



Go

2-Type de base

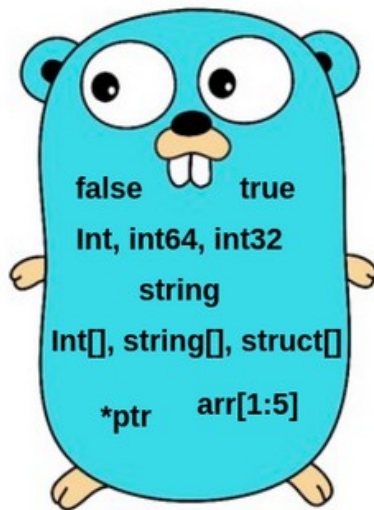
J. Vlasak

Go 2-Type de base

2

But du cours

- Savoir manipuler les variables en go.



Go 2-Type de base

3

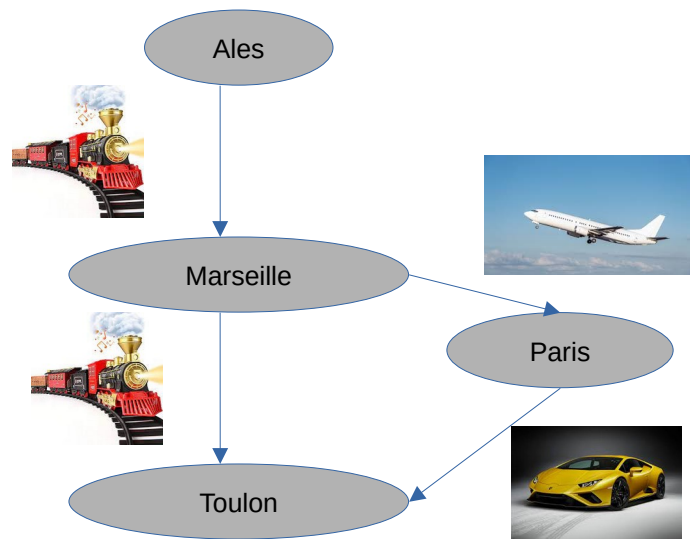
Visual Studio code

- Ouvrez une session de VSC car nous ferons des allers-retours entre le code et le cours



Souvenez vous de la machine de Turing

- Un problème se décrit sous la forme de graphes ou d'automates
- Un état sera mémorisé dans une variable ou des variables
- Une fonction ou instruction fera transiter l'automate, c'est à dire changer la valeur des variables
- Ici, par exemple l'état Ales peut se décrire par :
 - Coordonnées 44° 07' 41" nord, 4° 04' 54" est
 - Altitude Min. 116 m
 - Max. 356 m
 - Superficie 23,16 km²



Mémoriser un état : types de base

- Désigne les types définis dans le langage Go. Il s'agit des types les plus simples, qui ne sont pas dérivés d'autres types.
- Un état sera mémorisé dans une variable. Chaque variable occupera un espace mémoire (de 1 à n bits)
- Une fonction ou instruction fera transiter l'automate
- Ici, par exemple l'état Ales peut se décrire par :
 - Coordonnées 44° 07' 41" nord, 4° 04' 54" est
 - Altitude Min. 116 m
 - Max. 356 m
 - Superficie 23,16 km²



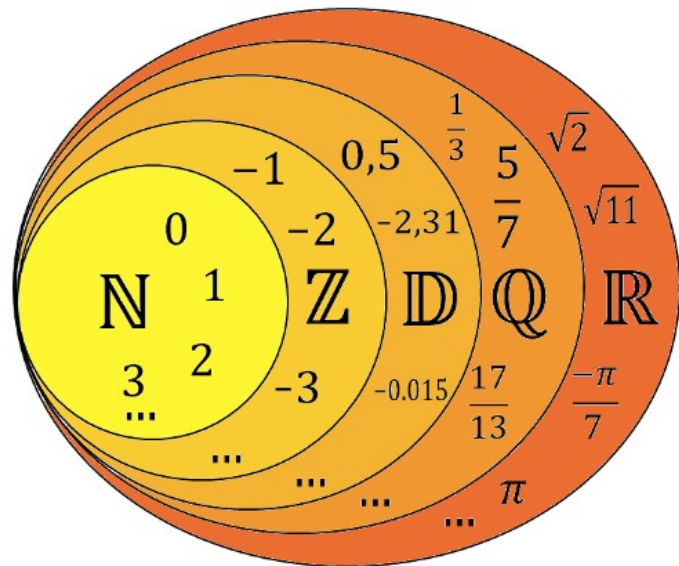
Go 2-Les types de base

6

Les types de base

- bool : type booléen
- int8, int16, int32, int64 : type entier
- uint8, uint16, uint32, uint64 : type entier non signé
- float32, float64 : type réel à virgule flottante
- complex64, complex128 : type complexe
- string : type chaîne de caractères, de taille variable.
- byte : alias pour le type uint8.
- rune : type caractère, de taille variable.

Toutes les variables sont initialisées à une valeurs par défauts, correspondant à la valeur « nulle » du type



IMPORTANT

Go 2-Les types de base

7

Valeur par défaut d'une variable

Nature de type	Valeur vierge
Booléens	false
Entiers signés ou non	0
Rune	'\x00'
Flottants réels et complexes	0.0 (== 0+0i)
Chaînes	""
Pointeurs	nil
Cartes	nil
Tranches	nil
Canaux	nil
Fonctions	nil
Interfaces	nil
Tableaux	Tableau valide où chaque élément est initialisé à la valeur vierge du type d'élément
Structure	Structure valide où chaque champ est initialisé à la valeur vierge de son type

Les booléens

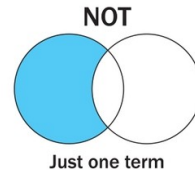
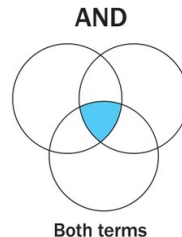
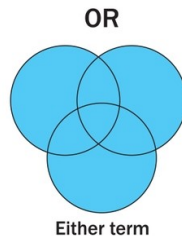
- Ne peut prendre que 2 valeurs: true et false.

```
var flag bool
var isAwesome = true

flag = true
```

- Valeur par défaut : false

BOOLEAN LOGIC



Les entiers

- Le type « int » existe, mais sa taille dépend du cpu. Ne pas l'utiliser !

```
var note int32
var pression = 64

note = 20
```

- Doit « caster » les variables pour les assignations entre type différents

```
var note int32
var pression int64 = 64
note = 20
pression = int64(note)
```

Type name	Value range
int8	−128 to 127
int16	−32768 to 32767
int32	−2147483648 to 2147483647
int64	−9223372036854775808 to 9223372036854775807
uint8	0 to 255
uint16	0 to 65536
uint32	0 to 4294967295
uint64	0 to 18446744073709551615

Go 2-Les types de base

10

Les opérations sur les entiers

Les classiques

$+$ $-$ \times \div $=$

$>$ \geq $<$ \leq \neq

Et le reste d'une division entière %

Handwritten long division of 132 by 26:

$$\begin{array}{r} 26 \overline{) 132} \\ \underline{-130} \\ 2 \end{array}$$

Labels: Dividende (132), Diviseur (26), Quotient (5), Reste (2).

Handwritten comparison of 7328 and 7346:

7328 et 7346

7 = 7
3 = 3
2 < 4

Donc
7328 < 7346

Opérateur et signe égale :

- $+=$, $-=$, $*=$, $/=$, et $\%=$.

Manipulation bit à bit :

- $\&=$, $|=$, $\^{}=$, $\&\^{}=$, $<<=$, et $>>=$

Le type réel à virgule flottante

- La déclaration des variables

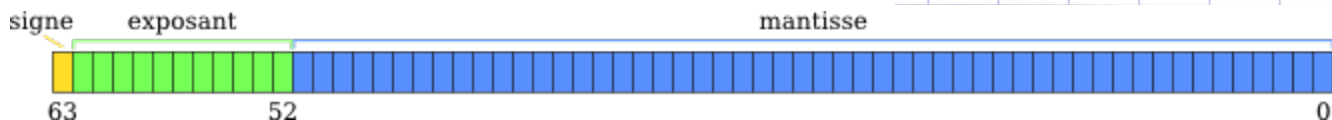
```
var pression float32
var temperature = 34.23
```

- Opérations classiques sur des nombres : +, *, /, -
- Échelle des grandeurs

Type name	Largest absolute value	Smallest (nonzero) absolute value
float32	3.40282346638528859811704183484516925440e+38	1.401298464324817070923729583289916131280e-45
float64	1.797693134862315708145274237317043567981e+308	4.940656458412465441765687928682213723651e-324

Le type réel à virgule flottante

- Codé selon la spécification IEEE 754



- Comparer des nombres flottants
 - Ne pas utiliser ==
 - Vérifier que l'écart des 2 chiffres est inférieur a epsilon

789,24

Partie entière: 789

Partie décimale: 0,24



Les complexes

- Opérations classiques sur des nombres : +, *, /, -

```
var x = complex(2.5, 3.1)
var y = complex(10.2, 2)

fmt.Println(x + y)
fmt.Println(x - y)
fmt.Println(x * y)
fmt.Println(x / y)
fmt.Println(real(x))
fmt.Println(imag(x))
fmt.Println(cmplx.Abs(x))
```

- Risque d'être supprimé dans les prochaines versions de go



Les runes et string

- Une rune représente un caractère. C'est un alias a « uint32 ». Permet de clarifier l'écriture du programme.

```
var runea rune= 'a'  
var runeaHexa rune= '\x61'  
var runeAunicode rune= '\U00000061'
```

- Un type string est un ensemble de rune.

```
var hello string = "hello!!"
```

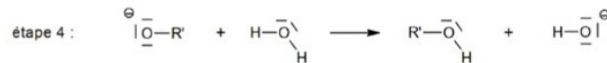
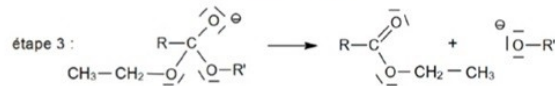
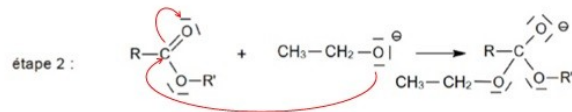
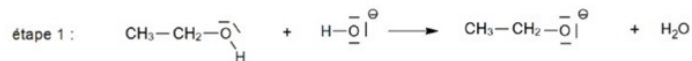
- Les opérations de comparaison ==, !=, >, >=, <, ou ≤ sont valide. De même que la concaténation.

Pas de conversion automatique en go

- Contrairement à d'autre langage, go impose d'expliciter toutes les conversions entre type

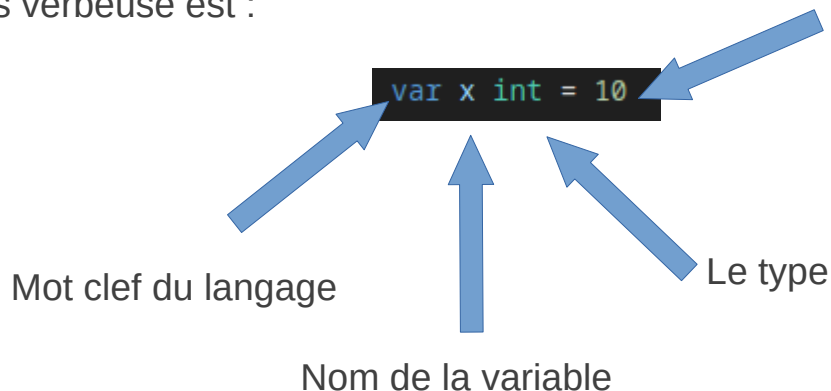
```
var x int = 10
var y float64 = 30.2
var z float64 = float64(x) + y
var d int = x + int(y)
```

- Permet de gagner en clarté et contrôle du résultat



Déclaration d'une variable

- Il existe plusieurs manière de déclarer une variable
- La plus verbeuse est :



Déclaration d'une variable

- Un peu moins verbeuse

```
var x = 10
```

- Et encore moins

```
x := 10
```

- Un fois le type définit, il n'est pas possible de le changer !!
- Ne peut assigner a x que des entiers.



Déclaration d'une variable

■ `x := 10` Ne s'emploie que dans une fonction, pas en variable globale

■ On peut déclarer plusieurs variables de même type `var x, y int = 10, 20`

■ Ou différent `var x, y = 10, "hello"`

■ En factorisant var

```
var (  
    x    int  
    y    = 20  
    z    int = 30  
    d, e  = 40, "hello"  
    f, g  string  
)
```

Déclaration d'un tableau

- Comme en C, il est possible de définir un tableau de type de base de longueur fixe

```
var a[3] int
var b[7] float32
```

- On manipule une valeur du tableau via l'opérateur [], et

```
a[0] = 1
b[2] = float32(a[0])
```

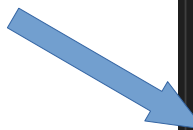
- Il est possible d'initialiser les donnée à la déclaration

```
b := [7]float32{0, 1, 2, 3}
```

Variable immuable

- Concept clef dans la programmation fonctionnel. On reviendra sur ce type de programmation
- Plus sur dans les contextes multi-thread.
- Programme un peu plus déterministe !

```
gaia@gaia:~/cours/tp1$ go build
# vjo/tp1
./main.go:18:2: cannot assign to x (neither addressable nor a map index expression)
./main.go:19:2: cannot assign to y (neither addressable nor a map index expression)
gaia@gaia:~/cours/tp1$
```



```
const x int64 = 10

const (
    idKey   = "id"
    nameKey = "name"
)

const z = 20 * 10

func main() {
    const y = "hello"

    fmt.Println(x)
    fmt.Println(y)
    x = x + 1
    y = "bye"
    fmt.Println(x)
    fmt.Println(y)
}
```

Cas particulier : variable anonyme

- La variable « `_` » a une signification particulière

```
_ = math.Cos(0.8)  
_ = 1000  
_ = "eeeeee"
```

- La valeur retournée par l'instruction est « oublié » et non utilisable



Visibilité d'une variable

- Une variable est définie à partir de sa déclaration jusqu'à la fin de son bloc
- Dans un nouveau bloc, il est possible d'avoir une variable de même nom. Mais ce n'est pas la même que celle du bloc parent !!

```
func main() {  
    var x int = 20  
  
    fmt.Println(x)  
    {  
        var x string = "interne bloc"  
        fmt.Println(x)  
    }  
}
```

Pointeur

- Les pointeurs s'utilisent comme en C
- Doit utiliser l'opérateur :
 - & : adresse
 - * : déclaration ou accès a la valeur de la variable pointée



```
var character byte = 10
var ptrCharacter *byte = &character
var ptrPtrCharacter **byte = &ptrCharacter

fmt.Println(ptrCharacter, character, ptrPtrCharacter)

*ptrCharacter = 20
```

Go 2-Variable

