String Formatting in Go – Ausführliche Erklärung, Aufgaben & Lösungen

Dieses Skript erklärt, was **String Formatting** ist, wie man es in Go mit dem Paket **fmt** nutzt, und enthält anschließend Übungsaufgaben mit ausführlichen Musterlösungen. Zu jeder Lösung gibt es eine kurze Erklärung der *Key-Komponente*, damit das Verständnis vertieft wird.

1. Was ist String Formatting?

Unter **String Formatting** versteht man das **gezielte Zusammensetzen** von Text aus festen Textteilen und variablen Werten (Zahlen, Strings, Typen, Strukturen). Statt Werte einfach aneinander zu „kleben“ (Stringverkettung), nutzt man **Formatvorlagen** mit **Platzhaltern**. Diese Platzhalter werden zur Laufzeit durch die übergebenen Werte ersetzt.

Warum ist das nützlich?

- **Lesbarkeit:** Ein Formatstring zeigt sofort, wie die Ausgabe aussehen soll.
- **Typsicherheit:** Go überprüft viele Fehler (z. B. falsche Verb/Typ-Kombinationen) frühzeitig.
- **Kontrolle:** Du bestimmst Breiten, Nachkommastellen, Ausrichtung und mehr.
- **Standard in Logs/CLI:** Konsolenprogramme und Log-Ausgaben nutzen Formatting fast immer.

2. Das fmt-Paket: Printf, Sprintf, Fprintf

In Go ist das wichtigste Werkzeug für String Formatting das Paket **fmt**. Darin gibt es drei besonders häufige Varianten:

- **fmt.Printf**(format, args...): formatiert und schreibt direkt auf die Konsole (stdout).
- **fmt.Sprintf**(format, args...): formatiert und **gibt einen String zurück** (keine Ausgabe).
- **fmt.Fprintf**(writer, format, args...): formatiert und schreibt in einen beliebigen Writer (z. B. Datei, Buffer).

Merksatz: **Printf** = Print (Ausgabe), **Sprintf** = String (Rückgabewert).

Mini-Beispiel

```
package main

import "fmt"

func main() {
name := "Max"
age := 21

// Ausgabe auf Konsole:
fmt.Printf("Name: %s, Alter: %d\n", name, age)

// String bauen:
s := fmt.Sprintf("Name: %s, Alter: %d", name, age)
fmt.Println(s)
}
```

3. Formatverben und Formatoptionen

Ein **Formatverb** beginnt mit **%** (z. B. **%s**, **%d**, **%.2f**). Du kannst zusätzlich **Breite**, **Ausrichtung** und **Genauigkeit** angeben.

Wichtige Verben im Überblick

Verb	Bedeutung	Beispiel
%s	String	fmt.Sprintf("Hallo %s", "Welt")
%d	int (dezimal)	fmt.Sprintf("%d", 42)
%f	float (Standard)	fmt.Sprintf("%f", 3.14)
%.2f	float mit 2 Nachkommastellen	fmt.Sprintf("%.2f", 3.14159)
%t	bool	fmt.Sprintf("%t", true)
%v	Standarddarstellung (wert)	fmt.Sprintf("%v", anyValue)
%T	Datentyp	fmt.Sprintf("%T", anyValue)
%%	gibt ein % aus	fmt.Sprintf("100%%")

Breite und Ausrichtung (tabellarische Ausgabe)

Mit einer **Breite** kannst du Spalten bauen. Beispiel: **%-10s** bedeutet: String linksbündig in einem Feld der Breite 10. Ohne Minus (**%10s**) ist es rechtsbündig.

```
fmt.Sprintf("%-10s %4d", "Max", 7) // linksbündiger Name, rechtsbündige Zahl
```

Genauigkeit bei floats

Bei Fließkommazahlen ist die Genauigkeit entscheidend (z. B. Geld). **%.2f** formatiert mit genau zwei Nachkommastellen.

```
fmt.Sprintf("Preis: %.2f €", 12.5) // "Preis: 12.50 €"
```

Stringverkettung vs. Formatting

Stringverkettung (mit +) funktioniert, wird aber schnell unübersichtlich und kann Fehler fördern (z. B. fehlende Leerzeichen, Konvertierungen). Formatting ist oft klarer und kontrollierbarer.

```
// Verkettung
"Name: " + name + ", Alter: " + strconv.Itoa(age)

// Formatting
fmt.Sprintf("Name: %s, Alter: %d", name, age)
```

4. Aufgabenstellung

Bearbeite die folgenden Aufgaben. Ziel ist, das passende Formatverb zu wählen und fmt.Sprintf korrekt einzusetzen. Empfehlung: Schreibe erst Tests oder Beispieleingaben und überprüfe die Ausgabe exakt.

Aufgabe 1 – Person formatieren

Implementiere **FormatPerson(name string, age int) string**. Der Rückgabestring soll exakt so aussehen:
Name: Max, Alter: 21
Achte auf Kommas und Leerzeichen.

Aufgabe 2 – Preis mit zwei Nachkommastellen

Implementiere **FormatPrice(price float64) string**. Ausgabeformat:
Preis: 12.50 €
Wichtig: Immer genau **zwei** Nachkommastellen, auch bei 12.5.

Aufgabe 3 – Bestellung formatieren

Implementiere **FormatOrder(product string, amount int, price float64) string**. Format:
Produkt: Apfel | Menge: 3 | Einzelpreis: 0.50 €
Nutze passende Verben für String, int und float.

Aufgabe 4 – Wert und Typ ausgeben

Implementiere **DescribeValue(v interface{}) string**. Format:
Wert: 42 | Typ: int
Hinweis: Nutze **%v** für den Wert und **%T** für den Typ.

Aufgabe 5 – Satz ohne Stringverkettung

Implementiere **UserText(name string, age int) string** ohne den + Operator. Format:
Der Benutzer Max ist 21 Jahre alt.
Achte auf Punkt am Ende.

Aufgabe 6 – Spaltenbündige Ausgabe

Implementiere **FormatScore(name string, score int) string**. Gib den Namen in einer Spalte der Breite 10 **linksbündig** aus und danach den Score.
Beispiel:
Max 120
Tipp: Nutze **%-10s**.

5. Musterlösungen mit Key-Erklärungen

Die folgenden Lösungen sind als Hilfe gedacht. Lies besonders die Key-Erklärungen: Sie erklären jeweils das zentrale Konzept, das du aus der Aufgabe mitnehmen sollst.

Lösung zu Aufgabe 1 – FormatPerson

```
import "fmt"

func FormatPerson(name string, age int) string {
return fmt.Sprintf("Name: %s, Alter: %d", name, age)
}
```

Key-Komponente: Die Kombination aus **%s** (String) und **%d** (int). Wichtig ist, dass die Argumente in der gleichen Reihenfolge wie die Platzhalter übergeben werden. Mit **Sprintf** wird ein String erzeugt (nicht direkt ausgegeben).

Lösung zu Aufgabe 2 – FormatPrice

```
import "fmt"

func FormatPrice(price float64) string {
return fmt.Sprintf("Preis: %.2f €", price)
}
```

Key-Komponente: %.2f erzwingt exakt zwei Nachkommastellen. Das ist für Geldbeträge wichtig (12.5 wird zu 12.50).

Lösung zu Aufgabe 3 – FormatOrder

```
import "fmt"

func FormatOrder(product string, amount int, price float64) string {
return fmt.Sprintf("Produkt: %s | Menge: %d | Einzelpreis: %.2f €", product, amount, price)
}
```

Key-Komponente: Mehrere Datentypen in einem Formatstring. Go ordnet die Werte positionsbasiert den Platzhaltern zu. Für float wieder **%.2f**, damit der Einzelpreis sauber aussieht.

Lösung zu Aufgabe 4 – DescribeValue

```
import "fmt"

func DescribeValue(v interface{}) string {
return fmt.Sprintf("Wert: %v | Typ: %T", v)
}
```

Key-Komponente: %v ist die Standarddarstellung des Wertes, **%T** gibt den Typ aus. **interface{}** erlaubt es, beliebige Typen zu übergeben (z. B. int, string, structs).

Lösung zu Aufgabe 5 – UserText

```
import "fmt"

func UserText(name string, age int) string {
return fmt.Sprintf("Der Benutzer %s ist %d Jahre alt.", name, age)
}
```

Key-Komponente: Formatting ersetzt Verkettung und vermeidet zusätzliche Konvertierungen. Gerade bei Mischtypen (string + int) ist **Sprintf** sauberer als + mit strconv.

Lösung zu Aufgabe 6 – FormatScore

```
import "fmt"

func FormatScore(name string, score int) string {
return fmt.Sprintf("%-10s %d", name, score)
}
```

Key-Komponente: %-10s bedeutet: linksbündig (**-**) und Feldbreite 10. So entstehen Spalten, die untereinander stehen. Für Tabellen in der Konsole ist das extrem praktisch.

6. Mini-Cheat-Sheet

Kurzüberblick für Übungen/Klausur:

- **fmt.Sprintf** → String zurückgeben
- **fmt.Printf** → direkt ausgeben
- **%s** string, **%d** int, **%.2f** float mit 2 Nachkommastellen
- **%v** Wert (Standard), **%T** Typ, **%%** Prozentzeichen
- **%-10s** linksbündig in Breite 10 (Spalten)