Programmier-Einführung mit Go

Listen

Reiner Hüchting

4. Dezember 2025

Listen – Überblick

- Listen
  - Arrays und Slices
  - Mehrdimensionale Arrays

Arrays und Slices

Arrays

- Basis-Datentyp für Listen von Elementen.
- Kommt in vielen Programmiersprachen vor.
- I.d.R. feste Größe/Länge und nur ein Element-Datentyp.
- Elemente liegen zusammenhängend im Speicher.

Arrays und Slices

Arrays

- Basis-Datentyp für Listen von Elementen.
- Kommt in vielen Programmiersprachen vor.
- I.d.R. feste Größe/Länge und nur ein Element-Datentyp.
- Elemente liegen zusammenhängend im Speicher.

Slices in Go

- Flexiblerer Listen-Datentyp.
- *Slices* sind *Views* auf *Arrays*.
  - Jede Slice hat ein zugrunde liegendes Array.
  - Mehrere Slices können auf das gleiche Array zeigen.

Arrays und Slices

Definition eines Arrays

```
1  func Example_arrayWithZeros() {
2    var a [5]int
3
4    for i := 0; i < len(a); i++ {
5      a[i] = i
6    }
7
8    fmt.Println(a)
9
10   // Output:
11   // [0 1 2 3 4]
12 }
```

Arrays und Slices

Initialisierung eines Arrays

```
1 func Example_arrayWithValues() {
2   b := [5]int{1, 2, 3, 4, 5}
3
4   fmt.Println(b)
5
6   // Output:
7   // [1 2 3 4 5]
8 }
```

Arrays und Slices

Leere Slice

```
1  func Example_emptySlice() {
2    var a []int
3
4    fmt.Println(len(a))
5    fmt.Println(a)
6
7    // Output:
8    // 0
9    // []
10 }
```

Arrays und Slices

Slice mit Werten

```
1  func Example_sliceWithValues() {
2    b := []int{1, 2, 3, 4, 5}
3
4    fmt.Println(len(b))
5    fmt.Println(b)
6
7    // Output:
8    // 5
9    // [1 2 3 4 5]
10 }
```

Arrays und Slices

Teil-Auschnnitt einer Slice

```
1  func Example_subSlice() {
2    a := []int{1, 2, 3, 4, 5}
3    b := a[1:3]
4
5    fmt.Println(a)
6    fmt.Println(b)
7
8    // Output:
9    // [1 2 3 4 5]
10   // [2 3]
11 }
```

Arrays und Slices

Verändern einer Slice

```
1  func Example_modifySubSlice() {
2    a := []int{1, 2, 3, 4, 5}
3    b := a[1:3]
4
5    b[0] = 99
6
7    fmt.Println(a)
8    fmt.Println(b)
9
10   // Output:
11   // [1 99 3 4 5]
12   // [99 3]
13 }
```

Arrays und Slices

Append-Funktion

```
1  func Example_append() {
2    a := []int{}
3
4    a = append(a, 1)
5    a = append(a, 2)
6    a = append(a, 3)
7
8    fmt.Println(a)
9
10   // Output:
11   // [1 2 3]
12 }
```

Arrays und Slices

Make-Funktion

```
1 func Example_make() {
2   a := make([]int, 5)
3
4   fmt.Println(a)
5
6   // Output:
7   // [0 0 0 0 0]
8 }
```

Mehrdimensionale Arrays

Mehrdimensionale Arrays

- Listen können auch mehrere Dimensionen haben.
- Ansatz: Listen von Listen.

Mehrdimensionale Arrays

2x2-Matrix

```
1  func Example_matrix() {
2    a := [2][2]int{
3      {1, 2},
4      {3, 4},
5    }
6
7    fmt.Println(a[0])
8    fmt.Println(a[1])
9    fmt.Println(a[0][0])
10   fmt.Println(a[1][1])
11
12   // Output:
13   // [1 2]
14   // [3 4]
15   // 1
16   // 4
17 }
```

Mehrdimensionale Arrays

Schleife über Matrix

```
1  func Example_loopMatrix() {
2    a := [2][2]int{
3      {1, 2},
4      {3, 4},
5    }
6
7    for i := 0; i < len(a); i++ {
8      for j := 0; j < len(a[i]); j++ {
9        fmt.Print(a[i][j])
10     }
11     fmt.Println()
12   }
13
14   // Output:
15   // 12
16   // 34
17 }
```

Mehrdimensionale Arrays

Schleife über Spalte

```
1  func Example_loopMatrixColumn() {
2    a := [2][2]int{
3      {1, 2},
4      {3, 4},
5    }
6
7    for i := 0; i < len(a); i++ {
8      fmt.Print(a[i][1])
9    }
10
11   // Output:
12   // 24
13 }
```