Tafeln und Übersichten für das Fach Informatik in der gymnasialen Oberstufe

Mecklenburg-Vorpommern

Version 2029/2030

Aufgabenkommission Informatik MV

Dieses Dokument liegt als Entwurfsfassung vor und kann Fehler enthalten. Über Inhalt und Umfang wurde noch keine endgültige Entscheidung getroffen.
Ein Einsatz im Rahmen von Leistungsüberprüfungen ist aktuell ausgeschlossen.

Rückmeldungen bitte an t.hempel_02@iq.bm.mv-regierung.de

1 Daten

1.1 Einheiten und Präfixe

Bit	kleinste Informationseinheit kleinster Speicherplatz hat genau einen von zwei logischen Werten typischerweise 0 oder 1
Byte	8 Bit

Binärpräfix		
Name	**Präfix**	**Wert**
Kibi	Ki	$2^{10} = 1024$
Mebi	Mi	$2^{20} = 1048576$
Gibi	Gi	2^{30}
Tebi	Ti	2^{40}
Pebi	Pi	2^{50}
Exbi	Ei	2^{60}

SI-Präfix ähnlicher Größe		
Name	**Präfix**	**Wert**
Kilo	k	$10^3 = 1000$
Mega	M	$10^6 = 1000000$
Giga	G	10^9
Tera	T	10^{12}
Peta	P	10^{15}
Exa	E	10^{18}

Beispiel: 298 MiByte = 298 · 2^{20} Byte = 312475648 Byte ≈ 312 MByte.
Hinweis: Im alltäglichen Umgang werden SI- und Binärpräfix fälschlicherweise oft synonym verwendet.

1.2 Datentypen

	Größe (typisch) **Wertebereich (typisch)**	**Beispiel**	**Operationen (Auszug)**	**Hinweis**
Wahrheitswert	1 Bit WAHR, FALSCH	`true` `false`	Negation logisches ODER	

Ganzzahl	32 Bit
	-2^{31} ... $2^{31}-1$
		`-142`
		`0`
		`2098382`
			Addition
			Subtraktion
			Multiplikation
			ganzzahlige Division
				↗ Seite 5

Gleitkommazahl	64 Bit mit
	– Vorzeichen: 1 Bit
	– Mantisse: 52 Bit
	– Exponent: 11 Bit
		`-0.25`
		`0.0`
		`98.775`
			Addition
			Subtraktion
			Multiplikation
			Division
				↗ Seite 6

Zeichen	je nach Codierung beispielsweise 1 bis 4 Byte bei UTF-8	`'a'` `'\u20AC'`	Vorgänger Nachfolger	↗ Seite 7
Zeichenkette	automatisch angepasst	`"Schule"`	Verketten	

1.3 Zahlsysteme

Zahlen im Dezimalsystem

Basis	10
Ziffern	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Stellentafel

10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
0	0	0	9	2	0	8	1

$92081_{[10]} = 0 \cdot 10^7 + 0 \cdot 10^6 + 0 \cdot 10^5 + 9 \cdot 10^4 + 2 \cdot 10^3 + 0 \cdot 10^2 + 8 \cdot 10^1 + 1 \cdot 10^0$
$= 90000 + 2000 + 80 + 1$

Zahlen im Binärsystem

Basis	2
Ziffern	0, 1

Stellentafel

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	1	0	1	1

$10011011_{[2]} = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
$= 128 + 16 + 8 + 2 + 1$
$= 155$

Additionsregeln	0 + 0 = 0 ; 0 + 1 = 1 ; 1 + 0 = 1 ; 1 + 1 = 0 Übertrag 1
Multiplikationsregeln	0 · 0 = 0 ; 0 · 1 = 0 ; 1 · 0 = 0 ; 1 · 1 = 0

Zahlen im Hexadezimalsystem

Basis	16
Ziffern	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Stellentafel

16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0
0	0	0	1	F	0	A	2

$1F0A2_{[16]} = 1 \cdot 16^4 + 15 \cdot 16^3 + 0 \cdot 16^2 + 10 \cdot 16^1 + 2 \cdot 16^0$
$= 65536 + 61440 + 160 + 2$
$= 127138$

Hexadezimalziffer und binäre Darstellung

Ziffer	2^3	2^2	2^1	2^0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1

Ziffer	2^3	2^2	2^1	2^0
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

1.4 Datencodierung

Codierung vorzeichenloser Ganzzahlen mit n Bit	
Struktur	Binärdarstellung
Wertebereich	$0 \dots 2^n - 1$
Beispiel	n = 8 → Wertebereich 0 … 255
	Umwandlung von $187_{[10]}$ in die binäre Darstellung mit 8 Bit: 187 : 2 = 93 Rest 1 93 : 2 = 46 Rest 1 46 : 2 = 23 Rest 0 23 : 2 = 11 Rest 1 11 : 2 = 5 Rest 1 5 : 2 = 2 Rest 1 2 : 2 = 1 Rest 0 1 : 2 = 0 Rest 1 Ergebnis: $187_{[10]} = 10111011_{[2]}$
	Umwandlung der Ganzzahl $11001011_{[2]}$ in die dezimale Darstellung: $11001011_{[2]} = 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ $= 128 + 64 + 8 + 2 + 1$ $= 203_{[10]}$

Codierung vorzeichenbehafteter Ganzzahlen mit n Bit mithilfe des Zweierkomplements	
Struktur	Binärdarstellung mit Negation des Werts der höchstwertigen Stelle
Wertebereich	$-2^{n-1} \dots 2^{n-1} - 1$
Beispiel	n = 8 → Wertebereich -128 … 127
	Umwandlung von $-35_{[10]}$ in die binäre Darstellung: (1) Umwandlung von $\lvert -35_{[10]} \rvert$ in binäre Darstellung mit 8 Bit 35 : 2 = 17 Rest 1 17 : 2 = 8 Rest 1 8 : 2 = 4 Rest 0 2 : 2 = 1 Rest 0 1 : 2 = 0 Rest 1 Ergebnis: $\lvert -35_{[10]} \rvert = 00010011_{[2]}$ (2) Bildung des Zweierkomplements von (1) durch bitweises Invertieren und anschließendes Inkrementieren: $00010011_{[2]} \rightarrow 11101101_{[2]}$ (3) Ergebnis: $-35_{[10]} = 11101101_{[2]}$
	Umwandlung von $11001011_{[2]}$ in die dezimale Darstellung: $11001011_{[2]} = -1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ $= -128 + 64 + 8 + 2 + 1$ $= -53_{[10]}$
Bildung eines Zweierkomplements	(1) Bitweises Invertieren der Binärdarstellung der Ausgangszahl (2) Bildung des Nachfolgers durch Addition von 1 (Inkrement)

Codierung vorzeichenloser Festkommazahlen mit n Bit	
Struktur	Binärdarstellung der Form $b_{k-1} \cdot 2^{k-1} + \ldots + b_0 \cdot 2^0 + b_{-1} \cdot 2^{-1} \ldots + b_{-m} \cdot 2^{-m}$ mit k Bit für Vorkommateil, m Bit für Nachkommateil und n = k + m
Wertebereich	$0 \ldots 2^k - 2^{-m}$ mit der Schrittweite 2^{-m}
Beispiel	n = 8, k = 5, m = 3 → Wertebereich 0 … 31,875 mit Schrittweite 0,125
	Umwandlung von $2{,}125_{[10]}$ in die binäre Darstellung: (1) Umwandlung des ganzen Teils der Dezimalzahl in die binäre Darstellung mit 5 Bit Teilergebnis: $2_{[10]} \rightarrow 00010_{[2]}$ (2) Umwandlung des Nachkommateils der Dezimalzahl in die binäre Darstellung mit 3 Bit 0,125 · 2 = 0,25 → Ganzzahl 0 0,25 · 2 = 0,50 → Ganzzahl 0 0,50 · 2 = 1,00 → Ganzzahl 1 Teilergebnis: $0{,}125_{[10]} \rightarrow 001_{[2]}$ (3) Zusammensetzen zum Ergebnis: $2{,}125_{[10]} = 00010001_{[2]}$
	Umwandlung von $11001011_{[2]}$ in die dezimale Darstellung: $11001011_{[2]} = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}$ $= 16 + 8 + 1 + 0{,}25 + 0{,}125$ $= 25{,}375_{[10]}$
Hinweise	Festkommazahlen ermöglichen ausschließlich diskrete Werte. Vorzeichenbehaftete Festkommazahlen mittels Zweierkomplement

Codierung einzelner Zeichen mit n Bit	
ASCII	Zuordnung Zeichen ↔ 7 Bit 128 Zeichen, davon 32 Steuerzeichen
Unicode	Zuordnung Zeichen ↔ Codepunkt in der Form U+0000 … U+10FFFF Verfahren zur Kodierung der Codepunkte: – UTF-8: Codierung als Folge von einem bis vier Bytes – UTF-16: Codierung als Folge von ein oder zwei 16 Bit-Einheiten

Codierung von Buchstaben

Zeichen	ASCII	Unicode	Zeichen	ASCII	Unicode	Zeichen	ASCII	Unicode
A	65	U+0041	U	85	U+0055	k	107	U+006B
B	66	U+0042	V	86	U+0056	l	108	U+006C
C	67	U+0043	W	87	U+0057	m	109	U+006D
D	68	U+0044	X	88	U+0058	n	110	U+006E
E	69	U+0045	Y	89	U+0059	o	111	U+006F
F	70	U+0046	Z	90	U+005A	p	112	U+0070
G	71	U+0047	Ä		U+00C4	q	113	U+0071
H	72	U+0048	Ö		U+00D6	r	114	U+0072
I	73	U+0049	Ü		U+00DC	s	115	U+0073
J	74	U+004A	ß		U+1E9E	t	116	U+0074
K	75	U+004B	a	97	U+0061	u	117	U+0075
L	76	U+004C	b	98	U+0062	v	118	U+0076
M	77	U+004D	c	99	U+0063	w	119	U+0077
N	78	U+004E	d	100	U+0064	x	120	U+0078
O	79	U+004F	e	101	U+0065	y	121	U+0079
P	80	U+0050	f	102	U+0066	z	122	U+007A
Q	81	U+0051	g	103	U+0067	ä		U+00E4
R	82	U+0052	h	104	U+0068	ö		U+00F6
S	83	U+0053	i	105	U+0069	ü		U+00FC
T	84	U+0054	j	106	U+006A	ß		U+00DF

Codierung von Sonderzeichen, Satzzeichen, Ziffern

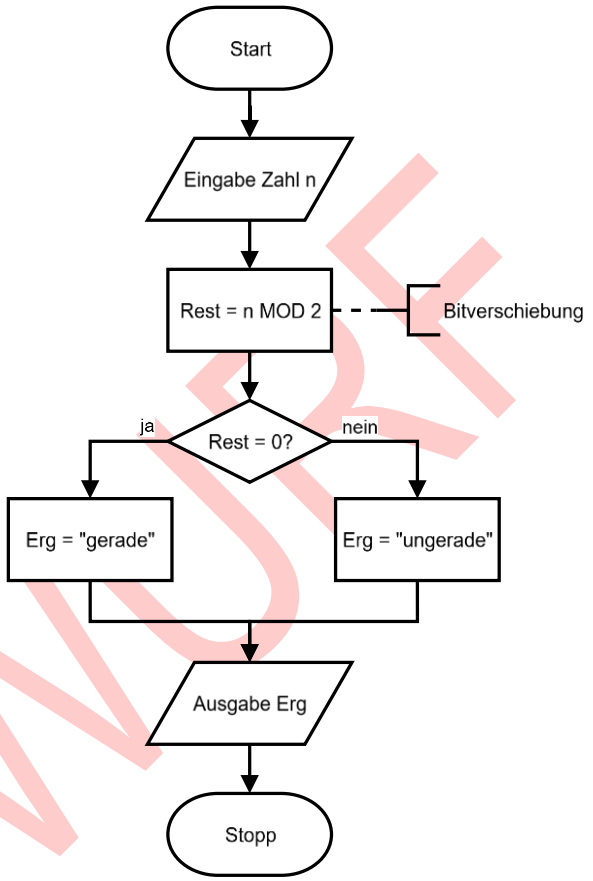
Zeichen	ASCII	Unicode	Zeichen	ASCII	Unicode	Zeichen	ASCII	Unicode
	32	U+0020	0	48	U+0030	@	64	U+0040
!	33	U+0021	1	49	U+0031	[	91	U+005B
"	34	U+0022	2	50	U+0032	\	92	U+005C
#	35	U+0023	3	51	U+0033	]	93	U+005D
$	36	U+0024	4	52	U+0034	^	94	U+005E
%	37	U+0025	5	53	U+0035	_	95	U+005F
&	38	U+0026	6	54	U+0036	`	96	U+0060
'	39	U+0027	7	55	U+0037	{	123	U+007B
(	40	U+0028	8	56	U+0038	|	124	U+007C
)	41	U+0029	9	57	U+0039	}	125	U+007D
*	42	U+002A	:	58	U+003A	~	126	U+007E
+	43	U+002B	;	59	U+003B	§		U+00A7
,	44	U+002C	<	60	U+003C	°		U+00B0
-	45	U+002D	=	61	U+003D	€		U+20AC
.	46	U+002E	>	62	U+003E	≠		U+2260
/	47	U+002F	?	63	U+003F	⛔		U+26D4

1.5 Datenschutz und Datensicherheit

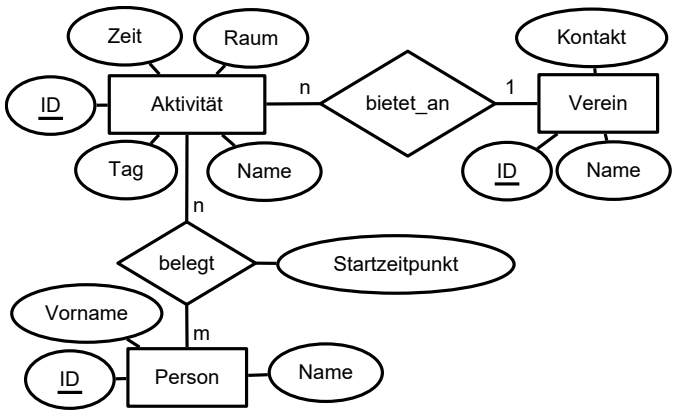
Datenschutz	Rechtliche Regelungen zur Verarbeitung personenbezogener Daten mit dem Ziel der Gewährleistung von Privatsphäre und informationeller Selbstbestimmung Prinzipien: Zweckbindung, Datenminimierung, Transparenz, Datensicherheit, Betroffenenrechte
Datensicherheit	Technische Maßnahmen zum Schutz von Daten zur Sicherstellung von Verfügbarkeit, Integrität und Vertraulichkeit

2 Diagramme

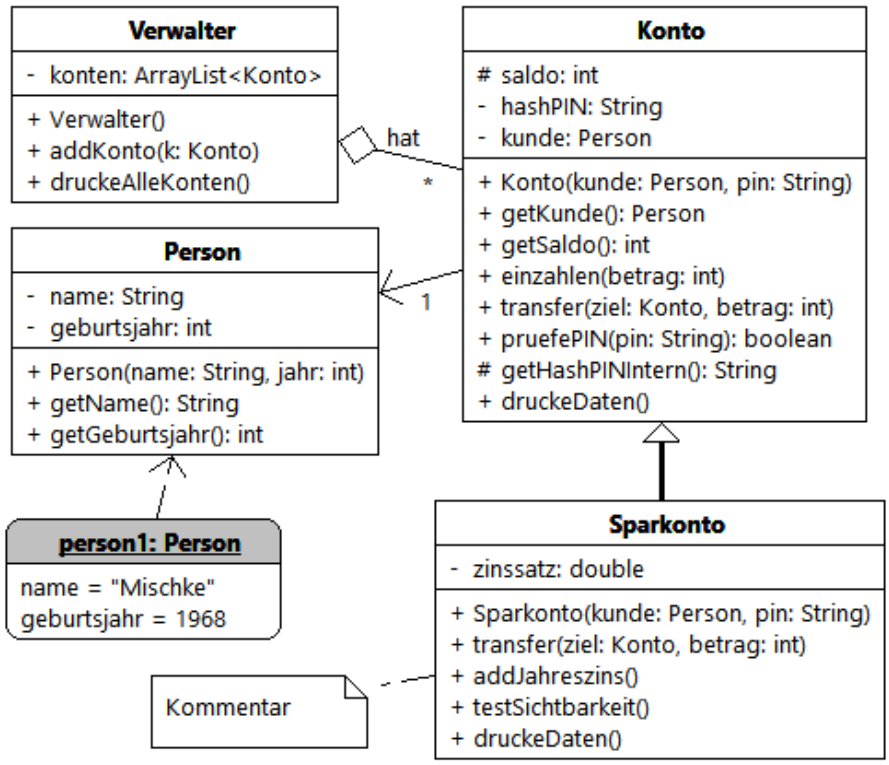
2.1 Ablaufdiagramm

Element	Symbol	Beispiel
Terminator	Start Stopp	
Operation		
Unterprogramm		
Verzweigung Entscheidung		
Eingabe/Ausgabe		
Ablaufrichtung		
Kommentar		

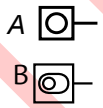
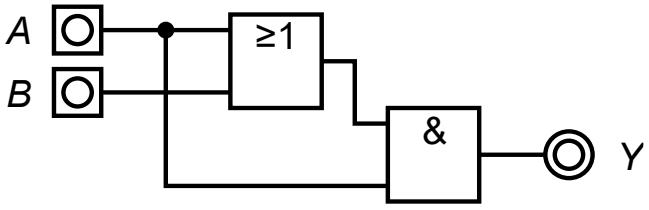
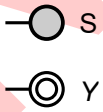
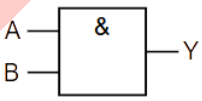
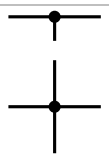
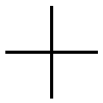
2.2 Entity-Relationship-Diagramm

Element	Symbol	Beispiel
Entitätstyp	Bezeichnung	
Attribut	Name	
Schlüsselattribut	ID	
Beziehungstyp	bezieht	

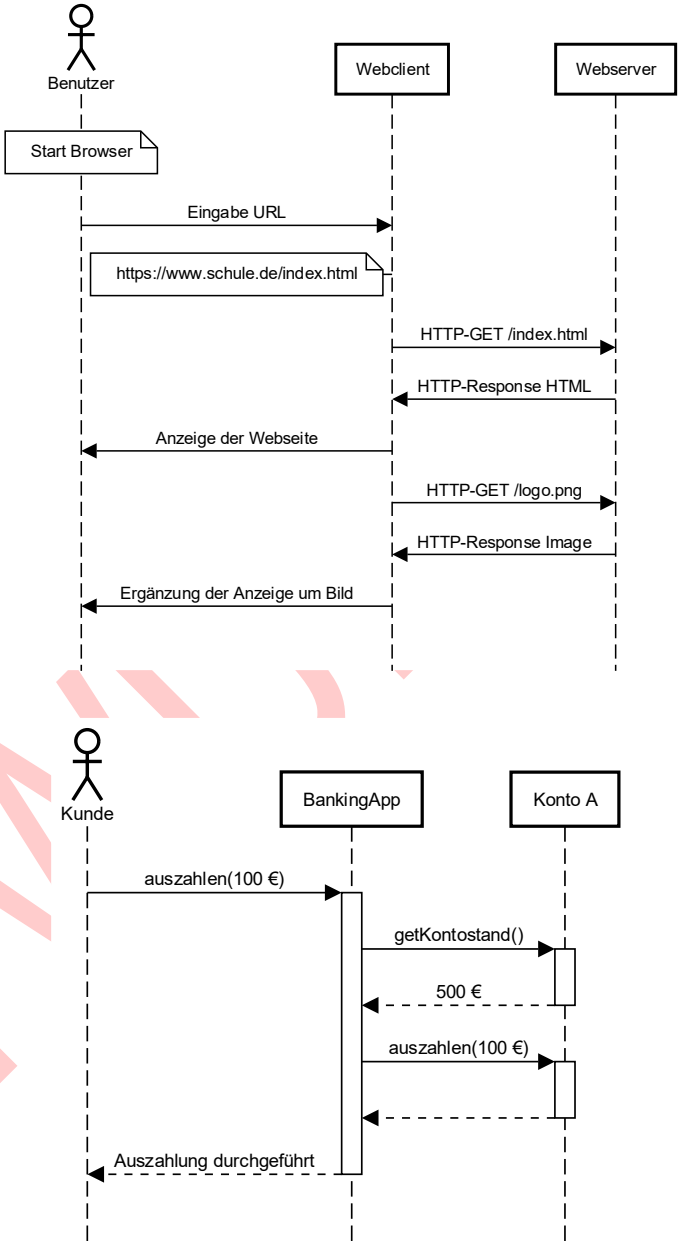
2.3 Klassendiagramm

Element ↗ Seite 29	Beispiel
Objekt Klasse Assoziation mit Multiplizität Aggregation mit Multiplizität Vererbung Kommentar	

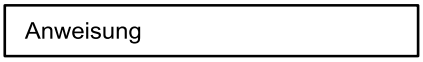
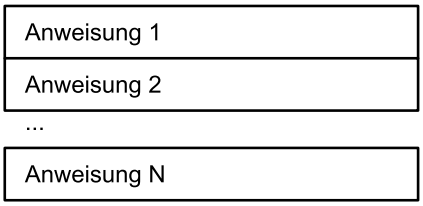
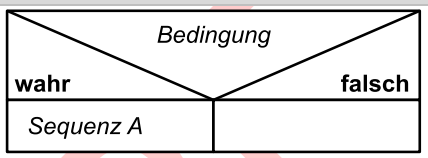
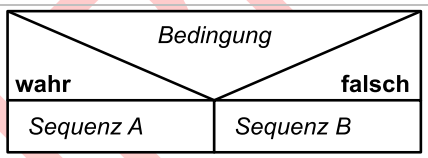
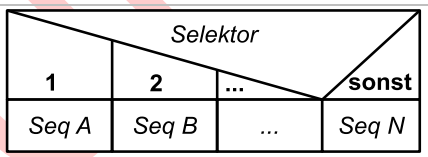
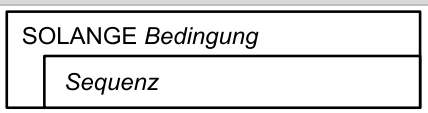
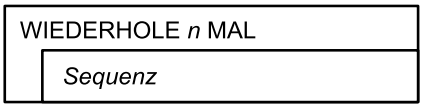
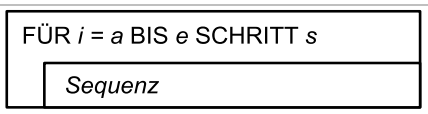
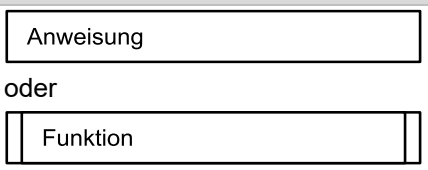
2.4 Schaltplan

Element	Symbol	Beispiel
Schalter Eingabe		
LED Ausgabe		
Gatter		
Verbindungsstelle		
Leiterkreuzung		

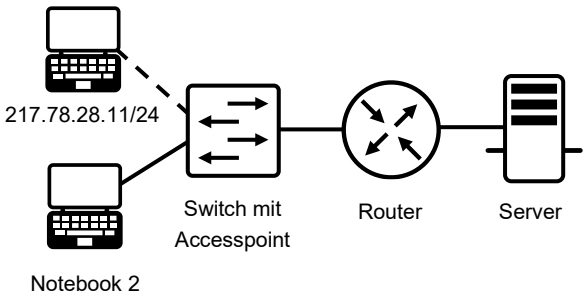
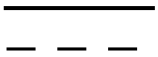
2.5 Sequenzdiagramm

Element	Symbol	Beispiele
Akteur		
Objekt System		
Lebenslinie		
Aktivierung		
Paket Nachricht Methodenaufruf		
Rückgabe		
Kommentar		

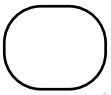
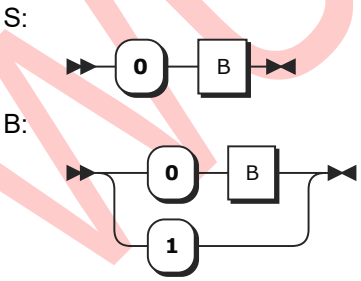
2.6 Struktogramm

Element	Verbale Formulierung	Struktogramm
Anweisung	*Anweisung*	
Sequenz	*Anweisung 1* *Anweisung 2* … *Anweisung n*	
Auswahl		
Einseitige Auswahl	WENN *Bedingung* DANN *Sequenz*	
Zweiseitige Auswahl	WENN *Bedingung* DANN *Sequenz 1* SONST *Sequenz 2*	
Mehrseitige Auswahl	WENN *Selektor* 1: *Seq 1* 2: *Seq 2* … SONST: *Seq n*	
Schleifen		
Bedingte Schleife	SOLANGE *Bedingung* WIEDERHOLE *Sequenz*	
Schleife mit fester Anzahl	WIEDERHOLE *n* MAL *Sequenz*	
Zählschleife	FÜR *i* VON *a* BIS *e* IN *s*-SCHRITT WIEDERHOLE *Sequenz*	
Funktion		
Funktionsaufruf	*Funktionsname*	

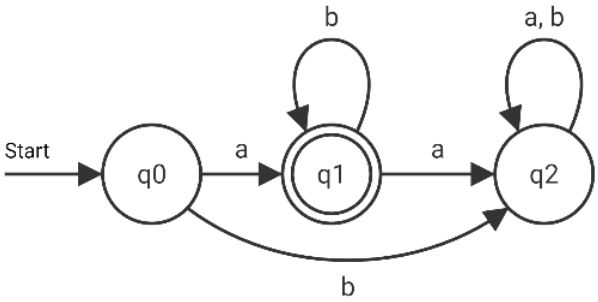
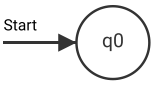
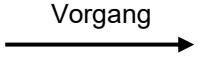
2.7 Netzstrukturdiagramm

Element	Symbolbeispiel	Beispiel
Endgerät		
Switch		
Router		
Wolke Black Box		
Kabel Funk		

2.8 Syntaxdiagramm

Element	Symbol	Beispiel
Terminal: Element des Alphabets		
Nichtterminal: Hilfssymbol, Variable		

2.9 Zustandsübergangsdiagramm

Element	Symbol	Beispiel
Zustand		
Startzustand		
Endzustand		
Zustandsübergang		

3 Datenbanksysteme

3.1 Entity-Relationship-Modell

Entitätstyp	Entitäten (Objekte) mit gleichartigen Eigenschaften (Attributen)
Beziehungstyp	Gleichartiger Zusammenhang zwischen Entitäten
Attribut	Eigenschaft der Entitäten eines Entitätstyps
Kardinalität	maximale Anzahl von Entitäten eines Entitätstyps, die mit genau einer Entität des anderen Entitätstyps in Beziehung stehen Grundformen: 1:1 1:n n:m
Darstellung im ER-Diagramm	↗ Seite 9

3.2 Relationales Datenbankmodell

Aktivität	ID	Name	Tag	Zeit	Raum	↑Verein.ID
	262	Fußball	Dienstag	14:30	TH1	Sp1
	285	Handball	Dienstag	16:00	TH1	Sp1
	286	Schach	Montag	14:30	2.01	Sp2
	319	Zeichnen	Mittwoch	15:00	2.01	Km5
	…		…	…	…	…

Notationen	
Grundnotation	`Relationsname (Primärschlüssel, Attribut, …, ↑Fremdschlüssel)`

Hinweise	Primärschlüssel wird oft mit `PK` für primary key und Fremdschlüssel mit `FK` für foreign key abgekürzt.
	Fremdschlüssel werden oft in einer der drei Weisen notiert:
	`↑Fremdschlüssel`
	`↑Tabelle.Schlüssel`
	`↑Tabelle.Schlüssel AS Fremdschlüssel`

Beispiel	`Aktivität (ID, Name, Tag, Zeit, Raum, ↑Verein.ID)`

Umwandlungsregeln ER-Modell → relationales Modell	
Entitätstyp	`Relationsname (Primärschlüssel, Attribut, …)`
n:m-Beziehung	`Relationsname (↑PK1, ↑PK2, Attribut, …)` Es entsteht eine neue Relation. Die Primärschlüssel der beiden Entitäten werden als Fremdschlüssel übernommen. Sie bilden zusammen den Primärschlüssel. Bei mehrfach möglichen Beziehungen kann der Primärschlüssel um weitere Attribute (z. B. Zeitangaben) erweitert werden.
1:n-Beziehung	Der Primärschlüssel der 1-Seite wird als Fremdschlüssel in die Relation der n-Seite übernommen. Beziehungsattribute werden ebenfalls dort zugeordnet.
1:1-Beziehung	Der Primärschlüssel einer der beiden Relationen wird als Fremdschlüssel in die andere Relation übernommen. Beziehungsattribute werden ebenfalls dort zugeordnet.

Beispiel	`Person (ID, Name, Vorname)`
	`Verein (ID, Name, Kontakt)`
	`Aktivität (ID, Name, Tag, Zeit, Raum, ↑Verein.ID)`
	`Belegung (↑Person.ID, ↑Aktivität.ID, Startzeitpunkt)`

Normalisierung	
Ziele	Strukturierung von Daten Minimierung von Redundanzen Vermeidung von Änderungs-, Einfüge- und Löschanomalien
1. Normalform	Die Relation enthält ausschließlich atomare Attributwerte.
2. Normalform	Die Relation liegt in erster Normalform vor. Jedes Nichtschlüsselattribut ist vom gesamten Primärschlüssel voll funktional abhängig.
3. Normalform	Die Relation liegt in zweiter Normalform vor. Es gibt keine transitiven Abhängigkeiten vom Primärschlüssel.

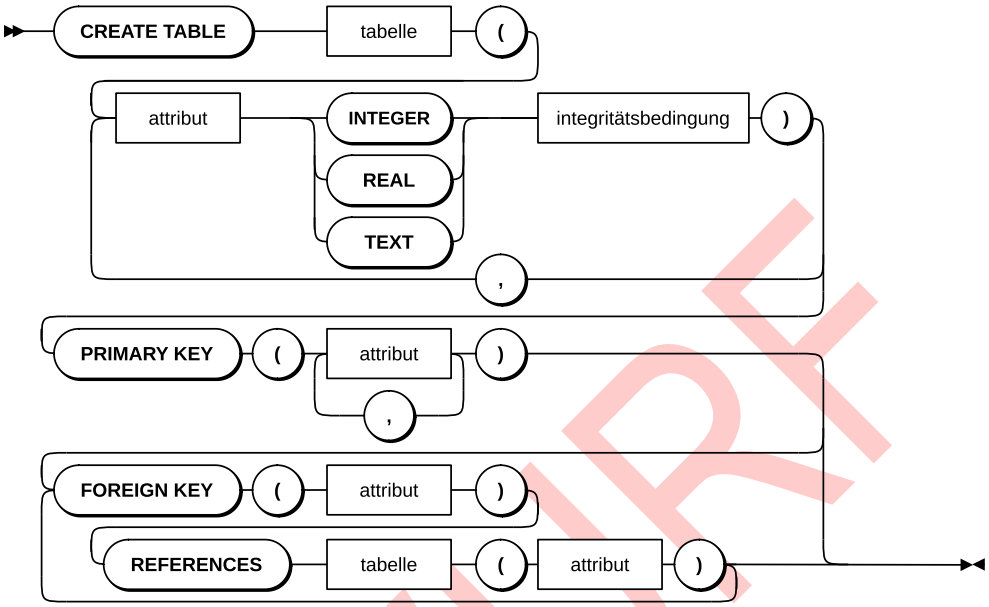
Elemente der Relationenalgebra	
Selektion	Auswahl von Datensätzen einer Relation, die einer Bedingung genügen. Notation: $S_{Bedingung}(Relation)$
Projektion	Reduktion einer Relation auf ausgewählte Attribute mit Entfernung von Duplikaten. Notation: $P_{AttributA,\ AttributB}(Relation)$
Verbund (Join)	Verknüpfung zweier Relationen zu einer neuen Relation über ein gemeinsames Attribut Notationen: $J_{Attribut}(RelationA, RelationB)$ $J_{RelationA.AttributA\ =\ RelationB.AttributB}(RelationA, RelationB)$ $RelationA \bowtie RelationB$

3.3 Datenbanksprache SQL

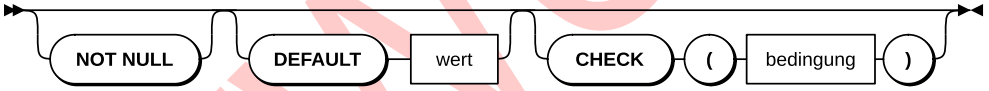
Hinweis: Die Diagramme zeigen eine inhaltlich reduzierte, nicht vollständig ausgeführte Syntax. Für eine vollständige Beschreibung wird auf die Dokumentation verwiesen.

Definition einer Relation und Manipulation von Daten

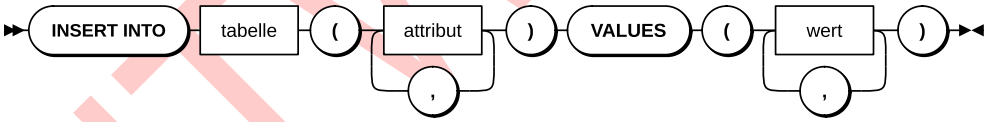
Erstellen



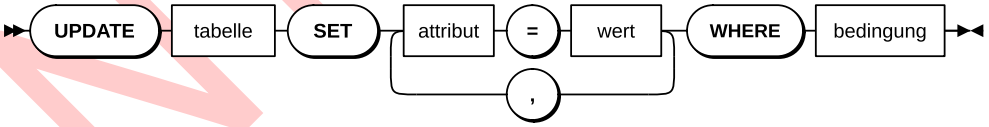
integritätsbedingung:



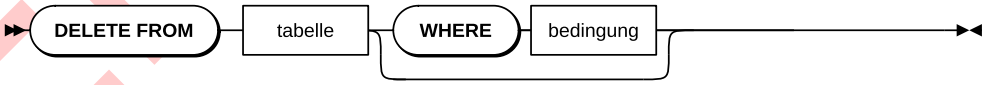
Einfügen



Ändern

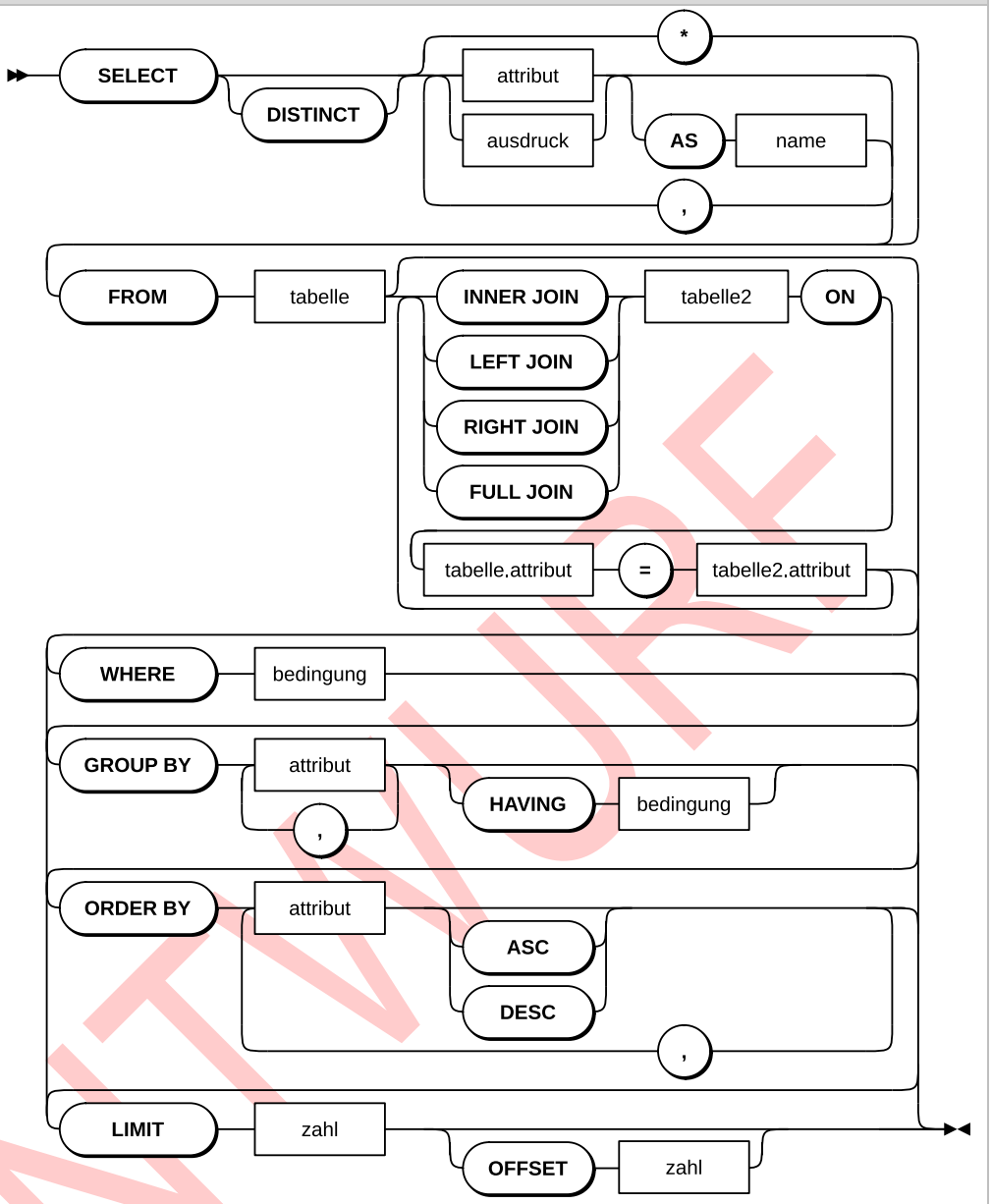


Löschen



Abfragen von Daten (Überblick)

Überblick



Bedingung	Ausdruck mit dem Ergebnis WAHR oder FALSCH	
	Vergleichsoperatoren	`= != < <= > >=`
	Logische Operatoren	`AND OR NOT`
	Bereichsvergleich von x bis y	`wert BETWEEN x AND y`
	Aufzählungsvergleich	`wert IN (a,b,c,d)`
	NULL-Wert-Vergleich	`wert IS NULL` `wert IS NOT NULL`

	Mustervergleich mit Platzhalter
	% … beliebige Zeichen
	_ … genau ein Zeichen
		`wert LIKE muster`
		
		`Beispiel: wert LIKE 'A%'`

Ausdruck	Mathematischer Term oder eine Funktion, die einen Wert bestimmt	
	Mathematische Operationen	`+ - * / %`
	Aggregatfunktionen	`COUNT() SUM() MAX() MIN() AVG()`

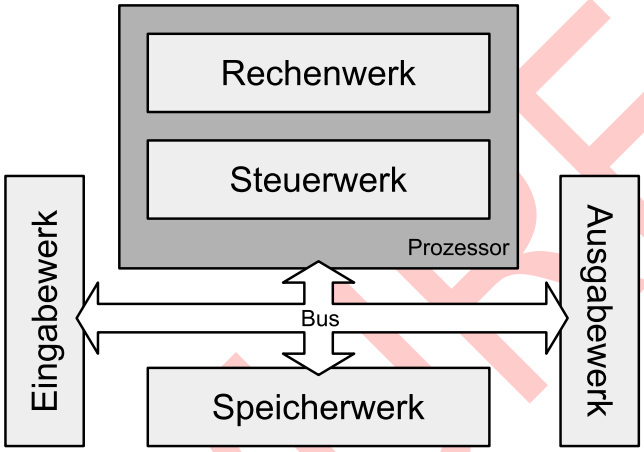
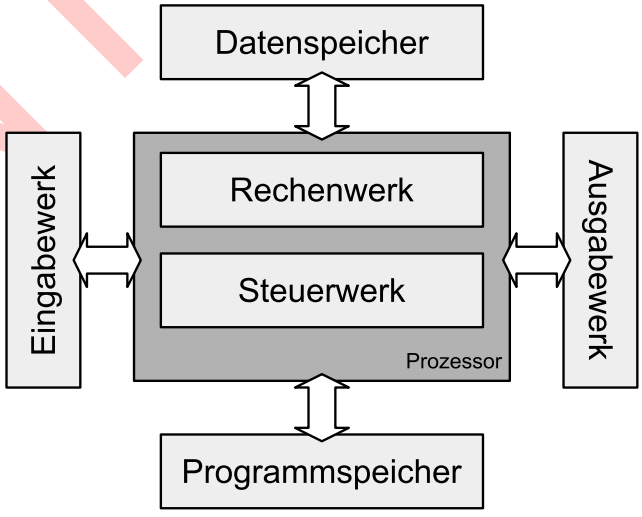
3.4 SQLite

Datentyp	Bezeichnung	Bemerkung
Ganzzahl	`INTEGER`	
Gleitkommazahl	`REAL`	
Zeichenkette	`TEXT`	
Wahrheitswert		Ersatztyp `INTEGER` mit `0` ≙ FALSCH, `1` ≙ WAHR
Zeitstempel		Ersatztyp `TEXT` im Format `YYYY-MM-DD hh:mm:ss`
Datum		Ersatztyp `TEXT` im Format `YYYY-MM-DD`
Uhrzeit		Ersatztyp `TEXT` im Format `hh:mm:ss`

Funktion (Auswahl)	Beschreibung
`CAST(a AS typ)`	a im Datentyp `typ`
`DATE(now)`	aktuelles Datum in der Form YYYY-MM-DD
`TIME(now)`	aktuelle Zeit in der Form hh:mm:ss
`LENGTH(z)`	Länge der Zeichenkette `z`
`LOWER(z)`	`z` in Kleinbuchstaben
`UPPER(u)`	`z` in Großbuchstaben
`REPLACE(z,z1,z2)`	`z` mit Ersetzung von `z1` durch `z2`
`SUBSTR(z,p,e)`	Teilzeichenkette von `z` ab Position `p` mit `e` Elementen
`TRIM(z)`	`z` ohne Leerzeichen am Anfang und Ende
`ABS(x)`	Betrag von `x`
`ROUND(x)`	`x` kaufmännisch gerundet
`RANDOM()`	Zufallszahl

4 Rechnerarchitektur

4.1 Von-Neumann- und Harvard-Architektur

Rechner/Computer	
Universalität	Eine Vielzahl von Problemen ist mithilfe geeigneter Programme und Daten lösbar.
Programm	Ein Programm ist eine Folge von Anweisungen, die entsprechend ihrer Abarbeitungsfolge hintereinander im Speicher steht. Sprungbefehle erlauben Abweichungen von der sequenziellen Abarbeitung der Befehle.
von-Neumann-Architektur mit Aufbau und Arbeitsprinzip	
Aufbau	
Speicher	Auf gleichgroße und fortlaufend nummerierte Zellen besteht über die Zellnummer (Adresse) lesender und schreibender Zugriff. Die Zellwerte sind binär codiert und enthalten Programmcode, Daten oder Ergebnisse.
Zyklus	
Elemente der Harvard-Architektur	
Aufbau	
Speicher	Programm- und Datenspeicher sind streng getrennt. Der Zugriff erfolgt über je einen eigenen Bus. Auf gleichgroße und fortlaufend nummerierte Zellen besteht über die Zellnummer (Adresse) lesender bzw. schreibender Zugriff. Die Zellwerte sind binär codiert.

4.2 Maschinennahe Programmierung

Johnny

Hinweise

Der Speicher ist über die Adressen 000 bis 999 erreichbar. Die Daten in einer Speicherzelle bestehen aus den zwei Teilen Hi und Lo. Es sind Werte von 00 000 bis 19 999 zulässig. Befehle werden auf dem Hi-Teil mit einem Wert von 01 bis 10 codiert. Der Operand steht dann im Lo-Teil und gibt eine Adresse an.

Befehlssatz

Befehl	Code	Funktion
`TAKE` *adr*	`01` *adr*	Lade den Wert von *adr* in den Akkumulator
`ADD` *adr*	`02` *adr*	Addiere den Wert von *adr* zum Akkumulator
`SUB` *adr*	`03` *adr*	Subtrahiere den Wert von *adr* vom Akkumulator
`SAVE` *adr*	`04` *adr*	Speichere den Akkumulatorwert auf *adr*
`JMP` *adr*	`05` *adr*	Springe zu *adr*
`TST` *adr*	`06` *adr*	Überspringe nächsten Befehl, falls Wert auf *adr* Null
`INC` *adr*	`07` *adr*	Erhöhe den Wert von *adr* um Eins
`DEC` *adr*	`08` *adr*	Vermindere den Wert von *adr* um Eins
`NULL` *adr*	`09` *adr*	Speichere den Wert 00 000 auf *adr*
`HLT`	`10`	Beende das Programm

MOPS

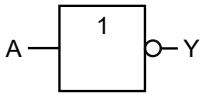
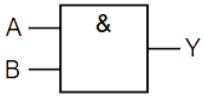
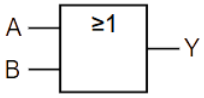
Hinweise

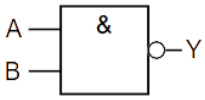
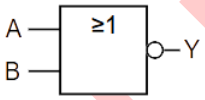
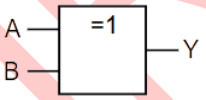
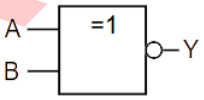
Adressen bestehen aus dem Zeichen $ gefolgt von der Nummer der Speicherzelle. Daten können ausschließlich auf den Adressen $64 bis $71 gespeichert werden. Das Ziel eines Sprungbefehls wird entweder durch die Adresse oder durch eine Zielmarke der Form „MNummer“ angegeben. Die Zielmarke wird hinter dem Befehl auf der Zieladresse in der Form „:MNummer“ definiert.

Befehlssatz

Befehl	Code	Funktion
`ld` *adr*	`10` *adr*	Lade den Wert von *adr* in Akkumulator
`ld` *val*	`11` *val*	Lade den Wert *val* in Akkumulator
`st` *adr*	`12` *adr*	Speichere den Akkumulatorwert auf *adr*
`in` *adr*	`20` *val*	Schreibe den Wert des Eingaberegisters auf *adr*
`out` *adr*	`22` *adr*	Schreibe den Wert von *adr* ins Ausgaberegister
`out` *val*	`23` *val*	Schreibe den Wert *val* ins Ausgaberegister
`add` *adr*	`30` *adr*	Addiere den Wert von *adr* zum Akkumulator
`add` *val*	`31` *val*	Addiere den Wert *val* zum Akkumulator
`sub` *adr*	`32` *adr*	Subtrahiere den Wert von *adr* vom Akkumulator
`sub` *val*	`33` *val*	Subtrahiere den Wert *val* vom Akkumulator
`mul` *adr*	`34` *adr*	Multipliziere den Wert von *adr* mit Akkumulator
`mul` *val*	`35` *val*	Multipliziere den Wert *val* mit Akkumulator
`div` *adr*	`36` *adr*	Dividiere ganzzahlig den Akkumulator durch den Wert auf *adr*
`div` *val*	`37` *val*	Dividiere ganzzahlig den Akkumulator durch den Wert *val*
`mod` *adr*	`38` *adr*	Berechne Divisionsrest von Akkumulator durch Wert auf *adr*
`mod` *val*	`39` *val*	Berechne Divisionsrest von Akkumulator durch Wert *val*
`cmp` *adr*	`40` *adr*	Vergleiche Akkumulator mit Wert auf *adr*
`cmp` *val*	`41` *val*	Vergleiche Akkumulator mit Wert *val*
`jmp` *tar*	`50` *tar*	Springe zu *tar*
`jlt` *tar*	`52` *tar*	Springe zu *tar*, falls Akkumulator bei vorherigem CMP kleiner
`jeq` *tar*	`54` *tar*	Springe zu *tar*, falls Akkumulator bei vorherigem CMP gleich
`jgt` *tar*	`56` *tar*	Springe zu *tar*, falls Akkumulator bei vorherigem CMP größer
`end`	`60 00`	Beende Programm

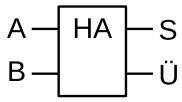
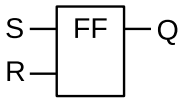
4.3 Logikgatter, Logikfunktionen und Schaltalgebra

Element	NOT nicht	AND und	OR oder
Symbol			
Funktion	$Y = \neg A$	$Y = A \wedge B$	$Y = A \vee B$
Wertetabelle	A Y 0 1 1 0	A B Y 0 0 0 0 1 0 1 0 0 1 1 1	A B Y 0 0 0 0 1 1 1 0 1 1 1 1

Element	NAND nicht und	NOR nicht oder	XOR exklusiv oder	XNOR Äquivalenz
Symbol				
Funktion	$Y = \neg(A \wedge B)$	$Y = \neg(A \vee B)$	$Y = A \oplus B$	$Y = \neg(A \oplus B)$
Wertetabelle	A B Y 0 0 1 0 1 1 1 0 1 1 1 0	A B Y 0 0 1 0 1 0 1 0 0 1 1 0	A B Y 0 0 0 0 1 1 1 0 1 1 1 0	A B Y 0 0 1 0 1 0 1 0 0 1 1 1

Bildungsvorschrift für XOR	$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$ $A \oplus B = (A \vee B) \wedge \neg(A \wedge B)$

Regeln der Schaltalgebra für NOT, AND und OR		
Reihenfolge	Klammern → NOT → AND → OR	
Doppelte Negation	$A = \neg(\neg A)$	
Kommutativgesetz	$A \wedge B = B \wedge A$	$A \vee B = B \vee A$
Assoziativgesetz	$A \wedge (B \wedge C) = (A \wedge B) \wedge C$	$A \vee (B \vee C) = (A \vee B) \vee C$
Distributivgesetz	$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$	$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
De-Morgan-Regeln	$\neg(A \wedge B) = \neg A \vee \neg B$	$\neg(A \vee B) = \neg A \wedge \neg B$
Neutralität	$A \wedge 1 = A$	$A \vee 0 = A$
Dominanzgesetz	$A \wedge 0 = 0$	$A \vee 1 = 1$
Selbstverknüpfung	$A \wedge A = A$	$A \vee A = A$
Komplementgesetz	$A \wedge \neg A = 0$	$A \vee \neg A = 1$
Absorptionsgesetz	$A \wedge (A \vee B) = A$	$A \vee (A \wedge B) = A$
Reduktionsregel	$A \wedge (\neg A \vee B) = A \wedge B$	$A \vee (\neg A \wedge B) = A \vee B$

Rechnen und Speichern		
Element	**Halbaddierer**	**RS-Flipflop**
Zweck	binäre Addition zweier Bits	Speichern eines Bits
Symbol		
Funktion	$S = A \oplus B$, $Ü = A \wedge B$	$Q_{(neu)} = f(S, R, Q_{(alt)})$
Wertetabelle	**A** **B** **S** **Ü** 0 0 0 0 0 1 1 0 1 0 1 0 1 1 0 1	**R** **S** **Q(neu)** 0 0 Q(alt) 0 1 1 1 0 0 1 1 nicht sinnvoll

Darstellung	
Schaltplan	↗ Seite 10

5 Vernetzte Systeme

5.1 Schichtenmodell und Protokolle

TCP/IP-Schicht	OSI-Schicht	Gerät Applikation	Protokoll
Anwendung	Anwendung Darstellung Sitzung	Client- und Server-Software	HTTP ..... Hypertext Transfer Protocol SMTP ..... Simple Mail Transfer Protocol POP3 ..... Post Office Protocol IMAP ..... Internet Message Access Protocol DNS ..... Domain Name System DHCP ..... Dynamic Host Configuration Protocol
Transport	Transport	Firewall	TCP ..... Transmission Control Protocol UDP ..... User Datagram Protocol
Internet	Vermittlung	Router Firewall	IP ..... Internet Protocol ICMP ..... Internet Control Message Protocol
Netzzugang	Sicherung Bitübertragung	Switch WLAN-AP Modem Netzkarte Netzkabel	ARP ..... Address Resolution Protocol MAC ..... Medium Access Control DSL ..... Digital Subscriber Line IEEE 802.11 .. Funknetzwerkprotokoll IEEE 802.3 .... Ethernetprotokoll

5.2 Netzstrukturen und Kommunikationsabläufe

Diagramme	
Netzstruktur	↗ Seite 13
Kommunikation	↗ Seite 11

5.3 Sichere Kommunikation

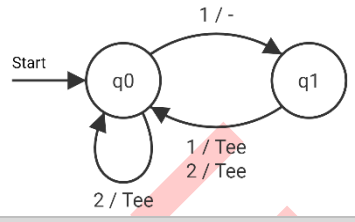
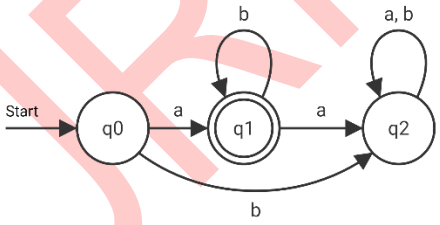
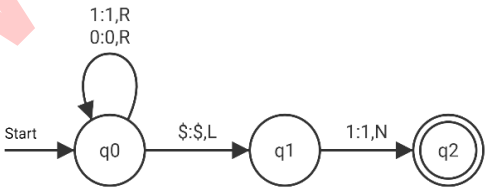
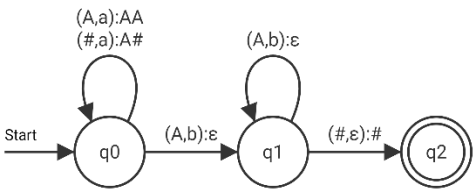
Sicherheitsziele	Vertraulichkeit, Integrität, Authentizität, Verbindlichkeit
Prinzip von Kerckhoffs	Das kryptografische Verfahren muss im Wesentlichen unentzifferbar sein und darf keine Geheimhaltung erfordern. Die Schlüssel bedürfen weiterhin der Geheimhaltung.
Schlüsselprinzip	symmetrisch ..... gleicher Schlüssel zum Ver- und Entschlüsseln asymmetrisch ..... verschiedene Schlüssel zum Ver- und Entschlüsseln
Klassische symmetrische Verfahren	Substitution ..... Zeichen werden ersetzt monoalphabetisch ..... genau ein Geheimtextalphabet polyalphabetisch ..... mehrere Geheimtextalphabete Transposition ..... Zeichen werden umgeordnet
Moderne Verfahren	Algorithmen ..... symmetrisch, asymmetrisch Protokolle ..... Schlüsseltausch, Signatur, hybride Systeme Infrastruktur ..... Zertifikate, Trust Center

Vigenère-Quadrat

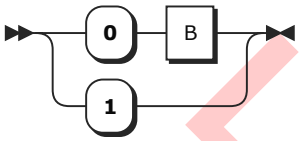
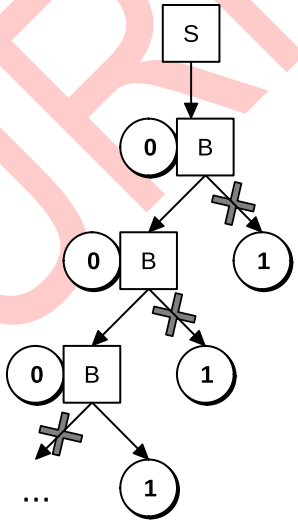
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

6 Sprachen und Automaten

6.1 Automatenmodelle

Formale Notation	Zustandsübergangsdiagramm ↗ Seite 13
Mealy-Automat MA = (X, Y, Z, δ, λ, q_0)	
X Eingabealphabet (nichtleer, endlich) Y Ausgabealphabet (nichtleer, endlich) Z Zustandsmenge (nichtleer, endlich) δ totale Überführungsfunktion Z×X → Z λ totale Ausgabefunktion Z×X → Y q_0 Startzustand $q_0 \in Z$	MA = ({10, 20}, {-, Tee}, {q_0, q_1}, δ, λ, q_0) 
Deterministischer endlicher Automat – Akzeptor A = (X, Z, δ, q_0, Z_E)	
X Eingabealphabet (nichtleer, endlich) Z Zustandsmenge (nichtleer, endlich) δ totale Überführungsfunktion Z×X → Z q_0 Startzustand $q_0 \in Z$ Z_E Endzustandsmenge $Z_E \subseteq Z$	A = ({a, b}, {q_0, q_1, q_2}, δ, q_0, {q_1}) L(A) = {ab^n, n ≥ 0} 
Deterministische Turingmaschine TM = (X, Z, Γ, δ, q_0, $, Z_E)	
X Eingabealphabet (nichtleer, endlich) Z Zustandsmenge (nichtleer, endlich) Γ Bandalphabet X ⊂ Γ δ partielle Überführungsfunktion Z×Γ → Z×Γ×{L, N, R} q_0 Startzustand $q_0 \in Z$ $ Bandvorbelegungszeichen $ ∈ Γ und $ ∉ X Z_E Endzustandsmenge $Z_E \subseteq Z$	TM = ({0, 1}, {q_0, q_1, q_2}, {$, 0, 1}, δ, q_0, $, {$q_2$}) L(TM) = {w | w endet auf 1} 
Eine Turingmaschine akzeptiert das Eingabewort, falls der aktuelle Zustand ein Endzustand und mit dem aktuellen Eingabezeichen kein Übergang mehr möglich ist. Das **Halteproblem** („Hält eine TM für eine gegebene Eingabe?“) ist nicht entscheidbar, da keine allgemeine Methode existiert, die dies für alle Programme bestimmt.	
Deterministischer/Nichtdeterministischer Kellerautomat KA = (X, Z, Γ, δ, z_0, k_0, Z_E)	
X Eingabealphabet (nichtleer, endlich) Z Zustandsmenge (nichtleer, endlich) Γ Kelleralphabet δ partielle Überführungsfunktion: deterministisch Z×(X ∪ {ε})×Γ → Z×Γ* nichtdeterministisch Z×(X ∪ {ε})×Γ → $\mathcal{P}$(Z×Γ*) z_0 Startzustand $q_0 \in Z$ k_0 Kellerleerzeichen $k_0 \in Γ$ und $k_0 \notin X$ Z_E Endzustandsmenge $Z_E \subseteq Z$	KA = ({a, b}, {q_0, q_1, q_2}, {#, A}, δ, q_0, #, {q_2}) L(KA) = {a^nb^n, n ≥ 1} 
Ein Kellerautomat akzeptiert das Eingabewort, falls er sich nach Abarbeitung des Wortes im Endzustand befindet und der Keller leer ist. *Hinweis: Beim Lesen des Kellerzeichens wird dieses aus dem Keller gelöscht. Soll es erhalten bleiben, muss es wieder in den Keller geschrieben werden.*	

6.2 Formale Sprachen und formale Grammatiken

Formale Notation	Grafische Notation anhand eines Beispiels
Grammatik G = (N, T, P, S)	
N Menge der Nichtterminale (nichtleer, endlich) T Menge der Terminale (nichtleer, endlich) P Menge der Produktionsregeln S Startsymbol S ∈ N	Syntaxdiagramm ↗ Seite 13 G = ({S, B}, {0,1}, {S → 0 B, B → 0 B | 1}, S): S:  B: 
Die formale Sprache L(G) ist die Menge aller von einer Grammatik G erzeugten Worte.	L(G) = {0^n1 mit n > 0}. Ableitungsbaum für das Wort w = 0001 ∈ L(G) 

Chomsky-Hierarchie			
Typ	**Grammatik**	**Produktionsregeln**	**Hinweis**
0	allgemein	α → β, α ≠ ε	A, B ..... Variable α, β, γ .... Terminal- und Nichtterminalfolgen ε..... leeres Terminal a ..... Terminal | ..... Verknüpfung „oder“
1	kontextsensitiv	α A β → α γ β, γ ≠ ε	
2	kontextfrei	A → α	
3	regulär	entweder A → ε | a | a B oder A → ε | a | B a	

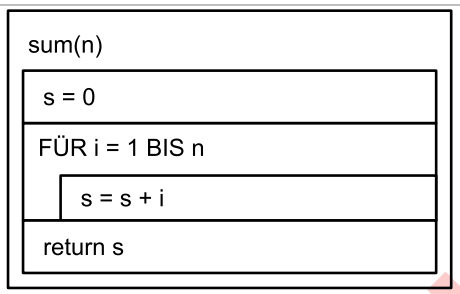
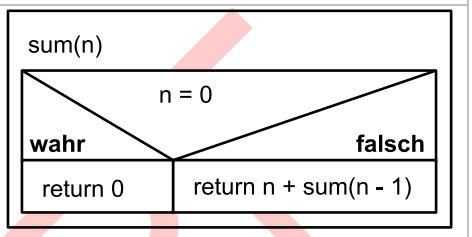
Zusammenhang zwischen der Chomsky-Hierarchie und erkennenden Automaten		
Typ	**Grammatik**	**Erkennende Automaten**
0	allgemein	Turingmaschine
1	kontextsensitiv	Turingmaschine
2	kontextfrei	Nichtdeterministischer Kellerautomat, Turingmaschine
3	regulär	Akzeptor, Kellerautomat, Turingmaschine

7 Softwareentwicklung

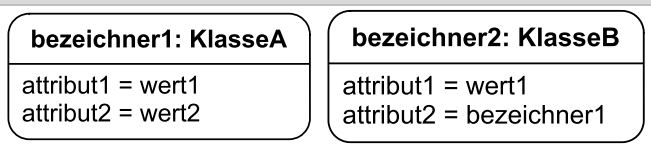
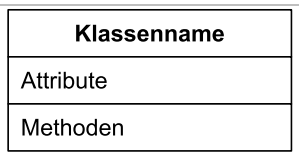
7.1 Algorithmen

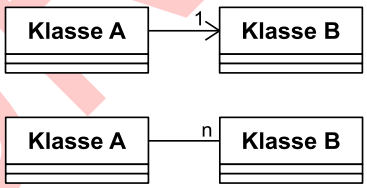
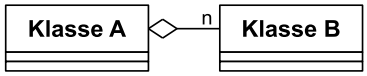
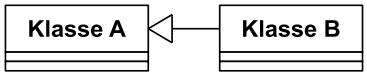
<table>
<tr><td>Beschreibung</td><td>Ein Algorithmus ist eine eindeutige Handlungsvorschrift zur Lösung einer Klasse von Problemen. Algorithmen bestehen aus einer endlichen und wohldefinierten Abfolge von Anweisungen, die systematisch ausgeführt werden. Durch den Algorithmus werden Eingabewerte schrittweise verarbeitet und in Ausgabewerte umgewandelt.
Nach der Church-Turing-These ist die Menge der algorithmisch berechenbaren Funktionen identisch mit der Menge der von Turingmaschinen berechenbaren Funktionen.</td></tr>
<tr><td>Parameter</td><td>Parameter ermöglichen flexible Eingangswerte.</td></tr>
<tr><td>Variable</td><td>Eine Variable ermöglicht das Speichern eines Datums (Singular von Daten) während der Abarbeitung des Algorithmus.</td></tr>
<tr><td>Eigenschaften</td><td>
<table>
<tr><td>allgemeingültig</td><td>Ein Algorithmus gilt für eine Klasse gleichartiger Probleme. Die Spezifizierung erfolgt mithilfe von Parametern.</td></tr>
<tr><td>ausführbar</td><td>Alle Anweisungen sind verständlich formuliert und ausführbar.</td></tr>
<tr><td>endlich</td><td>Die Beschreibung erfolgt in einem endlich langen Text.</td></tr>
<tr><td>eindeutig</td><td>Mit jeder Anweisung ist auch die nächstfolgende festgelegt. Gleiche Eingabewerte werden bei wiederholter Abarbeitung auf dieselben Ausgangswerte abgebildet.</td></tr>
<tr><td>terminiert</td><td>Nach endlich vielen Schritten ist eine Lösung erreicht.</td></tr>
</table>
</td></tr>
<tr><td>Darstellungen</td><td>Struktogramm ..... ↗ Seite 12
Programmablaufplan ..... ↗ Seite 9
Zustandsübergang Turingmaschine ..... ↗ Seite 25
Pseudocode ..... ↗ Seite 12
Quelltext ..... ↗ Seite 30</td></tr>
<tr><td>Zeitkomplexität</td><td>Beschreibung des Laufzeitverhaltens eines Algorithmus in Abhängigkeit von der Eingabegröße</td></tr>
<tr><td>O-Notation</td><td>obere Schranke der Zeitkomplexität

Beispiele für vergleichsbasierte Sortierverfahren:
<table>
<tr><th>Sortierverfahren</th><th>Best Case</th><th>Worst Case</th></tr>
<tr><td>Bubble Sort</td><td>O(n)</td><td>O(n^2)</td></tr>
<tr><td>Insertion Sort</td><td>O(n)</td><td>O(n^2)</td></tr>
<tr><td>Merge Sort</td><td>O(n log n)</td><td>O(n log n)</td></tr>
</table>
</td></tr>
</table>

Iterative und rekursive Algorithmen		
	Iteration	**Rekursion**
Beschreibung	Wiederholung einer Sequenz entsprechend einer Bedingung	Aufruf einer parameterbehafteten Funktion aus sich selbst heraus, bis eine Abbruchbedingung erfüllt ist. Zerlegung des Problems in Teilprobleme (Teile und Herrsche)
Beispiel	Algorithmus zur Berechnung der Summe der natürlichen Zahl bis n	
Formal	`sum(n) = 0 + 1 + 2 + … + n`	`sum(0) = 0` `sum(n) = n + sum(n-1)`
Struktogramm		

7.2 Objektorientierte Modellierung

Element	Symbol
Objekt	
Klasse	 *Hinweis: Vor Attributen und Methoden wird oft ihre Sichtbarkeit angegeben.*
Sichtbarkeit von Attribut und Methode	+..... öffentlich, public #..... geschützt, protected –..... privat, private

Beziehungen	Beschreibung	Symbol
Assoziation	Gerichtet: Ein Objekt der Klasse A steht mit mindestens einem Objekt der Klasse B in Beziehung. Ungerichtet: Die Beziehung besteht wechselseitig.	
Aggregation	Ein Objekt der Klasse A beinhaltet mindestens ein Objekt der Klasse B.	
Multiplizität bei Assoziation bzw. Aggregation	maximale Anzahl von Objekten einer Klasse, die mit einem Objekt der anderen Klasse in Beziehung stehen *Hinweis:* *Statt des Begriffs Multiplizität wird teilweise auch der Begriff Kardinalität verwendet.*	1 .....genau ein Objekt * .....beliebig viele Objekte n .....beliebig viele Objekte
Vererbung	Ein Unterklasse B ist eine Spezialisierung der Oberklasse A. B enthält von A alle vererbbaren Attribute und Methoden. B kann eigene Attribute und Methoden ergänzen oder vererbte Methoden überschreiben. Vererbbar sind Attribute und Methoden mit der Sichtbarkeit *öffentlich* oder *geschützt*.	

8 Programmierhilfen für Java und Python

8.1 Algorithmische Strukturen

Struktogramm	Java	Python
Anweisung	`Anweisung;`	`Anweisung`

Anweisung 1
Anweisung 2
...
Anweisung N
	`Anweisung 1;`
	`Anweisung 2;`
	`…`
	`Anweisung N;`
		`Anweisung 1`
		`Anweisung 2`
		`…`
		`Anweisung N`

Bedingung
wahr / **falsch**
Sequenz A
	`if (Bedingung) {`
	`  Sequenz`
	`}`
		`if Bedingung:`
		`  Sequenz`

Bedingung
wahr / **falsch**
Sequenz A / *Sequenz B*
	`if (Bedingung) {`
	`  Sequenz A`
	`}`
	`else {`
	`  Sequenz B`
	`}`
		`if Bedingung:`
		`  Sequenz A`
		`else:`
		`  Sequenz B`

Selektor
1 / **2** / **...** / **sonst**
Seq A / *Seq B* / ... / *Seq N*
	`switch (Selektor) {`
	`  case 1: Seq A`
	`  break;`
	`  case 2: Seq B`
	`  break;`
	`  …`
	`  default: Seq N;`
	`}`
		`if Selektor == 1:`
		`  Seq A`
		`elif Selektor == 2:`
		`  Seq B`
		`  …`
		`else:`
		`  Seq N`

SOLANGE *Bedingung*
Sequenz
	`while (Bedingung) {`
	`  Sequenz`
	`}`
		`while Bedingung:`
		`  Sequenz`

WIEDERHOLE *n* MAL
Sequenz
	`for (int i=1;i<=n;i++) {`
	`  Sequenz`
	`}`
		`for i in range(n):`
		`  Sequenz`

FÜR *i* = *a* BIS *e* SCHRITT *s*
Sequenz
	`for (int i=a;i<=e;i+=s) {`
	`  Sequenz`
	`}`
		`for i in range(a,e+1,s):`
		`  Sequenz`

FÜR *i* = *a* HINAB BIS *e* SCHRITT *s*
Sequenz
	`for (int i=a;i>=e;i-=s) {`
	`  Sequenz`
	`}`
		`for i in range(a,e-1,-s):`
		`  Sequenz`

FÜR JEDES *element* AUS *liste*
Sequenz
	`for (typ element: liste) {`
	`  Sequenz`
	`}`
		`for element in liste:`
		`  Sequenz`

8.2 Kommentare

Art	Java	Python
Einzeiliger Kommentar	`// Bemerkung`	`# Bemerkung`
Mehrzeiliger Kommentar	`/* Bemerkung über` `mehrere Zeilen */`	`""" Bemerkung über` `mehrere Zeilen """`

Dokumentation einer Klasse	`/** Name/Beschreibung der Klasse`
	`* @author Autor`
	`* @version Version`
	`*/`
		`""" Name/Beschreibung der Klasse`
		`mehrere Zeilen """`

Dokumentation einer Methode	`/** Name/Beschreibung der Methode`
	`* @param Name Beschreibung`
	`* @return Rückgabebeschreibung`
	`*/`
		`""" Name/Beschreibung der Klasse`
		`Parameters: Beschreibung`
		`Returns: Beschreibung`
		`"""`

8.3 Datentypen und Variablen

Variable	Java	Python
Typisierung	statisch	dynamisch
Deklaration	*`typ variablenname`*	
Deklaration mit Initialisierung	*`typ variablenname`* `= wert`	*`variablenname`* `= wert`
Gültigkeit	innerhalb der Struktur, in der sie deklariert wurde	innerhalb der Funktion/Methode oder der Klasse, in der sie deklariert wurde

Datentyp	Java			Python	
	Art	Bezeichner	Wrapperklasse	Art	Bezeichner
Ganze Zahl	primitiver Typ	`int`	`Integer`	Objekt	`int`
Gleitkommazahl	primitiver Typ	`double`	`Double`	Objekt	`float`
Wahrheitswert	primitiver Typ	`boolean`	`Boolean`	Objekt	`bool`
Zeichen	primitiver Typ	`char`	`Character`		
Zeichenkette	Objekttyp	`String`		Objekt	`str`
Liste	Objekttyp	`ArrayList`		Objekt	`list`

Eingabe ganzer Zahlen in verschiedenen Zahlsystemen			
Zahlensystem	**Basis**	**Präfix**	**Beispiel**
Dezimalsystem	10		`a = 26`
Binärsystem	2	`0b`	`a = 0b00011010`
Hexadezimalsystem	16	`0x`	`a = 0x1A`

Ausgabe ganzer Zahlen in verschiedenen Zahlsystemen als Zeichenkette mit Präfix		
Zahlsystem	**Java**	**Python**
Dezimalsystem	`Integer.toString(a)`	`str(a)`
Binärsystem	`"0b" + Integer.toHexString(a)`	`bin(a)`
Hexadezimalsystem	`"0x" + Integer.toBinaryString(a)`	`hex(a)`

8.4 Datentypenumwandlung

Aufgabe	Java	Python
Casting: Umwandlung der Variablen a in … *Hinweis: nur falls möglich und dann in der Regel mit Informationsverlust*		
eine ganze Zahl	`(int) a`	`int(a)`
eine Gleitkommazahl	`(double) a`	`float(a)`
ein Zeichen	`(char) a`	
eine Zeichenkette	`String.valueOf(a)`	`str(a)`
ein Objekt der Klasse K	`(K) a`	

Aufgabe	Java	Python
Parsing: Umwandeln einer Zeichenkette z in …		
eine ganze Zahl	`Integer.parseInt(z)`	`int(z)`
eine Gleitkommazahl	`Double.parseDouble(z)`	`float(z)`
einen Wahrheitswert	`Boolean.parseBoolean(z)`	`bool(z)`
ein Zeichen	`z.charAt(0)`	

Aufgabe	Java	Python
Bestimmung des Datentyps/der Klasse des Objekts o		
	`o.getClass()`	`type(o)`
	`o instanceof KLASSE` Für `KLASSE` muss der Name der Klasse eingetragen werden. Das Ergebnis ist ein Wahrheitswert.	

8.5 Ein- und Ausgaben

	Java	Python
Voraussetzung	vor der Klassendefinition bei Verwendung von Dialogfenster: `import javax.swing.*`	
Eingabe einer Zeichenkette in Variable a mit Ausgabe eines Hinweistexts z		
Dialogfenster	`a = JOptionPane.showInputDialog(null, z)`	
Konsole		`a = input(z)`
Ausgabe einer Zeichenkette z		
Dialogfenster	`JOptionPane.showMessageDialog(null, z)`	
Konsole	`System.out.println(z)` `System.out.print(z)`	`print(z)` `print(z, end="")`
Ausgabe einer Liste		
Besonderheit	Bei der Ausgabe einer Liste werden die Elemente kommasepariert und geklammert als Zeichenkette dargestellt.	

8.6 Operatoren

Beschreibung	Java	Python
Zuweisungsoperatoren		
Zuweisung	`=`	`=`
Erhöhung um	`+=`	`+=`
Verminderung um	`-=`	`-=`
Vervielfachung um	`*=`	`*=`
Inkrement (Erhöhung um 1)	*var*`++`	
Dekrement (Verminderung um 1)	*var*`--`	
Arithmetische Operatoren		
Addition/Vorzeichen	`+`	`+`
Subtraktion/Vorzeichen	`-`	`-`
Multiplikation	`*`	`*`
Division	`/`	`/`
Potenzieren	`Math.pow(b,e)`	`**`
ganzzahliger Teil bei Division zweier ganzer Zahlen	`/`	`//`
ganzzahliger Rest bei Division zweier ganzer Zahlen (Modulo)	`%`	`%`
Vergleichsoperatoren		
identisch	`==`	`==`
ungleich	`!=`	`!=`
kleiner als	`<`	`<`
größer als	`>`	`>`
kleiner gleich	`<=`	`<=`
größer gleich	`>=`	`>=`
Logische Operatoren		
logisch NICHT	`!`	`not`
logisch UND	`&&`	`and`
logisch ODER	`||`	`or`
logisch EXKLUSIV ODER	`^`	`^`
Bitorientierte Operatoren		
bitweise NICHT	`~`	`~`
bitweises UND	`&`	`&`
bitweises ODER	`|`	`|`
bitweises EXKLUSIV ODER	`^`	`^`
bitweises Linksschieben	`<<`	`<<`
bitweises Rechtsschieben	`>>`	`>>`

8.7 Mathematische Konstanten, Funktionen und Zufallszahlen

	Java	Python

Voraussetzung		vor Klassendefinition:
		`import math`
		`import random`

Konstanten		
π	`Math.PI`	`math.pi`
e	`Math.E`	`math.e`
Funktionen		
|x|	`Math.abs(x)`	`abs(x)`
$\sqrt{x}$	`Math.sqrt(x)`	`math.sqrt(x)`
x^a	`Math.pow(x,a)`	`math.pow(x,a)`
e^x	`Math.exp(x)`	`math.exp(x)`
x!		`math.factorial(x)`
ln(x)	`Math.log(x)`	`math.log(x)`
sin(x)	`Math.sin(x)`	`math.sin(x)`
cos(x)	`Math.cos(x)`	`math.cos(x)`
tan(x)	`Math.tan(x)`	`math.tan(x)`
arc sin(x)	`Math.asin(x)`	`math.asin(x)`
arc cos(x)	`Math.acos(x)`	`math.acos(x)`
arc tan(x)	`Math.atan(x)`	`math.atan(x)`
ggT(x,y)		`math.gcd(x,y)`
Umrechnung		
Bogenmaß in Gradmaß	`Math.toDegree(x)`	`math.degrees(x)`
Gradmaß in Bogenmaß	`Math.toRadians(x)`	`math.radians(x)`
Runden		
kaufmännisch[1]	`Math.round(x)`	`round(x)` `round(x,d)`
aufrunden	`Math.ceil(x)`	`math.ceil(x)`
abrunden	`Math.floor(x)`	`math.floor(x)`
Zufallszahlen		
Gleitkommazahl aus [0, 1)	`Math.random();`	`random.random()`
Ganze Zahl aus [a, b]	`(int) ((b-a+1)*Math.random()+a)`	`random.randint(a,b)`

[1] „Kaufmännisches Runden“ ist in den beiden Programmiersprachen nicht einheitlich umgesetzt.

8.8 Zeichenketten

Beschreibung	Java	Python
Besonderheiten		Hochkomma statt Anführungszeichen möglich
Zeichenkette z erzeugen und initialisieren		
leere Zeichenkette	`z = ""`	`z = ""`
Zeichenkette mit vordefiniertem Wert	`z = "Hallo Welt"`	`z = "Hallo Welt"`
Umwandeln von Zeichenketten		
Rückgabe: z in Kleinbuchstaben	`z.toLowerCase()`	`z.lower()`
Rückgabe: z in Großbuchstaben	`z.toUpperCase()`	`z.upper()`
Rückgabe: z mit Ersetzung von z1 durch z2	`z.replaceAll(z1, z2)`	`z.replace(z1, z2)`
Zugriff auf Zeichen in der Zeichenkette z		
Rückgabe: z an Stelle i	`z.substring(i, i+1)`	`z[i]`
Rückgabe: z vom Typ `char/int` an Stelle i	`z.charAt(i)`	
Rückgabe: z von Stelle m bis n	`z.substring(m, n+1)`	`z[m:n+1]`
Zeichenkette z prüfen		
nur WAHR, falls z1 identisch mit z2	`z1.equals(z2)`	`z1 == z2`
nur WAHR, falls z1 in z enthalten	`z.contains(z1)`	`z1 in z`
nur WAHR, falls z mit z1 beginnt	`z.startsWith(z1)`	`z.startswith(z1)`
nur WAHR, falls z mit z1 endet	`z.endsWith(z1)`	`z.endswith(z1)`
Rückgabe: Stelle des ersten Auftretens von z1 in z, sonst -1	`z.indexOf(z1)`	`z.find(z1)`
Rückgabe: Stelle des letzten Auftretens von z1 in z, sonst -1	`z.lastIndexOf(z1)`	`z.rfind(z1)`
nur WAHR, falls z1 lexikographisch kleiner als z2		`z1 < z2`
< 0, falls z lexikographisch kleiner als z1 = 0, falls z lexikographisch gleich z1 > 0, falls z lexikographisch größer als z1	`z.compareTo(z1)`	
sonstiges		
Zeichenketten z1 und z2 verknüpfen	`z1 + z2`	`z1 + z2`
Rückgabe: Länge von z	`z.length()`	`len(z)`

Rückgabe: Liste mit den durch z1 getrennten Teilzeichenketten aus z	`ArrayList<String> li`
	`li = new ArrayList<>(`
	`List.of(z.split(z1)))`
		`li = z.split(z1)`

8.9 Listen

Beschreibung	Java	Python
Besonderheit	`ArrayList<typ>` Elemente gleichen Objektyps	Elemente unterschiedlichen Typs
Voraussetzung	`import java.util.*;`	
Listen primitiven Typs	über Wrapper-Klasse, z. B.: `ArrayList<Integer> li = …`	
Liste deklarieren und initialisieren		
`li` deklarieren	`ArrayList<typ> li`	
`li` als leere Liste initialisieren	`li = new ArrayList<>()`	`li = []`
`li` mit Objekten initialisieren	`li = new ArrayList<>(` `  List.of(o1,o2,…))`	`li = [o1,o2,…]`

Beispiel Zeichenkettenliste	`ArrayList<String> li;`
	`li = new ArrayList<>();`
	`li.add("Ha");`
	`li.add("se");`
		`li = ["Ha", "se"]`

Beispiel Ganzzahlenliste	`ArrayList<Integer> li;`
	`li = new`
	`ArrayList<>(List.of(1,3));`
		`li = [1,3]`

Hinzufügen/Austauschen von Elementen		
Objekt `obj` an `li` hinten anfügen	`li.add(obj)`	`li.append(obj)`
Objekt `obj` an `li` vorn anfügen	`li.add(0, obj)`	`li.insert(obj)`
Objekt `obj` an Stelle `i` einfügen	`li.add(i, obj)`	`li.insert(i,obj)`
Objekt `obj` an Stelle `i` ersetzen	`li.set(i, obj)`	`li[i] = obj`
Entfernen von Elementen		
Element an Stelle `i` entfernen	`li.remove(i)`	`del li[i]` `li.pop(i)`
erstes Auftreten von `obj` entfernen	`li.remove(obj)`	`li.remove(obj)`
Liste leeren	`li.clear()`	`li.clear()`
Zugriff auf Elemente und Listeneigenschaften		
Element an Stelle `i` zurückgeben	`li.get(i)`	`li[i]`
Listenlänge zurückgeben	`li.size()`	`len(li)`
Listen und Elemente prüfen		
nur `WAHR`, falls Liste leer	`li.isEmpty()`	`len(li) == 0`
nur `WAHR`, falls beide Listen identisch	`li1.equals(li2)`	`li1 == li2`
nur `WAHR`, falls `obj` in Liste enthalten	`li.contains(obj)`	`obj in li`
Stelle des ersten Auftretens von `obj` falls `obj` in Liste enthalten, sonst `-1`	`li.indexOf(obj)`	`li.index(obj)`
Ändern der Reihenfolge		
Liste aufsteigend sortieren	`Collections.sort(li)`	`li.sort()`
Liste umkehren	`Collections.reverse(li)`	`li.reverse()`
Liste anfügen/zerlegen/kopieren		
`li2` an `li1` anfügen	`li1.addAll(li2)`	`li1 = li1 + li2`
neue Liste mit Elementen von `li` von Stelle `m` bis `n` zurückgeben	`li = new ArrayList<>(` `  li.subList(m,n+1))`	`li[m:n+1]`
Kopie der Liste mit referenzierten Elementen erstellen	`new ArrayList<>(li)`	`li.copy()`

8.10 Objektorientierte Programmierung

	Java	Python
Einfache Klasse		

Definition	`public class Klasse {`
	`  //Attribute`
	`  //Konstruktor`
	`  //Methoden`
	`}`
		`class Klasse:`
		`  #Konstruktor`
		`  #Methoden`

Attribut	`Sichtbarkeit Typ Attribut;`	Definition/Initialisierung im Konstruktor

Konstruktor	`public Klasse() {`
	`  …`
	`}`
		`def __init__(self):`
		`  self.SichtbarkeitAttribut = Wert`

Methode ohne Rückgabe	`Sichtbarkeit void Methode() {`
	`  …`
	`}`
		`def SichtbarkeitMethode(self):`
		`  …`

Methode mit Rückgabe	`Sichtbarkeit Typ Methode() {`
	`  …`
	`  return …;`
	`}`
		`def SichtbarkeitMethode(self):`
		`  …`
		`  return …`

Unterklasse		

Definition	`public class Unter extends Ober {`
	`  …`
	`}`
		`class Unter(Ober):`
		`  …`

Konstruktor	`public Unter(){`
	`  super();`
	`  …`
	`}`
		`def __init__(self):`
		`  super().__init__()`
		`  …`

Elemente einer Klasse		
Element	**Java**	**Python**
Sichtbarkeit		
öffentlich	`public`	ohne Zeichen
geschützt	`protected`	_ einzelner Unterstrich
privat	`private`	__ doppelter Unterstrich
Parameter-deklaration	`Typ pParameter`	`pParameter`
Instanziierung	`objektname = new Klasse();`	`objektname = Klasse()`
Methodenaufruf		
eigene Klasse	`this.methode();`	`self.methode()`
Oberklasse	`super.methode();`	`super().methode()`
anderes Objekt	`objektname.methode();`	`objektname.methode()`
Zugriff auf Attribute/Methoden der …		

eigenen Klasse	`this.attribut`
	`this.methode()`
	`this` kann bei Eindeutigkeit entfallen
		`self.attribut`
		`self.methode()`

Oberklasse	`super.attribut;` `super.methode();`	`super().attribut` `super().methode()`

9 Aufgabenoperatoren im Fach Informatik

Ein Operator impliziert eine Aufforderung. Er bestimmt die zu wählenden Mittel und Methoden für eine überprüfbare Bearbeitung der Aufgabenstellung.

Die nachfolgende Übersicht zeigt eine Auswahl typischer Operatoren.

Operator	Beschreibung
analysieren	eine Materialgrundlage untersuchen, Elemente identifizieren, Beziehungen zwischen den Elementen erfassen, das Untersuchungsergebnis darstellen
angeben	ohne Erläuterung, Begründung oder Lösungsweg nennen
begründen	durch rational nachvollziehbare Argumente basierend auf Regeln, Prinzipien oder kausale Zusammenhänge stützen
beschreiben	unter Verwendung der Fachsprache in eigenen Worten verständlich wiedergeben
bestimmen ermitteln	Ergebnisse unter Angabe von Annahmen, Regeln oder kausalen Zusammenhänge gewinnen
beurteilen bewerten	unter Verwendung von Fachwissen und Fachmethoden ein eigenständiges Sach- bzw. Werturteil formulieren und begründen
darstellen	in formalisierter Form grafisch oder fachsprachlich wiedergeben
entwickeln	unter Verwendung von Fachwissen und Fachmethoden nachvollziehbar herstellen oder konstruieren
erklären	einen Sachverhalt erfassen, in einen kausalen Zusammenhang einordnen und deuten
erläutern	mithilfe zusätzlicher Angaben (Beispiele, Vergleiche, …) verständlich machen
ermitteln	Ergebnisse i. d. R. durch Rechenoperationen nachvollziehbar gewinnen
implementieren	ein informatisches Modell auf einem oder für ein Informatiksystem umsetzen
interpretieren	einen Sachverhalt erfassen und deuten
modellieren	gemäß einer Problemanalyse ein informatisches Modell anfertigen
modifizieren	unter Berücksichtigung konkreter Vorgaben anpassen und ggf. erweitern
protokollieren dokumentieren	schrittweise untersuchen und dabei in formalisierter Form fachsprachlich wiedergeben
Stellung nehmen	die eigene Meinung zu einem Problem argumentativ entwickeln und darlegen
überführen	in ein anderes informatisches Modell oder eine andere Darstellungsform bringen
vergleichen	kriteriengeleitet Gemeinsamkeiten und Unterschiede angeben

Index

T

U

V

W

X

Z