Themenübersicht

- Grundlagen
- Listen und Arrays
- Record-Datentypen
- Pointer
- Rekursion
- Konzepte

Themenübersicht

Grundlagen – Hallo Welt

Ein erstes Go-Programm

Hallo Welt

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      fmt.Println("Hallo Welt")
7  }
```

Ausgabe

```
Hallo Welt
```

Grundlagen – Hallo Welt

Zentrale Komponente jedes Programms: main()

- Einstiegspunkt des Programms
- Jedes lauffähige Programm hat genau eine main()

Die Funktion main()

```
func main() {
    fmt.Println("Hallo Welt")
}
```

Grundlagen – Hallo Welt

Go-Programme sind in Packages organisiert.

- package gibt an, zu welchem Package diese Datei gehört.
- import gibt an, welche anderen Packages benötigt werden.

Package-Deklaration und Imports

```
package main

import "fmt"
```

Anmerkung: Für's Erste werden wir immer im Package main arbeiten. Die Definition eigener Packages erfolgt später.

Grundlagen – Variablen

Variablen

Eine Variable für Zahlen

```
1    var x int
2    x = 42
3    fmt.Println(x)
4
5    x = 23
6    fmt.Println(x)
```

Ausgabe

```
42
23
```

Grundlagen – Variablen

Variablen

Alternative Definition

```
1    y := 55
2    fmt.Println(y)
```

Ausgabe

```
55
```

Grundlagen – Variablen

Variablen

Eine Variable für Zeichenketten

```
1    s := "Hallo String"
2    fmt.Println(s)
3    fmt.Printf("%T\n", s)
```

Ausgabe

```
Hallo String
string
```

Grundlagen – Variablen

Variablen

- Speicherplätze für Daten
- Namen für die Speicheradressen, an denen Daten stehen können

Datentypen von Variablen

- Jede Variable hat einen Typ, z.B. `int` oder `string`.
- Semantik: Beschreibt die Art der Daten.
- Technisch: Bestimmt, wie viel Speicher gebraucht wird.
- Der Typ mus mit der Variable angegeben (*deklariert*) werden oder bei der *Definition* klar sein.

Grundlagen – Variablen

Beispiel: Summe der Zahlen von 1 bis 5

Variablen als Hilfsspeicher

```
1    result := 1
2    result = result + 2
3    result = result + 3
4    result = result + 4
5    result = result + 5
6    fmt.Println("Summe:", result)
```

Ausgabe

```
15
```

Grundlagen – Variablen

Beispiel: Umgang mit Strings

Einlesen von Strings

```
1    var name string
2    fmt.Print("Gib deinen Namen ein: ")
3    fmt.Scanln(&name)
4    fmt.Println()
5
6    fmt.Printf("Hallo %v!\n", name)
```

Ausgabe

```
Gib deinen Namen ein: <NAME>

Hallo <NAME>!
```

Grundlagen – Funktionen

Einfache Funktion

```
1 func Greet() {
2     fmt.Println("Funktion 'greet()' aufgerufen.")
3 }
```

Funktion mit Parametern

```
1 func PrintSum(x, y int) {
2     fmt.Printf("%v + %v == %v\n", x, y, x+y)
3 }
```

Funktion mit Rückgabe

```
1 func ComputeSum(x, y int) int {
2     return x + y
3 }
```

Grundlagen – Funktionen

Funktionen

- Wiederverwendbare Code-Blöcke
- Können Parameter haben.
  - Variablen, deren Wert beim Aufruf bestimmt wird
- Können return-Anweisungen haben.
  - Auch Funktionen haben einen Typ.

Vorteile

- Vermeidung von doppeltem Code
- Berechnungen für unterschiedliche Eingaben

Grundlagen – Kontrollfluss

Einfache for -Schleife

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

for -Schleife mit Schrittweite 2

```
1    for i := 0; i <= n; i += step {
2      fmt.Println(i)
3    }
```

Rückwärts-Schleife

```
1    for ; n >= 0; n-- {
2      fmt.Println(n)
3    }
```

Grundlagen – Kontrollfluss

for -Schleifen

- Wiederholen eine Reihe von Anweisungen.
- Abbruchbedingung steht im Schleifenkopf.
- Zusätzlich: Pre- und Post-Statements
  - Steuern meist Zähler in der Schleife.

while -Schleifen

- Wie for -Schleifen, haben aber keine Zähler.
- Können potenziell zu Endlosschleifen werden.

Grundlagen – Kontrollfluss

while -Schleifen

Funktion mit while -Schleife

```
1  func DisplayDoubles() {
2      input := 1
3
4      for input != 0 {
5          fmt.Print("Bitte eine Zahl eingeben: ")
6          fmt.Scanln(&input)
7          fmt.Printf("%v\n\n", 2*input)
8      }
9  }
```

Grundlagen – Kontrollfluss

Summenberechnung mittels Schleife

Summenfunktion mit Schleife

```
1  func Summe1BisN(n int) int {
2      result := 0
3      for i := 1; i <= n; i++ {
4          result += i
5      }
6      return result
7  }
```

Grundlagen – Kontrollfluss

Besonderheit: Range-Schleife

Range- for -Schleife

```
1  func PrintStringLetters(s string) {
2      for i, v := range s {
3          fmt.Printf("%v: %c, ", i, v)
4      }
5  }
```

Grundlagen – Kontrollfluss

Fallunterscheidungen

- Entscheiden aufgrund von Bedingungen, ob ein Code-Pfad ausgeführt wird.

Fallunterscheidung

```
1  func Judge(n int) {
2      if n == 42 {
3          fmt.Printf("%v ist eine gute Zahl!\n", n)
4      } else {
5          fmt.Printf("%v ist keine gute Zahl!\n", n)
6      }
7  }
```

Grundlagen – Kontrollfluss

Fallunterscheidungen

- Können auch ohne else vorkommen.

Fallunterscheidung ohne else

```
1  func IsLarge(n int) bool {
2      if n > 25 {
3          fmt.Println("Juhuuuu!")
4          return true
5      }
6      return false
7
8  }
```

Grundlagen – Kontrollfluss

Rekursion: Eine Schleife, ohne eine Schleife zu schreiben.

Rekursive Summenfunktion

```
1  func Sum1BisNRekursiv(n int) int {
2      if n <= 1 {
3          return n
4      }
5      return n + Sum1BisNRekursiv(n-1)
6  }
```

Themenübersicht

Themenübersicht

Grundlagen

Listen und Arrays

Record-Datentypen

- Definition eigener Datentypen
- Structs: Spezialisierte eigene Datentypen
- Methoden zur Manipulation von Structs
- Beispiel: Umbau/Verbesserung von Tic Tac Toe

Pointer

Rekursion

Konzepte

Themenübersicht

Themenübersicht

Grundlagen

Listen und Arrays

Record-Datentypen

Pointer

Rekursion

- Funktionen, die sich selbst aufrufen
- Beispiele
- Projekte: Logikrätsel
- Ausblick: Such- und Sortieralgorithmen, Baumstrukturen

Konzepte

Themenübersicht