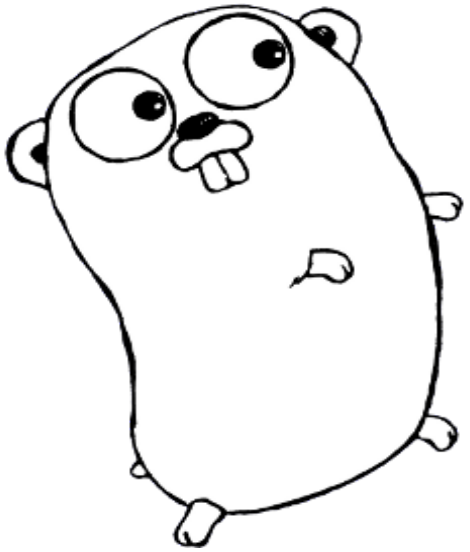
Programmiersprachen im Multicore Zeitalter

Google GO

Hendrik Donner



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

2 Jahre alte Sprache von Google

- systemnah
- C ähnlich, imperativ
- wenig OOP
- garbage collection
- native Nebenläufigkeit

Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Hauptentwickler

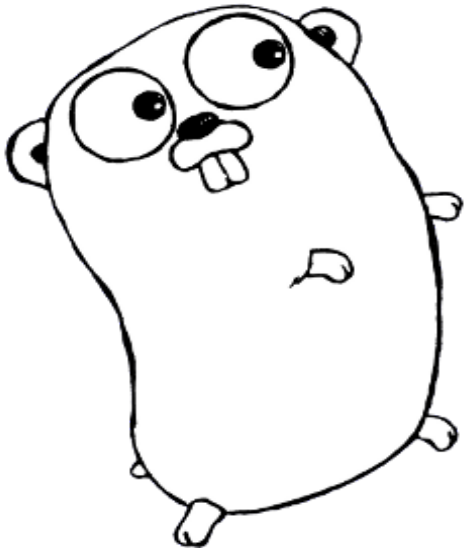
- Ken Thompson
  - Multics, UNIX, Plan 9, UTF-8, B, C
- Rob Pike
  - Plan 9, UTF-8
- Robert Griesemer
- Ian Taylor (gcc backend)

Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Tools

- 2 Compiler: gccgo und gc
- gofmt: formatiert Code automatisch
- gdb zum debuggen
- godoc: wie javadoc
- gorun: go Code zum Skripten nutzen



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

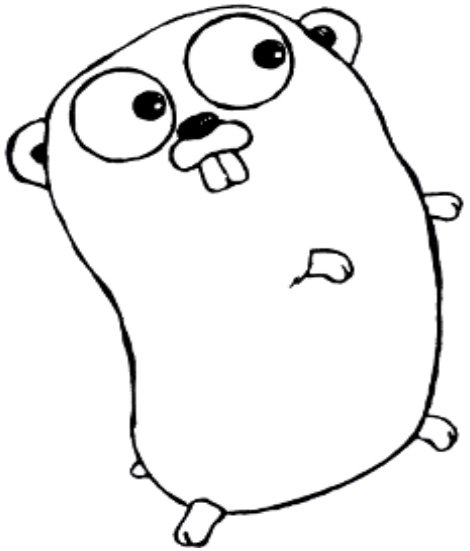
- Hello World:

```
package main
import  fmt "fmt" // Paket für printf etc.
func main() {
        fmt.Printf("Hello, world")
}
```

- Kompilieren und Linken:

```
$ 6g helloworld.go
$ 6l helloworld.6
$ ./6.out
Hello, world
```

6 für 64bit
3 für i386 (32bit)
5 für ARM
src: *.go, UTF-8 Textdatei



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Syntax:

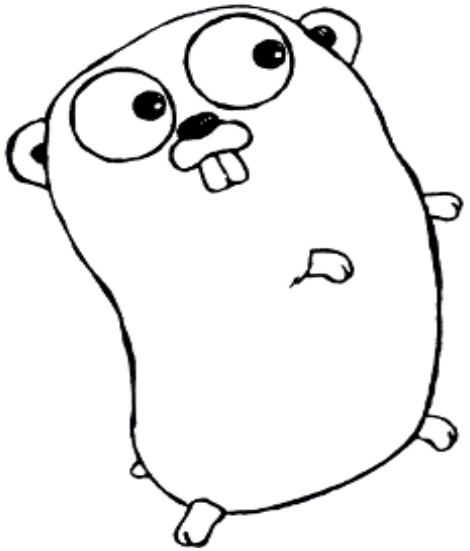
- Variablen und Konstanten:

```
var sum int
const pi = 3.1415... //ohne Typ, maximale Genauigkeit
var x, y *int  // 2 pointer!
var a [10]int  //array mit 10 Elementen, Länge Teil des Types
```

- Gruppierungen:

```
var (
        pi float64
        gamma float64
)
```

Geht auch mit func, import, const



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Syntax:

- Strukturen:

```
type Vector3D struct {
        x, y, z float64
}
```

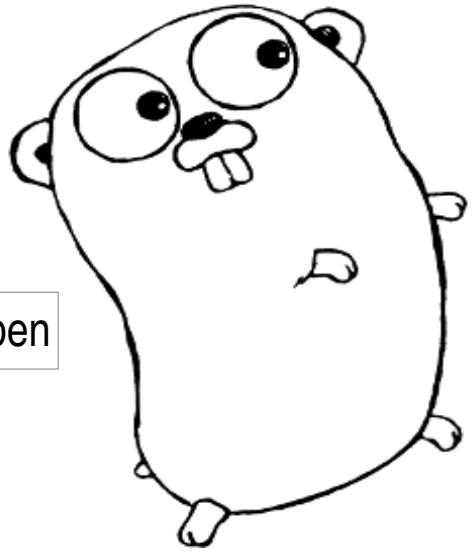
- Mehrfache Zuweisung:

```
a,b = b,a //swap
```

- Dynamische Typerkennung:

```
var s string = " " //direkte Initialisierung
var s = " "        //kann nur string sein, da " "
s := " "       //s ebenfalls string, nur in Funktionen erlaubt
```

Trotzdem statische Typen



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Syntax:

- Typen: feste Größen
- int32 → 32 bit
- float64 → 64 bit
- byte: alias auf uint8
- int → 32 oder 64 bit
- string eigener Typ
- Arraylänge Teil des Types

Architekturspezifisch

Go Gopher logo created by Renée French

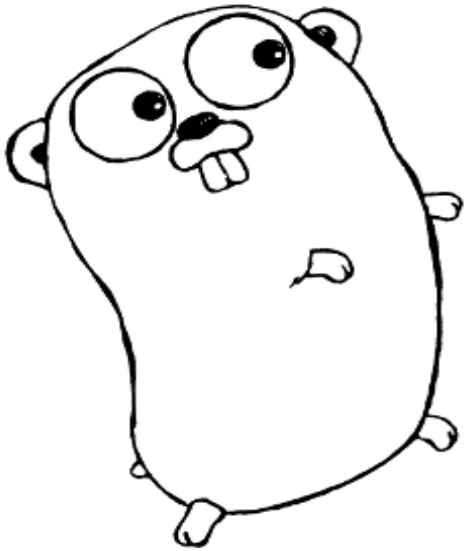
Programmiersprachen im Multicore Zeitalter

Syntax:

- Kontrollstrukturen:
  - for:

```
for i :=0; i<100; i++ {
        ...
}
for { //Endlos
        …
}
```

Keine Klammern um Bedingung
Codeblock allerdings immer



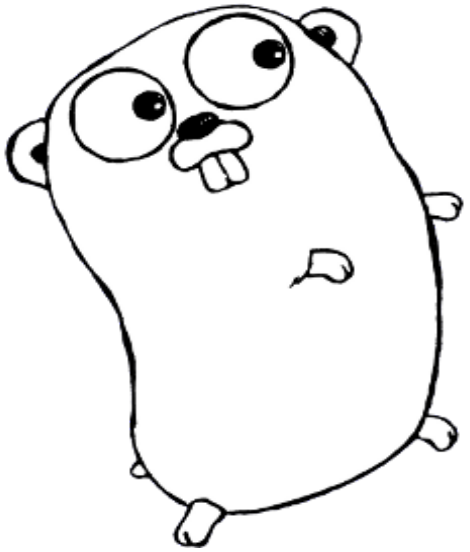
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Syntax:

- Kontrollstrukturen:
  - for:

```
for x < y { //wie while
        ...
}
//für bestimmte Typen wie z.B. map:
var m map[string]int // map[keytype]valuetype
for index, value := range m { //2 Rückgabewerte
        ...
}
```



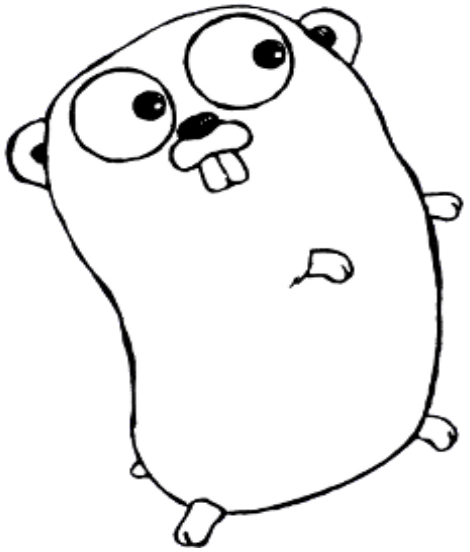
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Syntax:

- Kontrollstrukturen:
  - if:

```
//Das erzwingen des Codeblocks umgeht dangling else
if a < b {
        ...
}
if u < v {
        ...
} else {
        ...
}
```



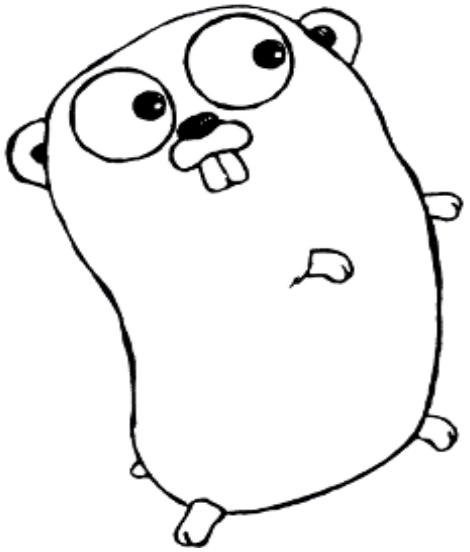
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Syntax:

- Funktionen und Methoden:

```
//der Typ des Rückgabewerts steht hinter der Parameterliste
func Add(x,y int) int {
        return x+y
}
func Sqrt(x float64) (float64, bool) {
        if x < 0 {
                return 0, false
        }
        return math.Sqrt(x), true //Multiple Rückgabewerte
}
```



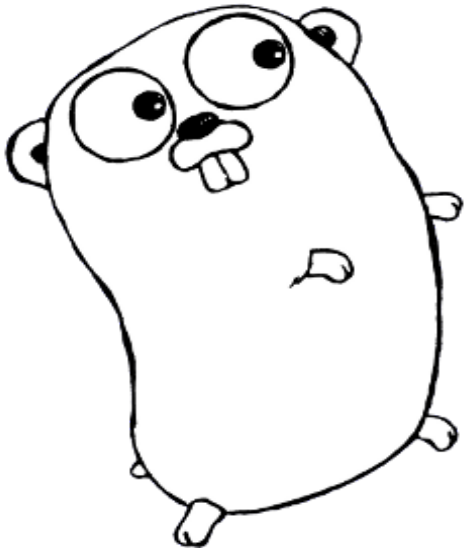
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Syntax:

- Funktionen und Methoden:

```
// Benannte Rückgabwerte
func Sqrt(x float64) (y float64, ok bool) {
        if x < 0 {
                y, ok = 0, false
        } else {
                y, ok = math.Sqrt(x), true
        }
        return
}
```



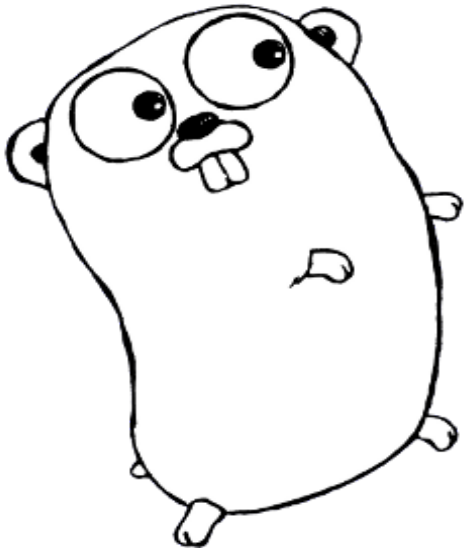
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Syntax:

- Funktionen und Methoden:

```
func (p Point2D) Length() int {
        return math.Sqrt(p.x*p.x + p.y*p.y)
}
...
SomePoint.Length()
```

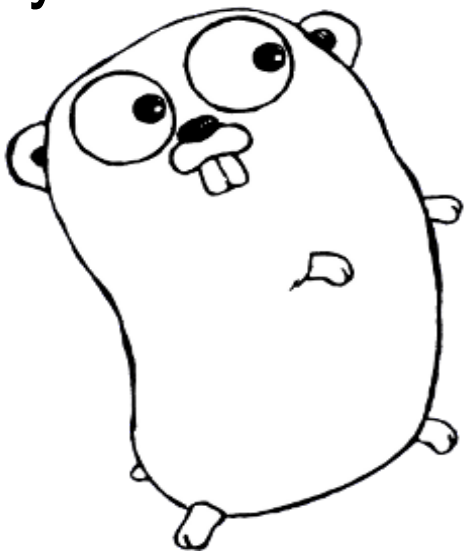


Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Motto: “Do not communicate by sharing memory. Instead, share memory by communicating.”
- Speichergekoppelt, aber Nachrichtenbasiert
- 2 Konzepte: goroutine und channel Typ



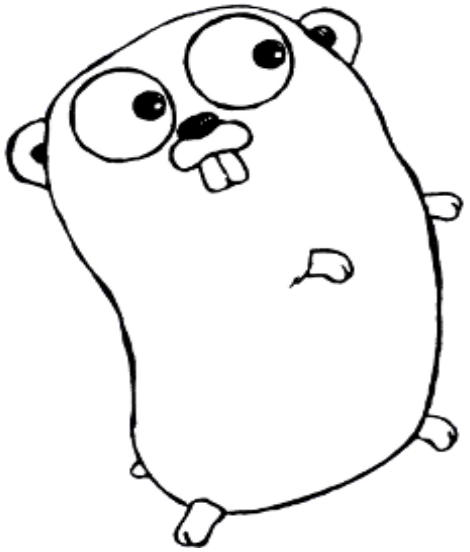
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- goroutine: nebenläufig ausgeführte Funktion
- kein klassischer Thread
- leichtgewichtig (nur eigener Stack)
- compilerspezifische Umsetzung:
  - gccgo: ein Thread pro goroutine
  - gc: schedulet goroutines auf Threads

GOMAXPROCS / runtime.GOMAXPROCS(n)



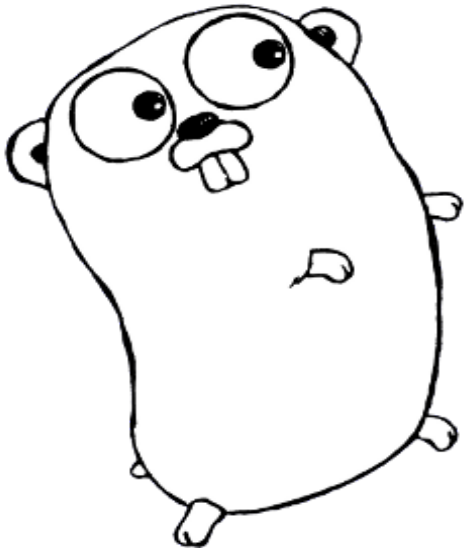
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Beispiel:

```
package main
import (
        fmt "fmt"
        time "time"
)
func Concurrent(thread int, minutes int64) {
        time.Sleep(minutes*60*1e9) // Nanosekunden
        fmt.Println("Thread", thread, "has finished") //Java ähnlich
}
```



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

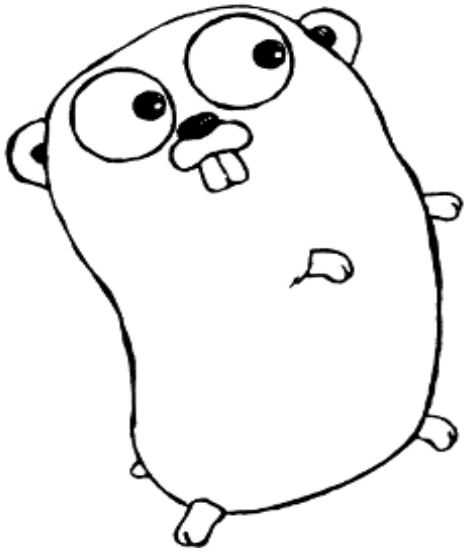
Nebenläufigkeit:

- Beispiel: Teil 2:

```
func main() {
        go Concurrent(1, 4)
        go Concurrent(2, 2)
        fmt.Println("Main waiting...")
        time.Sleep(5*60*1e9) // 5 Minuten
    }
```

- Ausgabe:

```
Main waiting...
Thread 2 has finished
Thread 1 has finished
```



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- channel Typ:

```
var ch chan int //channel für integer
var chch chan chan int //channel für channel für integer
type message struct {
        data int
        answer chan int
}
msg chan message //channel für eigene Typen
```

Go Gopher logo created by Renée French

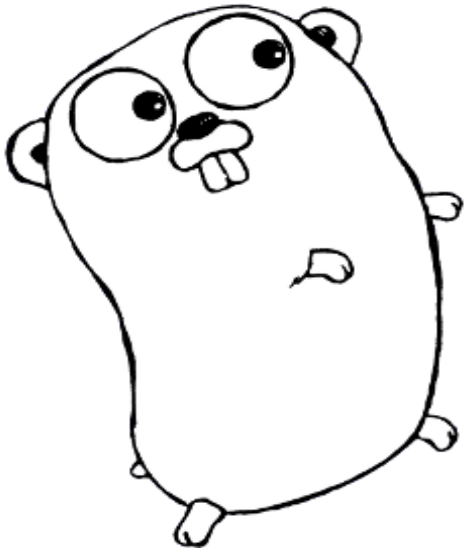
Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- channel Typ:
  - Erzeugung:

    ch := make(chan int) //ähnlich wie new, nur kein pointer
  - Senden/Empfangen:

    ch := make(chan int)
    //senden
    ch <- 1 //1 auf dem channel senden, blockiert bis empfangen wird
    ch <- 2 //2 senden
    ...
    //empfangen (sinnvollerweise in einer anderen goroutine)
    res := <- ch //res = 1, erst jetzt wird 2 gesendet



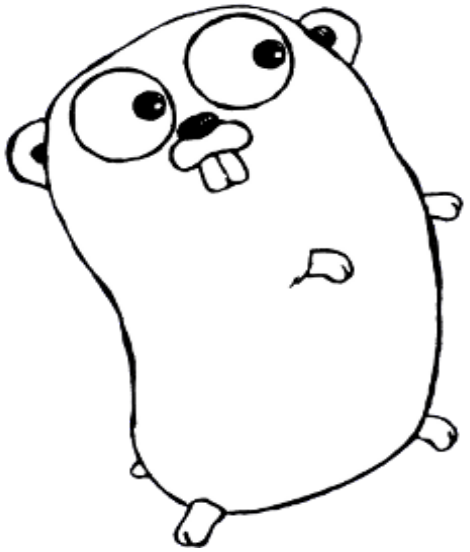
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- channel Typ und goroutine: Beispiel:

```
func Producer(out chan int, done chan bool) {
        for i:=0; i < 100; i++ {
                fmt.Println("Produced: ", i)
                out <- i //blockiert, bis konsumiert wird
                fmt.Println("Consumed: ", i)
        }
        done <- true
}
func Consumer(in chan int) {
        for {
                <- in
        }
}
```



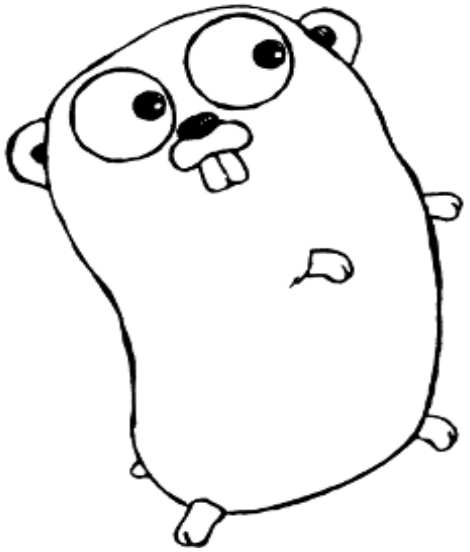
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- channel Typ und goroutine: Beispiel Teil 2:

```
package main
import fmt "fmt"
func main() {
        ch := make(chan int)
        done := make(chan bool)
        go Producer(ch, done)
        go Consumer(ch)
        go Consumer(ch)
        <-done //Signal zum beenden, wird verworfen
}
```



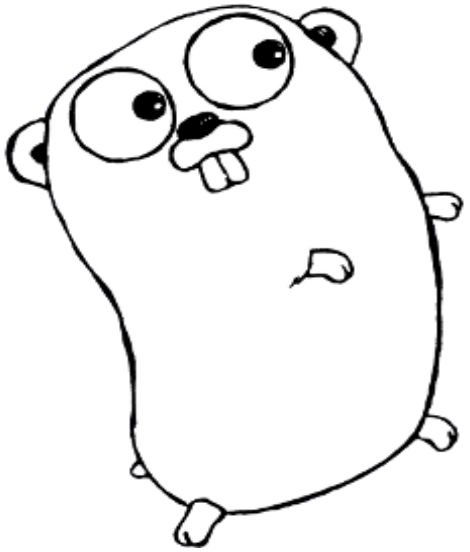
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- channel Typ und goroutine: Beispiel Ausgabe:

  Produced: 1
  Consumed: 1
  ...
  Produced: 99
  Consumed: 99



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

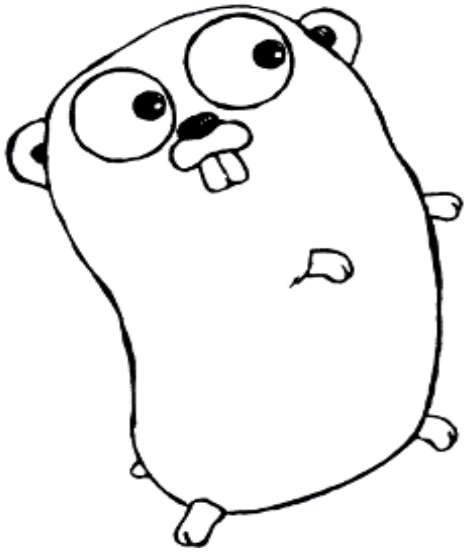
Nebenläufigkeit:

- channel Typ:
  - gerichtet:

    var recv <-chan int //integer channel nur zum empfangen
    var send chan<- int //integer channel nur zum senden
  - Schließen mit close(chan), testen mit:

    c := make(chan int)

    value, ok := <-c
    if ok { //chan offen }



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

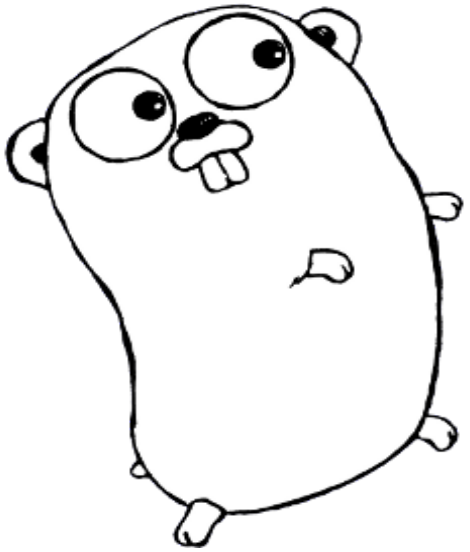
Nebenläufigkeit:

- channel Typ:
  - select: switch für channel:

```
ch1, ch2 := make(chan int),make(chan bool)
select {
        case <-ch1: fmt.Printf("Received from ch1!\n")
        case <-ch2: fmt.Printf("Received from ch2!\n")
        default: fmt.Printf("No ch ready!\n)
}
```

  - buffered: blockiert erst wenn buffer voll

```
ch := make(chan int, 10)//Platz um 10 int zu puffern
```



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Factory:

```
func StartServer() chan<- message {
        ch := make(chan message)
        go Server(ch)
        return ch
}
```

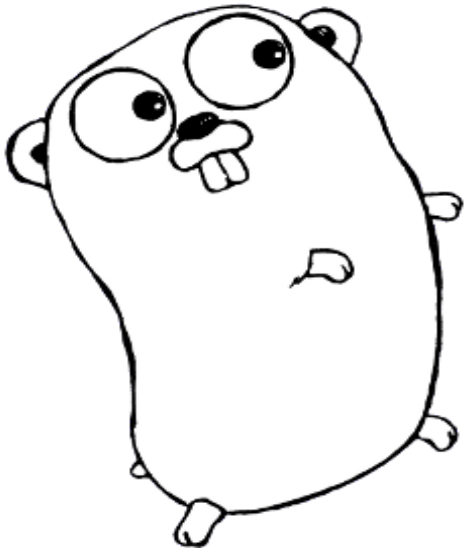
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Timeout:

```
ch, timedout = make(chan int), make(chan bool,1)
go func (){ //anonyme Funktion
        time.Sleep(10*1e9)//10 sek schlafen
        timedout <- true
}() //Parameterübergabe
go DoSometing(ch)
select {
        case <-ch: fmt.Printf("Received from ch!\n")
        case <-timedout: fmt.Printf("Timed out!\n")
}
```



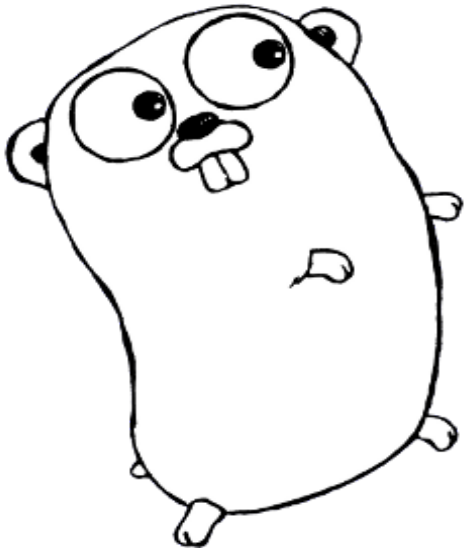
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Pumping:

```
//Random float Generator
for {
        ch <- rand.Float()
}
```



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Pumping – sinnvolleres Beispiel:

```
func (l *List) Iterator() <-chan ListItem {
        ch := make(chan ListItem);
        go func () {
                for i := 0; i < l.size; i++ {
                        ch <- l.items[i]
                }
        } ();
        return ch
}
...
```

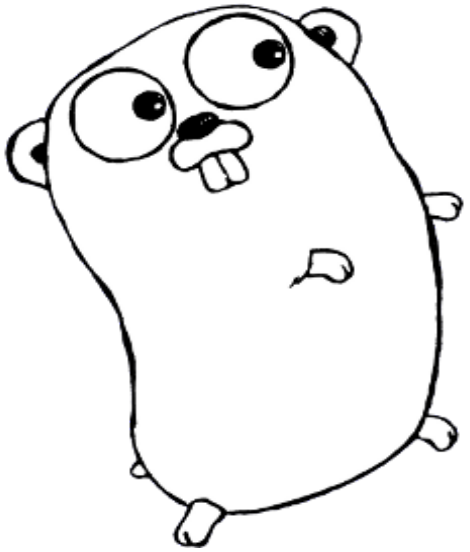
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Pumping – sinnvolleres Beispiel Teil 2:

```
for value := range List.Iterator() {
        ...
}
```



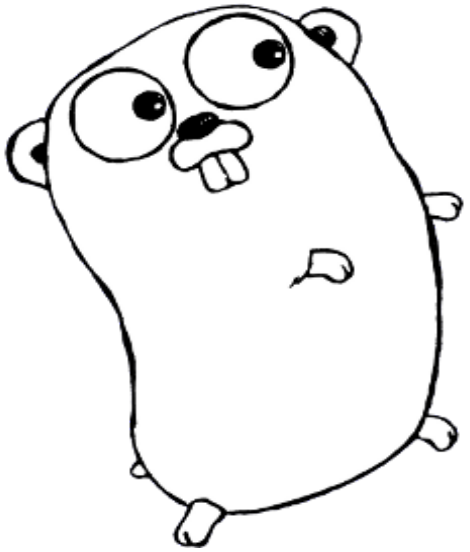
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Futures:

```
func FutureAdd(x, y int)  <-chan int {
        ch := make(chan int)
        go func() {
                ch <- (x+y)
        }()
        return ch
}
...
```



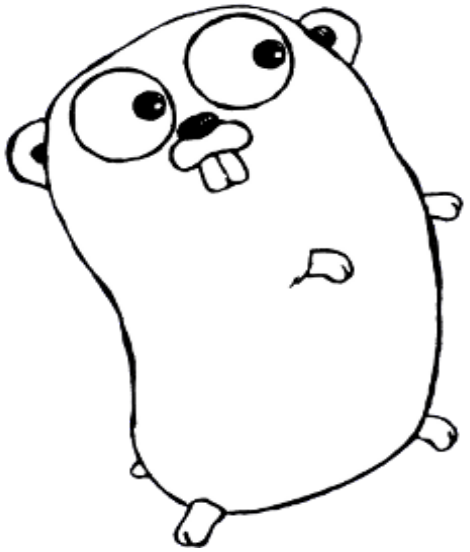
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Futures Teil 2:

```
a := FutureAdd(1,2)
b := FutureAdd(3,4)
... //irgendetwas Anderes berechnen
res <- a //Resultat abholen
```



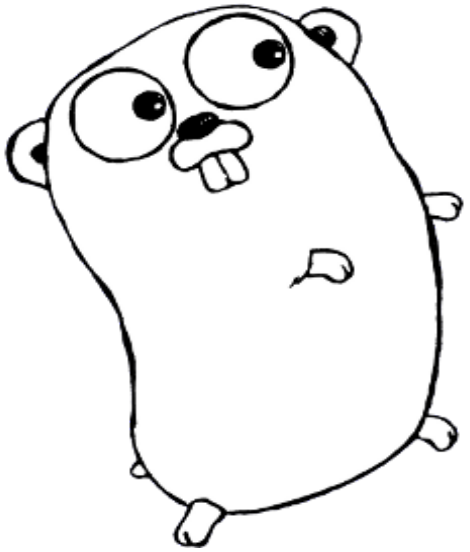
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Chaining:

```
func filter(filt int, in chan int) chan int {
        ch := make(chan int)
        go func(){
                for {
                        if i := <-in; i % filt == 0 {
                                ch <- i
                        }
                }
        }()
        return ch
}
```



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

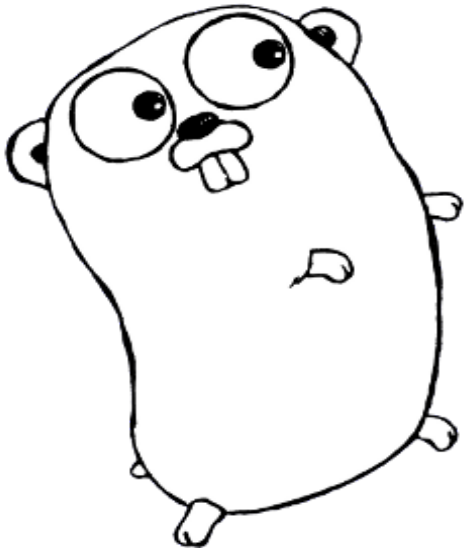
Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Chaining Teil 2:

    div2 := filter(2, in)

    div2or3 := filter(3, div2)

Primzahlsieb mit chaining auf golang.org



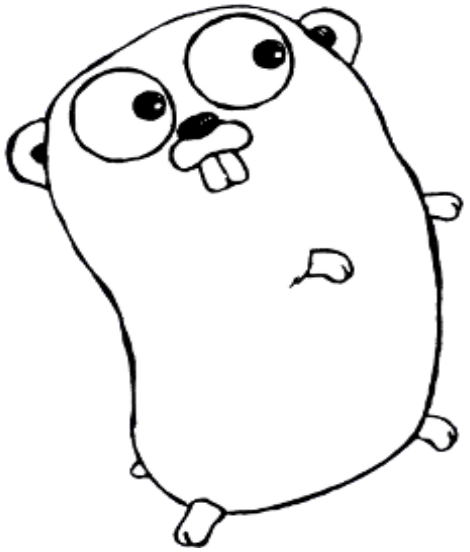
Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Nebenläufigkeit:

- Idiomatische Anwendungen:
  - Multiplexing:

```
type Request struct{
        req String
        reply chan String
}
func Server(in chan Request) {
        for {
                req := <- in
                req.reply <- Compute(req.req)
        }
}
```

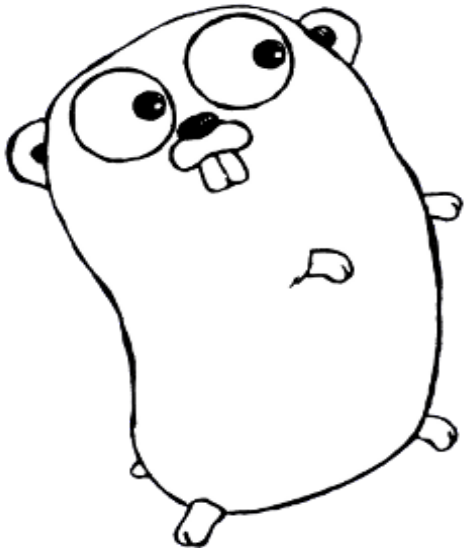


Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Mehr:

- Golang.org
  - Tutorials
  - Codewalks
  - Videos
  - Live Code ausprobieren (Tour of GO)
  - Dokumentation
  - Spezifikation
  - Blog
  - Mailing List



Go Gopher logo created by Renée French

Programmiersprachen im Multicore Zeitalter

Danke schön!
Zeit für Fragen...

Go Gopher logo created by Renée French