

# Teil 2 Go - Programmierung

Mohammadimohammadi, Nima

# Agenda Teil 2

- 1) Printf(...)
- 2) Mehrfachzuweisung /- deklaration
- 3) Switch Statements
- 4) Schleifen

## Printf()

---

- In den Aufgaben haben wir gesehen das formatieren viel zusätzlichen Code braucht
- Allein mit `fmt.Print()/Println()` werden wir nicht glücklich!
- Lösung: ( **Print** formatted )
  - `fmt.Printf(...)`
- Beispiel: "Meine Lieblingszahl lautet 42 und ich bin 32 geworden"
- Implementierung mit `Print/Println`:
  - `fmt.Print("Meine Lieblingszahl lautet ", zahl)`
  - `fmt.Println(" und ich bin", alter,"geworden")`
- **Finden wir das gut?**

## Nachtrag Printf()

---

Beispiel: "Meine Lieblingszahl lautet <zahl> und ich bin <alter> geworden"

- Template/Format:
  - "Meine Lieblingszahl lautet %d und ich bin %d geworden"
- `fmt.Printf("Meine Lieblingszahl lautet %d und ich bin %d geworden", zahl, alter)`
- **Sind wir fertig?**
- `\n` : Zeilenumbruch
- `fmt.Printf("Meine Lieblingszahl lautet %d und ich bin %d geworden\n", zahl, alter)`
- `fmt.Println("Zeile1\nZeile2\n")`
- `fmt.Println(...): fmt.Print(..., "\n")`

## Printf Spezial Literale

### Go `fmt.Printf()` - Wichtige Platzhalter

Platzhalter	Typ	Beschreibung	Beispiel
<code>%d</code>	<code>int</code>	Ganze Zahl	42
<code>%f</code>	<code>float64</code>	Kommazahl (Standard: 6 Nachkommastellen)	3.141590
<code>%.2f</code>	<code>float64</code>	Kommazahl mit 2 Nachkommastellen	3.14
<code>%s</code>	<code>string</code>	Zeichenkette	Hallo
<code>%t</code>	<code>bool</code>	Wahrheitswert	true
<code>%v</code>	beliebig	Standardausgabe für beliebigen Typ	123, "abc"
<code>%T</code>	beliebig	Typ der Variable	bool, float64
<code>%q</code>	<code>string</code>	In Anführungszeichen	"Hallo"
<code>%%</code>	-	Gibt ein Prozentzeichen aus	%
<code>\n</code>	-	Neue Zeile (Zeilenumbruch)	Zeile1\nZeile2

## Beispiel Printf()

---

```
name := "Anna"
alter := 29
groesse := 1.68
student := true
fmt.Printf("Name: %s\n", name)
fmt.Printf("Alter: %d Jahre\n", alter)
fmt.Printf("Größe: %.2f m\n", groesse)
fmt.Printf("Student: %t\n", student)
fmt.Printf("Typ von 'groesse': %T\n", groesse)
fmt.Printf("Go-Syntax von 'name': %#v\n", name)
fmt.Printf("Zitat: %q\n", name)
fmt.Printf("Prozentzeichen: %%%d\n%t\n%s", 5, true, "Moin")
```

# Mehrfachzuweisung /- deklaration

---


## Mehrfachdeklaration

- Syntax:  
`const|var <name1>, <name2>, ... (<type>)|(= <wert1>, <wert2>, ... )`
- Alternative:  
`<name1>, <name2>, ... := <wert1>, <wert2>, ...`

## Mehrfachzuweisung




- Syntax  
`<var1>, <var2>, ... = <wert1>, <wert2>, ...`

## Beispiele: Mehrfach-Zuweisung/Deklaration

<pre>var x, y byte = 2, 3 x, y = y, x fmt.Println(x,y)</pre>	<pre>var x, y int = 10, 42 _, y = y, x fmt.Println(x,y)</pre>
<pre>var x, y = int16(256), int8(3) x, y = y, x fmt.Println(x,y)</pre> 	<pre>var x, y byte = 32, 64 x, y = y, x * 2 fmt.Println(x,y)</pre>
<pre>x := 1 y, x := 2, x fmt.Println(x,y)</pre>	<pre>a,b,c,d := true, 1, 4.3, "Demacia" fmt.Println(b,c,b,a)</pre>



## Beispiele: Mehrfach-Zuweisung/Deklaration

<pre>const MwSt, RentenAlter = 0.19, 65 fmt.Println(MwSt, RentenAlter)</pre>	<pre>const XSize,YSize = XSize,XSize+2 fmt.Println(XSize,YSize)</pre> 
<pre>const Pi, alter = 3,14, 15 fmt.Println(Pi, alter)</pre> 	<pre>const x, y = 10, 12 x, y = x * 2, y * 2 fmt.Println(x, y)</pre> 
<pre>const Einkommen, Alter = uint64(2000000000), byte(69)  fmt.Println(Einkommen, Alter)</pre>	

## VAR/CONST Blöcke

---

Mehrfach Deklaration von mehreren Variablen / Konstanten

- Syntax:

```
var|const (  
    <name1> (<typ1>) | (= <wert1>)  
    <name1> (<typ2>) | (= <wert2>)  
    ...  
)
```

Beispiele:

```
const (  
    Pi      float32    = 3.14159  
    Euler   = 2.71828  
)
```

```
var (  
    welcome string = "Willkommen"  
    zahl      int  
)
```

## Beispiel (if-else)

---

```
var month byte = ...

if month == 1 {
    fmt.Println("Du bist in Januar geboren")
} else if month == 2 {
    fmt.Println("Du bist in Februar geboren")
} else if month == 3 {

...

} else if month == 12 {
    fmt.Println("Du bist in Dezember geboren")
}
```

# Switch Statement

---

- Syntax:

```
switch <var> {  
  case <val1>:  
    <statement 1.1>  
    <statement 1.2>  
[ case <val2>:  
]  
[ default:  
  <statement d.1>  
  <statement d.2>  
]  
}
```

- switch ist eine saubere Alternative zu langen if-else-Ketten
- Wird verwendet, um mehrere Bedingungen effizient zu prüfen
- Dabei wird auf gleichheit getestet
- default wenn kein Bedingung passt

## Beispiel (Switch)

---

```
var month byte = ...
```

```
switch month {  
case 1:  
    fmt.Println("Du bist im Januar geboren")  
case 2:  
    fmt.Println("Du bist im Februar geboren")  
case 3:  
    fmt.Println("Du bist im März geboren")  
...  
case 12:  
    fmt.Println("Du bist im Dezember geboren")  
default:  
    fmt.Println("FEHLER Monat existiert nicht")  
}
```

## Beispiel (Switch)

---

```
var month byte = ...

switch month {
case 12, 1, 2:
    fmt.Println("Du bist im Winter geboren")
case 3, 4, 5:
    fmt.Println("Du bist im Frühling geboren")
case 6, 7, 8:
    fmt.Println("Du bist im Sommer geboren")
case 9, 10, 11:
    fmt.Println("Du bist im Herbst geboren")
default:
    fmt.Println("FEHLER Monat existiert nicht")
}
```

## Beispiel (Switch Ranges)

---

```
var income int32 = ...
```

```
switch {  
case income < 1000 && income > 0:  
    fmt.Println("Schüler/Student")  
case income < 10000:  
    fmt.Println("Minijob")  
case income < 80000:  
    fmt.Println("Vollzeit Job")  
case income < 200000:  
    fmt.Println("Unternehmer")  
default:  
    fmt.Println("ELON MUSK STYLE")  
}
```

- Typisch um Bereiche abzufangen
- Syntax:

```
switch {  
case <bool>:  
    <statement 1.1>  
    <statement 1.2>  
[ case <bool>:  
]  
[ default:  
    <statement d.1>  
    <statement d.2>  
]  
}
```

## Beispiel Fallthrough

---

```
day := "Wednesday"
fmt.Println("Arbeitstage bis Wochenende")
switch day {
case "Monday":
    fmt.Println("- Monday")
    fallthrough
case "Tuesday":
    fmt.Println("- Tuesday")
    fallthrough
case "Wednesday":
    fmt.Println("- Wednesday")
    fallthrough
case "Thursday":
    fmt.Println("- Thursday")
    fallthrough
case "Friday":
    fmt.Println("- Friday")
}
```

- fallthrough um ein case zu "droppen"
- Beispiel:
  - day = "Monday"
  - day = "Wednesdays"
  - day = "Friday"



## Fazit Switch

---

Wann `if` ?

- Immer
- Bei Komplexe Logik

Wann `switch` ?

- Bei vielen `else if` Konstrukte
- Bei viele Äquivalenzvergleiche mit selben Variablen (effizient)

# Einführung in Schleifen

---

- Eine **Schleife** wiederholt ein **bestimmten** Code so lange, bis eine bestimmte Bedingung **nicht mehr erfüllt** ist
- In Go gibt es **nur eine Schleife Keyword** `for`
- **Warum sind Schleifen nützlich?**
  - Vermeiden von **wiederholtem Code**
  - Ermöglichen das Durchlaufen von Daten (z. B. Arrays)
  - Basis für viele Aufgaben: Zählen, Suchen, Wiederholen
  - Programmierbaukasten ist vervollständigt
    - a. Wir können jetzt jedes Problem lösen mithilfe von Schleifen

# Schleifen

---

- Drei Arten an Schleifen
  - `while` Schleife
  - `for` Schleife
  - `do while` Schleife (oft auch `repeat until` genannt) [Existiert in Go nicht!]

Eine **Schleifenbedingung** ist die Steuerung einer Schleife die festlegt, wann die Schleife beendet werden soll. Sie verhindert, dass eine Schleife endlos weiterläuft, indem sie eine oder mehrere Bedingungen definiert, die bei Nichterfüllung die Schleife abbricht.

In der Informatik bezeichnet man **Iteration**, als die wiederholte Ausführung einer Anweisungsfolge.

## While Schleife

---



Semantik: Inneren Scope wird ausgeführt, solange die **Schleifenbedingung** erfüllt ist, zum Eintritt der Schleife muss die **Schleifenbedingung** auch erfüllt sein

Syntax:   for <schleifenbedingung> {  
                    <Iteration Code>  
          }



Warum while Schleife?

- Schleifen die beim Eintritt des inneren Scopes, die **Schleifenbedingung** überprüfen, werden in den meisten Programmiersprachen mit den `while` eingeleitet.

## Beispiel While Schleifen

<pre>var x byte = 10  for x &lt; 20 {     fmt.Println(x)     x++ }</pre>	<pre>var x int8 = 80  for x &gt; 0 {     fmt.Print(x, " ")     x = x / 2 - 3 }</pre>	<pre>var x float64 = 0.00001  for x &lt;= 10.0 {     fmt.Println(x)     x++ }</pre>
<pre>var isValid bool  for isValid {     fmt.Println(42)     isValid = true }</pre>	<pre>var x byte = 1  for x &lt; 1000 {     fmt.Println(x)     x *= 2 }</pre> 	<pre>var x byte = 100  for x &gt;= 0 {     fmt.Println(x)     x -= 20 }</pre> 

## Beispiel While Schleifen Fortsetzung

<pre>var x int8 = 120  for x &gt; 0 {     fmt.Println(x)     x += 10 }</pre>	<pre>var x uint64 = 3 for x &gt;= 0 {     fmt.Println(x)     x-- }</pre> 	<pre>var y int8 = 10  for y &lt; 20 {     fmt.Println(y)     y += -2 }</pre>
<pre>var f float64 = 1.0  for f != 2.0 {     fmt.Println(f)     f += 0.1 }</pre> 	<pre>var y int32 = 10  for y &lt; 20 {     y += 2     fmt.Println(y) }</pre>	<pre>var i byte = 3 for i &lt; 10 {     i += i % 2     fmt.Print(i, " ")     i++ }</pre>

## Klassische For Schleife

---

Semantik: Identisch wie die while Schleife mit zusätzlichen Statements

Syntax:   for <init>; <schleifenbedingung>; <post> {  
                  <Code>  
          }

Teilnehmer:

<init> : Dieser Ausdruck wird Initial beim Eintritt der Schleife ausgeführt. (Einmalig)

<schleifenbedingung>: Kennen wir schon, muss ein bool sein

<post>: Dieser Ausdruck wird am Ende jeder Iteration ausgeführt

## Beispiele Klassische For Schleife

```
for ; false ; {  
    fmt.Println("Moin")  
}
```



```
for false {  
    fmt.Println("Moin")  
}
```

```
for ; ; {  
    fmt.Println("Moin")  
}
```



```
for {  
    fmt.Println("Moin")  
}
```



```
for i := 0; i < 5; fmt.Println("Zähler:", i) {  
    i++  
}
```

```
for i, j, k := 0, 1, 2; i < 3; i, j, k = i+1, j+1, k+1 {  
    fmt.Println(i, j, k)  
}
```



# Schleifen Auswahl

---

## Wann for Schleife?

- Iterationsanzahl ist schon bekannt
- Beispiel: 'Gebe mir 100 mal "Hello World" aus'

## Wann while Schleife?

- Iterationsanzahl unbekannt
- Beispiel:

```
var x int = -1
for x < 100 {
    fmt.Print("Geben sie eine große Zahl ein: ");
    fmt.Scan(&x)
}
```