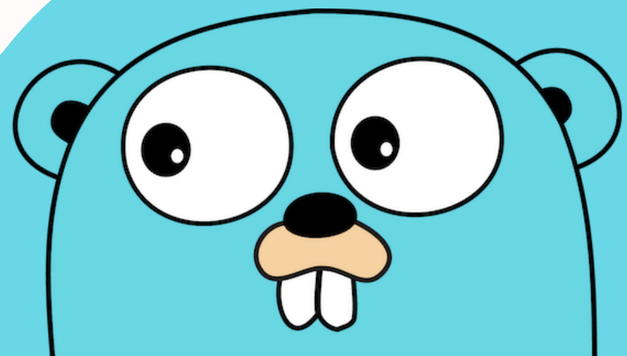
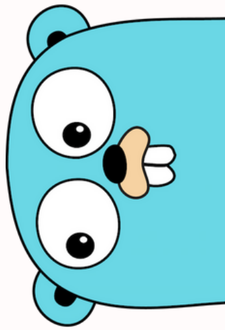
AIX ynov CAMPUS

CAPSULE

LES TABLEAUX ET MAPS

Par Cyril RODRIGUES



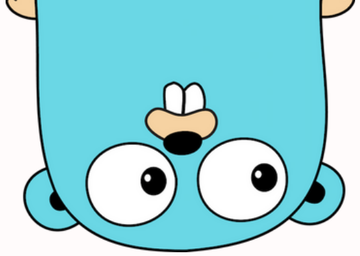


LES OBJECTIFS

Cette capsule a pour objectif de vous familiariser avec deux structures de données fondamentales du langage Go : les tableaux et les maps. À l'issue de cette leçon, vous serez capable de :

- Déclarer, initialiser et manipuler des tableaux.
- Parcourir des tableaux unidimensionnels et multidimensionnels.
- Déclarer, initialiser et manipuler des maps pour stocker des paires clé/valeur.

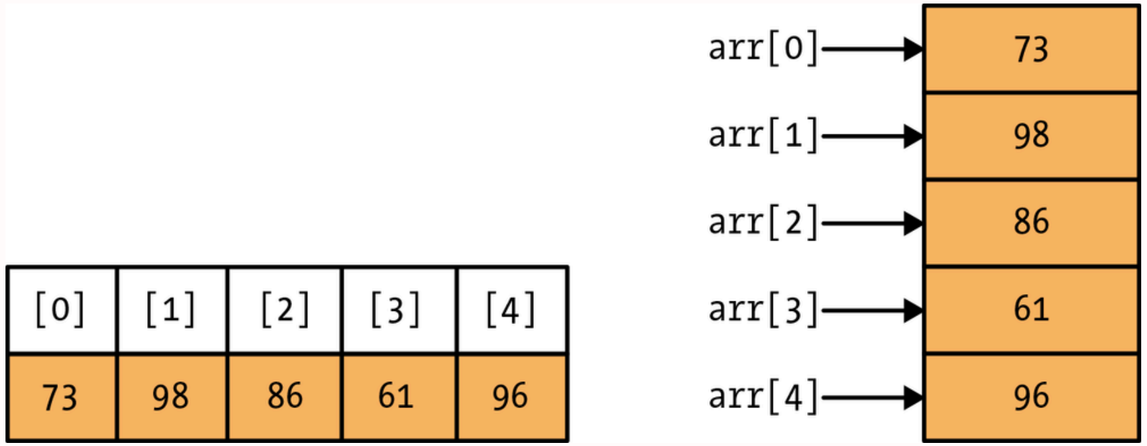
LES TABLEAUX



QU'EST CE QU'UN TABLEAU ?

Un tableau est un type de variable qui peut contenir une collection de données indexées du même type. On peut facilement accéder à un élément grâce à son index soit sa position. Les index dans un tableau commencent à 0, donc l'index du premier élément d'un tableau est 0. En Go, on distingue deux types de tableaux :

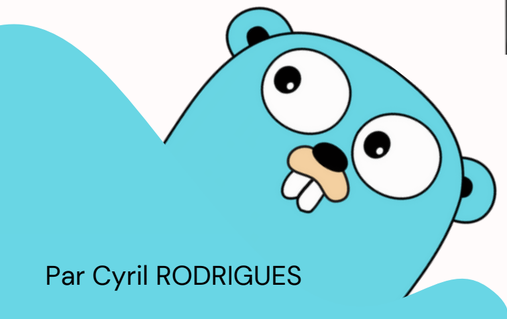
- Array (tableau statique), tableau offrant une taille fixe
- Slice (tableau dynamique), tableau offrent une taille modulable à l'exécution.



LES TABLEAUX STATIQUES (ARRAYS)

Un tableau statique ou array, possède une taille fixe déterminée lors de sa déclaration qui est indiquée entre crochets. Cette taille ne peut pas être modifiée par la suite. Si aucune initialisation n'est précisée, chaque élément est automatiquement assigné à la valeur par défaut de son type. Voici l'anatomie d'une déclaration et/ou initialisation de tableaux :

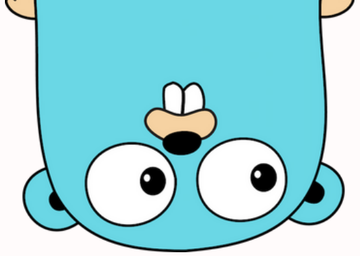
```
// Déclaration implicite ¬
var NomDuTableau [Taille]Type¬
NomDuTableau = [Taille]Type{valeurs...}¬
¬
var NomDuTableau [Taille]Type = [Taille]Type{valeurs...}¬
¬
// Déclaration explicite ¬
NomDuTableau := [Taille]Type{valeur...}¬
```



*On peut remplacer la taille entre crochets par ... pour laisser le compilateur déterminer automatiquement la longueur du tableau en fonction du nombre d'éléments fournis. Uniquement déclaration inférée !

EXEMPLE

```go
package main

func main() {
    // Exemples de déclarations implicites
    var listeEtudiants [2]string
    // listeEtudiant : ["",""]

    listeEtudiants = [2]string{"Cyril", "Kheir"}
    // listeEtudiant : ["Cyril","Kheir"]

    var nbrChance [4]int = [4]int{5, 8, 9, 7}
    // nbrChance : [5,8,9,7]

    //Exemples déclarations inférés
    checkList := [2]bool{true, false}
    // checkList : [true,false]

    banUsers := [4]string{"Alan", "Enzo", "Lucas", "Tristan"}
    // banUsers : ["Alan","Enzo","Lucas","Tristan"] }
}
```



LES TABLEAUX DYNAMIQUES (SLICES)

Un tableau dynamique aussi appelé slices, est un type de tableau qui ne possède pas de taille prédéfinie. Sa taille peut varier selon le nombre d'éléments qu'ils contiennent, ce qui permet d'ajouter ou de supprimer des éléments selon le besoins celui-ci s'ajoutera automatiquement. Voici l'anatomie d'une déclaration et/ou initialisation de tableaux :

```
// Déclaration implicite ¬
var NomDuTableau []Type¬
NomDuTableau = []Type{valeurs...}¬
¬
var NomDuTableau []Type = []Type{valeurs...}¬
¬
// Déclaration explicite ¬
NomDuTableau := []Type{valeur...}¬
```

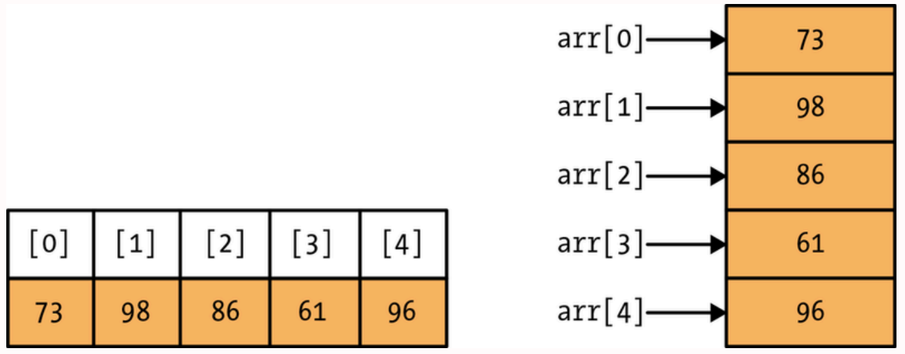
EXEMPLE

```go
package main

func main() {
    // Exemples de déclarations implicites
    var validateCheck []bool
    // validateCheck : []
    validateCheck = []bool{true, true, false}
    // validateCheck : [true,true,false]
    var drinksList []string = []string{"Coca-cola", "Pepsi", "Orangina"}
    // drinksList : [“"Coca-cola","Pepsi”,"Orangina”]
    //Exemples de déclarations inférés
    listCityCampus := []string{"Aix-en-Provence", "Paris", "Bordeaux"}
    // listCityCampus : ["Aix-en-Provence","Paris","Bordeaux"]
    listNotes := []int{10,16,20,19}
    // listNotes : [10,16,20,19]
}
```

ACCÉDER À UN ÉLÉMENT D'UN TABLEAU

Pour accéder à une valeur d'un tableau, on utilise l'index. Cela fonctionne aussi bien pour les tableaux dynamiques (slices) que pour les tableaux statiques (array). On peut accéder à une valeur pour la lire ou bien la modifier en indiquant le nom du tableau suivi de l'index (position) de la valeur désirée entre crochets. Voici des exemples pour illustrer comment accéder et modifier les éléments d'un tableau ou d'un slice.

```
// Pour lire la valeur
NomDuTableau[index]
// Pour modifier la valeur
NomDuTableau[index] = valeur
```



*Il ne faut pas oublier que l'indexage commence à 0.

RÉCUPÉRER
LA TAILLE D'UN TABLEAU

Pour récupérer la taille ou bien le nombre d'éléments dans un tableau qu'il soit dynamique (slice) ou statique (array), on peut utiliser la fonction len(). Cette fonction retourne un entier correspondant au nombre d'éléments dans le tableau.

```
// Retourne un entier correspondant au nombre d'éléments dans le tableau
len(NomDuTableau)
```

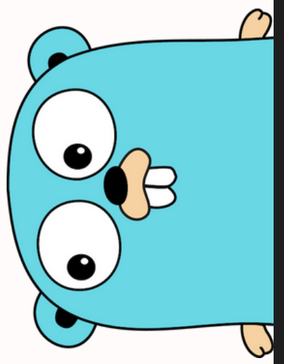
*La position du dernier élément sera donc len(NomDuTableau) – 1

EXEMPLE

```go
func main() {
    // Déclaration tableau "arr"
    arr := []int{10, 20, 30, 40, 50, 60}
    // Récupération du nombre d'éléments dans le tableau
    fmt.Println(len(arr))
    // 6 car le tableau arr contient 6 éléments du type entier
    // Modification de la valeur a l'index 3
    arr[3] = 0
    fmt.Println(arr)
    // [10,20,30,0,50,60] la valeur a 1l'index trois a été changée fmt.Printin(arr[4])
    fmt.Println(arr[4])
    // 50 puisque à l'index 4 se trouve la valeur 50
}
```

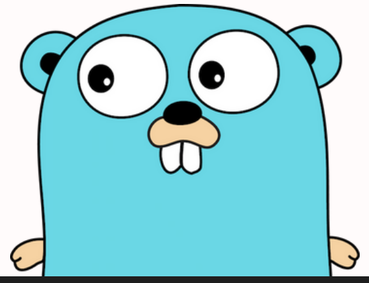
LES TRANCHES D'UN TABLEAU

Une tranche permet d'extraire une portion d'un array ou d'un autre slice en Go, grâce à l'opérateur de découpage ":". Vous pouvez spécifier un intervalle d'indices pour créer un nouveau slice qui reprend les éléments dont les indices vont de la borne de début (incluse) à la borne de fin (exclue).



```
// Pour récupérer tous les éléments
NomDuTableau[:]

// Pour récupérer les éléments jusqu'à Y (Y exclu)
NomDuTableau[:Y]  // indices 0 à Y-1

// Pour récupérer les éléments à partir de X (X inclus)
NomDuTableau[X:]  // indices X à len(NomDuTableau)-1

// Pour récupérer les éléments de X (inclus) à Y (exclu)
NomDuTableau[X:Y] // indices X à Y-1
```

EXEMPLE



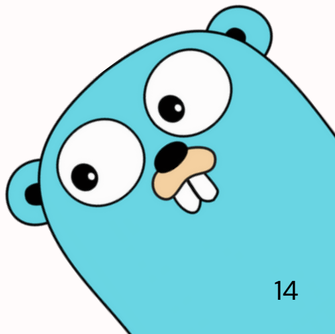
```go
func main() {
    // Exemple des tranches
    arr := [6]int{10, 20, 30, 40, 50, 60}

    fmt.Println(arr[:3])
    // [10,20,30] *car jusqu'à l'index 3 est exclu
    fmt.Println(arr[1:])
    // [20,30,40,50,60] *car à partir de l'index 1
    fmt.Println(arr[3:5])
    // [40,50] car à partir de l'index 3 jusqu'à l'index 5 exclu
}
```

AJOUTER UN ÉLÉMENT À UN SLICE

En Go, on utilise la fonction append pour créer un nouveau slice composé des éléments existants et de ceux que l'on souhaite ajouter. Que vous vouliez insérer une seule valeur, plusieurs valeurs, ou concaténer deux slices, append gère automatiquement la création du nouveau slice.

```
// Ajouter une seule valeur
NomDuTableau = append(NomDuTableau, Valeur)
// Ajouter plusieurs valeurs
NomDuTableau = append(NomDuTableau, Valeurl, Valeur2, Valeur3)
// Ajout de plusieurs valeurs a partir d'un autre tableau slice
NomDuTableau = append(NomDuTableau, TableauDeNouvellesValeurs...)
// L'opérateur "..." permet de récupérer les valeurs une a une donc de les
sortir du tableau (utilisable uniquement sur les tableaux dynamiques).
```

*append renvoie toujours un nouveau slice, n'oubliez donc pas d'affecter son résultat à votre variable.



SUPPRIMER UN ÉLÉMENT D'UN SLICE

Pour supprimer une valeur d'un tableau dynamique (slice), nous allons utiliser la fonction append en combinaison avec des tranches (slicing). Cette méthode consiste à créer un nouveau slice en omettant l'élément que l'on souhaite supprimer. Pour effectuer cela on récupérer tous les éléments avant l'index à supprimer, puis tous ceux après dans l'objectif de les concaténer. Voici comment faire :

```
// Permet de supprimer un élément se trouvant a l'index "IndexSup"
NomDuTableau = append(NomDuTableau[:IndexSup], NomDuTableau[IndexSup+1:]...)
```

EXEMPLE

```go
func main() {
    // Déclaration et initialisation du slice nommé "arr"
    arr := []string{"cyril", "kheir", "alan"}

    // Exemple ajouté une valeur a tableau dynamique
    arr = append(arr, "lucas")
    fmt.Println(arr)
    // arr : ["cyril","kheir","alan","lucas"]

    // Exemple ajouté plusieurs valeurs a un tableau dynamique

    arr = append(arr, "guillaume", "tristan", "cédric")
    fmt.Println(arr)
    // arr :
["cyril","kheir","alan","lucas","guillaume","tristan","cédric"]

    // Exemple suppression d'une valeur se trouvant a l'index 2
    arr = append(arr[:2], arr[3:]...)
    fmt.Println(arr)
    // arr : ["cyril","kheir","lucas","guillaume","tristan","cédric"]
}
```

PARCOURIR UN TABLEAU

En Go, les tableaux et la boucle for se combinent naturellement pour parcourir et manipuler des collections d'éléments. Il est possible de parcourir un tableau facilement en utilisant la boucle for avec itérateur ou bien de type range. Voici un exemple :

```go
func main() {
    // Liste des moyennes
    averageList := []float32{12.45, 10, 13.7, 2.15}
    // Boucle qui parse chaque élément du tableau : averageLis
    for i := 0; i < len(averageList); i++ {
        // Ajout d'un point à chaque moyenne
        averageList[i] += 1
    }
    fmt.Println(averageList)
    // Affichage : [13.45 11 14.7 3.15]
}
```

```go
func main() {
    arrNbr := []int{14, 78, 69, 52, 34}
    fmt.Println("==== CLASSEMENT ====")
    // Exemple utilisation boucle for/range
    for index, value := range arrNbr {
        fmt.Println("Place ", index, " Le joueur : ", value)
    }
}
```

*Dans une boucle for ... range, la variable de valeur (value) est une copie de l'élément. Si vous avez besoin de modifier l'élément original, il faudra passer par son indice (arrNbr[index]).

TABLEAUX MULTIDIMENSIONNELS

Un tableau multidimensionnel est un array dont chaque élément est lui-même un array, permettant de représenter des structures comme des grilles ou des matrices. Voici un exemple avec des tableaux à plusieurs dimensions :

*par convention, on limite les tableaux à trois dimensions pour garder le code lisible et simple.

```go
func main() {
    // Grille statique 3×3 de chaînes (valeur initiale : "")
    var gridGame [3][3]string

    // Matrice dynamique (slice de slices) avec valeurs prédéfinies
    matrix := [][]int{
        {10,  3,  8},
        { 5,  7,  9},
        { 6, 11,  4},
    }

    // Accès à une valeur de la matrice
    fmt.Println(matrix[0][1]) // Affiche : 3

    // Affectation dans la grille statique
    gridGame[2][1] = "X"

    // Parcours de la matrice et affichage
    for i := 0; i < len(matrix); i++ {
        for j := 0; j < len(matrix[i]); j++ {
            fmt.Printf("%d ", matrix[i][j])
        }
        fmt.Println()
    }
    // Résultat à l'écran :
    // 10 3 8
    // 5 7 9
    // 6 11 4
}
```

LES MAPS

LES MAPS

Une map en Go est une structure associative qui relie des clés uniques à des valeurs, formant ainsi une collection de paires clé/valeur. Pour déclarer une map, il suffit de préciser le type des clés et celui des valeurs. Voici l'anatomie d'une déclaration et/ou initialisation de maps :

Il est imporantat de noter qu'une map uniquement déclarer est égale à nil donc non utilisable

```
// Déclaration explicite
// nil, lecture returns zero-value, écriture panic
var nomMap map[typeKey]typeValue
// crée une map avec capacité initiale pour éviter une map nil
var nomMap map[typeKey]typeValue = make(map[typeKey]typeValue, size)

// Déclaration inféré
// littéral : map vide, prêt à l'emploi
nomMap := map[typeKey]typeValue{}
// équivalent par make, sans indiquer la capacité
nomMap := make(map[typeKey]typeValue)
```

make(map[K]V, cap) ne fixe pas la taille maximale, juste un hint de capacité pour optimiser les réallocations.

AJOUTER UNE PAIRE CLÉ/VALEUR

Pour ajouter une paire clé/valeur à une map en Go, il suffit simplement d'indiquer le nom de la map suivi entre crochets la nouvelle clé puis de lui associer la valeur. Cependant **attention si la clé est déjà existante la nouvelle valeur remplacera la valeur déjà présente pour cette clé.**

```go
func main() {
    // Déclaration d'une map avec la clé de type "string" et la valeur de type "int"
    var score map[string]int = map[string]int{"cyril": 152, "kheir": 214, "alan": 86}

    fmt.Println(score) // Afichage : map[alan:86 cyril:152 kheir:214]
    // Ajout d'une nouvelle clé "bob" avec comme valeur 345
    score["bob"] = 345

    fmt.Println(score) // Afichage : map[alan:86 bob:345 cyril:152 kheir:214]

    // Modification de la clé "bob"
    score["bob"] = 0
    fmt.Println(score) // Afichage : map[alan:86 bob:0 cyril:152 kheir:214]
}
```

RÉCUPÉRATION ET VÉRIFICATION DE L'EXISTENCE D'UNE CLÉ

Lors de la manipulation d’une map, il est fréquent de vouloir savoir si une clé y est présente avant d’utiliser sa valeur. **En Go, l’accès à une clé renvoie toujours deux résultats :**

- **La valeur associée à la clé** (ou la valeur zéro du type si la clé est absente)
- **Un booléen indiquant la présence** (true) ou l’absence (false) de la clé dans la map

Grâce à **cette double affectation, vous pouvez distinguer une véritable valeur par défaut d’une clé manquante.**

```
valeur, present := momMap[momKey]
if present {
    // la clé existe, 'valeur' contient la valeur réelle
} else {
    // la clé n'existe pas, 'valeur' vaut la valeur par défaut du type
}
```

EXEMPLE

```go
func main() {
    // Déclaration d'une map avec des clés de type "string" et des valeurs de type "int"
    var score map[string]int = map[string]int{"cyril": 152, "kheir": 214, "alan": 86}

    // Récupération de la valeur associée à la clé "alan"
    fmt.Printf("Score de Alan : %d\n", score["alan"]) // Affichage : Score de Alan : 86

    // Récupération d'une valeur d'une clé inexistante
    fmt.Printf("Score de Bob : %d\n", score["bob"]) // Affichage : Score de Bob : 0

    // Récupération de la valeur associée à une clé et vérification de l'existence de celle-ci
    valeur, existe := score["cyril"]
    if existe {
        fmt.Printf("Le score de cyril est de %d points\n", valeur)
    } else {
        fmt.Println("La clé \"cyril\" n'existe pas")
    }
}
```

*Par défaut, lorsqu'on accède à une clé on récupérer uniquement la valeur

SUPPRESSION D'UNE PAIRE CLÉ/VALEUR

En Go, on peut **retirer une entrée d'une map en utilisant la fonction intégrée delete.** Il suffit de passer en paramètres la map et la clé à supprimer :

- Si **la clé existe,** elle et sa valeur associée sont supprimées.
- Si **la clé n'existe pas,** l'appel ne fait rien (pas d'erreur).

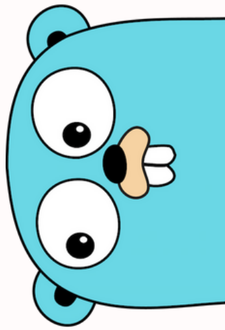
```go
func main() {
    // Déclaration d'une map avec la clé de type "string" et la valeur de type "int"
    var score map[string]int = map[string]int{"cyril": 152, "kheir": 214, "alan": 86}

    fmt.Println(score)
    // Affichage : map[alan:86 cyril:152 kheir:214]

    // Suppression de la clé "alan"
    delete(score, "alan")

    fmt.Println(score)
    // Affichage : map[cyril:152 kheir:214]
}
```

PARCOURIR UNE MAP

Pour itérer sur les paires clé/valeur d'une map en Go, on utilise la boucle for … range. À chaque itération, elle retourne la clé et la valeur associée. Attention, comme pour les tableaux la valeur est une copie !

```go
func main() {
    // Déclaration d'une map avec la clé de type "string" et la valeur de type "int"
    var score map[string]int = map[string]int{"cyril": 152, "kheir": 214, "alan": 86}

    fmt.Println("=== Tableau des scores ===")

    // Boucle sur la map "score" pour afficher toutes les paires clé-valeur
    for key, value := range score {
        fmt.Printf("\t %s : %d\n", key, value)
    }
}
```

*l'itération sur une map n'est pas ordonnée, les paires peuvent remonter dans un ordre différent à chaque exécution.



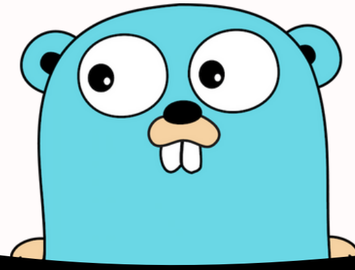
PACKAGES UTILES TABLEAUX & MAPS

Golang intègre nativement de nombreux packages standards qui simplifient la manipulation des tableaux et des maps :

- Package slices : fonctions utilitaires sur les slices (copie, recherche, comparaison, suppression).
- Package maps : fonctions pratiques pour maps (copier, comparer, extraire clés/valeurs).
- Package sort : tri de slices avec ordre naturel ou fonction personnalisée.

Attention les packages standards ne sont pas utilisables sur la plateforme Ytrack, mais pour vos projets personnels/professionnels leur usage est fortement recommandé

AIX ynov CAMPUS



À VOUS DE JOUER !