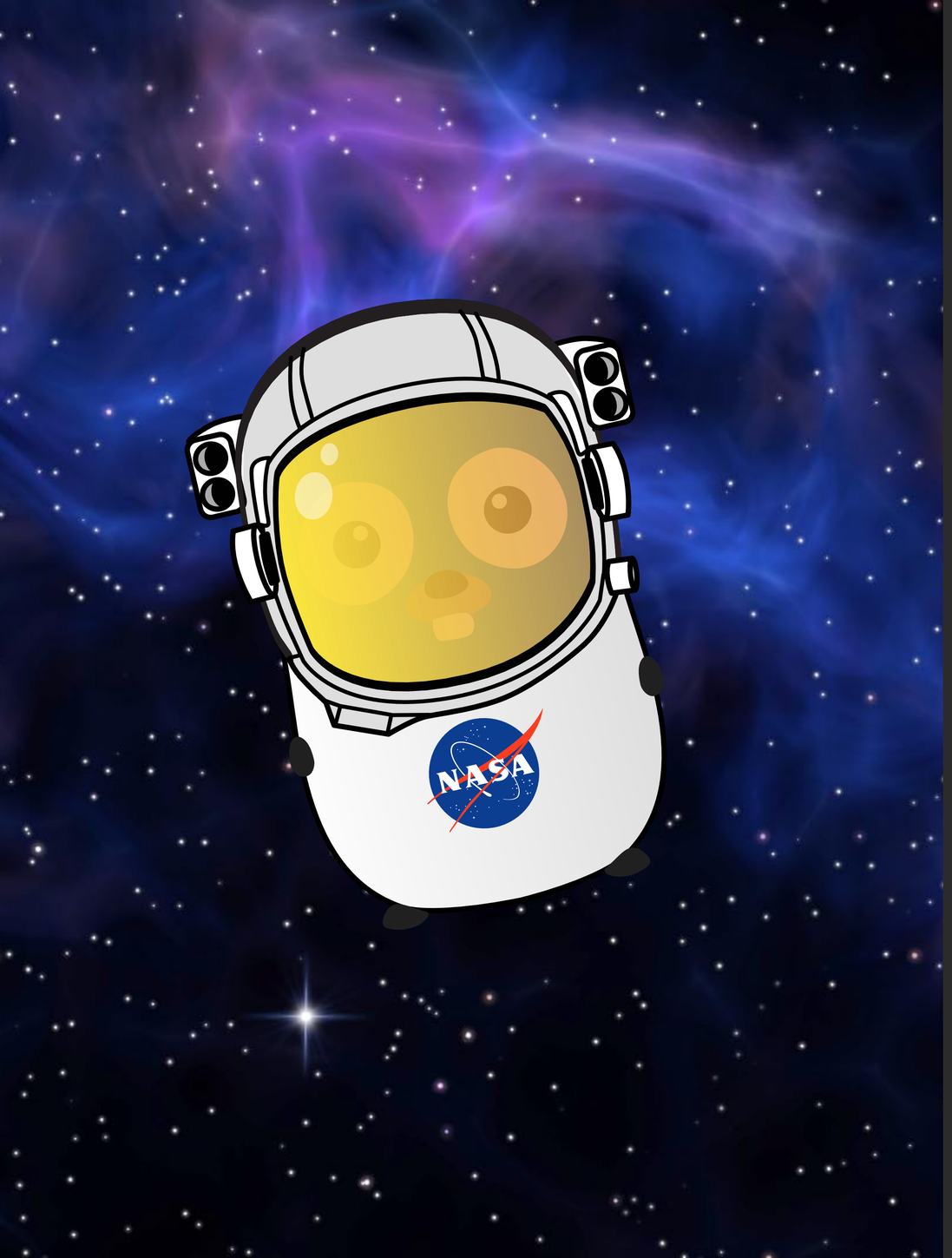


Hes·so VALAIS WALLIS

: Σ π ≈ &

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

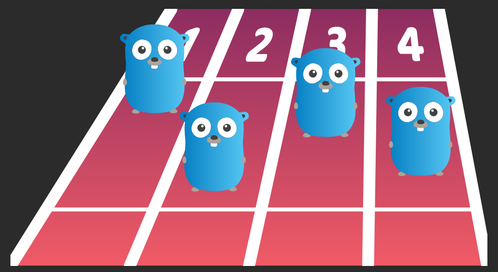
WHY ANOTHER PROGRAMMING LANGUAGE?

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  **pen-source**

Go has build-in **test support** 

Hes·so VALAIS WALLIS

BASIC SYNTAX

HELLO WORLD

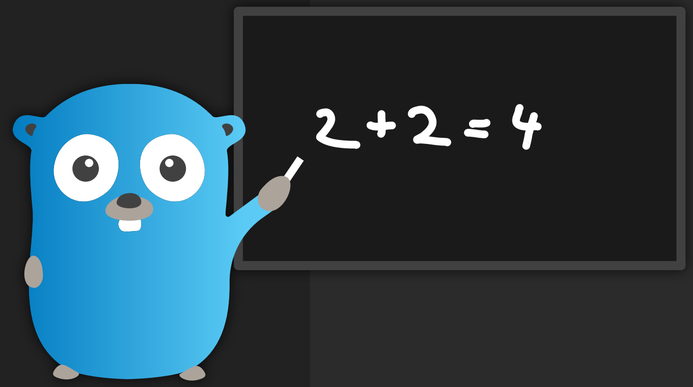
hello.go

```go
package main

import "fmt"

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


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

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

```
"go" + „lang"

2+2

7.0/3.0

true && false

true || false

!true
```



VARIABLES

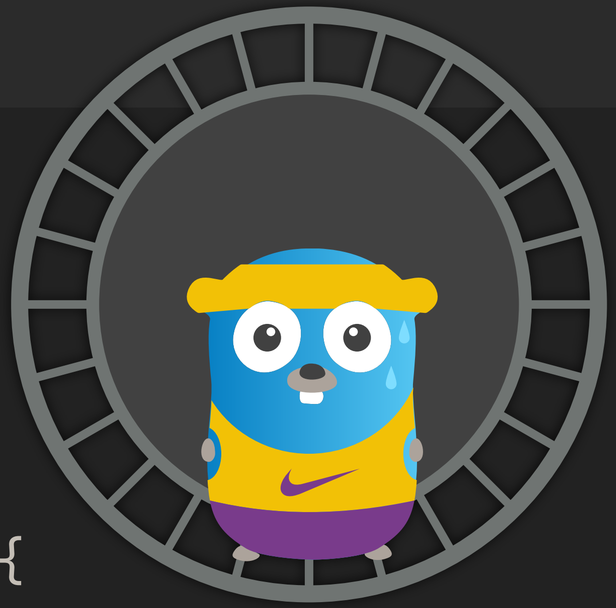
```
var a = „initial"

var b, c int = 1, 2

var d = true

var e int

f := "apple"
```

CONSTANTS

```
const s string = "constant"

const (
    c1 = 7
    c2 = 42
)

func main() {
    const n = 500000000

    const d = 3e20 / n

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

FOR

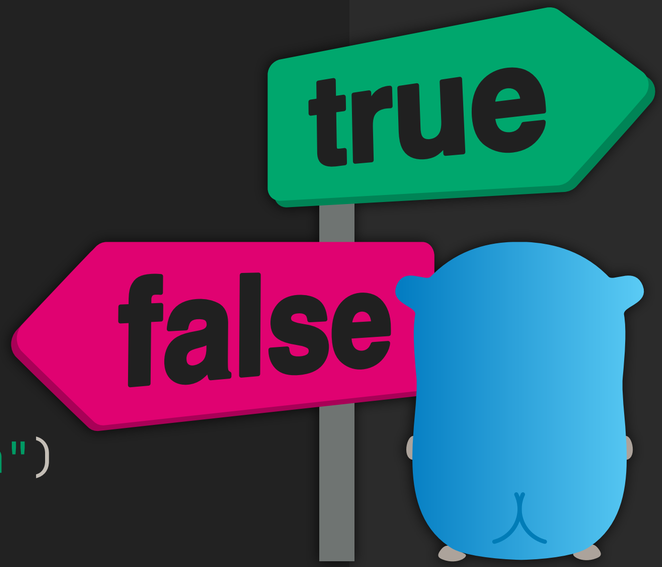
```go
i := 1
for i <= 3 {
    fmt.Println(i)
    i = i + 1
}

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

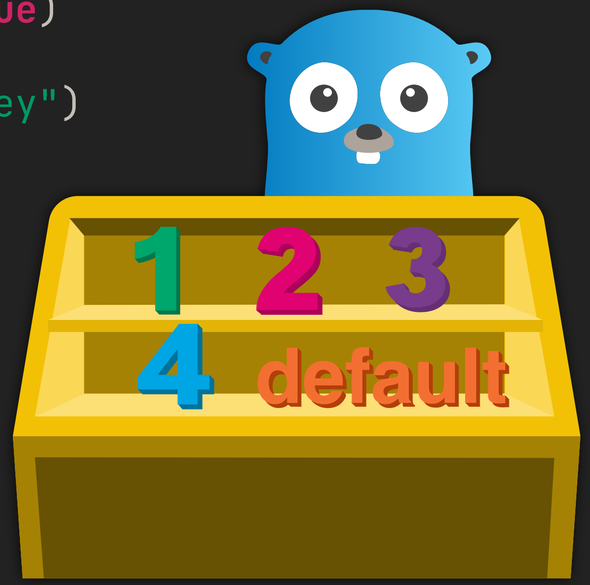
```go
if 7%2 == 0 {
    fmt.Println("7 is even")
} else {
    fmt.Println("7 is odd")
}

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

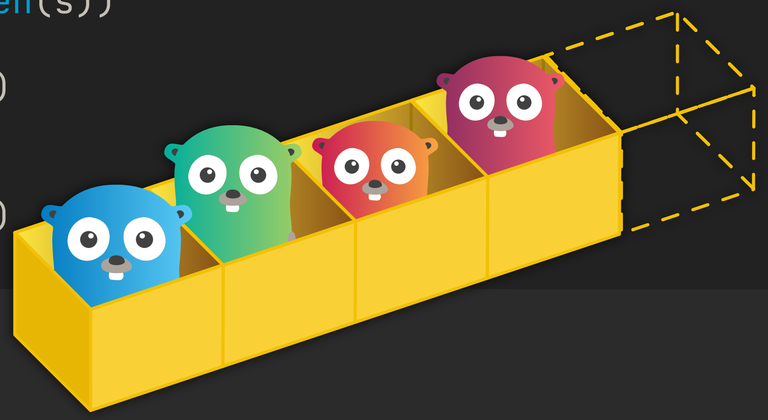
```go
i := 2
switch i {
case 1:
    fmt.Println("one")
case 2:
    fmt.Println("two")
case 3:
    fmt.Println("three")
}

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

t := time.Now()
switch {
case t.Hour() < 12:
    fmt.Println("It's before noon")
default:
    fmt.Println("It's after noon")
}
```

```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")
```



SLICES

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

s = make([]string, 3)
fmt.Println("emp:", s, "len:", len(s), "cap:",
  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)
```

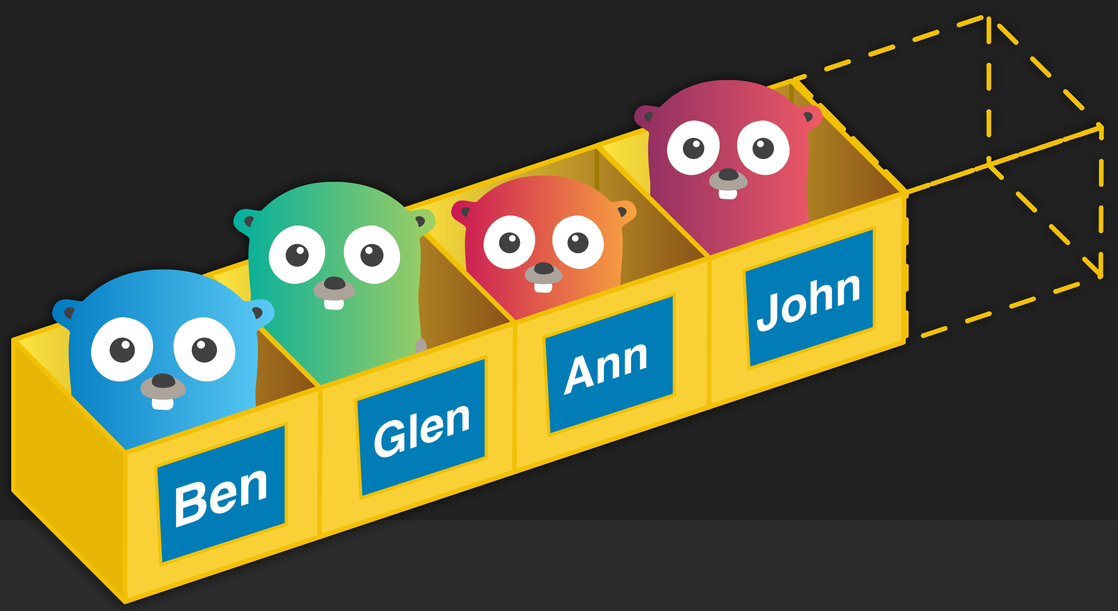


```
l = s[:5]
fmt.Println("sl2:", l)

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)
```

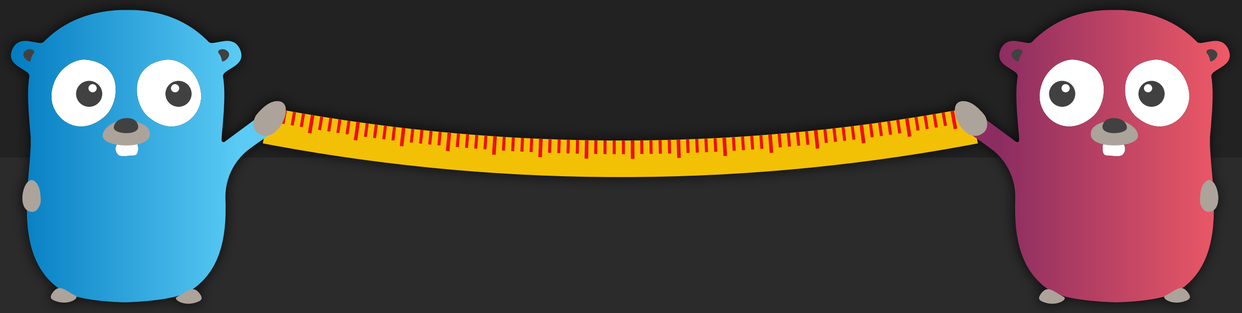
```go
_, prs := m["k2"]
fmt.Println("prs:", prs)

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

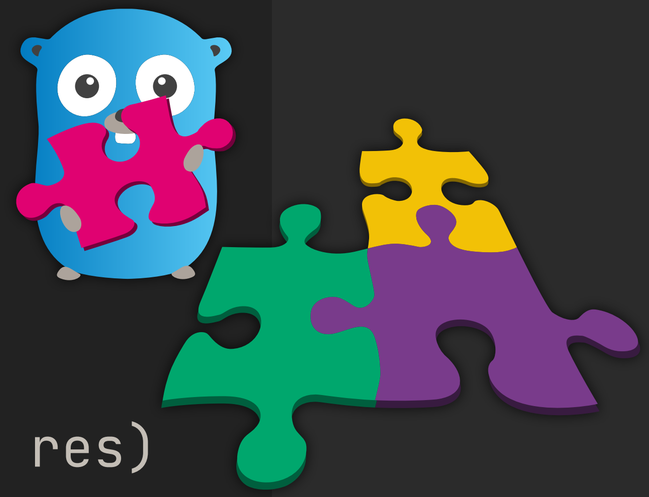
```go
nums := []int{2, 3, 4}
sum := 0
for _, num := range nums {
    sum += num
}
fmt.Println("sum:", sum)

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)
}
```

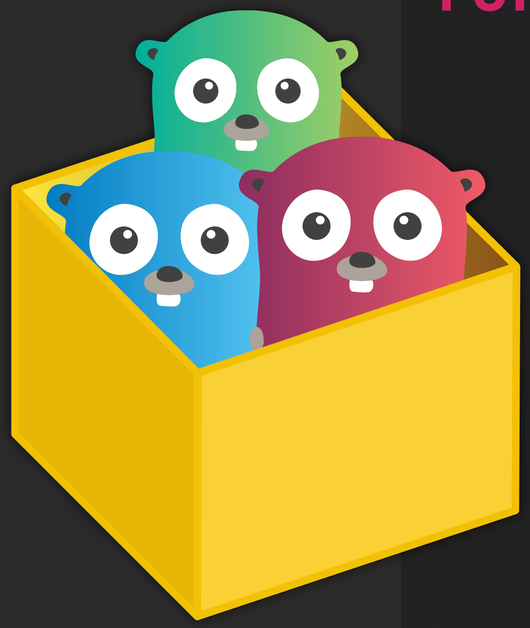


FUNCTIONS

```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)
}
```



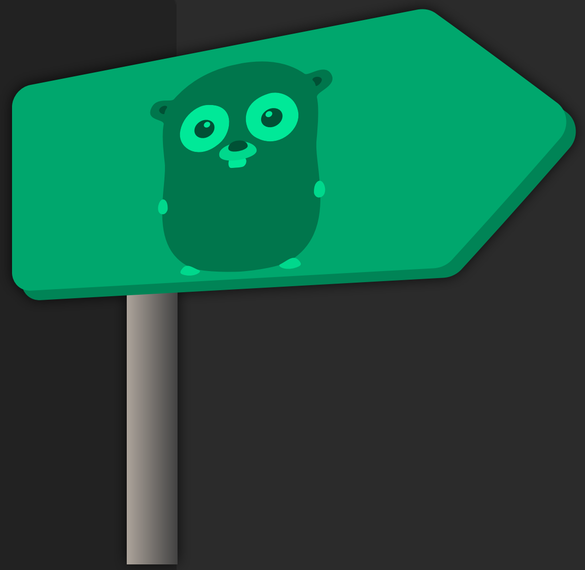
MULTIPLE RETURN VALUES



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

func main() {

    a, b := vals()
    fmt.Println(a)
    fmt.Println(b)

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

POINTERS

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

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

func main() {
    i := 1
    fmt.Println("initial:", i)

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

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

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



STRUCTS

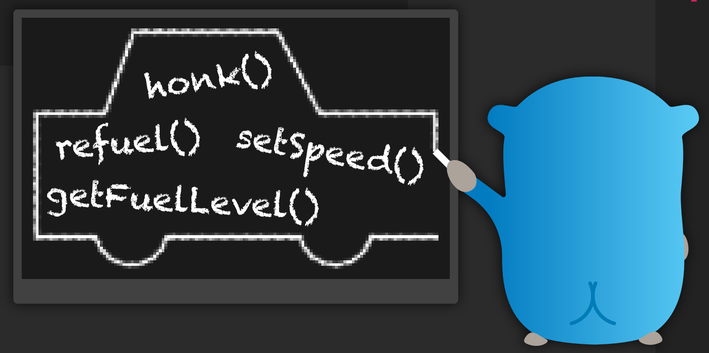
```go
type person struct {
    name string
    age  int
}
```

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

```go
func main() {
    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

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

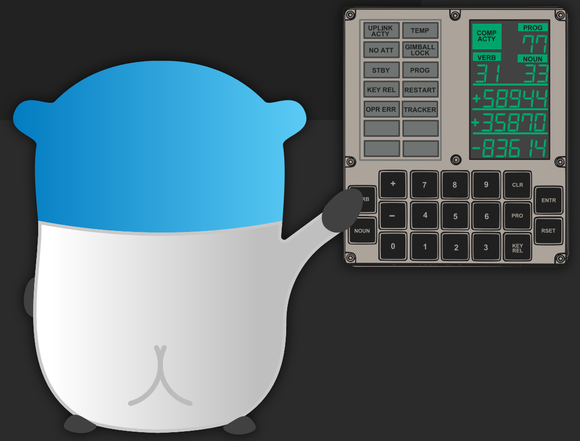



```
func (r *rect) area() int {
    return r.width * r.height
}

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

```
func main() {
    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())
}
```

INTERFACES

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



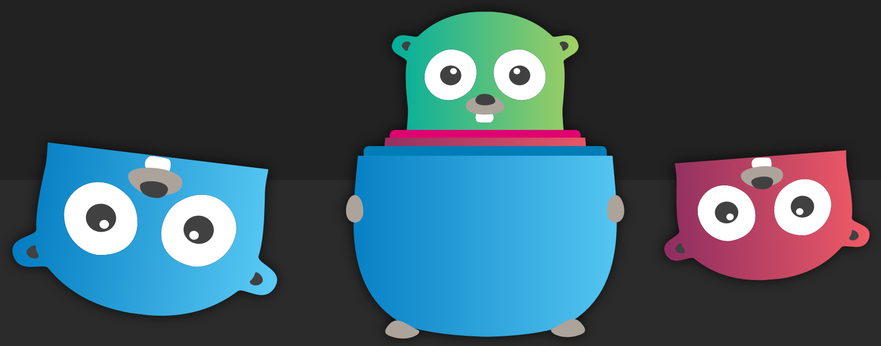

```
type rect struct {
    width, height float64
}

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

```
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
}
```

```
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
}
```



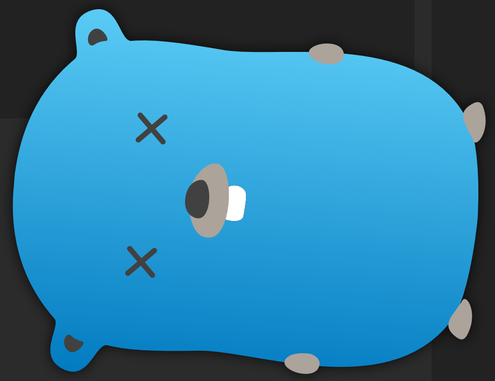
```go
func main() {
    co := container{
        base: base{
            num: 1,
        },
        str: "some name",
    }

    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

```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
}
```

ERRORS

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

    return arg + 3, nil
}

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
}
```



```go
func main() {
    for _, i := range []int{7, 42} {

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

    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

```
func main() {

    panic("a problem")

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

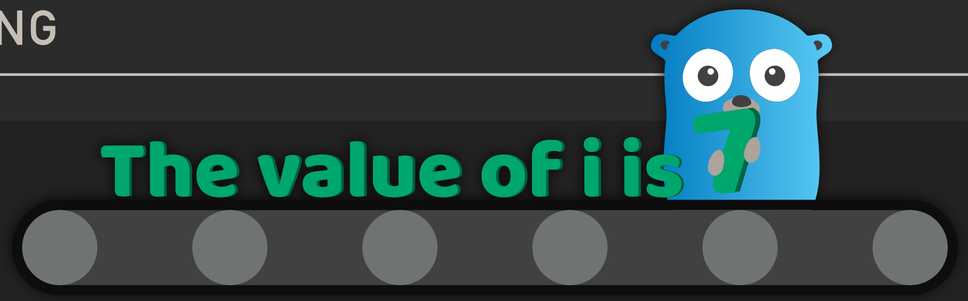
DEFER

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

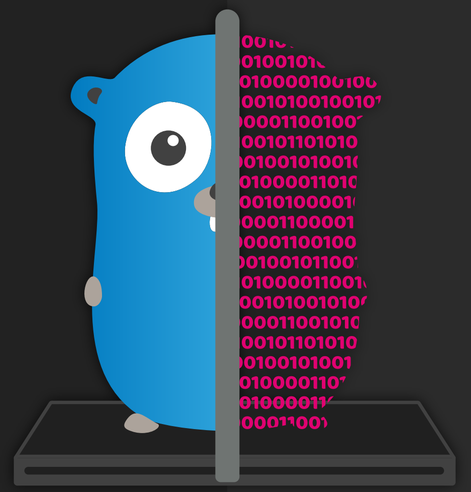
`%f` `%e` `%E` Prints **float**

`%s` `%q` Prints **string**

`%p` Prints **pointer address**

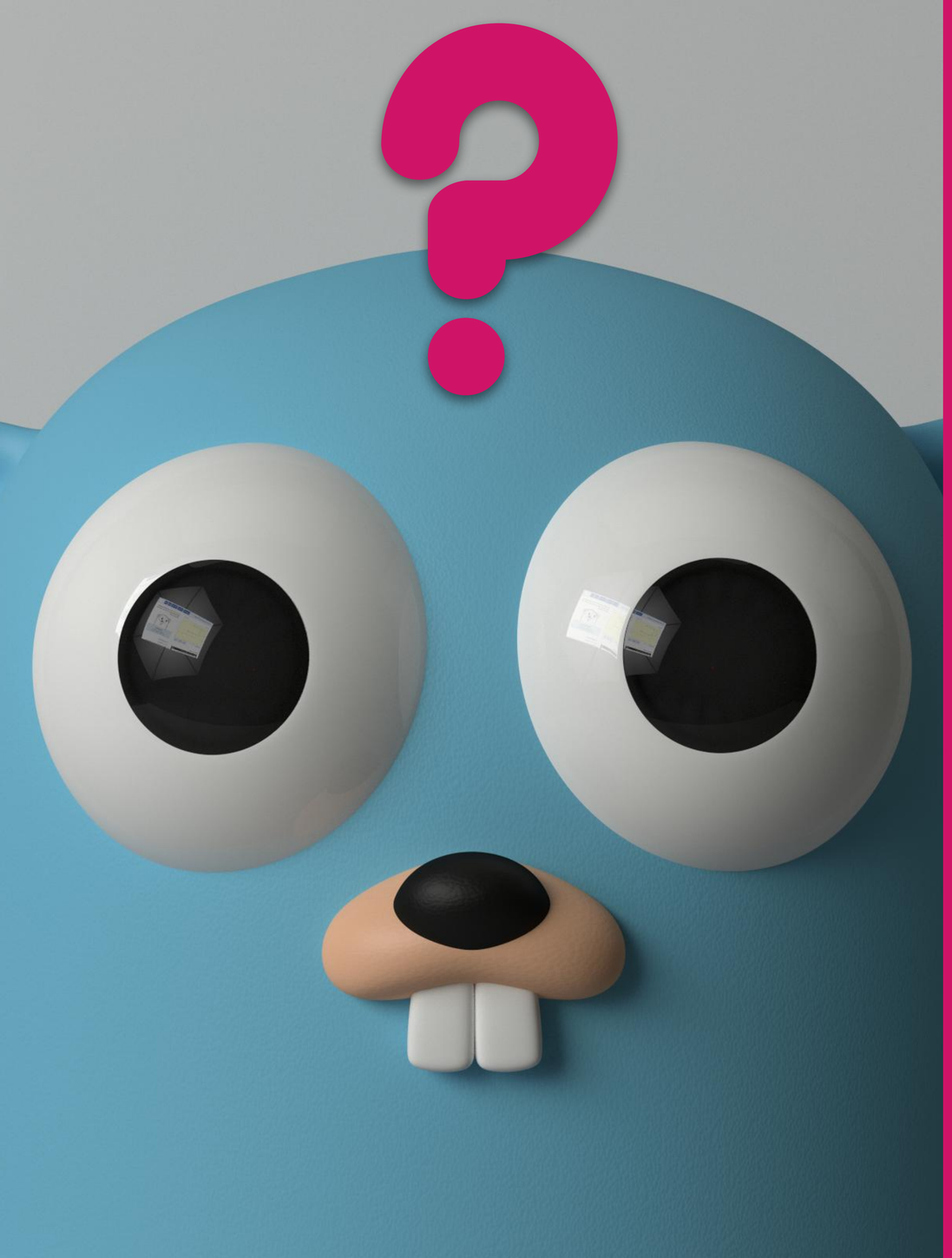
JSON ENCODING AND DECODING

```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)
}
```



NUMBER PARSING

```
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?

Hes·so VALAIS WALLIS

: Σ π ≈ &

ADVANCED TOPICS