Table des matières

1. Snippet d'exercice

```
package main

import "fmt"

func main() {

  var test = "Michel";

  nom := "Christophe";

  tableau := [3]string{test, nom, "Paul"}

  age := map[string]string{
    tableau[0]: "22",
    tableau[1]: "24",
    tableau[2]: "18",
  }

  if i := 10; i > 0 {
    tableau[0] = "42"
  }

  fmt.Println("Salut,",
        test + " (" + age["Michel"] + " ans),",
        nom + " (" + age[nom] + " ans) ainsi que",
        tableau[2])
}
```

2. Typage

Il existe 2 manières de déclarer une variable:

```
var <name> <type> = <value>
```

où

```
<name> := <value>
```

> ⚠ **Attention**
> La seconde méthode fonctionne uniquement pour une variable locale muable.

2.1. Tableaux

```
var <name> [<size>]<type> = [<size>]<type>{<value1>, <value2>, ...}

var tab [3]string = [3]string{"Michel", "Christophe", "Paul"}
name := [3]string{"Michel", "Christophe", "Paul"}
```

2.2. Maps

```
var <name> map[<key_type>]<value_type> = map[<key_type>]<value_type>{
  <key1>: <value1>,
  <key2>: <value2>,
  ...
}

var age map[string]string = map[string]string{
  "Michel": "22",
  "Christophe": "24",
  "Paul": "18",
}
```

Ensuite nous pouvons soit accéder à une valeur via sa clé:

```
age["Michel"] // "22"
```

Soit modifier une valeur via sa clé:

```
age["Michel"] = "42"
```

3. Conditionnelles

3.1. If

```
if <condition> {
  // code
} else if <condition> {
  // code
} else {
  // code
}
```

Il est possible d'initialiser une variable dans la condition:

```
if i := 10; i > 0 {
  // code
}
```

3.2. Switch

De même que pour les if, il est possible d'initialiser une variable dans la condition:

```
switch i := 10; i {
case 1:
  // code
case 2:
  // code
default:
  // code
}
```

3.3. Boucles

La boucle for est la seule boucle en Go. Elle peut être utilisée de plusieurs façons:

3.3.1. For classique

```
for i := 0; i < 10; i++ {
  // code
}
```

3.3.2. Foreach

```
for index, value := range <array_or_map> {
  // code
}
```

où `index` est l'index de l'élément dans le tableau ou la clé dans la map, et `value` est la valeur de l'élément.

3.3.3. While

```
for <condition> {
  // code
}
```

3.3.4. Boucle infinie

```
for {
  // code
}
```

4. Fonctions

Une des particularités de Go est que les fonctions peuvent retourner plusieurs valeurs:

```
func <name>(<param1> <type1>, <param2> <type2>, ...) (<return_type1>, <return_type2>, ...) {
  // code
  return <value1>, <value2>, ...
}
```

Cela peut s'avérer très pratique pour retourner une valeur et une erreur:

```
func divide(a int, b int) (int, error) {
  if b = 0 {
    return 0, fmt.Errorf("division by zero")
  }
  return a / b, nil
}
```

📝 **Note**

En Go, il est aussi possible d'utiliser de donner qu'un seul type à plusieurs paramètres consécutifs:

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

cela est aussi valable pour les types de retour.

4.1. Fonction anonyme

Il est possible de définir une fonction sans lui donner de nom:

```
func(<param1> <type1>, <param2> <type2>, ...) (<return_type1>, <return_type2>, ...) {
  // code
  return <value1>, <value2>, ...
}
```

Cela peut être utile pour passer une fonction en argument à une autre fonction:

```
var add = func(a int, b int) int {
  return a + b
}
```

5. Structures et composition

5.1. Déclaration de types

5.1.1. Nouveau type

```
type <new_type> <existing_type>
```

quelques exemples:

```
type intGenerator func() int

type image [][]uint32

type celsius float64
type farenheit float64

var ebulitionC celsius = 100
var ebuilitonF farenheit = ebulitionC // ERROR : Cannot assign celsius to farenheit
```

5.1.2. Alias de type

```
type <new_type> = <existing_type>
```

quelques exemples:

```
type celsius = float64
type farenheit = float64

var ebulitionC celsius = 100
var ebuilitonF farenheit = ebulitionC // Correct ; même type : float64
```

5.2. Structures

```
struct {
  <field1> <type1>
  <field2> <type2>
  //...
}
```

quelques exemples:

```
type Person struct {
  name string
  age int
}

// Création d'une struct avec champs "zéros"
var fred Person // {"", 0}

// Création avec valeurs initiales fournies
var fred = Person{"fred", 99} // {"fred", 99}

// Création avec valeurs initiales fournies et nommées, ordre arbitraire
var fred = Person{age: 99, name: "fred"} // {"fred", 99}

// Création avec valeurs initiales partiellement fournies, "zéros" ailleurs
var fred = Person{name: "fred"} // {"fred", 0}
```

📝 **Note**

Note: les règles d'export s'appliquent aux champs: pour qu'un champ soit public en dehors du package, il doit commencer par une majuscule.

5.2.1. Méthodes

Il est possible de définir des méthodes sur des types définis par l'utilisateur:

```
func (<name> <type>) <name> (<parameters>) <return_types> {
  // code
}
```

quelques exemples:

```
func (p Person) Majeur() bool {
  return p.age >= 18
}

if fred.Majeur() {}
```