GO Mémento

Opérateurs de comparaison :

Type	Opérations	Symbole
Caractères, Chaînes de caractères	Concaténations	&
	Comparaisons	!= ou ==
Entiers Réels	Arithmétique élémentaire	+ - * /
	Modulo	%
	Comparaisons	< > <= >= = !=
Booléens	Liaisons logiques	|| (OU) && (ET) ! (NON)
	Comparaisons	!= ou ==

Les base :

1 : Packages et Imports :

En Go, un package est une collection de fichiers source qui fournissent des fonctions et des types liés à un domaine spécifique. Chaque fichier source dans Go appartient à un seul package.

Exemple de code :

```
package main

import "fmt"
```

2 : Variable :

En Go, les variables sont déclarées en utilisant le mot-clé var suivi du nom de la variable.

Exemple de variables :

2.1 : Déclaration explicite avec type et valeur initiale :

```
var age int = 25
```

2.2 : Déclaration implicite avec type et valeur initiale :

```
name := "John"
```

La syntaxe := déduit automatiquement le type de la variable à partir de la valeur fournie.

2.3 : Déclaration explicite sans valeur initiale :

```
var count int
```

La variable est déclarée, mais elle n'a pas encore de valeur assignée.

2.4 : Déclaration multiple :

```
var x, y int = 10, 20
```

Vous pouvez déclarer et initialiser plusieurs variables sur une seule ligne.

2.5 : Déclaration de tableau :

```
var numbers [5]int
```

La déclaration d'un tableau de taille fixe.

1 : Entrées et sortie :

Lire :

```
fmt.Scanln (& nom_variable )
```

Ecrire :

```
fmt.Println ("Hello, World!")
```

2 : Instruction conditionnelles :

Si :

```
if a = b {
     toto + tata > 1
}
```

Si, sinon :

```
if a = b {
     toto + tata
      }
     else { toto + tata + tutu
}
```

3 : Boucles ou structure itérative :

Pour allant de *1 a *2 :

```
for nom_variable := 2 ; nom_variable < 100 ; nom_variable++ {

}
```

Tant que :

```
for {
     if x = y {
          break
     }
}
```

Faire jusqu'à :

```
if nom_variable != nil {
     fmt.Println ("Hello, World!")
     continue
}
```

4 : Les procédure :

Le langage Go n'utilise pas de procédure, comme d'autres langages de programmation. En Go, nous parlons généralement de fonctions plutôt que de procédures.

5 : Les fonctions :

Fonction de base :

Déclaration :

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

Appel :

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

Fonction avec arguments et valeur de retour :

Déclaration :

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

Appel :

```
func main() {
   result := addition(7, 8)
   fmt.Println("7 + 8 =", result)
}
```

Fonction anonyme :

Les fonctions anonymes sont utilisées lorsque nous voulons définir une fonction sans lui attribuer de nom,elles peuvent être déclarées et appelées directement depuis n'importe quelle bloc, elles peuvent aussi être utilisées en tant que paramètres.

```
func main() {
    func() {
        fmt.Println("Fonction anonyme !")
    }()
}
```

6 : Les slices (tableaux dynamique) :

Déclaration :

```
var Slice []int
```

Ajout d'éléments à la slice :

```
Slice = append(Slice, 1, 2, 3, 4, 5)
```

Affichage des éléments de la slice :

```
fmt.Println("Slice :", Slice)
```

Exemple de code :

```
func main() {

var Slice []int

Slice = append(Slice, 1, 2, 3, 4, 5)

fmt.Println("Slice :", Slice)

}
```