



Willkommen zum Modul  
Einführung in die Informatik

# Wer bin ich?

- Name: Julian Bensch - 25 Jahre
- Beruf: Softwareentwickler
- Hobbys: Programmieren, Lesen, Reisen
- Jahre in der IT: 11

## Beruflicher Hintergrund

Freelance Software Entwickler seit 2018

Davor BTA (Biologisch technischer Assistent)

Vor dieser Zeit aber auch schon entwickelt und an Computern rumgeschraubt

# Inhalte des Moduls 1/5

## **Einführung Zahlensysteme**

- Dezimalsystem (Base-10 System)
  - Binäres System (Base-2 System)
  - Octales System (Base-8 System)
  - Hexadezimales System (Base-16 System)
- 

## **Rechnen mit binären-, octalen und hexadezimalen Zahlen**

- Wo werden sie genutzt und wie rechne ich um?
- 

## **Einführung in die Logik**

- Aussagenlogik
  - Boolesche Algebra
- 

## **Automaten-Theorie**

# Inhalte des Moduls 2/5

## Mengenlehre

- Grundlagen
  - Schnittmenge
  - Vereinigungsmenge
  - Differenzmenge
- 

## Relationen

- Äquivalenzrelationen
  - Ordnungsrelationen
  - Funktionen
- 

## Binäre Bäume

- Traversierung
- Suchbäume

# Inhalte des Moduls 3/5

## Einfache Algorithmen

- Lineare Suche
- Binäre Suche
- InsertionSort
- BubbleSort
- SelectionSort
- MergeSort
- QuickSort

---

## Rekursion

- Fakultät
- Türme von Hanoi
- Fibonacci-Zahlen

# Inhalte des Moduls 4/5

## **Laufzeitkomplexität**

- Analyse und Vergleich von Algorithmen
  - O-Notation
- 

## **Kosten von Algorithmen**

- Speicherplatz
  - Rechenzeit
  - Energieverbrauch
  - Netzwerkverkehr
- 

## **Abstrakte Datentypen**

- Was sind Abstrakte Datentypen?
  - Warum braucht man sie?
  - Klassen
-

# Inhalte des Moduls 5/5

## Datenstrukturen

- Stack
- Liste
- Heap
- Queue
- Hash-Map

## Vektor-Rechnung

- Grundlagen
- Skalarprodukt
- Kreuzprodukt

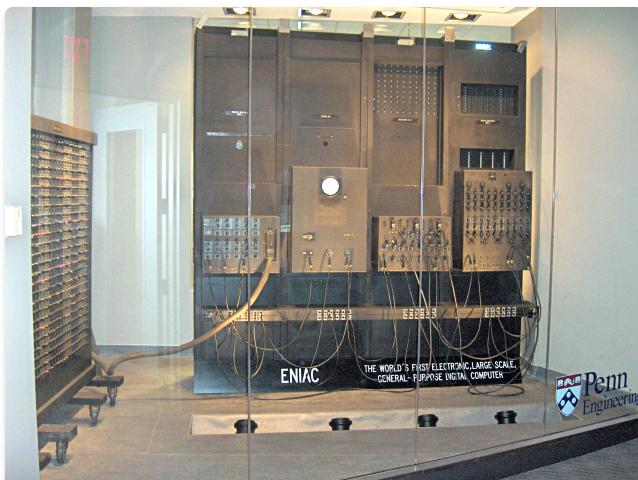
## Matrizen-Rechnung

- Addition
- Multiplikation
- Determinante

# Geschichte der Rechenmaschinen 1/5

## Erste Generation (Röhrenrechner - ab 1940)

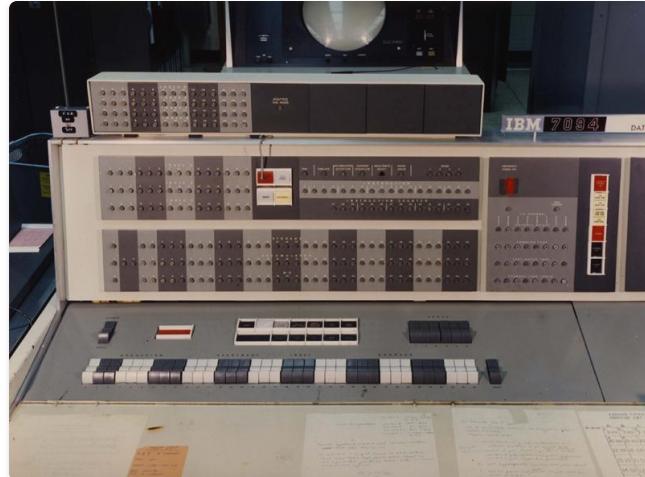
- Beispiel: ENIAC (Electronic Numerical Integrator and Computer) oder IBM 704/709
- Komponenten: Röhren
- Leistung: Wenige hunderte Rechenoperationen pro Sekunde
- Speicher: Elektronische Akkumulatoren & Externe Lochkarten
- Programmierung: Manuell durch Umstecken von Kabeln und Steckern



# Geschichte der Rechenmaschinen 2/5

## Zweite Generation (Transistorrechner - ab 1950)

- Beispiel: IBM 7090
- Komponenten: Transistoren
- Leistung: Wenige tausend Rechenoperationen pro Sekunde
- Speicher: Magnetkernspeicher
- Programmierung: Assembly



# Geschichte der Rechenmaschinen 3/5

## Dritte Generation (Integrierte Schaltkreise - ab 1960)

- Beispiel: IBM 360
- Komponenten: Integrierte Schaltkreise
- Leistung: Millionen Rechenoperationen pro Sekunde
- Speicher: Halbleiterspeicher (RAM und ROM)
- Programmierung: Assemblersprachen wie z.B. Fortran oder Cobol



# Geschichte der Rechenmaschinen 4/5

## Vierte Generation (Mikroprozessoren - ab 1970)

- Beispiel: Intel 8080 oder auch Apple II
- Komponenten: Mikroprozessoren (Durch die Miniaturisierung von Transistoren mittels Photolithographie)
- Leistung: Milliarden Rechenoperationen pro Sekunde
- Speicher: Halbleiterspeicher (RAM und ROM)
- Programmierung: Hochsprachen wie z.B. C oder Pascal



# Geschichte der Rechenmaschinen 5/5

## Fünfte Generation (Moderner Rechner - 2000er)

Heutzutage sind Computer modular und kommen selten von einem Hersteller.

Die Komponenten sind austauschbar und werden von verschiedenen Herstellern produziert.

Die Leistung ist inzwischen so hoch, dass sie nicht mehr in Rechenoperationen pro Sekunde gemessen wird, sondern in FLOPS (Floating Point Operations per Second).

- Beispiel: Multicore-Prozessoren, Quantencomputer, ...
- Komponenten: Modular, Mikroprozessoren, ASICs (Application Specific Integrated Circuit), ...
- Leistung: TeraFLOPS bis PetaFLOPS
- Speicher: Halbleiterspeicher (RAM und ROM)
- Programmierung: Hochsprachen wie z.B. C, Java oder Go

*PS: Bild braucht ihr nicht, schaut einfach auf euren Tisch.*

# Geschichte der Programmiersprachen 1/4

## Maschinensprache

Maschinensprache ist die Sprache, die ein Computer versteht.

Sie besteht aus einer Folge von 0 und 1.

**Beispiel:** 01001000 01100101 01101100 01101100 01101111 00100001

- Sie wird direkt vom Prozessor ausgeführt.
- Zur Entwicklung von Programmen ist sie nicht geeignet.
- Sie ist plattformabhängig.
- Sie ist schwer zu lesen und zu schreiben.

Erste Programme wurden in Maschinensprache geschrieben,  
hierbei wurden die 0 und 1 direkt in den Speicher geschrieben.

**(!) Die Magnete wurden händisch umgestellt.**

Wichtig:

Jeder Mikroprozessor hat einen eigenen Befehlssatz.

Hierbei unterscheidet man zwischen CISC und RISC:

RISC = Reduced Instruction Set Computing

# Geschichte der Programmiersprachen 2/4

## Höhere Programmiersprachen

Höhere Programmiersprachen sind eine Abstraktion von Assembler.  
Sie sind für Menschen leichter zu lesen und zu schreiben.  
Sie werden von einem Compiler in Maschinensprache übersetzt.

- Beispiel: C, C++, Java, Go, Python, ...
- Benötigte Software: Compiler
- Leistung: Abhängig von der Programmiersprache

Erster Vertreter der höheren Programmiersprachen: FORTRAN (1957)

(Erste Programmiersprache, die von einem Compiler übersetzt wurde und kommerziell vertrieben wurde)

# Geschichte der Programmiersprachen 3/4

## Interpretierte Sprachen (Skriptsprachen)

Skriptsprachen sind eine Abstraktion von höheren Programmiersprachen.

Sie sind für Menschen noch leichter zu lesen und zu schreiben.

Sie werden von einem Interpreter in Maschinensprache übersetzt.

- Beispiel: Python, Perl, PHP, JavaScript, ...
- Benötigte Software: Interpreter
- Leistung: Abhängig von der Programmiersprache

Erster Vertreter der Skriptsprachen: Perl (1987)

Differenz zwischen höheren Programmiersprachen und Skriptsprachen:

- Höhere Programmiersprachen werden in Maschinensprache übersetzt und ausgeführt
- Skriptsprachen werden in Maschinensprache übersetzt und interpretiert (Zeile für Zeile ausgeführt)

# Geschichte der Programmiersprachen 4/4

## Compiler

Ein Compiler ist ein Programm, das Quellcode in Maschinensprache übersetzt.

- Beispiel: GCC, Clang, ... In der Regel wird der Quellcode in einem Schritt übersetzt.  
Dadurch ist die Ausführungsgeschwindigkeit höher als bei einem Interpreter.  
Der compilerte Code ist aber in der Regel plattformabhängig.

## Interpreter

Ein Interpreter ist ein Programm, das Quellcode in Maschinensprache übersetzt und ausführt.

- Beispiel: Python, Perl, PHP, JavaScript, ... In der Regel wird der Quellcode Zeile für Zeile ausgeführt.  
Dadurch ist die Ausführungsgeschwindigkeit geringer als bei einem Compiler.  
Dafür ist die Entwicklungsgeschwindigkeit höher, da der Quellcode nicht erst übersetzt werden muss.

# Geschichte der Schalter

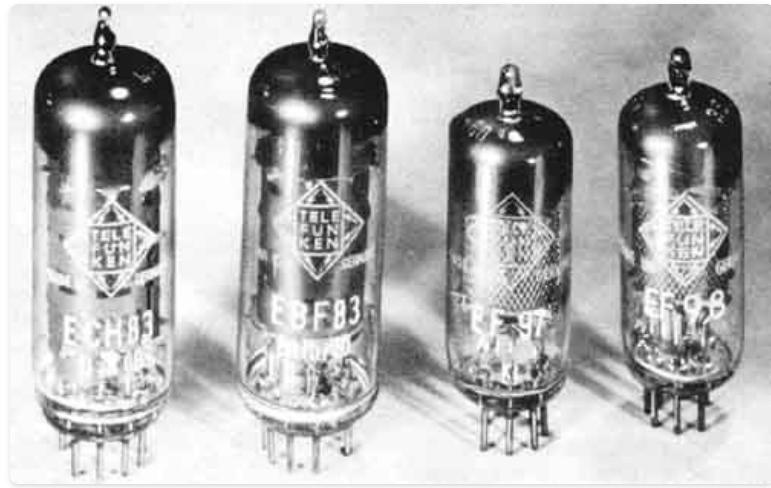
## Vakuumröhren

### Funktionsweise:

- Die Kathode wird erhitzt, dadurch lösen sich Elektronen von der Kathode
- Die Elektronen werden zur Anode gezogen und aufgenommen
- Der Stromkreis ist geschlossen

### Wichtig:

Wird die Kathode nicht erhitzt, fließt kein Strom.



### Bestandteile:

- Glaskolben
- Kathode
- Anode
- Heizwendel

# Geschichte der Schalter

## Transistoren

Bestandteile:

- Emitter (E)
- Basis (B)
- Kollektor (C)

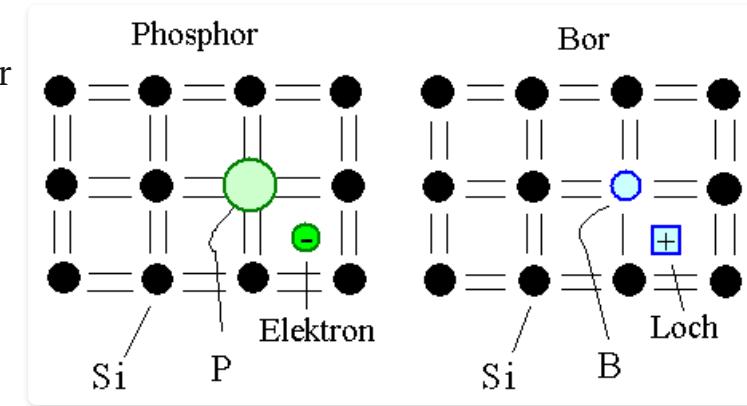
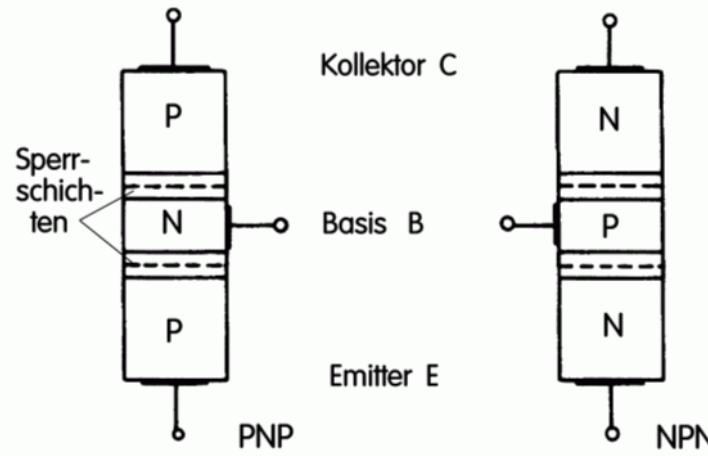
Besonderheit von Silizium:

Silizium hat 4 Elektronen in der äußersten Schale

Benachbartes Silizium hat ebenfalls 4 Elektronen in der äußersten Schale Dadurch können sich Siliziumatome zu einem Kristallgitter verbinden welches elektrisch neutral ist

Dotierung von Silizium:

Silizium kann mit Phosphor oder Bor dotiert werden was zu einer Veränderung der elektrischen Leitfähigkeit führt



# Übersicht der relevanten Zahlensysteme 1/3

Tabellarische Übersicht der Zahlensysteme

System	Base	Zahlen	Anwendungsbereiche
Dezimal	10	0-9	Alltag, Finanzen
Binär	2	0,1	Informatik
Octal	8	0-7	Ältere Systeme
Hex	16	0-9,A-F	Programmierung

# Übersicht der relevanten Zahlensysteme 2/3

## Dezimalsystem (base 10 system):

- Anzahl der Ziffern: 10 (0-9)
- Grund: 10 Finger, historisch/biologisch
- Anwendungen: Alltag, Finanzen, Wissenschaftliche Notation
- Beispiel:  $123 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$

## Binäres System (base 2 system):

- Anzahl der Ziffern: 2 (0,1)
- Grund: Elektronik, Schalter (an/aus)
- Anwendungen: Informatik
- Beispiel:  $101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$  (Dezimal: 5)

# Übersicht der relevanten Zahlensysteme 3/3

## Octales System (base 8 system):

- Anzahl der Ziffern: 8 (0-7)
- Grund: Einfachere Darstellung von Binärdaten & ältere Unix-Systeme
- Anwendungen: Frühe Computersysteme, gelegentlich in der Programmierung
- Beispiel:  $123 = 1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0$  (Dezimal: 83)

## Hexadezimales System (base 16 system):

- Anzahl der Ziffern: 16 (0-9, A-F)
- Grund: Einfachere Darstellung von Binärdaten & Speicheradressen
- Anwendungen: Programmierung, Webfarben
- Beispiel:  $7F2 = 7 \times 16^2 + 15 \times 16^1 + 2 \times 16^0$  (Dezimal: 2034)

# Umrechnen von Dezimal in andere Zahlensysteme 1/2

Dezimal zu Binär:

- Beispiel: 123
- $123 / 2 = 61$  Rest 1
- $61 / 2 = 30$  Rest 1
- $30 / 2 = 15$  Rest 0
- $15 / 2 = 7$  Rest 1
- $7 / 2 = 3$  Rest 1
- $3 / 2 = 1$  Rest 1
- $1 / 2 = 0$  Rest 1
- Ergebnis: 1111011 (von unten nach oben lesen)
- Kontrolle:  $1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 123$

# Umrechnen von Dezimal in andere Zahlensysteme 2/2

Dezimal zu Hexadezimal:

- Beispiel: 123
  - $123 / 16 = 7$  Rest 11 (Notation durch Buchstaben B)
  - $7 / 16 = 0$  Rest 7
  - Ergebnis: 7B (von unten nach oben lesen)
  - Kontrolle:  $7 \times 16^1 + 11 \times 16^0 = 123$
- 

Dezimal zu Oktal:

- Beispiel: 123
- $123 / 8 = 15$  Rest 3
- $15 / 8 = 1$  Rest 7
- $1 / 8 = 0$  Rest 1
- Ergebnis: 173 (von unten nach oben lesen)
- Kontrolle:  $1 \times 8^2 + 7 \times 8^1 + 3 \times 8^0 = 123$

# Umrechnen von anderen Zahlensystemen zu Dezimal

Binär zu Dezimal: Beispiel 1111011

- $1111011 = 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
  - $1111011 = 64 + 32 + 16 + 8 + 0 + 2 + 1$
  - $1111011 = 123$
- 

Hexadezimal zu Dezimal: Beispiel 7B

- $7B = 7 \times 16^1 + 11 \times 16^0$
  - $7B = 112 + 11$
  - $7B = 123$
- 

Oktal zu Dezimal: Beispiel 173

- $173 = 1 \times 8^2 + 7 \times 8^1 + 3 \times 8^0$
- $173 = 64 + 56 + 3$
- $173 = 123$

# Aufgabe: Umrechnen von Zahlensystemen

Typ	Aufgabe 1	Aufgabe 2	Aufgabe 3
Binär zu Dezimal	11011011	10011100	10101010
Dezimal zu Binär	173	219	255
Oktal zu Dezimal	357	521	777
Dezimal zu Oktal	317	465	511
Hexadezimal zu Dezimal	9A	FA	B2
Dezimal zu Hexadezimal	154	250	178

# Lösung: Umrechnen von Zahlensystemen

Aufgabentyp	Aufgabe 1	Aufgabe 2	Aufgabe 3
Binär zu Dezimal	$11011011 = 219$	$10011100 = 156$	$10101010 = 170$
Dezimal zu Binär	$173 = 10101101$	$219 = 11011011$	$255 = 11111111$
Oktal zu Dezimal	$357 = 239$	$521 = 337$	$777 = 511$
Dezimal zu Oktal	$317 = 475$	$465 = 721$	$511 = 777$
Hexadezimal zu Dezimal	$9A = 154$	$FA = 250$	$B2 = 178$
Dezimal zu Hexadezimal	$154 = 9A$	$250 = FA$	$178 = B2$

# Aufgabe: Umrechnen von Zahlensystemen (Mischformen)

Aufgabentyp

Binär zu Oktal

Oktal zu Hexadezimal

Hexadezimal zu Binär

Binär zu Hexadezimal

Hexadezimal zu Oktal

Oktal zu Binär

Aufgabe

11010110

641

3FA

10101011

1F

572

# Lösung: Umrechnen von Zahlensystemen (Mischformen)

Aufgabentyp

Binär zu Oktal

Oktal zu Hexadezimal

Hexadezimal zu Binär

Binär zu Hexadezimal

Hexadezimal zu Oktal

Oktal zu Binär

Aufgabe

$11010110 = 326$

$641 = 1A1$

$3FA = 1111111010$

$10101011 = AB$

$1F = 37$

$572 = 101111010$

# Detailierte Lösung: Binär zu Oktal 11010110

- $11010110 = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
- $11010110 = 128 + 64 + 16 + 4 + 2$
- $11010110 = 214$  (Dezimal)
- Nun Dezimal zu Oktal
- $214 / 8 = 26$  Rest 6
- $26 / 8 = 3$  Rest 2
- $3 / 8 = 0$  Rest 3
- Ergebnis: 326 (von unten nach oben lesen)

# Detailierte Lösung: Oktal zu Hexadezimal 641

- $641 = 6 \times 8^2 + 4 \times 8^1 + 1 \times 8^0$
- $641 = 384 + 32 + 1$
- $641 = 417$  (Dezimal)
- Nun Dezimal zu Hexadezimal
- $417 / 16 = 26$  Rest 1
- $26 / 16 = 1$  Rest 10 (Notation durch Buchstaben A)
- $1 / 16 = 0$  Rest 1
- Ergebnis: 1A1 (von unten nach oben lesen)

# Detailierte Lösung Hexadezimal zu Binär 3FA

## Umrechnen in Dezimal

- $3FA = 3 \times 16^2 + 15 \times 16^1 + 10 \times 16^0$
- $3FA = 768 + 240 + 10$
- $3FA = 1018$  (Dezimal)

## Umrechnen in Binär

- $1018 / 2 = 509$  Rest 0
- $509 / 2 = 254$  Rest 1
- $254 / 2 = 127$  Rest 0
- $127 / 2 = 63$  Rest 1
- $63 / 2 = 31$  Rest 1
- $31 / 2 = 15$  Rest 1
- $15 / 2 = 7$  Rest 1
- $7 / 2 = 3$  Rest 1
- $3 / 2 = 1$  Rest 1
- $1 / 2 = 0$  Rest 1
- Ergebnis: 1111111010 (von unten nach oben lesen)

# Detailierte Lösung Binär zu Hexadezimal 10101011

- $10101011 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
- $10101011 = 128 + 32 + 8 + 2 + 1$
- $10101011 = 171$  (Dezimal)
- Nun Dezimal zu Hexadezimal
- $171 / 16 = 10$  Rest 11 (Notation durch Buchstaben B)
- $10 / 16 = 0$  Rest 10 (Notation durch Buchstaben A)
- Ergebnis: AB (von unten nach oben lesen)

# Detailierte Lösung Hexadezimal zu Oktal 1F

- $1F = 1 \times 16^1 + 15 \times 16^0$
- $1F = 16 + 15$
- $1F = 31$  (Dezimal)
- Nun Dezimal zu Oktal
- $31 / 8 = 3$  Rest 7
- $3 / 8 = 0$  Rest 3
- Ergebnis: 37 (von unten nach oben lesen)

# Detailierte Lösung Oktal zu Binär 572

- $572 = 5 \times 8^2 + 7 \times 8^1 + 2 \times 8^0$
- $572 = 320 + 56 + 2$
- $572 = 378$  (Dezimal)
- Nun Dezimal zu Binär
- $378 / 2 = 189$  Rest 0
- $189 / 2 = 94$  Rest 1
- $94 / 2 = 47$  Rest 0
- $47 / 2 = 23$  Rest 1
- $23 / 2 = 11$  Rest 1
- $11 / 2 = 5$  Rest 1
- $5 / 2 = 2$  Rest 1
- $2 / 2 = 1$  Rest 0
- $1 / 2 = 0$  Rest 1
- Ergebnis: 101111010 (von unten nach oben lesen)

## Zusammenfasung:

- Umrechnen von anderen Zahlensystemen zu Dezimal:
  - Binär zu Dezimal: Addition der Potenzen
  - Hexadezimal zu Dezimal: Addition der Potenzen
  - Oktal zu Dezimal: Addition der Potenzen
- Umrechnen von Dezimal in andere Zahlensysteme:
  - Dezimal zu Binär: Division durch 2
  - Dezimal zu Hexadezimal: Division durch 16
  - Dezimal zu Oktal: Division durch 8

Zum umrechnen wird erst in Dezimal umgerechnet, dann in das gewünschte Zahlensystem.

# Aussagenlogik 1/5

- Atomare Aussagen:

Grundlegende, nicht weiter zerlegbare Aussagen.

Z.B.,

- p: Es regnet.
- q: Es ist kalt.

- Operatoren:

$\wedge$ : UND (Verbindet zwei Aussagen, die beide wahr sein müssen.)

$\vee$ : ODER (Verbindet zwei Aussagen; mindestens eine muss wahr sein.)

$\neg$ : NICHT (Kehrt den Wahrheitswert einer Aussage um.)

- Merksätze:

Atomare Aussagen sind die Bausteine der Aussagenlogik.

Operatoren: Symbole, die Beziehungen zwischen atomaren Aussagen herstellen.

Außerdem: Operatoren verknüpfen atomare Aussagen zu komplexeren Ausdrücken.

# Aussagenlogik 2/5

## De Morgan'sche Regel

- Grundsätzlich gilt:  
nicht (a und b) ist äquivalent zu ((nicht a) oder (nicht b)), sowie  
nicht (a oder b) ist äquivalent zu ((nicht a) und (nicht b)).

- Beispiel:  
Es regnet nicht und es ist nicht kalt.  
 $\neg(p \wedge q) = \neg p \vee \neg q$   
Es regnet nicht oder es ist nicht kalt.

# Aussagenlogik 3/5

## Beispiele

- Atomare Aussagen:

$p$ : Es regnet.

$q$ : Es ist kalt.

- komplexe Aussagen:

$p \wedge q$ : Es regnet und es ist kalt.

$\neg q$ : Es ist nicht kalt.

$\neg(p \wedge q)$ : Es regnet nicht und es ist nicht kalt.

- Sehr komplexe Aussage:

$(p \wedge q) \vee (\neg p \wedge \neg q)$ : Es regnet und es ist kalt oder es regnet nicht und es ist nicht kalt.

---

## Aufgaben:

Bildet 5 eigene atomare Aussagen und stellt sie vor.

Bildet 5 eigene komplexe Aussagen und stellt sie vor.

# Aussagenlogik 4/5

## Gesetze der Aussagenlogik

- Kommutativgesetz:  $p \wedge q = q \wedge p, p \vee q = q \vee p$
- Assoziativgesetz:  $p \wedge (q \wedge r) = (p \wedge q) \wedge r$
- Distributivgesetz:  $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$

---

Aufgabe: Vereinfacht folgende Aussagen

- $(p \wedge q) \vee (\neg p \wedge q)$
- $(p \vee q) \wedge (p \vee \neg q)$
- $(p \wedge q) \vee (p \wedge \neg q)$
- $\neg(p \vee q) \vee (p \wedge q)$
- $(p \vee q \vee r) \wedge (\neg p \vee q \vee r)$

# Aussagenlogik 5/5

Lösung: Gesetze der Aussagenlogik

Beispiel 1:

$$(p \wedge q) \vee (\neg p \wedge q)$$

Schritt 1: Distributivgesetz anwenden

Wir können das Distributivgesetz anwenden, um den Ausdruck zu vereinfachen. Dabei nehmen wir  $q$  als gemeinsamen Faktor und erhalten:

$$(p \vee \neg p) \wedge q$$

Schritt 3: Tautologie erkennen

Der Ausdruck  $(p \vee \neg p)$  ist eine Tautologie, da er immer wahr ist. Wir können ihn also durch 1 (wahr) ersetzen und erhalten:

$$1 \wedge q$$

Schritt 4: Vereinfachen

Da  $1 \wedge q$  immer  $q$  ist, können wir den Ausdruck vereinfachen zu:  $q$

Lösungen:

- $(p \wedge q) \vee (\neg p \wedge q) = q$
- $(p \vee q) \wedge (p \vee \neg q) = p$
- $(p \wedge q) \vee (p \wedge \neg q) = p$
- $\neg(p \vee q) \vee (p \wedge q) = \neg q$
- $(p \vee q \vee r) \wedge (\neg p \vee q \vee r) = q \vee r$

# Boolesche Algebra 1/2

- Bool'sche Variablen:

Grundlegende Variablen, die nur 1 (wahr) oder 0 (falsch) sein können.

Z.B., A: Es ist kalt. oder B: Es regnet.

- Grundoperationen:

AND: Beide Variablen müssen 1 sein.

OR: Mindestens eine Variable muss 1 sein.

NOT: Kehrt den Wert um.

- Merksätze:

Bool'sche Variablen sind die Grundlagen der Booleschen Algebra.

Grundoperationen: Elementare Operationen, die auf Bool'schen Variablen ausgeführt werden können.

Außerdem: Mit Grundoperationen bilden wir logische Ausdrücke und Funktionen.

Tipp: Auch hier gibt es eine De Morgan'sche Regel.

# Boolesche Algebra 2/2

## Beispiele

- Bool'sche Variablen:

A: Es ist kalt.

B: Es regnet.

- komplexe Ausdrücke:

A AND B: Es ist kalt und es regnet.

NOT A: Es ist nicht kalt.

NOT (A AND B): Es ist nicht kalt oder es regnet nicht. (oder beides nicht - De Morgan'sche Regel)

- Sehr komplexer Ausdruck:

(A AND B) OR (NOT A AND NOT B): Es ist kalt und es regnet oder es ist nicht kalt und es regnet nicht.

## Aufgaben:

Bildet 5 eigene Bool'sche Variablen und stellt sie vor.

Bildet 5 eigene komplexe Ausdrücke und stellt sie vor.

# Einführung in die Automatentheorie 1/8

Automaten sind Modelle für rechnende Maschinen welche einen Zustand (State) haben können.

## Erklärung

- Zustand: Wo der Automat gerade ist, z.B. "an" oder "aus".
- 

## Themen:

- Endliche Automaten
- Nichtdeterministische Automaten
- Übergangsfunktionen

# Einführung in die Automatentheorie 2/8

## Endliche Automaten (DFA)

Deterministic Finite Automaton (Deterministischer Endlicher Automat)

Eindeutiger nächster Zustand für jede Eingabe und jeden aktuellen Zustand.

Definition:

- Deterministisch: Eindeutiger nächster Zustand
  - Formale Definition:  $(Q, \Sigma, \delta, q_0, F)$
- 

Erklärung

- $Q$ : Menge aller Zustände
  - $\Sigma$ : Menge der Symbole (Eingabe)
  - $\delta$ : Übergangsfunktion
  - $q_0$ : Startzustand
  - $F$ : Menge der Endzustände
- 

Merksatz

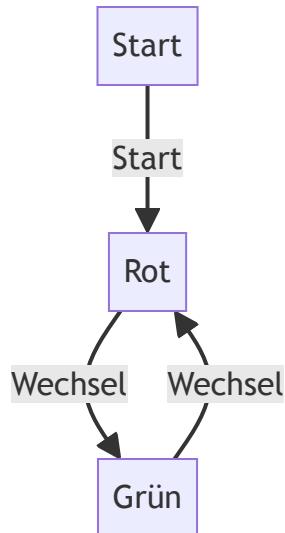
- DFA ist der einfachste Automatentyp.

# Einführung in die Automatentheorie 3/8

## Beispiel Aufgabe

- Zeichnet einen DFA für eine Ampel, die nur "rot" und "grün" kennt.

Beispiel:



# Einführung in die Automatentheorie 4/8

## Nichtdeterministische Automaten (NFA)

- Mehrere mögliche nächste Zustände
- Formale Definition:  $(Q, \Sigma, \Delta, q_0, F)$

## Merksatz

- NFA erlauben mehr Freiheit, aber gleiche Ausdrucksstärke wie DFA.

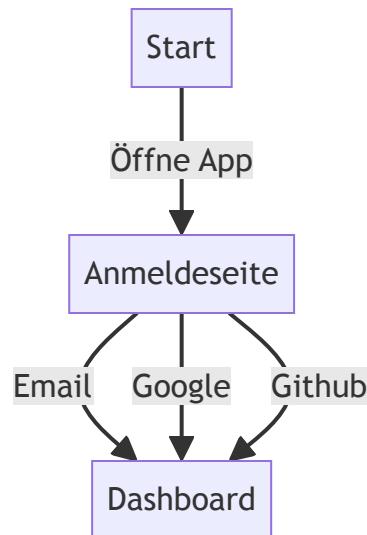
## Erklärung

- $2^Q$ : Menge der möglichen Zustandskombinationen

# Einführung in die Automatentheorie 5/8

## Beispiel Aufgabe

- Zeichnet einen NFA für eine App in der man sich nach dem Start mit Email, Google oder Github anmelden kann und danach zu einem Dashboard kommt.



# Einführung in die Automatentheorie 6/8

## Übergangsfunktionen

- DFA:  $\delta : Q \times \Sigma \rightarrow Q$
- NFA:  $\Delta : Q \times \Sigma \rightarrow 2^Q$

## Merksatz

- Übergangsfunktionen bestimmen die Dynamik des Automaten.

## Erklärung

- $\rightarrow$ : "führt zu" oder "wird zu"

# Einführung in die Automatentheorie 7/8

## Anwendungen der Automatentheorie

- Textverarbeitung: z.B. Rechtschreibprüfung
- Netzwerkprotokolle: Regeln für Datenübertragung
- Compilerbau: Umwandlung von Quellcode in Maschinencode

## Aufgabe

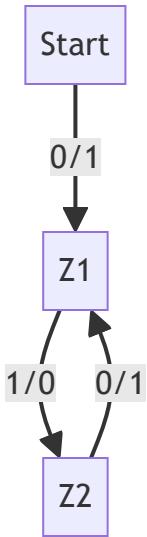
- Nennt 5 Anwendungsbeispiele für Automaten in der Praxis und zeichnet einen DFA oder NFA für jedes Beispiel.

Nutzt gerne: <https://mermaid.live/>

# Einführung in die Automatentheorie 8/8

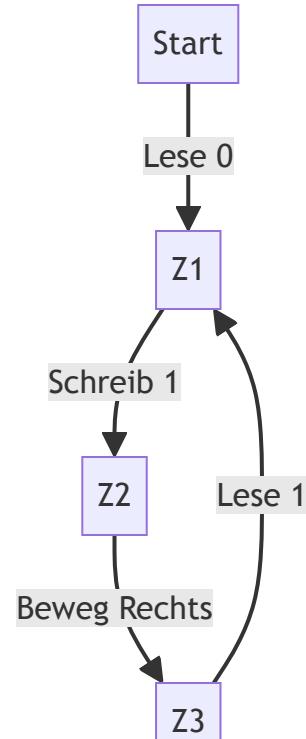
## Mealy- und Moore-Automat

Beide Automaten sind deterministisch und sind in der Lage Ausgaben zu erzeugen.



## Turingmaschine

Turingmaschinen sind nicht deterministisch und lassen nur mit nichtdeterministischen Automaten simulieren.



# Schaltungen 1/5

- Eine Schaltung ist eine Verknüpfung von logischen Gattern.
- Schaltalgebra ist die mathematische Beschreibung von Schaltungen.
- Aussagenlogik + Schaltalgebra ist die mathematische Beschreibung von Schaltungen.

Die Begriffe werden oft synonym verwendet.

Am Ende des Tages geht es darum, dass wir eine Schaltung mit einer mathematischen Formel beschreiben können.  
Dabei verwenden wir das binäre Zahlensystem.

**Konjunktion**

**Disjunktion**

**Negation**

**(UND)**

**(ODER)**

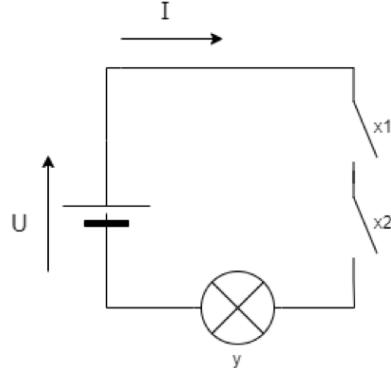
**(NICHT)**

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

	$\neg$
0	1
1	0

# Schaltungen 2/5



Logisches UND (AND / Konjunktion)

Die logische Verknüpfung von zwei Aussagen ist wahr, wenn beide Aussagen wahr sind.

Die logische Verknüpfung von zwei Aussagen ist falsch, wenn mindestens eine Aussage falsch ist.

Wahrheitstabelle

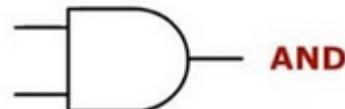
X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Logisches UND (AND) entspricht einer Reihenschaltung von Schaltern.

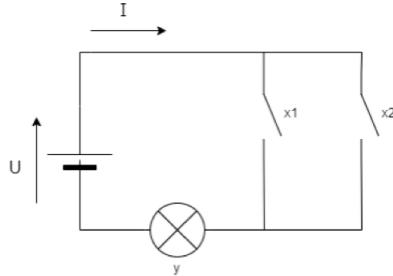
Die Schaltung ist nur dann geschlossen, wenn alle Schalter geschlossen sind.

Geschlossener Schalter = 1  
Offener Schalter = 0

Logischer Schalter mit AND:



# Schaltungen 3/5



Logisches ODER (OR / Disjunktion)

Die logische Verknüpfung von zwei Aussagen ist wahr, wenn mindestens eine Aussage wahr ist.

Die logische Verknüpfung von zwei Aussagen ist falsch, wenn beide Aussagen falsch sind.

Wahrheitstabelle

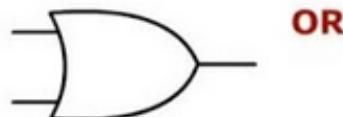
X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

Logisches ODER (OR) entspricht einer Parallelschaltung von Schaltern.

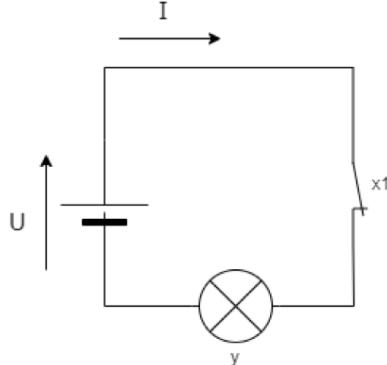
Die Schaltung ist geschlossen, wenn mindestens ein Schalter geschlossen ist.

Geschlossener Schalter = 1  
Offener Schalter = 0

Logischer Schalter mit OR:



# Schaltungen 4/5



## Logisches NICHT (NOT / Negation)

Die logische negation einer Aussage ist wahr, wenn die Aussage falsch ist.

Die logische negation einer Aussage ist falsch, wenn die Aussage wahr ist.

Es wird also der Wahrheitswert umgekehrt.

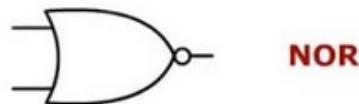
## Wahrheitstabelle

X	Y
0	1
1	0

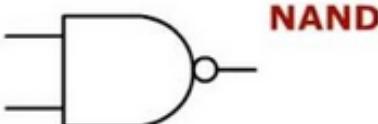
Logisches NICHT (NOT) entspricht einem Schalter, der umgekehrt schaltet.

Logische Schalter mit NOT:

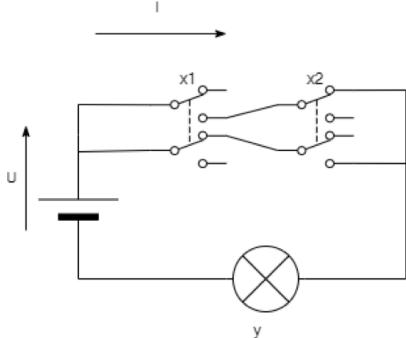
NOR Schaltung = NOT(OR)



NAND Schaltung = NOT(AND)



# Schaltungen 5/5



## Logisches XOR (XOR / Exklusives ODER)

Die logische Verknüpfung von zwei Aussagen ist wahr, wenn genau eine Aussage wahr ist.

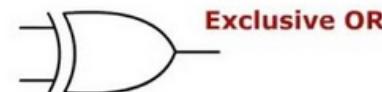
Die logische Verknüpfung von zwei Aussagen ist falsch, Logische Schalter mit XOR:  
wenn beide Aussagen falsch oder beide Aussagen wahr sind.

Wahrheitstabelle

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Logisches XOR (XOR) entspricht einer Parallelschaltung von Schaltern, bei der nur ein Schalter geschlossen sein darf.

Geschlossener Schalter = 1  
Offener Schalter = 0



# Mengenlehre Einführung

## Agenda

- Grundlagen
- Schnittmenge
- Vereinigungsmenge
- Differenzmenge
- Spezielle Mengen
- Anwendungen in der Informatik
- Aufgaben

# Mengenlehre 1/6

## Grundlagen

- Eine Menge ist eine Zusammenfassung von unterscheidbaren Elementen.
- Eine Menge kann endlich oder unendlich sein.
- Eine Menge kann leer sein.
- Die Anzahl aller Elemente einer Menge wird als Mächtigkeit oder Kardinalität bezeichnet.
- Die Reihenfolge der Elemente ist irrelevant.
- Elemente die doppelt vorkommen werden nur einmal gezählt und sind in der Menge nur einmal enthalten.
- Mengen werden mit Großbuchstaben bezeichnet und können entweder mit konkreten Elementen oder mit einer Beschreibung definiert werden.

Beispiel konkret:

$$M = \{1, 2, 3\}$$

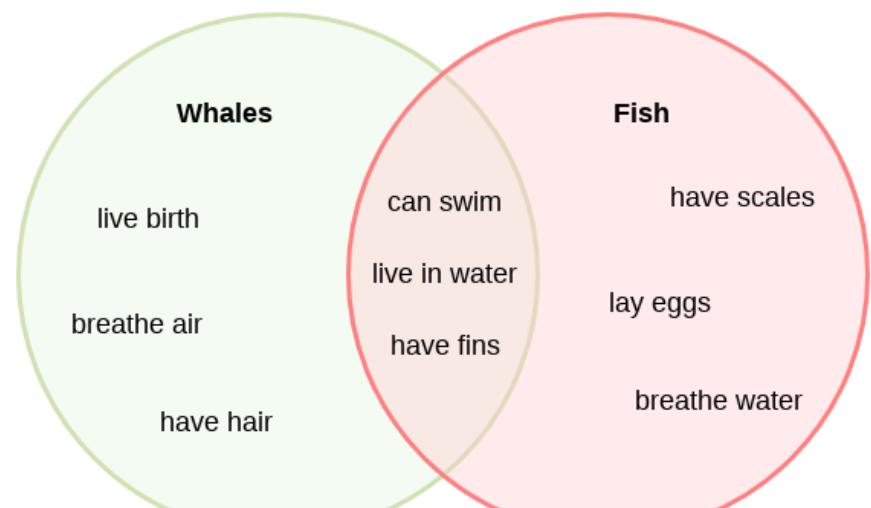
$$N = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

Oder beschreibend:

$$M = \{x \mid x \in \mathbb{N} \wedge x < 4\}$$

$$N = \{x \mid x \in \mathbb{N} \wedge x < 11\}$$

## Venn Diagramm



# Mengenlehre 2/6

Mengenverbindungen:

- Element von:  $x \in M$

Erklärung:  $x$  ist ein Element von  $M$

- Kein Element von:  $x \notin M$

Erklärung:  $x$  ist kein Element von  $M$

- Teilmenge:  $M \subseteq N$

Erklärung:  $M$  ist eine Teilmenge von  $N$

- Obermenge:  $M \supseteq N$

Erklärung:  $M$  ist eine Obermenge von  $N$

- Nullmenge:  $\emptyset$

Erklärung: Die Menge enthält keine Elemente

- Schnittmenge:  $M \cap N$  wenn  $M \cap N \neq \emptyset$

Erklärung: Die Menge enthält alle Elemente, die in  $M$  und  $N$  enthalten sind

- Vereinigungsmenge:  $M \cup N$

Erklärung: Die Menge enthält alle Elemente, die in  $M$  oder  $N$  enthalten sind

- Differenzmenge:  $M \setminus N$

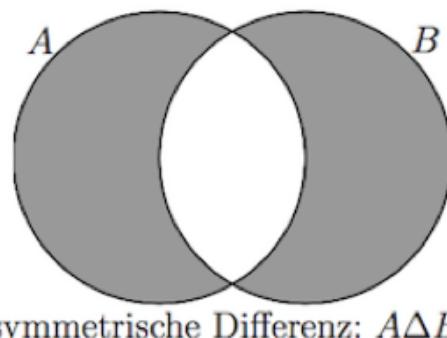
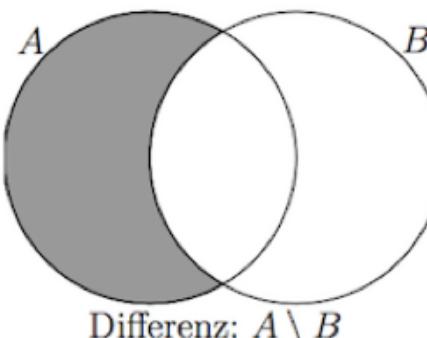
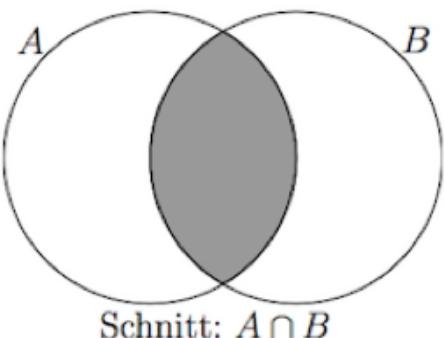
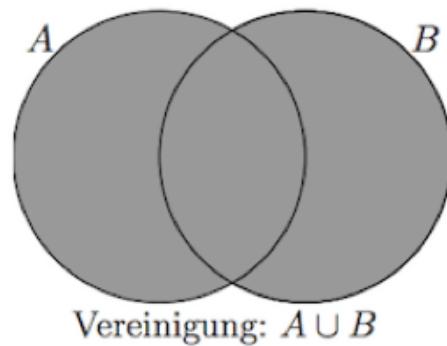
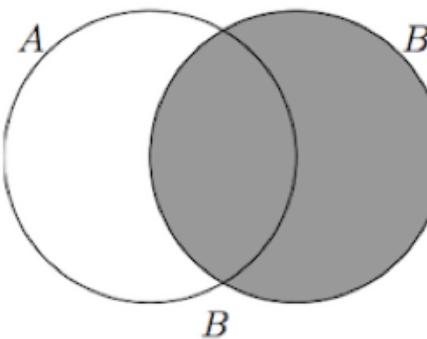
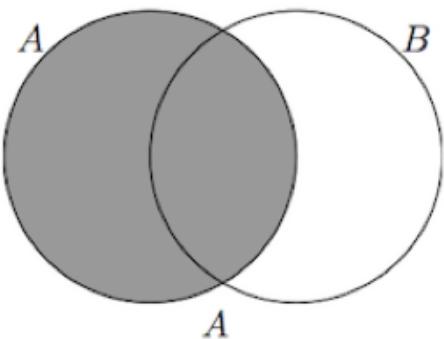
Erklärung: Die Menge enthält alle Elemente, die in  $M$  enthalten sind, aber nicht in  $N$

- Symmetrische Differenz:  $M \triangle N$

Erklärung: Die Menge enthält alle Elemente, die in  $M$  oder  $N$  enthalten sind, aber nicht in beiden

# Mengenlehre 3/6

Venn Diagramme für Mengenverbindungen:



# Mengenlehre 4/6

## Anwendungen in der Informatik (Beispiele)

- Zustandsraum eines Programms

$$Q = \{q_1, q_2, q_3\}$$

- Datenbanken (SQL) abfragen:

```
SELECT * FROM A INTERSECT SELECT * FROM B
```

- Python code:

```
# Erstellen von Mengen
set1 = {1, 2, 3}
set2 = {3, 4, 5}

# Vereinigung
union_set = set1 | set2 # Ergebnis: {1, 2, 3, 4, 5}

# Überschneidung
intersect_set = set1 & set2 # Ergebnis: {3}
```

# Mengenlehre 5/6

## Übungen

1. Definieren Sie eine Menge, die die ersten fünf Primzahlen enthält.

2. Gegeben sind die Mengen  $A = \{1, 3, 5\}$  und  $B = \{5, 6, 7\}$ .

Finden Sie  $A \cup B$ ,  $A \cap B$  und  $A \setminus B$ .

3. Gegeben sind die Mengen  $C = \{2, 4, 6\}$  und  $D = \{2, 4, 6, 8\}$ .

Ist  $C \subseteq D$ ?

4. Sind die Mengen  $\{1, 2\}$  und  $\{2, 1\}$  gleich?

5. Definieren Sie eine Menge, die die ersten fünf Quadratzahlen enthält.

6. Gegeben sind die Mengen  $E = \{2, 4, 6\}$  und  $F = \{4, 5, 6\}$ .

Finden Sie  $E \triangle F$ ,  $E \cap F$  und  $E \cup F$ .

7. Gegeben sind die Mengen  $G = \{10, 20, 30\}$  und  $H = \{20, 30, 40\}$ .

Finden Sie  $G \cap H$ ,  $G \cup H$  und  $G \setminus H$ .

8. Gegeben ist die Menge  $I = \{3, 6, 9, 12\}$ .

Ist  $6 \in I$ ?

# Mengenlehre 6/6

## Lösungen

1.  $M = \{2, 3, 5, 7, 11\}$

2.  $A \cup B = \{1, 3, 5, 6, 7\}$

$$A \cap B = \{5\}$$

$$A \setminus B = \{1, 3\}$$

3. Ja,  $C \subseteq D$

4. Ja, die Mengen sind gleich

5.  $M = \{1, 4, 9, 16, 25\}$

6.  $E \Delta F = \{2, 5\}$

$$E \cap F = \{4, 6\}$$

$$E \cup F = \{2, 4, 5, 6\}$$

7.  $G \cap H = \{20, 30\}$

$$G \cup H = \{10, 20, 30, 40\}$$

$$G \setminus H = \{10\}$$

8. Ja,  $6 \in I$

# Mengenlehre und Äquivalenzrelationen 1/8

## Inhaltsübersicht

- **Kartesisches Produkt**

- Definition

- Beispiel

- **Relationen**

- Definition

- Beispiel

- **Äquivalenzrelationen**

- Reflexivität

- Symmetrie

- Transitivität

- **Zuordnungsvorschrift**

- **Funktionsvorschrift**

- **Funktionen**

- **Operatorien**

- Injectivity

- Surjectivity

- Bijjectivity

- Totality

# Mengenlehre und Äquivalenzrelationen 2/8

## Kartesisches Produkt

Das kartesische Produkt von zwei Mengen  $A$  und  $B$ , notiert als  $A \times B$ ,

ist die Menge aller geordneten Paare  $(a, b)$ , wobei  $a \in A$  und  $b \in B$ .

Beispiel:

Wenn  $A = \{1, 2\}$  und  $B = \{x, y\}$ , dann ist

$$A \times B = \{(1, x), (1, y), (2, x), (2, y)\}.$$

## Relationen

Eine Relation beschreibt eine Beziehung zwischen zwei Mengen.

Eine Relation  $\sim$  zwischen zwei Mengen  $A$  und  $B$  ist eine Teilmenge von  $A \times B$ .

Wenn  $(a, b) \in \sim$ , dann schreiben wir  $a \sim b$ .

Beispiel mit Relation: "Ist gleich mod 2"

$$A = \{1, 2, 3\}$$

$$B = \{1, 2, 3\}$$

$$A \times B = \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)\}$$

Das heißt  $a \sim b$ , wenn  $a$  und  $b$  den gleichen Rest bei der Division durch 2 haben.

*Hier wäre es so, dass alle Paare mit entweder nur geraden oder nur ungeraden Zahlen in der Relation sind.*

# Mengenlehre und Äquivalenzrelationen 3/8

## Äquivalenzrelationen

Eine Äquivalenzrelation auf einer Menge  $M$  ist eine spezielle Art von Relation, die die folgenden Eigenschaften erfüllt:

- Reflexivität

Für alle ( $a \in M$ ), ( $a \sim a$ ).

- Symmetrie

Für alle ( $a, b \in M$ ), wenn ( $a \sim b$ ), dann ( $b \sim a$ ).

- Transitivität

Für alle ( $a, b, c \in M$ ), wenn ( $a \sim b$ ) und ( $b \sim c$ ), dann ( $a \sim c$ ).

# Mengenlehre und Äquivalenzrelationen 4/8

Zuordnungsvorschrift:

Die Zuordnungsvorschrift ist die Menge aller Paare  $(x, y)$ , die die Relation  $R$  erfüllen.

Beispiel:

$f : \mathbb{R} \rightarrow \mathbb{R}$  mit

$$f(x) = 4x + 6$$

Funktionsvorschrift:

Die Funktionsvorschrift stellt eine alternative Darstellung der Zuordnungsvorschrift dar.

$$f = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid y = 4x + 6\}$$

Eine Funktionsvorschrift ist keine Funktion.

Die Angabe des Quell- und Zielbereichs ist hierfür notwendig.

## Funktionen

Eine Funktion  $f : A \rightarrow B$  weist jedem Element  $a \in A$  genau ein Element  $f(a) \in B$  zu.

Funktionen können wie Mengen bzw. Relationen auf viele Arten dargestellt werden:

- durch eine Funktionsvorschrift
- grafisch durch ein Koordinatensystem
- durch eine Wertetabelle

Beispiele:

$$f(x) = x^2$$

$$g(x) = \sin(x)$$

# Mengenlehre und Äquivalenzrelationen 5/8

Injektivität (linkseindeutig)

Eine Funktion  $f : A \rightarrow B$  ist injektiv, wenn jedes Element in  $B$  von höchstens einem Element aus  $A$  als Funktionswert angenommen wird.

Das Prinzip der Injektivität:

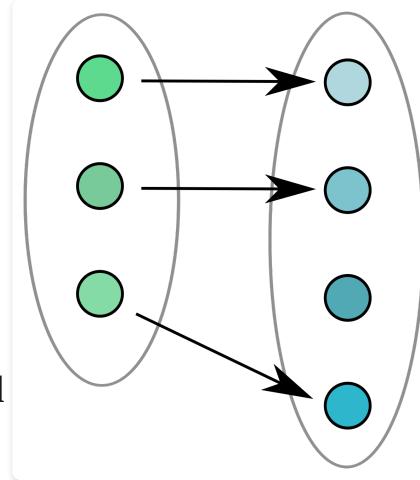
Jeder Punkt in der Zielmenge  $B$  wird höchstens einmal getroffen

Beispiel:

$f : N \rightarrow N$  mit  $f(x) = x^2$  ist injektiv. (Jede Zahl wird nur einmal getroffen)

$f : R \rightarrow R$  mit  $f(x) = x^2$  ist nicht injektiv.  
(Negative Zahl wird zweimal getroffen)

Diagramm zur Injektivität:



# Mengenlehre und Äquivalenzrelationen 6/8

## Surjektivität (rechtstotal)

Eine Funktion  $f : A \rightarrow B$  ist surjektiv, wenn für jedes  $b \in B$  ein  $a \in A$  existiert, so dass  $f(a) = b$ .

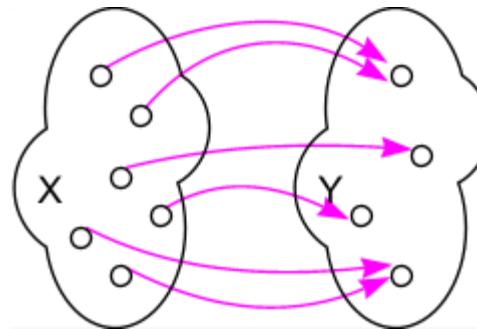
Das Prinzip der Surjektivität:

Jeder Punkt in der Zielmenge  $Y$  wird mindestens einmal getroffen

Beispiele:

- $f : R \rightarrow R$  mit  $f(x) = x^2$  ist surjektiv.  
(Jedes Element in der Zielmenge wird getroffen)
- $f : N \rightarrow N$  mit  $f(x) = x^2$  ist nicht surjektiv.  
(z.B. die 3 wird nicht getroffen da es keine Wurzel aus 3 gibt die in  $N$  liegt)

Diagramm zur Surjektivität:



# Mengenlehre und Äquivalenzrelationen 7/8

## Bijektivität

Eine Funktion ist bijektiv, wenn sie sowohl injektiv als auch surjektiv ist.

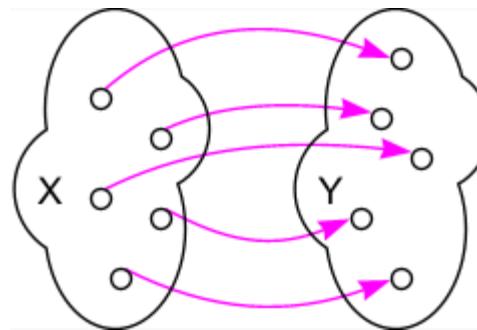
Das Prinzip der Bijektivität:

Jeder Punkt in der Zielmenge  $Y$  wird genau einmal getroffen.

Beispiele:

- $f : R \rightarrow R$  mit  $f(x) = x^2$  ist bijektiv.  
(Jeder Punkt in der Ziermenge wird **EXAKT** einmal getroffen)

Diagramm zur Bijektivität:



# Mengenlehre und Äquivalenzrelationen 8/8

## Totalität und Partielle Funktionen

Eine Funktion  $f : A \rightarrow B$  ist total, wenn sie jedem  $a \in A$  ein  $f(a) \in B$  zuweist (d.h., die Funktion ist definiert für alle  $a \in A$ ).

Eine Funktion ist partiell, wenn sie nicht total ist. Das heißt es gibt mindestens ein  $a \in A$ , für das  $f(a)$  nicht definiert ist.

Zum Beispiel ist  $f(x) = \frac{1}{x}$  eine partielle Funktion, wenn  $A = \mathbb{R}$  und  $B = \mathbb{R}$ . Der Grund ist, dass  $f(0)$  nicht definiert ist.

Außerdem ist  $f(x) = \sqrt{x}$  eine partielle Funktion, wenn  $A = \mathbb{R}$  und  $B = \mathbb{R}$ , da  $f(x)$  nicht definiert ist für  $x < 0$ .

Beispiel:

- $f : R \rightarrow R$  mit  $f(x) = \sqrt{x}$  ist total.  
(Jedes Element in der Quellmenge hat ein Element in der Zielmenge)
- $f : N \rightarrow N$  mit  $f(x) = \sqrt{x}$  ist partiell.  
(z.B. die 3 hat kein Element in der Zielmenge da es keine Wurzel aus 3 gibt die in N liegt)

# Software Lizenzen

Was ist eine Software Lizenz?

*Eine Software Lizenz ist eine Vereinbarung zwischen dem Lizenzgeber und dem Lizenznehmer.*

*Der Lizenzgeber ist der Urheber der Software und der Lizenznehmer ist der Käufer/Benutzer der Software.*

*Die Lizenz regelt die Nutzung der Software.*

*Die Lizenz kann auch die Weitergabe der Software regeln.*

Welche Arten von Software Lizenzen gibt es?

*Es gibt viele verschiedene Arten von Software Lizenzen.*

*Die bekanntesten sind die Open Source Lizenzen und die Closed Source Lizenzen.*

*Open Source Lizenzen erlauben es dem Lizenznehmer die Software zu verändern und weiterzugeben.*

*Closed Source Lizenzen erlauben dies nicht.*

*Die bekanntesten Open Source Lizenzen sind die GPL und die MIT Lizenz.*

*Die bekanntesten Closed Source Lizenzen sind die Microsoft EULA und die Adobe EULA.*

# MIT-Lizenz

## Einführung

- Eine der bekanntesten permissiven Open-Source-Lizenzen.
- Entwickelt am Massachusetts Institute of Technology.

## Hauptmerkmale

- Erlaubt Nutzung, Modifikation und Distribution.
- Minimale Einschränkungen, auch für kommerzielle Zwecke.

## Vorteile und Nachteile

- Vorteil: Große Freiheit und Flexibilität.
- Eventueller Nachteil: Kein Copyleft, abgeleitete Werke könnten proprietär sein.

## Anwendungsbeispiele

- Viele Open-Source-Projekte wie Ruby, Node.js.
- Beliebt bei Start-ups und Tech-Unternehmen.

## Rechtliche Aspekte

- Keine Gewährleistung oder Haftung.
- Einfache Einbindung in Projekte.

# GNU GPLv3

## Einführung

- Lizenz von Free Software Foundation entwickelt.
- Ziel: Schutz der Freiheiten von Endnutzern.

## Hauptmerkmale

- Copyleft-Lizenz: Abgeleitete Werke müssen ebenfalls unter GPLv3 stehen.
- Pflicht zur Offenlegung des Quellcodes.

## Vorteile und Nachteile

- Vorteil: Schützt die Freiheit der Software.
- Nachteil: Kann mit einigen proprietären Softwaremodellen inkompatibel sein.

## Anwendungsbeispiele

- Linux-Kernel, WordPress, GIMP.

## Rechtliche Aspekte

- Robuste Lizenz mit starkem Copyleft.
- Einige Unternehmen meiden sie wegen ihrer Einschränkungen.

# Apache 2.0

## Einführung

- Von der Apache Software Foundation entwickelt.
- Kombiniert permissive Eigenschaften mit Patentklauseln.

## Hauptmerkmale

- Permissive Lizenz mit Patentnichtangriffsklausel.
- Erlaubt kommerzielle Nutzung und Modifikation.

## Vorteile und Nachteile

- Vorteil: Schutz vor Patentklagen.
- Nachteil: Nicht so streng wie Copyleft-Lizenzen.

## Anwendungsbeispiele

- Apache HTTP Server, Apache Kafka, Android OS.

## Rechtliche Aspekte

- Bietet Schutz gegen Patenttroll-Angriffe.
- Beliebt in der Business-Community.

# BSD-Lizenzen (2- & 3-Klausel)

## Einführung

- Ursprung an der University of California, Berkeley.
- Ähnlich wie MIT, aber mit Unterschieden.

## Hauptmerkmale

- Permissive Lizenz, einige Versionen erfordern Namensnennung.

## Vorteile und Nachteile

- Vorteil: Einfach zu verwenden und zu integrieren.
- Nachteil: Kein Copyleft.

## Anwendungsbeispiele

- FreeBSD, OpenBSD, Teile von macOS.

## Rechtliche Aspekte

- Einfach und unkompliziert, aber ohne starken rechtlichen Schutz.

# GNU AGPLv3

## Einführung

- Erweiterung der GPLv3 für netzwerkbasierte Software.
- Schützt Benutzerfreundlichkeit bei Software-as-a-Service.

## Hauptmerkmale

- Copyleft-Lizenz für Netzwerkdienste.
- Pflicht zur Offenlegung des Quellcodes.

## Vorteile und Nachteile

- Vorteil: Schützt die Freiheit in Cloud-Umgebungen.
- Nachteil: Kann für einige Cloud-Dienste restriktiv sein.

## Anwendungsbeispiele

- MongoDB, Nextcloud, Mastodon.

## Rechtliche Aspekte

- Einzigartige Lizenz, die den Cloud-Bereich adressiert.
- Herausfordernd für einige SaaS-Modelle.

# Business Source License (BSL)

## Einführung

- Eine Lizenz, die zwischen Open Source und proprietärer Software steht.
- Entwickelt von Michael Widenius, dem Hauptautor von MySQL.

## Hauptmerkmale

- Ermöglicht die kostenlose Nutzung, Modifikation und Verteilung der Software mit einigen Einschränkungen.
- Nach einer festgelegten "Umstellungsperiode" wird die Software in eine Open-Source-Lizenz umgewandelt.

## Vorteile und Nachteile

- Vorteil: Bietet den Entwicklern eine Einnahmequelle, während sie der Gemeinschaft zurückgeben.
- Nachteil: Kann als nicht "wahrhaftig" Open Source angesehen werden.

## Anwendungsbeispiele

- Einige Datenbankprodukte und Business-Tools verwenden BSL.

## Rechtliche Aspekte

- Einzigartiges Lizenzmodell, das kommerzielle Interessen mit Open-Source-Prinzipien in Einklang bringt.
- Wichtig zu beachten: die genauen Bedingungen und die Umstellungsperiode.