Strings

Une variable de type `string` est une chaîne de caractères. En Golang, vous pouvez créer des chaînes de caractères de deux façons différentes :

- En utilisant des guillements (`""`) :

Ici, les littéraux de chaîne sont créés à l'aide de guillemets doubles (`""`). Ce type de chaîne prend en charge le caractère d'échappement `\` , mais ne s'étend pas sur plusieurs lignes. Ce type de littéraux de chaîne est largement utilisé dans les programmes Golang. Vous pouvez également additioner deux chaînes afin de les joindre, on appelle ça une **concaténation**.

Exemple :

```
package main

import "fmt"

func main() {
    foo := "Hello World"
    fmt.Println(foo) // Hello World

    bar := "Hello World avec un guillement \""
    fmt.Println(bar) // Hello World avec un guillement "

    abc := "Hello avec un saut de ligne\nA la fin"
    fmt.Println(abc)

    def := foo + " " + bar
    fmt.Println(def) // Hello World Hello World avec un guillement "
    // Hello avec un saut de ligne
    // A la fin
}
```

- En utilisant des backticks (``) :

Ici, les littéraux de chaîne sont créés à l'aide de backticks (``). Ils ne prennent pas en charge les caractères d'échappement, peuvent s'étendre sur plusieurs lignes et peuvent contenir n'importe quel caractère, à l'exception du backtick. Il est généralement utilisé pour écrire des messages sur plusieurs lignes, dans les expressions régulières et en HTML.

Exemple :

```
package main

import "fmt"

func main() {
    foo := `Ici on peut écrire des chaînes de caractères
    sur plusieurs lignes, "c'est cool"`

    fmt.Println(foo)
}
```

Boucle for ... range

En Golang, la boucle **for ... range**, est une boucle qui va itérer sur chaque élément d'une variable itérable. C'est très utilisé car ça permet de boucler directement sur les éléments contenus dans la variable.

```
package main

import "fmt"

func main() {
    for INDEX, VALEUR := range VARIABLE_ITERABLE {
        // Instructions à exécuter à chaque tour
    }
}
```

Imaginons un tableau de nombres entiers : `[]int{1, 2, 3, 4, 5}`

On veut boucler sur chaque élément et afficher le numéro à chaque itération :

```
package main

import "fmt"

func main() {
    nums := []int{1, 2, 3, 4, 5}
    for _, numero := range nums {
        fmt.Println(numero) // Affiche 1 au 1er tour, puis 2 au 2e tour, puis 3...
    }
}
```

Vous pouvez noter que lorsque vous ne vous servez pas de l'index (numéro correspondant à l'itération en cours), vous pouvez l'omettre en mettant un underscore `_` à la place.

Imaginons maintenant que l'on veuille boucler sur la chaîne de caractère `Hello World` et que pour chaque lettre, on veut afficher la position de la lettre dans la string (index) ainsi que la lettre en question.

```
package main

import "fmt"

func main() {
    maString := "Hello World"
    for index, lettre := range maString {
        fmt.Println("La lettre", string(lettre), "se trouve en position", index)
    }
}
```

Par défaut, quand vous bouclez sur une chaîne de caractères, la valeur de la lettre sera un `int32` correspondant au numéro ASCII de la lettre.

Pour afficher la lettre, on doit faire ce que l'on appelle un cast : on change le type de la variable en faisant : `string(lettre)`. Ici on converti un nombre entier en string, grâce au numéro ASCII de la lettre.

Pointer

La notion de `pointer` est présente en Go mais aussi dans une multitude de langage. Il est donc important de comprendre ce que c'est et comment l'utiliser.

Qu'est ce qu'un Pointer ?

Un `pointer` est une variable qui va stocker l'adresse mémoire d'une variable qu'il va pointer. Un `pointer` d'un certain type ne peut pointer que sur une variable de ce même type.

Déclarer un pointer

Un `pointer` se déclare comme une variable classique, seul un symbole va être rajouter lors de la déclaration.

```
var ptr *type
var intPtr *int // Pointer vers un int
```

Le symbole `*` permet de définir un `pointer` à la déclaration, voyons maintenant comment lui définir une valeur :

```
var example int = 42

var intPtr *int
intPtr = &example
```

Le symbole `&` permet quant à lui d'accéder à l'`adresse mémoire` qui stock la variable que l'on souhaite, nous avons donc déclaré notre pointer et on lui a assigné l'adresse mémoire d'une variable existante.

Comme les variables classiques, il existe plusieurs moyen de déclarer et initialiser un `pointer`

```
package main

import (
    "fmt"
)

func main() {
    // Variable d'exemple :
    var example int = 42

    // Déclaration rapide :
    var ptr1 *int = &example
    fmt.Println(ptr1)  // 0x40e020

    // Déclaration très rapide :
    ptr2 := &example
    fmt.Println(ptr2)  // 0x40e020
}
```

Déréférencer un pointer

Stocker des adresses mémoires c'est bien mais ce qui va nous intéresser généralement c'est de pouvoir récupérer la valeur de la variable pointé par notre `pointer`, pour ce faire nous allons devoir `déréférencer` notre `pointer`

```
package main

import (
    "fmt"
)

func main() {
    var example int = 42
    var ptr *int
    ptr = &example

    fmt.Println(ptr)  // 0x40e020
    fmt.Println(*ptr) // 42
}
```

Rajouter le symbole `*` permet d'aller en profondeur dans notre `pointer` et ainsi récupérer la valeur de la variable pointé.

Il est possible de multiplier les étages de `pointer` pour faire des `pointers` de `pointers`, pour accéder à la valeur il faudra rajouter autant de `*` que d'étages de `pointers`

```
package main

import (
    "fmt"
)

func main() {
    i := 42
    j := &i  // j est un pointer d'int
    k := &j  // k est un pointer de pointer d'int

    fmt.Println(i)  // 42

    fmt.Println(j)  // 0x40e020

    fmt.Println(*j) // 42 (valeur contenue dans l'adresse)

    fmt.Println(k)  // 0x40c138

    fmt.Println(*k) // 0x40e020 (adresse de j)

    fmt.Println(**k) // 42
}
```

Exemples d'utilisation

Les `pointers` présentent l'avantage de pouvoir modifier des valeurs sans avoir besoin d'utiliser des retours de fonctions ou de les déclarer en global (qui est une mauvaise pratique).
Ils sont beaucoup plus intéressant sur des projets s'étirant sur de nombreuses fonctions ou qui ont besoin de modifier une même variable à plusieurs endroits du code.

```
package main

import (
    "fmt"
)

func add2(arg int){
    arg += 2
}

func add2ptr(arg *int){
    *arg += 2
}

func main() {
    example := 0
    add2(example)
    fmt.Println(example) // 0

    ptr := &example
    add2ptr(ptr)
    fmt.Println(example) // 2
}
```