Go Programmierung Teil 1 Handout

Nima Mohammadimohammadi

June 14, 2025

1 Datentypen in Go

Die Programmiersprache Go kennt verschiedene grundlegende Datentypen. Die folgende Tabelle zeigt eine Auswahl der wichtigsten Typen:
(hier sind auch welche die wir nicht behandelt haben)

Typ	**Kategorie**	**Beispielwert**

`int`
`int8`
`int16`
`int32`
`int64`
	Ganze Zahlen (vorzeichenbehaftet)	`42`, `-7`

`uint`
`uint8`
`uint16`
`uint32`
`uint64`
	Ganze Zahlen (vorzeichenlos)	`10`, `255`

`byte`	Alias für `uint8`	`0, 129`
`float32` `float64`	Fließkommazahlen	`3.14`, `-0.1`
`string`	Zeichenkette	`"Hallo"`
`bool`	Wahrheitswert	`true`, `false`
`rune`	Unicode-Zeichen (Alias für `int32`)	`'A'`, `'ß'`

1.1 Wertebereiche von Zahlentypen in Go

Die folgende Tabelle zeigt die wichtigsten Ganzzahl- und Fließkommatypen in Go mit ihren Wertebereichen:

Typ	**Minimalwert**	**Maximalwert**
`int8`	-128	+127
`int16`	-32768	+32767
`int32`	-2147483648	+2147483647
`int64`	−9 Billionen	+9 Billionen
`uint8`	0	255
`uint16`	0	65535
`uint32`	0	4294967295
`uint64`	0	18 Billionen
`float32`	10^{-38}	10^{+38}
`float64`	10^{-308}	10^{+308}

2 Variablen / Konstanten

2.1 Bedeutung

Eine **Variable** ist ein Platzhalter für einen Wert, der sich ändern kann. In der Programmierung wird eine Variable verwendet, um Daten zu speichern, z. B. Zahlen, Texte oder andere Informationen, damit man später darauf zugreifen oder sie verändern kann.

Eine **Konstante** deklariert einen Wert, der sich im Programmablauf nicht ändert.

2.2 Syntax

```
1) var|const <name> <typ>

2) var|const <name> [<typ>] = <wert>

<typ>  : muss ein Datentyp von Go sein
<name> : beliebiger Bezeichner
<wert> : zugewiesener Wert
```

Listing 1: Variablen- und Konstantendeklaration in Go

```
1) <name> := <wert>

<name> : beliebiger Bezeichner
<wert> : beliebiges Literal
```

Listing 2: Alternative für Variablendeklaration

In **Listing 2** wird eine Variable deklariert, wobei der Typ der Variable aus dem zugewiesenen Wert automatisch abgeleitet wird.

Ausnahme:

- Ganzzahl-Literale werden standardmäßig als `int` interpretiert
- Kommazahl-Literale werden standardmäßig als `float64` interpretiert

3 Grundlegende Operationen

3.1 Boolesche Operationen

Boolesche Operationen arbeiten mit Wahrheitswerten (true/false) und sind grundlegend für die Steuerung von Programmabläufen zuständig.

- **UND** (AND, `&&`): Liefert `true`, wenn beide Operanden `true` sind.
- **ODER** (OR, `||`): Liefert `true`, wenn mindestens ein Operand `true` ist.
- **NICHT** (NOT, `!`): Kehrt den Wahrheitswert um.

Beispiele:
`true && false` = `false`
`true || false` = `false`
`! false` = `true`

3.2 Arithmetik

Arithmetische Operationen verarbeiten numerische Werte und umfassen:

- Addition: $a + b$
- Subtraktion: $a - b$
- Multiplikation: $a \times b$
- Division: $a \div b$
- Modulo (Rest bei Division): $a \bmod b$

Beispiel: $7 + 3 = 10$, $10 \bmod 3 = 1$

3.3 Vergleiche

Vergleichsoperatoren vergleichen zwei Werte und liefern einen booleschen Wert als Ergebnis:

- Gleichheit: $a == b$
- Ungleichheit: $a \neq b$
- Größer als: $a > b$
- Kleiner als: $a < b$
- Größer oder gleich: $a \geq b$
- Kleiner oder gleich: $a \leq b$

Beispiel: $5 > 3 =$ `true`

3.4 Zuweisung

Die Zuweisung weist einer Variablen einen Wert zu. Das geschieht in der Regel mit dem Gleichheitszeichen `=`.

$$x = 5$$

weist der Variablen x den Wert 5 zu.

Erweiterte Zuweisungsoperatoren Diese Operatoren kombinieren eine arithmetische Operation mit der Zuweisung und sparen Schreibarbeit:

- `+=` : $x+ = y \quad \Rightarrow \quad x = x + y$
- `-=` : $x- = y \quad \Rightarrow \quad x = x - y$
- `*=` : $x* = y \quad \Rightarrow \quad x = x \times y$
- `/=` : $x/ = y \quad \Rightarrow \quad x = x \div y$
- `%=` : $x\% = y \quad \Rightarrow \quad x = x \bmod y$

Diese Operatoren sind in vielen Programmiersprachen gebräuchlich und erleichtern das Schreiben von kompakterem Code.

3.5 Ein- und Ausgabe in Go

Ausgabe: In Go verwendet man üblicherweise das `fmt`-Paket für die Ausgabe. Die wichtigsten Funktionen sind:

- `fmt.Print`: Gibt Werte ohne Zeilenumbruch aus.
- `fmt.Println`: Gibt Werte mit Zeilenumbruch aus.

Beispiel:

```
fmt.Print("Hallo, ")
fmt.Println("Welt!")
```

Ausgabe:

```
Hallo, Welt!
```

Eingabe: Für Eingaben nutzt man ebenfalls das `fmt`-Paket mit `Scan`.

Beispiel:

```
var zahl int
fmt.Print("Gib eine Zahl ein: ")
fmt.Scan(&zahl)
```

Hier wird eine ganze Zahl eingelesen und in der Variablen `zahl` gespeichert.

3.6 Typkonvertierungen in Go

In Go sind Typkonvertierungen explizit durchzuführen, da die Sprache keine impliziten Typumwandlungen erlaubt. Eine Typkonvertierung besteht darin, einen Wert eines bestimmten Datentyps in einen Wert eines anderen, kompatiblen Datentyps zu überführen.
Die allgemeine Form der Typkonvertierung lautet:

$$\texttt{T}(v)$$

wobei v ein Ausdruck eines beliebigen Typs ist und `T` der Zieltyp der Konvertierung ist.

Beispiele:

- Umwandlung eines ganzzahligen Werts i in einen Gleitkommawert:

$$f := \texttt{float64}(i)$$

- Umwandlung eines Gleitkommawerts x in einen ganzzahligen Wert (mit möglichem Informationsverlust durch Abschneiden der Nachkommastellen):

$$y := \texttt{int32}(x)$$

- Umwandlung eines Ganzzahlwerts in einen Bytewert (8-Bit-Unsigned-Integer):

$$b := \texttt{byte}(i)$$

Hinweise:

- Typkonvertierungen sind nur zulässig zwischen kompatiblen Typen, beispielsweise zwischen verschiedenen numerischen Typen.
- Die Konvertierung kann zu Informationsverlust führen, z. B. durch Abschneiden von Nachkommastellen oder Überlauf bei zu kleinen Zieltypen.
- Eine direkte Typkonvertierung zwischen inkompatiblen Typen (z. B. `string` und `int`) ist nicht möglich und erfordert spezielle Funktionen zur Umwandlung.