Boucles

On va voir aujourd'hui qu'est-ce qu'une boucle et pourquoi s'en servir.

Premier exemple d'utilisation d'une boucle:

```
Imaginons que vous voulez rajouter une valeur et l'augmenter d'un cran à chaque
fois théoriquement on pourrait faire comme cela :
```

```go
package main

func main() {
    result := 0 + 1 + 2 + 3 + 4
}
```

```
Comme vous pouvez le voir cela fonctionne mais reste exhaustif pour nous de taper
tout à la main, pour ce faire il existe plusieurs façons d'y remédier.
```

Boucles itératives

Pour remplacer le code vu précédemment avec une boucle itérative il faudrait faire comme suivant:

```go
package main

func main() {
    result := 0
    for i := 0; i < 4; i++ {
        result += i
    }
}
```

Ce code la revient à faire la même chose qu'on a faites au début.

Comme vous pouvez le constater il y a plusieurs parties:

```
'i := 0' est la partie qu'on désignera comme une déclaration. On veut déclarer une nouvelle variable qu'on va vouloir assigner à 0.

'i < 4' est la condition de la boucle, c'est celle qui va déterminer la fin de boucle sinon notre boucle tournera à l'infini.

'i++' est l'opération, c'est la partie qui s'exécute à chaque fin de tour de boucle.
```

Pour résumer un peu tout ça, une boucle possède donc: une partie déclaration, une partie condition et une partie opération.

Refaisons un exemple: On souhaite additionner tous les nombres pairs entre 0 et 20 inclus

```go
package main

func main() {
    result := 0
    for i := 0; i <= 20; i + 2 {
        result += i
    }
}
```

Donc comme vous pouvez le voir, on démarre à 0 et mon opération fait + 2 à chaque nouvelle tour de boucle ce qui me permet de sauter les nombres impairs. Deplus ma condition s'arrête lorsque j'aurai atteint 20 tout en l'ajoutant.

Boucles récursives

Maintenant que nous avons vu les boucles itératives, il existe une autre manière de boucler celle de faire une boucle récursive.

Ce qu'il faut tout d'abord savoir sur une boucle récursive :

```
C'est une fonction qui se rappelle elle-même
```

Attention !

La fonction se rappelle elle-même mais si vous ne mettez pas de condition pour qu'elle s'arrête cela fera une boucle infini !

Exemple :

```
package main

func main() {
    RecursiveFunction()
}

func RecursiveFunction() int {
    return RecursiveFunction()
}
```

Pour pallier à ça on va rajouter un paramètre pour lui permettre de lui dire quand il doit s'arrêter.

```
package main

func main() {
    RecursiveFunction(8)
}

func RecursiveFunction(int number) int {
    return RecursiveFunction(number)
}
```

Donc maintenant cela devrait fonctionner.

Et non ! Attention on lui a juste rajouter un paramètre mais on ne lui dit toujours pas quand il doit s'arrêter.

Pour ce faire on va lui rajouter une condition qui va vérifier la valeur du paramètre, car c'est ce qu'on va faire varier.

```
package main

func main() {
    RecursiveFunction(8)
}

func RecursiveFunction(int number) int {
    if (number < 2) {
        return RecursiveFunction(number-1)
    }
    return 0
}
```

Comme vous pouvez le constater, nous avons rajouté une condition qui vient vérifier le paramètre s'il est inférieur a 2.

Par dessus cela lorsque l'on appelle la fonction on lui passe le paramètre - 1, car sinon le paramètre ne varierai jamais et cela ferait comme une boucle infini.

Vous avez maintenant toutes les clés en main pour pouvoir réussir cette quête !

Bon courage à tous !

N'oubliez pas de toujours tester votre code en locale !