LES FONCTIONS DANS LE LANGAGE DE PROGRAMMATION GO

Explication

Les fonctions vont nous permettre de structurer nos programmes en un groupe d'instructions qui effectueront un ensemble de tâches.

Elles permettent de **simplifier** notre code et de le rendre beaucoup plus **lisible** que ce soit pour nous ou pour les autres, mais surtout (rappelez vous que vous êtes un bon flemmard) elles nous permettrons de ne pas retaper le même code plusieurs fois d'affiler.

> **Information**
>
> La bibliothèque standard Go fournit de nombreuses fonctions intégrées que votre programme peut appeler comme par exemple la fonction `Println()` de la bibliothèque fmt.

Déclarer une fonction

Pour déclarer une fonction il nous faut :

- **Le nom de la fonction** (non obligatoire si votre fonction est anonyme, on verra les fonctions anonymes à la fin de ce chapitre) : C'est le nom qui décrit votre fonction, il faut juste penser à respecter les mêmes règles que pour les variables (pas d'accents, pas d'espaces, etc.). GoLang vous recommande de

nommée vos fonctions en Camel case, c'est à dire que chaque mot commence par une majuscule à l'exception du premier.

- **Le type de retour de la fonction** (non obligatoire si votre fonction ne retourne rien) : comme les variables les fonctions ont un type, plus précisément c'est le type de la valeur qu'elle retourne.
- **Des paramètres** (non obligatoire) : Ce sont des variables que la fonction va exploiter dans son bloc de code.

```
func nomDeLaFonction( liste_de_paramètres ) type_de_retour
{
    /* votre code */
}
```

> **Avertissement**
>
> Une fonction non anonyme doit être déclarée en dehors de la fonction `main()`

Fonction sans type de retour et sans paramètres

Voici comment on déclare une fonction sans type de retour et sans aucun paramètre.

```
package main

import (
    "fmt"
)

// déclaration de la fonction affichage()
func affichage() {
    fmt.Println("###################################")
    fmt.Println("\tBonjour")
    fmt.Println("###################################")
```

```
}

func main() {
    affichage() // appel de la fonction affichage()
}
```

Résultat :

```
##################################
        Bonjour
##################################
```

Fonction sans type de retour mais avec des paramètres

Une fonction est capable de prendre autant de paramètres que vous voulez et peu importe leurs types.

```
package main

import (
    "fmt"
)

// prend en paramètre un type string et un int
func affichage(nom string, age int) {
    fmt.Println("Bonjour", nom, "vous avez", age, "ans")
}

func main() {
    affichage("Hatim", 9)
    affichage("Alex", 12)
}
```

Résultat :

```
Bonjour Hatim vous avez 9 ans
Bonjour Alex vous avez 12 ans
```

Fonction avec un type de retour

On utilisera le mot-clé `return` pour renvoyer une valeur depuis notre fonction. Vous pouvez ensuite stocker la valeur retournée par votre fonction dans une variable.

```
package main

import "fmt"

// Fonction qui retourne un type int
func maxNbr(a int, b int) int {
    if a > b {
        return a // retourne le variable a de type int
    }
    return b // retourne le variable b de type int
}
func main() {
    max := maxNbr(10, 30) // stockage du retour de la fonction dans une variable
    fmt.Println(max)

    // Utilisation directe du retour de la fonction sans la stocker dans une variable
    fmt.Printf("Valeur : %d , Type : %T\n", maxNbr(50, 30), maxNbr(50, 30))
}
```

Résultat :

```
30
Valeur : 50 , Type : int
```

Je veux plus de valeurs !

Sur l'exemple précédant notre fonction ne retournait qu'un seul type de retour, dès fois nous aurons besoin de retourner plus de valeurs. GoLang nous propose cette fonctionnalité pour renvoyer autant de valeurs souhaitées.

```
package main

import (
    "fmt"
)
```

```
func main() {
    a := 5
    b := 8
    fmt.Println("Avant fonction a =", a, " b =", b)
    a, b = additionTrois(a, b) // stockage du retour de la fonction dans deux variabl
    fmt.Println("Après fonction a =", a, " b =", b)
}

// retourne deux types int
func additionTrois(a int, b int) (int, int) {
    return a + 3, b + 3
}
```

Résultat :

```
Avant fonction a = 5  b = 8
Après fonction a = 8  b = 11
```

Je veux plus de paramètres !

Il est possible en Go de déclarer une fonction qui peut prendre en compte des paramètres **infinis** aussi appelés paramètres **variadiques** .

Pour ce faire il faut rajouter des trois points `...` collés avant le type du paramètre.

Exemple :

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println(addition(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13))
}

// déclaration d'une fonction avec des paramètres infinis
func addition(param ...int) int {
    total := 0
    for _, value := range param { //j'ai mis un underscore "_" car je ne souhaite pas
        total += value
    }
    return total
}
```

Résultat :

```
91
```

Une 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 bloc de votre code enfin elles peuvent aussi être utilisées en tant que paramètres.

Je vais dans cet exemple déclarer et utiliser une fonction anonyme en tant que paramètre.

```
package main

import (
    "fmt"
    "math"
)

// Déclaration d'une fonction qui prend en paramètres un float64 et une fonction anon
func rajouterDix(a float64, fAnonyme func(float64) float64) /**/ {
    operation := fAnonyme(a) // Appel à notre fonction anonyme
    result := operation + 10
    fmt.Println(result)
}

func main() {
    // stockage de notre fonction anonyme dans une variable
    racineCarree := func(x float64) float64 { return math.Sqrt(x) }
    rajouterDix(9, racineCarree)

    /*
        il est possible aussi d'utiliser directement une fonction anonyme
        dans une variable sans forcement la stocker dans une variable
    */
    rajouterDix(5, func(x float64) float64 { return math.Pow(x, 2) })
}
```

Résultat :

```
13
35
```