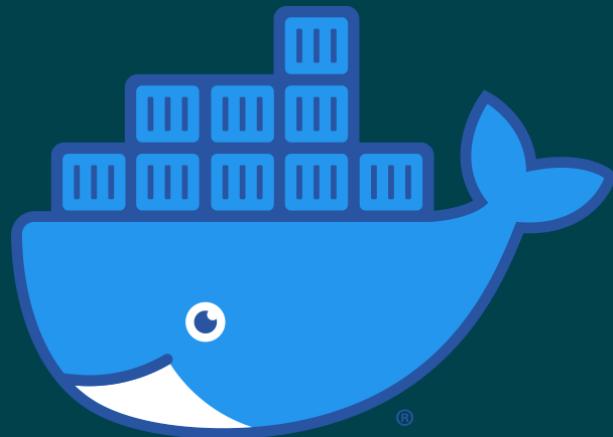




# EINSTIEG IN GO STANDARD-LIBRARY UND ÖKOSYSTEM





**467.882** ZEILEN CODE  
**2.130** CONTRIBUTORS



=CO



# 3 GRÜNDE FÜR GO

- 1 Einfach
- 2 Mächtig
- 3 Langweilig





# HELLO GOPHER

```
package main

import "fmt"

func main() {
    fmt.Println("Hello Gopher!")
}
```

## Ausführen

```
go build hellogopher.go // 1. Code kompilieren
./hellogopher           // 2. Binary ausführen
```

```
go run hellogopher.go   // Code kompilieren und ausführen
```



# ENTWICKLUNG



JetBrains  
GoLand



Visual  
Studio Code



Vim Go



# 5 FAKTEN ZU GO

- 1 statisches Typsystem
- 2 Garbage Collection
- 3 keine Vererbung
- 4 Concurrency eingebaut
- 5 native Ausführung

Linux, Win, z/OS, 386, amd64, ARM, wasm, ...





# VARIABLEN, SLICES, SCHLEIFEN

```
1 // Variable
2 var frank string = "Frank"
3 claire := "Claire"
4
5 // Array (fixe Länge)
6 namesArray := [3]string{frank, claire, "Zoe"}
7
8 // Slice (variable Länge)
9 namesSlice := make([]string, 2)
10 namesSlice[0] = frank
11
12 // Schleife
13 for i, name := range namesSlice {
14     fmt.Println("Hello " + name + "!")
15 }
```



# STRUCT STATT KLASSE

```
1 type Congressman struct {
2     Name string
3 }
4
5 func main() {
6     c := Congressman{Name: "Peter Russo"}
7     fmt.Println("Hello " + c.Name + "!")
8 }
```



# FUNCTION RECEIVER STATT INSTANZMETHODE

```
1 type Congressman struct {
2     Name string
3 }
4
5 func (c Congressman) swearOathOfOffice() {
6     fmt.Printf("I, %v, swear to serve the USA.", c.Name)
7 }
8
9 func main() {
10    c := Congressman{Name: "Peter Russo"}
11    c.swearOathOfOffice();
12 }
```



# INTERFACE

```
1 type Greeter interface {  
2     greet()  
3 }  
4  
5 func passBy(c1 Greeter, c2 Greeter)  
6     c1.greet()  
7     c2.greet()  
8 }  
9  
10 func main() {  
11     c := Congressman{Name: "Frank U."  
12     e := Enemy{}  
13     passBy(c, e)  
14 }
```

```
type Congressman struct {  
    Name string  
}  
  
func (c Congressman) greet() {  
    fmt.Println("Hello", c.Name)  
}
```

```
type Enemy struct{ }  
  
func (e Enemy) greet() {  
    fmt.Println("Go to hell!")  
}
```



# ZU EINFACH?



# STRUCT EMBEDDING

```
1 type Congressman struct {
2     Name string
3 }
4
5 type President struct {
6     Congressman // Embedded
7
8     NuclearWeaponCode string
9 }
10
11 func main() {
12     p := President{NuclearWeaponCode: "123"}
13     p.Name = "Frank Underwood"
14     p.swearOathOfOffice()
15 }
```



# FEHLER

```
1 // Fehler als Rückgabewert
2 func (c Congressman) bribe(amount float64) error {
3     if c.Name != "Peter Russo" {
4         return errors.New("Not corrupt!")
5     }
6     c.AccountBalance += amount
7     return nil
8 }
9
10 func main() {
11     c := Congressman{Name: "Jackie Sharp", AccountBalance: -10.0}
12
13     // Fehler behandeln
14     err := c.bribe(5000.0)
15     if err != nil {
16         fmt.Printf("%v is not bribable.", c.Name)
17     }
18 }
```



# GENERICSS

```
func printSliceOfInts(numbers []int) {  
    for _, num := range numbers {  
        fmt.Println(num, " ")  
    }  
}
```

Kommen in Go 1.18 im Februar 2022!

```
func printSliceOfStrings(strings []string) {  
    for _, num := range strings {  
        fmt.Println(num, " ")  
    }  
}
```



# MÄCHTIG



# CONCURRENCY

**GOROUTINE**

leichtgewichtiger Thread

**CHANNEL**

Kanal für Nachrichten, dient der



Kommunikation



Synchronisation  
von Goroutinen



# GOROUTINE

```
1 func HelloCongressman(name string) {  
2     fmt.Println("Hello Congressman", name)  
3 }  
4  
5 func main() {  
6     go HelloCongressman("Russo")  
7 }
```



# CHANNEL

```
1 // Deklarieren und Initialisieren
2 c := make(chan int)
3
4 // Nachricht über Channel senden
5 c <- 1
6
7 // Nachricht von Channel empfangen,
8 // "Pfeil" gibt Richtung des Datenfluss an
9 value = <-c
```



# DEBATTE MIT GOROUTINEN UND CHANNELS

```
1 func speaker(name string, debate chan int) {
2     for {
3         microphone := <-debate // Auf Mikro warten (Nachricht empfangen)
4
5         fmt.Printf("Turn %v: %v says '%v'\n", microphone, name, randomAnswer())
6         time.Sleep(400 * time.Millisecond)
7
8         microphone++
9         debate <- microphone // Mikro zurückgeben (Nachricht senden)
10    }
11 }
12
13 func main() {
14     debate := make(chan int) // Channel deklarieren und initialisieren
15
16     go speaker("Jackie", debate)
17     go speaker("Frank", debate)
18
19     microphone := 1
20     debate <- microphone           // Mikro geben und Diskussion starten
21     time.Sleep(2 * time.Second)    // Dauer der Diskussion
22     <-debate                      // Mikro nehmen und Diskussion beenden
23 }
```



# DEBATTE MIT GOROUTINEN UND CHANNELS

```
> go run debate.go
Turn 1: Frank says 'You liar.'
Turn 2: Jackie says 'Back off.'
Turn 3: Frank says 'Back off.'
Turn 4: Jackie says 'My point.'
Turn 5: Frank says 'You liar.'
Turn 6: Jackie says 'You're wrong.'
```



# STANDARDBIBLIOTHEK

# Tests

# HTTP(2) Server und Router

# JSON

# Logging



# EINFACH LANGWEILIG ZU EINFACH? MÄCHTIG

# Variablen,

Slices,

Schleifen

# Struct

# Struct

Embedding

# Fehler

# Generics

# Interface

# Goroutine

# Channel

#

Standardbibliothek



# GO LIEBT



Microservices



Serverless Functions



Kommandozeilen-Tools



DevOps und Cloud



# COMMUNITY

- # Gophers Slack
- # Go Time Podcast
- # Go Blog
- # Twitter Gophers List



# DEMO





**JAN STAMER**  
Solution Architect  
[jan.stamer@comdirect.de](mailto:jan.stamer@comdirect.de)

