



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

**Innovativ und vielfältig: die Hochschule
für Technik und Wirtschaft Berlin**

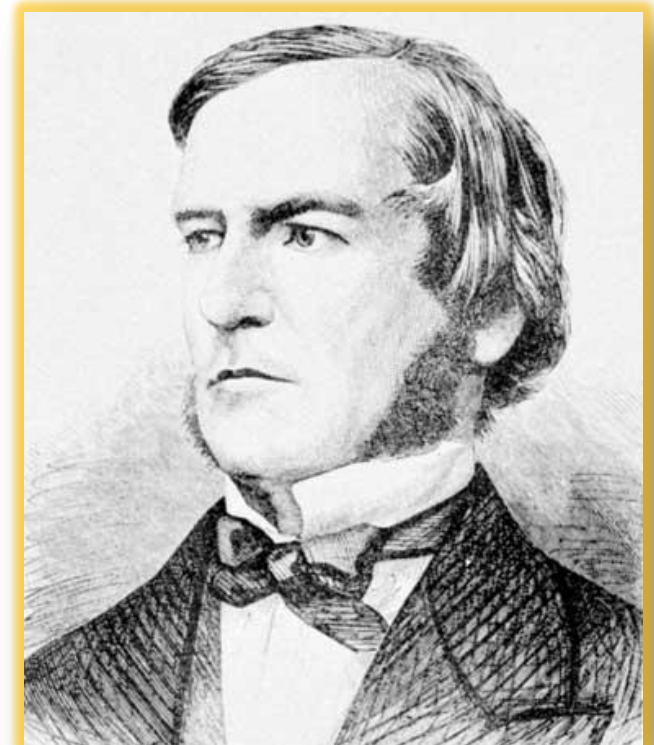
Fachbereich 2
Informatik
Vorkurs **Informatik**

Lektion 2
Brückenkurs Informatik

EINFÜHRUNG IN DIE DIGITALTECHNIK

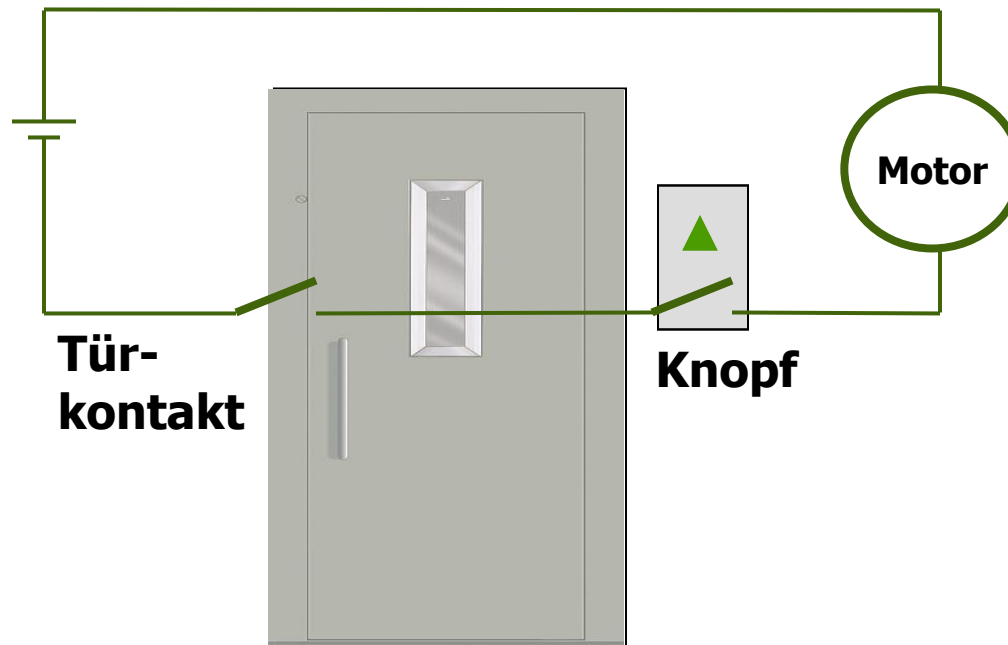
Boolean Logic / Boolsche Logik

- 1854 erfand George BOOLE, ein Britischer Mathematiker ein mathematisches System namens "Boolean Logic"
- Das ist die Mathematik mit Variablen deren Werte nur "True" oder "False" sein können
- 2 Zustände: **DIGITAL (0/1), (Nein/Ja), (Falsch/Wahr)**



Aufzugssteuerung

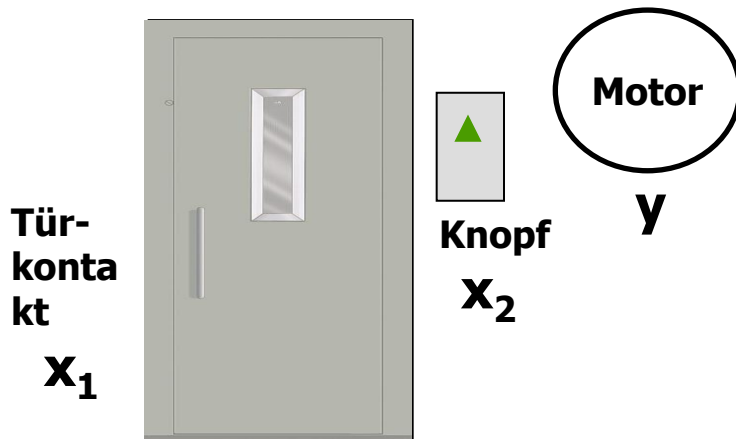
Problem: Ein Aufzug soll sich nur dann nach oben bewegen, wenn der Knopf gedrückt und die Tür zu ist.



Lösung mittels Elektrotechnik: **Reihenschaltung**

nach H. Bühler: Grundlagen einer Verständigung mit Computern

Binäre Kodierung mit Schaltvariablen



Schaltvariablen bzw. logische Variablen können nur zwei Werte annehmen:

0 bzw. false, „nein“ **oder**
1 bzw. true, „ja“

Übung: Interpretieren Sie die folgenden Systemzustände:

Definition der Schaltvariablen:

x_1 : „Tür geschlossen?“

x_2 : „Knopf gedrückt?“

y : „Motor läuft?“

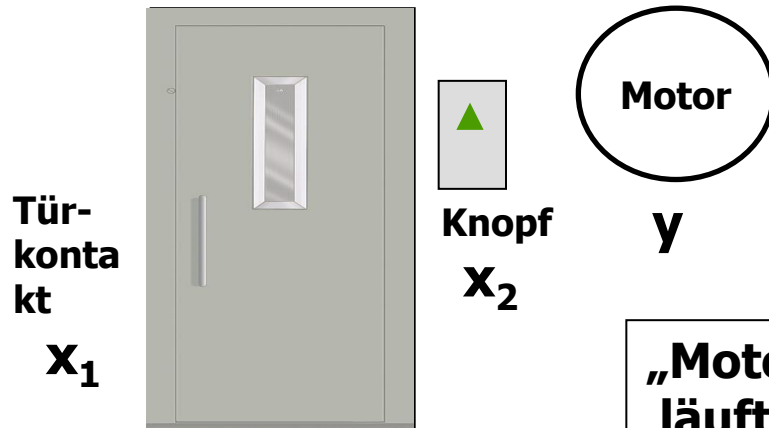
a) $x_1=1, x_2=0$

b) $x_1=0, x_2=1$

c) Ergänzen Sie
in der Schalttable
das gewünschte
Ausgabe-Verhalten:

| x_1 | x_2 | y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Logik-basierte Systembeschreibung ...



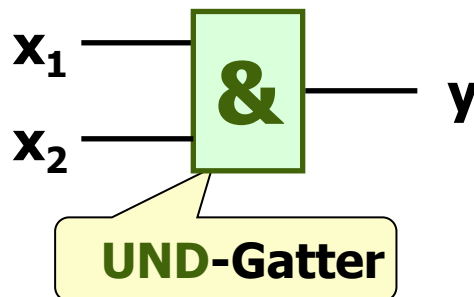
Beschreibung des Systemverhaltens durch eine aussagenlogische Formel (Schaltfunktion):

„Motor läuft“ = „Tür ist geschlossen“ und „Knopf ist gedrückt“

$y = x_1 \wedge x_2$

technische Realisierung mit Logikbaustein:

UND-Operator

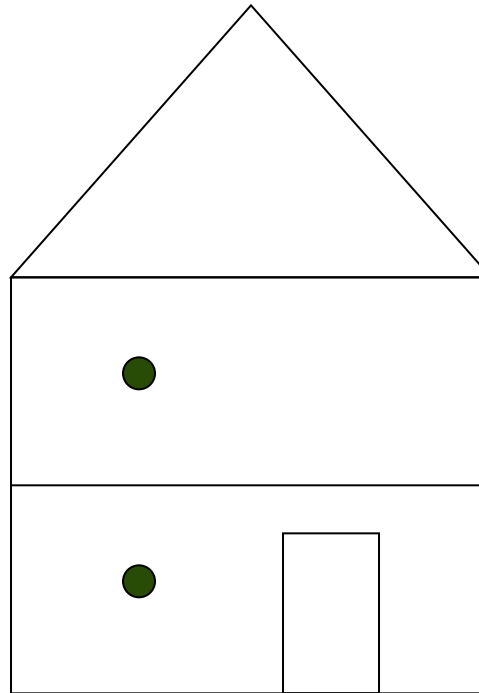


nur Signalfluss,
kein Stromfluss!

| x_1 | x_2 | y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Problem 2: Steuerung eines Türöffners

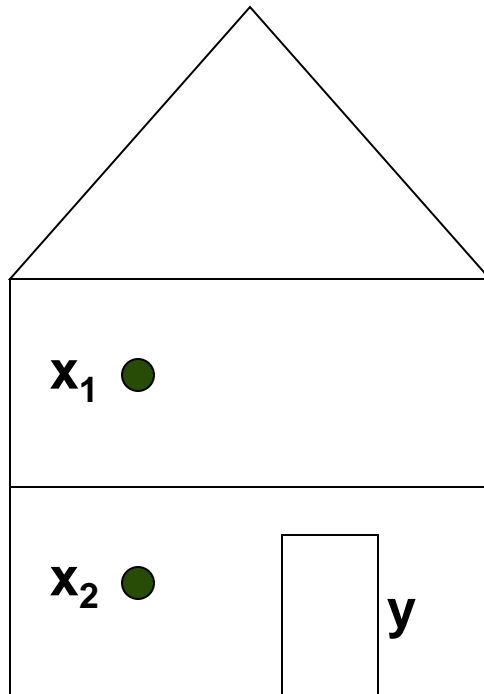
Problem: Die Haustür soll sich öffnen, wenn der Türöffner im ersten oder im zweiten Stock gedrückt wird.



Schaltvariablen?

nach H. Bühler: Grundlagen einer Verständigung mit Computern

Lösung mit ODER-Operator



**Beschreibung der Systemgrößen
mit Schaltvariablen:**

x_1 : „Türöffner im 1. Stock ist gedrückt“
 x_2 : „Türöffner im 2. Stock ist gedrückt“
 y : „Türschloss ist geöffnet“

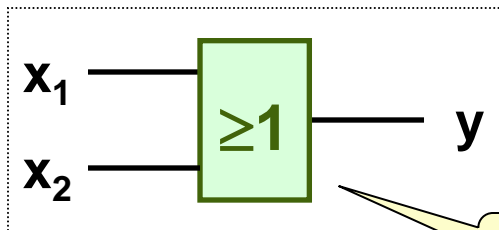
**Beschreibung des Systemverhaltens
mit logischen Operationen bzw.
Wertetabelle:**

$$y = x_1 \vee x_2$$

ODER-Operator

| x_1 | x_2 | y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

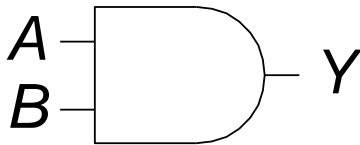
Realisierung:



ODER-Gatter

Logic Gates => Gatter

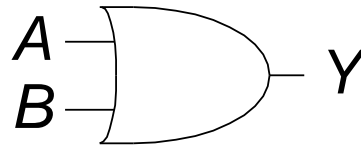
AND



$$Y = AB$$

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

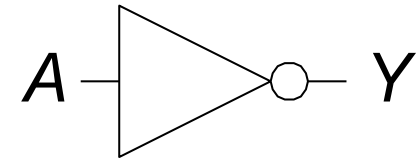
OR



$$Y = A + B$$

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

NOT

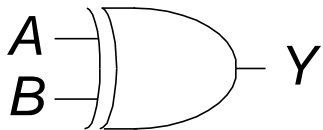


$$Y = \overline{A}$$

| A | Y |
|---|---|
| 0 | 1 |
| 1 | 0 |

More Two-Input Logic Gates

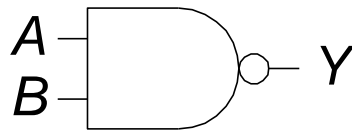
XOR



$$Y = A \oplus B$$

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

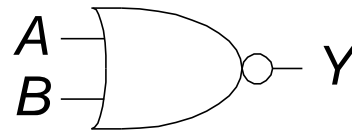
NAND



$$Y = \overline{AB}$$

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

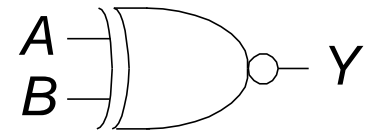
NOR



$$Y = \overline{A + B}$$

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

XNOR



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

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Aussagenlogik:

<http://www.fb10.uni-bremen.de/khwagner/grundkurs2/kapitel3.aspx>

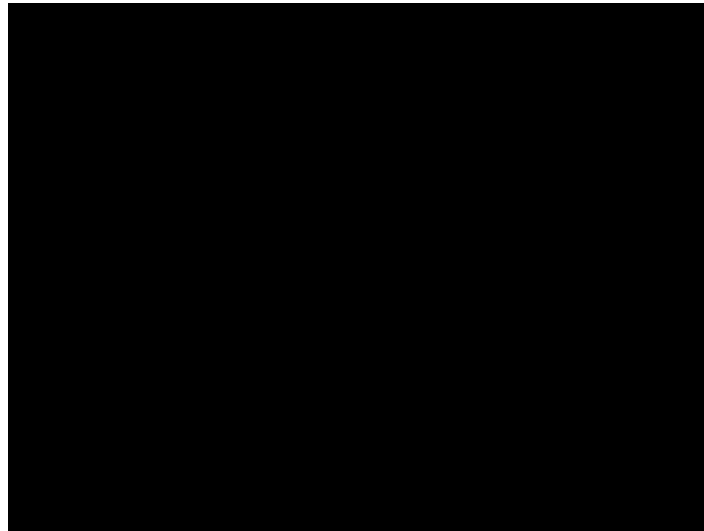
Wahrheitstabellen

| | | AND | OR | XOR | IF | equal |
|-----|-----|--------------|------------|------------------------|-------------------|-----------------------|
| A | B | $A \wedge B$ | $A \vee B$ | $A \underline{\vee} B$ | $A \Rightarrow B$ | $A \Leftrightarrow B$ |
| w | w | w | w | f | w | w |
| w | f | f | w | w | f | f |
| f | w | f | w | w | w | f |
| f | f | f | f | f | w | w |

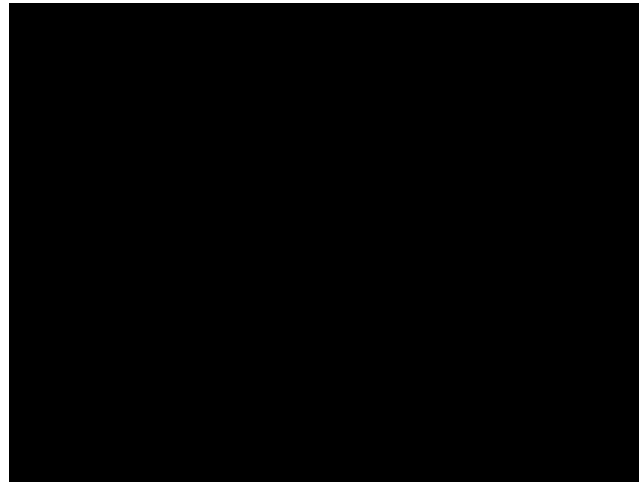
w=1 wahr true High

f =0 falsch false Low

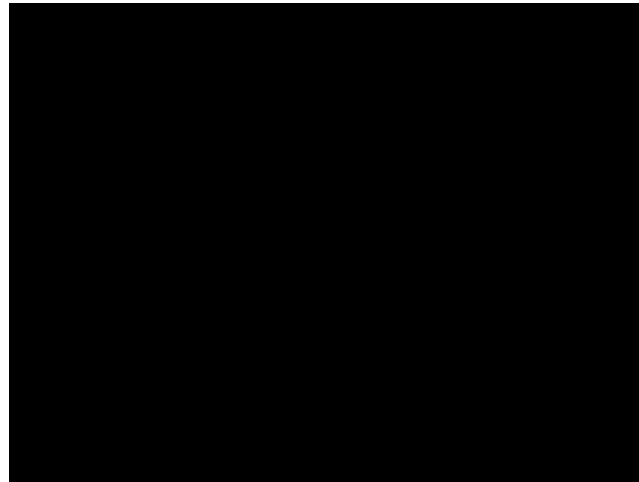
Wahrheitstabelle: Erklärung und Anwendung



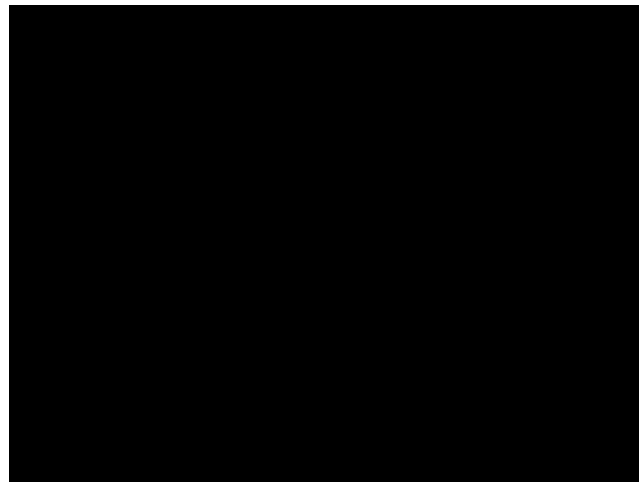
Negation, Konjunktion, Disjunktion – Aussagenlogik 1



Implikation & Äquivalenz



De Morgansche Gesetze – Aussagenlogik 3



Einfache Formallogik

- Sätze der Sprache als Aussagen, von denen es sinnvoll ist zu fragen, ob sie wahr oder falsch sind
- Jede Aussage besitzt eine Negation (Verneinung)
- \bar{A} (NOT())
- Die Und-Verknüpfung ist das „und“
- Die Oder-Verknüpfung wird als \wedge (AND)
- Entweder-Oder-Verknüpfung, auch exklusives Oder \vee (OR)
- Pfeil-Verknüpfung
- Äquivalenz $\underline{\vee}$ (XOR) mit $A \underline{\vee} B = (\bar{A} \wedge B) \vee (A \wedge \bar{B})$

$$A \Rightarrow B$$

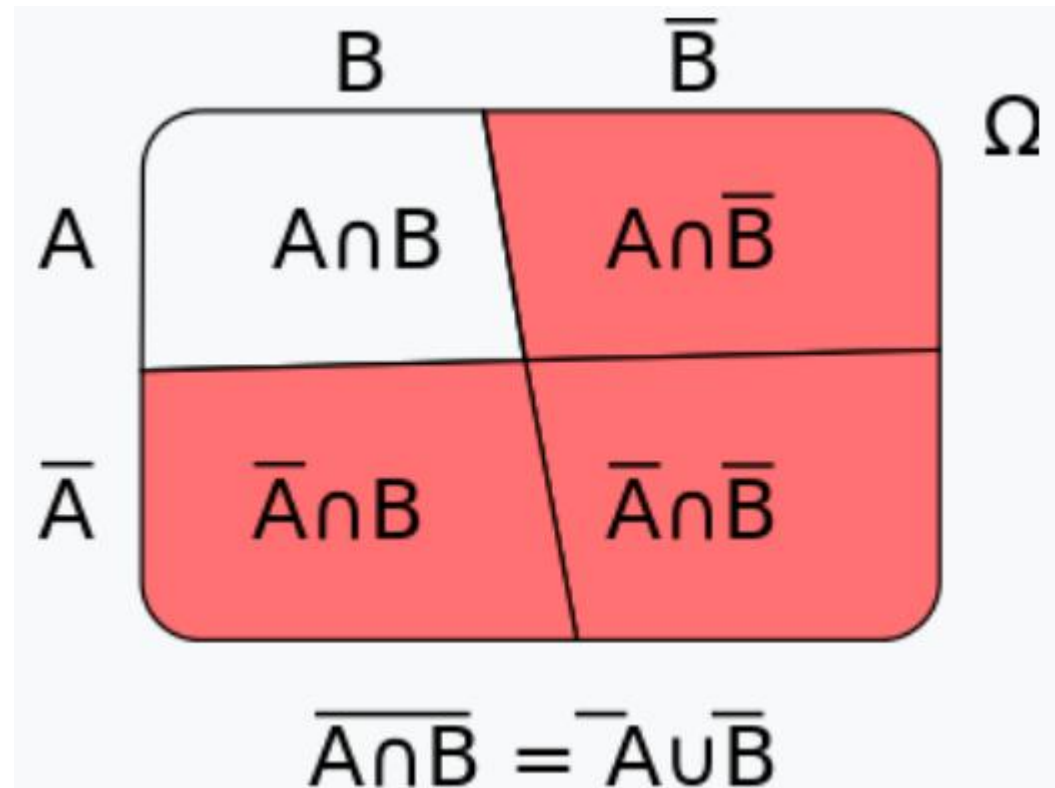
$$A \Leftrightarrow B$$

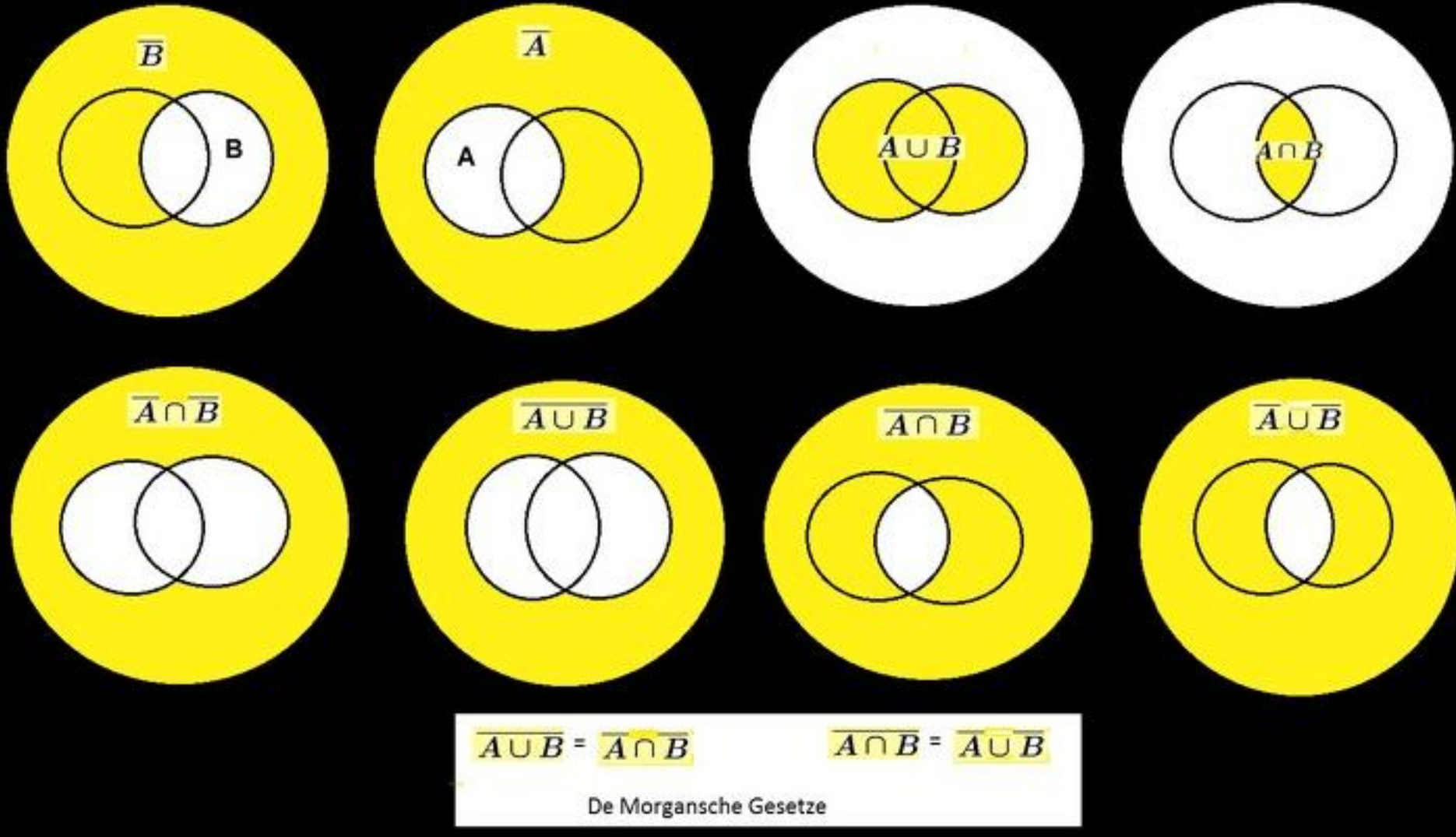
Übung: De Morgan'sche Regeln

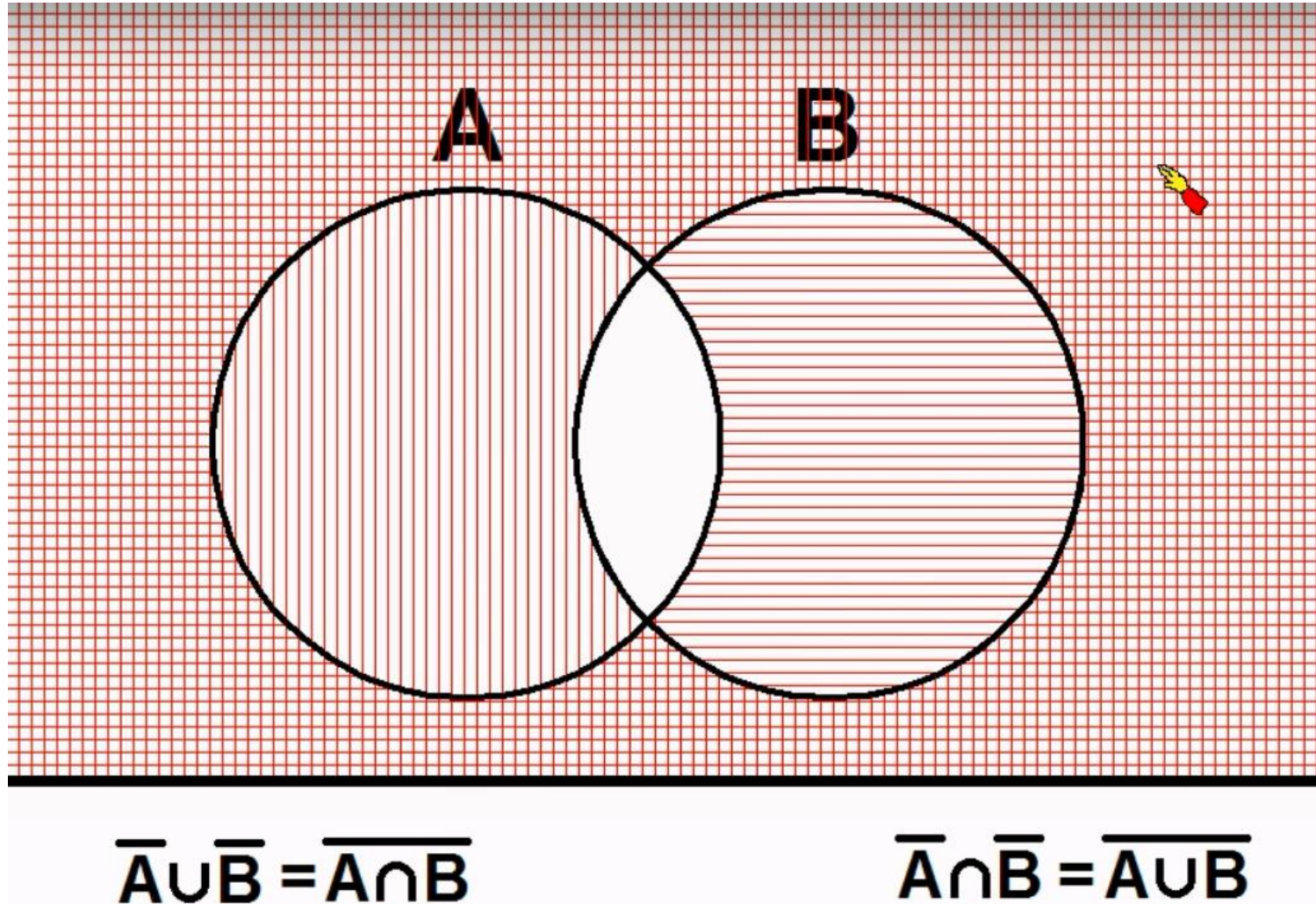
$$\overline{(A \wedge B)} \Leftrightarrow (\overline{A} \vee \overline{B})$$

$$\overline{(A \vee B)} \Leftrightarrow (\overline{A} \wedge \overline{B})$$

^ Vereinigung





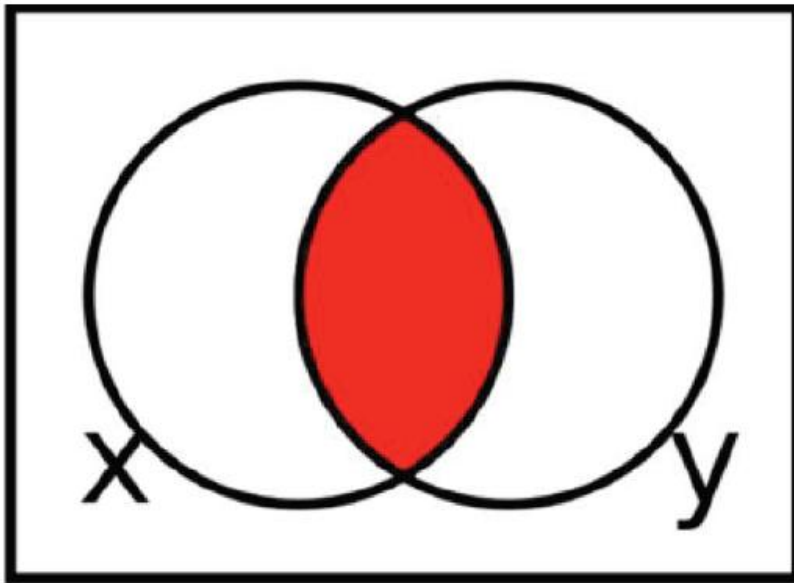


Alles irgendwie Gestreifte

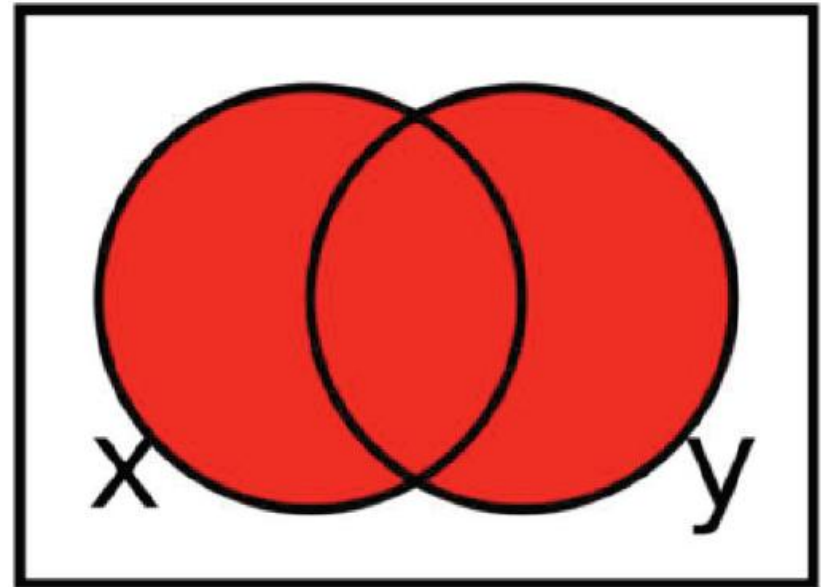
Alles außerhalb beider Kreise

Boolsche Logik

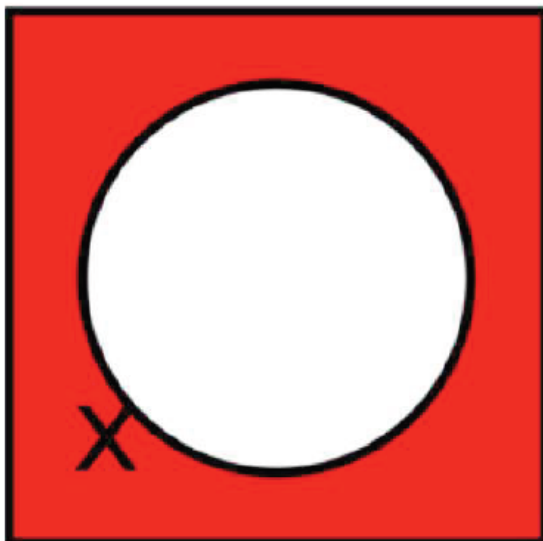
1

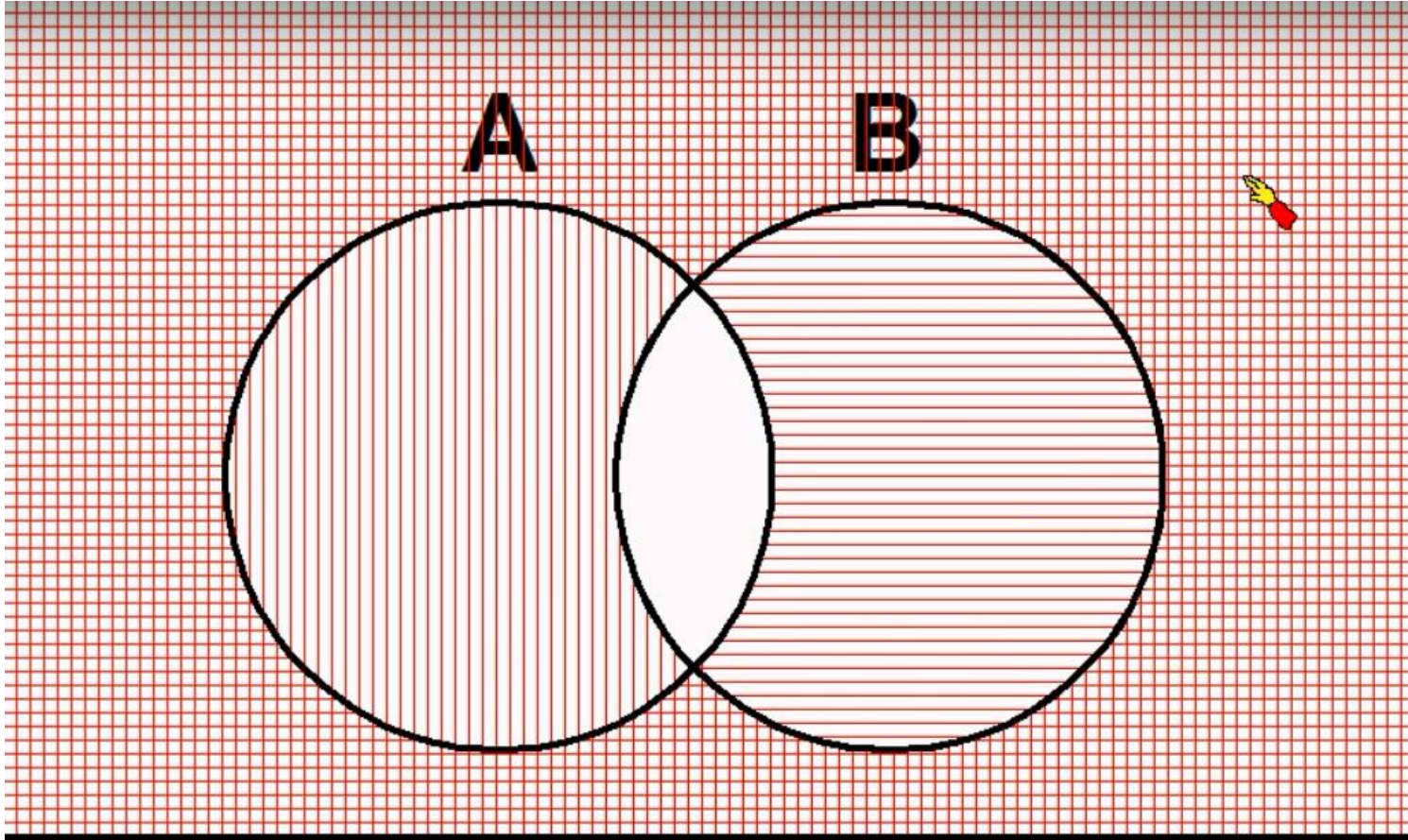


2



3





$$\overline{A \cup B} = \overline{A \cap B}$$

Alles irgendwie Gestreifte

$$\overline{A \cap B} = \overline{A \cup B}$$

Alles außerhalb beider Kreise
(kariert)

NAND und NOR

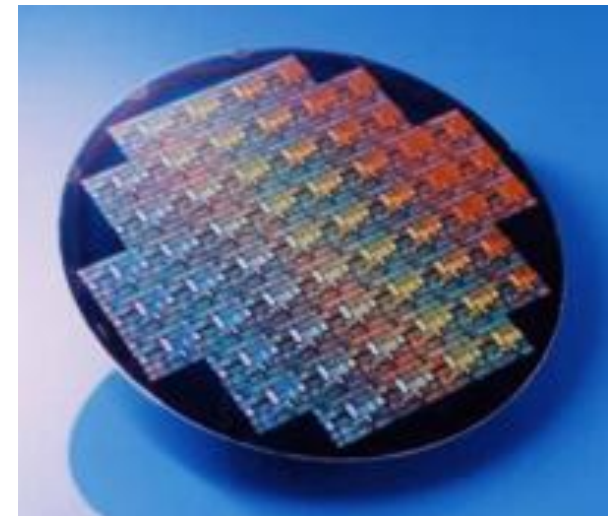
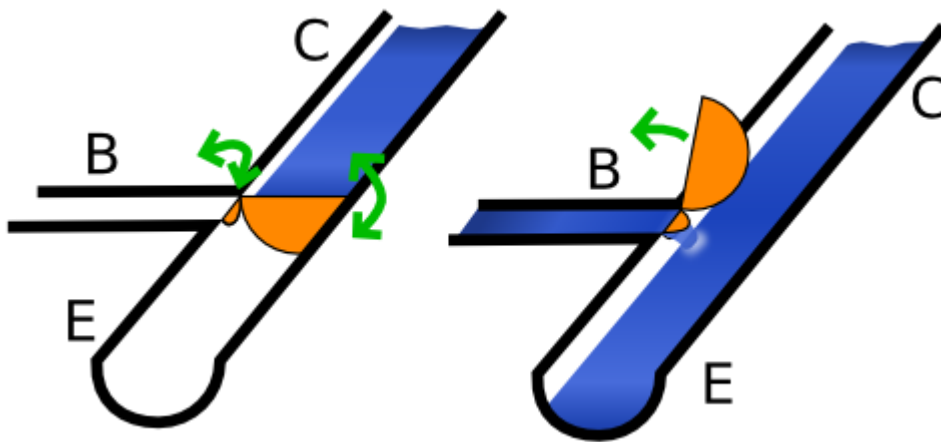
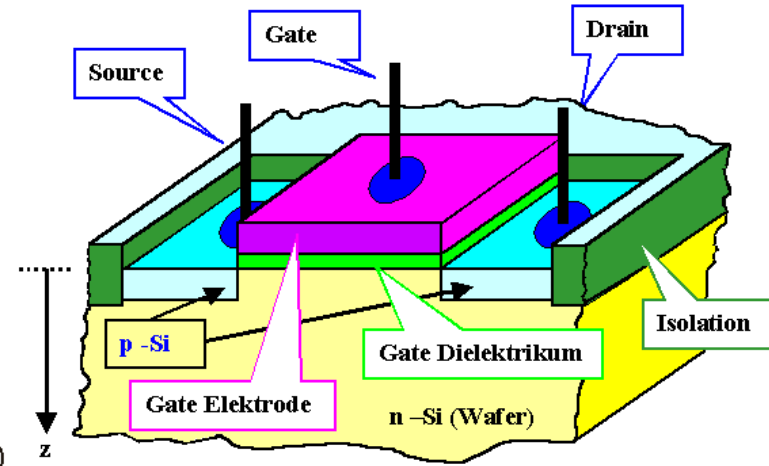
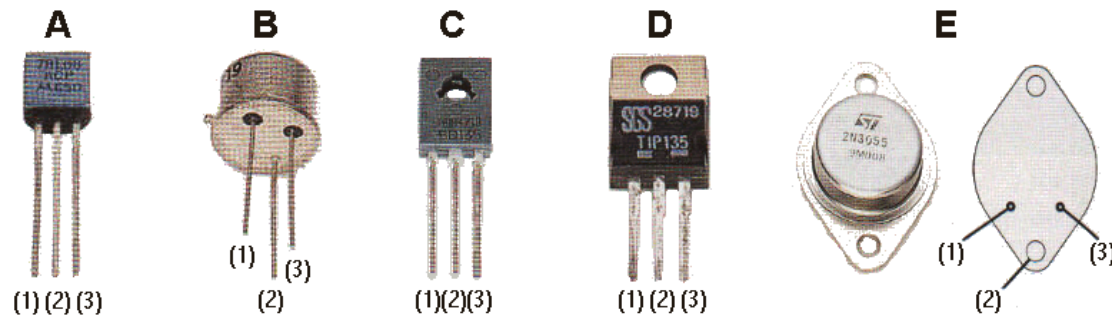
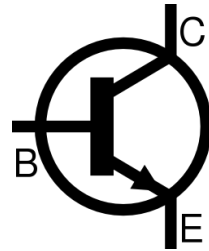
$$\text{NAND} \Leftrightarrow \overline{\wedge} \quad (\overline{A \wedge B}) \Leftrightarrow A \overline{\wedge} B$$

$$\text{NOR} \Leftrightarrow \overline{\vee} \quad (\overline{A \vee B}) \Leftrightarrow A \overline{\vee} B$$

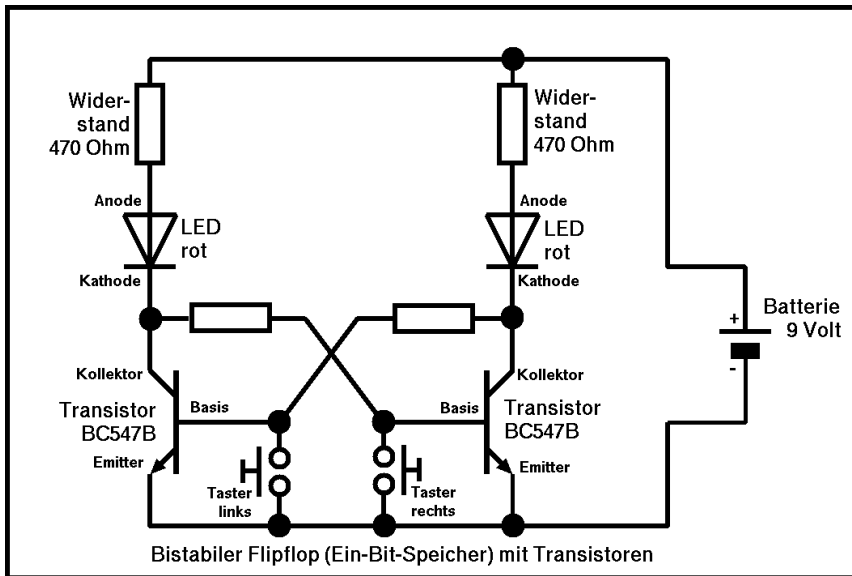
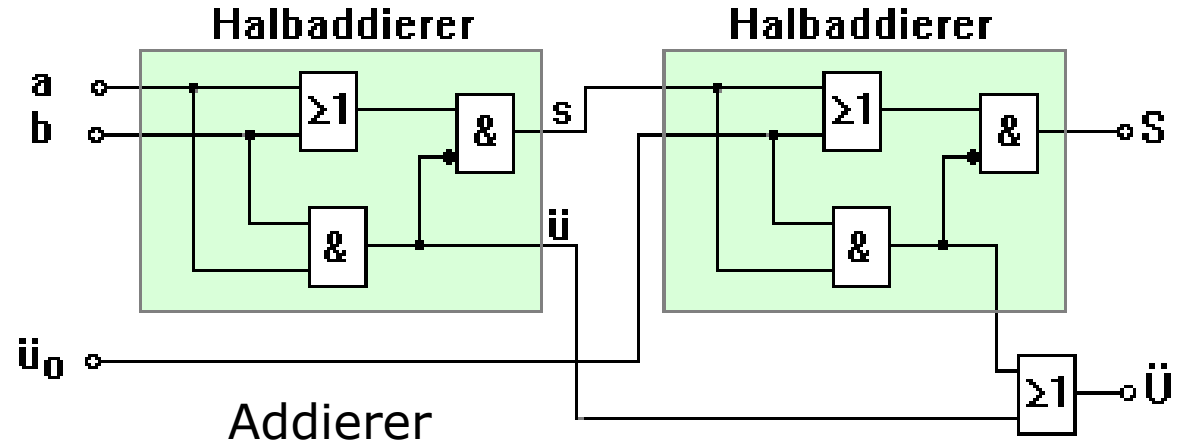
- NAND und NOR sind universelle Operatoren, weil man die anderen grundlegenden logischen Operatoren daraus bilden kann

| Operator Mit NOR $\overline{\vee}$ | | Mit NAND $\overline{\wedge}$ |
|------------------------------------|--|--|
| \overline{A} | $A \overline{\vee} A$ | $A \overline{\wedge} A$ |
| $A \vee B$ | $(A \overline{\vee} B) \overline{\vee} (A \overline{\vee} B)$ | $(A \overline{\wedge} A) \overline{\wedge} (B \overline{\wedge} B)$ |
| $A \wedge B$ | $(A \overline{\vee} A) \overline{\vee} (B \overline{\vee} B)$ | $(A \overline{\wedge} B) \overline{\wedge} (A \overline{\wedge} B)$ |
| $A \underline{\vee} B$ | $[(A \overline{\vee} A) \overline{\vee} (B \overline{\vee} B)] \overline{\vee} (A \overline{\vee} B)$ | $[A \overline{\wedge} (A \overline{\wedge} B)] \overline{\wedge} [B \overline{\wedge} (A \overline{\wedge} B)]$ |
| $A \overline{\vee} B$ | | $[(A \overline{\wedge} A) \overline{\wedge} (B \overline{\wedge} B)] \overline{\wedge}$ $[(A \overline{\wedge} A) \overline{\wedge} (B \overline{\wedge} B)]$ |
| $A \overline{\wedge} B$ | $[(A \overline{\vee} A) \overline{\vee} (B \overline{\vee} B)] \overline{\vee}$ $[(A \overline{\vee} A) \overline{\vee} (B \overline{\vee} B)]$ | |

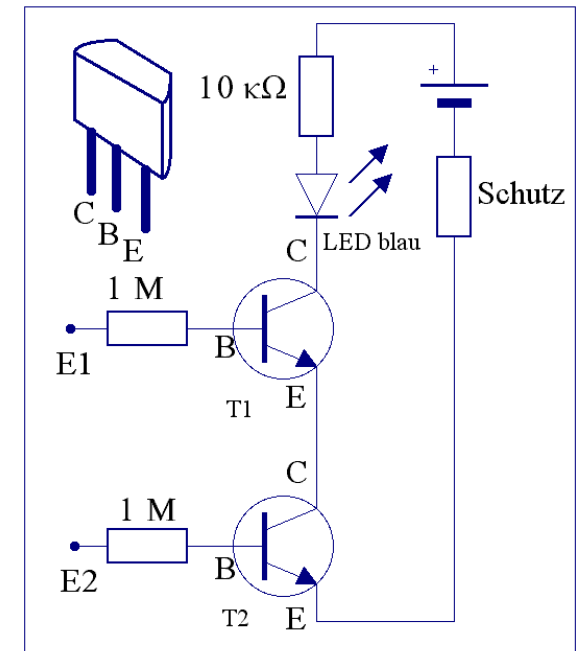
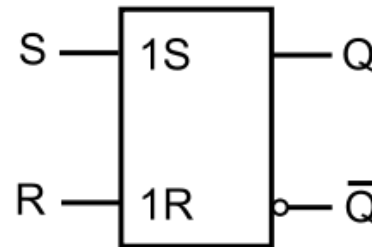
Transistor



Flipflop



Flipflop



AND Schaltung

Schaltwerke: Halbaddierer

Ein **Halbaddierer** (engl. *half adder*) besteht aus zwei Eingängen und zwei Ausgängen. Mit einem Halbaddierer kann man zwei einstellige Binärzahlen addieren. Dabei liefert der Ausgang *s* (engl. *sum* – „Summe“) die rechte und der Ausgang *c* (engl. *carry* – „Übertrag“) die linke Stelle des Ergebnisses.

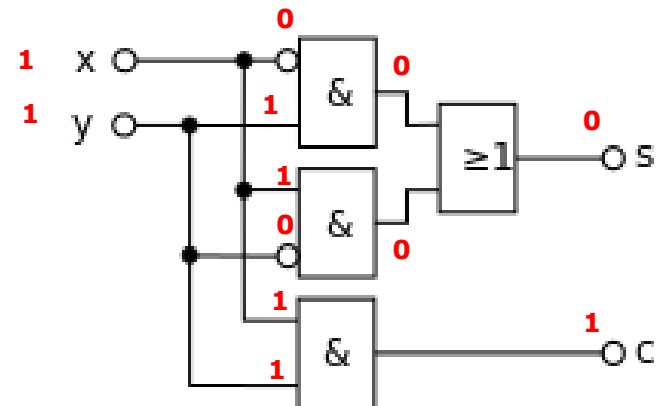
Wahrheitstabelle Halbaddierer:

| x | y | Übertrag c | Summe s |
|---|---|------------|---------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Al

nem zusätzlichen Oder-Gatter

kann ein Volladdierer aufgebaut werden.



Besuch im HTW-Computermuseum

Am Dienstag werden wir in der Nachmittagsveranstaltung das HTW-Computermuseum besuchen. Herr Prof. Dr. Frank Burghardt wird uns hierzu in drei Gruppen für jeweils eine Stunde in Empfang nehmen. Das Computermuseum befindet sich in Raum C610 des HTW-Campus Wilhelmienhofstr.

Folgende Uhrzeiten konnten reserviert werden:

- 13:30-14:30 Gruppe Raum 5011
- 14:30-15:30 Gruppe Raum 5027
- 15:30-16:30 Gruppe Raum 5020

Inside of computer

| | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 10000011 | 00000001 | 00010001 | 00000000 | 00111101 | 11111100 | 01110100 | 00111101 |
| 00000000 | 01000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 10010000 | 00000000 | 00000000 | 00000000 | 01010000 | 00000000 | 00000111 | 00110000 |
| 00001011 | 00000001 | 00001011 | 00000011 | 00001010 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00100000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00100000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 01110000 | 00010000 | 00000000 | 00100000 | 00000001 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00100000 | 00000001 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 01000000 | 00000001 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00100000 | 00000000 | 01000000 | 00000001 | 00000000 | 00000000 | 00000000 |
| 11111111 | 11111111 | 11111111 | 11111111 | 11111111 | 11111111 | 11111111 | 11111111 |
| 10010000 | 10000000 | 00000000 | 01000000 | 00000001 | 00000000 | 00000000 | 00000000 |
| 00101110 | 01100100 | 01111001 | 01101110 | 01100001 | 01101101 | 01101001 | 01100011 |
| 10110000 | 00000100 | 00000000 | 00100000 | 00000001 | 00000000 | 00000000 | 00000000 |
| 10110000 | 00000100 | 00000000 | 00100000 | 00000001 | 00000000 | 00000000 | 00000000 |
| 10100000 | 00000001 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 10110000 | 00000100 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00100000 | 00000000 | 00000000 |

Programme – Daten – Medien

Variablen

- Identifier
 - Namen, die der Programmierer erfindet
 - Buchstaben, Ziffern, "_", keine Ziffer am Anfang
 - case-sensitive
 - Aussagekräftige Namen
- Manche Zeichenketten bereits besetzt
 - Schlüsselworte: **if**, **switch**, **while**, ...
 - vordefinierte Typen: **int**, **float**, ...
- Variablen
 - Container zum Aufbewahren von Werten
 - werden vom Compiler als Speicherplatz angelegt
 - nach dem Programmende (...) verloren

Variablen - Dynamik

- Gültigkeit der Variablen
 - Ort der Vereinbarung
 - im 'Programm' => globale Variable
 - im Block => lokale Variable
 - in der Funktion => lokale Variable
 - entscheidet über Benutzbarkeit
- Lebenszeit von Variablen
 - lokal: in der Funktion - nur während des Funktionsaufrufes (Block)
 - global: während der gesamten Programmausführung
 - static macht auch lokale Variable permanent

- Literale
 - Wert fest (Konstante)
 - für Vergleiche, Initialisierung
 - Zahlen, Buchstaben, String
- `index = index + 1;`
- `if (zeichen > 'Z') ...;`
 `zeichen = zeichen + 32; /* macht Kleinbuchst. */`
- `pi = 3.1415926;`
- `mpi = -3.1415926;`

Variablendeklaration mit Vorbesetzung

- `Typ" " <Name> = <Ausdruck>;`
 - `int i = 12;`
 - `double pi = 3.1415926;`
 - `char space = ' ';`

Systeme zur Darstellung von Zahlen

Dezimalsystem (Basis = 10):
 $1850_{10} = 1 \cdot 10^3 + 8 \cdot 10^2 + 5 \cdot 10^1 + 0 \cdot 10^0$

Dualsystem (Basis = 2):
 $11100111010_2 =$
 $1 \cdot 2^{10} + 1 \cdot 2^9 + 1 \cdot 2^8 + 0 \cdot 2^7 + 0 \cdot 2^6$
 $+ 1 \cdot 2^5$
 $+ 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$
 $= 1850_{10}$

Allgemeine Darstellung einer n-stelligen Zahl Z bezogen auf die Basis b mit Exponent (Index i gibt die Stelle an):

$$Z_b = Z_{n-1} \cdot b^{n-1} + Z_{n-2} \cdot b^{n-2} + \dots (+ Z_i \cdot b^i \dots + Z_1 \cdot b^1 + Z_0 \cdot b^0) \sum_{i=0}^{n-1} z_i b^i$$

kurz $\sum_{i=0}^{n-1}$

Oktalsystem (Basis = 8):
 $3472_8 = 3 \cdot 8^3 + 4 \cdot 8^2 + 7 \cdot 8^1 + 2 \cdot 8^0$
 $= 1850_{10}$

Hexadezimalsystem (Basis = 16):
 $73A_{16} = 7 \cdot 16^2 + 3 \cdot 16^1 + 10 \cdot 16^0$
 $= 1850_{10}$

Römische Zahlen

I=1,V=5,X=10,L=50,C=100,D=500,M=1000

I,X und C nicht mehr als dreimal nebeneinander

VI =6=5+1

XIII =13=10+1+1+1

IV =4=5-1

MIM = (MCMIC=1999;MCMXCIX=1999)

| | | | | | | | |
|------|---|--|------|----|--|------|------|
| I | 1 | | X | 10 | | C | 100 |
| II | 2 | | XX | 20 | | CC | 200 |
| III | 3 | | XXX | 30 | | CCC | 300 |
| IV | 4 | | XL | 40 | | CD | 400 |
| V | 5 | | L | 50 | | D | 500 |
| VI | 6 | | LX | 60 | | DC | 600 |
| VII | 7 | | LXX | 70 | | DCC | 700 |
| VIII | 8 | | LXXX | 80 | | DCCC | 800 |
| IX | 9 | | XC | 90 | | CM | 900 |
| | | | | | | M | 1000 |

Zahlensysteme

| dezimal | binär | oktal | hex |
|---------|-------|-------|-----|
| 0 | 0000 | 00 | 0 |
| 1 | 0001 | 01 | 1 |
| 2 | 0010 | 02 | 2 |
| 3 | 0011 | 03 | 3 |
| 4 | 0100 | 04 | 4 |
| 5 | 0101 | 05 | 5 |
| 6 | 0110 | 06 | 6 |
| 7 | 0111 | 07 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

Rechnerinternen Zahlendarstellung



Darstellung ganzer Zahlen (Integer)

Beispiel: 16 Bit Zahlen

positive ganze 16-Bit-Zahlen

0000 0000 0000 0000 = 0

0000 0000 0000 0001 = 1

0000 0000 0000 0010 = 2

1111 1111 1111 1111 = 65 535

Wertebereich 0 ... 65.535

positive und negative ganze 16-Bit-Zahlen

0111 1111 1111 1111 = 32 767

0000 0000 0000 0001 = 1

0000 0000 0000 0000 = 0

1000 0000 0000 0001 = -1

1000 0000 0000 0010 = -2

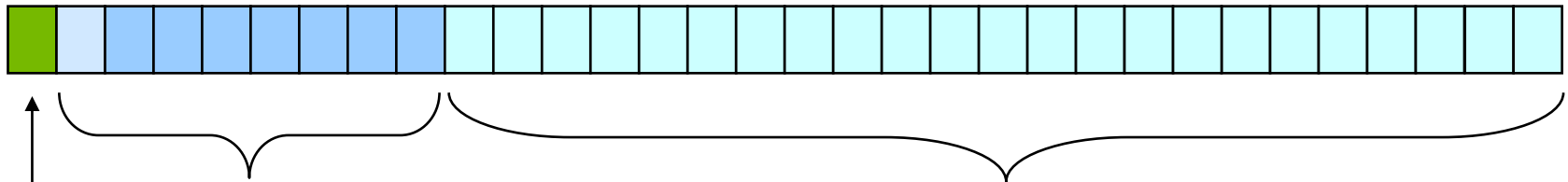
1111 1111 1111 1111 = -32 768

Anzahl darstellbarer Werte
gleich, aber:

- positiver Wertebereich: 0 ... 32.767
- negativer Wertebereich: -1 ... 32.768

Interne Darstellung von Gleitkommazahlen nach IEEE Standard 754

Darstellung von Gleitkommazahlen als Single Typ (= 32 Bit)



8 Bit für Exponent ($p = 8$)

23 Bit für Mantisse ($m = 23$)

$s = 0(+)$ oder $1(-)$

$m = 23$

$p = 8$

Bias berechnen:

Kleinster Betrag:

Größter Betrag:

$$2,997.924.58 \cdot 10^8$$

jede Zahl z lässt sich in der
Form $z = m \cdot b^e$ darstellen.

m Mantisse,

b Basis der Zahlen-darstellung

e Exponent.

Float und Double (reelle Zahlen)

$$\begin{aligned}(11.011)_2 &= 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} \\ &= 2 + 1 + 0 \cdot 0.5 + 0.25 + 0.125 = (3.375)_{10}\end{aligned}$$

Die Dezimalzahl

$$17.625 = 1 \cdot 10^1 + 7 \cdot 10^0 + 6 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3}$$

entspricht der binären Zahl:

$$\begin{aligned}&16 + 1 + 1/2 + 1/8 \\ &= 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 10001.101 \cdot 2^0\end{aligned}$$

Die entsprechende normalisierte Form erhält man, indem man den Dezimalpunkt hinter die erste signifikante Ziffer „schiebt“ und den Exponenten entsprechend anpasst:

$$1.0001101 \cdot 2^4$$

$$2.3756 \cdot 10^3$$

Mantisse(2.3756) und Exponent(3), der ganzzahlig ist

Aufbau von Nachrichten: Nachrichten und Nachrichtenraum

Definition Nachricht:

Eine Nachricht ist eine aus den Zeichen eines Alphabets gebildete Zeichenfolge. Diese Zeichenfolge muss nicht endlich sein, aber abzählbar, d.h. man muss die einzelnen Zeichen durch Abbildung auf die natürlichen Zahlen durchnummerieren können, damit die Identifizierbarkeit der Zeichen sichergestellt ist.

Definition Nachrichtenraum:

Die Menge aller Nachrichten, die mit den Zeichen eines Alphabets A gebildet werden können, heißt Nachrichtenraum $N(A)$ oder A^* über A .

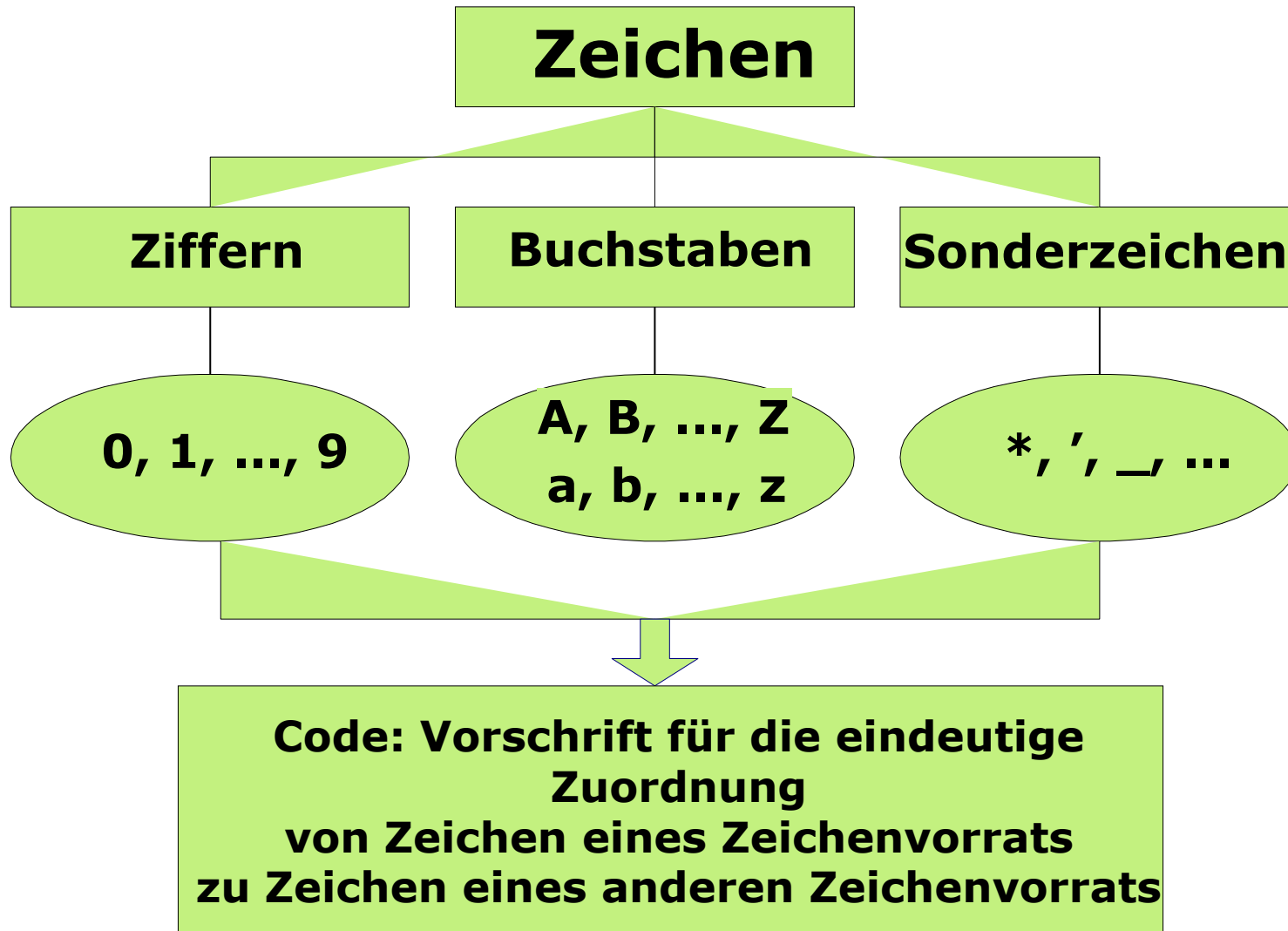
Aufbau von Nachrichten: Codierung

Für die Informatik wichtig: binäres Alphabet $\{0,1\}$

Wie kann man die Buchstaben des deutschen Alphabets binär codieren?

| | | |
|------|----------|-----|
| 0001 | bedeutet | a |
| 0010 | bedeutet | b |
| 0011 | bedeutet | c |
| 0100 | bedeutet | d |
| 0101 | bedeutet | e |
| ... | ... | ... |

Darstellung von Zeichen



ASCII Code

Ziffer als numerischer Wert

5: 00000101 (Dezimal5)

Ziffer als ASCII-Code

'5': 00110101 (Dezimal53)

ASCII code nutzt nur 128 Zeichen

ASCII -Zeichen

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- 7-Bit-Code (ISO 646)
- Codiert Zeichen und Steuercodes
- Problem: viele wichtige Zeichen fehlen
- Lösung: der Code wird erweitert
 - Bekannteste Erweiterung ISO-8859-1, genannt Latin 1
 - Enthält (ä,ö,...) und Sonderzeichen für westeuropäische Sprachen.
 - Unicode, UTF-8
- Mehrere Bytes werden zu einem (Maschinen-)Wort zusammengefasst (Wortlänge des Rechners)
 4byte = 32bit-Rechner
 8byte = 64bit-Rechner

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|---|-----|---|-----|----|-----|---|-----|---|-----|---|
| 0 | | 32 | | 64 | Q | 96 | ` | 128 | Ç | 160 | á | 192 | À | 224 | α |
| 1 | ␣ | 33 | ! | 65 | A | 97 | a | 129 | ü | 161 | í | 193 | Á | 225 | β |
| 2 | ␣ | 34 | " | 66 | B | 98 | b | 130 | é | 162 | ó | 194 | Â | 226 | γ |
| 3 | ♥ | 35 | # | 67 | C | 99 | c | 131 | â | 163 | ú | 195 | Ã | 227 | δ |
| 4 | ♦ | 36 | \$ | 68 | D | 100 | d | 132 | ä | 164 | ñ | 196 | Ä | 228 | Σ |
| 5 | ♣ | 37 | % | 69 | E | 101 | e | 133 | à | 165 | ñ | 197 | Å | 229 | σ |
| 6 | ♣ | 38 | & | 70 | F | 102 | f | 134 | å | 166 | ª | 198 | Æ | 230 | ρ |
| 7 | • | 39 | ' | 71 | G | 103 | g | 135 | ç | 167 | º | 199 | ⦿ | 231 | τ |
| 8 | ◼ | 40 | (| 72 | H | 104 | h | 136 | ê | 168 | ¿ | 200 | ⦿ | 232 | ϑ |
| 9 | ◊ | 41 |) | 73 | I | 105 | i | 137 | ë | 169 | ¬ | 201 | ⦿ | 233 | θ |
| 10 | ◼ | 42 | * | 74 | J | 106 | j | 138 | è | 170 | ¬ | 202 | ⦿ | 234 | Ω |
| 11 | ♠ | 43 | + | 75 | K | 107 | k | 139 | í | 171 | ½ | 203 | ⦿ | 235 | δ |
| 12 | ♀ | 44 | , | 76 | L | 108 | l | 140 | î | 172 | ¾ | 204 | ⦿ | 236 | • |
| 13 | ♠ | 45 | - | 77 | M | 109 | m | 141 | ï | 173 | ¿ | 205 | ⦿ | 237 | • |
| 14 | ♠ | 46 | . | 78 | N | 110 | n | 142 | ä | 174 | « | 206 | ⦿ | 238 | € |
| 15 | ♠ | 47 | / | 79 | O | 111 | o | 143 | å | 175 | » | 207 | ⦿ | 239 | π |
| 16 | ♠ | 48 | 0 | 80 | P | 112 | p | 144 | É | 176 | ⦿ | 208 | ⦿ | 240 | ≡ |
| 17 | ♠ | 49 | 1 | 81 | Q | 113 | q | 145 | æ | 177 | ⦿ | 209 | ⦿ | 241 | ± |
| 18 | ♠ | 50 | 2 | 82 | R | 114 | r | 146 | ff | 178 | ⦿ | 210 | ⦿ | 242 | ≥ |
| 19 | !! | 51 | 3 | 83 | S | 115 | s | 147 | ð | 179 | | 211 | ⦿ | 243 | ≤ |
| 20 | ♠ | 52 | 4 | 84 | T | 116 | t | 148 | ö | 180 | | 212 | ⦿ | 244 | ∫ |
| 21 | ♠ | 53 | 5 | 85 | U | 117 | u | 149 | ò | 181 | | 213 | ⦿ | 245 | J |
| 22 | — | 54 | 6 | 86 | V | 118 | v | 150 | û | 182 | | 214 | ⦿ | 246 | ÷ |
| 23 | ± | 55 | 7 | 87 | W | 119 | w | 151 | ù | 183 | ⦿ | 215 | ⦿ | 247 | ≡ |
| 24 | ↑ | 56 | 8 | 88 | X | 120 | x | 152 | ÿ | 184 | | 216 | ⦿ | 248 | • |
| 25 | ↓ | 57 | 9 | 89 | Y | 121 | y | 153 | ÿ | 185 | ⦿ | 217 | ⦿ | 249 | · |
| 26 | → | 58 | : | 90 | Z | 122 | z | 154 | Ü | 186 | ⦿ | 218 | ⦿ | 250 | · |
| 27 | ← | 59 | ; | 91 | [| 123 | { | 155 | ç | 187 | ⦿ | 219 | ⦿ | 251 | J |
| 28 | ↶ | 60 | < | 92 | \ | 124 | | 156 | £ | 188 | ⦿ | 220 | ⦿ | 252 | π |
| 29 | ↷ | 61 | = | 93 |] | 125 | } | 157 | ¥ | 189 | ⦿ | 221 | ⦿ | 253 | z |
| 30 | ▲ | 62 | > | 94 | ^ | 126 | ~ | 158 | ℞ | 190 | ⦿ | 222 | ⦿ | 254 | ■ |
| 31 | ▼ | 63 | ? | 95 | _ | 127 | △ | 159 | f | 191 | ⦿ | 223 | ⦿ | 255 | |

Codetabelle ASCII

| ASCII-Codetabelle | | | | | | | | | | |
|-------------------|---|---|---|---|---|---|----|---|---|---|
| + | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 30 | | | | ! | " | # | \$ | % | & | ' |
| 40 | (|) | * | + | , | - | . | / | 0 | 1 |
| 50 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; |
| 60 | < | = | > | ? | @ | A | B | C | D | E |
| 70 | F | G | H | I | J | K | L | M | N | O |
| 80 | P | Q | R | S | T | U | V | W | X | Y |
| 90 | Z | [| \ |] | ^ | _ | ` | a | b | c |
| 100 | d | e | f | g | h | i | j | k | l | m |
| 110 | n | o | p | q | r | s | t | u | v | w |
| 120 | x | y | z | { | | } | ~ | | | |

ASCII codes

(Abk. f. "american standard code for information interchange,,)

7-Bit-Code, ordnet kleinen und großen Buchstaben, Zahlen und einigen

Sonderzeichen jeweils eine Zahl zu. Die ersten 32 Codes sind Steuerzeichen für Zeilenvorschub usw. vorbehalten.

dezimal: „K“ = 70+5 hex: „K“ == 4B

Daten – Datentypen - ASCII

American **S**tandard **C**ode for **I**nformation **I**nterchange

‘HTW’ ⇒

| | | |
|---|----|-----------|
| ‘ | 39 | 0 0100111 |
| H | 72 | 0 1001000 |
| T | 84 | 0 1010100 |
| W | 87 | 0 1010111 |
| ‘ | 39 | 0 0100111 |

Paritäts-
Bit



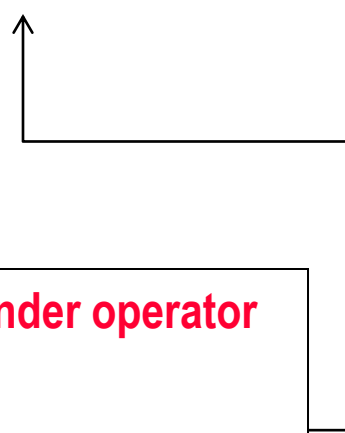
| | | |
|------|----------|-----|
| 0001 | bedeutet | a |
| 0010 | bedeutet | b |
| 0011 | bedeutet | c |
| 0100 | bedeutet | d |
| 0101 | bedeutet | e |
| ... | ... | ... |

Example: Caesar cipher

- for simplicity, we'll assume the message is made of lowercase letters only

```
ALPHABET = "abcdefghijklmnopqrstuvwxyz"

def caesar(word):
    copy = ""
    for ch in word:
        index = ALPHABET.find(ch)
        nextIndex = (index + 3) % 26
        copy += ALPHABET[nextIndex]
    return copy
```



wrap-around is handled using the remainder operator

for the letter "z", index = 25
nextIndex = $(25+3)\%26 = 28\%26 = 2$
so, "z" → "c"

Unicode

Unicode ist ein internationaler Standard, in dem langfristig für jedes sinntragende Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt wird. Ziel ist es, die Verwendung unterschiedlicher und inkompatibler Kodierungen in verschiedenen Ländern oder Kulturkreisen zu beseitigen. Dies ermöglicht einen internationalen Datenaustausch ohne Kompatibilitätsprobleme.

- Neuer Standard
- Es besteht eine direkte Kompatibilität mit dem ASCII-Code: Dieser bildet eine Untermenge (erste 256 Zeichen).
- Enthält 1:1 Entsprechungen wichtiger Industriezeichensätze (ISO-Normen)
- Wird verwaltet vom Unicode-Konsortium (<http://www.unicode.de>)
- Unterstützt verschiedene Codierungsformate (**U**nicode **T**ransformation **F**ormat) UTF-8, UTF-16
- Speicherung und Übertragung erfolgt im Internet und bei den meisten Betriebssystemen im UTF-8-Format
- Es können 1,1 Mio Zeichen im Unicode codiert werden.

Ohne Unicode-Installation

Mojibake (Zeichensalat) der japanischen Wikipedia unter Windows, wenn kein Unicode installiert ist:



Arabisch mit Unicode-Installation

إنشاء حساب

بحث

اعرض التاريخ

عدل

اقرأ

مقالة



ويكيبيديا
الموسوعة الحرة

الصفحة الرئيسية

الأحداث الجارية

أحدث التغييرات

أحدث التغييرات الأساسية

تصفح

المواضيع

أبجدي

بوابات

مقالة عشوائية

مشاركة

طباعة

أدوات

لغات

يونيكود [عدل]

من ويكيبيديا، الموسوعة الحرة

في علم الحاسوب، الترميز الموحد (يونيكود^[1] أو يُونُكُود^[2]) معيار يمكن الحواسيب من تمثيل النصوص المكتوبة بأغلب نظم الكتابة ومعالجتها، بصورة متناسقة. يتكون يونيكود من 100,000 محرف، وطُفم من مخططات الرموز كمرجع مرئي، ونهج في الترميز، وطُفم من ترميزات المحارف المعيارية، وسرد لخصائص المحارف، وطُفم من البيانات المرجعية، وعدد من الأمور المتعلقة مثل خصائص المحارف، وقواعد تطبيق النص، وفك الحروف لوحدها الأولية، والترتيب، والتصيير، وثنائية الاتجاه (لعرض النصوص الذي يحتوي على كتابات من اليمين لليسار، مثل العربية، مع كتابات من اليسار لليمين، مثل اللاتينية).^[3] يطور يونيكود بالتوازي مع معيار طُفم المحارف العالمي، وينشر على شكل كتاب يحمل الاسم معيار يونيكود (Unicode Standard).

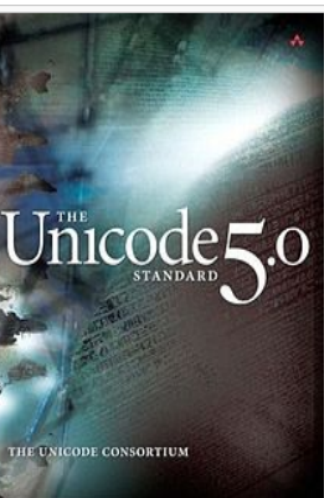
يطمح مجمع يونيكود -المنظمة غير الربحية التي تنسق تطوير يونيكود- في النهاية إلى استبدال ترميزات المحارف الموجودة حالياً، ليحل محلها يونيكود وتنسيق يونيكود المعياري للتحويل (Unicode Transformation Format، UTF)، حيث أن الكثير من الترميزات الحالية محدودة السعة والمدى، ولا تتوافق مع البيانات متعددة اللغات.

أدى نجاح يونيكود في توحيد أطقم المحارف إلى انتشاره وغلبيه استخدامه في توطين وعولمة برمجيات الحاسوب. وجرى تطبيق البرنامج في العديد من التقنيات الحديثة، مثل XML، ولغة البرمجة جافا وأنظمة التشغيل الحديثة.

محتويات [أخف]

1 الأصل والتطور

1.1 المعيار



معيار نظام الحروف الدولي الموحد، نسخة 5.0

pedia.org/wiki/ملف:Unicodeconsortium_bookv5.jpg

Emojis

Unicode Codierung:

<https://unicode.org/emoji/charts/full-emoji-list.html>



<https://emojipedia.org/slack/>

Basis-Datentypen

char: Menge der Zeichen

int: Menge der ganzen Zahlen, die im Rechner darstellbar sind

float: Menge der darstellbaren Gleitkommazahlen mit einfacher Genauigkeit,

double: Menge der darstellbaren Gleitkommazahlen mit doppelter Genauigkeit,

Array: Zusammenfassung von zusammengehörigen Daten des gleichen Typs

String: Array von Zeichen

Arrays

Speicherung großer Datenmengen vom gleichen Datentyp

Feste Länge (Anzahl der Elemente)
die Reihenfolge der Daten spielt eine Rolle spielt
ein Element über den Index ansprechen

- `int led[] = {4,5,6,7}`
- `int led2[7]`
- `person teilnehmer [30]`

Vorteile?

Datenbehälter – Container – Collection

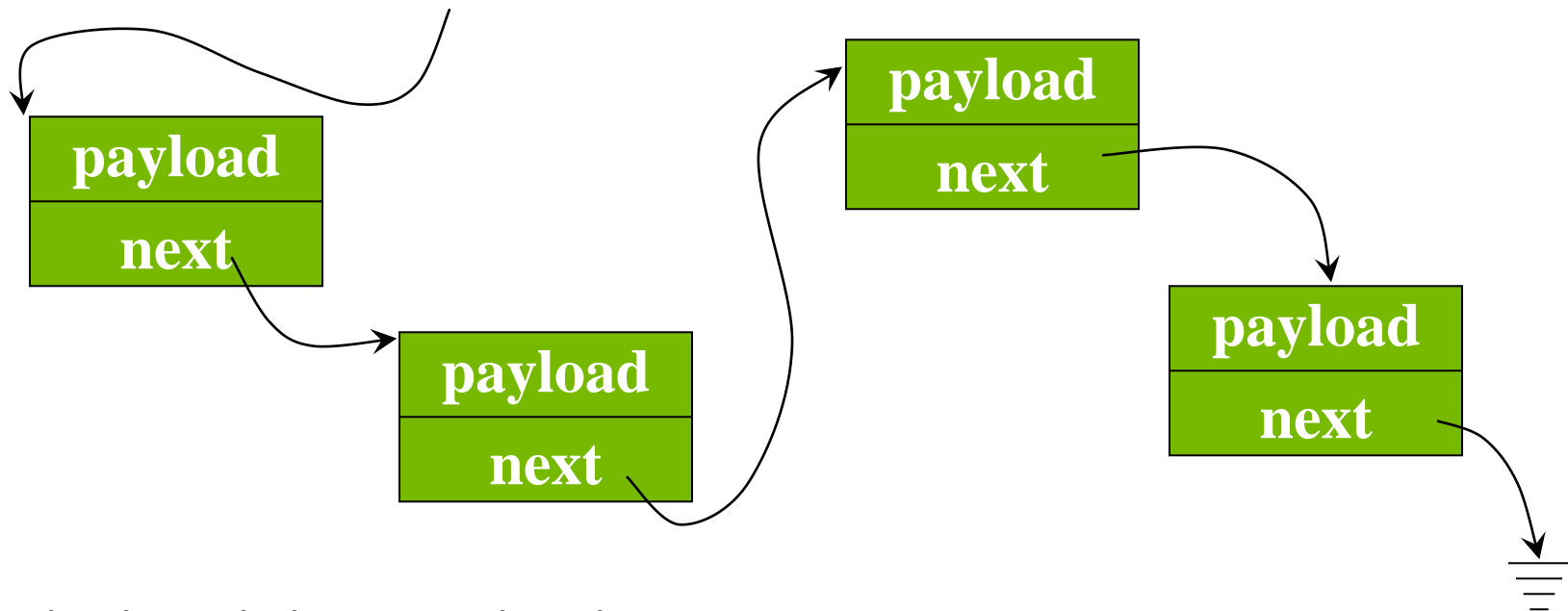
- **Schlange** (queue) First In First Out: FIFO
Druckerwarteschlange, Tastaturpuffer, ...
- **Keller** (Stack, Stapel) Last In First Out : LIFO
Methoden-Stack, Compilieren von Programmen, ...
- **Tabelle** (table) Zugriff über Schlüssel
Datenbanken
- **Baum** (tree) Hierarchie
Datenbanken, Suchen, Sortieren, ...
- **Liste** (list, Folge) vor-zurück-anfang-ende
Zeileneditor, Grundstruktur für Container
- **Menge** (set) jedes Element nur einmal

Verkettete Listen

- Folge von Elementen, die dynamisch während des Programmablaufs verlängert bzw. verkürzt werden kann
- Cursor oder Pointer
- Ausgeben einer Liste, Einfügen eines Elements in die Liste und Löschen eines Elements aus der Liste
- Items (Elemente) der Liste haben üblicherweise den gleichen Typ

Linked List (continued)

```
struct listItem {
    type payload;
    struct listItem *next;
};
struct listItem *head;
```



Payload ~ Inhalt, Daten des Elements

Next = Pointer, Zeiger auf anderes Element ~ Speicheradresse

Data Structures — Lists and Trees

Suchen eines Elementes

Eingabe: Verkettete Liste *liste*,
Index *i* des gewünschten Elements

Ausgabe: Das Element an Index *i*

```
aktuellesElement := liste.first()
```

```
j := 0
```

```
Wiederhole solange j < i
```

```
aktuellesElement := aktuellesElement.next()
```

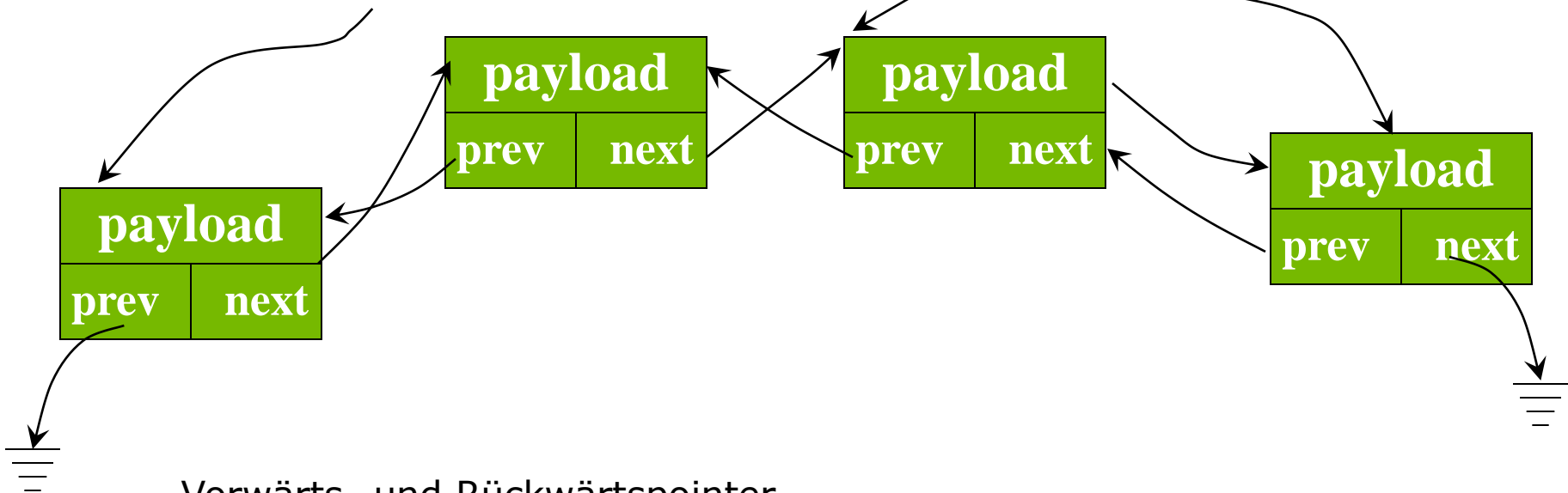
```
j := j + 1
```

```
Return aktuellesElement
```


Doubly-Linked List (doppelt verkettet)

```
struct listItem {
    type payload;
    listItem *prev;
    listItem *next;
};
struct listItem *head, *tail;
```

In-class exercise:— how to add
a new item *q* after a list item *p*



Vorwärts- und Rückwärtspointer

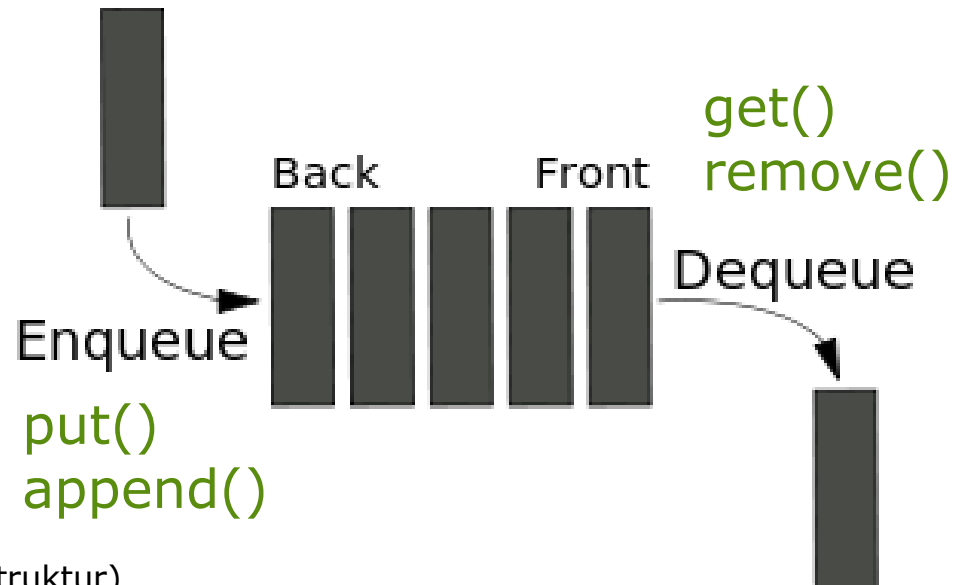
Data Structures — Lists and Trees

Anwendung: Queue (Warteschlange)

put(): fügt ein Element am Ende der Warteschlange hinzu.

get(): entnimmt ein Element am Anfang der Warteschlange und liefert es zurück.

Queues arbeiten nach dem FIFO-Prinzip
(first-in-first-out)



[https://de.wikipedia.org/wiki/Warteschlange_\(Datenstruktur\)](https://de.wikipedia.org/wiki/Warteschlange_(Datenstruktur))

Sortierte Listen



http://www.rolandschule-oberhausen.de/Bilder/fotosPWFrueher_Heute_2014.php

| Symbol ▾ | Aufgabe ▾ | Status ▾ |
|----------|------------|----------------|
| ● | Aufgabe 01 | In Bearbeitung |
| ○ | Aufgabe 02 | Offen |
| ✓ | Aufgabe 03 | Fertig |
| ✓ | Aufgabe 04 | Fertig |
| ● | Aufgabe 05 | In Bearbeitung |
| ○ | Aufgabe 06 | Offen |
| 🚩 | Aufgabe 07 | Dringend |
| 🚩 | Aufgabe 08 | Dringend |
| ● | Aufgabe 09 | In Bearbeitung |
| ○ | Aufgabe 10 | Offen |

↑ *Unsortiert*

| Symbol ▾ | Aufgabe ▾ | Status ▾ |
|----------|------------|----------------|
| 🚩 | Aufgabe 07 | Dringend |
| 🚩 | Aufgabe 08 | Dringend |
| ● | Aufgabe 01 | In Bearbeitung |
| ● | Aufgabe 05 | In Bearbeitung |
| ● | Aufgabe 09 | In Bearbeitung |
| ○ | Aufgabe 02 | Offen |
| ○ | Aufgabe 06 | Offen |
| ○ | Aufgabe 10 | Offen |
| ✓ | Aufgabe 03 | Fertig |
| ✓ | Aufgabe 04 | Fertig |

↑ *Sortiert*

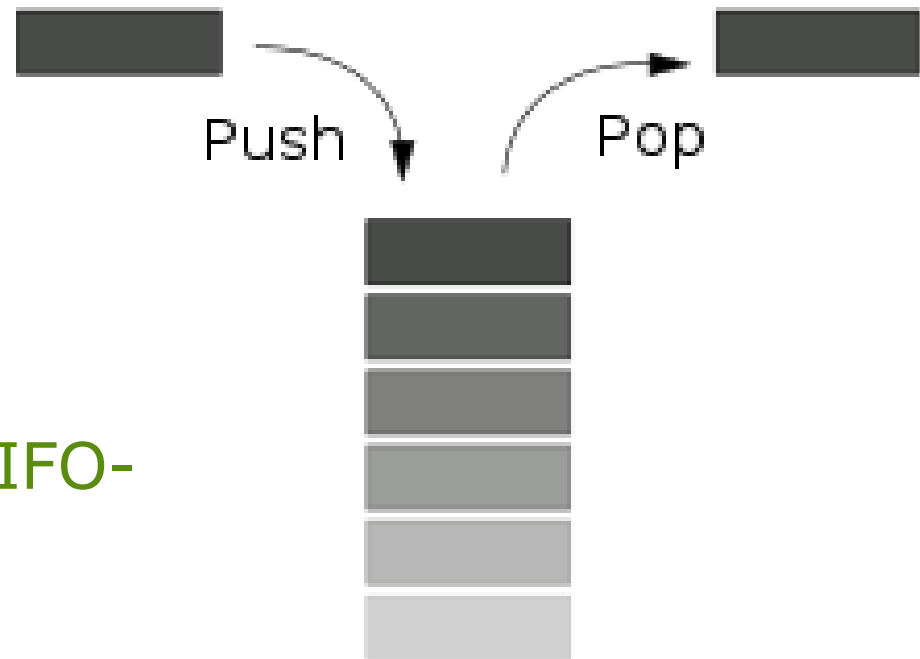
<http://www.huegemann-informatik.de/blog/excel/schnelle-statusuebersicht-listen-nach-symbolen-sortieren>

Stack (Stapel)

push():legt ein Element an oberster Stelle auf dem Stack ab

pop() :entfernt das oberste Element aus dem Stack

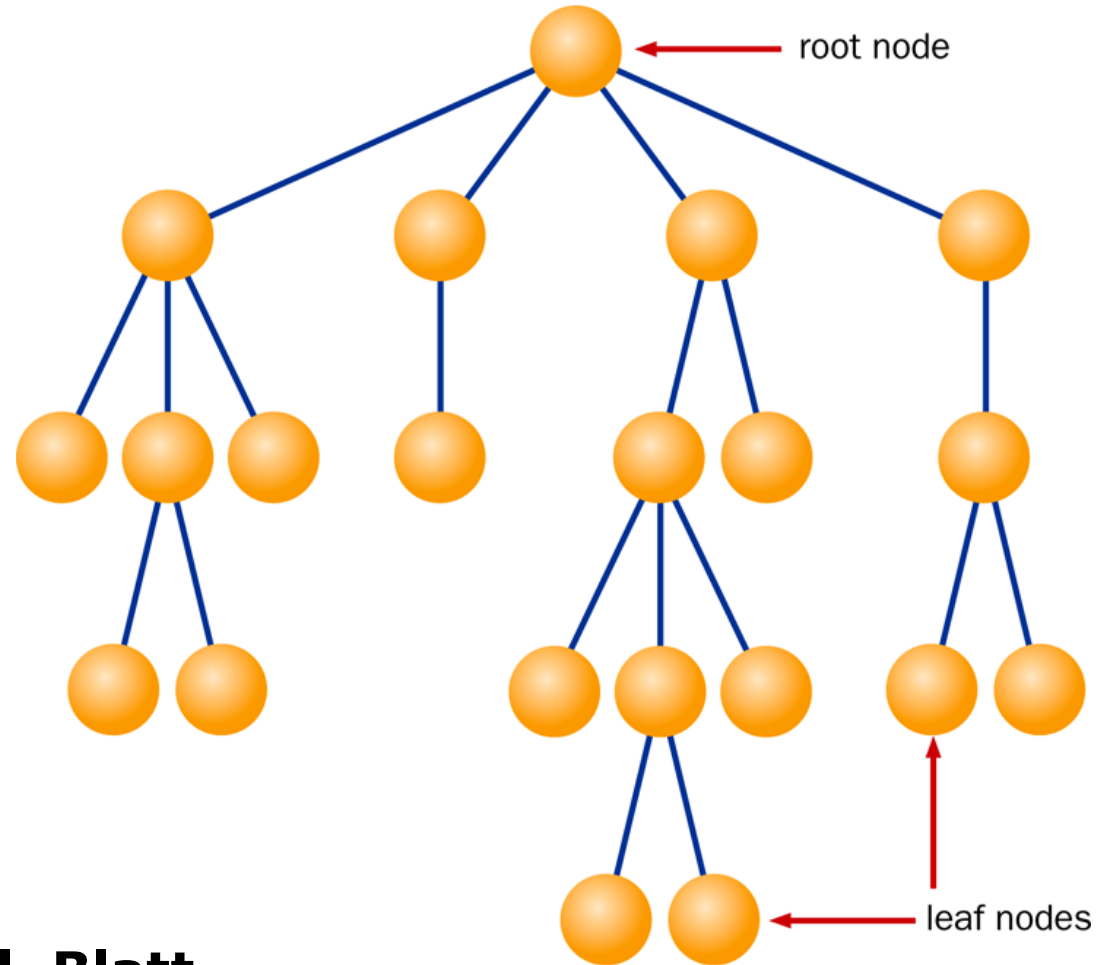
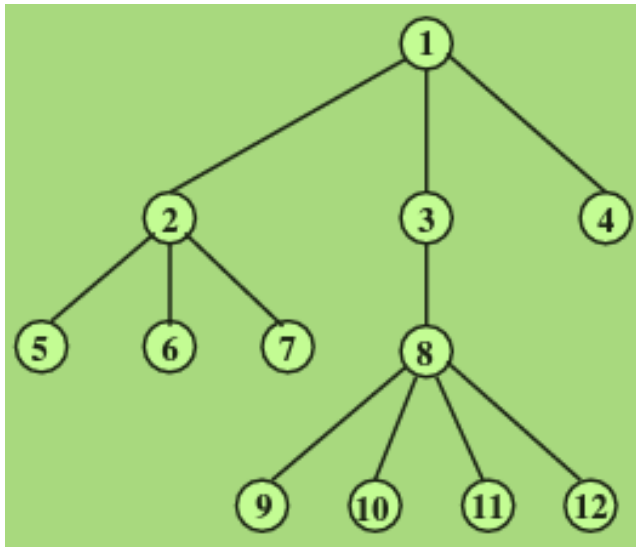
top() :liefert oberstes Element des Stacks, entfernt dieses aber nicht dort



Stacks verwenden das LIFO-Prinzip (last-in-first-out)

<https://de.wikipedia.org/wiki/Stapelspeicher>

Trees (Bäume)

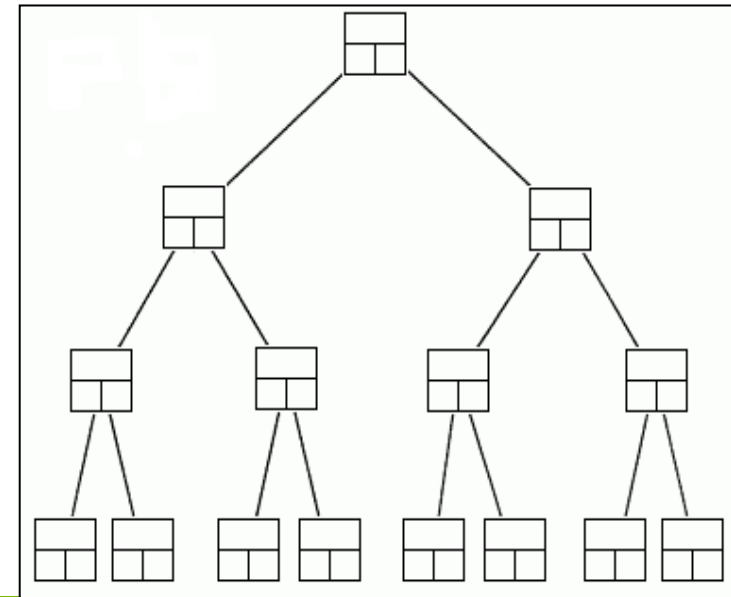


Knoten, Kante, Wurzel, Blatt

Trees (Bäume)

Binärer Baum:

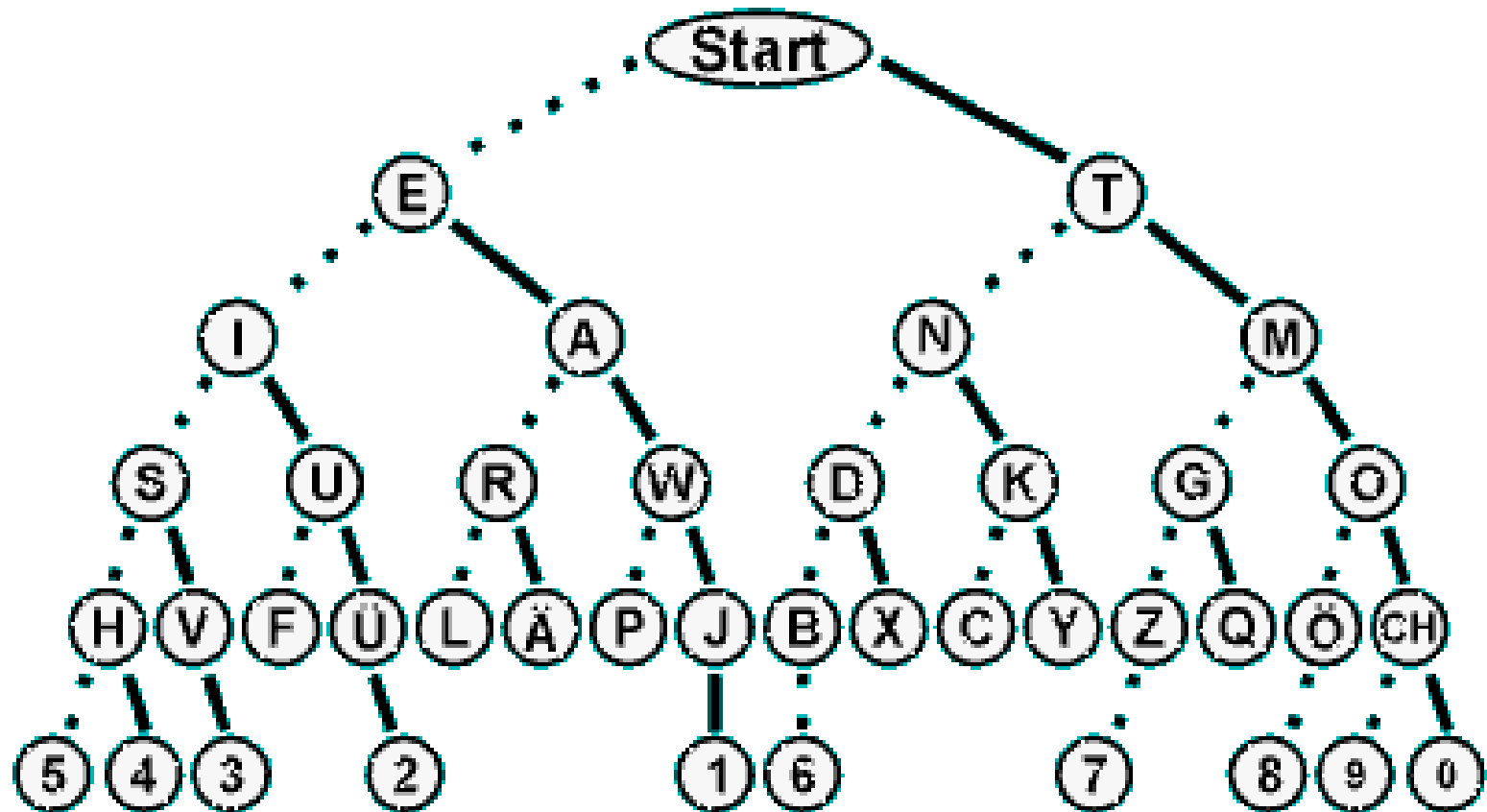
- geordneter Baum mit zwei Typen von Knoten: inneren und äußeren Knoten.
- Innere Knoten haben immer maximal zwei direkte geordnete Nachfolger, die man auch als linker und rechter Nachfolger bezeichnet.
- Äußere Knoten sind Knoten sind ohne Nachfolger.



Suchen

- **Lineare Suche:** von links oder rechts durch die Liste gehen, bis das gesuchte Element gefunden ist
- **Binäre Suche:** Voraussetzung: Elemente sind sortiert. Man schaut auf das mittlere Element, dann links oder rechts wieder das mittlere Element usw.

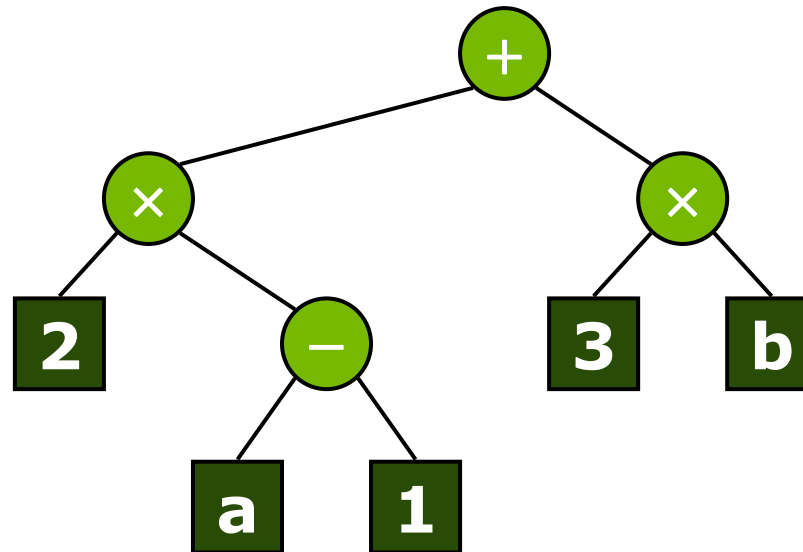
Binary Tree



Arithmetic Expression Tree

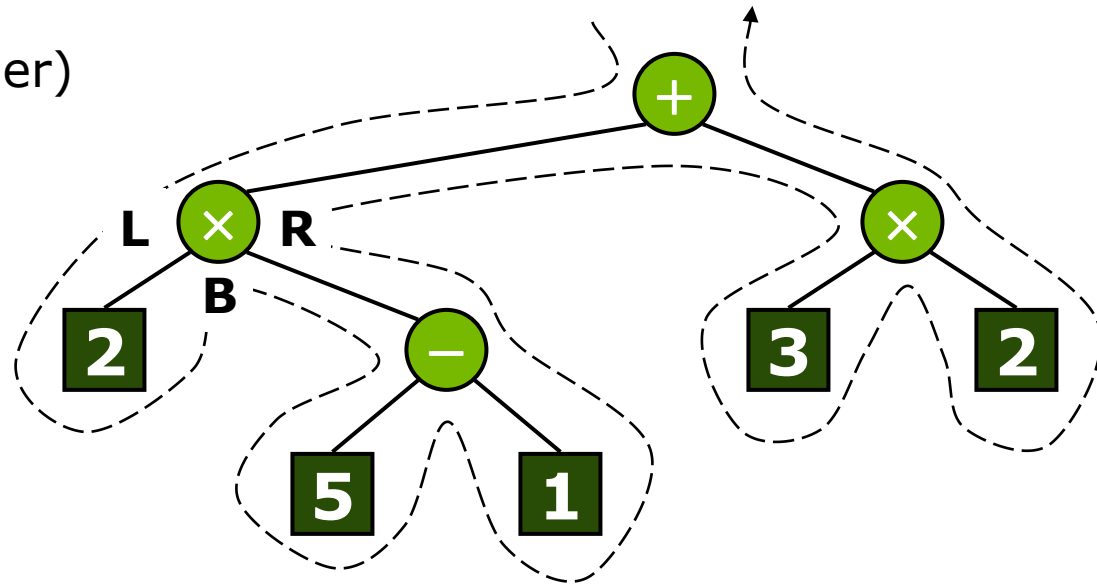
- Binary tree associated with an arithmetic expression
 - internal nodes: operators
 - external nodes: operands
- Example: arithmetic expression tree for the expression

$$(2 \times (a - 1) + (3 \times b))$$



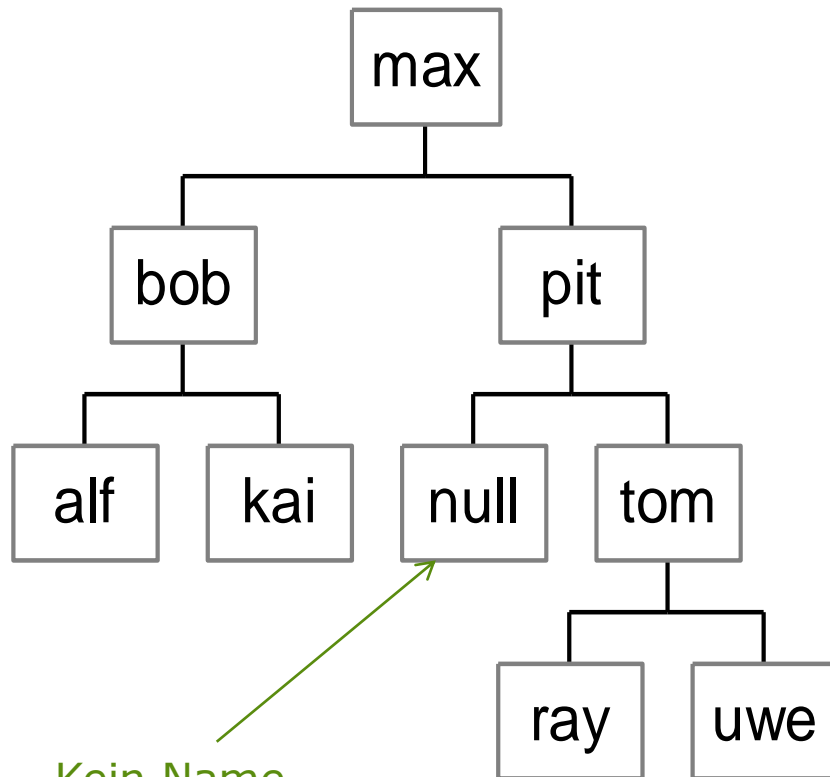
Euler Tour Traversal

- Generic traversal of a binary tree
- Includes as special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
 - on the left (preorder)
 - from below (inorder)
 - on the right (postorder)



$$(2 \times (5 - 1) + (3 \times 2))$$

Arbeiten mit Bäumen



Kein Name
sondern „Leerstelle“

Zum Bearbeiten:

- einen Teilbaum,
- dann Wurzel/Knoten,
- dann anderen Teilbaum

Aufgabe:

Einfügen von olaf

alf, bob, kai, max, pit, ray, tom, uwe

„If“ und „Else“

```
word = raw_input("Please enter a four-letter word: ")
if len(word) == 4:
    print (word + " is a four-letter word. Careful now!")
else:
    print ("That's not a four-letter word. Try harder.")
```

Statement Condition

↓ ↓

if today is sunny:

 Call friends

 Walk to park

 Buy ice cream

Indentation →

↑ ↑ ↑ ↑

Four spaces
(not tabs)

```
if today is sunny:
    Call friends
    Walk to park
    Buy ice cream
else:
    Play video games
```

If- und else-if

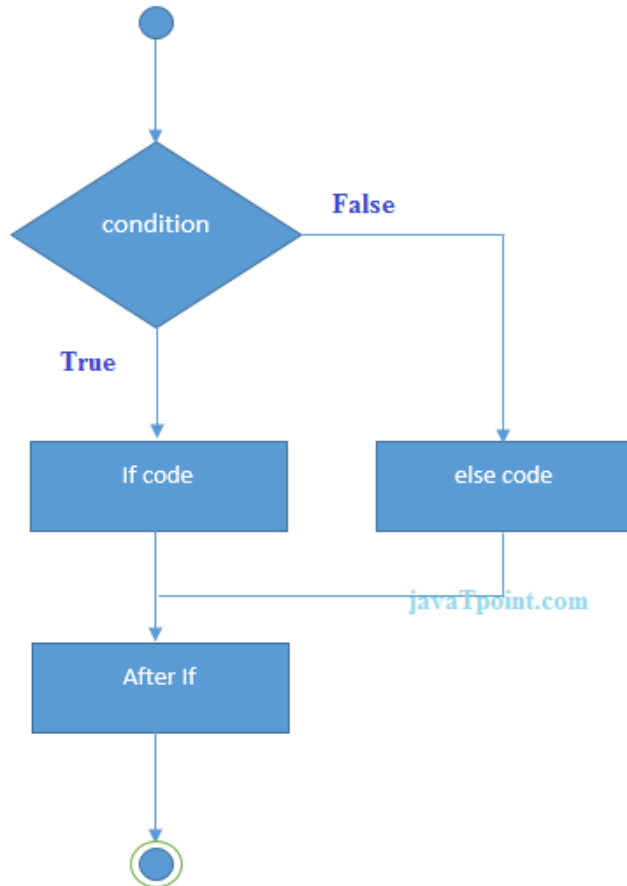
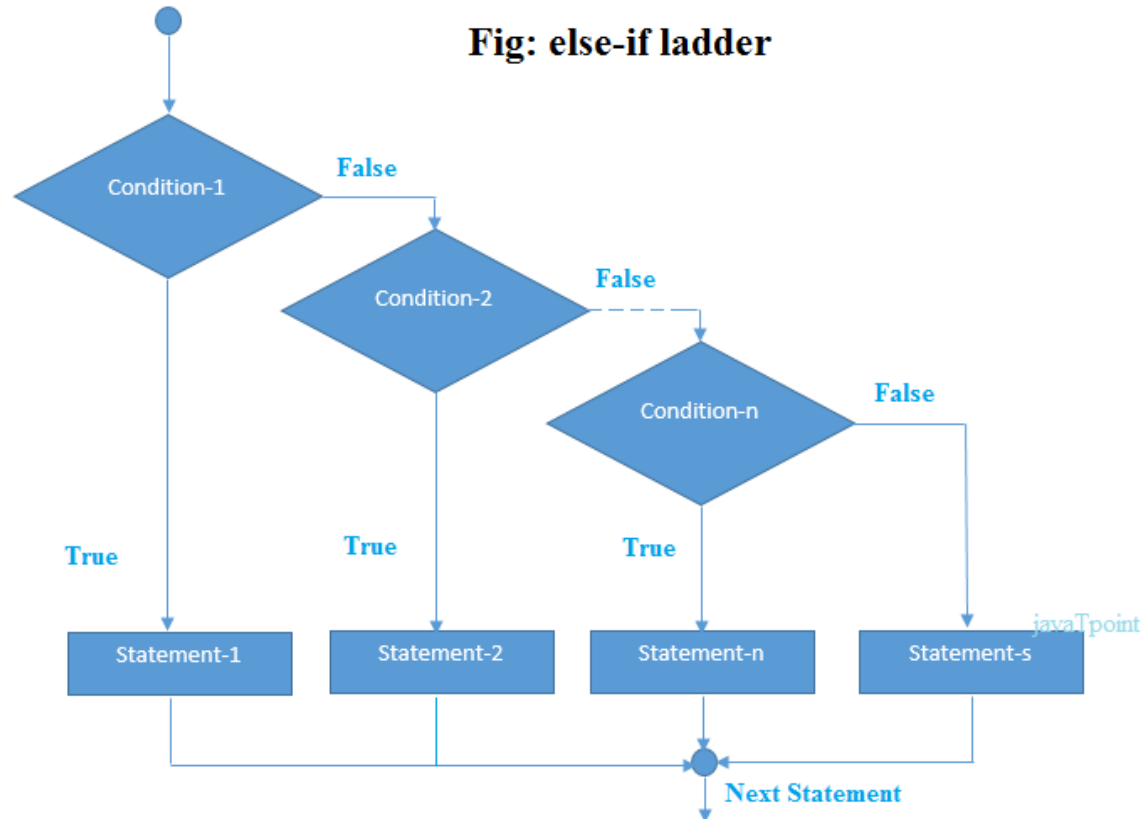


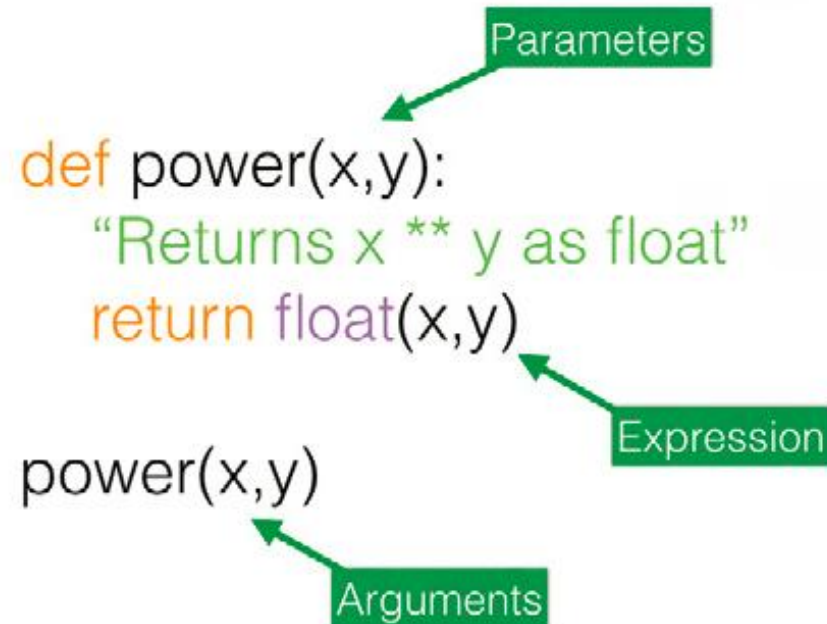
Fig: else-if ladder



Eigene Funktionen

```
def function_name(parameters):  
    function code  
    return expression
```

```
def square(number):  
    return number * number  
  
print (square(4))
```



Funktionen und Prozeduren

- **Definierte Aufgaben** gesondert programmieren
 - stepwise refinement
 - Problem zerlegen
 - häufig gebrauchte Komponenten
 - nur aktuelle Werte unterscheiden
 - Abstraktion
- **Definition einer Funktion**
 - Anzahl und Typ der Argumente
 - Statements im Rumpf der Prozedur
- **Rückgabewert**
 - mit Funktionsresultat
 - return-Statement:
 - Funktion wird sofort verlassen

```
int max (int a, int b)
{
    if (a>b) return a;
    else return b;
}
```

```
int summe (int zahl)
{
    ...
    return -z;
    return -1;
    return z;
}
```

Rekursion

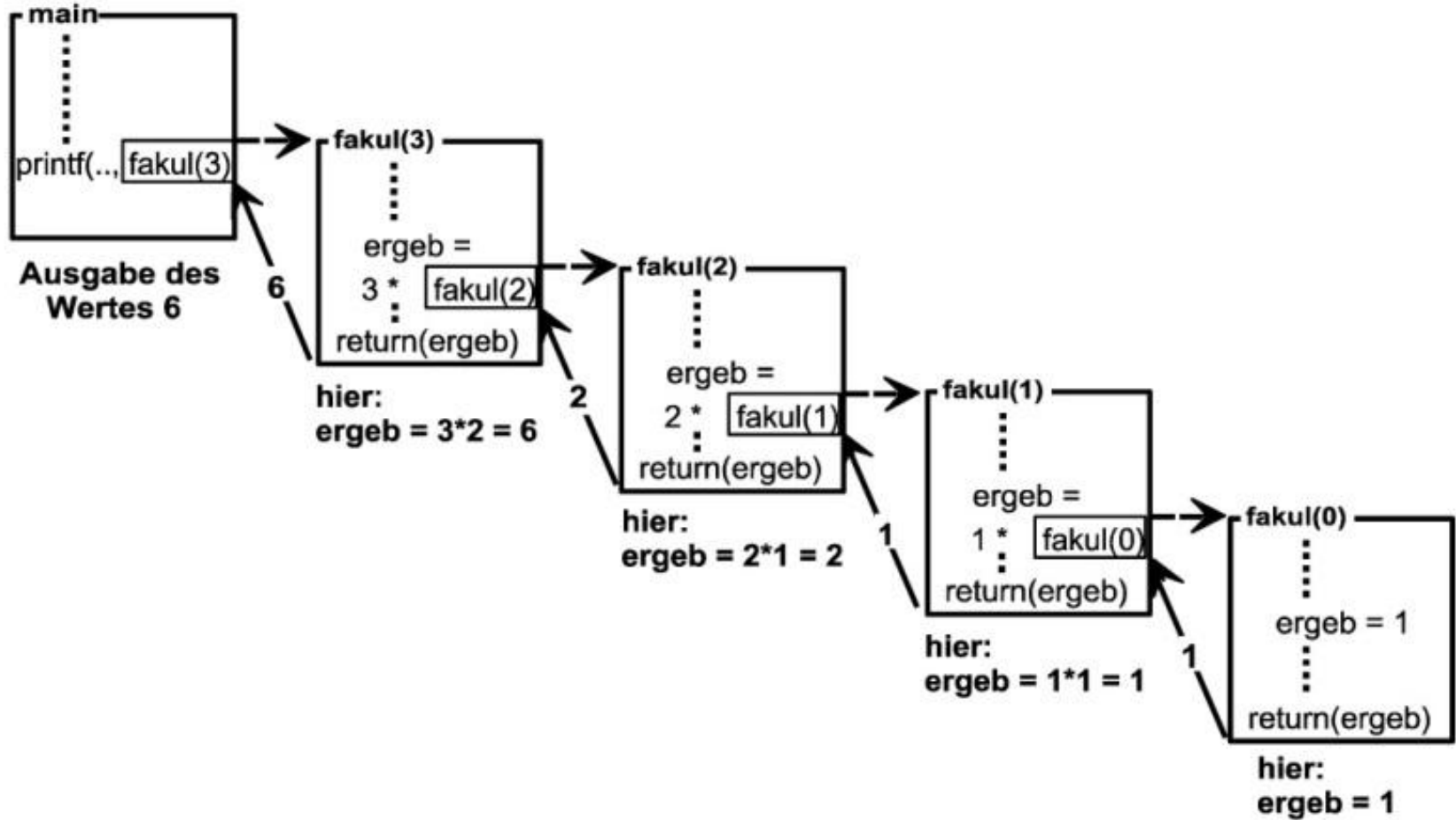


Abbildung 7.31: Rekursiver Aufruf der Funktion fakul()

Endlichkeitsbedingung (1/2)

Endlichkeitsbedingung

Ein Algorithmus muss endlich beschreibbar sein, das heißt er muss durch einen endlichen Text formulierbar sein.

- Die Bedingung der Endlichkeit bezieht sich auf die **Beschreibung** eines Algorithmus, **nicht** auf seine **Ausführung**.
- Die Beschreibung besteht aus **endlich vielen elementaren Operationen**. Zur Ausführungszeit des Algorithmus können jedoch zum Beispiel durch Schleifen beliebig viele Operationen durchlaufen werden. Dies kann allerdings dazu führen, dass der Algorithmus **nicht endet** (nicht terminiert).

Endlichkeitsbedingung (2/2)

Gegenbeispiel:

Die Kreiszahl π soll ausgegeben werden.

- `std::cout << "3";`
- `std::cout << ", ";`
- `std::cout << "1";`
- `std::cout << "4";`

Weil die Zahl π unendlich viele Stellen nach dem Komma hat, muss unser Programm (wenn es in der links dargestellten Form vorliegt) aus unendlich vielen Zeilen bestehen (1 Zeile pro Nachkommastelle).

Durch die sequenzielle Ausgabe einzelner Zeichen der unendlich langen Zahl π , wird der Programmcode ebenfalls unendlich lang.

⇒ Die Bedingung der Endlichkeit ist verletzt.

Terminiertheit (1/3)

Terminiertheit

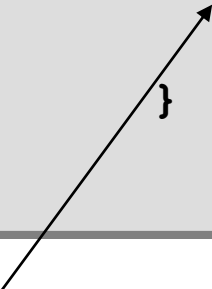
Ein Algorithmus ist terminierend, wenn er für jede erlaubte Eingabe nach einer endlichen Zahl von Schritten zu einem Ergebnis kommt.

- Ein Algorithmus muss **nach endlicher Zeit** kontrolliert **enden**.
- Die tatsächliche **Zahl der ausgeführten Schritte** kann dabei **beliebig groß** sein.
- Im Allgemeinen ist es für jeden beliebigen Algorithmus nicht möglich zu entscheiden, ob dieser terminiert oder nicht. **Viele Algorithmen sind zu komplex**, als dass sie sich durch ein mathematisches Regelsystem beschreiben lassen.

Terminiertheit (2/3)

Gegenbeispiel 1:

```
double multiply(double a, int n)
{
    double x = 0;
    while (n > 0)
    {
        x += a;
    }
    return x;
}
```



Achtung: Ein Algorithmus, der mit einem endlichen Quelltext (Programmtext) beschrieben ist, kann trotzdem eine unendliche Laufzeit haben.

Das ist eine Endlosschleife, weil n in der Schleife nicht dekrementiert wird: Die Abbruchbedingung in der while Anweisung wird nie erfüllt. Das ist ein typischer Anfängerfehler in der Programmierung.

Terminiertheit (3/3)

Gegenbeispiel 2:

Eingabe: n

wiederhole

$n = n + 1$

bis $n = 50$

Ausgabe: n

- Bei Eingabe $n < 50$ wird die Schleife nach Erreichen von $n = 50$ abgebrochen und der Algorithmus terminiert.
- Bei Eingabe $n \geq 50$ wird die Abbruchbedingung nie erfüllt werden.

⇒ Die Bedingung der Terminiertheit ist verletzt.

