen – unabhängig davon, ob sich zwischen den Elementen weitere, im Selektor nicht genannte, Elemente befinden.

```
h1 ~ p {font-weight: bold;}
```

```
<p>Erster Absatz.</p>
<h1>Der Geschwisterselektor</h1>
<p>Zweiter Absatz.</p>
<hr>
<p>Dritter Absatz.</p>
```

In diesem Beispiel werden sowohl der zweite als auch der dritte Absatz mit fett formatierter Schrift dargestellt, da beide Elemente auf das h1-Element folgen. Die Trennlinie zwischen den beiden Absätzen verhindert dies nicht. Einzig der erste Absatz wird mit normaler Schrift dargestellt, da er vor dem bedingenden h1-Element steht.

From:
http://elearn.bgamstetten.ac.at/wiki/ - **Wiki**

Permanent link:
**http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf5ai_202324:07_webentwicklung:7_04:7_04_10**

Last update: **2024/09/12 07:07**