Le langage GO

Marwan HALLAL, RICM5

Le projet

- Crée par Google (2007)
- Ken Thompson, Robert Pike
- Motivation = avancés techno. + pas de nouveaux langages
- open-source en 2009

Ce qu’on a fait avec

- Docker
- Vitess (MySQL clustering)
- Google download server
- InfluxDB

Un langage système..

	Go	vs.	C
Compilé, typage statique	✓		✓
Garbage collection	✓		✗
Support Unicode	✓		✗
Interfaces	✓		✗
Méthodes	✓		✗
Concurrence ***by design***	✓		✗ (pthreads)

Une librairie standard exhaustive..

- Réseaux
- Crypto
- Unicode
- Drivers SQL
- Introspection (*reflect*)

Syntaxe de base

- Syntaxe à la C..
- Pas de ; (ajoutées en run-time)
- Variables:
  - var x int
  - var x = 5 ou x := 5 (inférence de type)
- Les pointeurs
  - var iptr *int = &x

- Et les structures

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

- Mais pas de headers
  - Définitions et déclarations mélangées

Les structures de contrôle

- if … else

```
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")
}
```

- for

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

for j := 7; j <= 9; j++ {
    fmt.Println(j)
}

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

- switch

```
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")
}
```

Les fonctions

- Aggrégation de params

```
func plus(a int, b int) int {


    return a + b
}

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

- Valeurs de retour multiples
  - résultats + erreurs

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

```
a, b := vals()
```

```
_, c := vals()
```

C’est de l’OO ?

Les méthodes

- État + comportement (OO)
- Admet un type récepteur
- Agit sur l’état d’un type

```
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
}
```

C’est de l’OO ?

Les interfaces

- Notion de contrat (abstraction)
- Implémenter une interface

  =

  implémenter toutes ses méthodes

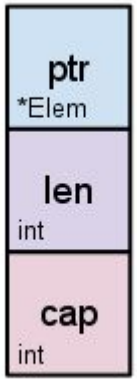
  ex:

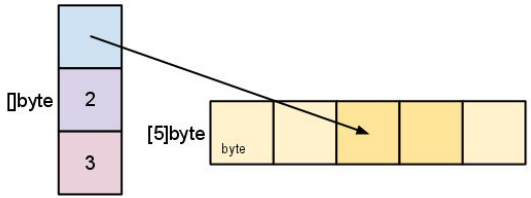
```
type error interface {
    Error() string
}
```

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

type rect struct {
    width, height float64
}
type circle struct {
    radius float64
}

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

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

Les types built-in

- Tableaux
  - var tab [5]int
  - var tab2D [2][3]int
- Et les **slices** ?
  - Associé à un tableau
  - typés par les éléments, pas la taille
  - ≅ vue sur une sous-partie du tableau





Les types built-in

- **map**
  - Ensemble non-ordonné de clés/valeurs
- **chan**

La concurrence *à la Hoare*

“Do not communicate by sharing memory. Share memory by communicating”

- Goroutines:
  - entités d’exécution
  - même espace d’adressage

- Channels:
  - Canaux de communication

Goroutines

- Lancer une goroutine
  - **go** *nom_fonction*

```go
package main

import "fmt"

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

func main() {

    f("direct")


    go f("goroutine")
```

Channels

- Bidirectionnel
  - ***chan T***
- envoyer des floats
  - ***chan<- float64***
- recevoir des entiers
  - ***<-chan int***
- Composition de canaux !
  - ***chan<- chan int***
  - ***<-chan <-chan int***

Exemple simple de communication

- *messages*: canal bidirectionnel
- Goroutine *func()*: envoie “ping” sur *messages*
- “ping” reçu dans la fonction main

```go
package main

import "fmt"

func main() {

    messages := make(chan string)


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



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