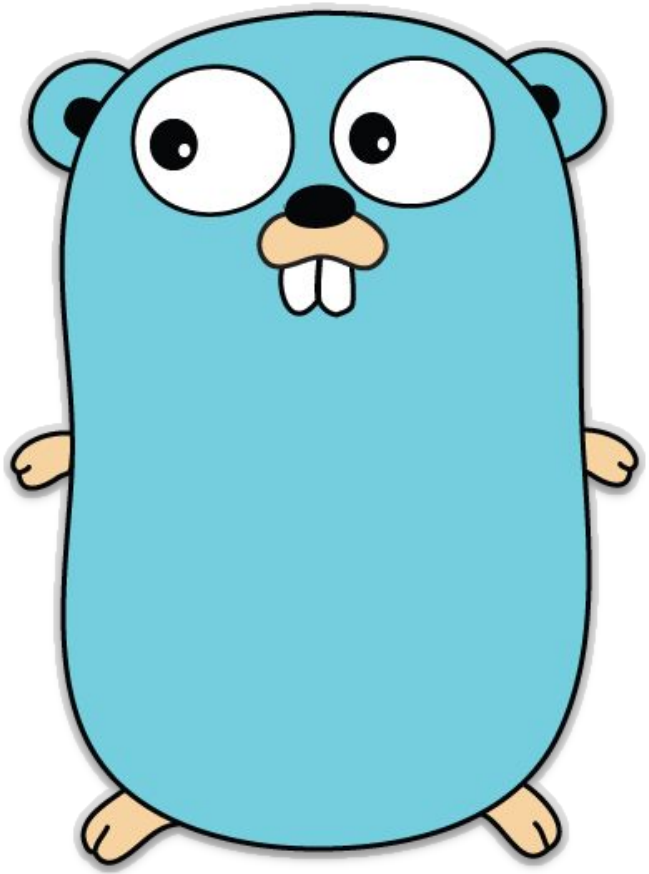
Go für Programmierer

Michael Stapelberg
2018-05-11



Inhalt

- **Schnellrückblick: Syntax (5 Minuten)**
- Live Demo: setup, coding, testing
- Einzigartiges: Goroutines, Channels, Interfaces
- Live Demo: concurrency, profiling

Syntax: Datentypen

```
bool
string

(u)int                  // native Größe
(u)int(8|16|32|64) // uint8 auch “byte”
      float(32|64)

m := make(map[string]os.FileInfo)
```

Syntax: Arrays/Slices

```
var buffer [24]byte

// Zweiten uint32 laden:
binary.LittleEndian.Uint32(buffer[4:8])
```

Syntax: Struct Types

```
type notebook struct {
  model          string
  width, height int     // pixels
}

xps := notebook{
  model:  "Dell XPS 13 9360 (2017)",
  width:  3200,
  height: 1800,
}
```

Syntax: Funktionen und Methoden

```
func add(a, b int) int {
  return a + b
}

func (n *notebook) prettyPrint() {
  fmt.Println(n.model)
  fmt.Printf("Screen: %d x %d px",
             n.width, n.height)
}
```

Syntax: Packages und Exporting

```
package ioutil

// ReadFile reads the file named by filename
// and returns the contents.
func ReadFile(filename string) ([]byte, error) {
   // …
}
```

Package ioutil

```
import "ioutil"
```


Overview ▾

Index ▾

Package files

ioutil.go

func **ReadFile**

```
func ReadFile(filename string) ([]byte, error)
```

ReadFile reads the file named by filename and returns the contents.

Live Demo: hello (ca. 10 Minuten)

- setup
- coding
- testing

Einzigartiges: Goroutines

```
fmt.Println(time.Now())       // 2009-11-10 23:00:00
time.Sleep(1*time.Second)
fmt.Println(time.Now())       // 2009-11-10 23:00:01
```

Einzigartiges: Goroutines

```
fmt.Println(time.Now())       // 2009-11-10 23:00:00
time.Sleep(1*time.Second)
fmt.Println(time.Now())       // 2009-11-10 23:00:01

fmt.Println(time.Now())       // 2009-11-10 23:00:01
go time.Sleep(1*time.Second)
fmt.Println(time.Now())       // 2009-11-10 23:00:01
```

Einzigartiges: Goroutines

```
var wg sync.WaitGroup
for _, fn := range files {
  wg.Add(1)
  fn := fn      // fn kopieren, range überschreibt
  go func() {
    worker(fn) // Rückgabewert ignoriert
    wg.Done()
  }()
}
wg.Wait()
```

Einzigartiges: Channels

- goroutine-safe (keine Mutexes nötig)
- speichern/senden Werte an Goroutinen
- FIFO-Semantik
- blockiert/entblockt Goroutinen

Einzigartiges: Channels (“task queue”)

```
ch := make(chan string, 2)
for n := 0; n < 6; n++ {
  go workloop(ch)
}
for _, fn := range files {
  if validate(fn) {
    ch <- fn
  }
}
// …
```

```
func workloop(ch) {
 for fn := range ch {
   worker(fn)
 }
}
```

Einzigartiges: Channels (“semaphor”)

```
var sem = make(chan struct{}, 1)
func tmp(w http.ResponseWriter, r *http.Request) {
  sem <- struct{}{}          // (blockingly) acquire
  defer func() { <-sem }() // release
  // clear /tmp...
}
func main() {
  http.Handle("/cleartmp", tmp)
  log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Einzigartiges: Interfaces

```
type blobReader interface {
  ReadBlob(key string) ([]byte, error)
}

type attachmentServer struct {
  blobReader blobReader
}

attachmentServer{blobReader: aws.NewBlob(...)}
attachmentServer{blobReader: gcp.NewBlob(...)}
```

Live Demo: debpkg (ca. 20 Minuten)

- concurrency
- profiling

Wie geht’s weiter?

- Beginner’s resources (golang slack)

- Community:
  https://golang.org/help/

- Nächst-größeres Feature: Versionen
  https://research.swtch.com/vgo

Ende

Fragen?

Feedback geben