Mémento language Go (golang)

Barillot
Esteban

Présentation :
Le Golang est un langage de programmation compilé qui est rapide, statiquement typé et orienté objet. Il est aussi open-source et destiné à un usage général.

I. Structure générale

1. Implémentation du code

```
package main

import "fmt"

func main() {
   fmt.Println("Hello, World!")
}
```

2. Faire des commentaires

Pour déclarer un commentaire en une seule ligne, il suffit de mettre **//** avant le commentaire

exemple :

```
import "fmt" // importation du package fmt
```

Pour déclarer un commentaire en plusieures lignes, il suffit de mettre **/*** pour ouvrir le commentaire et ***/** pour fermer le commentaire.

exemple :

```
/*
   Votre commentaire
*/
```

II. Exécuter un fichier dans le terminal

1. Ouvrir le terminal VS Code

Appuyez sur : Ctrl + J

2. Exécuter le programme

Ecrivez dans le terminal : go run nom_du_fichier.go

III. Variables

1. Les types de variables

Type	Valeurs possibles	Commentaire
uint8 ou byte	0 à 255	Ensemble de tous les entiers non signés de 8 bits
uint16	0 à 65535	Ensemble de tous les entiers non signés de 16 bits
uint32 ou rune	0 à 4294967295	Ensemble de tous les entiers non signés de 32 bits
uint64	0 à 18446744073709551615	Ensemble de tous les entiers non signés de 64 bits
int8	-128 à 127	Ensemble de tous les entiers signés de 8 bits
int16	-32768 à 32767	Ensemble de tous les entiers signés de 16 bits
int32 ou uint ou int	-2147483648 à 2147483647	Ensemble de tous les entiers signés de 32 bits
int64 ou uint ou int	-9223372036854775808 à 9223372036854775807	Ensemble de tous les entiers signés de 64 bits
float32	Environ 7 chiffres décimaux	Ensemble de tous les nombres à virgule flottants 32 bits
float64	Environ 16 chiffres décimaux	Ensemble de tous les nombres à virgule flottants 64 bits
bool	true : vrai	Ensemble des valeurs

	false : faux	booléennes
string	Caractères ou chaînes de caractères.	

2. Les déclarations des variables

La déclaration d'une variable se fait sous cette forme :

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

Attention :

Il y a certaines règles à respecter, tels que :

- les **espaces** sont interdits
- les **accents** sont interdits
- le nom de votre variable ne peut pas débuter par un **chiffre**

exemple :

```go
package main

import (
    "fmt"
)

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

3. Les constantes

Une constante possède une valeur fixe, elle ne peut en aucun cas être modifiée.

```go
package main

import "fmt"

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

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

Résultat :

```
ma Constante :  50
```

IV. Les entrées et sorties

1. Instructions d'entrée

```
fmt.Scan(&variable)
```

2. Instruction de sortie

```
fmt.Println(variable)
```

V. Les calculs

1. Les opérateurs de calcul

Opérations	Symboles
additionne deux valeurs	+
soustrait deux valeurs	-
multiplie deux valeurs	*
divise deux valeurs	/
calcul le modulo (reste de la division euclidienne)	%

2. Les opérateurs d'assignation

Opérations	Symboles
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	%=

3. Les opérateurs d’incrémentation

Opérations	Symboles	Résultat (x:=9)
augmentation d'une unité de la variable x	++	10
diminution d'une unité de la variable x	--	8

4. Les opérateur de logique

Opérations	Symboles
ET logique	&&
OU logique	||

5. Les opérateurs de comparaison

Opérations	Symboles
==	Compare deux valeurs et vérifie leur égalité
<	Vérifie si la variable est strictement inférieure à une valeur
<=	Vérifie si la variable est inférieure ou égale à une valeur
>	Vérifie si la variable est supérieure inférieure à une valeur
>=	Vérifie si la variable est inférieure ou égale à une valeur
!=	Vérifie si la variable est différente à une autre valeur

VI. Les tableaux

1. Les tableaux

Déclarer le tableau

```
var tab = [10]int{0*10}
```

Attention ces tableaux ne sont pas dynamique c'est-à-dire que l'on ne peut pas ajouter de cases à celui-ci

2. Les tableaux dynamiques ou slices

1- Déclaration

Déclarer le tableau sous forme de slices

```
var tab = []int{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

Ce tableau peut ainsi être modifié

2- Ajouter un élément dans une slice

```
var mois []string

mois = append(mois, "Janvier")
fmt.Println(mois)

mois = append(mois, "Février")
fmt.Println(mois)
```

Résultat

```
[Janvier]
[Janvier Février]
```

3- Supprimer un élément dans une slice

```
mois := []string{"Janvier", "Février", "Mars", "Avril", "Juin",
"Juillet"}

fmt.Println(mois)

// index à supprimer de notre slice
indexASupprimer := 1

// Suppression de l'index 1 du tableau soit "Février"
mois = append(mois[:indexASupprimer], mois[(indexASupprimer+1):]...)
```

```
fmt.Println(mois)
```

Résultat

```
[Janvier Février Mars Avril Juin Juillet]
[Janvier Mars Avril Juin Juillet]
```

4- Copier le contenu d'une slice

```
animaux1 := []string{"Lion", "Cheval", "Ours"}
fmt.Println("Contenu du tableau animaux1 :", animaux1)

// création d'une slice cible avec la même taille que la slice source
animaux2 := make([]string, len(animaux1))

// copie du contenu de la slice source vers la slice cible
copy(animaux2, animaux1)

fmt.Println("Contenu du tableau animaux2 :", animaux2)
```

Résultat

```
Contenu du tableau animaux1 : [Lion Cheval Ours]
Contenu du tableau animaux2 : [Lion Cheval Ours]
```

VII. Les conditions

1. Si / Sinon

Le mot clé **si** est déclaré par le mot clé **if** et le mot clé **sinon** est déclaré par le mot clé **else**.

```
if condition {
  // exécution du code si la condition est passée
} else {
  // exécution du code si la condition n'est pas passée
}
```

2. Sinon Si

Le mot clé **sinon si** est déclaré par le mot clé **else if**.

```
if condition {
  // exécution du code si la condition est passée
} else if {
  // exécution du code si la 1ère condition n'est pas passée
} else {
  // exécution du code si toutes les conditions ne sont pas passées
}
```

VIII. Les boucles

1. Les boucles Pour

Pour déclarer une boucle Pour, il suffit d'utiliser le mot clé **for**.

```
for initialisation ; condition ; itération {
  /* le code qui sera répété */
}
```

- **initialisation** →initialise le compteur de la boucle
- **condition** → détermine la fin de la boucle
- **itération** → opération de l'indice d'initialisation

2. Les boucles While

Les boucles tant que ou while sont déclaré d'abords avec une boucle **for** sans condition d'arrêt, puis dans votre boucle for vous devez mettre une instruction conditionnelle qui permet de stopper votre boucle.

```
for {
  if condition {
     break // permet de stopper la boucle
  }
  ...
}
```

ou bien

```
for condition {
  ...
}
```

3. break et continue

Pour manipuler des boucles, il faut savoir qu'il existe des fonctionnalités tels que :

- **break** : qui permet d'interrompre une boucle, cette fonctionnalité nous permet donc de quitter la boucle
- **continue** : qui permet ici de revenir au début de la boucle

Ces fonctionnalités sont très souvent utilisées dans des boucles infinies.

exemples :

```
package main

import "fmt"

func main() {
   for i := 1; i <= 5; i++ {
       if i == 3 {
           break // Quitte la boucle lorsque i est égal à 3
       }
       fmt.Println(i)
   }
}
```

Dans cet exemple, nous utilisons une boucle `for` pour afficher les nombres de 1 à 5. Lorsque `i` est atteint la valeur 3, l'instruction `break` est exécutée, ce qui fait sortir de la boucle.

```go
package main

import "fmt"

func main() {
   for i := 1; i <= 5; i++ {
       if i%2 == 0 {
           continue // Ignore les nombres pairs et passe à l'itération
suivante
       }
       fmt.Println(i)
   }
}
```

Dans cet exemple, nous utilisons une boucle `for` pour afficher les nombres de 1 à 5, mais nous utilisons l'instruction `continue` pour ignorer les tous nombres pairs. Ainsi, on veut que les nombres impairs.

IX. Les fonctions et les procédures

1. Les procédures

Les procédures ne renvoient qu'un résultat imprimé, ce résultat ne peut donc pas être stocké.

```go
package main

import (
   "fmt"
)

func main() {
   var param type
   fmt.Println("param :")
   fmt.Scanln(&nombre)

   procedure(nombre)
}
func procedure(param type) {
   var return type
   // exécution du code
  fmt.Println(return) // ce que le code va imprimer au final
}
```

2. Les fonctions

Les fonctions se déclarent comme les procédures sauf qu'à la fin de la déclaration de la fonction nous devons mettre le type de l'élément que la fonction retourne.

À la fin de la fonction vous devez mettre une ligne de **return(variable)**

```
func fonction(param type) return type {
   var variable type
   return(variable)
}
```

Exemple :

Voici une fonction qui permanent d'inverser les cases d'un tableau

```
package main

import (
   "fmt"
)

func main() {
   var tab = []int{}

   var nbr int
   fmt.Println("Entrez le nombre :")
   fmt.Scanln(&nbr)

   for nbr != 0 {
       tab = append(tab, nbr)
       fmt.Println("Entrez le nombre :")
       fmt.Scanln(&nbr)
   }

   fmt.Println(tab)
   inverserTableau(tab)
}

func inverserTableau(tab []int) {
   var tab2 []int
   var taille int = len(tab) - 1
   for i := taille; i >= 0; i-- {
       tab2 = append(tab2, tab[i])
   }
```

```
    fmt.Println(tab2)
}
```

3. L’appel des fonctions

L’appel de d’une fonction se fait pareil que l’appel d’une procédure

```
fonction(param)
```

X. Importation fréquemment utilisées

1. Comment importer

Pour importer un module il est indispensable d’utiliser le mot clé import puis de mettre entre guillemets le module que l’on souhaite importer

```
import "fmt"
```

2. L’importation fmt

En Go (golang), l'importation du package `fmt` permet d'accéder à diverses fonctionnalités de formatage des entrées/sorties. Voici quelques-unes des possibilités offertes par l'importation de `fmt` (il en existe d’autre) :

XI. Formatage et affichage de texte :
- A. `fmt.Print(...)` : Affiche les arguments fournis sans saut de ligne.
- B. `fmt.Println(...)` : Affiche les arguments fournis avec un saut de ligne à la fin.
- C. `fmt.Printf(format, ...)` : Affiche les arguments suivant le format spécifié.
- D. `fmt.Sprintf(format, ...)` : Retourne une chaîne de caractères formatée sans l'afficher.

XII. Lecture depuis l'entrée standard :
- A. `fmt.Scan(&variable)` : Lit une valeur depuis l'entrée standard et la stocke dans la variable fournie.
- B. `fmt.Scanf(format, &variable)` : Lit les valeurs de l'entrée standard suivant le format spécifié et les stocke dans les variables fournies.
- C. `fmt.Scanln(...)` : Lit des valeurs depuis l'entrée standard jusqu'à un saut de ligne et les stocke dans les variables fournies.

XIII. Formatage d'autres types de données :
   A. `fmt.Sprintf("%d", x)` : Convertit un entier en chaîne de caractères.
   B. `fmt.Sprintf("%f", x)` : Convertit un nombre à virgule flottante en chaîne de caractères.
   C. `fmt.Sprintf("%s", x)` : Convertit une chaîne de caractères en chaîne de caractères (aucun changement).

XIV. Affichage de valeurs dans un format spécifique :
   A. `fmt.Printf("%d", x)` : Affiche un entier dans un format spécifique.
   B. `fmt.Printf("%f", x)` : Affiche un nombre à virgule flottante dans un format spécifique.
   C. `fmt.Printf("%s", x)` : Affiche une chaîne de caractères dans un format spécifique.

3. L'importation math

En Go (golang), l'importation du package `math` permet d'accéder à diverses fonctions et constantes mathématiques. Voici quelques-unes des possibilités offertes par l'importation de `math` (il en existe bien d'autres) :

1. Fonctions mathématiques courantes :
   - `math.Abs(x)` : Retourne la valeur absolue de `x`.
   - `math.Sqrt(x)` : Retourne la racine carrée de `x`.
   - `math.Pow(x, y)` : Retourne `x` élevé à la puissance `y`.
   - `math.Log(x)` : Retourne le logarithme naturel (base e) de `x`.
   - `math.Log10(x)` : Retourne le logarithme en base 10 de `x`.

2. Constantes mathématiques :
   - `math.Pi` : Pi (π).
   - `math.E` : Euler's number (e).

Exemple d'utilisation :

Pour un programme donnant la puissance d'un nombre avec un exposant

```
package main

import "fmt"

func main() {
    base := 2.0
   exposant := 3.0

   resultat := math.Pow(base, exposant)

   fmt.Printf("%.1f élevé à la puissance %.1f est égal à %.1f\n", base,
exposant, resultat)

}
```

XV. Exemple de programme et explications

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!") // afficher du texte
}
```

1. Le package main

```
package main
```

Les programmes GO ne peuvent s'exécuter que dans des packages, c'est une déclaration obligatoire pour chaque programme go. Il permet de définir le nom du paquet dans lequel le programme devrait se trouver. Le **package main** permet d'informer votre compilateur Go que le paquet doit être compilé en tant que programme exécutable.

2. Importation de la bibliothèque fmt

```
import "fmt"
```

Cette ligne permet d'informer votre compilateur qu'il est nécessaire d'importer la bibliothèque **fmt** avant d'exécuter le programme.

3. La fonction main

```
func main()
```

C'est une fonction nommée **main()** qui représente la **fonction principale** du début de l'exécution de votre programme.

fmt est une bibliothèque qui permet de formater votre texte

4. La fonction Println

```
fmt.Println (...)
```

Cette ligne nous permet ainsi d'imprimer / d'afficher quelque chose avec un saut de ligne à la fin.