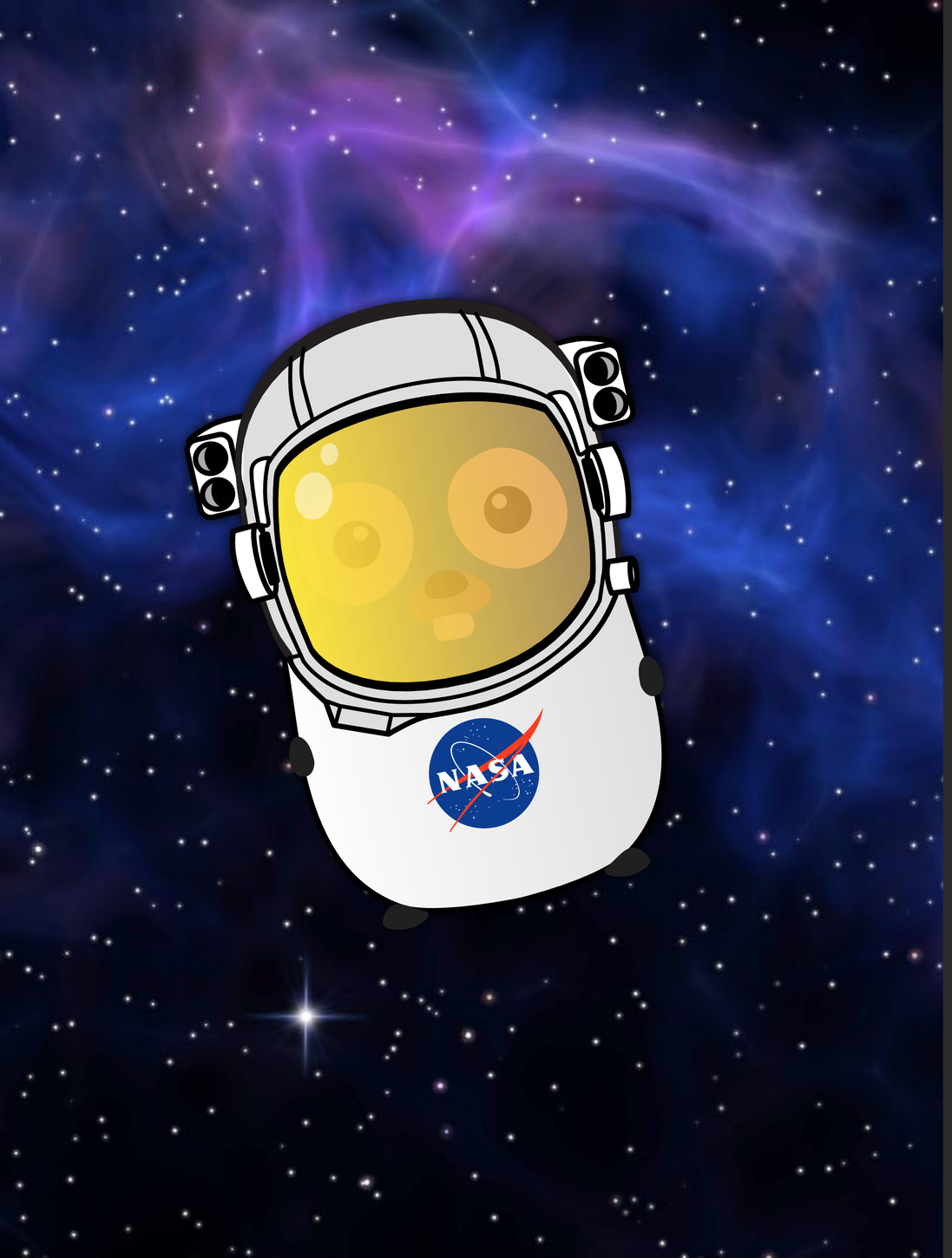


Hes·so VALAIS WALLIS

Michael Clausen

THE GO PROGRAMMING LANGUAGE

hes·so ingenious.

CONTENT

- Why another programming language?
- Basic Syntax
  - Hello World!
  - Values, Variables and Constants
  - For, If/Else, Switch
  - Slices and Maps
  - Ranges
  - Functions
  - Pointers
  - Structs, Methods and Interfaces
  - Struct embedding
  - Visibility
  - Errors and Panic
  - Defer
  - String Functions and String Formatting
  - JSON Encoding and Decoding
  - Number Parsing
- Advanced Topics
  - Embed Directive
  - Variadic Functions
  - Closures
  - Rekursion
  - Strings and Runes
  - Generics
  - Custom Errors
  - Goroutines and Channels
  - Select
  - Timeouts
  - Non-blocking Channel Operations and Closing Channels
  - Range over Channels
  - Time, Timers and Tickers
  - Worker Pools
  - WaitGroups
  - Rate Limiting
  - Atomic Counters
  - Mutexes
  - Sorting and Sorting by Functions
  - Regular Expressions

WHY ANOTHER PROGRAMMING LANGUAGE?

It's a bit **cumbersome**…

It's **nearly impossible**…

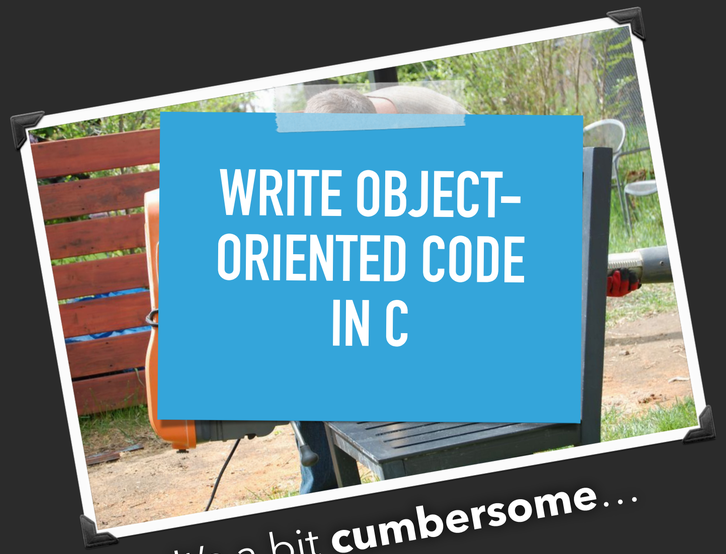
It **is impossible**…

What is the worst than can happen if you use the wrong tool?

It's **completely pointless**…

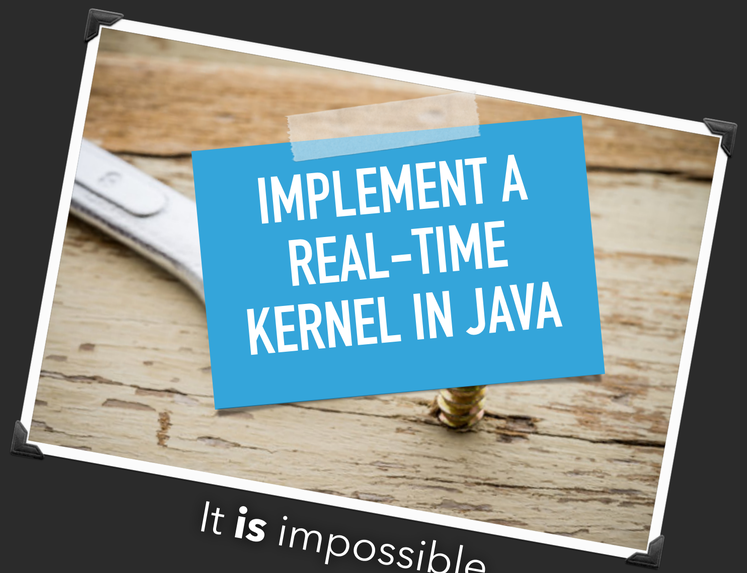
It is just **dangerous**…

WHY ANOTHER PROGRAMMING LANGUAGE? (2)



It's a bit **cumbersome**...

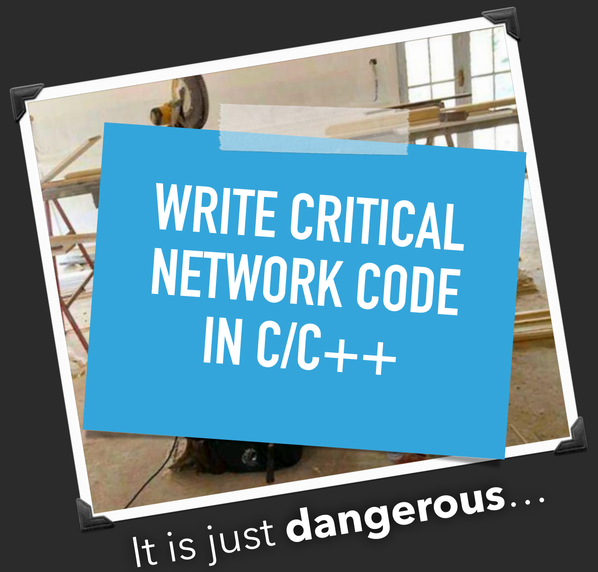
It's **nearly** impossible...



It **is** impossible...

Need examples?

It's **completely pointless**...



It is just **dangerous**...

WHY ANOTHER PROGRAMMING LANGUAGE? (3)

Java/Kotlin

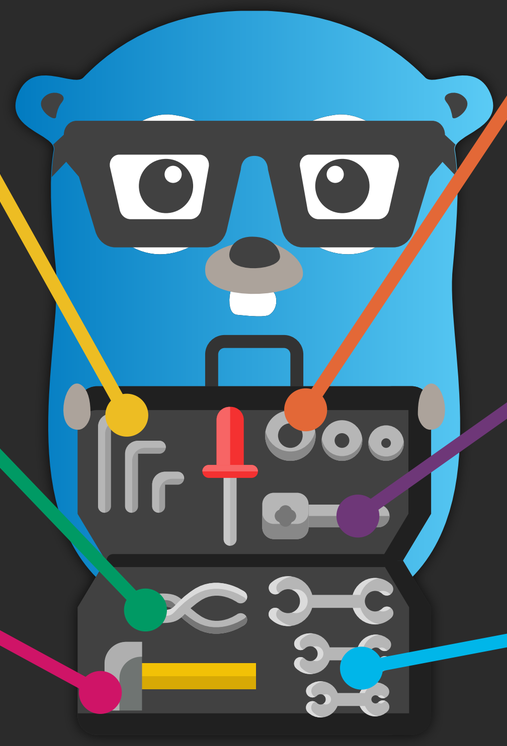
Enterprise Applications & Android Development

Swift

iOS & macOS development

Python

Scripting and Prototyping



Typescript

Web Frontend Development

C/C++

System & Embedded Development

Go

IoT and Micro-Service Development

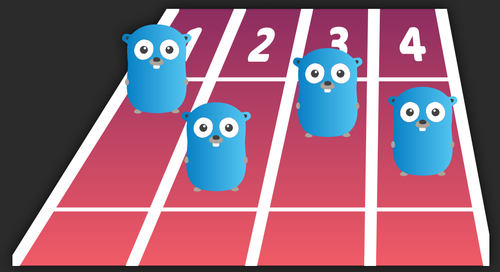
WHY ANOTHER PROGRAMMING LANGUAGE? (4)

Go is **simple**

Go has **good tooling**

Go is **statically typed**

Go is **concurrent**



Go is ***efficient***

Go has a **garbage collector**

Go offers many **libraries**

Go is **Open-source**

Go has build-in **test support**

Hes·so VALAIS WALLIS

BASIC SYNTAX

hes·so ingenious.

HELLO WORLD

hello.go

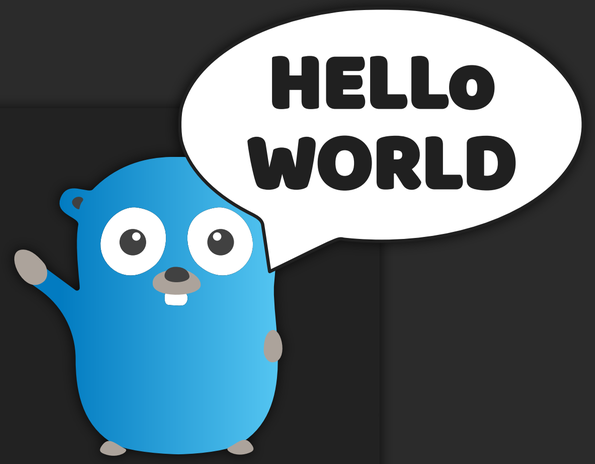
Package `main` contains main function

`import` adds other modules

`main()` is the programs entry point

No return value

```go
package main

import "fmt"

func main() {
    fmt.Println("hello world")
}
```

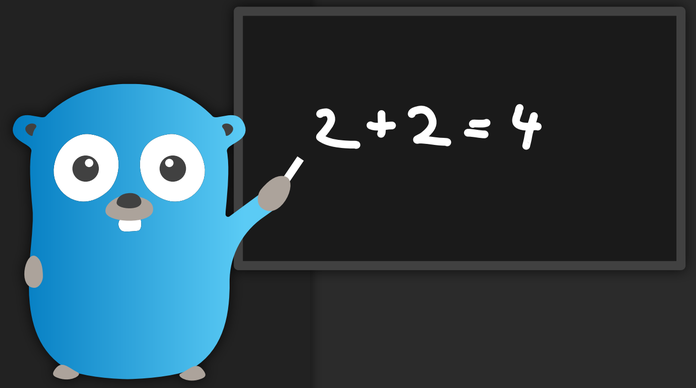



```
> go run hello.go
```

```
> go build hello.go
```

VALUES

```
"go" + "lang"

2+2

7.0/3.0

true && false

true || false

!true
```



VARIABLES

Type is inferred

```
var a = "initial"
```

Type can be explicitly provided

```
var b, c int = 1, 2
```

```
var d = true
```

Variables are zero initialized

```
var e int
```

Shorthand for `var f = “apple“`

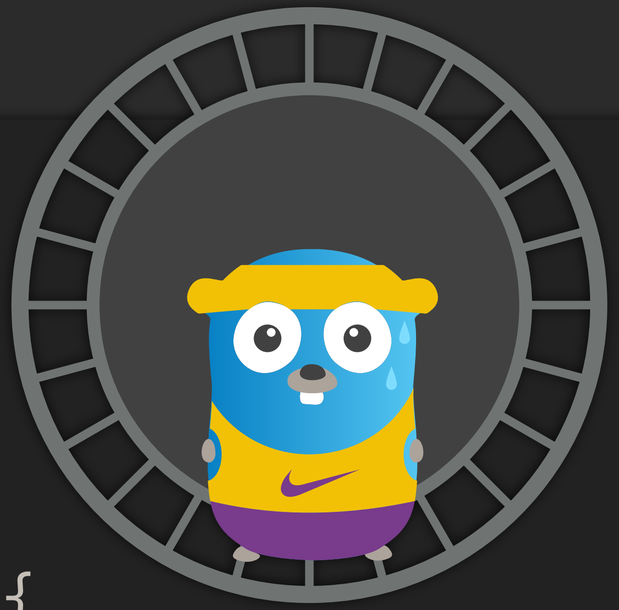
```
f := "apple"
```

CONSTANTS

```go
const s string = "constant"

const (
    c1 = 7
    c2 = 42
)

func main() {
    const n = 500000000

    const d = 3e20 / n

    fmt.Println(int64(d))
}
```

FOR

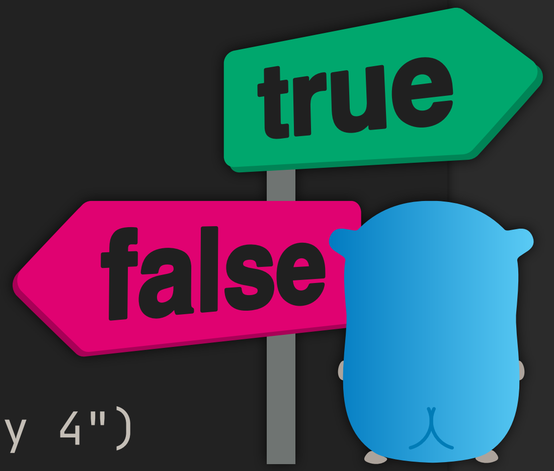
```go
i := 1
for i ≤ 3 {
    fmt.Println(i)
    i = i + 1
}

for j := 0; j < 3; j++ {
    fmt.Println(j)
}

for i := range 3 {
    fmt.Println("range", i)
}
```



```go
for {
    fmt.Println("loop")
    break
}

for n := range 6 {
    if n%2 == 0 {
        continue
    }
    fmt.Println(n)
}
```

IF/ELSE

```go
if 7%2 == 0 {
    fmt.Println("7 is even")
} else {
    fmt.Println("7 is odd")
}

if 8%4 == 0 {
    fmt.Println("8 is divisible by 4")
}

if 8%2 == 0 || 7%2 == 0 {
    fmt.Println("either 8 or 7 are even")
}

if num := 9; num < 0 {
    fmt.Println(num, "is negative")
} else if num < 10 {
    fmt.Println(num, "has 1 digit")
} else {
    fmt.Println(num, "has multiple digits")
}
```



SWITCH

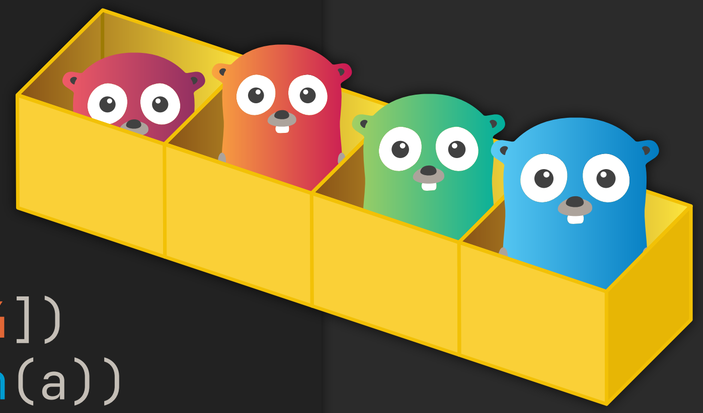
```go
i := 2
switch i {
case 1:
    fmt.Println("one")
case 2:
    fmt.Println("two")
case 3:
    fmt.Println("three")
}
```

```go
switch time.Now().Weekday() {
case time.Saturday, time.Sunday:
    fmt.Println("It's the weekend")
default:
    fmt.Println("It's a weekday")
}
```

```go
whatAmI := func(i interface{}) {
    switch t := i.(type) {
    case bool:
        fmt.Println("I'm a bool")
    case int:
        fmt.Println("I'm an int")
    default:
        fmt.Printf("Don't know type %T\n", t)
    }
}

whatAmI(true)
whatAmI(1)
whatAmI("hey")
```

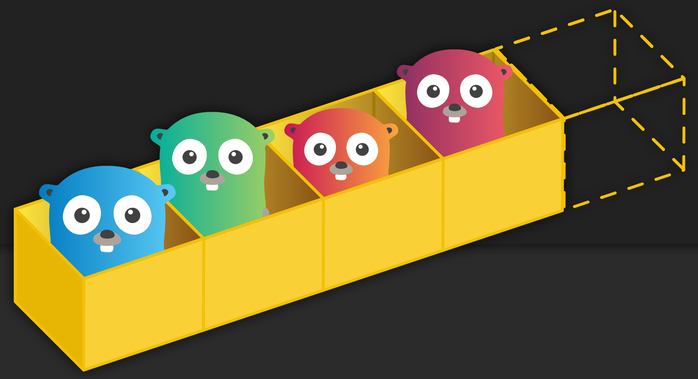
ARRAYS

```go
var a [5]int
fmt.Println("emp:", a)

a[4] = 100
fmt.Println("set:", a)
fmt.Println("get:", a[4])
fmt.Println("len:", len(a))

b := [5]int{1, 2, 3, 4, 5}
fmt.Println("dcl:", b)

var twoD [2][3]int
for i := 0; i < 2; i++ {
    for j := 0; j < 3; j++ {
        twoD[i][j] = i + j
    }
}
fmt.Println("2d: ", twoD)
```



SLICES

```go
var s []string
fmt.Println("uninit:", s, s == nil, len(s) == 0)

s = make([]string, 3)
fmt.Println("em:", s, "le:", len(s), "ca:", cap(s))

s[0] = "a"
s[1] = "b"
s[2] = "c"
fmt.Println("set:", s)
fmt.Println("get:", s[2])
fmt.Println("len:", len(s))

s = append(s, "d")
s = append(s, "e", "f")
fmt.Println("apd:", s)

c := make([]string, len(s))
copy(c, s)
fmt.Println("cpy:", c)

l := s[2:5]
fmt.Println("sl1:", l)
l = s[:5]
fmt.Println("sl2:", l)
```

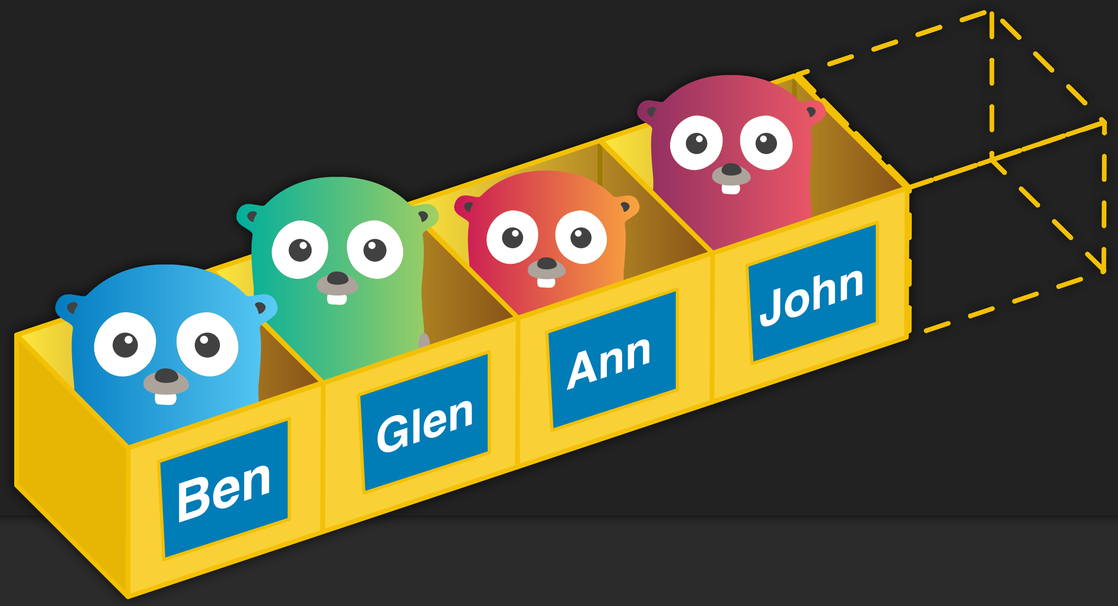


```go
l = s[2:]
fmt.Println("sl3:", l)

t := []string{"g", "h", "i"}
fmt.Println("dcl:", t)

t2 := []string{"g", "h", "i"}
if slices.Equal(t, t2) {
    fmt.Println("t == t2")
}

twoD := make([][]int, 3)
for i := 0; i < 3; i++ {
    innerLen := i + 1
    twoD[i] = make([]int, innerLen)
    for j := 0; j < innerLen; j++ {
        twoD[i][j] = i + j
    }
}
fmt.Println("2d: ", twoD)
```

MAPS

```go
m := make(map[string]int)
m["k1"] = 7
m["k2"] = 13
fmt.Println("map:", m)

v1 := m["k1"]
fmt.Println("v1:", v1)

v3 := m["k3"]
fmt.Println("v3:", v3)
fmt.Println("len:", len(m))

delete(m, "k2")
fmt.Println("map:", m)

clear(m)
fmt.Println("map:", m)

_, ok := m["k2"]
fmt.Println("ok:", ok)
```

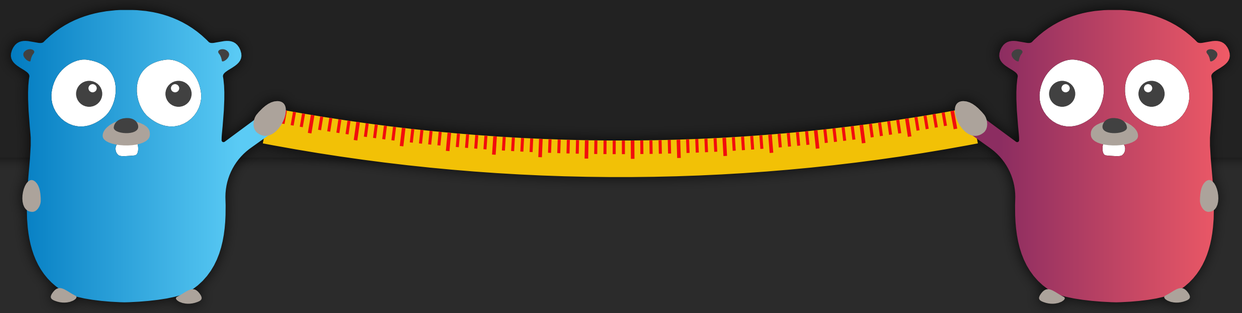
```go
n := map[string]int{"foo": 1, "bar": 2}
fmt.Println("map:", n)

n2 := map[string]int{"foo": 1, "bar": 2}
if maps.Equal(n, n2) {
    fmt.Println("n == n2")
}
```



RANGE

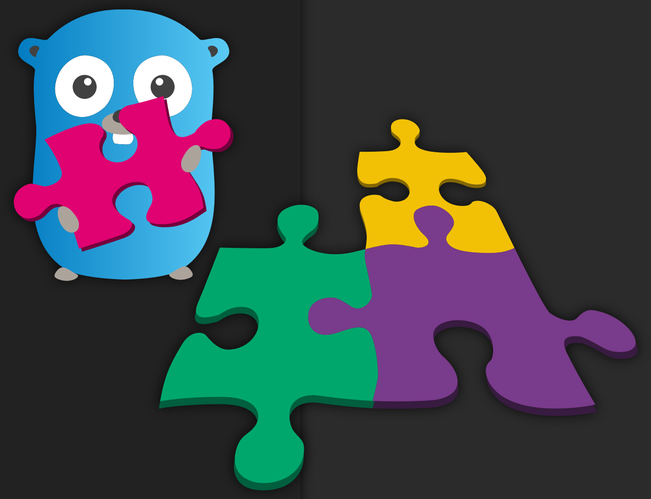
```go
nums := []int{2, 3, 4}
sum := 0
for _, num := range nums {
    sum += num
}
fmt.Println("sum:", sum)

for i, num := range nums {
    if num == 3 {
        fmt.Println("index:", i)
    }
}
```

```go
kvs := map[string]string{"a": "apple", "b": "banana"}
for k, v := range kvs {
    fmt.Printf("%s → %s\n", k, v)
}

for k := range kvs {
    fmt.Println("key:", k)
}

for i, c := range "go" {
    fmt.Println(i, c)
}
```

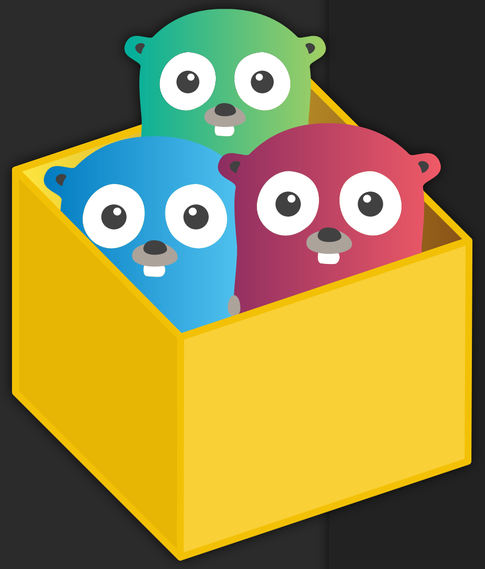


FUNCTIONS

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

func plusPlus(a, b, c int) int {
    return a + b + c
}

func main() {
    res := plus(1, 2)
    fmt.Println("1+2 =", res)

    res = plusPlus(1, 2, 3)
    fmt.Println("1+2+3 =", res)
}

func init() {
   fmt.Println("init() called")
}
```

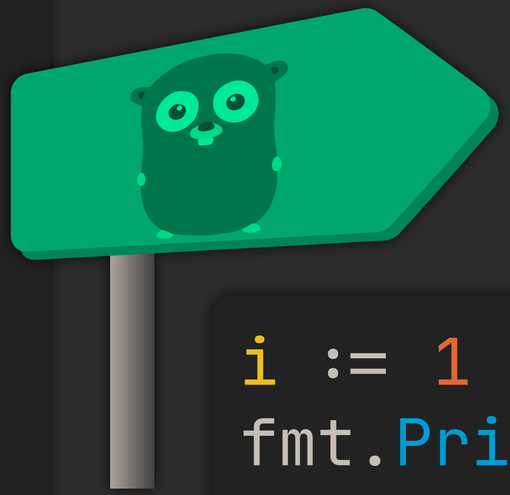


MULTIPLE RETURN VALUES



```go
func vals() (int, int) {
    return 3, 7
}

func main() {
    a, b := vals()
    fmt.Println(a)
    fmt.Println(b)

    _, c := vals()
    fmt.Println(c)
}
```

```go
func zeroval(ival int) {
    ival = 0
}

func zeroptr(iptr *int) {
    *iptr = 0
}
```



```go
i := 1
fmt.Println("initial:", i)

zeroval(i)
fmt.Println("zeroval:", i)

zeroptr(&i)
fmt.Println("zeroptr:", i)

fmt.Println("pointer:", &i)
```

& address of variable

* dereference a pointer

STRUCTS

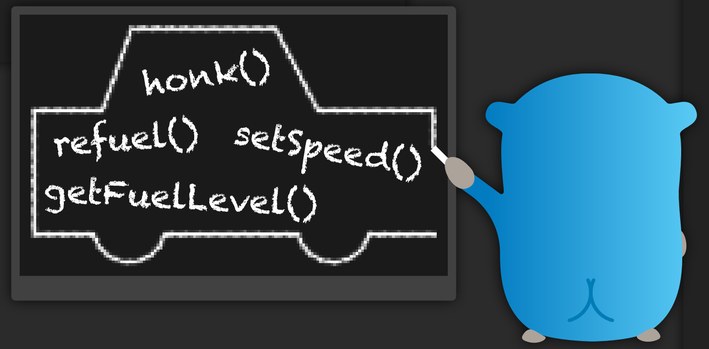
```go
type person struct {
    name string
    age  int
}
```

```go
func newPerson(name string) *person {
    p := person{name: name}
    p.age = 42
    return &p
}
```

```go
fmt.Println(person{"Bob", 20})

fmt.Println(person{name: "Alice", age: 30})

fmt.Println(person{name: "Fred"})

fmt.Println(&person{name: "Ann", age: 40})

fmt.Println(newPerson("Jon"))

s := person{name: "Sean", age: 50}
fmt.Println(s.name)

sp := &s
fmt.Println(sp.age)

sp.age = 51
fmt.Println(sp.age)

dog := struct {
    name   string
    isGood bool
}{
    "Rex",
    true,
}
fmt.Println(dog)
```

METHODS

```go
type rect struct {
    width, height int
}
```



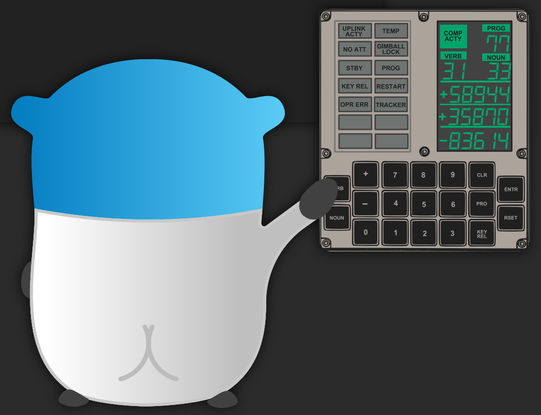

```go
r := rect{width: 10, height: 5}

fmt.Println("area: ", r.area())
fmt.Println("perim:", r.perim())

rp := &r
fmt.Println("area: ", rp.area())
fmt.Println("perim:", rp.perim())
```

```go
func (r *rect) area() int {
    return r.width * r.height
}

func (r rect) perim() int {
    return 2*r.width + 2*r.height
}
```

INTERFACES

```go
type geometry interface {
    area() float64
    perim() float64
}
```

```go
type rect struct {
    width, height float64
}

func (r rect) area() float64 {
    return r.width * r.height
}
func (r rect) perim() float64 {
    return 2*r.width + 2*r.height
}
```

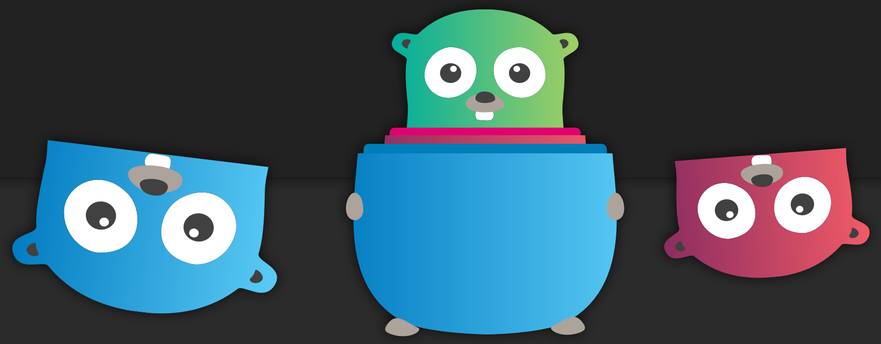


```go
type circle struct {
    radius float64
}

func (c circle) area() float64 {
    return math.Pi * c.radius * c.radius
}
func (c circle) perim() float64 {
    return 2 * math.Pi * c.radius
}
```

```go
func measure(g geometry) {
    fmt.Println(g)
    fmt.Println(g.area())
    fmt.Println(g.perim())
}

func main() {
    r := rect{width: 3, height: 4}
    c := circle{radius: 5}

    measure(r)
    measure(c)
}
```

STRUCT EMBEDDING

```go
type base struct {
    num int
}

func (b base) describe() string {
    return fmt.Sprintf("base with num=%v", b.num)
}

type container struct {
    base
    str string
}
```



```go
co := container{
    base: base{
        num: 1,
    },
    str: "some name",
}

fmt.Printf("co={num: %v, str: %v}\n", co.num, co.str)

fmt.Println("also num:", co.base.num)

fmt.Println("describe:", co.describe())

type describer interface {
    describe() string
}

var d describer = co
fmt.Println("describer:", d.describe())
```

VISIBILITY

Visibility only
at package level

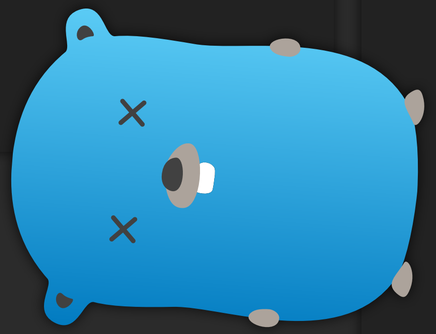
```go
package toto

type Shape interface {
    SetHeight(h int)
    SetWidth(w int)
    CalculateArea() int
}

type rectangle struct {
    height, width int
}

func NewRectangle(height, width int) Shape {
    return &rectangle{height: h, width: w}
}

func (r *rectangle) SetHeight(h int) {
    r.height = h
}

func (r *rectangle) SetWidth(w int) {
    r.width = w
}

func (r *rectangle) CalculateArea() int {
    return r.height * r.width
}
```

exported

not exported

ERRORS

```go
func f(arg int) (int, error) {
    if arg == 42 {
        return -1, errors.New("can't work with 42")
    }

    return arg + 3, nil
}

for _, i := range []int{7, 42} {

    if r, e := f(i); e != nil {
        fmt.Println("f failed:", e)
    } else {
        fmt.Println("f worked:", r)
    }
}
```



```go
var ErrOutOfTea = fmt.Errorf("no more tea available")
var ErrPower = fmt.Errorf("can't boil water")

func makeTea(arg int) error {
    if arg == 2 {
        return ErrOutOfTea
    } else if arg == 4 {

        return fmt.Errorf("making tea: %w", ErrPower)
    }
    return nil
}

for i := range 5 {
    if err := makeTea(i); err != nil {
        if errors.Is(err, ErrOutOfTea) {
            fmt.Println("We should buy new tea!")
        } else if errors.Is(err, ErrPower) {
            fmt.Println("Now it is dark.")
        } else {
            fmt.Printf("unknown error: %s\n", err)
        }
        continue
    }

    fmt.Println("Tea is ready!")
}
```

PANIC

```go
func main() {
    panic("a problem")

    _, err := os.Create("/tmp/file")
    if err ≠ nil {
        panic(err)
    }
}
```

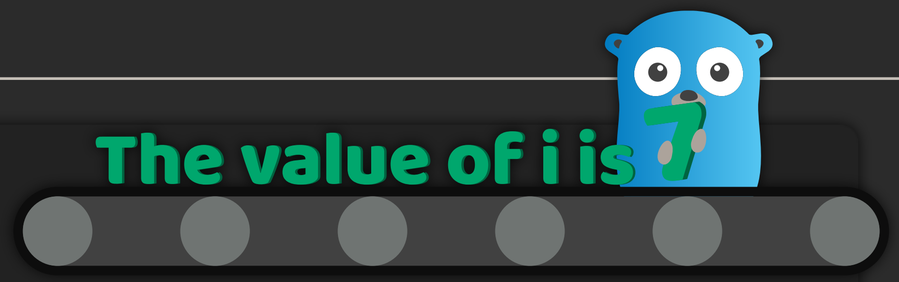
DEFER

```go
func main() {
    f, _ := os.Create("/tmp/defer.txt")
    defer f.Close()

    // Many operations that can cause errors...

    fmt.Fprintln(f, "data")
}
```

STRING FUNCTIONS

```go
var p = fmt.Println

func main() {
    p("Contains:  ", s.Contains("test", "es"))
    p("Count:     ", s.Count("test", "t"))
    p("HasPrefix: ", s.HasPrefix("test", "te"))
    p("HasSuffix: ", s.HasSuffix("test", "st"))
    p("Index:     ", s.Index("test", "e"))
    p("Join:      ", s.Join([]string{"a", "b"}, "-"))
    p("Repeat:    ", s.Repeat("a", 5))
    p("Replace:   ", s.Replace("foo", "o", "0", -1))
    p("Replace:   ", s.Replace("foo", "o", "0", 1))
    p("Split:     ", s.Split("a-b-c-d-e", "-"))
    p("ToLower:   ", s.ToLower("TEST"))
    p("ToUpper:   ", s.ToUpper("test"))
}
```

STRING FORMATTING




```go
type point struct {
    x, y int
}

func main() {
    p := point{1, 2}
    fmt.Printf("struct1: %v\n", p) // struct1: {1 2}
    fmt.Printf("struct2: %+v\n", p) // struct2: {x:1 y:2}
    fmt.Printf("struct3: %#v\n", p) // struct3: main.point{x:1, y:2}

    fmt.Printf("type: %T\n", p) // type: main.point

    fmt.Printf("bool: %t\n", true) // bool: true
    fmt.Printf("int: %d\n", 123) // int: 123
    fmt.Printf("bin: %b\n", 14) // bin: 1110
    fmt.Printf("char: %c\n", 33) // char: !
    fmt.Printf("hex: %x\n", 456) // hex: 1c8
    fmt.Printf("float1: %f\n", 78.9) // float1: 78.900000
    fmt.Printf("float2: %e\n", 123400000.0) // float2: 1.234000e+08
    fmt.Printf("float3: %E\n", 123400000.0) // float3: 1.234000E+08
    fmt.Printf("str1: %s\n", "string") // str1: "string"
    fmt.Printf("str2: %q\n", "string") // str2: "\"string\""
    fmt.Printf("str3: %x\n", "hex this") // str3: 6865782074686973
    fmt.Printf("pointer: %p\n", &p) // pointer: 0xc0000ba000

    fmt.Printf("width1: |%6d|%6d|\n", 12, 345) // width1: |    12|   345|
    fmt.Printf("width2: |%6.2f|%6.2f|\n", 1.2, 3.45) // width2: |  1.20|  3.45|
    fmt.Printf("width3: |%-6.2f|%-6.2f|\n", 1.2, 3.45) // width3: |1.20  |3.45  |
    fmt.Printf("width4: |%6s|%6s|\n", "foo", "b") // width4: |   foo|     b|
    fmt.Printf("width5: |%-6s|%-6s|\n", "foo", "b") // width5: |foo   |b     |

    s := fmt.Sprintf("sprintf: a %s", "string")
    fmt.Println(s) // sprintf: a string

    fmt.Fprintf(os.Stderr, "io: an %s\n", "error") // io: an error
}
```

%v	Prints **value**
%T	Prints **type**
%+v	Prints **value** and **field names**
%#v	Prints **value as go code**
%t	Prints **bool**
%d	Prints **integer**
%c	Prints **char**
%x	Prints as **hex**
%f %e %E	Prints **float**
%s %q	Prints **string**
%p	Prints **pointer address**

JSON ENCODING AND DECODING

```go
type response struct {
    Page   int      `json:"page"`
    Fruits []string `json:"fruits"`
}

func main() {
    intB, _ := json.Marshal(1) // works for bool, int, float and string → 1

    slcD := []string{"apple", "peach", "pear"}
    slcB, _ := json.Marshal(slcD) // → ["apple", "peach", "pear"]

    mapD := map[string]int{"apple": 5, "lettuce": 7}
    mapB, _ := json.Marshal(mapD) // → {"apple": 5, "lettuce": 7}

    res1D := &response{
        Page:   1,
        Fruits: []string{"apple", "peach", "pear"}}
    res1B, _ := json.Marshal(res1D) // → {"page": 1, "fruits": ["apple", "peach", "pear"]}


    byt := []byte(`{"num":6.13,"strs":["a","b"]}`)
    var dat map[string]interface{}
    if err := json.Unmarshal(byt, &dat); err ≠ nil {
        panic(err)
    }
    num := dat["num"].(float64)
    strs := dat["strs"].([]interface{})
    str1 := strs[0].(string)

    str := `{"page": 1, "fruits": ["apple", "peach"]}`
    res := response{}
    json.Unmarshal([]byte(str), &res)
}
```

TIME

```go
p := fmt.Println

now := time.Now()
p(now) // → 2024-09-12 16:44:55.295275 +0200 CEST m=+0.000057584
then := time.Date(
    2009, 11, 17, 20, 34, 58, 651387237, time.UTC)
p(then) // → 2009-11-17 20:34:58.651387237 +0000 UTC
```

```go
diff := now.Sub(then)
p(diff)                // → 129906h9m56.643887763s
p(diff.Hours())        // → 129906.16573441327
p(diff.Minutes())      // → 7.794369944064796e+06
p(diff.Seconds())      // → 4.676621966438776e+08
p(diff.Nanoseconds())  // → 467662196643887763
```

```go
p(then.Year())        // → 2009
p(then.Month())       // → November
p(then.Day())         // → 17
p(then.Hour())        // → 20
p(then.Minute())      // → 34
p(then.Second())      // → 58
p(then.Nanosecond())  // → 651387237
p(then.Location())    // → UTC
p(then.Weekday())     // → Tuesday
```

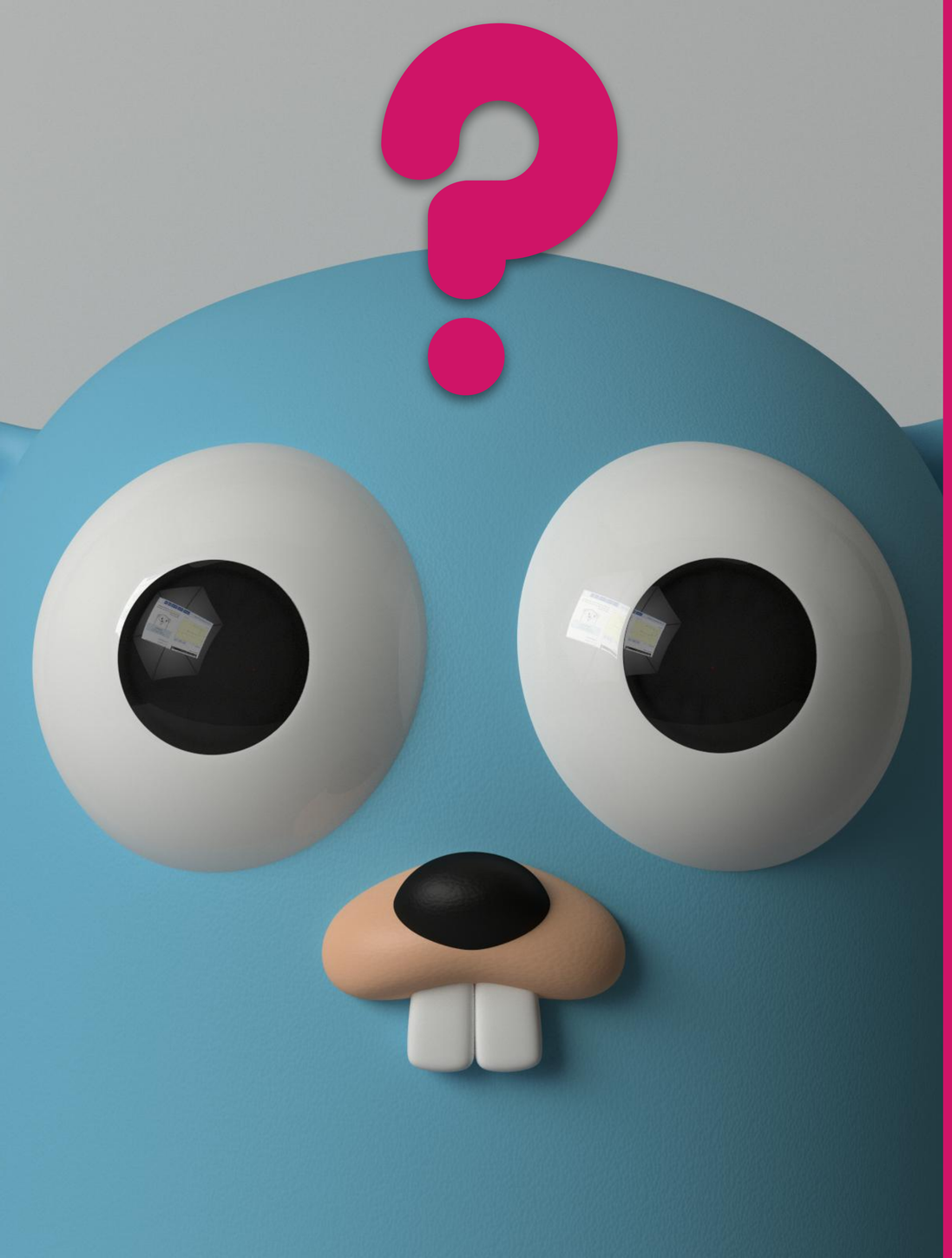
```go
p(then.Add(diff))  // → 2024-09-12 14:44:55.295275 +0000 UTC
p(then.Add(-diff)) // → 1995-01-23 02:25:02.007499474 +0000 UTC
```

```go
p(then.Before(now)) // → true
p(then.After(now))  // → false
p(then.Equal(now))  // → false
```

```go
p(now.Unix())                  // → 1726152295
p(now.UnixMilli())             // → 1726152295295
p(now.UnixNano())              // → 1726152295295275000

p(time.Unix(now.Unix(), 0))    // → 2024-09-12 16:44:55 +0200 CEST
p(time.Unix(0, now.UnixNano())) // → 2024-09-12 16:44:55.295275 +0200 CEST
```

NUMBER PARSING

```go
func main() {
    f, _ := strconv.ParseFloat("1.234", 64)
    fmt.Println(f)

    i, _ := strconv.ParseInt("123", 0, 64)
    fmt.Println(i)

    d, _ := strconv.ParseInt("0x1c8", 0, 64)
    fmt.Println(d)

    u, _ := strconv.ParseUint("789", 0, 64)
    fmt.Println(u)

    k, _ := strconv.Atoi("135")
    fmt.Println(k)

    _, e := strconv.Atoi("wat")
    fmt.Println(e)
}
```



QUESTIONS?

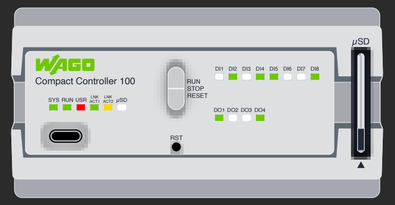
ADVANCED TOPICS

CROSS COMPILATION

GOOS Operating system

GOARCH CPU Architecture

Windows/Intel



Linux/ARM

hello.go

```go
package main

import "fmt"

func main() {
    fmt.Println("hello world")
}
```

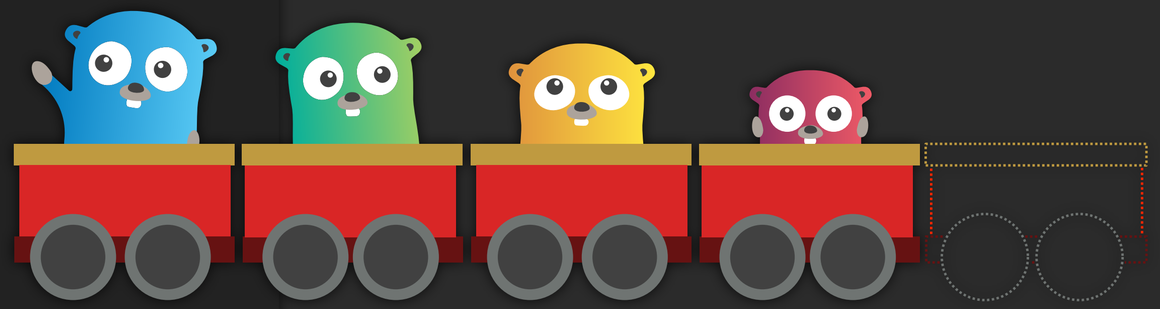
```
# set GOOS linux
# set GOARCH arm
# go build -o hello-linux-arm hello.go
```

EMBED DIRECTIVE

```go
package main

import (
    "embed"
)

//go:embed folder/single_file.txt
var fileString string

//go:embed folder/single_file.txt
var fileByte []byte

//go:embed folder/single_file.txt
//go:embed folder/*.hash
var folder embed.FS

func main() {

    print(fileString)
    print(string(fileByte))

    content1, _ := folder.ReadFile("folder/file1.hash")
    print(string(content1))

    content2, _ := folder.ReadFile("folder/file2.hash")
    print(string(content2))
}
```

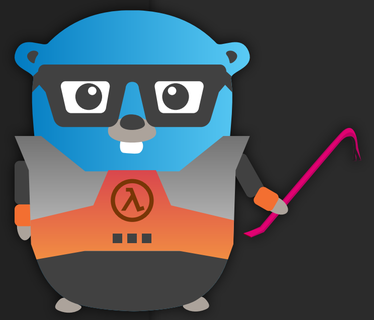
VARIADIC FUNCTIONS

```go
package main

import "fmt"

func sum(nums ...int) {
    fmt.Print(nums, " ")
    total := 0

    for _, num := range nums {
        total += num
    }
    fmt.Println(total)
}

func main() {

    sum(1, 2)
    sum(1, 2, 3)

    nums := []int{1, 2, 3, 4}
    sum(nums...)
}
```



CLOSURES

```go
package main

import "fmt"

func intSeq() func() int {
    i := 0
    return func() int {
        i++
        return i
    }
}

func main() {

    nextInt := intSeq()

    fmt.Println(nextInt())
    fmt.Println(nextInt())
    fmt.Println(nextInt())

    newInts := intSeq()
    fmt.Println(newInts())
}
```

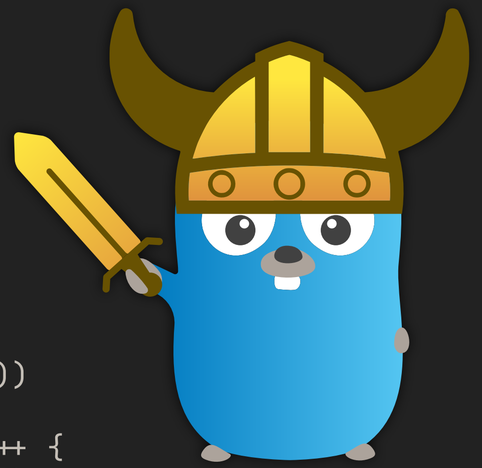


RECURSION

```go
package main

import "fmt"

func fact(n int) int {
    if n == 0 {
        return 1
    }
    return n * fact(n-1)
}

func main() {
    fmt.Println(fact(7))

    var fib func(n int) int

    fib = func(n int) int {
        if n < 2 {
            return n
        }

        return fib(n-1) + fib(n-2)
    }

    fmt.Println(fib(7))
}
```

STRINGS AND RUNES

```go
package main

import (
    "fmt"
    "unicode/utf8"
)

func main() {
    const s = "สวัสดี"

    fmt.Println("Len:", len(s))

    for i := 0; i < len(s); i++ {
        fmt.Printf("%x ", s[i])
    }
    fmt.Println()

    fmt.Println("Rune count:", utf8.RuneCountInString(s))

    for idx, runeValue := range s {
        fmt.Printf("%#U starts at %d\n", runeValue, idx)
    }

    fmt.Println("\nUsing DecodeRuneInString")
    for i, w := 0, 0; i < len(s); i += w {
        runeValue, width := utf8.DecodeRuneInString(s[i:])
        fmt.Printf("%#U starts at %d\n", runeValue, i)
        w = width

        examineRune(runeValue)
    }
}
```



```go
func examineRune(r rune) {
    if r == 't' {
        fmt.Println("found tee")
    } else if r == 'ส' {
        fmt.Println("found so sua")
    }
}
```

GENERICS

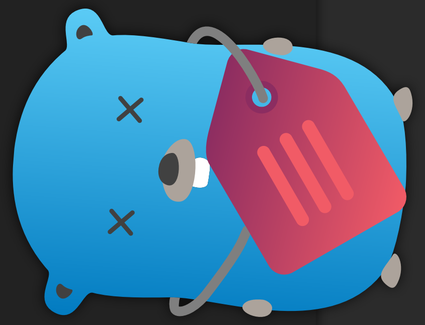
```go
package main

import "fmt"

func MapKeys[K comparable, V any](m map[K]V) []K {
    r := make([]K, 0, len(m))
    for k := range m {
        r = append(r, k)
    }
    return r
}
```

```go
type List[T any] struct {
    head, tail *element[T]
}

type element[T any] struct {
    next *element[T]
    val  T
}

func (lst *List[T]) Push(v T) {
    if lst.tail == nil {
        lst.head = &element[T]{val: v}
        lst.tail = lst.head
    } else {
        lst.tail.next = &element[T]{val: v}
        lst.tail = lst.tail.next
    }
}

func (lst *List[T]) GetAll() []T {
    var elems []T
    for e := lst.head; e != nil; e = e.next {
        elems = append(elems, e.val)
    }
    return elems
}
```

```go
func main() {
    var m = map[int]string{1: "2", 2: "4", 4: "8"}

    fmt.Println("keys:", MapKeys(m))

    _ = MapKeys[int, string](m)

    lst := List[int]{}
    lst.Push(10)
    lst.Push(13)
    lst.Push(23)
    fmt.Println("list:", lst.GetAll())
}
```

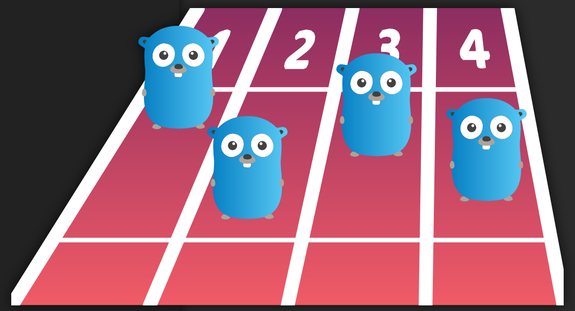
CUSTOM ERRORS

```go
package main

import (
    "errors"
    "fmt"
)

type argError struct {
    arg     int
    message string
}

func (e *argError) Error() string {
    return fmt.Sprintf("%d - %s", e.arg, e.message)
}

func f(arg int) (int, error) {
    if arg == 42 {
        return -1, &argError{arg, "can't work with it"}
    }
    return arg + 3, nil
}

func main() {
    _, err := f(42)
    var ae *argError
    if errors.As(err, &ae) {
        fmt.Println(ae.arg)
        fmt.Println(ae.message)
    } else {
        fmt.Println("err doesn't match argError")
    }
}
```



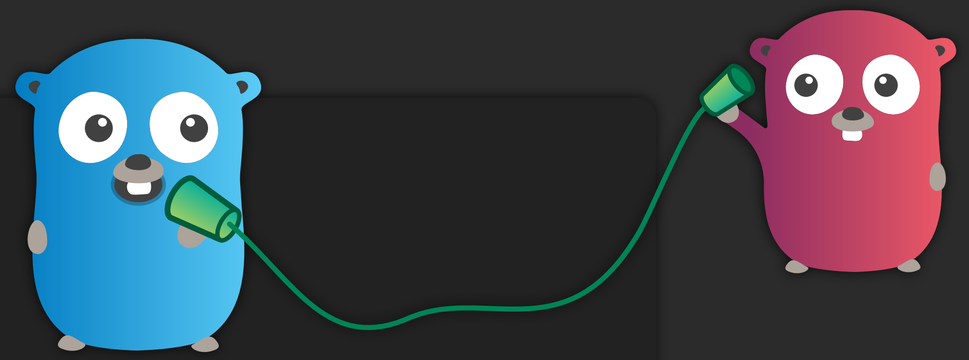
GOROUTINES

```go
package main

import (
    "fmt"
    "time"
)

func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}

func main() {

    f("direct")

    go f("goroutine")

    go func(msg string) {
        fmt.Println(msg)
    }("going")

    time.Sleep(time.Second)
    fmt.Println("done")
}
```



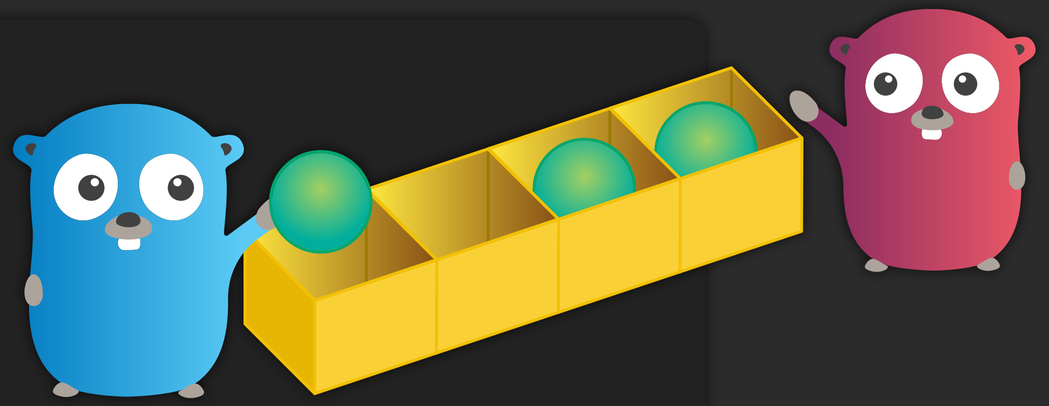
CHANNELS

```go
package main

import "fmt"

func main() {

    messages := make(chan string)

    go func() { messages <- "ping" }()

    msg := <-messages
    fmt.Println(msg)
}
```



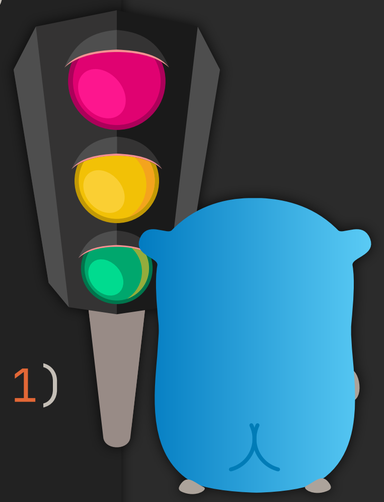
CHANNEL BUFFERING

```go
package main

import "fmt"

func main() {

    messages := make(chan string, 2)

    messages <- "buffered"
    messages <- "channel"

    fmt.Println(<-messages)
    fmt.Println(<-messages)
}
```



CHANNEL SYNCHRONIZATION

```go
package main

import (
    "fmt"
    "time"
)

func worker(done chan bool) {
    fmt.Print("working...")
    time.Sleep(time.Second)
    fmt.Println("done")

    done ← true
}

func main() {
   done := make(chan bool, 1)
    go worker(done)

    ←done
}
```

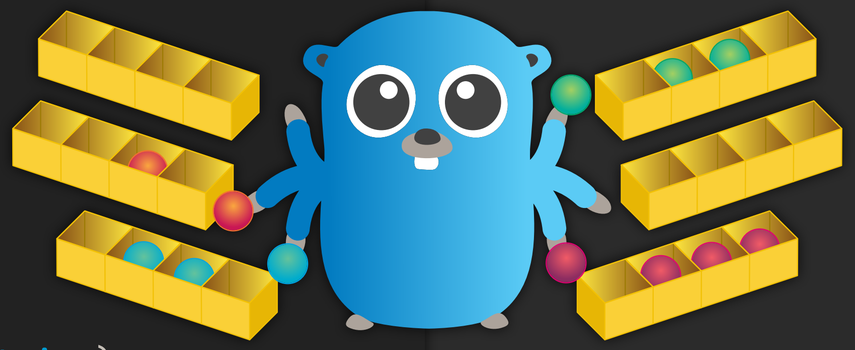


CHANNEL DIRECTIONS

```go
package main

import "fmt"

func ping(pings chan<- string, msg string) {
    pings <- msg
}

func pong(pings <-chan string, pongs chan<- string) {
    msg := <-pings
    pongs <- msg
}

func main() {
    pings := make(chan string, 1)
    pongs := make(chan string, 1)
    ping(pings, "passed message")
    pong(pings, pongs)
    fmt.Println(<-pongs)
}
```

SELECT

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    c1 := make(chan string)
    c2 := make(chan string)

    go func() {
        time.Sleep(1 * time.Second)
        c1 <- "one"
    }()
    go func() {
        time.Sleep(2 * time.Second)
        c2 <- "two"
    }()

    for i := 0; i < 2; i++ {
        select {
        case msg1 := <-c1:
            fmt.Println("received", msg1)
        case msg2 := <-c2:
            fmt.Println("received", msg2)
        }
    }
}
```

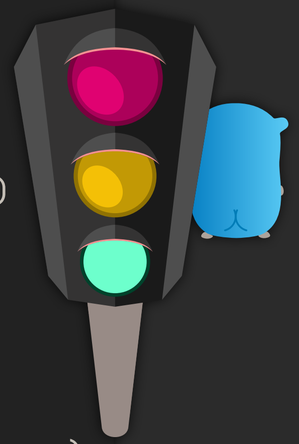


TIMEOUTS

```go
package main

import (
    "fmt"
    "time"
)

func main() {
   c1 := make(chan string, 1)
    go func() {
        time.Sleep(2 * time.Second)
        c1 <- "result 1"
    }()

    select {
    case res := <-c1:
        fmt.Println(res)
    case <-time.After(1 * time.Second):
        fmt.Println("timeout 1")
    }

    c2 := make(chan string, 1)
    go func() {
        time.Sleep(2 * time.Second)
        c2 <- "result 2"
    }()
    select {
    case res := <-c2:
        fmt.Println(res)
    case <-time.After(3 * time.Second):
        fmt.Println("timeout 2")
    }
}
```

NON-BLOCKING CHANNEL OPERATIONS

```go
package main

import "fmt"

func main() {
    messages := make(chan string)
    signals := make(chan bool)

    select {
    case msg := <-messages:
        fmt.Println("received message", msg)
    default:
        fmt.Println("no message received")
    }

    msg := "hi"
    select {
    case messages <- msg:
        fmt.Println("sent message", msg)
    default:
        fmt.Println("no message sent")
    }

    select {
    case msg := <-messages:
        fmt.Println("received message", msg)
    case sig := <-signals:
        fmt.Println("received signal", sig)
    default:
        fmt.Println("no activity")
    }
}
```

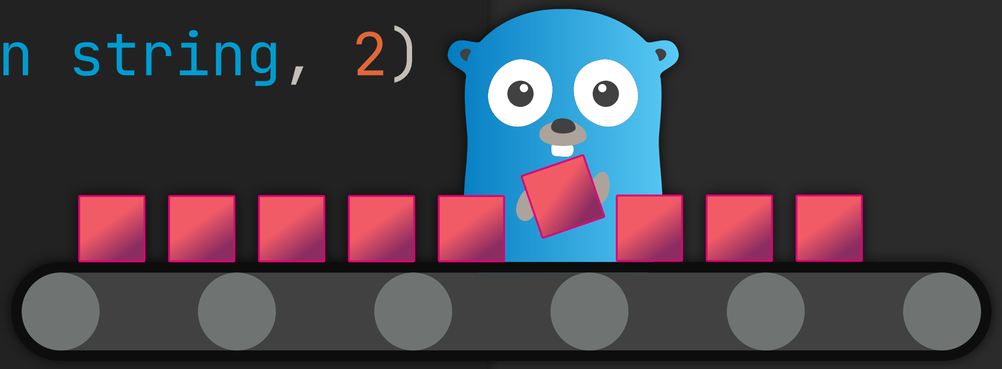


CLOSING CHANNELS

```go
package main

import "fmt"

func main() {
    jobs := make(chan int, 5)
    done := make(chan bool)

    go func() {
        for {
            j, more := <-jobs
            if more {
                fmt.Println("received job", j)
            } else {
                fmt.Println("received all jobs")
                done <- true
                return
            }
        }
    }()

    for j := 1; j <= 3; j++ {
        jobs <- j
        fmt.Println("sent job", j)
    }
    close(jobs)
    fmt.Println("sent all jobs")

    <-done

    _, ok := <-jobs
    fmt.Println("received more jobs:", ok)
}
```

RANGE OVER CHANNELS

```go
package main

import "fmt"

func main() {

    queue := make(chan string, 2)
    queue ← "one"
    queue ← "two"
    close(queue)

    for elem := range queue {
        fmt.Println(elem)
    }
}
```



TIMERS

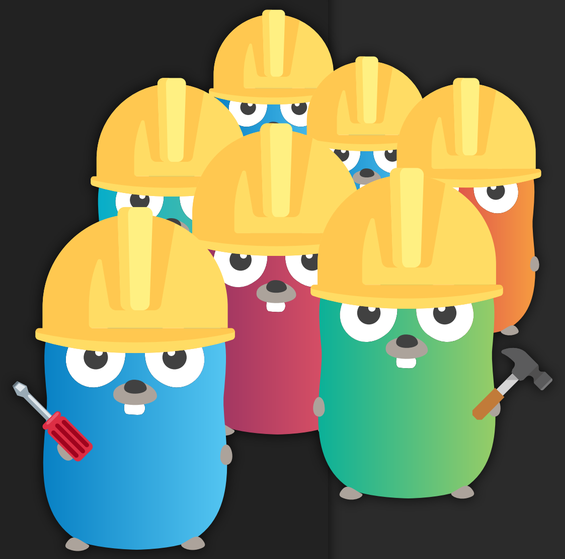
```go
package main

import (
    "fmt"
    "time"
)

func main() {

    timer1 := time.NewTimer(2 * time.Second)

    <-timer1.C
    fmt.Println("Timer 1 fired")

    timer2 := time.NewTimer(time.Second)
    go func() {
        <-timer2.C
        fmt.Println("Timer 2 fired")
    }()
    stop2 := timer2.Stop()
    if stop2 {
        fmt.Println("Timer 2 stopped")
    }

    time.Sleep(2 * time.Second)
}
```

TICKERS

```go
package main

import (
    "fmt"
    "time"
)

func main() {

    ticker := time.NewTicker(500 * time.Millisecond)
    done := make(chan bool)

    go func() {
        for {
            select {
            case <-done:
                return
            case t := <-ticker.C:
                fmt.Println("Tick at", t)
            }
        }
    }()

    time.Sleep(1600 * time.Millisecond)
    ticker.Stop()
    done <- true
    fmt.Println("Ticker stopped")
}
```

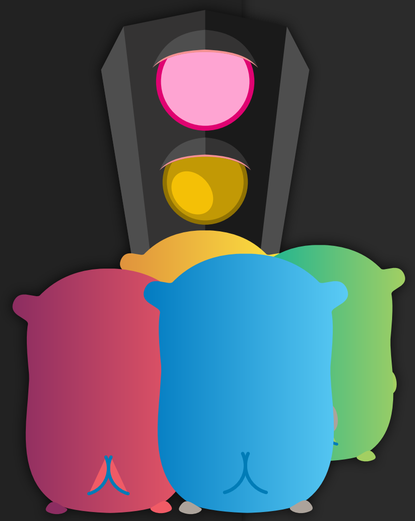
WORKER POOLS

```go
package main

import (
    "fmt"
    "time"
)

func worker(id int, jobs <-chan int, results chan<- int) {
    for j := range jobs {
        fmt.Println("worker", id, "started  job", j)
        time.Sleep(time.Second)
        fmt.Println("worker", id, "finished job", j)
        results <- j * 2
    }
}

func main() {
    const numJobs = 5
    jobs := make(chan int, numJobs)
    results := make(chan int, numJobs)

    for w := 1; w <= 3; w++ {
        go worker(w, jobs, results)
    }

    for j := 1; j <= numJobs; j++ {
        jobs <- j
    }
    close(jobs)

    for a := 1; a <= numJobs; a++ {
        <-results
    }
}
```



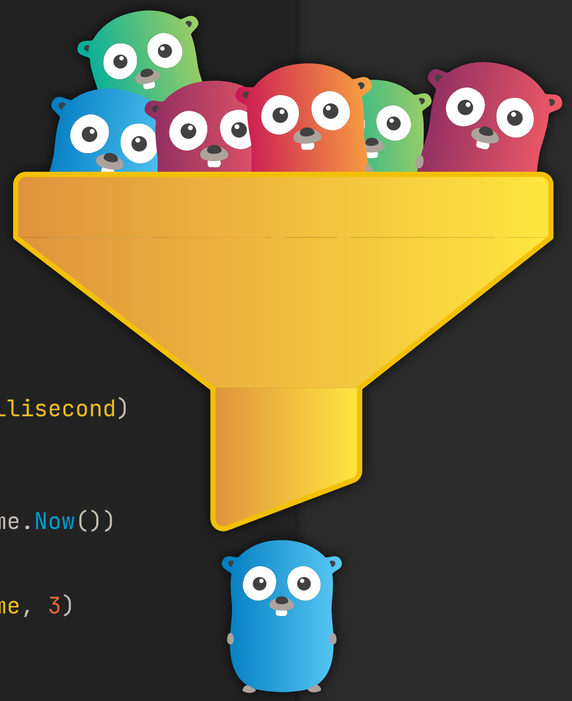
WAITGROUPS

```go
package main

import (
    "fmt"
    "sync"
    "time"
)

func worker(id int) {
    fmt.Printf("Worker %d starting\n", id)

    time.Sleep(time.Second)
    fmt.Printf("Worker %d done\n", id)
}

func main() {

    var wg sync.WaitGroup

    for i := 1; i ≤ 5; i++ {
        wg.Add(1)

        go func() {
            defer wg.Done()
            worker(i)
        }()
    }

    wg.Wait()

}
```



RATE LIMITING

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    requests := make(chan int, 5)
    for i := 1; i ≤ 5; i++ {
        requests ← i
    }
    close(requests)

    limiter := time.Tick(200 * time.Millisecond)

    for req := range requests {
        ←limiter
        fmt.Println("request", req, time.Now())
    }

    burstyLimiter := make(chan time.Time, 3)

    for i := 0; i < 3; i++ {
        burstyLimiter ← time.Now()
    }

    go func() {
        for t := range time.Tick(200 * time.Millisecond) {
            burstyLimiter ← t
        }
    }()

    burstyRequests := make(chan int, 5)
    for i := 1; i ≤ 5; i++ {
        burstyRequests ← i
    }
    close(burstyRequests)
    for req := range burstyRequests {
        ←burstyLimiter
        fmt.Println("request", req, time.Now())
    }
}
```



ATOMIC COUNTERS

```go
package main

import (
    "fmt"
    "sync"
    "sync/atomic"
)

func main() {
   var ops atomic.Uint64

    var wg sync.WaitGroup

    for i := 0; i < 50; i++ {
        wg.Add(1)

        go func() {
            for c := 0; c < 1000; c++ {

                ops.Add(1)
            }

            wg.Done()
        }()
    }

    wg.Wait()

    fmt.Println("ops:", ops.Load())
}
```

MUTEXES

```go
package main

import (
    "fmt"
    "sync"
)

type Container struct {
    mu       sync.Mutex
    counters map[string]int
}

func (c *Container) inc(name string) {
    c.mu.Lock()
    defer c.mu.Unlock()
    c.counters[name]++
}
```

```go
func main() {
    c := Container{
        counters: map[string]int{"a": 0, "b": 0},
    }

    var wg sync.WaitGroup

    doIncrement := func(name string, n int) {
        for i := 0; i < n; i++ {
            c.inc(name)
        }
        wg.Done()
    }

    wg.Add(3)
    go doIncrement("a", 10000)
    go doIncrement("a", 10000)
    go doIncrement("b", 10000)

    wg.Wait()
    fmt.Println(c.counters)
}
```

SORTING

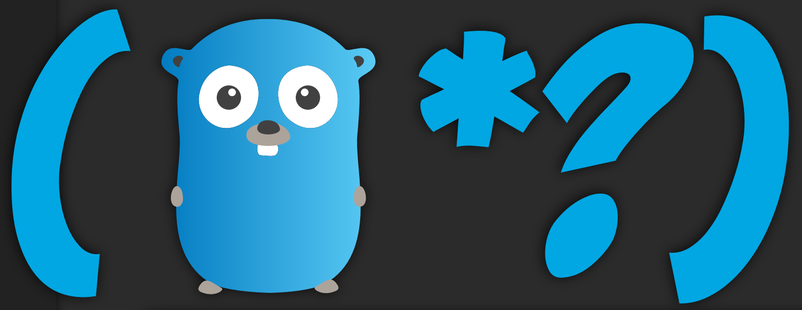
```go
package main

import (
    "fmt"
    "slices"
)

func main() {
   strs := []string{"c", "a", "b"}
    slices.Sort(strs)
    fmt.Println("Strings:", strs)

    ints := []int{7, 2, 4}
    slices.Sort(ints)
    fmt.Println("Ints:   ", ints)

    s := slices.IsSorted(ints)
    fmt.Println("Sorted: ", s)
}
```

SORTING BY FUNCTIONS

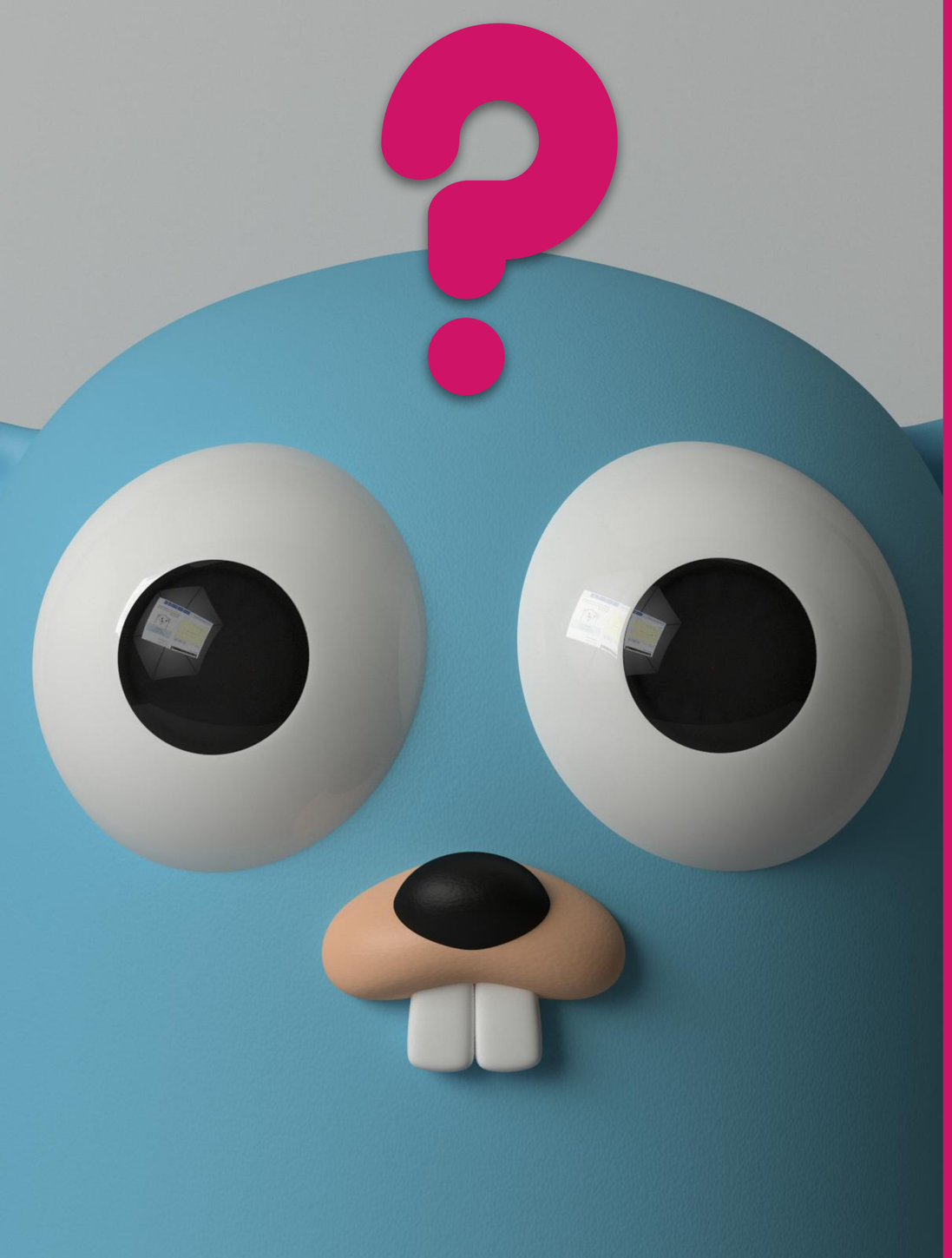
```go
package main

import (
    "cmp"
    "fmt"
    "slices"
)

func main() {
    fruits := []string{"peach", "banana", "kiwi"}

    lenCmp := func(a, b string) int {
        return cmp.Compare(len(a), len(b))
    }

    slices.SortFunc(fruits, lenCmp)
    fmt.Println(fruits)

    type Person struct {
        name string
        age  int
    }

    people := []Person{
        Person{name: "Jax", age: 37},
        Person{name: "TJ", age: 25},
        Person{name: "Alex", age: 72},
    }

    slices.SortFunc(people,
        func(a, b Person) int {
            return cmp.Compare(a.age, b.age)
        })
    fmt.Println(people)
}
```

REGULAR EXPRESSIONS

```go
package main

import (
    "bytes"
    "fmt"
    "regexp"
)
func main() {
   match, _ := regexp.MatchString("p([a-z]+)ch", "peach")
    fmt.Println(match)

    r, _ := regexp.Compile("p([a-z]+)ch")

    fmt.Println(r.MatchString("peach"))

    fmt.Println(r.FindString("peach punch"))

    fmt.Println("idx:", r.FindStringIndex("peach punch"))

    fmt.Println(r.FindStringSubmatch("peach punch"))
```




```go
    fmt.Println(r.FindStringSubmatchIndex("peach punch"))

    fmt.Println(r.FindAllString("peach punch pinch", -1))

    fmt.Println("all:", r.FindAllStringSubmatchIndex(
        "peach punch pinch", -1))

    fmt.Println(r.FindAllString("peach punch pinch", 2))

    fmt.Println(r.Match([]byte("peach")))

    r = regexp.MustCompile("p([a-z]+)ch")
    fmt.Println("regexp:", r)

    fmt.Println(r.ReplaceAllString("a peach", "<fruit>"))

    in := []byte("a peach")
    out := r.ReplaceAllFunc(in, bytes.ToUpper)
    fmt.Println(string(out))
}
```



QUESTIONS?