FirstName LastName, Second Author

Programmieren mit Java

Begleitunterlagen

Zu dem Online Kurs

E.Tutorial
www.et.ethz.ch

teachIT

Programmieren mit Java

Begleitunterlagen

Zu dem Online Kurs

FirstName LastName, Second Author

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

Trotz sorgfältiger Arbeit schleichen sich manchmal Fehler ein. Die Autoren sind Ihnen für Anregungen und Hinweise per Email an et@ethz.ch dankbar!

Herstellung und Verlag: BoD – Books on Demand, Norderstedt

ISBN 978-3-527-70594-8

Inhaltsverzeichnis

Wie soll dieses Buch verwendet werden?

```
For x = 1 To 3
  For y = 1 To 3
    Cells(x,y)
  Next y
Next x
```

Einen Zellbereich bearbeiten

Möchte man den Zellbereich A1 bis C3 bearbeiten, können zwei ineinander verschachtelte Schleifen eingesetzt werden:

```
For x = 1 To 3
  For y = 1 To 3
    Cells(x,y)
  Next y
Next x
```

`Cells` enthält als Zeilen- und Spaltenwerte die beiden Laufvariablen. Dadurch werden sämtliche Kombinationen erzeugt:

`Cells(1,1)`, `Cells(1,2)`, ..., `Cells(3,3)`.

Das vorliegende Buch enthält alle Begleitunterlagen zum Onlinekurs **Programmiergrundlagen mit Python und Matlab**. Für den Kurs können Sie sich über eine der folgenden beiden Lehrveranstaltungen registrieren und einschreiben:

- **Grundlagen der Informatik** (252-0852-00L), *www.gdi.ethz.ch*
- **Einsatz von Informatikmitteln** (252-0839-00L), *www.evim.ethz.ch*

Der Kurs besteht aus folgenden **4 Modulen**:

Jedes Modul dauert abhängig von Ihrem Vorwissen 4 bis 8 Arbeitsstunden. Die Materialien in diesem Buch und auf der Webseite begleiten Sie von der Einführung der Begriffe und Konzepte, über deren Verwendung in einfachen Programmier-Beispielen bis

	Titel	Sprache
1	Variablen und Datentypen	Python
2	Kontrollstrukturen und Logik	Python
3	Arrays, Simulieren und Modellieren	Python
4	Matrizenrechnen	Matlab

hin zur selbstständigen Anwendung der Programmierkonzepte in kleinen Programmier-Projekten.

Jedes Modul ist wie folgt organisiert:

1. **SEE**: In diesem Buch und in der Vorlesung erhalten Sie eine Einführung in die wichtigsten Begriffe und Konzepte der Programmierung.
2. **TRY**: In diesem Schritt wenden Sie die Konzepte erstmals anhand überschaubarer Programmierbeispielen an. Angeleitet werden Sie dabei von einem elektronischen Tutorials (E.Tutorial).
3. **DO**: In diesem Schritt setzten Sie selbstständig kleinere Programmier-Projekte um.
4. **EXPLAIN**: In diesem abschlissenden Schritt diskutieren Sie mit einer Assistenzperson im Computerraum über die Programmier-Projekte aus Phase 3. Dabei versuchen Sie, die neuen Konzepte der Phase 1 anhand Ihrer Lösung aufzuzeigen.

Dieses Buch enthält alle Begleitmaterialien für die Phasen 1 und 3.

Danksagung:

Wir danken folgenden Personen:

Prof. Dr. Hans Hinterberger und Dr. Barbara Scheuner für das Bereitstellen von Unterlagen und Aufgabenstellungen aus früheren Kursen.

Prof. Dr. Juraj Hromkovic für die finanzielle Unterstützung.

Dr. Hans Joachim Böckenhauer für das Korrekturlesen.

E.Tutorial
www.et.ethz.ch

teachIT

A Capitalized Title

Module o

Theorieteil

Autoren:

FirstName LastName, Second Author

Begriffe

keywords Java

not capitalized

Theorieteil

0.1 Modulübersicht Test 2

Die beiden Konzepte Variablen und Datentypen sind für jede Programmierung grundlegend. Bei **Variablen** handelt es sich um Speicherbereiche, in denen Werte gespeichert werden können und der **Datentyp** gibt an, welche Werte erlaubt sind (z.B. nur Ganzzahlen). In einem Programm werden Daten verarbeitet, die sich in ihrer Art unterscheiden, z.B. Zeichen, Zahlen oder logische Daten. Digitale Daten werden immer durch Ziffern dargestellt. Daher auch der Name, *digit* bedeutet Ziffer.

0.2 Darstellung von Zahlen und Zeichen im Computer

Um die Darstellung von Zeichen, Zahlen und Texten im Computer zu verstehen, muss man das **binäre System** verstehen.

0.2.1 Binäres System

Alle Rechner stellen Information im binären System dar. Dieses kennt nur zwei Ziffern, nämlich 0 und 1 (im Gegensatz zum Dezimalsystem mit den Ziffern 0 bis 9). Eine solche Ziffer wird als **Bit** bezeichnet (Abkürzung für *Binary Digit*, übersetzt „Binäre Ziffer“). Ein Bit stellt den kleinsten speicherbaren Wert in einem Computer dar. Jeweils 8 Bits werden zu einem **Byte** zusammengefasst. Ein Byte kann somit $2^8 = 256$ verschiedene Sequenzen von je 8 Bit speichern.

0.2.2 Darstellung von Zahlen im binären System

Betrachten wir die Zahl 91, die binär mit 8 Bit als 01011011 dargestellt wird (siehe Tabelle 0.1). Wir reden deswegen in diesem Zusammenhang von der **Binärdarstellung** von 91 (und nicht von der Dezimaldarstellung, die für uns lesefreundlicher ist).

Bit	8	7	6	5	4	3	3	1	
Binärwert	0	1	0	1	1	0	1	1	
Wertigkeit	$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	
Dezimalwert	0	64	0	16	8	0	2	1	= 91

Tabelle 0.1: Binäre Darstellung der Dezimalzahl 91. Details siehe Text.

Eine 8-Bit-Zahl, wie in unserem Beispiel, kann Werte zwischen 00000000 (0 im Dezimalsystem) und 11111111 (255 im Dezimalsystem) speichern. Für die Umrechnung vom Binär- in den Dezimalwert multiplizieren wir für jedes Bit den Binärwert mit der Wertigkeit des Bits und summieren diese auf. Im binären System können wir mit 8 Bit nur die ganzen Zahlen 0 bis 255 darstellen. Ist die Zahl, die wir darstellen wollen, grösser, muss auch ein grösserer Speicherbereich bereitgestellt werden.

0.2.3 Darstellung von Zeichen im binären System

Für die Darstellung von Zeichen im Computer wurde der so genannte **ASCII-Code** entwickelt. ASCII steht für *American Standard Code for Information Interchange*, was übersetzt so viel heisst wie Amerikanische Standardcodierung für den Datenaustausch. Mit Hilfe des 7-Bit-ASCII-Codes können 128 verschiedene Zeichen (2^7) dargestellt werden oder umgekehrt wird jedem Zeichen ein Bitmuster aus 7 Bit zugeordnet (siehe Tabelle 0.2). Die Zeichen entsprechen weitgehend einer Computertastatur. Der ASCII-Code wurde später auf 8 Bit erweitert, was die Darstellung von 256 Zeichen (2^8) erlaubt.

Die ASCII-Tabelle enthält auch nicht darstellbare Zeichen (wie etwa ein Zeichen, das einen Zeilenumbruch repräsentiert). Die wichtigsten sind in Tabelle 0.2.3 dargestellt:

0.3 Datentypen

Der **Datentyp** gibt an, welche Daten in einem Programm gespeichert und bearbeitet werden können. Programmiersprachen besitzen vordefinierte Datentypen, die sich in der Art der Interpretation der gespeicherten Daten und in der Grösse unterscheiden.

- Typ für Zahlenwerte
- Typ für Zeichenwerte
- Typ für Wahrheitswerte (Boolsche Werte) (siehe Modul 2)

Tabelle 0.4 gibt einen Überblick über die wichtigsten Datentypen, die in vielen Programmiersprachen vorkommen.

0-31		31-63		64-95		96-127	
Dez	Zeichen	Dez	Zeichen	Dez	Zeichen	Dez	Zeichen
0	NUL	32	SP	64	@	96	‘
1	SOH	33	!	65	A	97	a
2	STX	34	“	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	’	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Tabelle 0.2: ASCII-Tabelle

Dez	Zeichen	Bedeutung
8	BS	Backspace. Linkes Zeichen löschen
10	NL	New Line. Neue Zeile beginnen
32	SP	Space. Leerzeichen
127	DEL	Delete. Rechtes Zeichen löschen

Tabelle 0.3: Nicht darstellbare Zeichen der ASCII-Tabelle

Typ	Beschreibung	Grösse in Bit	Wertebereich
boolean	Boolscher Wert	1	true oder false
char	Zeichen	16	Unicode-Zeichen
byte	Ganzzahl	8	$-128 \ldots 127$ $(-2^7 \ldots + 2^7 - 1)$
short		16	$-32'768 \ldots 32'767$ $(-2^{15} \ldots + 2^{15} - 1)$
int		32	$-2'147'483'648 \ldots 2'147'483'647$ $(-2^{31} \ldots + 2^{31} - 1)$
long		64	$-9'223'372'036'854'775'808 \ldots$ $9'223'372'036'854'775'807$ $(-2^{63} \ldots + 2^{63} - 1)$
float	Gleitkommazahl	32	$+/- 3.40282347 \times 10^{38}$
double		64	$+/- 1.79769313486231569 \times 10^{308}$

Tabelle 0.4: Die wichtigsten Datentypen in Java

0.4 Variablen und Konstanten

Variablen einer Programmiersprache sind denen der Mathematik ähnlich. Variablen haben einen Namen über den sie angesprochen werden, und dienen dazu, einen Wert im Bereich eines bestimmten Datentyps zu speichern. Der Wert der Variable kann sich im Verlaufe des Programms ändern (er kann variieren, daher der Name). Um eine Variable in einem Programm verwenden zu können, sind folgende Operationen notwendig:

1. Deklaration
2. Initialisierung

0.4.1 Deklaration

Bevor eine Variable in einem Programm verwendet werden kann, muss sie **deklariert** werden. Das heisst, dass Sie als Programmiererin oder Programmierer einen Speicherbereich für einen bestimmten Datentyp belegen und diesem Speicherplatz einen Namen geben. Über diesen Namen kann der Speicherbereich während des Programmablaufs aufgerufen werden. Namen von Variablen beginnen in Java gemäss Konvention jeweils mit einem Kleinbuchstaben, sie dürfen keine Leerzeichen enthalten und sollten möglichst aussagekräftig sein.

Schreibweise:

```
Datentyp name;
```

Beispiel:

```
// Variable a vom Typ Integer
int a;

// Variable b vom Typ Double
double b;

// Variable c vom Typ Character
char c;
```

Mehrere Variablen vom gleichen Typ können auch wie in folgendem Beispiel in einer Deklaration geschrieben werden:

```
// 3 Variablem vom Typ Integer
int meineZahl1, meineZahl2, meineZahl3;
```

0.4.2 Initialisierung und Wertzuweisung

Das **Zuweisen** von Werten geschieht mit dem Zuweisungsoperator. In Java wird hierfür ein **Gleichheitszeichen** (=) verwendet. Dabei wird der Wert des Ausdrucks rechts des Zuweisungsoperators der Variablen auf der linken Seite zugewiesen. Wenn einer Variable das erste Mal ein Wert zugewiesen wird, spricht man von einer **Initialisierung**.

Schreibweise:

```
variable = wert;
```

Beispiel:

```
meineZahl = 4;
// meineZahl hat den Wert 4
```

Der Wert einer Variablen kann sich im Verlaufe eines Programms ändern. In folgendem Beispiel wird in der Variablen `meineZahl` zuerst der Wert 4 gespeichert, der dann in einer weiteren Zeile mit dem Wert 6 überchrieben wird.

```
meineZahl = 4;
// Wert von meineZahl ist 4

meineZahl = 6;
// Wert von meineZahl ist 6
```

Bei einer Zuweisung handelt es sich also immer um einen schreibenden Zugriff auf eine Variable mit dem Resultat, dass sich deren Wert ändern kann. Der alte Wert wird überschrieben. Damit einer Variablen ein Wert zugewiesen werden kann, darf die Variable nicht als Konstante definiert sein (siehe nächster Abschnitt) und der Typ der Variablen muss mit dem Typ des Werts kompatibel sein. Auf jeden Fall kompatibel sind Variablen und Werte desselben Datentypes. Wenn die Datentypen nicht übereinstimmen, nimmt Java eine implizite **Typkonvertierung** vor. Dies ist jedoch eine häufige Fehlerquelle und sollte daher vermieden werden. Bei der impliziten Typkonvertierung in Java werden nur Typkonvertierung durchgeführt, wenn sie ohne Informationsverlust erfolgen kann, also wenn der Zieldatentyp einen gleichen oder grösseren Wertebereich hat als der Ausgangsdatentyp.

Beispiel: Ein Wert vom Typ `int` kann einer Variablen vom Typ `double` zugewiesen werden:

```
// Variable ganzeZahl vom Typ Integer
int ganzeZahl;
// Variable kommaZahl vom Typ Double
double kommaZahl;

ganzeZahl = 4;
kommaZahl = ganzeZahl;
// Variable kommaZahl wird in zum Typ Integer konvertiert
```

Möchte man eine Zuweisung machen, bei der der Zieldatentyp einen gleineren Wertebereich hat, muss eine so genannte explizite Typenkonvertierung durchgeführt werden, das sogenannte Typecasting. Die Programmiererin/der Programmierer ist dabei selber dafür verantwortlich, dass die Zuweisung möglich ist.

Beispiel:

```
// Variable d vom Typ Double
double d = 1.3;

float f = (float)d;
// Variable d muss explizit zum Typ Float konvertiert
// werden
```

Eine Variable kann in einem Programm nur in einem bestimmten Bereich des Programms gelten. Weiteres dazu erfahren Sie in einem späteren Modul.

0.4.3 Konstanten

Konstanten werden wie Variablen mit einem Namen bezeichnet. Sie enthalten während der gesamten Programmausführung einen konstanten Wert. Es kann also nach der Initialisierung keine weitere Wertzuweisung vorgenommen werden. Konstanten können jedoch Teil einer Wertzuweisung an Variablen sein. Eine Konstante wird zusätzlich zu Namen und Datentyp mit dem Schlüsselwort `final` deklariert.

Schreibweise:

```
final Datentyp name;
```

Beispiel:

```
//Deklaration und Initialisierung der Konstante k
final int k = 4;
```

0.5 Operatoren und Ausdrücke

0.5.1 Operatoren (Teil I)

Um in einem Programm Berechnungen durchzuführen zu können, stehen folgende **arithmetische Operatoren** zur Verfügung:

Operator	Ausdruck	Beschreibung	Liefert
+	a + b	Addition	Summe
-	a - b	Subtraktion	Differenz
*	a * b	Multiplikation	Produkt
/	a / b	Division	Quotient
%	a % b	Modulo	Ganzzahliger Rest einer Division

Tabelle 0.5: Arithmetische Operatoren in Java

Weitere Operatoren (logische und Vergleichsoperatoren) lernen Sie in Modul 2 kennen.

0.5.2 Ausdrücke

Ausdrücke (engl. *expressions*) sind in einer Programmiersprache Teil der kleinsten ausführbaren Einheiten eines Programms. Dabei handelt es sich um Verarbeitungsvorschriften, die sich aus **Variablen**, **Konstanten** und **Operatoren** zusammensetzen können und ein Resultat ergeben. Variablen und Konstanten, die mit einem Operator verknüpft werden, nennt man **Operanden**. Ein Ausdruck kann auch aus einer einzelnen Variablen bestehen.

Folgendes Beispiel zeigt einen Ausdruck, der aus einer Variablen `i`, einem Operator `+` und einem Operanden `5` besteht.

```
i + 5
```

Das Resultat des Ausdrucks kann wieder in einer Variablen gespeichert werden. In folgendem Beispiel wird das Resultat in der Variablen `i` gespeichert. Der vorherige Wert von `i` wird dadurch überschrieben.

```
i = i + 5;
```

Die Reihenfolge, mit der Ausdrücke bearbeitet werden, kann durch die Wahl des Operators und durch Klammern beeinflusst werden. Hierfür gelten die mathematischen Regeln, wie wir sie in der Schule gelernt haben, also „Klammern zuerst, dann Punkt vor Strich".

Beispiel:

```
5 * (2 + 10)
```

Die Klammern erzwingen, dass die Addition vor der Multiplikation ausgeführt wird.

0.5.3 Weitere Arithmetische Operatoren

Es gibt in Java noch weitere arithmetische Operatoren.

Zuweisungsoperator:

```
i += 1; \\entspricht i = i + 1;
i -= 1; \\entspricht i = i - 1;
i *= 1; \\entspricht i = i * 1;
i /= 1; \\entspricht i = i / 1;
i %= 1; \\entspricht i = i % 1;
```

Die Zuweisungsoperatoren dienen dazu die Anweisungen kompakter darzustellen, da man weniger Zeichen benötigt.

Increment und Decrement Operatoren:

```
i++; \\entspricht i = i + 1;
i--; \\entspricht i = i - 1;
```

Diese Operatoren sind meistens in sogenannten for-Schleifen anzutreffen, wo sie einen Zähler hochzählen (siehe Modul 2) .

0.6 Der Datentyp String

Der Datentyp **String** unterscheidet sich von den bisher thematisierten Datentypen insofern, dass er eine Zusammenfassung von mehreren gleichartigen Variablen darstellt. Dieser Datentyp ist auch kein primitiver Datentyp mehr, da er mehrere Elemente zusammenfasst. Er speichert nämlich alle Buchstaben einzeln in je einer char-Variablen. Wie diese Zusammenfassung der einzelnen Buchstaben funktioniert, lernen Sie, wenn es um die Objektorientierung geht. Die Deklaration und Initialisierung der Variablen funktioniert jedoch wie in 1.4.1 und 1.4.2 beschreiben.

Bei der Initialisierung von String-Variablen muss der Wert zwischen **doppelten Hochkommata** (“) angegeben werden.

Beispiel:

```
// Deklaration des Strings vorname
String vorname;

// Initialisierung mit dem Wert "Paul"
vorname = "Paul";
```

Da ein String mehrere char-Variablen enthält, kann dem String auch ein einzelner char zugewiesen werden.

Beispiel:

```
String name;
name = 'a';
```

Mehrere Strings können mit einem **Plus** (+) verbunden werden. So entsteht aus mehreren Einzelteilen ein neuer Text.

Beispiel:

```
String text;
text = "Hallo, "+ "das " + "sind " + "mehrere " + "Wörter.";
```

0.7 Bildschirm- Ein- und Ausgabe

Oft möchte man, dass die Benutzerin oder der Benutzer des Programms mit dem Programm interagieren kann. Das bedeutet, dass das Programm eine Ausgabe macht oder

dass die Benutzerin oder der Benutzer etwas eingeben kann. Um dies zu realisieren verwenden wir Funktionalitäten, welche von Java zur Verfügung gestellt werden.

0.7.1 Ausgabe

Damit die Benutzerin oder der Benutzer auch sieht, was im Programm berechnet wurde, kann im Programmcode angegeben werden, dass ein bestimmter Text oder der Wert einer Variablen ausgegeben wird.

Beispiel: Ausgabe eines vorgegebenen Texts

```
System.out.println("Das Programm hat geendet.");
```

Im obigen Beispiel wird der Text „Das Programm hat geendet.” ausgegeben. Der Text, der ausgegeben wird, steht zwischen einem Paar von Anführungs- und Schlusszeichen (“), die nicht mit ausgegeben werden. Man möchte aber nicht immer nur vorgegebenen Text ausgeben, sondern z.B. das Resultat einer Berechnung, welches in einer Variablen (z.B. `ganzeZahl`) gespeichert ist.

Beispiel: Ausgabe des Wertes einer Variablen

```
System.out.println(ganzeZahl);
```

Wenn man Variablenwerte und Text verbinden möchte, geschieht dies mit einem Additions-Zeichen (+).

Beispiel: Ausgabe von Text und Variablenwert

```
System.out.println("Das Resultat ist:" + ganzeZahl);
```

0.7.2 Eingabe

Oft möchte man den Wert einer Variablen durch die Benutzerin oder den Benutzer eines Programms bestimmen lassen. Eine einfache Möglichkeit ist die Eingabe mit der Tastatur über das Konsolenfenster.

Eine Benutzereingabe ist in Java etwas aufwändiger als bei anderen Programmiersprachen. Sie beinhaltet folgende zwei Schritte:

Paket einbinden: Mit einer Importanweisung zu Beginn unseres Java-Programms muss zunächst die Klasse Scanner des Pakets util eingebunden werden:

```
import java.util.Scanner;
```

Werte einlesen: Mit folgenden zwei Zeilen können wir Werte in Form von Zeichenketten (String) vom Konsolenfenster einlesen und einer Variablen (z.B. wert) zuweisen:

```
Scanner eingabe = new Scanner(System.in);
String wert = eingabe.next();
```

Nun weisen wir den eingelesenen Wert unserer Variablen `zahl` zu. Hierfür muss der eingelesene Text noch in einen Integer umgewandelt werden:

```
Integer.parseInt(wert);
```

Beispiel: Mit den folgenden Anweisungen übergeben wir eine Eingabezahl von der Konsole an die Variable `x` vom Typ Integer:

```
int x;
Scanner eingabe = new Scanner(System.in);
String wert = eingabe.next();
x = Integer.parseInt(wert);
```

Einlesen von Datentypen

Für das Einlesen von den Standard Datentypen (siehe Tabelle 0.4) bietet Scanner auch Möglichkeiten, diese direkt einzulesen.

Beispiel:

```
int ganzeZahl;
double kommaZahl;
ganzeZahl= eingabe.nextInt();
kommaZahl= eingabe.nextDouble();
```

Theorieteil

0.1 Modulübersicht Test 2

Die beiden Konzepte Variablen und Datentypen sind für jede Programmierung grundlegend. Bei **Variablen** handelt es sich um Speicherbereiche, in denen Werte gespeichert werden können und der **Datentyp** gibt an, welche Werte erlaubt sind (z.B. nur Ganzzahlen). In einem Programm werden Daten verarbeitet, die sich in ihrer Art unterscheiden, z.B. Zeichen, Zahlen oder logische Daten. Digitale Daten werden immer durch Ziffern dargestellt. Daher auch der Name, *digit* bedeutet Ziffer.

0.2 Darstellung von Zahlen und Zeichen im Computer

Um die Darstellung von Zeichen, Zahlen und Texten im Computer zu verstehen, muss man das **binäre System** verstehen.

0.2.1 Binäres System

Alle Rechner stellen Information im binären System dar. Dieses kennt nur zwei Ziffern, nämlich 0 und 1 (im Gegensatz zum Dezimalsystem mit den Ziffern 0 bis 9). Eine solche Ziffer wird als **Bit** bezeichnet (Abkürzung für *Binary Digit*, übersetzt „Binäre Ziffer“). Ein Bit stellt den kleinsten speicherbaren Wert in einem Computer dar. Jeweils 8 Bits werden zu einem **Byte** zusammengefasst. Ein Byte kann somit $2^8 = 256$ verschiedene Sequenzen von je 8 Bit speichern.

0.2.2 Darstellung von Zahlen im binären System

Betrachten wir die Zahl 91, die binär mit 8 Bit als 01011011 dargestellt wird (siehe Tabelle 0.1). Wir reden deswegen in diesem Zusammenhang von der **Binärdarstellung** von 91 (und nicht von der Dezimaldarstellung, die für uns lesefreundlicher ist).

Bit	8	7	6	5	4	3	3	1	
Binärwert	0	1	0	1	1	0	1	1	
Wertigkeit	$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	
Dezimalwert	0	64	0	16	8	0	2	1	= 91

Tabelle 0.1: Binäre Darstellung der Dezimalzahl 91. Details siehe Text.

Eine 8-Bit-Zahl, wie in unserem Beispiel, kann Werte zwischen 00000000 (0 im Dezimalsystem) und 11111111 (255 im Dezimalsystem) speichern. Für die Umrechnung vom Binär- in den Dezimalwert multiplizieren wir für jedes Bit den Binärwert mit der Wertigkeit des Bits und summieren diese auf. Im binären System können wir mit 8 Bit nur die ganzen Zahlen 0 bis 255 darstellen. Ist die Zahl, die wir darstellen wollen, grösser, muss auch ein grösserer Speicherbereich bereitgestellt werden.

0.2.3 Darstellung von Zeichen im binären System

Für die Darstellung von Zeichen im Computer wurde der so genannte **ASCII-Code** entwickelt. ASCII steht für *American Standard Code for Information Interchange*, was übersetzt so viel heisst wie Amerikanische Standardcodierung für den Datenaustausch. Mit Hilfe des 7-Bit-ASCII-Codes können 128 verschiedene Zeichen (2^7) dargestellt werden oder umgekehrt wird jedem Zeichen ein Bitmuster aus 7 Bit zugeordnet (siehe Tabelle 0.2). Die Zeichen entsprechen weitgehend einer Computertastatur. Der ASCII-Code wurde später auf 8 Bit erweitert, was die Darstellung von 256 Zeichen (2^8) erlaubt.

Die ASCII-Tabelle enthält auch nicht darstellbare Zeichen (wie etwa ein Zeichen, das einen Zeilenumbruch repräsentiert). Die wichtigsten sind in Tabelle 0.2.3 dargestellt:

0.3 Datentypen

Der **Datentyp** gibt an, welche Daten in einem Programm gespeichert und bearbeitet werden können. Programmiersprachen besitzen vordefinierte Datentypen, die sich in der Art der Interpretation der gespeicherten Daten und in der Grösse unterscheiden.

- Typ für Zahlenwerte
- Typ für Zeichenwerte
- Typ für Wahrheitswerte (Boolsche Werte) (siehe Modul 2)

Tabelle 0.4 gibt einen Überblick über die wichtigsten Datentypen, die in vielen Programmiersprachen vorkommen.

0-31		31-63		64-95		96-127	
Dez	Zeichen	Dez	Zeichen	Dez	Zeichen	Dez	Zeichen
0	NUL	32	SP	64	@	96	‘
1	SOH	33	!	65	A	97	a
2	STX	34	“	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	’	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Tabelle 0.2: ASCII-Tabelle

Dez	Zeichen	Bedeutung
8	BS	Backspace. Linkes Zeichen löschen
10	NL	New Line. Neue Zeile beginnen
32	SP	Space. Leerzeichen
127	DEL	Delete. Rechtes Zeichen löschen

Tabelle 0.3: Nicht darstellbare Zeichen der ASCII-Tabelle

Typ	Beschreibung	Grösse in Bit	Wertebereich
boolean	Boolscher Wert	1	true oder false
char	Zeichen	16	Unicode-Zeichen
byte	Ganzzahl	8	$-128 \ldots 127$ $(-2^7 \ldots + 2^7 - 1)$
short		16	$-32'768 \ldots 32'767$ $(-2^{15} \ldots + 2^{15} - 1)$
int		32	$-2'147'483'648 \ldots 2'147'483'647$ $(-2^{31} \ldots + 2^{31} - 1)$
long		64	$-9'223'372'036'854'775'808 \ldots$ $9'223'372'036'854'775'807$ $(-2^{63} \ldots + 2^{63} - 1)$
float	Gleitkommazahl	32	$+/- 3.40282347 \times 10^{38}$
double		64	$+/- 1.79769313486231569 \times 10^{308}$

Tabelle 0.4: Die wichtigsten Datentypen in Java

0.4 Variablen und Konstanten

Variablen einer Programmiersprache sind denen der Mathematik ähnlich. Variablen haben einen Namen über den sie angesprochen werden, und dienen dazu, einen Wert im Bereich eines bestimmten Datentyps zu speichern. Der Wert der Variable kann sich im Verlaufe des Programms ändern (er kann variieren, daher der Name). Um eine Variable in einem Programm verwenden zu können, sind folgende Operationen notwendig:

1. Deklaration
2. Initialisierung

0.4.1 Deklaration

Bevor eine Variable in einem Programm verwendet werden kann, muss sie **deklariert** werden. Das heisst, dass Sie als Programmiererin oder Programmierer einen Speicherbereich für einen bestimmten Datentyp belegen und diesem Speicherplatz einen Namen geben. Über diesen Namen kann der Speicherbereich während des Programmablaufs aufgerufen werden. Namen von Variablen beginnen in Java gemäss Konvention jeweils mit einem Kleinbuchstaben, sie dürfen keine Leerzeichen enthalten und sollten möglichst aussagekräftig sein.

Schreibweise:

```
Datentyp name;
```

Beispiel:

```
// Variable a vom Typ Integer
int a;

// Variable b vom Typ Double
double b;

// Variable c vom Typ Character
char c;
```

Mehrere Variablen vom gleichen Typ können auch wie in folgendem Beispiel in einer Deklaration geschrieben werden:

```
// 3 Variablem vom Typ Integer
int meineZahl1, meineZahl2, meineZahl3;
```

0.4.2 Initialisierung und Wertzuweisung

Das **Zuweisen** von Werten geschieht mit dem Zuweisungsoperator. In Java wird hierfür ein **Gleichheitszeichen** (=) verwendet. Dabei wird der Wert des Ausdrucks rechts des Zuweisungsoperators der Variablen auf der linken Seite zugewiesen. Wenn einer Variable das erste Mal ein Wert zugewiesen wird, spricht man von einer **Initialisierung**.

Schreibweise:

```
variable = wert;
```

Beispiel:

```
meineZahl = 4;
// meineZahl hat den Wert 4
```

Der Wert einer Variablen kann sich im Verlaufe eines Programms ändern. In folgendem Beispiel wird in der Variablen `meineZahl` zuerst der Wert 4 gespeichert, der dann in einer weiteren Zeile mit dem Wert 6 überchrieben wird.

```
meineZahl = 4;
// Wert von meineZahl ist 4

meineZahl = 6;
// Wert von meineZahl ist 6
```

Bei einer Zuweisung handelt es sich also immer um einen schreibenden Zugriff auf eine Variable mit dem Resultat, dass sich deren Wert ändern kann. Der alte Wert wird überschrieben. Damit einer Variablen ein Wert zugewiesen werden kann, darf die Variable nicht als Konstante definiert sein (siehe nächster Abschnitt) und der Typ der Variablen muss mit dem Typ des Werts kompatibel sein. Auf jeden Fall kompatibel sind Variablen und Werte desselben Datentypes. Wenn die Datentypen nicht übereinstimmen, nimmt Java eine implizite **Typkonvertierung** vor. Dies ist jedoch eine häufige Fehlerquelle und sollte daher vermieden werden. Bei der impliziten Typkonvertierung in Java werden nur Typkonvertierung durchgeführt, wenn sie ohne Informationsverlust erfolgen kann, also wenn der Zieldatentyp einen gleichen oder grösseren Wertebereich hat als der Ausgangsdatentyp.

Beispiel: Ein Wert vom Typ `int` kann einer Variablen vom Typ `double` zugewiesen werden:

```
// Variable ganzeZahl vom Typ Integer
int ganzeZahl;
// Variable kommaZahl vom Typ Double
double kommaZahl;

ganzeZahl = 4;
kommaZahl = ganzeZahl;
// Variable kommaZahl wird in zum Typ Integer konvertiert
```

Möchte man eine Zuweisung machen, bei der der Zieldatentyp einen gleineren Wertebereich hat, muss eine so genannte explizite Typenkonvertierung durchgeführt werden, das sogenannte Typecasting. Die Programmiererin/der Programmierer ist dabei selber dafür verantwortlich, dass die Zuweisung möglich ist.

Beispiel:

```
// Variable d vom Typ Double
double d = 1.3;

float f = (float)d;
// Variable d muss explizit zum Typ Float konvertiert
// werden
```

Eine Variable kann in einem Programm nur in einem bestimmten Bereich des Programms gelten. Weiteres dazu erfahren Sie in einem späteren Modul.

0.4.3 Konstanten

Konstanten werden wie Variablen mit einem Namen bezeichnet. Sie enthalten während der gesamten Programmausführung einen konstanten Wert. Es kann also nach der Initialisierung keine weitere Wertzuweisung vorgenommen werden. Konstanten können jedoch Teil einer Wertzuweisung an Variablen sein. Eine Konstante wird zusätzlich zu Namen und Datentyp mit dem Schlüsselwort `final` deklariert.

Schreibweise:

```
final Datentyp name;
```

Beispiel:

```
//Deklaration und Initialisierung der Konstante k
final int k = 4;
```

0.5 Operatoren und Ausdrücke

0.5.1 Operatoren (Teil I)

Um in einem Programm Berechnungen durchzuführen zu können, stehen folgende **arithmetische Operatoren** zur Verfügung:

Operator	Ausdruck	Beschreibung	Liefert
+	a + b	Addition	Summe
-	a - b	Subtraktion	Differenz
*	a * b	Multiplikation	Produkt
/	a / b	Division	Quotient
%	a % b	Modulo	Ganzzahliger Rest einer Division

Tabelle 0.5: Arithmetische Operatoren in Java

Weitere Operatoren (logische und Vergleichsoperatoren) lernen Sie in Modul 2 kennen.

0.5.2 Ausdrücke

Ausdrücke (engl. *expressions*) sind in einer Programmiersprache Teil der kleinsten ausführbaren Einheiten eines Programms. Dabei handelt es sich um Verarbeitungsvorschriften, die sich aus **Variablen**, **Konstanten** und **Operatoren** zusammensetzen können und ein Resultat ergeben. Variablen und Konstanten, die mit einem Operator verknüpft werden, nennt man **Operanden**. Ein Ausdruck kann auch aus einer einzelnen Variablen bestehen.

Folgendes Beispiel zeigt einen Ausdruck, der aus einer Variablen `i`, einem Operator `+` und einem Operanden `5` besteht.

```
i + 5
```

Das Resultat des Ausdrucks kann wieder in einer Variablen gespeichert werden. In folgendem Beispiel wird das Resultat in der Variablen `i` gespeichert. Der vorherige Wert von `i` wird dadurch überschrieben.

```
i = i + 5;
```

Die Reihenfolge, mit der Ausdrücke bearbeitet werden, kann durch die Wahl des Operators und durch Klammern beeinflusst werden. Hierfür gelten die mathematischen Regeln, wie wir sie in der Schule gelernt haben, also „Klammern zuerst, dann Punkt vor Strich".

Beispiel:

```
5 * (2 + 10)
```

Die Klammern erzwingen, dass die Addition vor der Multiplikation ausgeführt wird.

0.5.3 Weitere Arithmetische Operatoren

Es gibt in Java noch weitere arithmetische Operatoren.

Zuweisungsoperator:

```
i += 1; \\entspricht i = i + 1;
i -= 1; \\entspricht i = i - 1;
i *= 1; \\entspricht i = i * 1;
i /= 1; \\entspricht i = i / 1;
i %= 1; \\entspricht i = i % 1;
```

Die Zuweisungsoperatoren dienen dazu die Anweisungen kompakter darzustellen, da man weniger Zeichen benötigt.

Increment und Decrement Operatoren:

```
i++; \\entspricht i = i + 1;
i--; \\entspricht i = i - 1;
```

Diese Operatoren sind meistens in sogenannten for-Schleifen anzutreffen, wo sie einen Zähler hochzählen (siehe Modul 2) .

0.6 Der Datentyp String

Der Datentyp **String** unterscheidet sich von den bisher thematisierten Datentypen insofern, dass er eine Zusammenfassung von mehreren gleichartigen Variablen darstellt. Dieser Datentyp ist auch kein primitiver Datentyp mehr, da er mehrere Elemente zusammenfasst. Er speichert nämlich alle Buchstaben einzeln in je einer char-Variablen. Wie diese Zusammenfassung der einzelnen Buchstaben funktioniert, lernen Sie, wenn es um die Objektorientierung geht. Die Deklaration und Initialisierung der Variablen funktioniert jedoch wie in 1.4.1 und 1.4.2 beschreiben.

Bei der Initialisierung von String-Variablen muss der Wert zwischen **doppelten Hochkommata** (“) angegeben werden.

Beispiel:

```
// Deklaration des Strings vorname
String vorname;

// Initialisierung mit dem Wert "Paul"
vorname = "Paul";
```

Da ein String mehrere char-Variablen enthält, kann dem String auch ein einzelner char zugewiesen werden.

Beispiel:

```
String name;
name = 'a';
```

Mehrere Strings können mit einem **Plus** (+) verbunden werden. So entsteht aus mehreren Einzelteilen ein neuer Text.

Beispiel:

```
String text;
text = "Hallo, "+ "das " + "sind " + "mehrere " + "Wörter.";
```

0.7 Bildschirm- Ein- und Ausgabe

Oft möchte man, dass die Benutzerin oder der Benutzer des Programms mit dem Programm interagieren kann. Das bedeutet, dass das Programm eine Ausgabe macht oder

dass die Benutzerin oder der Benutzer etwas eingeben kann. Um dies zu realisieren verwenden wir Funktionalitäten, welche von Java zur Verfügung gestellt werden.

0.7.1 Ausgabe

Damit die Benutzerin oder der Benutzer auch sieht, was im Programm berechnet wurde, kann im Programmcode angegeben werden, dass ein bestimmter Text oder der Wert einer Variablen ausgegeben wird.

Beispiel: Ausgabe eines vorgegebenen Texts

```
System.out.println("Das Programm hat geendet.");
```

Im obigen Beispiel wird der Text „Das Programm hat geendet.” ausgegeben. Der Text, der ausgegeben wird, steht zwischen einem Paar von Anführungs- und Schlusszeichen (“), die nicht mit ausgegeben werden. Man möchte aber nicht immer nur vorgegebenen Text ausgeben, sondern z.B. das Resultat einer Berechnung, welches in einer Variablen (z.B. `ganzeZahl`) gespeichert ist.

Beispiel: Ausgabe des Wertes einer Variablen

```
System.out.println(ganzeZahl);
```

Wenn man Variablenwerte und Text verbinden möchte, geschieht dies mit einem Additions-Zeichen (+).

Beispiel: Ausgabe von Text und Variablenwert

```
System.out.println("Das Resultat ist:" + ganzeZahl);
```

0.7.2 Eingabe

Oft möchte man den Wert einer Variablen durch die Benutzerin oder den Benutzer eines Programms bestimmen lassen. Eine einfache Möglichkeit ist die Eingabe mit der Tastatur über das Konsolenfenster.

Eine Benutzereingabe ist in Java etwas aufwändiger als bei anderen Programmiersprachen. Sie beinhaltet folgende zwei Schritte:

Paket einbinden: Mit einer Importanweisung zu Beginn unseres Java-Programms muss zunächst die Klasse Scanner des Pakets util eingebunden werden:

```
import java.util.Scanner;
```

Werte einlesen: Mit folgenden zwei Zeilen können wir Werte in Form von Zeichenketten (String) vom Konsolenfenster einlesen und einer Variablen (z.B. wert) zuweisen:

```
Scanner eingabe = new Scanner(System.in);
String wert = eingabe.next();
```

Nun weisen wir den eingelesenen Wert unserer Variablen `zahl` zu. Hierfür muss der eingelesene Text noch in einen Integer umgewandelt werden:

```
Integer.parseInt(wert);
```

Beispiel: Mit den folgenden Anweisungen übergeben wir eine Eingabezahl von der Konsole an die Variable `x` vom Typ Integer:

```
int x;
Scanner eingabe = new Scanner(System.in);
String wert = eingabe.next();
x = Integer.parseInt(wert);
```

Einlesen von Datentypen

Für das Einlesen von den Standard Datentypen (siehe Tabelle 0.4) bietet Scanner auch Möglichkeiten, diese direkt einzulesen.

Beispiel:

```
int ganzeZahl;
double kommaZahl;
ganzeZahl= eingabe.nextInt();
kommaZahl= eingabe.nextDouble();
```