08.08.2024

Zahlensysteme

TECH STARTER

Agenda

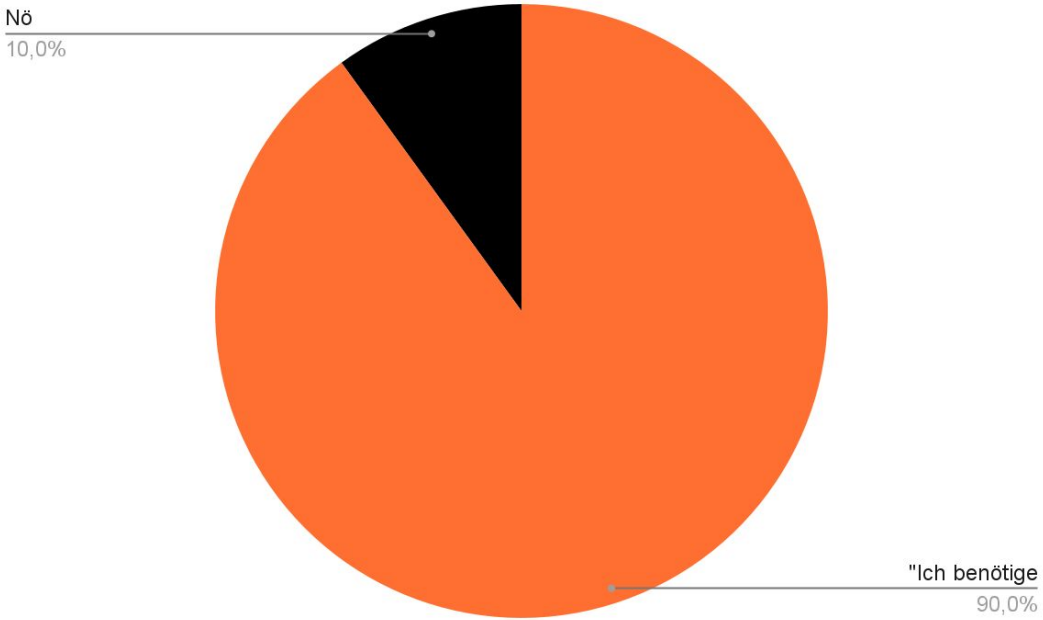
1. Mathematik in der Informatik
2. Boolsche und Aussagenlogik
3. Zahlensysteme: Binär, Dezimal, Hexadezimal
4. Umrechnung
5. Maßeinheiten und Präfixe in der IT

Muss ich als Informatiker richtig gut in Mathe sein?

Schätz mal...

90%

Laut Institut für Arbeitsmarkt- und Berufsforschung (IAB) benötigen etwa 90 % der Informatiker in Deutschland fundierte Mathematikkenntnisse.



Grundlagen von IT-Systemen

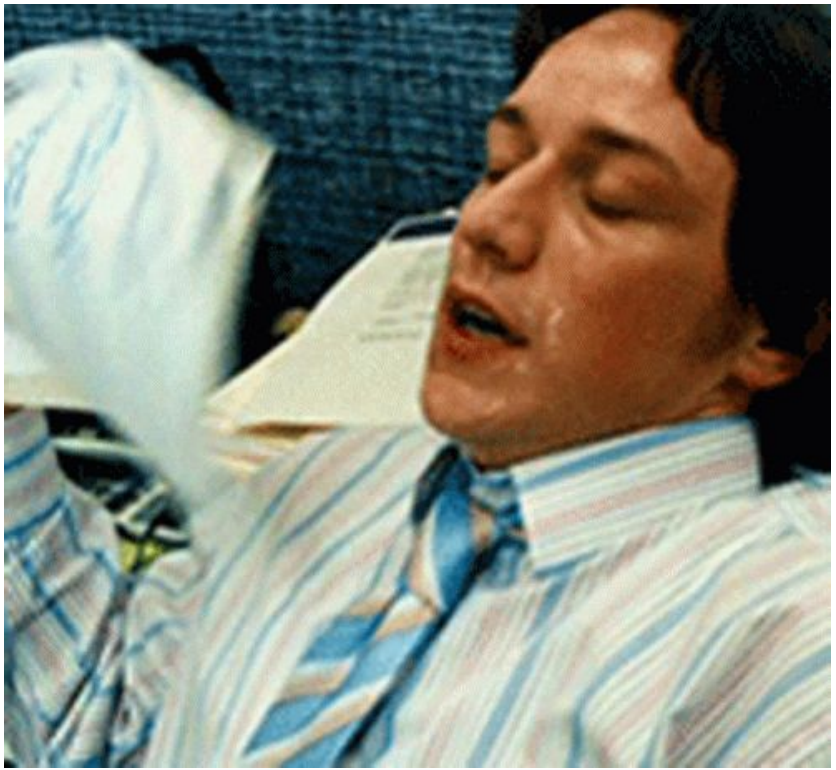
Muss ich ein Mathegenie sein, um alles zu verstehen?

Schätz mal...

TECH STARTER

Nö

Es reicht ein grundlegendes Verständnis von Logik und Algorithmik.



"In der Welt der Informatik geht es genauso viel um Computer, wie in der Astronomie um Teleskope. In beiden Disziplinen geht es darum, zu verstehen, wie die Welt funktioniert und wie man Probleme lösen kann und dafür ist die Mathematik das grundlegende Werkzeug." (Paul Graham)

Die Mathematik ist das Herzstück der Informatik, sie mag abstrakt erscheinen, ist aber der Klebstoff, der die vielen Aspekte der Informatik zusammenhält und ihr Tiefe verleiht.

Mathematik in der Informatik

- ➔ Mathematik bietet die Werkzeuge zur präzisen und effizienten **Problemlösung**
- ➔ Sie hilft die **Komplexität** von Daten und Algorithmen zu verstehen und zu beherrschen

Mathematik in der Informatik

➔ **Zahlensysteme**: Binär, dezimal und hexadezimal sind essentiell für das “Denken” und “Kommunizieren” von Computern

Mathematik in der Informatik

- ➔ **Zahlensysteme**: Binär, dezimal und hexadezimal sind essentiell für das “Denken” und “Kommunizieren” von Computern
- ➔ **Boolesche Logik** und **Aussagenlogik** bilden das Fundament aller Programmiersprachen und ermöglichen die Anwendung von Computern zur Ausführung bestimmter Aufgaben

Mathematik in der Informatik

- ➔ Mathematik ist mehr als Regeln und Formeln, sie ist eine **Denkweise** und eine Art, die Welt zu sehen und zu verstehen.
- ➔ Sie ist eine **Sprache**, die tiefgreifende und bedeutungsvolle Interaktionen mit Computern ermöglicht.

Welchen Zustand kann eine Glühbirne haben?

Schätz mal...

Zustände

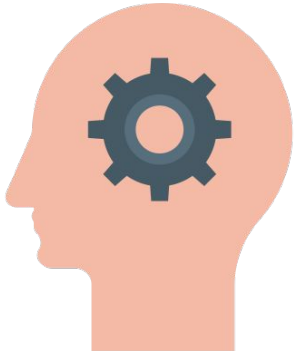
➔ 0
➔ 1

Boolsche und Aussagenlogik

➔ Herzschlag der Informatik: **Logik** ist zentral für den Betrieb von Computern und die Lösung komplexer Probleme

Boolsche und Aussagenlogik

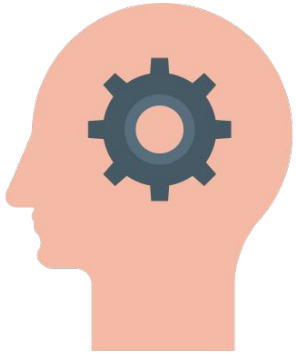
Formale Logik



- ➔ Strukturiert Argumente und Aussagen präzise
- ➔ Grundbaustein für Softwareentwicklung und algorithmische Problemstellungen

Boolsche und Aussagenlogik

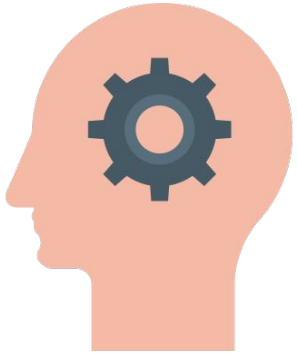
Boolesche Logik



- ➔ Benannt nach **George Boole**
- ➔ Ermöglicht Entscheidungen durch **logische Auswertung** (wahr/falsch)
- ➔ Vier grundlegende Operationen: AND, OR, NOT, XOR

Boolsche und Aussagenlogik

Boolsche Operatoren



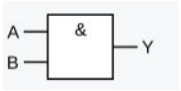
- ➔ AND, OR, NOT, XOR
- ➔ Mit Operatoren Bedingungen formulieren und Entscheidungen treffen
- ➔ Wichtig für die Verarbeitung binärer Daten
- ➔ In Programmierung für komplexe Kontrollstrukturen

Hast du schon mal etwas von Logikgattern gehört?

Schätz mal...

Boolsche und Aussagenlogik

AND-Operator



A	B	A AND B
Wahr	Wahr	Wahr
Wahr	Falsch	Falsch
Falsch	Wahr	Falsch
Falsch	Falsch	Falsch

- ➔ symbolisiert durch ∧ oder &
- ➔ Wenn wir zwei Aussagen, sagen wir A und B, mit dem AND-Operator verknüpfen, so ist das Ergebnis nur dann wahr, wenn sowohl A als auch B wahr sind.
- ➔ Wenn eine oder beide Aussagen falsch, ist das Ergebnis falsch

Boolsche und Aussagenlogik

AND-Operator

A	B	A AND B
Wahr	Wahr	Wahr
Wahr	Falsch	Falsch
Falsch	Wahr	Falsch
Falsch	Falsch	Falsch

- ➔ symbolisiert durch ∧ oder &
- ➔ Wenn wir zwei Aussagen, sagen wir A und B, mit dem AND-Operator verknüpfen, so ist das Ergebnis nur dann wahr, wenn sowohl A als auch B wahr sind

In der Programmierung wird der AND-Operator oft verwendet, um Bedingungen zu überprüfen, bei denen mehrere Kriterien **gleichzeitig** erfüllt sein müssen. Er spielt eine entscheidende Rolle in Kontrollstrukturen wie Schleifen und Verzweigungen.

Boolsche und Aussagenlogik

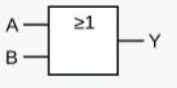
AND-Operator

A	B	A AND B
Wahr	Wahr	Wahr
Wahr	Falsch	Falsch
Falsch	Wahr	Falsch
Falsch	Falsch	Falsch

Nehmen wir an, dass A die Aussage **"Es regnet"** und B die Aussage **"Ich habe einen Regenschirm"** ist. Die zusammengesetzte Aussage **"Es regnet UND ich habe einen Regenschirm"** ist nur dann wahr, wenn sowohl A als auch B wahr sind - also, wenn es wirklich regnet und ich tatsächlich einen Regenschirm dabei habe. Ist eine oder gar beide Aussagen falsch, dann ist auch die zusammengesetzte Aussage falsch.

Boolsche und Aussagenlogik

OR-Operator



A	B	A OR B
Wahr	Wahr	Wahr
Wahr	Falsch	Wahr
Falsch	Wahr	Wahr
Falsch	Falsch	Falsch

- ➔ symbolisiert durch ∨ oder |
- ➔ Wenn wir zwei Aussagen, sagen wir A und B, mit dem OR-Operator verknüpfen, so ist das Ergebnis wahr, wenn mindestens eine der beiden Aussagen wahr ist.
- ➔ Nur wenn beide Aussagen falsch ist, ist das Ergebnis falsch.

Boolsche und Aussagenlogik

OR-Operator

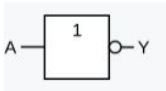
A	B	A OR B
Wahr	Wahr	Wahr
Wahr	Falsch	Wahr
Falsch	Wahr	Wahr
Falsch	Falsch	Falsch

- ➔ symbolisiert durch ∨ oder |
- ➔ Wenn wir zwei Aussagen, sagen wir A und B, mit dem OR-Operator verknüpfen, so ist das Ergebnis wahr, wenn mindestens eine der beiden Aussagen wahr ist

In der Programmierung wird der OR-Operator oft dazu verwendet, Bedingungen zu überprüfen, bei denen **mindestens eines** von mehreren Kriterien erfüllt sein muss. Er wird häufig in Kontrollstrukturen eingesetzt, um **flexible Bedingungen** zu schaffen, bei denen mehrere verschiedene Zustände zu einem bestimmten Ergebnis führen können.

Boolsche und Aussagenlogik

NOT-Operator



A	NOT A
Wahr	Falsch
Falsch	Wahr

- ➔ dargestellt durch ¬ oder !
- ➔ Kein binärer Operator, sondern unärer: er wirkt nur auf eine einzelne Aussage und invertiert den Wahrheitswert
- ➔ Wenn A wahr ist, dann ist NOT A falsch und umgekehrt.
- ➔ Der NOT-Operator wird verwendet, um den Wahrheitswert einer Aussage umzukehren und wird oft in mathematischen Beweisen, in der Programmierung und in der logischen Analyse verwendet.

Boolsche und Aussagenlogik

NOT-Operator

A	NOT A
Wahr	Falsch
Falsch	Wahr

- ➔ dargestellt durch ¬ oder !
- ➔ Kein binärer Operator, sondern unärer: er wirkt nur auf eine einzelne Aussage und invertiert den Wahrheitswert
- ➔ Wenn A wahr ist, dann ist NOT A falsch und umgekehrt.
- ➔ Der NOT-Operator wird verwendet, um den Wahrheitswert einer Au age in n, in der er ndet.

In der Programmierung ist der NOT-Operator ein kraftvolles Werkzeug, das oft dazu verwendet wird, Bedingungen umzukehren oder zu negieren. Es ist besonders nützlich in Kontrollstrukturen, wenn eine Bedingung erfüllt sein muss, die das Gegenteil einer bestimmten Aussage ist.

Boolsche und Aussagenlogik

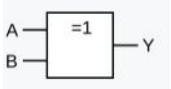
NOT-Operator

A	NOT A
Wahr	Falsch
Falsch	Wahr

Wenn A die Aussage **"Es regnet"** ist, dann ist NOT A die Aussage **"Es regnet nicht"**. Wenn es tatsächlich regnet (A ist wahr), dann ist die Aussage "Es regnet nicht" (NOT A) falsch. Wenn es nicht regnet (A ist falsch), dann ist "Es regnet nicht" (NOT A) wahr.

Boolsche und Aussagenlogik

XOR-Operator



A	B	A XOR B
Wahr	Wahr	Falsch
Wahr	Falsch	Wahr
Falsch	Wahr	Wahr
Falsch	Falsch	Falsch

- ➔ dargestellt durch ⊕ oder ^
- ➔ Ergebnis genau dann wahr ist, wenn eine der beiden Aussagen wahr ist, aber nicht beide.
- ➔ Wenn wir zwei Aussagen, sagen wir A und B, mit dem XOR-Operator verknüpfen, dann ist die zusammengesetzte Aussage "A XOR B" nur dann wahr, wenn genau eine der beiden Aussagen wahr ist.

Boolsche und Aussagenlogik

XOR-Operator

A	B	A XOR B
Wahr	Wahr	Falsch
Wahr	Falsch	Wahr
Falsch	Wahr	Wahr
Falsch	Falsch	Falsch

- ➔ dargestellt durch ⊕ oder ^
- ➔ Ergebnis genau dann wahr ist, wenn eine der beiden Aussagen wahr ist, aber nicht beide.
- ➔ Wenn wir zwei Aussagen, sagen wir A und B, mit dem XOR-Operator verknüpfen, dann ist die zusammengeset e Aussage n wahr, der beiden .

In der Programmierung wird der XOR-Operator oft verwendet, um Bedingungen zu überprüfen, bei denen genau eines von zwei Kriterien erfüllt sein muss. Er wird auch in Bereichen wie der Kryptographie verwendet, um Daten sicher zu übertragen

Boolsche und Aussagenlogik

XOR-Operator

A	B	A XOR B
Wahr	Wahr	Falsch
Wahr	Falsch	Wahr
Falsch	Wahr	Wahr
Falsch	Falsch	Falsch

Wenn A die Aussage **"Es regnet"** und B die Aussage **"Es ist sonnig"** ist, dann ist die Aussage **"Es regnet XOR es ist sonnig"** nur dann wahr, wenn es entweder regnet (aber nicht sonnig ist), oder wenn es sonnig ist (aber nicht regnet). Wenn es sowohl regnet als auch sonnig ist, oder wenn es weder regnet noch sonnig ist, ist die zusammengesetzte Aussage falsch. XOR wird oft in Bereichen wie der Fehlererkennung und -korrektur in der Datenkommunikation eingesetzt.

Boolsche und Aussagenlogik

Zusammenfassung

Operator	Zeichen	Regel	Beispiel
AND	A ∧ B A & B	A AND B ist wahr, wenn A und B beide wahr sind. Andernfalls ist es falsch.	*"Es regnet AND Es ist kalt" ist wahr, wenn sowohl "Es regnet" als auch "Es ist kalt" wahr sind.*
OR	A ∨ B A | B	A OR B ist wahr, wenn A oder B oder beide wahr sind. Andernfalls ist es falsch.	*"Es regnet OR Es ist kalt" ist wahr, wenn "Es regnet", "Es ist kalt", oder beides wahr ist.*
NOT	¬A !A	NOT A ist wahr, wenn A falsch istund umgekehrt.	*"NOT Es regnet" ist wahr, wenn "Es regnet" falsch ist.*
XOR	A ⊕ B A ^ B	A XOR B ist wahr, wenn genau einer der Operanden (A oder B, aber nicht beide) wahr ist. Andernfalls ist es falsch.	*"Es regnet XOR Es ist kalt" ist wahr, wenn entweder "Es regnet" oder "Es ist kalt" wahr ist, aber nicht beide.*

Boolsche und Aussagenlogik

Anwendung in der Programmierung

➔ Nicht nur abstrakte Mathematik, sondern Anwendung in der realen Welt wie in der Programmierung

Boolsche und Aussagenlogik

Anwendung in der Programmierung

Operator	Darstellung in Javascript	Beispiel-Code in Javascript	Beispiel-Code in Python

AND	**&&**	let A = true;
		let B = false;
		let result = A && B;
		// result ist falsch, weil B falsch ist
			A = True
			B = False
			result = A and B
			# result ist falsch, weil B falsch ist

OR	**||**	let A = true;
		let B = false;
		let result = A || B;
		// result ist wahr, weil A wahr ist
			A = True
			B = False
			result = A or B
			# result ist True, weil A wahr ist

NOT	**!**	let A = true;
		let result = !A;
		// result ist falsch, weil A wahr ist
			A = True
			result = not A
			# result ist falsch, weil A wahr ist

XOR	gibt es nicht direkt in JavaScript, kann aber mit einer Kombination von Operatoren erreicht werden	let A = true;
		let B = false;
		let result = (A && !B) || (!A && B);
		// result ist wahr
			A = True
			B = False
			result = A ^ B
			# result ist wahr

Boolsche und Aussagenlogik

Aussagenlogik

- ➔ Verknüpft Aussagen durch logische Operatoren
- ➔ Fundament für algorithmische Logik, Programmiersprachen, KI

Boolsche und Aussagenlogik

Aussagenlogik

- ➔ Aussage = klare Sätze, die entweder wahr oder falsch sein können (z.B. Die Sonne scheint)
- ➔ Verbindung von Aussagen mit logischen Operatoren (AND, OR, NOT, XOR) und neue Wahrheiten schaffen
- ➔ Logische Schlussfolgerung mithilfe von Aussagen: Wenn es wahr ist, dass "Alle Vögel Flügel haben" und "Ein Papagei ist ein Vogel", dann könnt ihr schlussfolgern, dass "Ein Papagei Flügel hat."

Boolsche und Aussagenlogik

Aussagenlogik

🔎 Nehmen wir zwei einfache Aussagen:

A: ***"Es regnet."***

B: ***"Der Boden ist nass."***

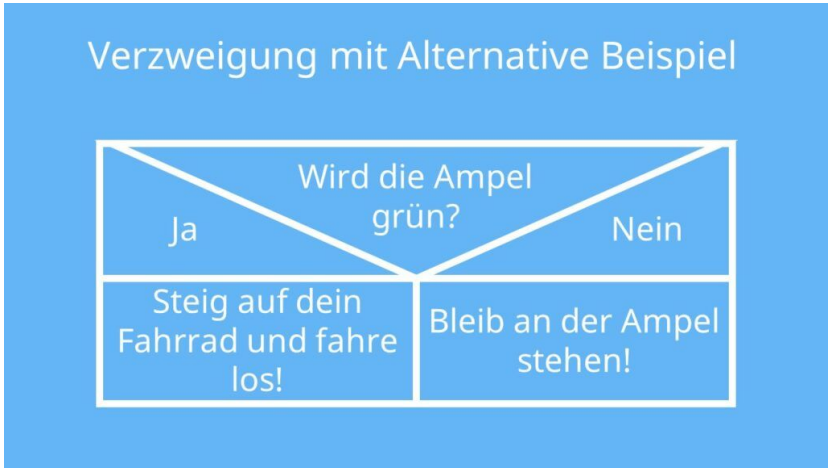
Wir können den AND-Operator verwenden, um eine neue Aussage zu bilden:

"Es regnet UND der Boden ist nass."

Diese zusammengesetzte Aussage ist nur dann wahr, wenn beide Teilaussagen wahr sind. Das bedeutet, wenn es regnet und der Boden nass ist, dann ist die gesamte Aussage wahr. Wenn einer der Teile nicht zutrifft, ist die ganze Aussage falsch.

Boolsche und Aussagenlogik

Struktogramm



➔ Verzweigungen als Kontrollstruktur im Programmiercode

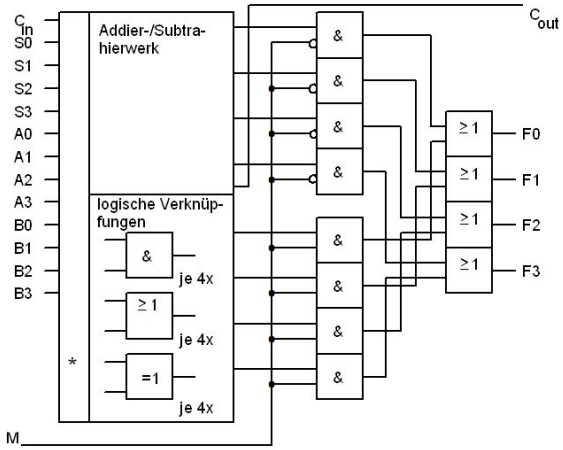
Boolsche und Aussagenlogik

Anwendung in Schaltkreisen

➔ Grundlage für Computerchips und komplexe Berechnungen

Boolsche und Aussagenlogik

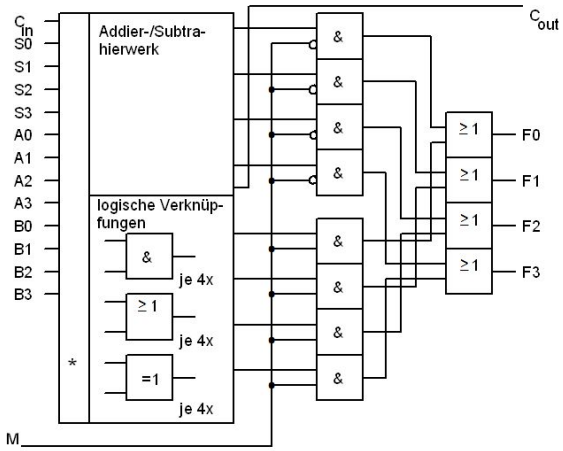
Anwendung in Schaltkreisen



➔ Grundlage für Computerchips und komplexe Berechnungen

Boolsche und Aussagenlogik

Anwendung in Schaltkreisen



Assembler-Programmierung

➔ Grundlage für Computerchips und komplexe Berechnungen

Boolsche und Aussagenlogik

Warum brauchen wir das?

- ➔ Logik ist das Rückgrat der Entscheidungsfindung in Computersystemen
- ➔ Ermöglicht die Formulierung von Instruktionen für Computer
- ➔ Unerlässlich für moderne Informatik
- ➔ Logik = Kunst der Kommunikation mit Computern und effiziente Problemlösung

Zahlensysteme

Warum brauchen wir das?

- ➔ Fundament, auf dem alle **Berechnungen** und **Datenverarbeitungen** aufbauen
- ➔ **Grammatik** der Zahlen
- ➔ Drei Haupttypen: binär, dezimal, hexadezimal

Zahlensysteme

Warum brauchen wir das?

Verschiedene Zahlensysteme existieren, weil sie jeweils für spezifische Anwendungen und in verschiedenen Kontexten optimiert sind. Das **Dezimalsystem** eignet sich gut für unseren **alltäglichen Gebrauch**, das **Binärsystem** ist ideal für **Computer** und das **Hexadezimalsystem** bietet eine effiziente Möglichkeit, **binäre Daten** darzustellen.

Binäres Zahlensystem

- ➔ Grundlegende Sprache von Computern
- ➔ Passt gut zu An-/Aus-Schaltern (Transistoren)
- ➔ Jede Zahl im Binärsystem wird durch eine Kombination dieser zwei Symbole dargestellt
- ➔ Methode, um Zahlen darzustellen, indem nur die Ziffern 0 und 1 verwendet werden.

Binäres Zahlensystem

$$111_2 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 = 1 + 2 + 4 = 7_{10}$$

$$111000_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 32 + 16 + 8 = 56_{10}$$

- ➔ Jede Stelle in einer binären Zahl repräsentiert eine Potenz von 2 (2^0, 2^1, ...)
- ➔ Horner-Schema:

Binäres Zahlensystem

Wie komme ich da drauf ?

$$111_2 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^3 = 1 + 2 + 8 = 11_{10}$$

$$111000_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 32 + 16 + 8 = 56_{10}$$

➔ Modulo-Rechnung = Division mit Rest

$$\begin{aligned} 56 &: 2 = 28\,R\,0 \\ 28 &: 2 = 14\,R\,0 \\ 14 &: 2 = 7\,R\,0 \\ 7 &: 2 = 3\,R\,1 \\ 3 &: 2 = 1\,R\,1 \\ 1 &: 2 = 0\,R\,1 \end{aligned}$$

Dezimales Zahlensystem

🔎 Nehmen wir die Dezimalzahl 2345:

- Die erste Stelle von rechts repräsentiert 10^0 und da hier eine 5 steht, wird dieser Wert mit 5 multipliziert.
- Die zweite Stelle von rechts repräsentiert 10^1 und da hier eine 4 steht, wird dieser Wert mit 4 multipliziert.
- Die dritte Stelle von rechts repräsentiert 10^2 und da hier eine 3 steht, wird dieser Wert mit 3 multipliziert.
- Die vierte Stelle von rechts repräsentiert 10^3 und da hier eine 2 steht, wird dieser Wert mit 2 multipliziert.

Die Summe der Werte ergibt:

5 * 10^0+ 4 * 10^1+ 3 * 10^2+ 2 * 10^3= 5 + 40 + 300 + 2000 = 2345

- ➔ Standard-Zahlensystem
- ➔ Ziffern 0 bis 9
- ➔ Basis für Berechnungen und Messungen
- ➔ Jede Zahl kann repräsentiert werden von einer Kombination aus 10er-Potenzen (ähnlich Horner-Schema in Binär)

Hexadezimales Zahlensystem

🔎 Nehmen wir die Hexadezimalzahl 1A3:

- Die erste Stelle von rechts repräsentiert 16^0 und da hier eine 3 steht, wird dieser Wert mit 3 multipliziert.
- Die zweite Stelle von rechts repräsentiert 16^1 und da hier ein A steht (was 10 entspricht), wird dieser Wert mit 10 multipliziert.
- Die dritte Stelle von rechts repräsentiert 16^2 und da hier eine 1 steht, wird dieser Wert mit 1 multipliziert.

Die Summe der Werte ergibt:

3 * 16^0+ 10 * 16^1+ 1 * 16^2= 3 + 160 + 256 = 419

TECH STARTER

- ➔ 16 Symbole
- ➔ Effiziente und kompakte Darstellung von Binärzahlen
- ➔ 0-9, A-F (für 10-15)
- ➔ Häufig in Systemprogrammierung oder Farbcodierung, um Binärdaten einfacher zu handhaben/interpretieren
- ➔ Jede Stelle in Hexadezimalzahl repräsentiert wieder eine Potenz von 16 ...

Überblick Zahlensysteme

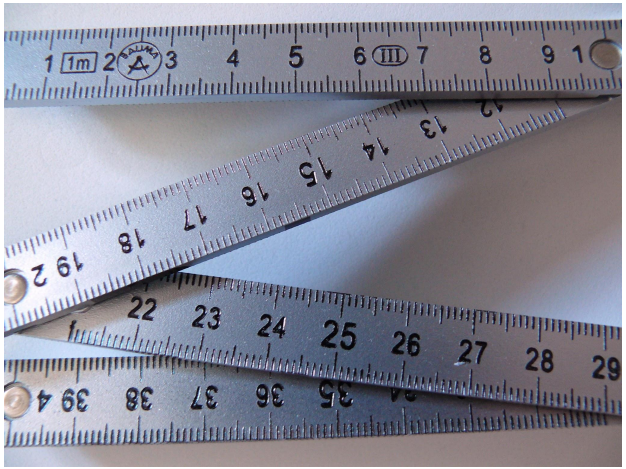
Zahlensystem	Anzahl der Symbole	Einsatzbereich	Kurze Erklärung
Binär	2 (0 und 1)	Computertechnik, logische Operationen	Ideal für Computer, da es nur zwei Zustände gibt, was den An-/Aus-Schaltern in Computern entspricht.
Dezimal	10 (0 bis 9)	Alltag, Wissenschaft, Handel	Alltäglich von Menschen verwendet, 10 Symbole entsprechen unserer Basiszählweise und bilden die Grundlage für unser Verständnis von Mathematik.
Hexadezimal	16 (0 bis 9, A-F)	Systemprogrammierung, Farbcodierung	Effizient für bestimmte Programmieraufgaben, da 16 Symbole eine kompakte Darstellung von Binärdaten ermöglichen. Verwendet Zahlen und Buchstaben zur Repräsentation.

Dezimal	Binär	Hexadezimal
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
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Was kennst du für Maßeinheiten in der IT?

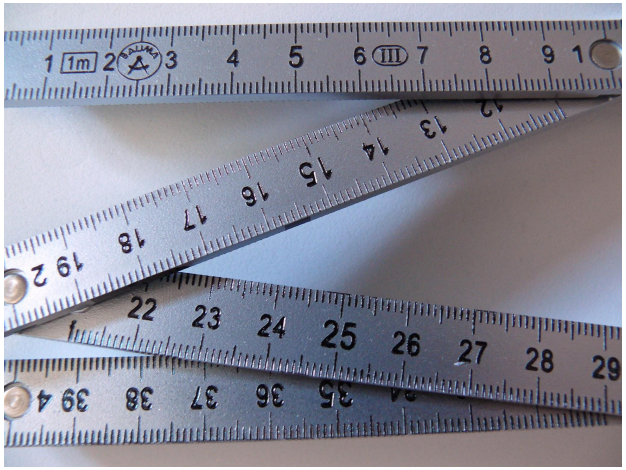
Schätz mal...

Maßeinheiten und Präfixe in der IT



- **Bedeutung:** Maßeinheiten und Präfixe formen die Struktur von Daten und Informationen in der IT.
- **Funktion:** Sie messen Raum, Geschwindigkeit und Effizienz von Daten.
- **Beispiele:** Gigabyte, Terabyte, Megahertz.
- **Relevanz:** Beschreiben die Menge an Information, die digital gespeichert oder übertragen werden kann.
- **Zweck der Präfixe:** Vereinfachen das Verständnis der Größenordnung.

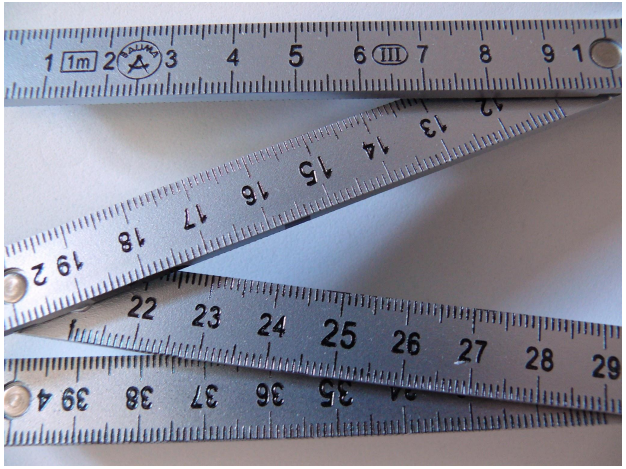
Maßeinheiten und Präfixe in der IT



Was sind Maßeinheiten & Präfixe?

- ➔ **Maßeinheiten:** Grundlagen zum Messen von Dingen wie Datenmenge (z.B., Byte).
- ➔ **Präfixe:** Etiketten, die Maßeinheiten vergrößern oder verkleinern (z.B., Kilo- für tausend).

Maßeinheiten und Präfixe in der IT

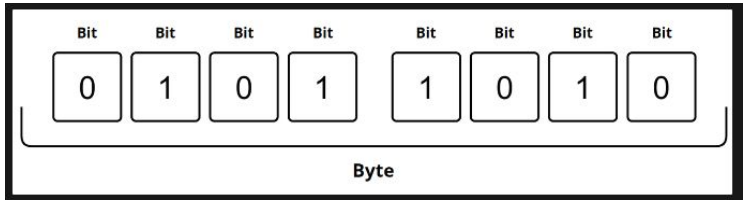


Warum verschiedene Maßeinheiten & Präfixe?

- ➔ **Zweck:** Beschreiben von Dingen in unterschiedlichen Größen und Mengen.
- ➔ **Beispiele:** Kilobyte, Megabyte, Gigabyte.
- ➔ **Nutzen:** Halten Zahlen übersichtlich und verständlich.

Maßeinheiten und Präfixe in der IT

Bits und Bytes



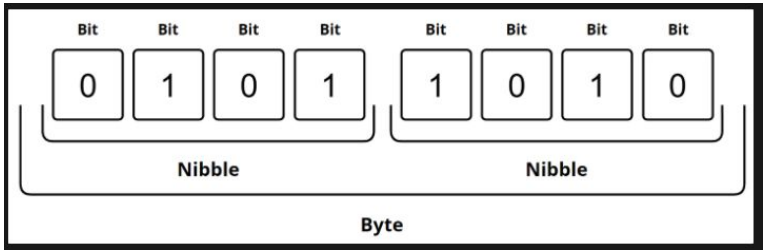
- ➔ **Bit:** Kleinste Informationseinheit, kann 0 oder 1 sein.
- ➔ **Byte:** Besteht aus 8 Bits, repräsentiert 256 verschiedene Werte.
- ➔ **Nützlichkeit:** Verwendet für die Darstellung von Zeichen in Textdateien.

Beispiel: Internetgeschwindigkeit

- ➔ **100 Megabit/s:** Entspricht 12,5 Megabyte/s.
- ➔ **Wichtigkeit:** Unterscheidung entscheidend für Verständnis von Datenübertragungsraten.

Maßeinheiten und Präfixe in der IT

Nibbles und Blöcke



Nibbles

- ➔ **Definition:** Halbbyte, besteht aus 4 Bits.
- ➔ **Nutzung:** Darstellung einer hexadezimalen Ziffer.
- ➔ **Beispiel:** Fehlererkennung und -korrektur.

Blöcke

- ➔ **Definition:** Gruppierung von Bytes.
- ➔ **Anwendung:** Dateispeicherung, Datenübertragung, Kryptografie.
- ➔ **Vorteil:** Effizientere Verwaltung und Kontrolle.

Maßeinheiten und Präfixe in der IT

SI-Präfixe

Präfix	Symbol	Potenz von 10	Bedeutung	Beispiel
Kilo-	k	103	Tausend	Kilobyte (KB): 1.000 Bytes
Mega-	M	106	Million	Megabyte (MB): 1.000.000 Bytes
Giga-	G	109	Milliarde	Gigabyte (GB): 1.000.000.000 Bytes
Tera-	T	1012	Billion (Tausend Milliarden)	Terabyte (TB): 1.000.000.000.000 Bytes
Peta-	P	1015	Billiarde (Million Milliarden)	Petaflop im Supercomputing
Exa-	E	1018	Trillion (Tausend Billionen)	Exabyte in großen Datenmengen
Zetta-	Z	1021	Trilliarde (Million Billionen)	Zettabyte im digitalen Universum
Yotta-	Y	1024	Quadrillion (Tausend Trillionen)	Yottabytes in theoretischen Berechnungen

Bedeutung: Standardisierte Präfixe zur Beschreibung großer oder kleiner Zahlen.

Anwendung in IT: Speicherplatz, Prozessorleistung, Internetgeschwindigkeit.

Beispiele:

- ➔ **Kilo-:** Tausend (1.000 Bytes)
- ➔ **Mega-:** Million (1.000.000 Bytes)
- ➔ **Giga-:** Milliarde (1.000.000.000 Bytes)

Maßeinheiten und Präfixe in der IT

Binäre Präfixe

Präfix	Symbol	Potenz von 2	Entsprechung in Bytes	Beispiel
Kibi-	Ki	210 (1.024)	1 KiB = 1.024 Bytes	1 KiB Speicherplatz
Mebi-	Mi	220 (1.048.576)	1 MiB = 1.024 KiB = 1.048.576 Bytes	1 MiB Speicherplatz
Gibi-	Gi	230 (1.073.741.824)	1 GiB = 1.024 MiB = 1.073.741.824 Bytes	1 GiB RAM
Tebi-	Ti	240 (1.099.511.627.776)	1 TiB = 1.024 GiB = 1.099.511.627.776 Bytes	1 TiB Festplatte
Pebi-	Pi	250 (1.125.899.906.842.624)	1 PiB = 1.024 TiB = 1.125.899.906.842.624 Bytes	Große Datenarchive
Exbi-	Ei	260 (1.152.921.504.606.846.976)	1 EiB = 1.024 PiB = 1.152.921.504.606.846.976 Bytes	Supercomputer-Speicher
Zebi-	Zi	270 (1.180.591.620.717.411.303.424)	1 ZiB = 1.024 EiB = 1.180.591.620.717.411.303.424 Bytes	Globale Datenmengen
Yobi-	Yi	280 (1.208.925.819.614.629.174.706.176)	1 YiB = 1.024 ZiB = 1.208.925.819.614.629.174.706.176 Bytes	Theoretische Speicherkapazitäten

Unterschied zu SI-Präfixen: Basieren auf Potenzen von 2.

Relevanz: Genauere Darstellung von Speichergrößen in der IT.

Beispiele:

- ➔ **Kibi-:** 2^{10} (1.024 Bytes)
- ➔ **Mebi-:** 2^{20} (1.048.576 Bytes)
- ➔ **Gibi-:** 2^{30} (1.073.741.824 Bytes)

Maßeinheiten und Präfixe in der IT

Binäre Präfixe

Präfix	Symbol	Potenz von 2	Entsprechung in Bytes	Beispiel
Kibi-	Ki	210 (1.024)	1 KiB = 1.024 Bytes	1 KiB Speicherplatz
Mebi-	Mi	220 (1.048.576)	1 MiB = 1.024 KiB = 1.048.576 Bytes	1 MiB Speicherplatz
Gibi-	Gi	230 (1.073.741.824)	1 GiB = 1.024 MiB = 1.073.741.824 Bytes	1 GiB RAM
Tebi-	Ti	240 (1.099.511.627.776)	1 TiB = 1.024 GiB = 1.099.511.627.776 Bytes	1 TiB Festplatte
Pebi-	Pi	250 (1.125.899.906.842.624)	1 PiB = 1.024 TiB = 1.125.899.906.842.624 Bytes	Große Datenarchive
Exbi-	Ei	260 (1.152.921.504.606.846.976)	1 EiB = 1.024 PiB = 1.152.921.504.606.846.976 Bytes	Supercomputer-Speicher
Zebi-	Zi	270 (1.180.591.620.717.411.303.424)	1 ZiB = 1.024 EiB = 1.180.591.620.717.411.303.424 Bytes	Globale Datenmengen
Yobi-	Yi	280 (1.208.925.819.614.629.174.706.176)	1 YiB = 1.024 ZiB = 1.208.925.819.614.629.174.706.176 Bytes	Theoretische Speicherkapazitäten

Unterschied zu SI-Präfixen: Basieren auf Potenzen von 2.

Relevanz: Genauere Darstellung von Speichergrößen in der IT.

Beispiele:

- **Kibi-:** 2^{10} (1.024 Bytes)
- **Mebi-:** 2^{20} (1
-

Binäre Präfixe werden speziell in der Informatik angewandt, da sie auf den binären Charakter eines Computers abgestimmt sind.