Vorkurs für Informatikstudierende

Dr. Julien Klaus

Skript zur Vorlesung

Inhaltsverzeichnis

1 Mengenlehre

1.1 Einleitung

> Eine **Menge** ist eine beliebige Zusammenfassung von bestimmten wohlunterschiedenen Objekten unserer Anschauung oder unseres Denkens zu einem Ganzen. - *Georg Cantor (1845 – 1918)*

Mengen begegnen uns in der Mathematik ständig. Selbst in der Schule haben wir uns schon mit Mengen befasst ohne dies zu wissen. Ein Beispiel für eine Menge sind die natürlichen Zahlen $\mathbb{N}$. So ist 4 beispielsweise ein **Element** der Menge $\mathbb{N}$. Wir schreiben dafür kurz $4 \in \mathbb{N}$. Allerdings ist die Menge der natürlichen Zahlen nicht allumfassend. Die gebrochenrationalen Zahlen gehören nicht dazu. Wir können also schreiben $\frac{1}{2} \notin \mathbb{N}$.

1.2 Eigenschaften von Mengen

Zwei Mengen sind **gleich**, wenn sie die gleichen Elemente haben. Nehmen wir als Beispiel die zwei Mengen $M = \{1, 2, 3\}$ und $N = \{2, 3, 1, 2\}$. Hier sind die Elemente innerhalb der Mengen gleich. Es gilt also $M = N$. Dies bringt uns auch gleich zwei Eigenschaften von Mengen. Sie sind **nicht sortiert** und **doppelte Elemente** zählen nur einfach. Weiter werden Mengen häufig mit großen Buchstaben bezeichnet.

Die Mengen in unseren Beispielen wurden bisher nur durch Angabe ihrer Elemente beschreiben. Mengen können auch durch die Angabe von Eigenschaften angegeben werden. Die natürlichen Zahlen könnten alternativ auch als $\mathbb{N} := \{x \mid x \text{ ist eine natürliche Zahl}\} = \{1, 2, 3, \dots\}$ definiert werden. Wollen wir die null als Element der natürlichen Zahlen haben, schreiben wir $\mathbb{N}_0 = \{0, 1, 2, \dots\}$.

Um einen weiteren wichtigen Begriff in der Mengenlehre einzuführen, müssen wir uns nun überlegen wieviele Elemente die Menge $M = \{x \mid x \in \mathbb{N} \text{ und } x < 1 \text{ und } x > 1\}$ hat. Diese Menge bestitzt keine Elemente und wir bezeichnen solch eine Menge als **leere Menge**. Diese wird als $\emptyset$ gekennzeichnet.

Definition 1 (Teilmenge)**.** *Eine Menge A wird als **Teilmenge** einer Menge B bezeichnet, falls für alle Elemente $a \in A$ gilt: $a \in B$. Man schreibt in diesem Fall $A \subseteq B$. Falls B Elemente enthält, die A nicht enthält schreibt man $A \subset B$.*

Die Menge aller Teilmengen einer Menge wird als Potenzmenge bezeichnet. Ein Beispiel für Teilmengen sind die natürlichen und die reellen Zahlen. Es gilt also $\mathbb{N} \subset \mathbb{R}$. Durch die Definition der Teilmenge können wir nun folgenden Satz beweisen.

Satz 1 (Gleichheit von Mengen)**.** *Es seien A und B Mengen. Dann sind die folgenden zwei Aussagen äquivalent*

1. $A = B$,
2. $A \subseteq B$ *und* $B \subseteq A$.

Beweis. Beweis durch die Teilmengendefinition. □

Definition 2 (Kardinalität)**.** *Die Größe n einer Menge A wird als **Kardinalität** bezeichnet und durch die Schreibweise $\mid A \mid = n$ angeben.*

1.3 Operationen auf Mengen

Definition 3 (Komplement)**.** *Das **Komplement** einer Menge $A \subseteq M$ wird als $\bar{A} = \{x \in M \mid x \notin A\}$ definiert.*

Wenn wir also das Komplement einer Menge angeben möchten, brauchen wir eine Übermenge, die unsere Menge enthält. Beispielsweise ist das Komplement der Menge $\{0\} \in \mathbb{N}_0 \subseteq \mathbb{N}_0$ gleich der Menge $\mathbb{N}$.

Definition 4 (Durchschnitt und Vereinigung)**.** *Als der **Durchschnitt** $A \cap B$ zweier Mengen A und B werden alle Elemente bezeichnet, die sowohl in A als auch in B vorkommen.*

*Die **Vereinigung** zweier Mengen A und B schreiben wir als $A \cup B$. Diese enthält alle Elemente, die entweder in A oder in B oder in beiden vorhanden sind.*

Beispiel 1 (Durchschnitt und Vereinigung)**.** *Sei die Menge $A = \{1, 2, 3\}$ und die Menge $B = \{2, 3, 4\}$. Die Vereinigung dieser Mengen $A \cup B$ ist gleich $\{1, 2, 3, 4\}$ und der Durchschnitt $A \cap B$ ist gleich $\{2, 3\}$.*

Definition 5 (Differenz)**.** *In der **Differenz** $A \setminus B$ zweier Mengen A und B sind die Elemente aus A, die nicht in B vorkommen.*

Satz 2 (Komplement von B in A)**.**

$$A \setminus B \text{ ist äquivalent zu } A \cap \bar{B}.$$

Beweis. In der Differenz von $A \setminus B$ sind alle Element von A ohne die von B. Die Menge, die alle Element enthält, die nicht in B sind ist das Komplement $\bar{B}$. Der Durchschnitt von $A \cap \bar{B}$ enthält nun also alle Elemente von A ohne die aus B. □

Definition 6 (Symmetrische Differenz)**.** *Die **Symmetrische Differenz** zweier Mengen A und B ist definiert als:*

$$A \triangle B = (A \setminus B) \cup (B \setminus A).$$

Satz 3 (Eigenschaften von Mengenoperationen)**.** *Für die zweistelligen Operationen auf Mengen gelten folgende Eigenschaften. Seien hierzu A, B, C Mengen.*

- $A \cap B = B \cap A$ *(Kommutativität)*
- $(A \cap B) \cap C = A \cap (B \cap C)$ *(Assoziativität)*
- $A \subseteq B$ *und* $B \subseteq C$ *folgt* $A \subseteq C$
- $A \subseteq B$ *folgt* $A \cup B = B$
- $A \subseteq B$ *folgt* $A \cap B = A$
- $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$
- $\overline{(A \cup B)} = \bar{A} \cap \bar{B}$

Beweis. Der Beweis kann über Mengendiagramme durchgeführt werden. Dies wird in der Übung gezeigt. □

Wie wir bereits gelernt haben sind Mengen nicht sortiert. Falls wir eine Sortierung brauchen, können wir dies mit dem Konstrukt aus der nächsten Definition lösen.

Definition 7 (n-Tupel)**.** *Seien $x_1, \ldots, x_n \in M, n \in \mathbb{N}$ n Elemente einer Grundmenge M. Ein **n-Tupel** ist eine Zusammenfassung dieser Elementen, dargestellt als*

$$(x_1, \ldots, x_n),$$

wobei die Reihenfolge dieser Elemente von Bedeutung ist. An der i-ten Stelle steht dabei das Element x_i.

Bemerkung 1 (Geordnete Paar)**.** *Eine Sonderform des n-Tupel ist das **geordnete Paar**. Dies ist ein Tupel, welches nur zwei Elemente enthält*

Definition 8 (Kartesisches Produkt)**.** *Seien A, B beliebige Mengen. Die Menge aller geordneten Paare zwischen A und B wird als **kartesisches Produkt** $A \times B$ bezeichnet. Diese sist definiert als:*

$$A \times B = \{(a, b) \mid a \in A \text{ und } b \in B\}.$$

Falls $A = B$ verabschieden wir die Schreibweise A^2.

Beispiel 2. *Bei der Anwendung des Kreuzproduktes erhalten wieder eine Menge.*

$$(A \times C) \cup (B \times C) = \{(x, y) | x \in A \text{ und } y \in C \text{ oder } x \in B \text{ und } y \in C\} = (A \cup B) \times C$$

Natürlich können wir diese Definition auch beliebig auf n-Tupel erweitern. Für die nächste Schreibweise, die wir definieren möchte, benötigen wir in dem einfachen Fall erst einmal nur geordnete Paare.

Definition 9 (Abbildung)**.** *Seien $\mathcal{D}, \mathcal{W}$ Mengen. Eine **Abbildung** (Funktion) $f : \mathcal{D} \to \mathcal{W}$ ist dann die Menge aller geordneten Paare*

$$\{(x, f(x)) \mid x \in \mathcal{D}, f(x) \in \mathcal{W}\}.$$

Man bezeichnet $\mathcal{D}$ als den Definitionsbereich und $\mathcal{W}$ als den Wertebereich.

1.3.1 Kurzschreibweisen für Mengen

Einige Mengen werden in der Mathematik häufig benötigt. Wir haben bereits einige gesehen, wie $\mathbb{R}$ oder $\mathbb{N}$. In diesem kurzen Kapitel wollen wir uns noch Intervall- und Indexmengen definieren.

Definition 10 (Intervallmengen)**.** *Sei $x, y \in \mathbb{R}, x \leq y$. Dann definieren wir die folgenden Mengen*

$$\begin{aligned}(x, y) &= \{z \mid z > x \text{ und } z < y\},\\ (x, y] &= \{z \mid z > x \text{ und } z \leq y\},\\ [x, y) &= \{z \mid z \geq x \text{ und } z < y\},\\ [x, y] &= \{z \mid z \geq x \text{ und } z \leq y\}.\end{aligned}$$

Definition 11 (Indexmenge)**.** *Sei $n \in \mathbb{N}$. Dann definieren wir die folgende Menge*

$$[n] = \{i \mid i \in \{1, 2, \ldots, n\}\}.$$

2 Logik

2.1 Aussagen und Wahrheitswerte

Für Informatiker ist die Logik ist eines der wichtigsten Gebiete der Mathematik. Die Logik bassiert dabei auf Aussagen denen ein Wahrheitswert zugeordnet wird.

Definition 12 (Aussage)**.** *Eine **Aussage** ist eine Zeichenfolge, der genau einer der beiden Wahrheitswerte **wahr** (1, w) oder **falsch** (0, f) zugeordnet werden kann.*

Wenn p eine Aussage ist, dann bezeichnen wir mit $\alpha(p) \in \{w, f\}$ den Wahrheitswert von p.

Aussagen können die verschiedensten Formen annehmen. Beispiele für Aussagen sind unter anderen:

- $4 \in \mathbb{N}$,
- 3 ist eine Primzahl,
- Hello ist das englische Wort für Hallo,
- Montag ist ein Tag des Wochenendes,
- $\frac{1}{2} = 0.87$.

Wir können für diese Beispiele natürlich die Wahrheitswerte angeben. Wie für Mengen existieren auch für Aussagen Operationen.

2.2 Operationen auf Aussagen

Die einfachste Operation auf Aussagen ist die der Negation. Diese werden wir auch als einzige einstellige Operation einführen. Operationen werden durch die Anwendung einer Wahrheitswerttabelle dargestellt. Diese gibt für Belegungen einer Aussage, die Wahrheitswerte die durch die Operation entstehen an.

Definition 13 (Negation)**.** *Sei p eine Aussage. Die **Negation** $\neg p$ von p wird durch die folgende Wahrheitswerttabelle definiert.*

$\alpha(p)$	$\alpha(\neg p)$
w	f
f	w

Oftmals gibt es für die Negation von Aussagen andere Schreibweisen. Ein Beispiel hierfür könnte die Aussage $4 \in \mathbb{N}$ sein. Deren Negation kann man als $\neg(4 \in \mathbb{N})$ oder kurz als $4 \notin \mathbb{N}$ schreiben.

Folgerung 1 (Zweifache Verneinung)**.** *Es sei p eine Aussage. Dann gilt*

$$\alpha(\neg(\neg(p))) = \alpha(p)$$

Beweis. Beweis in der Übung. □

Definition 14 (Konjugation, Disjunktion, Implikation, Äquivalenz)**.** *Seien p und q Aussagen. Die wichtigsten Operationen der Logik (neben der Negation) sind die*

- ***Konjugation*** *$p \wedge q$ (und),*
- ***Disjunktion*** *$p \vee q$ (oder),*
- ***Implikation*** *$p \to q$ (wenn, dann) und*
- ***Äquivalenz*** *$p \leftrightarrow q$ (genau dann, wenn).*

Diese sind in der folgenden Wahrheitswerttabelle definiert.

$\alpha(p)$	$\alpha(q)$	$\alpha(p \wedge q)$	$\alpha(p \vee q)$	$\alpha(p \to q)$	$\alpha(p \leftrightarrow q)$
w	w	w	w	w	w
w	f	f	w	f	f
f	w	f	w	w	f
f	f	f	f	w	w

Wie man in der Defintion sieht, ist es für die logische Oder-Operation nicht ausschließlich. Umgangssprachlich ist dies oft als 'und/oder' formuliert. Das ausschließende Oder ist eine andere Verknüpfung die als $p \oplus q$ zwischen den Aussagen p und q dargestellt wird. Diese ist wie folgt definiert:

$$p \oplus q \leftrightarrow (\neg p \wedge q) \vee (p \wedge \neg q).$$

Wichtig ist hier, dass wir die beiden Aussagen mit einer Äquivalenzoperation verknüfen (die linke Aussage ist genau dann wahr, wenn die rechte Aussage wahr ist). Diese Aussage ist also immer wahr. Man bezeichnet Aussagen, die immer wahr sind als **Tautologie**. Wichtige Tautologien sind im folgenden Satz zusammengefasst.

Satz 4 (Tautologien)**.** *Seien p, q, r Aussagen. Dann sind dei folgenden Aussagen Tau-*

tologien.

$$p \leftrightarrow \neg(\neg p) \tag{2.1}$$
$$(p \wedge p) \leftrightarrow p \tag{2.2}$$
$$(p \wedge q) \leftrightarrow (q \wedge p) \tag{2.3}$$
$$((p \wedge q) \wedge r) \leftrightarrow (p \wedge (q \wedge r)) \tag{2.4}$$
$$(p \vee p) \leftrightarrow p \tag{2.5}$$
$$(p \vee q) \leftrightarrow (q \vee p) \tag{2.6}$$
$$((p \vee q) \vee r) \leftrightarrow (p \vee (q \vee r)) \tag{2.7}$$
$$((p \wedge q) \vee r) \leftrightarrow ((p \vee r) \wedge (q \vee r)) \tag{2.8}$$
$$((p \vee q) \wedge r) \leftrightarrow ((p \wedge r) \vee (q \wedge r)) \tag{2.9}$$
$$((p \vee q) \wedge q) \leftrightarrow q \tag{2.10}$$
$$((p \wedge q) \vee q) \leftrightarrow q \tag{2.11}$$
$$(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q)) \tag{2.12}$$
$$(\neg(p \vee q)) \leftrightarrow ((\neg p) \wedge (\neg q)) \tag{2.13}$$
$$(p \to q) \leftrightarrow (\neg p \vee q) \tag{2.14}$$
$$(p \to q) \leftrightarrow (\neg q \to \neg p) \tag{2.15}$$
$$(p \leftrightarrow q) \leftrightarrow ((p \to q) \wedge (q \to p)) \tag{2.16}$$
$$((p \to q) \wedge (q \to r) \wedge (r \to p)) \leftrightarrow ((p \leftrightarrow q) \wedge (q \leftrightarrow r) \wedge (r \leftrightarrow p)) \tag{2.17}$$

Beweis. Alle diese Aussagen können über eine Wahrheitswerttabelle bewiesen werden. Ein alternativer Weg, diese Aussagen zu beweisen ist über geschicktes Umformen.

Wir zeigen dies für Aussage (15), $(p \to q) \leftrightarrow (\neg q \to \neg p)$. Hierzu nehmen wir an, dass die Tautologien 1, 6 und 14 bereits bewiesen sind.

$$\begin{aligned} & (p \to q) & \\ \leftrightarrow\ & \neg p \vee q & (2.14)\\ \leftrightarrow\ & \neg p \vee \neg(\neg q) & (2.1)\\ \leftrightarrow\ & \neg(\neg q) \vee \neg p & (2.6)\\ \leftrightarrow\ & \neg q \to \neg p & (2.14) \end{aligned}$$

In der Übungen werden die weiteren und angenommenen Tautologien bewiesen. □

Bemerkung 2 (Schreibweise)**.** *Das Symbol $\to$ kann nur zwischen Aussagen stehen. Beispielsweise ist es korrent für $x \in \mathbb{N}$ zu schreiben $(x-1)^2 = 0 \to x = 1$. Allerdings wäre es falsch zu schreiben $(x-1)^2 \to x^2 - 2x + 1$. Die richtige Schreibweise ist hier $(x-1)^2 = x^2 - 2x + 1$, was eine wahre Aussage ist.*

2.3 Prädikatenlogik

Der Begriff Prädikatenlogik soll hier nicht abschreckend wirken. Wir benötigen ihn um umfassendere Aussagen zu beschreiben. In Aussagen müssen alle Variablen durch eine

Belegung gebunden sein. Beispielsweise ist $x^2 = 4$ keine Aussage, da man ihr keinen eindeutigen Wahrheitswert zuweisen kann. Erst durch die Information, dass es mindestens ein x gibt, so dass gilt $x^2 = 4$ oder durch das setzten von x auf einen festen Wert wird es zu einer Aussage.

Definition 15 (All- und Existenzquantor)**.** *Es sei I eine nicht leere Menge und für alle $i \in I$ sei p_i eine Aussage. Dann kann man diese Aussagen mithilfe des Allquantor $\forall$ oder des Existenzquantor $\exists$ verknüpfen. Dabei steht $\forall i \in I : p_i$ dafür, dass für alle $i \in I$ die Aussage p_i gilt. Die Aussage $\exists i \in I : p_i$, dass es mindestens ein $i \in I$ gibt, für das p_i gilt.*

Die Wahrheitswerte der verknüfungen ist intuitiv definiert als:

$$\alpha(\forall i \in I : p_i) = w \leftrightarrow \alpha(p_i) = w \text{ für alle } i \in I$$

und

$$\alpha(\exists i \in I : p_i) = w \leftrightarrow \alpha(p_i) = w \text{ für mindestens ein } i \in I.$$

Satz 5 (Negation von Quantoren)**.** *Die folgenden Aussagen sind Tautologien über der nicht leeren Menge I und den Aussagen $p_i, i \in I$.*

$$\neg(\forall i \in I : p_i) \leftrightarrow \exists i \in I : \neg p_i \tag{2.18}$$

$$\neg(\exists i \in I : p_i) \leftrightarrow \forall i \in I : \neg p_i \tag{2.19}$$

. Von Formel 2.18]

$$\begin{aligned}
& & \neg(\forall i \in I : p_i) \\
\leftrightarrow & & \neg \bigwedge_{i \in I} p_i \\
\leftrightarrow & & \neg(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \\
\leftrightarrow & & \neg p_1 \vee \neg p_2 \vee \ldots \neg p_n \\
\leftrightarrow & & \bigvee_{i \in I} \neg p_i \\
\leftrightarrow & & \exists i \in I \neg p_i
\end{aligned}$$

Formel 2.19 verläuft synonym. □

Bemerkung 3 (Verknüfung von Quantoren)**.** *Quantoren können beliebig verknüft werden. Ein Beispiel hierfür ist die Definition der Kommutivität der Multiplikation.*

$$\forall a \in \mathbb{R} \forall b \in \mathbb{R} : a \cdot b = b \cdot a$$

2.4 Aussagen über Mengen

Nachdem wir Mengenlehre- und Logikkenntnisse erworben haben können wir jetzt komplexere Aussagen über Mengen beweisen. Wir wollen dies hier an einem Beispiel zeigen.

Beispiel 3. *Seien $A, B, C \neq \emptyset$ nichtleere Mengen. Wir wollen zeigen, dass gilt:*

$$(A \cap B) \times C = (A \times C) \cap (B \times C).$$

Beweis. Wir müssen für alle (x, y) aus der Menge $D = (A \cap B) \times C$ zeigen, dass dies auch in $(A \times C) \cap (B \times C)$ liegen:

$$\begin{aligned}
\forall (x,y) \in D : (x,y) \in (A \cap B) \times C &\leftrightarrow \forall (x,y) \in D : x \in (A \cap B) \wedge y \in C \\
&\leftrightarrow \forall (x,y) \in D : (x \in A \wedge x \in B) \wedge y \in C \\
&\leftrightarrow \forall (x,y) \in D : x \in A \wedge y \in C \wedge x \in B \wedge y \in C \\
&\leftrightarrow \forall (x,y) \in D : (x \in A \wedge y \in C) \wedge (x \in B \wedge y \in C) \\
&\leftrightarrow \forall (x,y) \in D : (x,y) \in (A \times C) \wedge (x,y) \in (B \times C) \\
&\leftrightarrow \forall (x,y) \in D : (x,y) \in (A \times C) \cap (B \times C).
\end{aligned}$$

□

3 Einführung Informatik

> Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mithilfe von Computern. - Gesellschaft für Informatik
>
> Informatik ist die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen, besonders der automatischen Verarbeitung mithilfe von Digitalrechnern - Wikipedia

Angewandte Informatik beschäftigt sich mit konkreten Anwendungsgebieten der Informatik und ihren spezifischen Anforderungen (Beispiele: Bioinformatik, Wirtschaftsinformatik, medizinische Informatik).

Praktische Informatik behandelt Konzepte der Programmierung und der Entwicklung von Programmen und komplexen, informationsverarbeitenden Systemen (Beispiele: Softwaretechnik, Datenbanksysteme).

Technische Informatik beschäftigt sich mit Computer-Hardware, deren Entwurf und maschinennahen Aspekten (Beispiele: Rechnerarchitektur, Signalanalyse).

Theoretische Informatik biete theoretische Grundlagen für alle Teilbereiche der Informatik (Beispiele: Komplexitätstheorie, Algorithmik).

Abbildung 3.1: Grundlegender Ablauf in der Informatik

Die Informatik ist im Grunde die ganze Zeit damit beschäftigt Probleme zu lösen. Den grundlegenden Ablauf vom Problem bis zu Lösung oder Ausführung ist in Abbildung 3.1 dargestellt.

4 Algorithmik

Wir kommen nun von der Formalisierung unseres Problems zur Entwicklung und Analyse von Algorithmen.

4.1 Motivation

Algorithmen werden in vielen Bereichen des täglichen Lebens verwendet. Ob es ein Kochrezept ist, die IKEA Anleitung oder einfach die Reihenfolge in der wir uns am Morgen anziehen oder unsere Kleidung auswählen. Algorithmen sind allgegenwärtig. Selbst in diesem Vorkurs habt ihr bereits einige Algorithmen verwendet. Beispielsweise zur Umrechnung von dezimalen Zahlen in binär Zahlen.

Versuchen wir einmal einen eigenen Algorithmus für ein Problem zu entwickeln.

Wer ist der jüngste Studierenden hier im Raum? Entwickeln wir einen Algorithmus:

Problem Wir wollen den jüngsten Studierenden in der Vorlesung finden.

Formalisierung Wir könnten das Alter oder das Geburtsdatum benutzen. Weiter müssen wir uns überlegen ob wir nur die Alter der gerade anwesenden oder die Liste mit Anmeldungen benutzen wollen. Ist die Lösung eindeutig?

Algorithmenentwurf Wir durchlaufen die Liste und merken uns den kleinsten Wert. Weiter können wir auch eine Divide and Conquer Variante nehmen oder eine geeignete Datenstruktur nutzen.

4.2 Eigenschaften von Algorithmen

Jede Abfolge von Aktionen ist nicht gleich ein Algorithmus. Für Algorithmen gelten bestimmte Eigenschaften. So müssen Algorithmen Eindeutig, Ausführbar, Endlich, Determiniert, Deterministisch und Terminiert sein.

Definition 16 (Eindeutig)**.** *An jeder Stelle des Algorithmus muss* ***eindeutig*** *festgelegt sein, was als nächstes zu tun ist. Die Anweisung muss also unmissverständlich formuliert sein.*

Definition 17 (Endlichkeit)**.** *Ein Algorithmus darf nur eine* ***endliche*** *Länge besitzen. Zudem dürfen seine Daten nur endlich viel Platz belegen.*

Definition 18 (Deterministisch)**.** *Für jede Anweisung gibt es genau eine folge Anweisung. Wenn man sich aussuchen kann wie man weiter macht spricht man von* ***Nichtdeterministisch****.*

Definition 19 (Determiniertheit)**.** *Für die gleichen Eingabedaten muss der Algorithmus das gleiche Resultat liefern.*

Definition 20 (Terminierung)**.** *Ein Algorithmus muss nach einer* ***endlichen*** *Anzahl von Anweisungen beendet sein oder anhalten.*

Beispiel 4 (Sortieralgorithmus)**.** *Wir können uns die Eigenschaften von Algorithmen an einem einfachen Sortieralgorithmus verdeutlichen.*

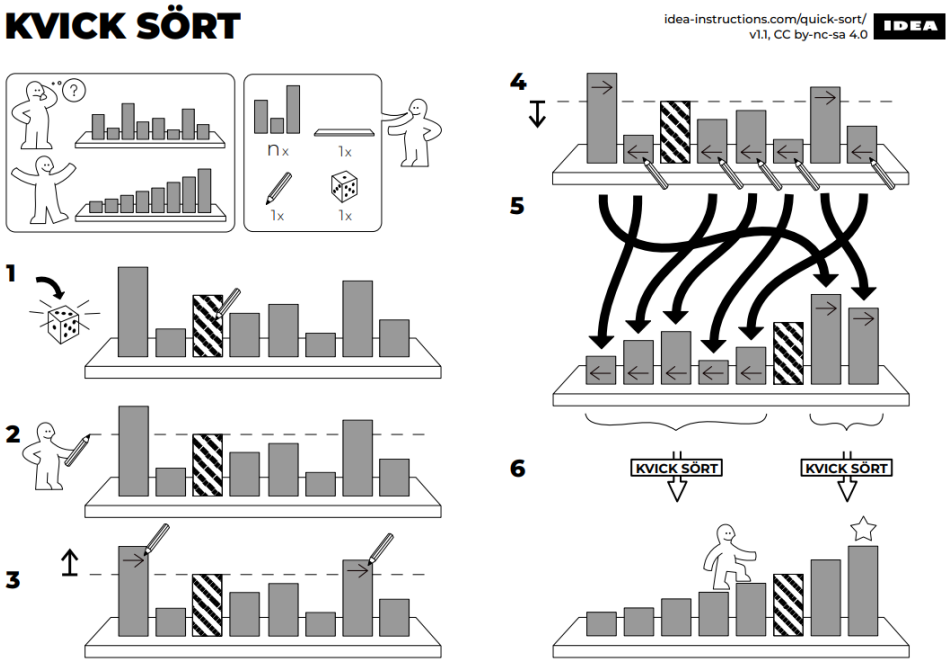


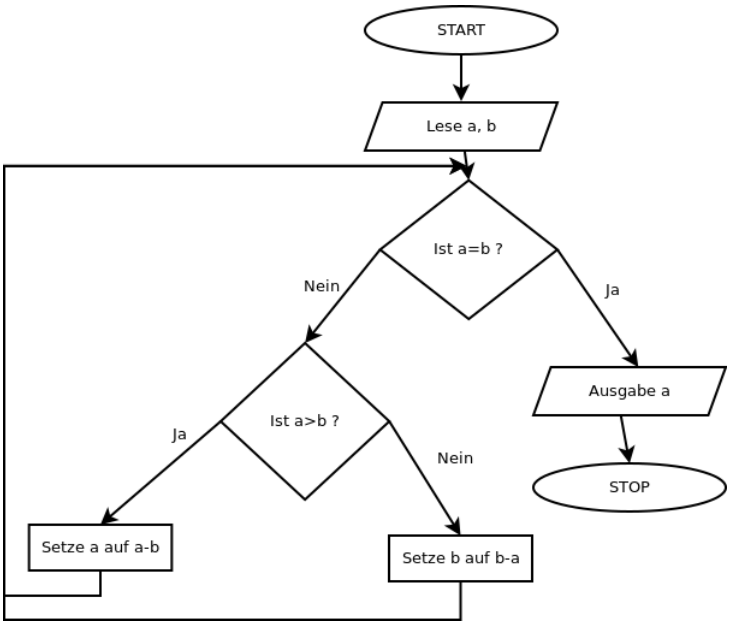
Abbildung 4.1: Erklärung des Quicksort Algorithmus mithilfe einer IKEA Anleitung.

Der dargestellte Sortieralgorithmus arbeitet, indem in jedem Schritt ein Element zur Teilung (in der Abbildung 4.1 schraffiert) ausgewürfelt wird, die Elemente bezüglich dieses Elementes sortiert und anschließend die Elemente kleiner und größer jeweis wieder mit dem Quicksort Algorithmus sortiert werden.

Die Schritte für diesen Algorithmus sind an jeder Stelle ***eindeutig*** *und durch die Nummerierung auch* ***deterministisch****. Weiter werden die Elemente in jedem Schritt geteilt und die Listen links und rechts werden kleiner (das Element zur Teilung wird nie wieder betrachtet). Dies führt dazu, das der Algorithmus nach einer* ***endlichen*** *Dauer endet (****Terminierung****) und jede Liste von Elementen am Ende sortiert ausgibt (****Determiniertheit****).*

4.3 Flussdiagramme und Pseudocode

Wenn man Algorithmen entwickelt muss man sie irgendwie darstellen. Viele tun dies mit Flussdiagrammen, noch mehr mit Pseudocode und sehr wenige mit Struktogrammen[1]




```
PROCEDURE fun (a,b)
  BEGIN
    INPUT a, b
    WHILE a is not b
      IF a > b
        THEN a <- a - b
        ELSE b <- b - a
      END IF
    END WHILE
    PRINT a
  END
END PROCEDURE
```

Pseudocode verwendet dafür bereits definierte Schlüsselworte. Vermutlich werden Sie recht schnell eine eigene Art entwickeln das Verhalten von Programmen zu beschreiben.

4.3.1 Flussdiagramme

Flussdiagramme sind eine grafische Darstellung von Algorithmen. Sie bestehen im Grunde aus Pfeilen und Symbolen. Wie diese Pfeile und Symbole auszusehen haben und welche Funktion sie haben sind in der DIN 66001 genormt.

4.3.2 Start und Stop

Um den Start und das Ende von Ablaufplänen zu zeigen gibt es spezielle Symbole, welche START und STOP beinhalten.

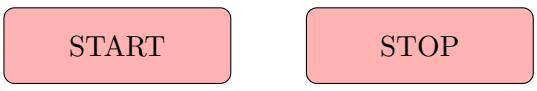


Abbildung 4.3: Beginn und Ende des Flussdiagrammes.

[1] Vergessen Sie gleich wieder.

4.3.3 Anweisungen

Für einfache Anweisungen wie $a = b + c$ verwendet man Rechtecke.

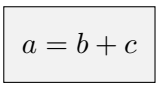


Abbildung 4.4: Eine einfache Anweisung.

4.3.4 Verzweigungen

Verzweigungen werden genutzt um zwischen Alternativen zu unterscheiden. Beispielsweise kann man das Maximum zweier Zahlen a, b bestimmen indem man fragt ob $a < b$ ist. Die Ausgehenden Pfeile werden meist links und rechts angebracht und mit JA und NEIN oder True und False oder $+$ und $-$ bezeichnet.

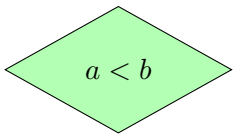


Abbildung 4.5: Eine Verzweigung.

4.3.5 Ein- und Ausgabe

Natürlich möchte man in seinem Algorithmus auch Daten eingeben oder Dinge ausgeben. Hierzu werden Ein- und Ausgabeblöcke verwendet.

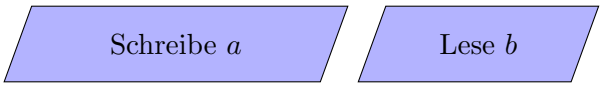


Abbildung 4.6: Eine Ausgabe und eine Eingabe.

4.3.6 Programmaufrufe

Oftmals möchte man wiederkehrende Funktionen nicht mehrfach aufschreiben und verwendet deshalb Programmaufrufe oder Unterprogramme.

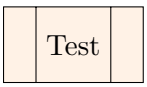
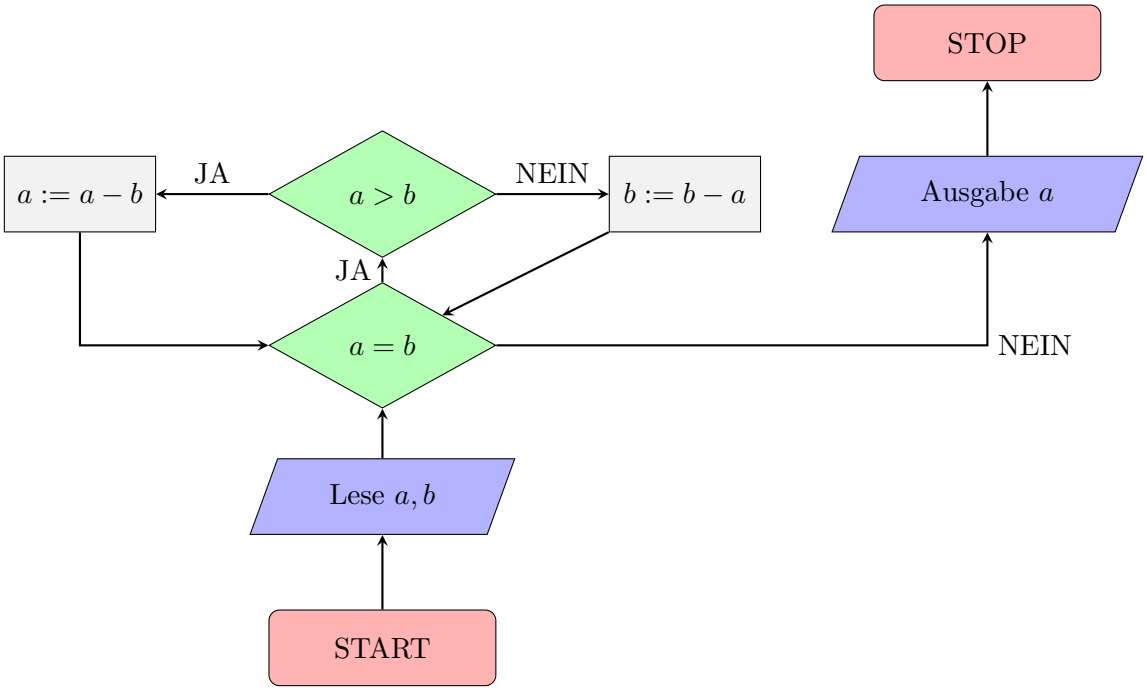


Abbildung 4.7: Ein Aufruf eines Unterprogrammes.

4.3.7 Anwendung von Flussdiagrammen

Hier nun nochmal das Beispiel von oben mit schönen bunten Bildern.



4.3.8 Andere Darstellungen

Pseudocode

Hier soll das oben dargestellte Beispiel noch einmal kurz im Pseudocode angegeben werden.

Algorithm 1 Beispielpseudocode

```
1: Lese a, b
2: while a ≠ b do
3:     if a > b then
4:         a := a − b
5:     else
6:         b := b − a
7:     end if
8: end while
9: Ausgabe a
```

Wie bereits erwähnt benutzt Pseudocode Schlüsselworte um die verschiedenen Kontrollfunktionen aus Flussdiagrammen zu ersetzen. Verzweigungen werden oftmals mit **if,**

else und Schleifen mit **while, for** eingeleitet. Weiter gibt es diverse Möglichkeiten die Blöcke zu begrenzen (in unserem Beispiel **do, end if, end while**).

Struktogramme

Der Vollständigkeit halber möchte ich zusätzlich ein Struktogramm zeigen. Dieses ist in Abbildung 4.8 abgebildet.

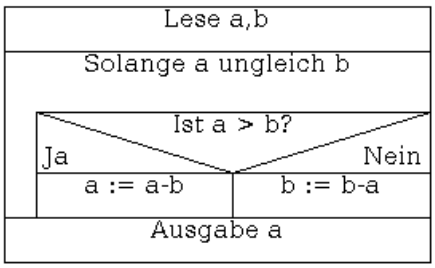


Abbildung 4.8: Ein Struktogramm unseres Beispieles.

5 Python

5.1 Geschichte

Python wurde 1991 von Guido van Rossum als Weiterentwicklung von ABC entwickelt. Das Ziel war eine leicht zu erlernende Programmiersprache zu entwickeln, die der Englischen Sprache angelehnt ist. Die erste Vollversion wurde 1994 veröffentlicht. Danach wurde 2000 Python 2.0, 2008 Python 3.0, 2015 Python 3.5 und 2016 Python 3.6 veröffentlicht. Der Name geht nicht wie viele Denken auf die Schlange zurück, sondern auf die Komikertruppe MontyPython. Trotzdem gibt es viele Anspielungen auf Schlangen in weiteren Paketen wie Boa oder in dem Tool Anacondo.

5.2 Installation und Start

Python kann sehr einfach über die Website `www.python.org` installiert werden. Wir verwenden in der Vorlesung Python 3. Die interaktive Konsole kann unter Linux mithilfe des Befehles `$python3` gestartet werden. Danach können hinter `>>>` die Python Befehle geschrieben werden.

5.3 Syntax

> Unter Syntax versteht man allgemein ein Regelsystem zur Kombination elementarer Zeichen zu zusammengesetzten Zeichen in natürlichen oder künstlichen Zeichensystemen. Die Zusammenfügungsregeln der Syntax stehen hierbei den Interpretationsregeln der Semantik gegenüber. - Wikipedia

Im folgenden besprechen wir einige Syntaxregeln der Sprache Python 3.0. Um mit Python zu programmieren können wir die interaktive Umgebung nutzen. Wir starten sie über die Konsole Mithilfe des Befehls `python3`.

```
$ python3
>>>
```

Listing 5.1: Start der interaktiven Konsole von Python3

5.3.1 Anweisungen

Anweisungen werden einfach eingegeben und mit ENTER bestätigt. Zuweisungen können mit `=` vorgenommen werden. In dem Beispiel wird a der Wert 3 zugewiesen und in der nächsten Zeile $2 + a$ berechnet. Das Ergebnis 5 wird sofort ausgegeben.

```
>>> i = 0
>>> n = 10
>>> while i < n:
        print(i)
        i = i+1
0
1
..
9
>>> for i in range(10):
        # range produziert d
        print(i)
0
1
..
9
```

Listing 5.4: Beispiele zu Schleifen

```
>>> a = 3
>>> 2 + a
5
```

Listing 5.2: Einfache Anweisungen

5.3.2 Verzweigungen

Verzweigungen werden Mithilfe des Schlüsselwortes `if` eingeleitet. Nach der Bedingung folgt ein :. In der nächsten Zeile werden nun mit einem Einschub die Anweisungen der `wahr` Fälle eingegeben. Folgend kann optional `elif` oder `else` angegeben werden. `elif` erwartet wieder eine Bedingung. Beide werden mit einem : beendet und in der nächsten Zeile mit Einschub die Anweisungen angegeben.

```
>>> if a > 2:
      print(a)
    elif a == 2:
      print("a ist 2")
    else:
      print("Fehler")
a ist 2
```

Listing 5.3: Verzweigung über a

5.3.3 Schleifen

Schleifen werden entweder mit `while` oder `for` eingeleitet. Der Unterschied besteht in der Verwendung. Die `while`-Schleife erwartet eine Bedingung und wird so lange ausgeführt, wie die Bedingung zu wahr ausgewertet wird. Die `for`-Schleife im Gegensatz kann dazu verwendet werden Elemente einer Liste oder eines Bereiches (`range`) zu durchlaufen.

```
>>> a = list()
>>> a.append(1)
>>> a.append(4)
>>> a.append(3)
>>> a.sort()
>>> print(len(a))
3
>>> for i in range(len(a)):
      print(i)
0
1
2
>>> for i in a:
      print(i)
1
3
4
>>> a.pop(1)
3
>>> print(a)
[1,4]
```

Listing 5.6: Verzweigung über a

5.3.4 Ein- und Ausgabe

Ausgaben werden einfach mit dem `print` Befehl realisiert. Mithilfe des Befehls `input` können Inhalte eingelesen werden. Diese werden als Zeichenkette eingelesen und können mittels `int` oder `float` umgewandelt werden.

```
>>> a = int(input("Zahl:-"))
Zahl: 3
>>> print(a)
3
```

Listing 5.5: Ein- und Ausgabe

5.3.5 Listen

Listen sind ein elementarer Datentyp von Python. Eine Liste kann einfach als `[]` oder `list()` angelegt werden. Mithilfe von `append` können Elemente hinzugefügt werden und `pop` entfernt ein Element. Die Länge der Liste liefert der Befehl `len`. Mithilfe von `sort` können Listen sortiert werden.

5.3.6 Funktionen

Wiederkehrende Aufgaben möchte man häufig nicht jedes mal neu schreiben sondern wiederbenutzen. Hierzu eigenet sich die Definition einer Funktion. Eine Beispielfunktion ist die Fakultät. Wir wollen nicht jedes mal neu beschreiben, wie man die Fakultät aussrechnen sondern lieber einfach die Zahl n angeben und das Ergebnis erhalten. Dies könnte die Funktion fak sein. n nennt man hierbei den Parameter.

```
def fak(n):
  m = 1
  for i in range(1,n+1):
    m *= i
  return m
```

Abbildung 5.1: Beispiel einer Fakultätsfunktion mit Parameter n.

Beispiel 5 (Binomialkoeffizient)**.** *Der Binomialkoeffizient ist eine Funktion die Anzahl der Möglichkeiten aus einer Menge mit n Objekten k Objekte auszuwählen zu berechnen. Angenommen wir schauen uns sämtliche Binomialkoeffizienten von* $n <= 100$ *und* $k < n$ *an. Wie viele dieser sind größer als* 10^6*?*

Um dieses Problem zu lösen, brauchen wir zuerst zwei Funktionen die uns die Fakultät und den Binomialkoeffizienten aussrechnet.

```
def fak(n):
  m = 1
  for i in range(1,n+1):
  m *= i
  return m

def binom(n,k):
  return fak(n)/(fak(k)*fak(n-k))
```

Abbildung 5.2: Die Funktionen Fakultät und Binomialkoeffizient

Nun können wir alle Möglichkeiten durchlaufen und schauen ob der Binomialkoeffizient größer als $10**6$ *ist. Wenn dies der Fall ist zählen wir eine Variable l eins hoch.*

```
l = 0
for n in range(1,101):
  for k in range(1,n+1):
    if binom(n,k) > 10**6:
      l += 1

print(l)
```

Abbildung 5.3: Durchtesten aller Möglichkeiten

Das `print` *an der letzten Stelle liefert uns die Antwort* 4075*.*

5.3.7 Rekursion

Wenn eine Funktionen einen Funktionsaufruf von sich selbst enthält reden wir von einer Rekursion. So können wir die Fakultätsberechnung aus dem vorherigen Beispiel auch rekursiv ausrechnen.

Der Aufruf `fak(3)` würde in diesem Fall als `fak(3) = 3*fak(2) = 3*2*fak(1) = 3*2*1 = 6` ausgewertet werden. Weitere prominente Beispiele sind die Berechnung der

Fibonacci Zahlen $f_n = f_{n-1} + f_{n_2}, f_1 = f_2 = 1, n \geq 3$ oder die Ackermann Funktion

$$\begin{aligned} a(0,m) &= m+1, \\ a(n+1,0) &= a(n,1), \\ a(n+1,m+1) &= a(n,a(n+1,m)). \end{aligned}$$

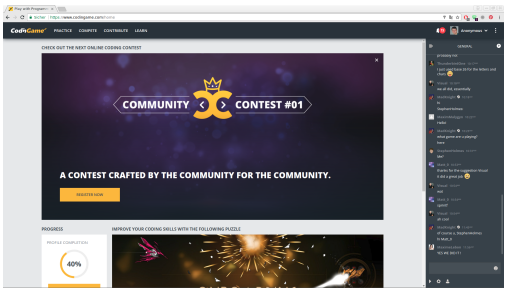
Sie können diese zur Übung selbstständig in Python Code übersetzen.

```
def fak(n):
  if n <= 1:
    return 1
  else:
    return n*fak(n-1)
```

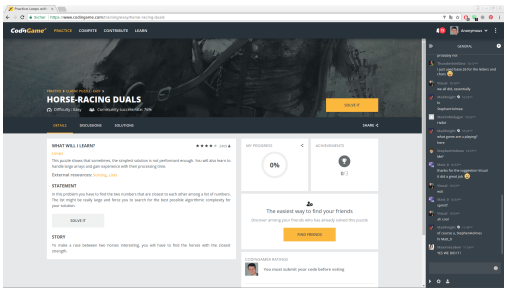
Abbildung 5.4: Rekursive Fakultätsberechnung

5.4 CodinGame

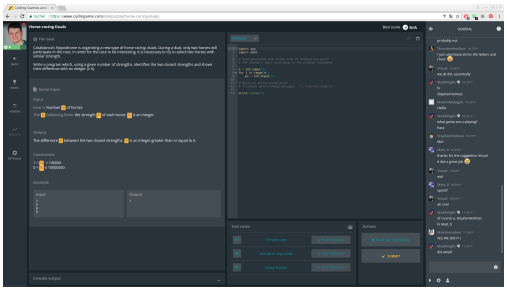
Für die Übung verwenden wir die Website CodinGame `www.codingame.com`. Die folgenden Bilder enthalten eine Übersicht über das Lösen eines Spieles. Wir verwenden als Beispiel das Spiel **Horse-racing Duals**.



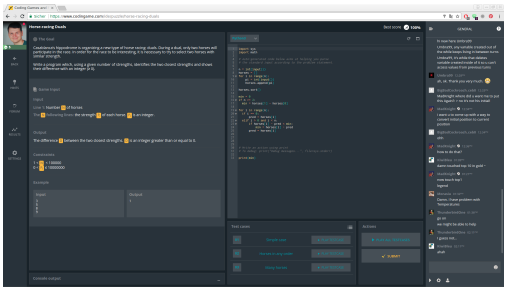
(a) Startseite von CodinGame



(b) Ein Beispielspiel



(c) Die Umgebung



(d) Die Umgebung mit Lösung

6 Kommandozeile

Das folgende Kapitel soll den Umgang mit der Konsole an einigen Beispielen demonstrieren. Hierzu werden Befehle angegeben. Die Befehle haben die Form `Befehl` *Parameter*.

6.1 Bewegen in Verzeichnissen

Am meisten muss man in der Konsole von einem Verzeichnis in ein anderes Verzeichnis mit den benötigten Dateien wechseln. Hierzu helfen die folgenden Befehle.

`cd` **Verzeichnis** wechselt in das Verzeichnis. Mit `cd ..` wird ein Verzeichnis nach oben gegangen.

`pwd` gibt den aktuellen Verzeichnispfad aus.

`ls` listet alle Ordner und Dateien des Verzeichnisses auf.

6.2 Hilfe

Sollte man einmal nicht weiter wissen kann man mit `man` *Befehl* das Manual zu dem Befehl öffnen oder mit `info` *Thema*, Befehle zu einem Thema finden.

6.3 Dateioperationen

Wenn man nun einmal in dem Verzeichnis gelandet ist, welches man benötigt kann man dort Dateien erzeugen, betrachten und löschen. Hierzu helfen folgende Befehle.

`cp` **Quelle Ziel** kopiert die Datei von der Quelle zum Ziel.

`mv` **Quelle Ziel** verschiebt die Datei von der Quelle zum Ziel.

`mkdir` **Verzeichnis** legt das Verzeichnis an. `rmdir` kann zum Löschen verwendet werden.

`touch` **Datei** erzeugt die Datei. `rm` kann zum Löschen verwendet werden.

`cat` **Datei** zeigt den Inhalt der Datei in der Konsole an. Ein alternativer Befehl ist `less`. Wenn man nur den Anfang oder das Ende der Datei anzeigen möchte kann man `head` oder `tail` benutzen.

`nano` **Datei** öffnet die Datei in einen Texteditor in der Konsole. Wenn die Datei noch nicht existiert, wird sie erzeugt.

`gedit` **Datei** öffnet die Datei in einem externen Texteditor.

6.4 Programme

Oft möchte man Programme in der Konsole starten. Diese werden einfach als `Befehl` gestartet. In unserem Kurs haben wir beispielsweise Java als `java` *-jar petri.jar* gestartet oder aber die interaktive Umgebung von Python3 als `python3`.

6.5 Weitere Befehle

Es stehen im Terminal noch weitere wichtige Befehle zur Verfügung, die uns die Arbeit im Linux Betriebssystem vereinfacht.

`clear` fügt genügend Leerzeilen ein, damit die aktuelle Befehlszeile wieder am oberen Anfang des Terminals steht.

`whatis` **Programm** liefert eine kurze Beschreibung eines übergebenen Programmes.

`which` **Programm** gibt den Installationspfad zum Programm an.

`df` zeigt die Auslastung des Dateisystems (disk free).

`ps -e` gibt eine Übersicht über die aktuell laufenden Prozesse aus. Jedem Prozess wird dabei eine PID zugewiesen.

`sudo killall -9` **PID** beendet den Prozess mit der übergebenen PID.

`shutdown -h` **delay** fährt den Rechner nach delay Minuten herunter. Falls kein delay angegeben wird fährt der Rechner sofort herunter.