1 Bits and Bytes: Wie funktioniert ein Computer?

1.0.1. Jegliche Information (Texte, Musik, Videos) wird in Computern als Folge von **Bits** (= binary digits = Binärziffern) gespeichert, also durch eine Folge von Nullen und Einsen. Der Grund dafür ist, dass sich zwei Zustände (AN/AUS = ON/OFF = Spannung vorhanden/nicht vorhanden) elektronisch leicht realisieren lassen. Ein **Byte** ist eine Folge von 8 Bits.

1.1 Stellenwertsysteme allgemein

1.1.1. Es gibt viele Möglichkeiten, eine natürliche Zahl zu notieren. Hier sind drei Schreibweisen derselben Zahl:

- steinzeitlich (auch unäre Notation genannt): ●●●●●●●●●●●●●●●●●●●●●●●●
- römisch: XXIV
- heutzutage, im Dezimalsystem = Zehnersystem: 24

Unterschiedliche Notationen sind unterschiedlich gut zum Rechnen geeignet, weshalb es geschichtlich durchaus einen Unterschied gemacht hat (Effizienz in Handel und Verwaltung), welche Zahlenschreibweisen verwendet wurden bzw. überhaupt bekannt waren.

Heutzutage schreiben wir Zahlen meistens im Dezimalsystem, das heisst im *Stellenwertsystem zur Basis 10*. Die *indisch-arabischen Ziffern* 0, 1, . . . , 9, die wir verwenden, kann man bereits um etwa 300 v. Chr. in Indien nachweisen. Fibonacci (= Leonardo da Pisa) lernte das Dezimalsystem in Algerien kennen und verbreitete es mit seinem Buch «Liber abaci» («Buch des Rechnens»[1], vollendet 1202) in Italien und Europa. Erst im 15. Jahrhundert setzte es sich im deutschen Sprachraum gegen die römische Zahlschreibweise durch.

Statt der Basis 10 kann man jede beliebige andere natürlich Zahl ≥ 2 als Basis nehmen.

Erklärung 1.1.2 Stellenwertsysteme, erste Beispiele

Stellenwertsysteme sind gewisse Notationssysteme für Zahlen, in denen Zahlen durch Folgen von Ziffern dargestellt werden. Die *Stelle* (= Position) einer jeden Ziffer entscheidet über ihren *Wert*. Die Stellen werden von rechts her durchnummeriert, wobei die Stelle ganz rechts die Nummer 0 bekommt. Als **Basis** eines Stellenwertsystems kann man jede natürliche Zahl $b \geq 2$ nehmen. Die erlaubten Ziffern sind dann Symbole für die Zahlen von Null bis $b-1$.
Beispiele:

- **Stellenwertsystem zur Basis 10 = Dezimalsystem = Zehnersystem = 10er-System**
  Die erlaubten Ziffern sind 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (beachte: $9 = 10-1$). Die Ziffer an der i-ten Stelle gibt an, wie oft die Zahl 10^i genommen wird.

Stelle 4	Stelle 3	Stelle 2	Stelle 1	Stelle 0
10000er= 10^4er	1000er= 10^3er	100er= 10^2er	10er= 10^1er	1er= 10^0er

- **Stellenwertsystem zur Basis 5 = Fünfersystem = 5er-System**
  Die erlaubten Ziffern sind 0, 1, 2, 3, 4 (beachte: $4 = 5-1$). Die Ziffer an der i-ten Stelle gibt an, wie oft die Zahl 5^i genommen wird.

Stelle 4	Stelle 3	Stelle 2	Stelle 1	Stelle 0
625er= 5^4er	125er= 5^3er	25er= 5^2er	5er= 5^1er	1er= 5^0er

  Die dargestellte Zahl ist die Summe der Produkte aus Ziffer und Wert der Stelle. ✎

1.1.3. Man beachte, dass die Null eine enorm wichtige Funktion in Stellenwertsystemen hat. Beispielsweise sind die Zahlen 2024 und 224 verschieden. Die Entstehung der Null als Zeichen für nichts ist historisch interessant.

[1] Das Wort «abacus» bezog sich damals allgemein auf jede Rechenmethode und nicht nur auf den Abakus.

5er-System (als erstes Beispiel eines Stellenwertsystems)

⚒ **Aufgabe A1** Kindergarten im Fünferland (im Fünferland wird das Fünfersystem verwendet)

(a) Schreibe alle zweiunddreissig Zahlen von Null bis Einunddreissig im Fünfersystem auf (untereinander).

(b) Schreibe die aktuelle Jahreszahl im Fünfersystem.

⚒ **Aufgabe A2** Primarschule im Fünferland

Notiere in der Tabelle links das «Kleine Eins-plus-Eins» und in der Tabelle rechts das «Kleine Ein-mal-Eins» im Fünferland.

+	0	1	2	3	4	10
0						
1						
2						
3						
4						
10						

·	0	1	2	3	4	10
0						
1						
2						
3						
4						
10						

Bemerkung: Ob man hier die mit 10 benannten Zeilen und Spalten hinschreibt oder weglässt, ist Geschmackssache. In der Primarschule lässt man in der Regel die mit 0 benannten Zeilen und Spalten weg.

⚒ **Aufgabe A3** Schriftliches Addieren funktioniert im 5er-System «genauso» wie im Zehnersystem. Man muss eigentlich nur aufpassen, dass man niemals eine der Ziffern 5, 6, 7, 8, 9 verwendet. Wer mag, kann die obige Additionstabelle «Kleines Eins-plus-Eins im 5er-System» verwenden.

Addiere schriftlich im 5er-System (die Zahlen sind im 5er-System angegeben); die beiden Summanden sind zunächst rechtsbündig untereinanderzuschreiben. Kontrolliere deine Rechnung anschliessend im Dezimalsystem.

a) (ohne Übertrag) $1423 + 2011$ b) (mit Überträgen) $1443 + 243$

⚒ **Aufgabe A4** Schriftliches Multiplizieren funktioniert im 5er-System «genauso» wie im Zehnersystem. Wer mag, kann die obige Multiplikationstabelle «Kleines Ein-mal-Eins im 5er-System» verwenden.

Multipliziere schriftlich im 5er-System (die Zahlen sind im 5er-System angegeben). Kontrolliere deine Rechnung anschliessend im Dezimalsystem.

a) $1402 \cdot 3$ b) $432 \cdot 123$

Notation 1.1.4. Wenn man Zahlen in verschiedenen Stellenwertsystemen notiert, sollte man angeben, welches Stellenwertsystem wo verwendet wird. Dies geschieht häufig durch Angabe der *im Dezimalsystem geschriebenen* Basis des Stellenwertsystems als rechtem unterem Index. Zum Beispiel gilt

$$2010_{10} = 31020_5$$

Lässt man bei einer Ziffernfolge den Index weg, so handelt es sich in der Regel um die Darstellung einer Zahl im Zehnersystem; es kann aber auch sein, dass aus dem Kontext klar ist, in welchem Stellenwertsystem man arbeitet.

1.1.5. Wenn man eine Zahl im Zehnersystem schreibt, ist die Ziffer ganz rechts ihr Rest bei Division durch 10 und die verbleibenden Ziffern sind das Ergebnis der Ganzzahldivision. Zum Beispiel gilt

$2024 : 10 = 202$ Rest 4 oder gleichbedeutend $2024 = 202 \cdot 10 + 4$

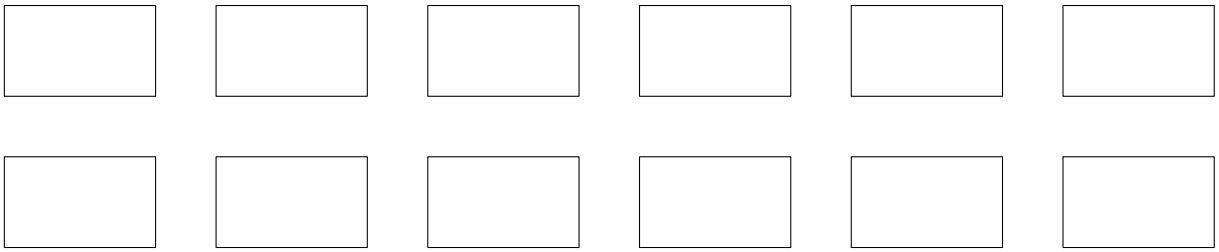
Die analoge Aussge gilt im Fünfersystem. Beispielsweise gilt (Zahlen sind im Zehnersystem notiert) ✎

Daraus folgt: Schreibt man $2024 = 2024_{10}$ im 5er-System, so ist 4 die Einerziffer.

Dies motiviert den folgenden Algorithmus: Iterierte Division mit Rest durch 5 liefert alle Ziffern im 5er-System.

Algorithmus 1.1.6 Divisionsalgorithmus: Umrechnen einer Zahl ins 5er-System (erklärt an einem Beispiel)

Schreibe die umzurechnende Zahl in die Box rechts oben (etwa die aktuelle Jahreszahl).
Dividiere diese Zahl (mit Rest) durch 5. Schreibe das Ergebnis (den sogenannten Ganzzahlquotienten) in die Box links daneben und den Rest in die Box darunter.
Dividiere dann die neue Zahl in der ersten Zeile durch 5. Schreibe das Ergebnis in die Box links daneben und den Rest in die Box darunter. Wiederhole dies, bis beim Dividieren Null als Ergebnis (nicht als Rest) herauskommt.



Als Ergebnis erhält man in der unteren Zeile die gesuchte Darstellung im 5er-System.

In unserem Beispiel gilt: ✎

Dieses aussagekräftige Beispiel beweist im Wesentlichen, dass der Algorithmus korrekt ist.
Mit «demselben» Algorithmus kann man die Darstellung jeder natürlichen Zahl in jedem Stellenwertsystem ermitteln (Division mit Rest durch b, wenn b die Basis des betrachteten Stellenwertsystems ist).

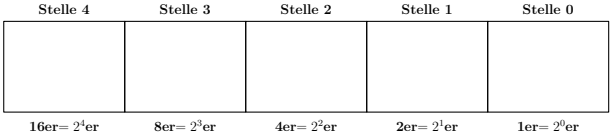
Aufgabe A5 Verwende das Divisionsverfahren (Algorithmus 1.1.6), um die folgenden Dezimalzahlen im 5er-System darzustellen.
Hinweis: Wer im Kopf durch 5 dividieren möchte: Division durch 5 ist dasselbe wie Multiplikation mit 2 und Division durch 10, denn $\frac{1}{5} = \frac{2}{10}$.

a) 194_{10} b) 678_{10} c) 2930_{10}

Binär- und Hexadezimalsystem: Die beiden wichtigsten Stellenwertsysteme in der Informatik

Erklärung 1.1.7 Stellenwertsystem zur Basis 2 = Binärsystem = Dualsystem = 2er-System

Binärsystem:
Die erlaubten Ziffern sind 0, 1. Die Ziffer an der i-ten Stelle gibt an, wie oft die Zahl 2^i genommen wird.



Schreibweise: Der besseren Lesbarkeit wegen werden längere Binärzahlen meist in Vierer-Gruppen notiert:

z. B. ✎

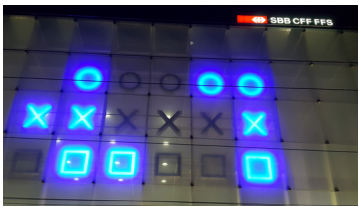
Grund dafür ist einerseits die übliche Gruppierung von 8 Bits in ein Byte, andererseits die einfache Umrechnung ins 16er-System (siehe unten).

⚒ Aufgabe A6 Umrechnen vom Binärsystem ins Zehnersystem.

Bei der St. Galler Bahnhofsuhr wird die aktuelle Uhrzeit zeilenweise (Stunden, Minuten, Sekunden) im Binärsystem angegeben. Licht an = Ziffer 1; Licht aus = Ziffer 0.

Die Form der Symbole (Kreis, Kreuz, Quadrat) dient nur der Unterscheidung von Stunden, Minuten und Sekunden.

Welche Uhrzeit wird jeweils angezeigt?

a)


b)


⚒ Aufgabe A7

(a) Kindergarten im 2erland: Zähle binär von 0 bis 33

(b) Primarschule im 2erland: Fülle die Eins-plus-Eins- und Einmal-Eins-Tabelle rechts aus. Als Zeilen- und Spaltenbeschriftungen sind die Zahlen 0 und 1 zu verwenden (die Zahl 10 lassen wir hier weg).

+		

·		

⚒ Aufgabe A8 Addiere schriftlich im Binärsystem. Die Zahlen sind untereinander zu schreiben. Kontrolliere deine Rechnung anschliessend im Dezimalsystem.

a) $10011 + 1111$ b) $11111 + 11111$

⚒ Aufgabe A9 Schriftliches Multiplizieren im Binärsystem ist relativ einfach, denn man muss jeweils entweder mit der Ziffer 1 oder der Ziffer 0 multiplizieren (am Ende muss man zugegebenermassen schriftlich addieren)). Multipliziere schriftlich und kontrolliere deine Rechnung anschliessend im Dezimalsystem.

a) $10011 \cdot 11001$ b) $11111 \cdot 11111$

⚙ Aufgabe A10 (Das Zahnradsymbol ⚙ bedeutet «Bonus-Aufgabe»)

Schriftliches Dividieren (mit Rest oder mit Nachkommastellen) funktioniert in jedem Stellenwertsystem.

- Dividiere schriftlich im Binärsystem und kontrolliere dein Resultat im Dezimalsystem:
  a) $11110000 : 10$ b) $111100 : 101$ c) (mit Rest) $111111 : 101$
- Es gibt auch binäre Kommazahlen. Die Nachkommastellen im Binärsystem stehen für $2^{-1} = \frac{1}{2}$, $2^{-2} = \frac{1}{4}$, etc. Berechne $1011 : 101 = \frac{1011}{101}$ als Kommazahl.

Erklärung 1.1.8 Stellenwertsystem zur Basis 16 = Hexadezimalsystem = 16er-System

Hexadezimalsystem: Die erlaubten Ziffern sind 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Dabei stehen

«A» für zehn, «B» für elf, «C» für zwölf, «D» für dreizehn, «E» für vierzehn, «F» für fünfzehn.

Die Ziffer an der i-ten Stelle gibt an, wie oft die Zahl 16^i genommen wird.

Stelle 4	Stelle 3	Stelle 2	Stelle 1	Stelle 0
65536er= 16^4**er**	**4096er**= 16^3**er**	**256er**= 16^2**er**	**16er**= 16^1**er**	**1er**= 16^0**er**

✎

1.1.9. Der Leser überlege sich, warum man A schreibt statt 10, B statt 11 etc.

Aufgabe A11 Kindergarten im 16erland: Zähle hexadezimal von 0 bis 33.

Aufgabe A12 Stelle die folgenden (im Hexadezimalsystem angegebenen Zahlen) im Dezimalsystem dar.

a) FF_{16} b) AFFE_{16} c) $\mathrm{CAFE07}_{16}$

1.1.10. Das Divisionsverfahren (Algorithmus 1.1.6) funktioniert sinngemäss für jedes Stellenwertsystem. Beispielsweise muss man beim Umrechnen ins Binärsystem Division durch 2 und beim Umrechnen ins Hexadezimalsystem Division durch 16 verwenden.

Aufgabe A13 Wandle die folgenden Dezimalzahlen mit Hilfe des Divisionsverfahrens (Algorithmus 1.1.6) sowohl ins Binär- als auch ins Hexadezimalsystem um.

a) 2024_{10} b) 3386_{10}

Algorithmus 1.1.11 Umwandlung zwischen Binär- und Hexadezimalsystem und umgekehrt

Will man eine Binärzahl in eine Hexadezimalzahl umwandeln, so schreibt man ihre Ziffern in die Quadrate in der oberen Zeile der folgenden «Tabelle» (so, dass die Ziffer ganz rechts im Quadrat ganz rechts steht). Nun wandelt man jeden «Vierer-Block» von Binärziffern in die zugehörige (einstellige) Hexadezimalzahl um und schreibt diese in das Rechteck darunter. Dann steht in der unteren Zeile die gesuchte Hexadezimalzahl.

In unserem Beispiel gilt also

Die Umwandlung vom Hexadezimalsystem ins Binärsystem geht «umgekehrt»: Aus jeder Hexadezimalziffer wird ein Block von vier Binärziffern.
Dieses Verfahren funktioniert wegen $2^4 = 16$. Die folgende Rechnung erklärt dies in einem Beispiel:

$$\begin{aligned}
\mathrm{AC7}_{16} &= 10 \cdot 16^2 + 12 \cdot 16 + 7 \cdot 1 \\
&= (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2 + 0) \cdot (2^4)^2 + (1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2 + 0 \cdot 1) \cdot 2^4 + (0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 1) \cdot 1 \\
&= 1010'1100'0111_2
\end{aligned}$$

Aufgabe A14 Verwende den Algorithmus 1.1.11, um die folgenden Zahlen vom Binärsystem ins Hexadezimalsystem bzw. vom Hexadezimalsystem ins Binärsystem umzuwandenln.

a) $1111\ 0000\ 0101_2$ b) $11\ 0011\ 1101_2$

c) FF_{16} d) $\mathrm{CAFE07}_{16}$

1.1.12. Computer rechnen mit Binärzahlen. Ein Nachteil von Binärzahlen ist, dass relativ kleine Zahlen bereits relativ viele Stellen benötigen; zum Beispiel hat die Dezimalzahl 2048 binär bereits zwölf Stellen,

$$2048_{10} = 1000'0000'0000_2$$

Hexadezimalzahlen benötigen hingegen nur etwa ein Viertel der Stellen, z. B. gilt

$$1000'0000'0000_2 = 800_{16}$$

Da die Umrechnung zwischen Binär- und Hexadezimalzahlen sehr einfach ist (und deutlich einfacher als die Umrechnung ins Dezimalsystem), werden Hexadezimalzahlen im informatischen Kontext oft verwendet (etwa bei der Kodierung von Farben, siehe später).

Bits und Bytes

Definition 1.1.13 Bit, Byte

- **Bit** = **b**inary digi**t** = Binärziffer, also 0 oder 1.
- **Byte** = Folge von 8 Bit = 8-stellige Binärzahl, z. B. 0100 1101

Im Kontext von Speichermedien, etwa Festplatten, meint man mit einem Byte meist die Möglichkeit, ein Byte zu speichern.

Aufgabe A15

(a) Wie viele verschieden Werte kann man in einem Byte speichern? Welchen Dezimalzahlen entsprechen diese achtstelligen Binärzahlen? Welchen Hexadezimalzahlen entsprechen diese achtstelligen Binärzahlen?

(b) Bei einer Binärzahl:

- Wie kann man sofort erkennen, ob sie gerade bzw. ungerade ist?
- Woran kann man sofort erkennen, ob sie durch 2 bzw. durch 4 bzw. durch 8 teilbar ist?
- Wie verdoppelt bzw. vervierfacht man eine Binärzahl?

(c) Kann man mit Hilfe der Quersumme ein Aussage über die Teilbarkeit einer Binärzahl treffen? (So wie die Quersumme einer Dezimalzahl genau dann durch 3 teilbar ist, wenn die Dezimalzahl durch 3 teilbar ist.)

Stellenwertsysteme in Python

Beispiel 1.1.14. Das folgende Programm wandelt eine beliebig vorgegebene natürliche Zahl `zahl` zwischen 0 und 4095 mit Hilfe des Divisionsalgorithmus 1.1.6 in eine Hexadezimalzahl um.

```
# beliebige (Dezimal-)Zahl zwischen 1 und 4095:
zahl = 2026

ziffernsymbole = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
x = zahl
ergebnis = ""

ziffer = x % 16
x = x // 16
ergebnis = ziffernsymbole[ziffer] + ergebnis
print(x, ziffer)

ziffer = x % 16
x = x // 16
ergebnis = ziffernsymbole[ziffer] + ergebnis
print(x, ziffer)

ziffer = x % 16
x = x // 16
ergebnis = ziffernsymbole[ziffer] + ergebnis
print(x, ziffer)

print(f'Die Dezimalzahl {zahl} ist {ergebnis} als Hexadezimalzahl.')
```

Aufgabe A16 Divisionsalgorithmus 1.1.6 (für beliebige Basis) in Python implementieren

(a) Verstehe das Programm in Beispiel 1.1.14.

(b) Schreibe eine Funktion `konvertiere16(zahl)` in Python, die eine (Dezimal-)Zahl `zahl` ins Hexadezimalsystem umwandelt und das Ergebnis als String zurückliefert.

Orientiere dich dabei an dem obigen Beispiel und verwende das folgende Programmgerüst.

```
ziffernsymbole = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def konvertiere16(zahl):
    if zahl == 0:
        return "0"
    x = zahl
    ergebnis = ""
    while # Abbruchbedingung zu ergänzen
        # Code zu ergänzen
    return ergebnis

# Tests:
print(konvertiere16(2026))    # 7EA
print(konvertiere16(4095))    # FFF
print(konvertiere16(45054))   # AFFE
```

(c) Warum mögen Vegetarier die Zahl $3'735'928'559_{10}$ im Hexadezimalsystem nicht?

(d) Ändere deine Funktion in eine Funktion `konvertiere(zahl, basis)` in Python, die eine Zahl `zahl` ins Stellenwertsystem zur Basis `basis` umwandelt und das Ergebnis als String zurückliefert. Teste deine Funktion mindestens mit den folgenden Aufrufen:

```
print(konvertiere(42, 2))      # 101010
print(konvertiere(3267, 7))    # 12345
```

Aufgabe A17

(a) Lies das untige Merke 1.1.15 und ermittle damit die Ausgabe des folgenden Programms.

```
d = 42
e = 0b1000011
f = 0b11_1111
g = 0x2A
h = 0xaffe
print(d, e, f, g, h, type(h))
```

(b) Prüfe deine Antwort am Computer.

Merke 1.1.15 Notation für Binär- und Hexadezimalzahlen in Python

In Python (und vielen anderen Programmiersprachen) können Zahlen direkt im Binär- bzw. im Hexadezimalsystem geschrieben werden (sowohl im Programmtext als auch bei der Eingabe). Die Kennzeichnung geschieht mit den Präfixen («Vorsilben»)

- `0b` für das Binärsystem und
- `0x` für das Hexadezimalsystem (das erste Zeichen ist jeweils eine Null, kein grosses O).

Als Gruppierungszeichen können Bodenstriche (= Unterstriche) verwendet werden.
Die Hexadezimalziffern A, B, C, D, E, F dürfen auch klein geschrieben werden.

Aufgabe A18

Lies Merke 1.1.16 und ermittle die Ausgabe des folgenden Programms zuerst ohne Computer, dann mit Computer.

```
a = hex(51)
b = hex(67)
c = bin(51)
```

```
d = bin(67)
print(a, b, c, d, type(d))
```

Merke 1.1.16

In Python gibt es die vordefinierten Funktionen

- `bin(n)` zum Umwandeln einer Zahl `n` in eine Binärzahl (Rückgabewert ist ein String, der mit dem Präfix `0b` beginnt);
- `hex(n)` zum Umwandeln einer Zahl `n` in eine Hexadezimalzahl (Rückgabewert ist ein String, der mit dem Präfix `0x` beginnt).

Aufgabe A19 Schreibe eine Funktion `dezimal_aus_16er(s)`, die aus einer als String `s` gegebenen Hexadezimalzahl die zugehörige (Dezimal-)Zahl berechnet oder allgemeiner eine Funktion `dezimal_aus(s, basis)`, die im Stellenwertsystem zur Basis `basis` (als String) gegebene Zahlen berechnet, d. h. als Dezimalzahl ausgibt.

1.1.17. Bonuswissen:

- Die Konvertierungsfunktion in ein Stellenwertsystem zu einer beliebigen Basis, die du in A16 geschrieben hast, gibt es in Python in der Bibliothek `numpy`:

  ```
  import numpy as np
  print(np.base_repr(42, 5)       # Ausgabe 132, Rückgabewert ist String
  ```

- Man kann auch Python beauftragen, eine Zeichenkette, die für eine Zahl in einem gewissen Stellenwertsystem steht, als Zahl aufzufassen (was du in Aufgabe A19 selbst programmiert hast).

  Beispielsweise «entschlüsselt» der Befehl `int('31100', 5)` die (als String im 5er-System gegebene) Zahl $(31100)_5 = (2025)_{10}$.

  Der Befehl `int('cafe', 16)` liefert dasselbe wie `0xcafe`.

- Zahlen im 8er-System = Oktalsystem kann man mit dem Präfix `0o` eingeben (etwa `0o71`). Die Funktion `oct(n)` wandelt eine Zahl `n` in das Oktalsystem um (etwa `oct(57)`).

1.2 Logischer Entwurf digitaler Systeme bzw. Einführung in die Digitaltechnik

1.2.1. Ziel dieses Abschnitt ist, zu verstehen, wie ein Computer zwei Binärzahlen addiert. Konkret bedeutet dies, dass wir dem Computer das schriftliche Addieren beibringen werden.

Als Grundbausteine werden wir die sogenannten «logischen Verknüpfungen» UND, ODER, NICHT verwenden, die sich relativ einfach als elektronische Schaltungen realisieren lassen.

Sobald die nötige Theorie verstanden ist, werden wir einen Binär-Addierer mit Hilfe einer geeigneten Simulations-Software für elektronische Schaltungen bauen.

Boolesche Algebra bzw. Logik: Rechnen mit Wahrheitswerten

1.2.2. In der «normalen» Algebra rechnet man mit Zahlen. In der **Booleschen Algebra** rechnet man mit den beiden Wahrheitswerten «wahr» und «falsch».

- Statt «wahr» schreibt man meist «1».
- Statt «falsch» schreibt man meist «0».

Der Name «Boolesche Algebra» geht auf den englischen Mathematiker George Boole (1815 – 1864) zurück. Statt von Wahrheitswerten spricht man auch von «booleschen Werten».

1.2.3. In der «normalen» Algebra verwendet man die Rechenzeichen $+$ für die Addition, $\cdot$ für die Multiplikation und $-$ als Vorzeichen.

In der booleschen Algebra verwendet man die Rechenzeichen $\vee$ für das **logische Oder** und $\wedge$ für das **logische Und** (Definition folgt unten). Ausserdem gibt es die **logische Verneinung**, die mit Hilfe eines «Strichs über dem zu verneinden Ausdruck» geschrieben wird.

Definition 1.2.4 Logische Verknüpfungen

Die Rechenoperationen **logisches Oder**, **logisches Und** und **logische Verneinung** sind wie folgt durch sogenannte **Wahrheitstafeln** (= **Wahrheitstabellen**) definiert.

- **logisches Oder** (Disjunktion), Rechenzeichen $\vee$:
  Sprechweise: $a \vee b$ wird als «a oder b» gelesen.
  Die zweite Zeile der Tabelle besagt beispielsweise:
  $0 \vee 1 = 1$ oder in Wahrheitswerten: «falsch oder wahr ist wahr».
  Merke: $a \vee b$ hat genau dann den Wert 1 (= wahr), wenn mindestens einer der beiden «Inputs» a und b den Wert 1 (= wahr) hat.

a	b	$a \vee b = a$ OR b
0	0	
0	1	
1	0	
1	1	

- **logisches Und** (Konjunktion), Rechenzeichen $\wedge$:
  Sprechweise: $a \wedge b$ wird als «a und b» gelesen.
  Die zweite Zeile der Tabelle besagt beispielsweise:
  $0 \wedge 1 = 0$ oder in Wahrheitswerten: «falsch und wahr ist falsch».
  Merke: $a \wedge b$ hat nur dann den Wert 1 (= wahr), wenn sowohl a als auch b den Wert 1 (= wahr) haben.

a	b	$a \wedge b = a$ AND b
0	0	
0	1	
1	0	
1	1	

- **logische Verneinung** (Negation), Rechenzeichen $\overline{\ }$ («Überstrich»):
  Sprechweise: $\overline{a}$ wird als «nicht a» gelesen.
  Merke: Die Verneinung von 1 (= wahr) ist 0 (= falsch) und umgekehrt.

a	$\overline{a} = \mathrm{NOT}(a)$
0	
1	

1.2.5. Man könnte logisches Oder und logisches Und auch durch eine Verknüpfungstafel angeben, also in derselben Art, wie das «Kleine-Ein-mal-Eins» die Verknüpfungstafel für die Multiplikation der natürlichen Zahlen von 1 bis 10 ist.

⚒ Aufgabe A20

(a) Vervollständige die folgende Wahrheitstabelle! In jede der vier Ergebnisspalten sind also die richtigen Werte einzutragen. Beispiel: Der rote Eintrag ist das Ergebnis der Rechnung $\overline{0 \vee 1} = \overline{1} = 0$.

a	b	$\overline{a \vee b}$	$\overline{a} \vee \overline{b}$	$\overline{a \wedge b}$	$\overline{a} \wedge \overline{b}$
0	0				
0	1	0			
1	0				
1	1				

(b) Welche Rechengesetze zeigt die gerade ausgefüllte Tabelle? (Diese heissen de-Morgansche Gesetze.)

✎

Beachte, dass es sich wirklich um einen mathematischen Beweis dieser Gesetze handelt, denn wir haben alle möglichen Fälle durchgespielt.

(c) Vervollständige die folgende Wahrheitstabelle!
Bemerkung: In einer Wahrheitstabelle sind links des senkrechten Doppelstrichs **alle** Belegungen der Variablen anzugeben. Meist sind diese Belegungen «aufsteigend als Binärzahlen» aufgeschreiben.

a	b	c	$a \wedge (b \vee c)$	$(a \wedge b) \vee (a \wedge c)$	$a \vee (b \wedge c)$	$(a \vee b) \wedge (a \vee c)$
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

(d) Welche Rechengesetze zeigt die gerade ausgefüllte Tabelle? ✎

(e) Überlege dir, dass die folgenden beiden Assoziativgesetze für alle Belegungen der Variablen a, b und c gelten.
Hinweis: Mögliche Lösungswege:
- Wann genau werden die jeweiligen Ausdrücke 1 (= wahr)?
- Verwende eine Wahrheitstafel.

$$a \vee (b \vee c) = (a \vee b) \vee c$$
$$a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

(f) Klammern sind wichtig: Überlege dir, dass die folgenden beiden Ausdrücke **nicht** für alle Belegungen der Variablen a, b und c das gleiche Ergebnis liefern. Hinweis: Ein «Gegenbeispiel» genügt.
- $(a \vee b) \wedge c$
- $a \vee (b \wedge c)$

Aufgabe A21 Finde für jede der «Ergebnis-Spalten» der folgenden Wahrheitstabelle einen logischen Ausdruck, der die angegebenen Werte liefert. Beispiel: Für die erste Ergebnis-Spalte ist eine Lösung bereits in rot eingetragen (auch $\overline{\overline{a} \vee b}$ wäre eine Lösung).

Ein logischer Ausdruck ist beispielsweise $\overline{a} \wedge (\overline{b \vee a})$. Die gesuchten logischen Ausdrücke sollen neben den Variablen a und b nur die drei logischen Verknüpfungen «Und», «Oder» und «Nicht» enthalten.

a	b	$a \wedge \overline{b}$			
0	0	0	0	1	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	0	0	0	0

Bemerkung: Die letzte Spalte ist die Wahrheitstabelle des **logischen Entweder-Oders** = **exklusiv-Oders** = **exclusive Or** = XOR: Sind a und b Wahrheitswerte, so ist a XOR b genau dann 1 (= wahr), wenn genau einer der beiden Inputs 1 (= wahr) ist (aber nicht beide).

Definition 1.2.6 XOR = eXclusive OR = exklusives Oder = logisches Entweder-Oder

XOR: Das logische XOR (Entweder-Oder) ist definiert durch die rechts angegeben Wahrheitstafel.

Merke: a XOR b hat genau dann den Wert 1 (= wahr), wenn genau einer der beiden «Inputs» a, b den Wert 1 (= wahr) hat.

a	b	a XOR b
0	0	
0	1	
1	0	
1	1	

1.2.7. Video zeigen, wie logisches Und (= Konjunktion), Oder (= Disjunktion) und Nicht (= Negation) elektronisch realisiert werden können: Sebastian Lague: Exploring How Computers Work, https://www.youtube.com/watch?v=QZwneRb-zqA bis 6:16.

Disjunktive Normalform (DNF)

Definition 1.2.8

Eine **Wahrheitswertefunktion** oder **Boolesche Funktion** ist eine Funktion, deren Inputs endliche viele Wahrheitswerte sind und die als Output einen Wahrheitswert liefert.

1.2.9. Jede Wahrheitstafel stellt in offensichtlicher Weise eine boolesche Funktion dar.
Umgekehrt ist eine boolesche Funktion eindeutig durch ihre Wahrheitstafel gegeben.
Jeder logische (= boolesche) Term (etwa $a \wedge (b \vee \overline{a})$ stellt eine boolesche Funktion dar.

Satz 1.2.10 Disjunktive Normalform für boolesche Funktionen

Jede boolesche Funktion kann durch einen logischen (= booleschen) Term beschrieben werden, ja sogar durch einen logischen Term in **disjunktiver Normalform** (**DNF**), d. h. durch eine «Ver-Oder-ung (= Disjunktion) von Ver-Und-ungen der Inputs bzw. deren Verneinungen».

Folgerung: Insbesondere ist jeder boolesche Term zu einem Term in disjunktiver Normalform äquivalent.

Beweis. Wir erklären das allgemeine Verfahren «Bilden der **disjunktiven Normalform**» an einem aussagekräftigen Beispiel (mit drei Inputs).

a	b	c	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Der folgende logische Ausdruck löst also unser Problem:

□

Schaltungen bauen mit Logik-Simulations-Software (etwa Logisim)

1.2.11. Installiere Logisim über den folgenden Link (vermutlich wirst du ausserdem Java installieren müssen - während der Logisim-Installation wirst du hoffentlich auf die entsprechende Java-Installations-Webseite geleitet): https://sourceforge.net/projects/circuit/

1.2.12. Demonstration in der Lektion:

- erste Erklärungen zur Bedienung von Logisim
- Bau des gerade konstruierten logischen Ausdrucks als logische Schaltung in Logisim.
- Test der Schaltung, indem man alle möglichen Variablenbelegungen eingibt.
- Wie man in Logisim die Wahrheitstafel zu einer Schaltung anzeigt.
- Eventuell bereits hier erklären, wie man eine solche Schaltung zu einem neuen Bauteil macht.

⚒ **Aufgabe A22** Betrachte die Wahrheitswertefunktion $f = f(a, b, c)$ mit drei Inputs a, b und c, die genau dann 1 liefert, wenn genau zwei der drei Inputs 1 sind.

(a) Schreibe die zugehörige Wahrheitstafel auf.

(b) Finde mit Hilfe des im Beweis von Satz 1.2.10 erklärten Verfahrens einen booleschen Ausdruck in disjunktiver Normalform für die Funktion $f = f(a, b, c)$.

(c) Baue die entsprechende Schaltung mit Logisim.

(d) Stimmt die von Logisim berechnete Wahrheitstafel?

Aufgabe A23 Baue mit Logisim das Bauteil «XOR» (= exklusiver Oder = Entweder-Oder). Es hat zwei Inputs x und y und einen Output. Der Output wird genau dann 1, wenn genau einer der Inputs 1 ist.

Empfohlenes Vorgehen:

- Erstelle zunächst die gewünschte Wahrheitstabelle (zwei Spalten für die Inputs, eine Output-Spalte).
- Finde einen geeigneten logischen Ausdruck für den Output (etwa per disjunktiver Normalform).
- Realisiere das Bauteil in Logisim und nenne es XOR.
- Teste dein Bauteil (von Hand oder per automatisch berechneter Wahrheitstafel).

Aufgabe A24 Halbaddierer: Addition zweier Bits = einstelliger Binärzahlen

Baue mit Logisim das Bauteil «HA» = «Halbaddierer»; was dieses Bauteil macht, wird sogleich erklärt; du wirst dieses Bauteil in den nachfolgenden Aufgaben benötigen.

Erklärung: Ein «Halbaddierer» hat zwei Inputs x und y und zwei Outputs s und c.

Wenn man die beiden Inputs x und y als einstellige Binärzahlen interpretiert, so sollen die beiden Output-Bits c und s nebeneinandergeschrieben die Summe $x + y$ darstellen. Beispielsweise soll bei den Inputs $x = 1$ und $y = 1$ als Output $c = 1$ und $s = 0$ herauskommen, denn binär gilt $1 + 1 = 10$. Allgemein soll also die folgende «binäre Gleichung»

$$x + y = cs \qquad \text{bzw. gleichbedeutend} \qquad \begin{array}{r} x \\ +\ y \\ \hline cs \end{array}$$

gelten, wobei cs als zweistellige Binärzahl aufzufassen ist.

Bemerkung: s steht für *sum*=Summe, c für carry (bit)=Übertrag.

Empfohlenes Vorgehen:

- Erstelle zunächst die gewünschte Wahrheitstabelle (zwei Spalten für die Inputs, zwei Spalten für die Outputs).
- Finde geeignete logische Ausdrücke für die beiden Outputs.
- Realisiere das Bauteil in Logisim.
- Teste das Bauteil.

Aufgabe A25 Volladdierer: Addition dreier Bits = einstelliger Binärzahlen

Baue mit Logisim das Bauteil «VA» = «Volladdierer». Es hat drei Eingänge/Inputs x, y, z und zwei Ausgänge s und c.

Wenn man die drei Inputs x, y, z als einstellige Binärzahlen interpretiert, so sollen die beiden Output-Bits c und s nebeneinandergeschrieben die Summe $x + y + z$ darstellen. Beispielsweise soll bei den Inputs $x = 1$ und $y = 1$ und $z = 1$ als Output $c = 1$ und $s = 1$ herauskommen, denn binär gilt $1 + 1 + 1 = 11$. Allgemein soll also gelten:

$$x + y + z = cs \qquad \text{bzw. gleichbedeutend} \qquad \begin{array}{r} x \\ +\ y \\ +\ z \\ \hline cs \end{array}$$

Empfohlenes Vorgehen:

- Erstelle zunächst die gewünschte Wahrheitstabelle (drei Input-Spalten, zwei Output-Spalten).
- Realisiere das Bauteil in Logisim.

  Im folgenden werden drei sinnvolle Lösungswege angedeutet. Du solltest mindestens zwei dieser Lösungswege verstehen und die zugehörigen Schaltungen in Logisim bauen.

  (1) Statt drei Bits auf einmal zu addieren, addiere man zuerst zwei Bits (mit einem Halbaddierer) und addiere zum Resultat das dritte Bit (mit einem Halbaddierer und einem weiteren Bauteil).

  (2) Finde einen logischen Ausdruck für jeden Output durch folgende Überlegungen:

  – s wird genau dann 1, wenn «ungeradzahlig viele» der Inputs 1 sind (dies klingt nach XOR).

  – c wird genau dann 1, wenn mindestens zwei der Inputs 1 sind.

  (3) Finde einen logischen Ausdruck für jeden Output per disjunktiver Normalform.

1.2.13. Wenn man an einen Voll-Addierer an einen der drei Eingänge konstant den Wert 0 legt, erhält man einen Halbaddierer.

⚒ **Aufgabe A26** Baue nun einen 4-Bit-Addierer mit Logisim. Ein 4-Bit-Addierer addiert zwei 4-stellige Binärzahlen und hat

- 8 Inputs x_3, x_2, x_1, x_0 und y_3, y_2, y_1, y_0
- 5 Outputs s_4, s_3, s_2, s_1, s_0

Der Zusammenhang zwischen Inputs und Outputs ist durch die folgende «binäre Gleichung» gegeben

$$x_3x_2x_1x_0 + y_3y_2y_1y_0 = s_4s_3s_2s_1s_0 \qquad \text{bzw. gleichbedeutend} \qquad \begin{array}{rr} & x_3\,x_2\,x_1\,x_0 \\ + & y_3\,y_2\,y_1\,y_0 \\ \hline & s_4\,s_3\,s_2\,s_1\,s_0 \end{array}$$

wobei der «x-Input» ebenso wie der «y-Input» als 4-stellige Binärzahl und der «s-Output» als 5-stellige Binärzahl aufzufassen sind.

Hinweis: In der schriftlichen Addition sind ein Halbaddierer und drei Volladdierer «versteckt».

Bemerkung: Wer mag, kann die zwei addierten Zahlen und ihre Summe mit Hilfe einer Hexadezimalanzeige anzeigen lassen. Diese findet man in Logisim unter dem Menüpunkt «Input/Output». Zusätzlich benötigt man dafür eine Splitter, den man unter dem Menüpunkt «Wiring» findet; beim Splitter muss man noch «Fan Out» und «Bit Width In» auf 4 setzen.

1.3 Speicherung von Daten am Computer allgemein

1.3.1. Computer können nur Nullen und Einsen abspeichern. Alle Daten (Zahlen, Texte, Bilder, Musik, Videos, Programme, Computerspiele, ...) müssen deshalb in geeignete Folgen von Nullen und Einsen umgewandelt werden.

Für die Spezialisten: Dass alle digitale Information in Einsen und Nullen vorliegt, stimmt fast uneingeschränkt, wenn man nur die Softwareseite betrachtet.
Flashspeicher arbeiten heute aber tatsächlich mit bis zu 16 Ladungszuständen, speichern also bis zu 4 Bits in einer Speicherzelle.
Gigabit Ethernet arbeitet mit 5 Spannungszuständen, wobei diese nicht einfach nur codierend sind, sondern immer auch noch geeignet wechseln müssen, um eine stabile Signalübertragung zu gewährleisten.

Merke 1.3.2

Jede Datei bzw. auf dem Computer gespeicherte Information wird durch eine Folge von Nullen und Einsen codiert, die als eine (sehr grosse) natürliche Zahl im Binärsystem aufgefasst werden kann!

1.4 Speicherung von Text

1.4.1. Jeder Text, der am Computer abgespeichert werden soll, muss in eine Folge von Nullen und Einsen (= eine Binärzahl) umgewandelt werden. Naheliegend ist die Idee, jedem Zeichen eine Zahl zuzuordnen und diese als Binärzahl abzuspeichern.

Definition 1.4.2

Eine fixierte Zuordnung von Zeichen zu gewissen anderen Objekten (meist Zahlen) nennt man eine **Kodierung** oder kurz einen **Code**.
Die Zeichen, die man dabei betrachtet, nennt man das **Alphabet**. Das Alphabet könnte beispielsweise aus allen Emojis oder aus allen Grossbuchstaben bestehen.
(Statt Zeichen zu kodieren, kann man auch andere Objekte kodieren, etwa Farben, Töne, etc.)

Beispiel 1.4.3. Ein wichtiges Beispiel einer Kodierung ist der Morse-Code; bekannt ist vermutlich die Morse-Sequenz «drei kurz, drei lang, drei kurz» für «SOS», symbolisch `... --- ...`. Beim Morsen wird jedem Buchstaben eine Folge langer und kurzer Töne zugeordnet; unterschiedliche lange Pausen dienen der Abgrenzung von Tönen, übermittelten Buchstaben und Wörtern.

Aufgabe A27

(a) Wie würden Sie die folgende Morse-Sequenz codieren, wenn Sie nur Nullen und Einsen verwenden dürfen?

```
--. .- .-. / -. .. -.-. .... - / ... --- / . .. -. ..-. .- -.-. ....
```

(b) Finden Sie heraus, was diese Sequenz bedeutet.

(c) Welche Buchstaben werden durch besonders wenige Morse-Zeichen codiert oder wohl genauer durch möglichst kurz dauernde Morse-Signale? Warum wohl?

(d) Wie kann man erreichen, dass gewisse Übertragungsfehler korrigiert werden (fehlerkorrigierender Code.)

ASCII-Code

1.4.4. Zur Speicherung von (englischsprachigen) Texten auf Computern hat sich der sogenannte ASCII-Code durchgesetzt (*American Standard Code for Information Interchange*; erste Version von 1963, aktuelle Version von 1968).

Er ordnet jedem Zeichen eine 7-stellige Binärzahl zu. Die Codierung geht aus der dargestellten vierspaltigen ASCII-Tabelle hervor. Quelle: https://github.com/stevenlinx/Four-Column-ASCII

00	01	10	11	
NUL	Spc	@	`	00000
SOH	!	A	a	00001
STX	"	B	b	00010
ETX	#	C	c	00011
EOT	$	D	d	00100
ENQ	%	E	e	00101
ACK	&	F	f	00110
BEL	'	G	g	00111
BS	(	H	h	01000
TAB	)	I	i	01001
LF	*	J	j	01010
VT	+	K	k	01011
FF	,	L	L	01100
CR	-	M	m	01101
SO	.	N	n	01110
SI	/	O	o	01111
DLE	0	P	p	10000
DC1	1	Q	q	10001
DC2	2	R	r	10010
DC3	3	S	s	10011
DC4	4	T	t	10100
NAK	5	U	u	10101
SYN	6	V	v	10110
ETB	7	W	w	10111
CAN	8	X	x	11000
EM	9	Y	y	11001
SUB	:	Z	z	11010
ESC	;	[	{	11011
FS	<	\	|	11100
GS	=	]	}	11101
RS	>	^	~	11110
US	?	_	DEL	11111

Vom Zeichen zum ASCII-Code: Das Zeichen F findet sich in der Spalte 10 und der Zeile 00110; hintereinandergeschrieben ergibt sich der zugehörige ASCII-Code 10 00110 = $(1000110)_2 = (46)_{16} = (70)_{10}$.
Der ASCII-Code eines Zeichens ist eine Zahl, die man in jedem beliebigen Stellenwertsystem angeben kann.

Vom ASCII-Code zum Zeichen: Gegeben ist der ASCII-Code $(43)_{10} = (2B)_{16}$ (als Dezimal- bzw. Hexadezimalzahl). Man wandle diese Dezimalzahl in eine **siebenstellige** Binärzahl (mit führenden Nullen) um: $(43)_{10} = (2B)_{16} = (0101011)_2$. Ihre vorderen zwei Ziffern 01 liefern die Spalte und ihre hinteren fünf Ziffern 01011 ihre Zeile in der Tabelle rechts: Das zugehörige Zeichen ist das Pluszeichen.

Steuerzeichen: Die Zeichen in der Spalte 00 sind sogenannte Steuerzeichen (control characters); beispielsweise steht LF für *line feed* (Zeilenvorschub; neue Zeile); CR steht für *carriage return* (Wagenrücklauf, etwa zum Steuern von Druckern). Auch DEL (ganz rechts unten) ist ein Steuerzeichen, es steht für *delete*.

Druckbare Zeichen: Die anderen 128 - 32 - 1 = 95 Zeichen sind druckbare Zeichen (Spc steht für *space* (Leerzeichen)).

ASCII ist (ursprünglich) eine 7-Bit-Zeichenkodierung: Der ASCII-Code ist eine 7-Bit-Zeichenkodierung, die $2^7 = 128$ Zeichen, die sogenannten ASCII-Zeichen, durch 7-stellige-Binärzahlen kodiert. Da Byte (= 8 Bit) heutzutage die übliche Speichereinheit ist, ergänzt man diese 7-stelligen Binärzahlen durch eine führende Null zu 8-stelligen Binärzahlen, so dass man in jedem Byte ein ASCII-Zeichen speichern kann.

Vielleicht wurde bemerkt, dass im klassischen 7-Bit-ASCII einige Zeichen fehlen (etwa die Umlaute ä, ö, ü, das Euro-Zeichen). Einige dieser Zeichen wurden später in gewissen 8-Bit-Erweiterungen von ASCII aufgenommen.

Aufgabe A28

(a) Welche Zeichenfolge (die im Wesentlichen so auf der Festplatt eines Computers stehen könnte) codiert die folgende Folge von (8-stelligen) Binärzahlen (= Bytes)?
Hinweis: Die erste Ziffer jeder Binärzahl ist Null und kann ignoriert werden. Die verbleibende 7-stellige Binärzahl ist mit der ASCII-Tabelle zu dekodieren.
0100'0001 0110'1100 0110'1100 0110'0101 0111'0011 0010'0000 0110'1001 0111'0011 0111'0100
0010'0000 0100'0010 0110'1001 0110'1110 0110'0001 0110'0101 0111'0010 0111'1010 0110'0001
0110'1000 0110'1100 0010'0001

(b) Welche Zeichenfolge codiert die folgende Folge von (zweistelligen) Hexadezimalzahlen (= Bytes)?
47 75 74 20 67 65 6D 61 63 68 74 21
Hinweis: Jede zweistellige Hexadezimalzahl ist in eine 8-stellige Binärzahl umzuwandeln, welche dann per ASCII-Tabelle für ein Zeichen steht.

(c) (freiwillig) Wer mag, kann etwa in Visual Studio Code einen Hex-Editor (= Hexadezimal-Editor) als Erweiterung/Extension installieren und sich eine beliebige Datei oder die gerade entschlüsselten Texte damit anschauen (Rechtsklick auf Datei, «Open/Reopen Editor With ...»).

	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded Text
00000000	57 69 65 20 73 70 65 69 63 68 65 72 74 20 65 69	W i e s p e i c h e r t e i
00000010	6E 20 43 6F 6D 70 75 74 65 72 20 64 69 65 73 65	n C o m p u t e r d i e s e
00000020	6E 20 54 65 78 74 20 61 6C 73 20 46 6F 6C 67 65	n T e x t a l s F o l g e
00000030	20 76 6F 6E 20 48 65 78 61 64 65 7A 69 6D 61 6C	v o n H e x a d e z i m a l
00000040	2D 5A 61 68 6C 65 6E 20 3D 20 42 79 74 65 73 3F	- Z a h l e n = B y t e s ?

Abbildung 1: Text-Datei im Hex-Editor; die Hexadezimalzahlen links entsprechen genau den Zeichen rechts (mittels ASCII). Die Tabelle links ist ein «Blick auf die Festplatte» des Computers, wenn man die Hexadezimalzahlen noch in Binärzahlen umrechnet. Sie zeigt, wie Textdateien per ASCII auf dem Computer als Folge von Nullen und Einsen gespeichert werden.

Unicode

1.4.5. Die 95 ASCII-Zeichen reichen nicht aus, wenn man etwa in anderen Schriften (chinesisch, kyrillisch, arabisch, griechisch, hebräisch, aramäisch etc.) schreiben oder andere Zeichen (etwa Emojis) oder gewisse mathematische oder musikalische Notation verwenden möchte.

Deswegen wurde eine neue Zeichencodierung kreiert, der sogenannte Unicode, der den ASCII-Code erweitert.

Er wird fortlaufend um neue Zeichen ergänzt. Die erste Unicode-Version aus dem Jahr 1991 kodierte 7'161 Zeichen, heutzutage umfasst Unicode fast 150'000 Zeichen (Version 15.1 vom September 2023).

Unicode ist wie ASCII eine Zuordnung zwischen gewissen Zeichen und Zahlen. Diese Zahlen schreibt man meist hexadezimal mit `U+` als Präfix und nennt so etwas *Unicode code point*.

Zum Beispiel ist dem Eurozeichen € der Unicode code point U+20AC zugeordnet

$$\text{Zeichen: €} \quad \xleftrightarrow{\text{Unicode}} \quad \text{Unicode code point: U+20AC}$$

Im folgenden Screenshot sind einige Unicode-Zeichen abgebildet, startend mit dem Unicode code point U+1F63C.

Bisher sind die $1'114'112 = 17 \cdot 16^4$ Codepunkte von U+0000 bis U+10FFFF „erlaubt", von denen aber, wie oben erwähnt, bisher «nur» ca. 150'000 belegt sind.

Der Unicode code point wird im Computer meist platzsparend per UTF-8 (Unicode transformation format) abgespeichert. Für ASCII-Zeichen benötigt diese Umwandlung nur ein Byte, für kompliziertere Zeichen 2 bis 5 Bytes. Diese platzsparende Speicherung der Unicode code points ist de facto Standard auf dem Web und den meisten Systemen (abgesehen von Windows). Wer sich dafür interessiert, wie diese Umwandlung genau geht, lese etwa https://de.wikipedia.org/wiki/UTF-8#Kodierung.

⚒ Aufgabe A29

(a) Für welche Zeichen stehen die folgenden Unicode code points?

U+2764 U+0061 U+1f99c U+1fbf9 U+1f939 U+1f3bc U+4e23

Hinweise:

- Internet-Suche oder:
- Unter Windows kann man teilweise Unicode-Zeichen direkt eingeben. Bei mir hat Folgendes in Word und in Outlook funktioniert: Gib einen Unicode code point, etwa `U+1f99c`, ein und direkt danach `Alt+c`. Man kann auch U+ weglassen und nur `1f99c` eingeben und danach `Alt+c` drücken.

  Unter Linux klappt oft `Ctrl+Shift+u` gefolgt von dem Unicode code point.
- «Emoji Picker» verwenden, den man vermutlich mit `Windows+.` (also Windows-Taste gefolgt von einem Punkt) öffnen kann.

(b) Achtung: Manche Unicode-Zeichen sehen sich sehr ähnlich!

Suche im Internet, für welches Zeichen der code point `U+0430` steht! Kannst du es vom Zeichen zu `U+0061` unterscheiden?

Möglicherweise können Betrüger so etwas ausnutzen, um Benutzer auf Fake-Webseiten zu locken. Dies ist ein Grund, weshalb man etwa beim Online-Banking die Bankadresse per Tastatur eingeben sollte.

Speicherplatzbedarf

1.4.6 (Erinnerung (Merke 1.1.13): Bits and Bytes)**.**

- **Bit** = **b**inary digi**t** = Binärziffer, also 0 oder 1.
- **Byte** = Folge von 8 Bit = 8-stellige Binärzahl, z. B. 0100 1101 bzw. im Kontext von Speichermedien (etwa Festplatten) die Möglichkeit, eine solche Zahl zu speichern.

Weil an jeder der 8 Positionen zwei mögliche Ziffern stehen können, kann ein Byte $2^8 = 256$ verschiedene Werte annehmen, nämlich alle Binärzahlen von `0b0000'0000` bis `0b1111'1111`, also alle Hexadezimalzahlen von `0x00` bis `0xFF` bzw. alle Dezimalzahlen von 0 bis 255.

1.4.7. Vermutlich ist bekannt, dass

- 1 Kilobyte, 1 kB, für $1'000 = 10^3$ Byte steht;
- 1 Megabyte, 1 MB, für $1'000'000 = 10^6$ Byte steht;
- 1 Gigabyte, 1 GB, für $1'000'000'000 = 10^9$ Byte steht;
- 1 Terabyte, 1 TB, für $1'000'000'000'000 = 10^{12}$ Byte steht.

Es gibt neben Dezimalpräfixen auch auch «Binärpräfixe» wie «Kibi», «Mebi», «Gibi», «Tebi»:

- 1 Kibi-Byte = 1 KiB = $2^{10} = 1024$ Byte;
  («Kibi» ist wohl eine Abkürzung von «Kilo-binär»: $10^3 = 1000 \approx 1024 = 2^{10}$)
- 1 Mebi-Byte = 1 MiB = $2^{20} = 1'048'576$ Byte;
  (analog ist «Mebi» wohl eine Abkürzung von «Mega-binär»: $10^6 = 1'000'000 \approx 1'048'576 = 2^{20}$)
- 1 Gibi-Byte = 1 GiB = $2^{30} = 1'073'741'824$ Byte;
- 1 Tebi-Byte = 1 TiB = $2^{40} = 1'099'511'627'776$ Byte.

Aufgabe A30 Auf dem Computer sind mit kB, MB, GB, TB normalerweise die Einheiten mit Binärpräfixen gemeint, wenn man etwa die Grösse einer Datei anzeigen lässt; es sollte also dort besser KiB, MiB, GiB und TiB heissen.

(a) Speichermedienhersteller verwenden aber fast immer Dezimalpräfixe. Warum machen sie das wohl?

(b) Du kaufst eine externe Harddisk mit 5 TB Kapazität und schliesst sie zu Hause an deinen Computer an. Welche Kapazität in TiB wird das System in etwa anzeigen? (Das System schreibt vermutlich TB, auch wenn es TiB anzeigt.) Warum ist die angezeigte Kapazität vermutlich etwas kleiner als die direkt von TB nach TiB umgerechnete Kapazität?

Aufgabe A31

(a) Auf eine Seite DIN-A4-Papier passen ca. 2000 Text-Zeichen (in normaler Grösse). Wenn man jedes Zeichen per ASCII durch ein Byte codiert, wieviel Speicherplatz benötigt man, um den Inhalt von 500 Seiten Text abzuspeichern?

(b) Wie viele solche 500-seitigen Bücher (die nur aus Text bestehen) kann man auf einer handelsüblichen 400 Gigabyte-Festplatte abspeichern?

(c) Die grösste Biblothek der Welt ist die British Library mit ca. 14 Millionen Büchern (unter etwa 200 Millionen Medieneinheiten, laut englischer Wikipedia vom 17.01.2022). Wenn man vereinfachend annimmt, dass ein Buch durchschnittlich 500 Seiten hat und nur aus Text besteht, wie viele handelsübliche Laptops mit 400 Gigbyte-Festplatten benötigt man in etwa, um diese 14 Millionen Bücher abzuspeichern?

1.5 Caesar-Verschlüsselung

1.5.1. Die Funktion `ord()` in Python nimmt als Argument ein einzelnes Zeichen (= einen String der Länge eins) und liefert den ASCII-Code bzw. allgemeiner den Unicode code point dieses Zeichens.

Beispielsweise liefert der Funktionsaufruf `ord('a')` die Zahl 97 (= der ASCII-Code von 'a').

Umgekehrt nimmt die Python-Funktion `chr()` eine Zahl als Argument entgegen und liefert das zugehörige Zeichen (unter der ASCII- oder allgemeiner Unicode-Codierung).

Beispielsweise liefert der Funktionsaufruf `chr(97)` das Zeichen `'a'`.

Aufgabe A32 (Caesar-Verschlüsselung) (Vermutlich sind noch einige Erklärungen zu Strings zu geben, bevor diese Aufgabe gelöst werden kann.)

Die sogenannte Caesar-Verschlüsselung eines Textes funktioniert wie folgt:

- Wähle eine natürliche Zahl n als Verschiebungsparameter (meist liegt n zwischen 0 und 25).
- Dann wird jeder Buchstabe des Alphabets durch den Buchstaben ersetzt, der n Positionen weiter hinten im Alphabet steht. Wenn man bei 'z' ankommt, fängt man wieder vorne bei 'a' an.
  (So etwas nennt man eine zyklische Vertauschung: Wenn man sich das Alphabet kreisförmig (= zyklisch) aufgeschrieben vorstellt, geht man jeweils n Position weiter.)

Beispiel: Das Wort `'hallo'` wird per Verschiebung um $n = 3$ zu `'kdoor'`.

Schreibe ein Python-Programm, dass einen Text mit Hilfe der Caesar-Verschlüsselung verschlüsselt.

Am Anfang des Programms sollen zwei Variablen `text` und `n` definiert werden. Die Variable `text` speichert den zu verschlüsselnden Text als String. Die Variable `n` ist der Verschiebeparameter.

Die Ausgabe des Programms soll die Caesar-Verschlüsselung des Texts sein.

Hinweis: Verwende die oben erklärten Python-Funktionen `ord()` und `chr()`.

Bonus-Aufgabe:

Der folgende Text wurde mit der Caesar-Verschlüsselung verschlüsselt (der Einfachheit halber besteht er nur aus Kleinbuchstaben und enthält keine Umlaute oder Sonderzeichen). Entschlüssele ihn.

Hinweis: Mir fallen spontan zwei Möglichkeiten ein:

- Naiv durch Ausprobieren. Verschlüssele den Text mit allen möglichen Verschiebeparametern und schaue, welche Verschlüsselung möglichst sinnvoll aussieht. Dieses Ausprobieren kann man von Hand durchführen, man kann aber auch den Computer alle Entschlüsselungen erzeugen lassen (und zum Beispiel testen, ob das Wort 'und' vorkommt).
- Durch eine Häufigkeitsanalyse, d. h. verwende die folgende Information: Der Buchstabe 'e' kam im Originaltext am häufigsten vor. (In deutschen Texten ist 'e' mit etwa 17 % der Buchstaben der häufigste Buchstabe.) Welcher Buchstabe kommt im codierten Text am häufigsten vor? (Diese Frage kann man durch Abzählen oder durch ein Python-Programm beantworten.

```
jviyhi wglsiriv ksixxivjyroir
xsglxiv eyw ipmwmyq
amv fixvixir jiyivxvyroir
lmqqpmwgli himr limpmkxlyq
himri deyfiv fmrhir amihiv
aew hiv qshi wglaivh kixlimpx
fixxpiv aivhir jyivwxirfvyihiv
as himr werjxiv jpyikip aimpx
wimh yqwglpyrkir qmppmsrir
hmiwir oyww hiv kerdir aipx
fvyihiv yifivq wxivrirdipx
qyww imr pmifiv zexiv aslrir
```

1.6 Lösungen-produzierendes Python-Programm

1.6.1. Das folgende Programm bezieht sich vermutlich auf eine alte Version des Skripts. Deswegen stimmen sowohl die Aufgabennummern als auch die Lösungen nicht mehr ganz (einige Aufgaben wurden sicherlich leicht verändert).

Das Python-Programm sollte in der pdf-Datei als Anhang vorhanden sein, vermutlich auf der ersten Seite als anklickbarer «Pin». Dieser Pin wird eventuell nicht in allen pdf-Viewern angezeigt. Ausweg: pdf-Datei abspeichern, mit verschiedenen pdf-Viewern öffnen.

Listing 1: Python-Code, der die Lösung einiger Aufgaben ausgibt

```
from numpy import *
import datetime
import operator

def tabelle(name, verknuepfung, zeichen, basis):
    print(name)
    print('␣␣␣' + zeichen +'␣|␣', end='')
    for i in range(basis + 1):
        print(f'{base_repr(i,␣basis):>4}', end='')
    print()
    print((basis + 2) * '---')
    for j in range(basis + 1):
```

```
        print(f'{base_repr(j, basis):>4} | ', end='')
        for i in range(basis + 1):
            print(f'{base_repr(verknuepfung(i, j), basis):>4}', end='')
        print()
    print()

def schriftlich(a, b, wort, verknuepfung, basis):
    x = int(a, basis)
    y = int(b, basis)
    print(f'{base_repr(x, basis)} {wort} {base_repr(y, basis)} ist {base_repr(verknuep

def aufgabe(n):
    s = f'Aufgabe {n}'
    print()
    print(len(s) * '-')
    print(s)
    print(len(s) * '-')
    print()

aufgabe(1)

for i in range(32):
    print(base_repr(i, 5), end=', ')
print()

heute = datetime.date.today()
jahr = heute.year
print(f'dezimal {jahr} ist im Fuenfersystem (pental?) {base_repr(heute.year, 5)}')

aufgabe(2)

tabelle('Addition', operator.add, '+', basis=5)
tabelle('Multiplikation', operator.mul, '*', basis=5)

aufgabe(3)

schriftlich('1423', '2011', 'plus', operator.add, basis=5)
schriftlich('1443', '243', 'plus', operator.add, basis=5)

aufgabe(4)

schriftlich('1402', '3', 'mal', operator.mul, basis=5)
schriftlich('432', '123', 'mal', operator.mul, basis=5)

aufgabe(5)

print(base_repr(194, 5))
print(base_repr(678, 5))
print(base_repr(2930, 5))

aufgabe(6)

print(f'{0b10011}:{0b110001}:{0b11001}')
print(f'{0b10110}:{0b110111}:{0b111011}')

aufgabe(7)

tabelle('Addition', operator.add, '+', basis=2)
```

```
tabelle('Multiplikation', operator.mul, '*', basis=2)

aufgabe(8)

schriftlich('10011', '1111', 'plus', operator.add, basis=2)
schriftlich('11111', '11111', 'plus', operator.add, basis=2)

aufgabe(9)

schriftlich('10011', '11001', 'mal', operator.mul, basis=2)
schriftlich('11111', '11111', 'mal', operator.mul, basis=2)

aufgabe(10)

schriftlich('11110000', '10', 'durch', operator.floordiv, basis=2)
schriftlich('111100', '101', 'durch', operator.floordiv, basis=2)

# print('\nDivision mit Rest')
# schriftlich('111111', '101', 'durch (Ganzzahlquotient)', operator.floordiv, basis=2)
# schriftlich('111111', '101', 'modulo (Rest der Division)', operator.mod, basis=2)

a = '111111'
b = '101'
x = int(a, 2)
y = int(b, 2)
print(f'{a} durch {b} ist {base_repr(x // y, 2)} Rest {base_repr(x % y, 2)}')


print('\nErgebnis als Kommazahl')
a = '1011'
b = '101'
x = int(a, 2)
y = int(b, 2)
# Achtung, von Hand zwei Nullen nach Komma eingefuegt!
# Koennte 1/2 addieren, falls erste Nachkommastelle nicht 1, danach die erhaltene 1 zu
print(f'{a} durch {b} ist {base_repr(x // y, 2)}.00{base_repr(x % y * 2**20 // y, 2)}.

aufgabe(11)

for i in range(34):
    print(base_repr(i, 16), end=', ')
print()

aufgabe(12)

print(int('FF', 16))
# Alternative:
# print(0xFF)
# print(0xff)
print(int('AFFE', 16))
print(int('cafe07', 16))

aufgabe(13)

x = 2024
print(f'{x} ist binaer {base_repr(x, 2)} und hexadezimal {base_repr(x, 16)}')
x = 3386
print(f'{x} ist binaer {base_repr(x, 2)} und hexadezimal {base_repr(x, 16)}')
```

```
aufgabe(14)

print(f'{0b111100000101:0x}')
print(f'{0b1100111101:0x}')
print(f'{0xff:0b}')
print(f'{0xcafe07:0b}')

aufgabe(15)

def konvertiere(x, b):
    # b sollte maximal 35 sein
    s = ''
    while x > 0:
        r = x % b
        if r < 10:
            s = str(r) + s
        else:
            s = chr(ord('A') + r - 10) + s
        x = x // b
    return s

print('Die␣gesuchte␣Funktion␣"konvertiere"␣steht␣im␣Programm.␣Hier␣2024␣im␣2er,␣5er,␣1
print(konvertiere(2024, 2))
print(konvertiere(2024, 5))
print(konvertiere(2024, 10))
print(konvertiere(2024, 16))

aufgabe(16)

listeasciicodes = [0b01000001,
                   0b01101100,
                   0b01101100,
                   0b01100101,
                   0b01110011,
                   0b00100000,
                   0b01101001,
                   0b01110011,
                   0b01110100,
                   0b00100000,
                   0b01000010,
                   0b01101001,
                   0b01101110,
                   0b01100001,
                   0b01100101,
                   0b01110010,
                   0b01111010,
                   0b01100001,
                   0b01101000,
                   0b01101100,
                   0b00100001]

s = ''
for code in listeasciicodes:
    s = s + chr(code)
print(s)

listeasciicodes = [71, 117, 116, 32, 103, 101, 109, 97, 99, 104, 116, 33]
```

```
s = ''
t = ''
for code in listeasciicodes:
    s = s + chr(code)
    t = t + base_repr(code, 16) + ' \quad '
print(s)
print(t)

aufgabe(17)

print("Siehe Loesung im Skript.")

aufgabe(18)

import ttg
print(ttg.Truths(['a', 'b'], ['not(a or b)', 'not(a) or not(b)', 'not(a and b)', 'not(

print(ttg.Truths(['a', 'b', 'c'], ['a and (b or c)', '(a and b) or (a and c)', 'a or (

aufgabe(19)

import ttg
print(ttg.Truths(['a', 'b'], ['a and not(b)', 'not(a) and b', 'not(a) and not(b)', '(n


# s = ''
# for code in listeasciicodes:
#     s = s + base_repr(code, 16) + ' \quad '
# print(s)

# print(type(0x7FFFFFFF), 0x7FFFFFFF)
# print(type(0xFFFFFFFF+1), 0xFFFFFFFFFF)

# message = 'Python is fun'

# # convert string to bytes
# byte_message = bytes(message, 'utf-8')
# print(byte_message)
# byte_message = bytes(message, 'utf-16')
# print(byte_message)

# size = 5
# arr = bytes(size)
# print(arr)

# schriftlich('1011', '101', 'durch (Ziffernfolge vor Komma)', operator.floordiv, basi
# schriftlich('10110000000000000000000', '101', 'durch (Ziffernfolge (Vor- und Nachko

# print(int('1234', 5)) # 194
# print(int('10203', 5)) # 678
# print(int('43210', 5)) # 2930
# print(int('D3A', 16)) # 2930

# print(int('0x1F', 16))
# print(0xFF)
# print(0b1110)
```

1.7 Lösungen

Hinweise zu den Symbolen:

⚒ Diese Aufgaben könnten (mit kleinen Anpassungen) an einer Prüfung vorkommen. Für die Prüfungsvorbereitung gilt: "If you want to nail it, you'll need it".

✸ Diese Aufgaben sind wichtig, um das Verständnis des Prüfungsstoffs zu vertiefen. Die Aufgaben sind in der Form aber eher nicht geeignet für eine Prüfung (zu grosser Umfang, nötige «Tricks», zu offene Aufgabenstellung, etc.). **Teile solcher Aufgaben können aber durchaus in einer Prüfung vorkommen!**.

Diese Aufgaben sind dazu da, über den Tellerrand hinaus zu schauen und/oder die Theorie in einen grösseren Kontext zu stellen.

⚒ Lösung zu A1 ex-kindergarten

(a) 0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30, 31, 32, 33, 34, 40, 41, 42, 43, 44, 100, 101, 102, 103, 104, 110, 111

(b) $2025_{10} = 31100_5$

⚒ Lösung zu A2 ex-primarschule-5erland

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	10
2	2	3	4	10	11
3	3	4	10	11	12
4	4	10	11	12	13

·					
	0	0	0	0	0
	0	1	2	3	4
	0	2	4	11	13
	0	3	11	14	22
	0	4	13	22	31

⚒ Lösung zu A3 ex-schriftlich-addieren-5er

a)

	5er	*10er*
	1423	238
+	2011	256
	3434	494

b)

	5er	*10er*
	1443	248
+	243	73
	2241	321

⚒ Lösung zu A4 ex-schriftlich-multiplizieren-5er

⚒ Lösung zu A5 ex-umrechnen

⚒ Lösung zu A6 ex-bahnhofsuhr-ablesen

⚒ Lösung zu A7 ex-kindergarten-2

⚒ Lösung zu A8 ex-schriftlich-addieren

⚒ Lösung zu A9 ex-schriftlich-multiplizieren

Lösung zu A10 ex-schriftlich-dividieren

Lösung zu A11 ex-kindergarten-16

Lösung zu A12 ex-umrechnen-hexadezimal-dezimal

Lösung zu A13 ex-umrechnen-nach-binaer-und-hexadezimal

Lösung zu A14 ex-binaer-hexadezimal

a) $F05_{16}$
b) $33D_{16}$
c) $1111\ 1111_2$
d) $1100\ 1010\ 1111\ 1110\ 0000\ 0111_2$

Lösung zu A15 ex-byte-zahlentheorie

(a) $2^8 = 256$ Werte
Diese Werte sind alle Binärzahlen von 0 = 0000'0000 bis 1111'1111, also alle Dezimalzahlen von 0 bis 255, also alle Hexadezimalzahlen von 0 = 00 bis FF.

(b)
- gerade = letzte Ziffer 0; ungerade = letzte Ziffer 1;
- teilbar durch 2: letzte Ziffer 0; teilbar durch 4: letzte beiden Ziffern 0; teilbar durch 8: letzte drei Ziffern 0.
- Verdoppeln: eine Null rechts dranhängen; Vervierfache = zweimal Verdoppeln: zwei Nullen rechts dranhängen.

(c) Vermutlich geht das nicht (ausser bei Teilbarkeit durch 1, je nachdem, ob die Quersumme 0 ist oder positiv): Die Quersumme ändert sich nicht, wenn man Nullen einfügt oder beseitigt. (Einen genauen Beweis müsste ich mir noch überlegen.)

Lösung zu A16 ex-umrechnen-python

(a) selbst
(b)
```
ziffernsymbole = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def konvertiere16(zahl):
    if zahl == 0:
        return "0"
    x = zahl
    ergebnis = ""
    while x > 0:
        ziffer = x % 16
        x = x // 16
        ergebnis = ziffernsymbole[ziffer] + ergebnis
    return ergebnis

# Tests:
print(konvertiere16(2026))   # 7EA
print(konvertiere16(4095))   # FFF
print(konvertiere16(45054))  # AFFE
```
(c) DEADBEAF (vgl. Lösung der nächsten Teilaufgabe)

(d)
```
ziffernsymbole = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def konvertiere(zahl, basis):
    if zahl == 0:
        return "0"
    x = zahl
    ergebnis = ""
    while x > 0:
        ziffer = x % basis
        x = x // basis
        ergebnis = ziffernsymbole[ziffer] + ergebnis
    return ergebnis

# Tests:
print(konvertiere(42, 2))            # 101010
print(konvertiere(3267, 7))          # 12345
print(konvertiere(3735928559, 16))   # DEADBEEF
```

Lösung zu A17 ex-notation-fuer-binaer-und-hexadezimalzahlen-in-python

```
42 67 63 42 45054 <class 'int'>
```

Lösung zu A18 ex-zahlkonvertierung-python

```
0x33 0x43 0b110011 0b1000011 <class 'str'>
```

Lösung zu A19 ex-ziffernfolge-in-stellenwertsystem-interpretieren

```
ziffernsymbole = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def dezimal_aus_16er(s):
    ergebnis = 0
    position = len(s)-1
    exponent = 0
    while position >= 0:
        zeichen = s[position]
        wert = ziffernsymbole.find(zeichen)
        ergebnis = ergebnis + wert*16**exponent
        position = position-1
        exponent = exponent+1
    return ergebnis

def dezimal_aus(s, basis):
    ergebnis = 0
    position = len(s)-1
    exponent = 0
    while position >= 0:
        zeichen = s[position]
        wert = ziffernsymbole.find(zeichen)
        ergebnis = ergebnis + wert*basis**exponent
        position = position-1
        exponent = exponent+1
    return ergebnis
```

```
print(dezimal_aus_16er('1'))
print(dezimal_aus_16er('10'))
print(dezimal_aus_16er('100'))
print(dezimal_aus_16er('FF'))

print(dezimal_aus('1', 5))
print(dezimal_aus('10', 5))
print(dezimal_aus('100', 5))
print(dezimal_aus('44', 5))
print(dezimal_aus('31100', 5))
```

Lösung zu A20 ex-wahrheitstabelle

(a)

a	b	$\overline{a \vee b}$	$\overline{a} \vee \overline{b}$	$\overline{a \wedge b}$	$\overline{a} \wedge \overline{b}$
0	0	1	1	1	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	0	0	0	0

(b) Die beiden **de-Morganschen Gesetze**

$$\overline{a \vee b} = \overline{a} \wedge \overline{b}$$
$$\overline{a \wedge b} = \overline{a} \vee \overline{b}$$

(c)

a	b	c	$a \wedge (b \vee c)$	$(a \wedge b) \vee (a \wedge c)$	$a \vee (b \wedge c)$	$(a \vee b) \wedge (a \vee c)$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

(d) Die beiden **Distributivgesetze**

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

Man beachte, dass es hier im Gegensatz zur normalen Arithmetik zwei Distributivgesetze gibt (sowohl $\wedge$ verteilt «über» $\vee$ als auch andersherum). In der normalen Arithmetik gilt nur ein Assoziativgesetz $a \cdot (b + c) = a \cdot b + a \cdot c$ (Multiplikation verteilt «über» Addition), jedoch ist $a + (b \cdot c) = (a + b) \cdot (a + c)$ im Allgemeinen falsch.

(e) Die beiden Ausdrücke in der ersten Zeile werden genau dann wahr, wenn mindestens eine der drei Variablen 1 ist. Die beiden Ausdrücke in der zweiten Zeile werden genau dann wahr, wenn alle drei Variablen 1 sind.

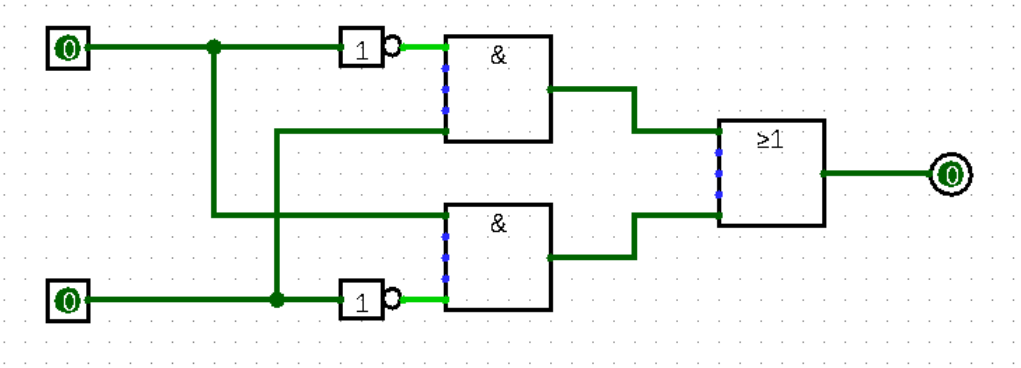
(f) Für $a = 1$, b beliebig und $c = 0$ ist der erste Ausdruck 0, der zweite aber 1.

Lösung zu A21 ex-formel-zu-wahrheitstabelle

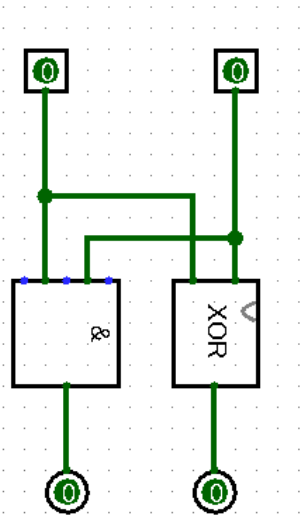
a	b	$a \wedge \overline{b}$	$\overline{a} \wedge b$	$\overline{a} \wedge \overline{b}$	$(\overline{a} \wedge b) \vee (a \wedge \overline{b})$
0	0	0	0	1	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	0	0	0	0

Lösung zu A22 ex-disjunktive-normalform

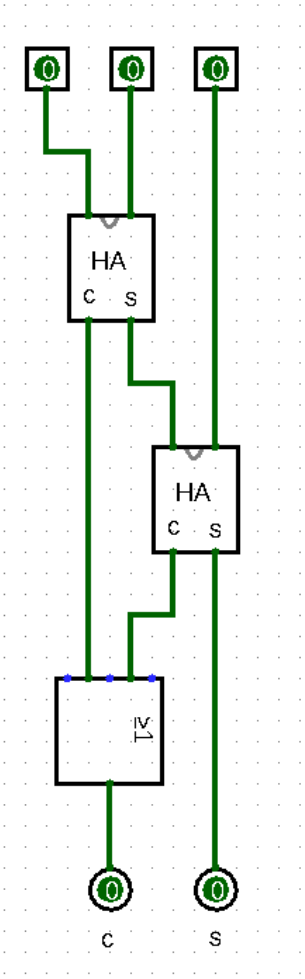
Lösung zu A23 ex-logisim-xor



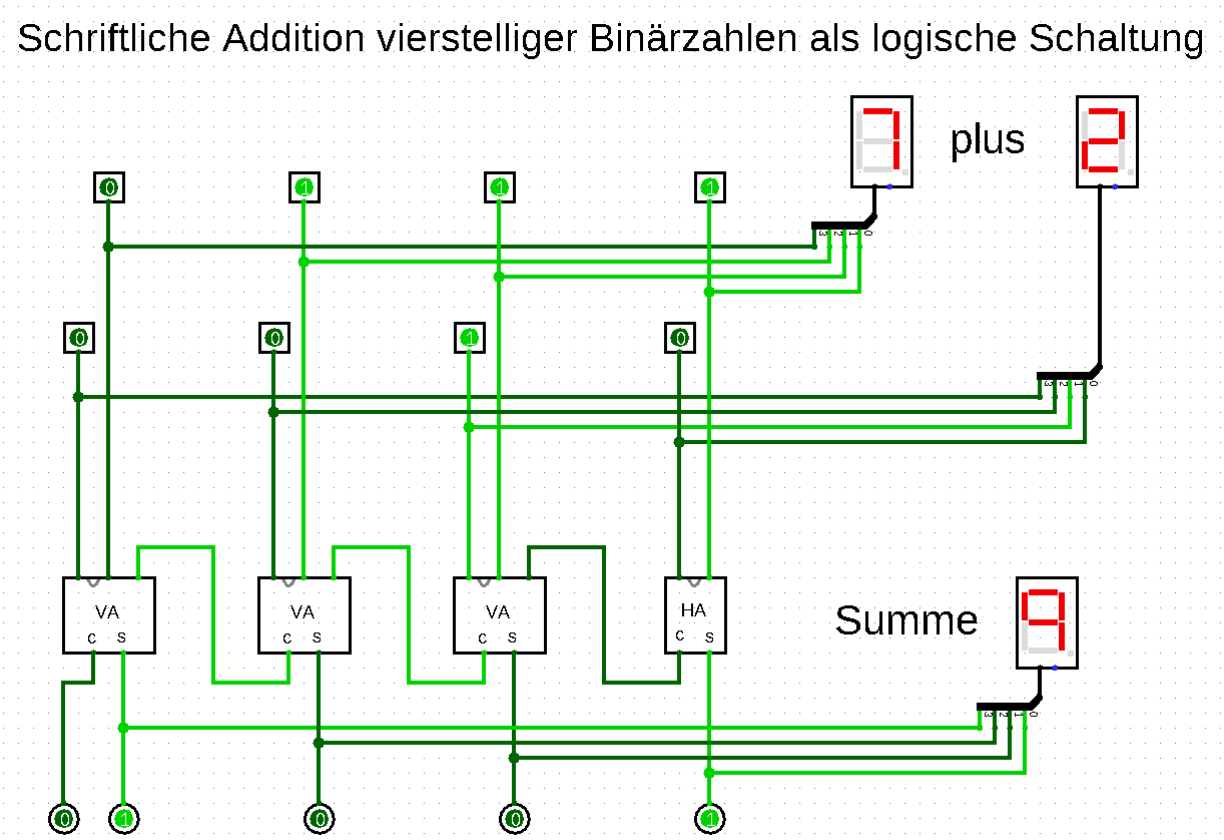
Lösung zu A24 ex-logisim-halbaddierer



Lösung zu A25 ex-logisim-volladdierer



⚒ Lösung zu A26 ex-logisim-addierer



Lösung zu A27 ex-morsecode-in-binaer

(a) Einfach Strich und Punkt zu Eins und Null (oder umgekehrt) zu übersetzen, funktioniert nicht, weil damit weder die Abstände zwischen den Buchstaben, noch jene zwischen den Wörtern codiert werden können.
Damit haben wir effektiv 4 Symbole (Punkt, Strich, Abstand, Worttrenner), die codiert werden müssen. Das kann mit 2 Bits geschehen, die zusammen 4 Zustände annehmen können, z.B. wie folgt:

- 00: Punkt
- 01: Strich
- 10: Buchstabe fertig
- 11: Wort fertig

Andere Möglichkeit:

- 10: Punkt
- 110: Strich
- 00: Buchstabe fertig
- 000: Wort fertig

Dasselbe etwas anders aufgeschrieben: Man verwende ein weiteres Symbol für die Trennung zwischen zwei Morsesignalen (wobei ein Morsesignal hier per Definition kurz oder lang ist).

- 1: Punkt
- 11: Strich
- 0: Trennung zwischen Morsesignalen
- 00: Trennung zwischen Buchstaben
- 000: Trennung zwischen Worten

(b) Die Morse-Sequenz steht für «gar nicht so einfach».

(c) Die im Deutschen (bzw. der verwendeten Sprache) am häufigsten vorkommenden Zeichen sollten möglichst wenig Übertragungszeit benötigen. (Ich hoffe, dass dies der Realität entspricht.)

(d) Wenn man fehlerkorrigierend kodieren möchte, könnte man folgendes machen:

- 11111: Punkt
- 1111111111: Strich
- 00000: Trennung zwischen Morsesignalen
- 0000000000: Trennung zwischen Buchstaben
- 000000000000000: Trennung zwischen Worten

Lösung zu A28 ex-ascii-entschluesseln

(a) Alles ist Binaerzahl!

(b) Gut gemacht!

Lösung zu A29 ex-unicode-entschluesseln

Lösung zu A30 ex-dec-vs-bin

(a) Für die gleiche Datenmenge erhält man mit dezimalen Präfixen eine grössere Masszahl als mit binären Präfixen. Das verkauft sich besser. Ein konkretes Zahlenbeispiel wird in der nächsten Teilaufgabe erläutert.

(b) Etwa 4.55 TiB. Das kann man etwa mit einem Dreisatz berechnen:

$$2^{40}\ \text{Byte} = 1\ \text{TiB}$$
$$1\ \text{Byte} = \frac{1}{2^{40}}\ \text{TiB}$$
$$1\ \text{TB} = 10^{12}\ \text{Byte} = \frac{10^{12}}{2^{40}}\ \text{TiB}$$
$$5\ \text{TB} = 5 \cdot 10^{12}\ \text{Byte} = 5 \cdot \frac{10^{12}}{2^{40}}\ \text{TiB} \approx 4.55\ \text{TiB}$$

Weil zusätzlich Verwaltungsinformation gespeichert werden muss (Dateinamen, etc.) wird die effektiv nutzbare Kapazität etwas kleiner sein.

Lösung zu A31 ex-buecher-auf-festplatte

(a) Pro Seite benötigt man 2000 Byte = 2 Kilobyte. Für 500 Seiten benötigt man also $500 \cdot 2$ Kilobyte = 1000 Kilobyte = 1 Megabyte.

(b) Wegen 400 Gigabyte = 400'000 Megabyte kann man 400'000 Bücher darauf abspeichern.

(c) Pro Laptop kann man 400'000 Bücher speichern, man benötigt also etwa $\frac{14'000'000}{400'000} = \frac{140}{4} = 35$ Laptops.

Lösung zu A32 ex-text-caesar-verschluesseln