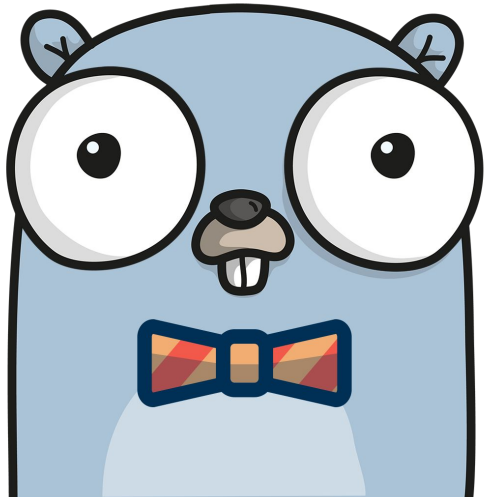
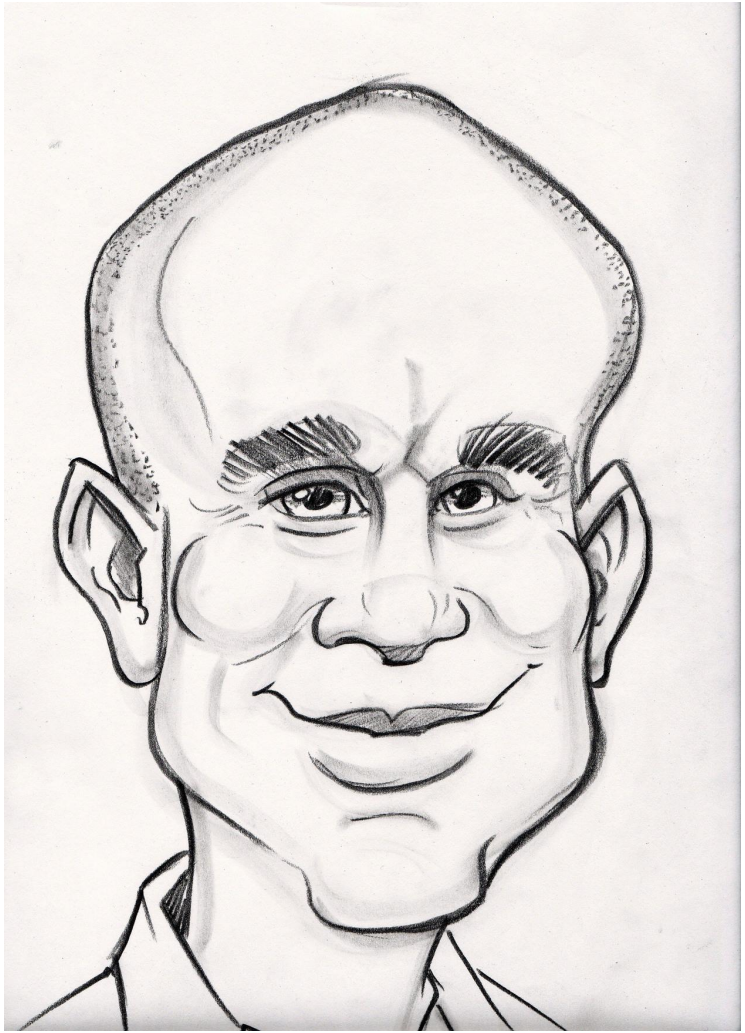
Google

Check Point .:. June 2017



Miki Tebeka

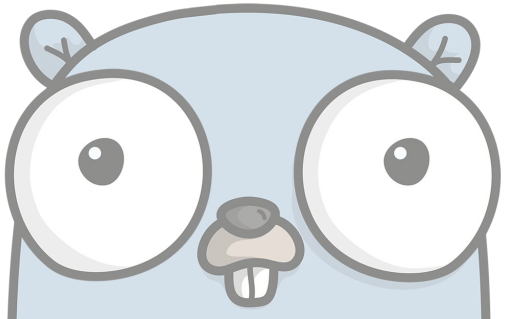


CEO, CTO, UFO ...

353Solutions

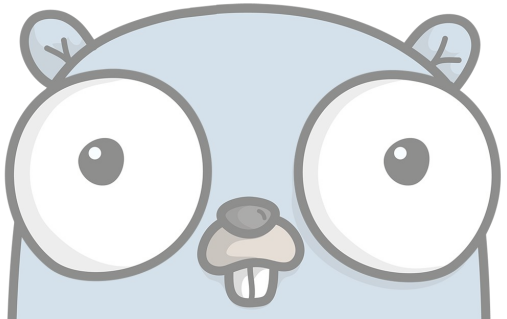
Background

- Developed at Google
  - Robert Griesemer, Rob Pike and Ken Thompson
- Open sourced November 2009
- Version 1 March 2012
  - Currently at 1.8



Notable Users

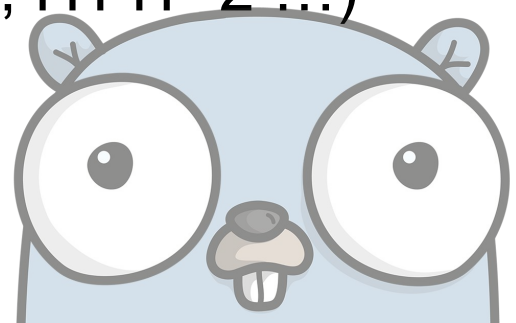
- Google
  - dl.google.com
- Docker is written in Go
- Dropbox
- Facebook
- Netflix
- And more ... (see here)
  - More than 20 Israeli companies



Why Go?

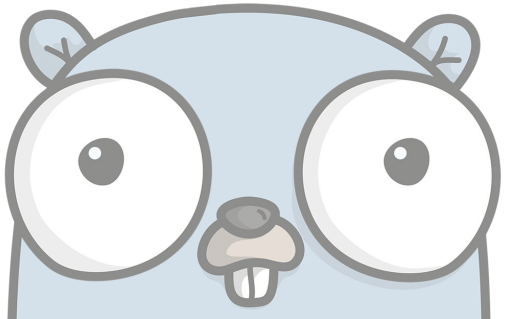
Built for Modern Hardware

- The free lunch is over
  - goroutines
  - channels
- The C10k problem
  - goroutines
  - Production ready HTTP server (TLS, HTTP 2 …)



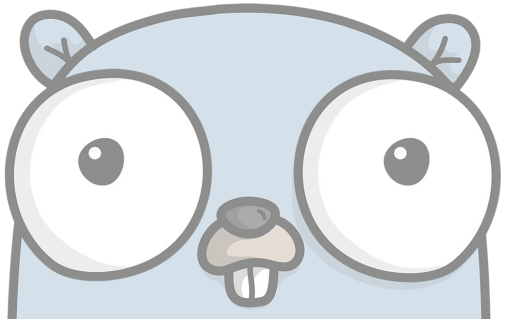
Built for Large Teams

- Small language
  - C based syntax
- Simple language
  - Easy to understand
- Module system
  - Reusability
- Interfaces
  - Modularity



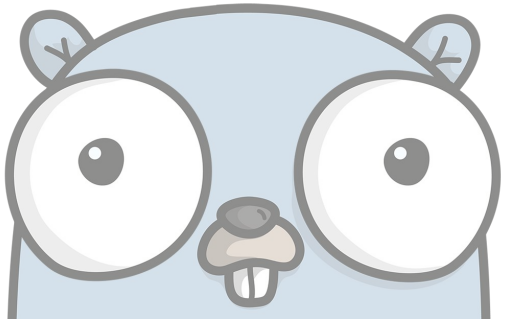
Robust & Productive

- Static types
  - Yet feels dynamic
- Garbage collector
  - But have “unsafe” package :)
- Easy integration with C
- Fast compilation
- Forces you to check errors



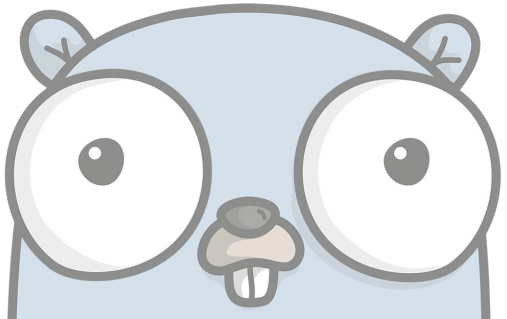
Will Save You Money

- Compiles to static executable
  - Easy deployment
  - Efficient (iron.io went down from 30 servers to 2)
- Stable API
- “go” tool for project management
  - And the upcoming “dep” tool
- Easy to cross compile

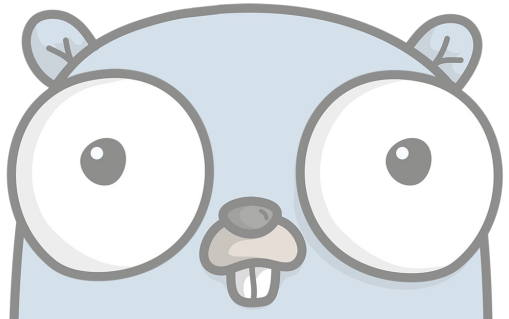


Great Community

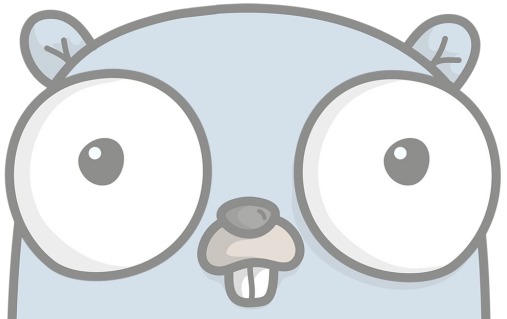
- People will help you
- A lot of reference material
- Conventions



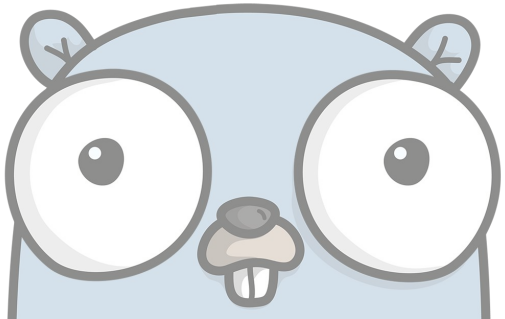
Show Me Some Code



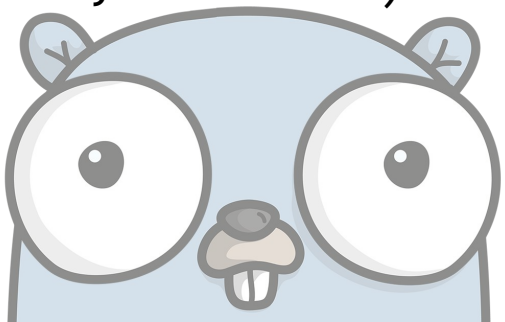
```go
package main

import "fmt"

func main() {
   fmt.Println("Check Point שלום")
}
```



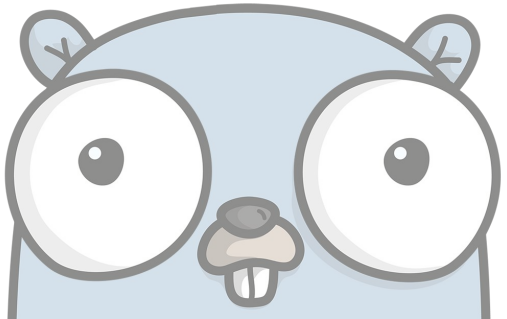
```go
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Check Point שלום")
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```



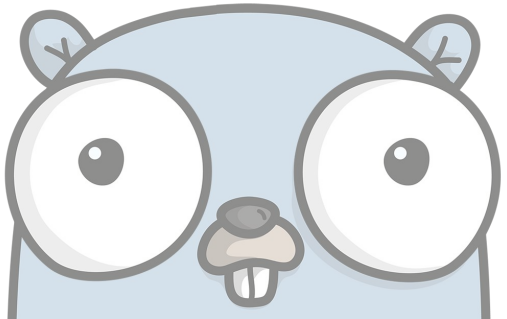
```go
package io

type Reader interface {
    Read(p []byte) (n int, err error)
}

type Writer interface {
    Write(p []byte) (n int, err error)
}

func Copy(dst Writer, src Reader) (written int64, err error)
```



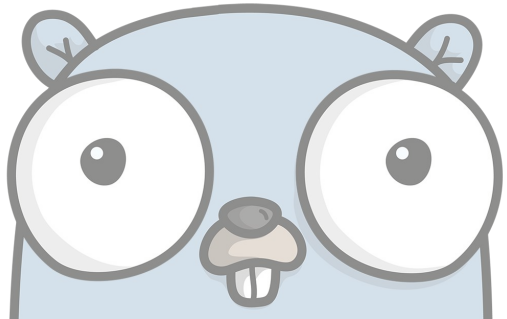
```go
file, err := os.Open("hash.go")
if err != nil {
   log.Fatal(err)
}
defer file.Close()

hash := sha256.New()
io.Copy(hash, file)
fmt.Printf("%x\n", hash.Sum(nil))
```



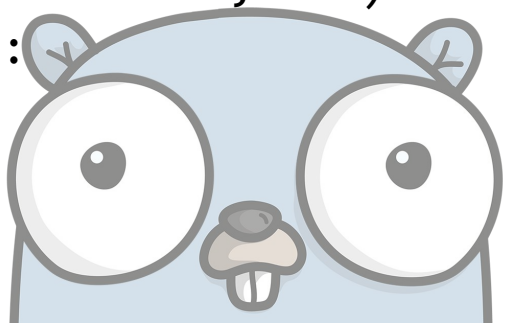
```
resp, err := http.Get("https://www.checkpoint.com/")
if err != nil {
    log.Fatal(err)
}
defer resp.Body.Close()

out, err := os.Create("checkpoint.html.gz")
if err != nil {
    log.Fatal(err)
}
defer out.Close()
gz := gzip.NewWriter(out)
defer gz.Close()
io.Copy(gz, resp.Body)
```



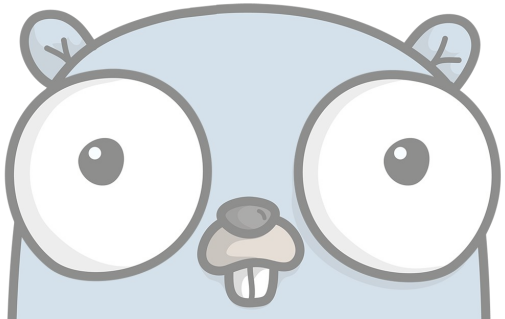
```go
// forward proxies traffic between local socket and remote
// backend
func forward(local net.Conn, remoteAddr string) {
    remote, err := net.Dial("tcp", remoteAddr)
    if err != nil {
        log.Printf("remote dial failed: %v\n", err)
        local.Close()
        return
    }
    go io.Copy(local, remote)
    go io.Copy(remote, local)
}
```



```go
func main() {
    url := "https://www.checkpoint.com"
    out := make(chan *http.Response)
    go fetch(url, out)

    select {
    case resp, ok := <-out:
        if !ok {
            break
        }
        fmt.Printf("got %d from %s\n", resp.StatusCode, url)
    case <-time.After(300 * time.Millisecond):
        fmt.Printf("timeout")
    }
}
```

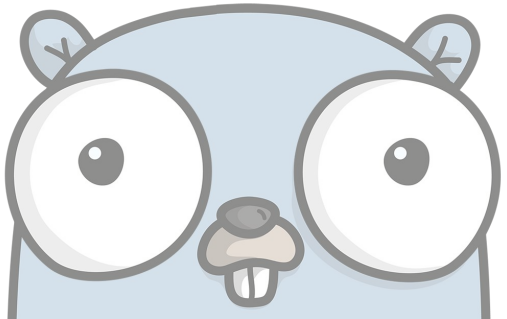


```go
package main

import "fmt"

// #include <math.h>
// #cgo LDFLAGS: -lm
import "C"

func main() {
    v := 16.0
    s := C.sqrt(C.double(v))
    fmt.Printf("sqrt(%f) = %f\n", v, s)
}
```

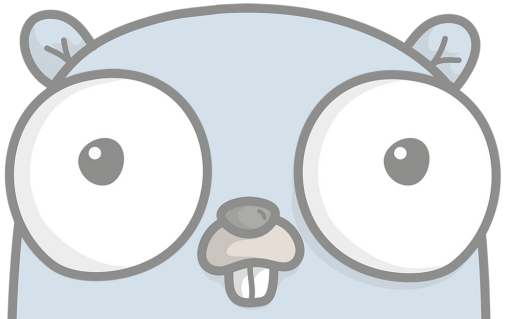


7 BOOM

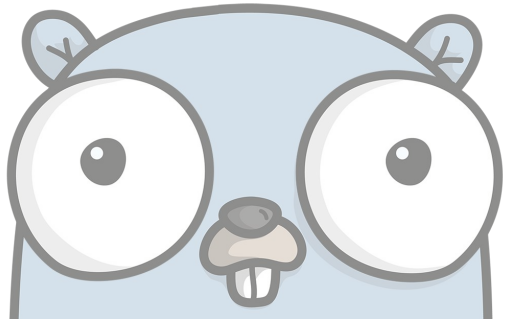
code



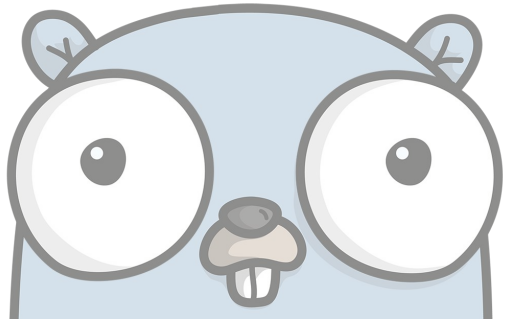
```go
// Player is a player in the 7boom game
type Player struct {
    ID   int
    in   chan int
    out  chan int
    done chan bool
}

// NewPlayer return a new player
func NewPlayer(id int, done chan bool) *Player {
    return &Player{
        ID:   id,
        out:  make(chan int, 1),
        done: done,
    }
}
```



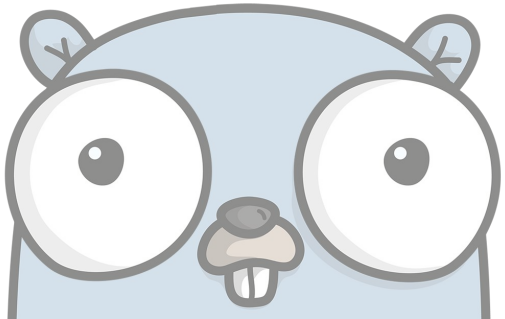
```
// Play is the player game loop
func (p *Player) Play() {
     for {
          select {
          case v := <-p.in:
               // Simulate work
               time.Sleep(1 * time.Second)
               if isBoom(v) {
                    fmt.Printf("Player %d: BOOM\n", p.ID)
               } else {
                    fmt.Printf("Player %d: %d\n", p.ID, v)
               }
               p.out <- v + 1
          case <-p.done:
               fmt.Printf("Player %d: QUIT\n", p.ID)
               return
          }
     }
}
```



```go
// makeChain creates a chain of players, return the 1st player and done channel
// It will also invode the Play method of each player in a gouroutine
func makeChain(n int) (*Player, chan bool) {
	var prev *Player
	var first *Player
	done := make(chan bool)

	// Create chain of players
	for i := 0; i < n; i++ {
		player := NewPlayer(i, done)
		if prev != nil {
			player.in = prev.out
		}

		if first == nil {
			first = player
		} else {
			go player.Play()
		}
		prev = player
	}
	first.in = prev.out
	go first.Play()
	return first, done
}
```



```
func main() {
    first, done := makeChain(3)

    fmt.Println("Play time!")
    first.in <- 1
    time.Sleep(70 * time.Second) // Let them play

    fmt.Println("Stopping Game")
    close(done)
    time.Sleep(200 * time.Millisecond) // for QUIT prints
}
```



Thank You!

miki@353solutions.com