LES VARIABLES DANS LE LANGAGE DE PROGRAMMATION GO

Définition

Avant de vous montrer comment déclarer et utiliser une variable, il faut d'abord comprendre ce qu'est une variable.

> **Rappel**
>
> *"Go est un langage à typage statique (les types des variables sont connus lors de la compilation et doivent être spécifiés expressément par le programmeur)"*

Une variable est le nom donné à un **emplacement mémoire** pour stocker une valeur, elle permet de **stocker des données**, ces données vont pouvoir être utilisées plus tard.

Une variable est constituée d'un **type**. Le type d'une variable permet :

- De déterminer la quantité d'espace occupé dans la mémoire.
- D'avertir le compilateur Go de quelle manière il doit interpréter votre variable

Les différents types de variables

il existe une multitude de types de variables en go, nous allons voir les types les plus utilisés.

Types entiers

Ce type de variable permet de stocker des **nombres entiers** : 1, 2, 3, 4...

Il existe deux catégories d'entiers :

- **unsigned** (non signé) : permet de stocker que des nombres positifs
- **signed** (signé): permet de stocker des nombres positifs et négatifs

Il y a plusieurs types d'entiers possibles, tout dépend de la valeur que vous voulez stocker dans votre variable. Je vous présente ci-dessous un tableau qui liste les types d'entiers qu'il est possible d'utiliser en GoLang ainsi que les nombres qu'ils peuvent conserver.

Type	Taille	Valeurs possibles
`uint8`	8 bits	0 à 255
`uint16`	16 bits	0 à 65535
`uint32`	32 bits	0 à 4294967295
`uint64`	64 bits	0 à 18446744073709551615
`int8`	8 bits	-128 à 127
`int16`	16 bits	-32768 à 32767
`int32`	32 bits	-2147483648 à 2147483647
`int64`	64 bits	-9223372036854775808 à 9223372036854775807

> **Information**
>
> le bit est la plus petite valeur informatique : soit 0 soit 1.

Types flottants

Ce type permet de stocker des **nombres décimaux** (nombres à virgule) : 1.5, 1.6, 128.5

Type	Taille	Précision
`float32`	32 bits	Environ 7 chiffres décimaux

Type	Taille	Précision
`float64`	64 bits	Environ 16 chiffres décimaux

Autres types numériques

Il existe également un ensemble de types numériques avec des tailles spécifiques :

Type	description
`byte`	Identique à uint8
`rune`	Identique à int32
`uint`	32 ou 64 bits non signé
`int`	Même taille que uint mais signé

Types booléens

Un **booléen** est un type de variables qui possède deux états :

- L'état « vrai » : **true** en anglais
- Ou bien l'état « faux » : **false** en anglais

C'est un type qui va nous permettre par exemple dans un jeu de savoir si est un joueur est vivant ou pas.

On utilise le mot-clé `bool` pour déclarer une variable de type booléen

Type string

C'est un type qui nous permet de stocker des **chaines de caractères** (du texte).

On utilise le mot-clé `string` pour déclarer une variable de type chaines de caractères

> **Information**
>
> Je n'ai cité que les variables plus utilisées mais il faut savoir qu'il existe d'autres types de variables que je ne vais pas détailler dans cette séance.

Déclaration des variables en GoLang

Comme la plupart des langages de programmation Il y'a certaines règles communes à respecter pour le nommage de vos variables :

- Les espaces sont interdits
- Les accents sont interdits
- Le nom de votre variable ne peut pas débuter par un chiffre

Voici comment on déclare une variable dans le langage Go :

```
var [nom_de_la_variable] [type]
```

Exemple :

```
package main

import (
    "fmt"
)

func main() {
    var vie int // déclaration de la variable vie
    fmt.Println(vie) // affichage de la variable vie
}
```

Par défaut la valeur d'un type int sera égale à 0

Résultat :

```
0
```

Pour changer la valeur par défaut de la variable, il suffit de la surcharger comme ci-dessous :

```
package main

import (
    "fmt"
)

func main() {
    var vie int = 12 // surchargement de la variable vie
    fmt.Println(vie)
}
```

Résultat :

```
12
```

Il est possible en Go de déclarer plusieurs variables dans la même ligne de code :

```
package main

import (
    "fmt"
)

func main() {
    var vie, argent, puissance int // déclaration de plusieurs variables sur une seul
    fmt.Println("vie : ", vie)
    fmt.Println("argent : ", argent)
    fmt.Println("puissance : ", puissance)
}
```

Résultat :

```
vie :  0
argent :  0
puissance :  0
```

Et même de les surcharger !

```
package main

import (
    "fmt"
)

func main() {
    var vie, argent, puissance int = 10, 20, 30 // surchargement des variables
    fmt.Println("vie : ", vie)
    fmt.Println("argent : ", argent)
    fmt.Println("puissance : ", puissance)
}
```

Résultat :

```
vie :  10
argent :  20
puissance :  30
```

Il est possible en Go de déclarer plusieurs variables avec des types différents.

Soit avec la syntaxe classique :

```
package main

import (
    "fmt"
)

func main() {
 var vie int = 20
 var nom string = "Default"
 var vitesse float32 = 5.4

    fmt.Println("vie : ", vie)
    fmt.Println("nom : ", nom)
    fmt.Println("vitesse : ", vitesse)
}
```

Résultat :

```
vie :  20
```

```
nom :  Default
vitesse :  5.4
```

Soit avec cette nouvelle syntaxe :

```
package main

import (
    "fmt"
)

func main() {
    var (
        vie     int     = 20
        nom     string  = "Default"
        vitesse float32 = 5.4
    )

    fmt.Println("vie : ", vie)
    fmt.Println("nom : ", nom)
    fmt.Println("vitesse : ", vitesse)
}
```

Résultat :

```
vie :  20
nom :  Default
vitesse :  5.4
```

Typage dynamique en Go

Il est tout à fait possible en GoLang de déclarer des variables dynamiquement (pas besoin de spécifier le type lors de la déclaration), dans ce cas on va laisser l'ordinateur choisir pour nous le type de variables en fonction de la valeur qu'on lui a transmise.

Pour déclarer une **variable dynamique**, il suffit de supprimer le mot-clé `var` et de remplacer le signe `=` par `:=`

Exemple :

```
package main

import "fmt"

func main() {
    /*
        Déclaration des variables dynamiques
    */
    flt := 15.5 //  sera automatiquement de type float
    in := 5 //  sera automatiquement de type int
    st := "hello" //  sera automatiquement de type string
    bol := true //  sera automatiquement de type boolean

    fmt.Printf("Le type de la varialbe flt est %T\n", flt)
    fmt.Printf("Le type de la varialbe in est %T\n", in)
    fmt.Printf("Le type de la varialbe st est %T\n", st)
    fmt.Printf("Le type de la varialbe bol est %T\n", bol)
}
```

Résultat :

```
Le type de la varialbe flt est float64
Le type de la varialbe in est int
Le type de la varialbe st est string
Le type de la varialbe bol est bool
```

Information

La fonction `Printf()` permet aussi d'afficher du texte et des variables mais contrairement à la fonction `Println()` la fonction `Printf()` permet d'afficher plus d'informations, il suffit de rajouter un symbole spécial à l'endroit où l'on veut afficher la variable.

Voici la description de certains symboles :

- %T : affiche le type d'une valeur

- %d : affiche un entier
- %s : affiche une chaîne de caractères
- %f : affiche un nombre décimal
- %b : affiche une représentation binaire

Les constantes

Par définition une **constante** possède une valeur fixe, elle ne peut en aucun cas être modifié. Voici comment on déclare une constante.

```
package main

import "fmt"

func main() {
    const maConstante int = 50 // déclaration d'une constante

    fmt.Println("ma Constante : ", maConstante)
}
```

Résultat :

```
ma Constante :  50
```

Tentons de modifier notre constante .

```
package main

import "fmt"

func main() {
    const maConstante int = 50

    maConstante = 50
```

```
    fmt.Println("ma Constante : ", maConstante)
}
```

Résultat :

```
.\test.go:8:14: cannot assign to maConstante
```

il nous envoie bouler. Il n'aime pas trop quand on tente de modifier ses constantes .

Les calculs

Il est possible avec le langage Go de faire des opérations très simples, telles quelles :

- L'addition
- La soustraction
- La multiplication
- La division
- Le modulo (le reste d'une division)

Les opérateurs de calcul

Il existe plusieurs manières de modifier la valeur d'une variable mathématiquement, l'une d'entre elles se fait par des opérateurs de calcul.

Opérateurs de calcul	Effet
`+`	Additionne deux valeurs
`-`	Soustrait deux valeurs
`*`	Multiplie deux valeurs
`/`	Divise deux valeurs
`%`	Calcul le reste de la division de deux valeurs

Exemple :

```
package main

import "fmt"

func main() {
    var a int = 4
    var b int = 2

    fmt.Println("a + b  = ", a+b) // addition de la variable a et b
    fmt.Println("a - b  = ", a-b) // soustraction de la variable a et b
    fmt.Println("a * b  = ", a*b) // multiplication de la variable a et b
    fmt.Println("a / b  = ", a/b) // division de la variable a et b
    fmt.Println("a % b  = ", a%b) // modulo de la variable a et b
}
```

Résultat :

```
a + b  =  6
a - b  =  2
a * b  =  8
a / b  =  2
a % b  =  0
```

Les opérateurs d'assignation

les opérateurs d'assignation permettent de modifier la valeur d'une variable mathématiquement et de stocker le résultat dans la variable.

Opérateurs d'assignation	Effet
`+=`	Additionne deux valeurs et stocke le résultat dans la variable
`-=`	Soustrait deux valeurs et stocke le résultat dans la variable
`*=`	Multiplie deux valeurs et stocke le résultat dans la variable
`/=`	Divise deux valeurs et stocke le résultat dans la variable
`%=`	Divise deux valeurs et stocke le reste dans la variable

Exemple :

```
package main

import "fmt"

func main() {
    var a int = 4
    var b int = 2

 a += b
 fmt.Println("a += b  = ", a)

 a -= b
 fmt.Println("a -= b  = ", a)

 a *= b
 fmt.Println("a *= b  = ", a)

 a /= b
 fmt.Println("a /= b  = ", a)

 a %= 3
 fmt.Println("a %= b  = ", a)
}
```

Résultat :

```
a += b  =  6
a -= b  =  4
a *= b  =  8
a /= b  =  4
a %= b  =  1
```

Les opérateurs d'incrémentation

Comme dans d'autres langages de programmation, il est possible sur Go de faire une **incrémentation** (augmentation d'une unité) et une **décrémentation** (diminution d'une unité) avec ce qu'on appelle des opérateurs d'incrémentation.

Opérateurs d'incrémentation	Syntaxe	Effet	Résultat (x := 9)
`++`	augmentation d'une unité de la variable x	x++	10
`--`	diminution d'une unité de la variable x	x--	8

Exemple :

```
package main

import "fmt"

func main() {
    var a int = 4

    a ++ // incrémentation
    fmt.Println("incrémentation de 1 : ", a)

    a -- // décrémentation
    fmt.Println("décrémentation de 1 : ", a)
}
```

Résultat :

```
incrémentation de 1 :  5
décrémentation de 1 :  4
```

Ne touche pas à mon type !

J'ai évoqué au début de ce cours que le type d'une variable permet entre autres de faire comprendre à notre compilateur Go de quelle manière il doit interpréter notre variable

Voyons voir ceci plus en détails en additionnant deux variables de type int :

```
package main

import "fmt"

func main() {
    var x int = 50
    var y int = 30

    fmt.Printf("x + y = ", x+y) //addition de deux variables de type int
}
```

Résultat :

```
x + y = %!(EXTRA int=80)
```

> **Information**
>
> Ne faites pas attention à au mot-clé **EXTRA**, l'important ici est le type de retour du calcul (ici un type int) et la valeur (ici 80)

Jusqu'ici tout va bien on obtient le résultat attendu, tentons maintenant d'additionner un type float32 avec int :

```
package main

import "fmt"

func main() {
    var x int = 50
    var y float32 = 30.5

    fmt.Printf("x + y = ", x+y) // addition d'une variable de type int et et une autr
}
```

Erreur :

```
invalid operation: x + y (mismatched types int and float32)
```

Le compilateur Go nous avertit qu'il n'arrive pas à calculer le résultat avec des types différents.

Pour éviter ce type erreur, il suffit soit de changer dés le début définitivement le type de la variable, soit de faire un **cast** notre variable c'est à dire de convertir la variable temporairement juste le temps de faire notre calcul pour pouvoir garder son type original (ici x int) sur la suite de notre code.

Je vais ici choisir la 2eme solution :

```
package main

import "fmt"

func main() {
    var x int = 50
    var y float32 = 30.5

    fmt.Printf("x + y = ", float32(x )+ y) // convertir le type de la variable x de i
}
```

Résultat :

```
x + y = %!(EXTRA float32=80.5)
```

Comme le montre le résultat le type de retour de notre calcul est bien de type float32 car nous avons additionné un type float32 avec un type int convertit en float32.