

Objectifs de l'atelier

Comprendre le fonctionnement des Tableaux, Tableaux dynamique et Dictionnaires

Être capable de décider quel type de collection utiliser

Être autonome et capable d'approfondir

Les types de collections

Les tableaux (arrays)

Les tableaux dynamique (slices)

Les dictionnaires (maps)

Les tableaux



Les tableaux : création

```
var t [n]T
```

Les tableaux : création

```
t := [n] T { }
```

Les tableaux : création

```
var t [1]int

a := new([2]string)

b := [2]bool{}

c := [3]int{0:1,1:2}

d := [4]int{1,2,3,4,5}

e := [...]int{1,2,3,4,5}

f := make([6]int,6,6)
```

Les tableaux : création

```
var t [1]int           // création d'une variable t : []  
  
a := new([2]string)   // récupère l'adresse du pointeur : &["" ""]  
  
b := [2]bool{}        // inférence de type littérale : [false false]  
  
c := [3]int{0:1,1:2}   // initialise les valeurs : [1 2 0]  
  
✗ d := [4]int{1,2,3,4,5} // ne compile pas !  
  
e := [...]int{1,2,3,4,5} // calcul la taille du tableau à la compilation  
  
✗ f := make([6]int,6,6) // ne compile pas !
```

Les tableaux : boucle avec incrément

```
a := [3]int{1, 2, 3}
for i := 0; i < len(a); i++ {
    print(a[i]) // output 123
}
```


Les tableaux : boucles avec range

```
a := [3]int{1, 2, 3}  
for _, v := range a {  
    print(v)  
}
```

boucle par copie de valeur

```
a := [3]int{1, 2, 3}  
for k, _ := range a {  
    print(a[k])  
}
```

boucle par clé

```
a := [3]int{1, 2, 3}  
for k := range a {  
    print(a[k])  
}
```

boucle par clé simple

Les tableaux : modification

```
1 s := [5]int{1,2,0,4,5}
2 fmt.Println(s) // output : [1 2 0 4 5]
3
4 s[2] = 3
5 fmt.Println(s) // output : [1 2 3 4 5]
6
```

Les tableaux : copie

```
1  a := [3]uint8{1, 2, 3}
2
3  b := a
4  b[0] = 0
5
6  fmt.Println(a)
7  fmt.Println(b)
8
```

Les tableaux : copie

```
1  a := [3]uint8{1, 2, 3}
2
3  b := a
4  b[0] = 0
5
6  fmt.Println(a) // output : [1, 2, 3]
7  fmt.Println(b) // output : [0, 2, 3]
8
```

Les tableaux : tip n°1

cap == len

cap == len

Les tableaux : tip n°1

```
1 a := [...]int{100:2}  
2  
3 fmt.Println(len(a))  
4  
5 fmt.Println(cap(a))  
6
```

cap == len

Les tableaux : tip n°1

```
1 a := [...]int{100:2}  
2  
3 fmt.Println(len(a)) // 101  
4  
5 fmt.Println(cap(a)) // 101  
6
```

Les tableaux : tip n°2

&share

Les tableaux : tip n°2

```
1 func main() {  
2     a := [3]int{1, 2, 3}  
3     fmt.Printf("%p\n", &a)  
4     PassArray(a)  
5 }  
6  
7  
8 func PassArray(a [3]int) {  
9     fmt.Printf("%p", &a)  
10 }
```

Les tableaux : tip n°2

```
1 func main() {  
2     a := [3]int{1, 2, 3}  
3     fmt.Printf("%p\n",&a) // 0x10434114  
4     PassArray(a)  
5 }  
6  
7  
8 func PassArray(a [3]int) {  
9     fmt.Printf("%p",&a) // 0x10434130  
10 }
```

Les tableaux : tip n°2

```
1 func main() {  
2     a := [3]int{1, 2, 3}  
3     fmt.Printf("%p\n",&a) // 0x10434114  
4     PassArray(&a)  
5 }  
6  
7  
8 func PassArray(a *[3]int) {  
9     fmt.Printf("%p",a) // 0x10434114  
10 }
```

Les tableaux : tip n°3

contiguous memory
allocation

contiguous memory
allocation

Les tableaux : tip n°3

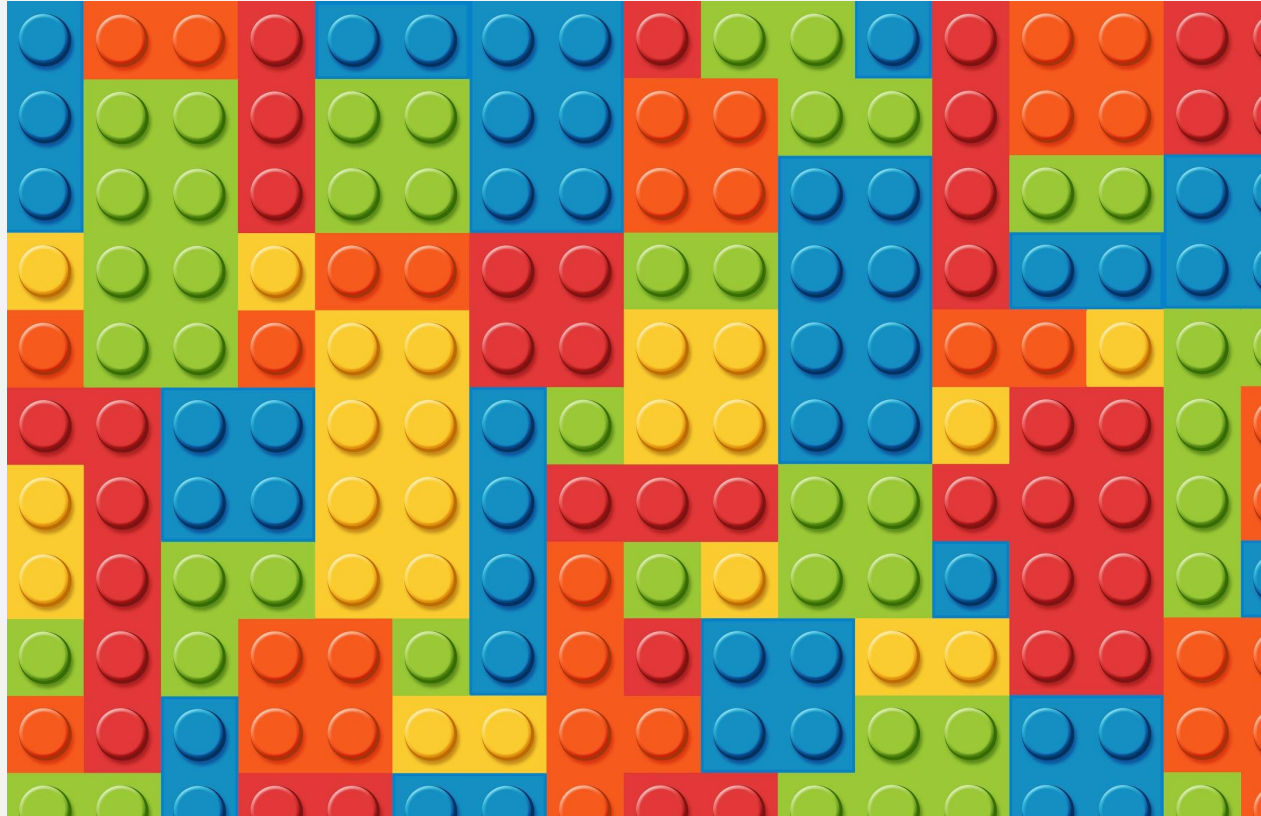
```
1 func main() {  
2     a := [3]uint8{1, 2, 3}  
3     for k := range a {  
4         fmt.Printf("%p\n", &a[k])  
5     }  
6 }  
7  
8 // output ?  
9  
10
```

Les tableaux : tip n°3

```
1 func main() {  
2     a := [3]uint8{1, 2, 3}  
3     for k := range a {  
4         fmt.Printf("%p\n", &a[k])  
5     }  
6 }  
7  
8 // 0x10434114  
9 // 0x10434115  
10 // 0x10434116
```

Les tableaux : tip n°3

opérateurs sur pointeurs ?
heuu oui :)



Les tableaux : tip n°4

const with size

const with size

Les tableaux : tip n°4

```
const NUMBER = 5
...
var a [3 * NUMBER]uint8
fmt.Println( len(a) )
```

+

&

-

|

*

^

/

<<

%

>>

&^

const with size

Les tableaux : tip n°4

```
type Letter int

const (
    A Letter = iota
    B
    C
    Max
)

var ListLetter = [Max]string{
    "a",
    "b",
    "c",
}
```

```
func (e Letter) String() string {
    return ListLetter[e]
}

func main() {
    fmt.Println(A, B, C)
}
// a b c
```

Les tableaux : **exercice**



1. Créer un tableau de 110 000 éléments de type string
2. Insérer la valeur “a” à la position 999
3. Afficher cette dernière valeur via le package fmt
4. Afficher la longueur du tableau

Vous pouvez utiliser la zone Play

Les tableaux : conclusion

Type primitif : utilisation basique

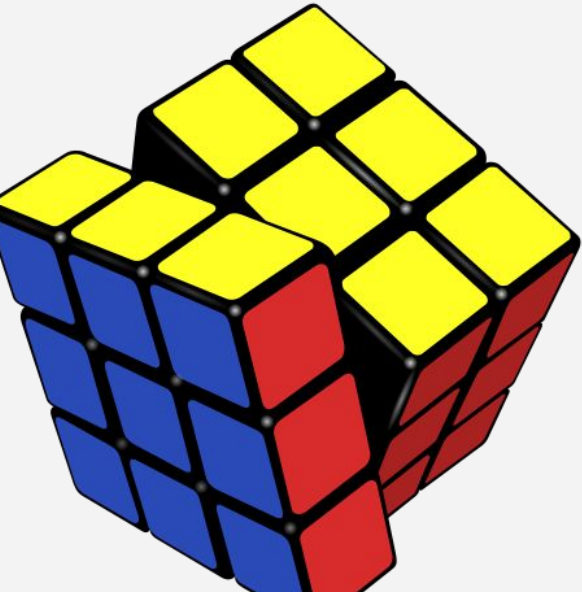
La taille est dans le type

Capacité == Longueur

Attention au partage par copie, si besoin d'utiliser des pointeurs !

Zone de mémoire contiguë

Les tableaux dynamiques



Les tableaux dynamiques : création

```
var t []T
```



Les tableaux dynamiques : création

```
t := []T{}
```

Les tableaux dynamiques : création

```
var t []int  
  
a := new([]int)  
  
b := []int{}  
  
c := []int{0:1,1:2}  
  
d := []int{1,2,3,4,5}  
  
e := []int{1,2,3,4,5}  
  
f := make([]int,6,6)
```


Les tableaux dynamiques : création

```
var t []int           // création d'une variable t : []int
a := new([]int)       // récupère l'adresse du pointeur : &[]
b := []int{}          // inférence de type littérale : []
c := []int{0:1,1:2}   // initialise les valeurs : [1 2]
 d := []int{1,2,3,4,5} // compile !
e := []int{1,2,3,4,5} // comme d
 f := make([]int,6,6) // compile !
```

Les tableaux dynamiques : création

Tableau

```
var t [1]int  
  
a := new([2]int)  
  
b := [2]int{}  
  
c := [3]int{0:1,1:2}  
  
✗ d := [4]int{1,2,3,4,5}  
  
e := [...]int{1,2,3,4,5}  
  
✗ f := make([6]int,6,6)
```

Tableau dynamique

```
var t []int  
  
a := new([]int)  
  
b := []int{}  
  
c := []int{0:1,1:2}  
  
d := []int{1,2,3,4,5}  
  
e := []int{1,2,3,4,5}  
  
f := make([]int,6,6)
```

Les tableaux dynamiques : boucles

```
a := []int{1, 2, 3}
for i := 0; i < len(a); i++ {
    print(a[i]) // output 123
}
```

Les tableaux dynamiques : boucles

```
a := []int{1, 2, 3}
for _, v := range a {
    print(v)
}
```

boucle par copie de valeur

```
a := []int{1, 2, 3}
for k, _ := range a {
    print(a[k])
}
```

boucle par clé

```
a := []int{1, 2, 3}
for k := range a {
    print(a[k])
}
```

boucle par clé simple

Les tableaux dynamiques : modification

```
1 s := []int{1,2,0,4,5}
2 fmt.Println(s) // output : [1 2 0 4 5]
3
4 s[2] = 3
5 fmt.Println(s) // output : [1 2 3 4 5]
6
```

Les tableaux dynamiques : ajouter

```
1 s := []int{1,2,3}
2 fmt.Println(s) // output : [1 2 3]
3
4 s = append(s,4,5)
5 fmt.Println(s) // output : [1 2 3 4 5]
6
```

Les tableaux dynamiques : ajouter

```
1 s := []int{1,2,3,4,5}
2 fmt.Println("len:", len(s), "cap:", cap(s))
3 // len: 5 cap: 5
4
5 s = append(s, 6, 7)
6 fmt.Println("len:", len(s), "cap:", cap(s))
7 // len: 7 cap: 12
```

Les tableaux dynamiques : tip n°1

len ≤ cap

Les tableaux dynamiques : tip n°1

```
1 s := make([]int,6,12)
2 fmt.Println("len:",len(s),"cap:",cap(s))
3 // output ?
4
5
6
```

Les tableaux dynamiques : tip n°1

```
1 s := make([]int,6,12)
2 fmt.Println("len:",len(s),"cap:",cap(s))
3 // len: 6 cap: 12
4
5
6
```

Les tableaux dynamiques : **internal**

```
a := []int{1, 2, 3, 4}
```

```
a = struct{ len: 4, cap: 4, tbl: *[4]int }
```

tbl =

0	1	2	3
1	2	3	4

len: 4 cap: 4

Les tableaux dynamiques : **internal**

```
a := make([]int, 4, 6)
```

```
a = struct{ len: 4, cap: 6, tbl: *[6]int }
```

tbl =

0	1	2	3		
0	0	0	0	0	0

len: 4 cap: 6

Les tableaux dynamiques : tip n°1

copy != [:]

Les tableaux dynamiques : **copy**

```
1  a := []int{1, 2, 3, 4, 5}
2  b := make([]int, len(a))
3  copy(b, a)
4  a[0] = 0
5
6  fmt.Println(a) // [0 2 3 4 5]
7  fmt.Println(b) // [1 2 3 4 5]
```

`copy(new, old)`

Les tableaux dynamiques : slicing

```
1  a := []int{1, 2, 3, 4, 5}
2  b := make([]int, 5)
3  b = a[:]
4  a[0] = 0
5
6  fmt.Println(a) // [0 2 3 4 5]
7  fmt.Println(b) // [0 2 3 4 5]
```


Les tableaux dynamiques : slicing

Un tableau dynamique contient un tableau

Réaliser une tranche d'un tableau dynamique revient à copier la valeur pointeur du tableau et changer les indexes de longueur et de capacité

Pour ne plus avoir la même référence de tableau, utiliser `copy` ou `append` si la capacité doit changer

Les tableaux dynamiques : slicing

```
1  a := []int{1, 2, 3, 4, 5}
2  var b []int
3  b = append(b, a[:]...)
4  a[0] = 0
5
6  fmt.Println(a) // [0 2 3 4 5]
7  fmt.Println(b) // [1 2 3 4 5]
```

Les tableaux dynamiques : slicing

```
1  a := []int{1, 2, 3, 4, 5}
2  b := a[:]
3  c := a[1:3]
4  d := a[1:3:3]
5
6  fmt.Println(a,"cap:",cap(a),"len:",len(a)) // [1 2 3 4 5] cap: 5 len:5
7  fmt.Println(b,"cap:",cap(b),"len:",len(b)) // [1 2 3 4 5] cap: 5 len:5
8  fmt.Println(c,"cap:",cap(c),"len:",len(c)) // [2 3] cap: 4 len:2
9  fmt.Println(d,"cap:",cap(d),"len:",len(d)) // [2 3] cap: 2 len:2
```

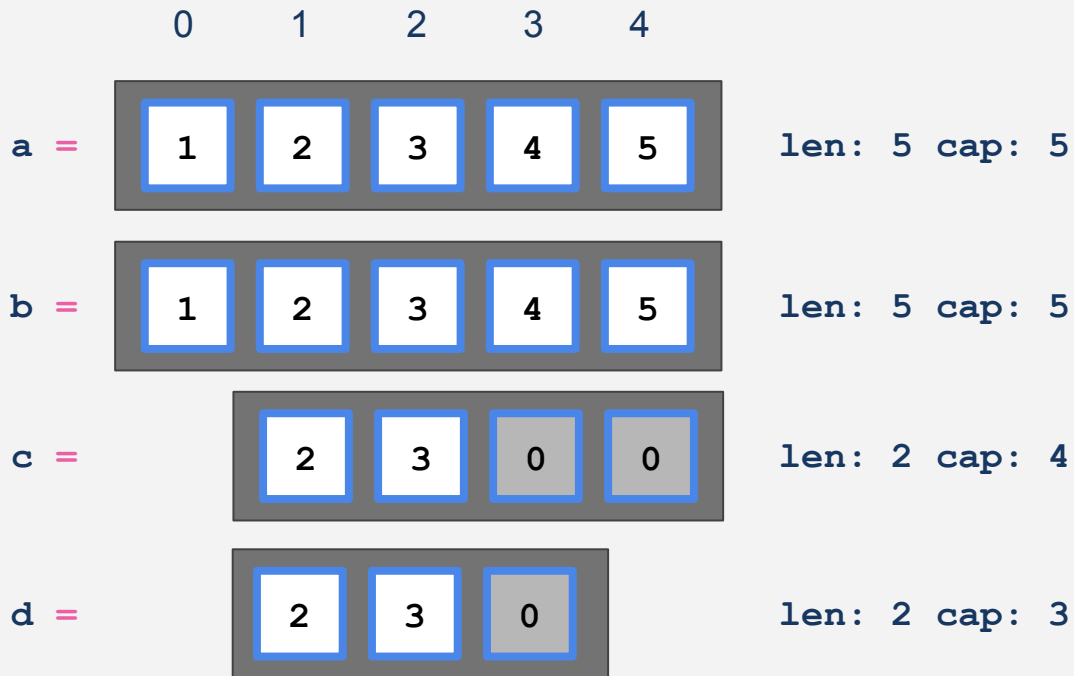
Les tableaux dynamiques : slicing

1	<code>a[:]</code>	<code>a[0:len(a):cap(a)-0]</code>
2		
3	<code>a[1:]</code>	<code>a[1:len(a):cap(a)-1]</code>
4		
5	<code>a[:2]</code>	<code>a[0:2:cap(a)-0]</code>
6		
7	<code>a[1:2]</code>	<code>a[1:2:cap(a)-1]</code>
8		
9	<code>a[1:3:3]</code>	<code>a[1:3:3]</code>

mySlice[start:end:cut]

```
a := []int{1, 2, 3, 4, 5}
b := a[:]
c := a[1:3]
d := a[1:3:4]
```

```
fmt.Println(a) // [1 2 3 4 5]
fmt.Println(b) // [1 2 3 4 5]
fmt.Println(c) // [2 3]
fmt.Println(d) // [2 3]
```



Les slices : tip n°2

just share !

Les tableaux dynamiques : partage

```
1 func main() {  
2     a := []int{1, 2, 3}  
3     fmt.Printf("%p\n", a)    // output ?  
4     PassSlice(a)  
5 }  
6  
7  
8 func PassSlice(a []int) {  
9     fmt.Printf("%p", a)      // output ?  
10 }  
11
```

Les tableaux dynamiques : partage

```
1 func main() {  
2     a := []int{1, 2, 3}  
3     fmt.Printf("%p\n", a)    // 0x10434114  
4     PassSlice(a)  
5 }  
6  
7  
8 func PassSlice(a []int) {  
9     fmt.Printf("%p", a)    // 0x10434114  
10 }  
11
```


Les tableaux dynamiques : vider

```
1  a := []int{1, 2, 3, 4, 5}  
2  a = ?  
3  
4  
5  
6  
7  
8  
9  
10  
11
```

Les tableaux dynamiques : vider

```
1  a := []int{1, 2, 3, 4, 5}
2  a = nil
3  fmt.Println(a, ":", "len :", len(a), "cap:", cap(a))
4  // output : [] : len : 0 cap: 0
5
6
7
8
9
10
11
```

Les tableaux dynamiques : vider

```
1  a := []int{1, 2, 3, 4, 5}
2  a = nil
3  fmt.Println(a, ":", "len :", len(a), "cap:", cap(a))
4  // output : [] : len : 0 cap: 0
5
6  b := []int{1, 2, 3, 4, 5}
7  b = append(a[0:0])
8  fmt.Println(b, ":", "len :", len(b), "cap:", cap(b))
9  // output : [] : len : 0 cap: 5
10
11
```

Les tableaux dynamiques : **exercice**

1. Créer un tableau dynamique “a” de type string avec les éléments suivants : “A”, ”B”, ”C”, ”D”
2. Afficher ses valeurs via le package fmt
3. Copier cette slice dans un nouveau tableau dynamique “b”
4. Modifier à la position 0 du tableau “a” la valeur en assignant “z”
5. Afficher avec fmt les valeurs de “a” puis “b”

Les tableaux dynamiques : exercice

1. Créer un tableau dynamique vide d'int
2. Créer une boucle incrémentale qui vas jusqu'à 2000
3. utiliser l'incrément pour l'ajouter au tableau dynamique
4. Afficher quand la capacité change dans la boucle

Les tableaux dynamiques : conclusion

Type évolué : utilisation avancée

La taille n'est pas dans le type

Longueur \leq Capacité

Partage du tableau sous-jacent par valeur pointeur

Tous comme le tableau zone de mémoire contiguë

բոխոյց - Ուրիցցուկայ ի գոտիս երկուսն
մերձ ի գաղաշարար: Ասի ևս Malléolo.
CHEVRE. ից. Իս. Capra. (լատ. Capra յն. ան.)
Այծ. առ. քելի. - Կենդ. էշ նոխարի կամ բազի
(bouc), մեծութեամբ հասարար ոչխարի, բայց դե-
բաշարած, ժուժկալ և կորովի բան զայն, և արածի
ի բորձուն և յառապարտան, և ունի մորուս ընդ
կզակաւ (Ձ. 240): Եղբերք նորա սպողորգը առ ար-



Ձ. 240

մին, և ի բորձունս խոնարհեալք յետս կոյն և
ձոխեալք երկուստեք. բայց գոն այժմ և առ նոց
եղջեր, որոց կամն առաւել նախապատիւ հաս-
նի զիկցուցանել զմանկունս: Մնանի ի հինգերորդ ամ.
սեան, և բերել սովորաբար երկուս ձագս միանգա-
մայն. կեայ մինչև ցերկոտասան և ցնդեկոտասան ամ.
ունի երկուս ստինս, և սուսն կարճ. կամն նորա
երկիցս առաւել է քան զոչխարն, և լինի անտի պա-
նիր ընտիր: Սակ նորա է երկայն և կակուզ, որ կոչի
ի մեզ Պրոն, զոր նիւթեն և յօրինեն ցփսիս հասա-
րակն և աղնիւս բառ տեսակցոյ այժից. Այծիք կաշ-
միր (Ձ. 241): Ուրալի, Գաղափոյի ձագբաւորք
են յոյժ փան նրաթմել մաղին: - Են այժից 32



— Առ Յոյն նախերեւ էր այժն Արամայիս ի
յիշատակ այժն Ամալթեայ, որ զիկցուցն զնա: Մե-
ծաբայ էր յայժ և առ Եշխարայիս:
Ասողար. կոչի Chèvre Այծ, այն փայլուն առաջն
առաջնոյ կարգի, որ է ի վերայ ձախուռուս կասու-
վար (Cocher) աստեղատան: — Ըստ առասպելն,
էստ այժն Ամալթեա (Amalthée), ստնառա Արա-
մազդայ:

Տնտես. գեղջ. Տախտակ յորոյ վերայ առնեն զգա-
նիր, և շիճուկն հոսել ի վայր ընդ կորի:

Մերեն. Մեքենայ բորձարացուցանելոյ ի վեր յայլ
և այլ յարկս տան զկարեւոր նիւթս շիճուկեան, որ
գիւղիք են քարանդ, դերանդ, և այլն: Զմերենայս
զայս կոչելք վերամարմ մեքենայ, որ է բաղադրու-
թիւն ուրանի, Տախտակի և երբեմն աստմեաւոր
անուոյ: Պարզն բաղկանայ յերկուց մեծամեծ գե-
րանոց զոգեկոյ ընդ միմեանս եռանկիւնաձև, և ուս
ի ներքուս կապելալ այլով կարճ փերանու. որ կոչի
Pied-de-chèvre Նից, ի վերին ծայրն հաստաւալ
է Տախտակի և յերեւեռնոցն տան (treuil,
որով որոյց քարն արագ և քիչ ջուրով շիճուկեան
որա, և առաւել ի բորձարանն (Ձ. 242):



Les dictionnaires

CHEVREAU. ար. Իս. Capretto, Cavretto. (յն.
չիբրետ, չիբրետ. լատ. Caper, Hoedus). Այլիկ. առ.
չիբիմ. - Կենդ. Արւս. ձագ այժի, որ եմէկ իցէ էզ
ասի Այլք կամ Այծ: Ոյ է հասարակ արուի և իրի
ձագուց այժեայ, որպէս Բուծ օգեայ: Մօրն նորա
յաղոց եղեալ և կաշեգործեալ, լինի քաշազկայ ի
յօրինել ձեռնոցս և կոչիկս կանացիս: - Ի հետեւ
զոհէին ամիկս աստուածոյն Պանի (Faune) և այլոց
զեղջիկան աստուածոց:

Ասողար. Chevreau. Այլիք. Են երբեմն աս-
տեղք կանափար (Cocher) աստեղատան, որ յօրինել
փոքրիկ եռանկիւն հասարարարուն, մերձ Այծն
(Chèvre).

CHEVREFEUILLE. ար. Իս. Caprifoglio. (իլա-
capra այծ, folium տերև, զի այժք սիրեն Տարակիւ
գրեւն նոսա). *Ամիսիւս. առ. քիւի կափարար, ձագ



Ձ. 243

առ. կէյիք սրորն. - Կենդ. Այլք էզ
և կարճ Եղբերք (Ձ. 244), Ամ



Ձ. 244

մանց ունին ի ծայր կնճթի խորե
եղեկեալ ի սեւաւ: Այծեանն փո-
ն զվիթ (daim), և մօրն նորա
բուն կոչի Broquart, և Էզն Che-
շէրա: Այծեանն է մի յընտիր
TILLOPE.

CHEVRON. ար. Իս. Travie
զուան), Գալիլ. առ. լաբր տիբ
բան փորք իբր 10 կամ 15 հար-
ձուրթեամբ, որով ծածկեն զձե-
րինեով գառ ի թափի, և ի վերայ
տակն, և գնորոց արկանեն կզ-
բար (ardoise).

Կենքար. Են երկաթին տափա-
նան ընդ միմեանս ի վերադոյն
քուսիք ծայրիւք, և միւս երկ-
ցուսիւք, բառաբնակի կիսով չ-
— Զատ կարեմք կոչել ի մեզ կ

Չիւն. Են աստեղնէ փայլաւ
լինին երբեմն նաեւ ասիկզն
կրեն զինաւորք ի վերայ ձախ
այլապիկն մանց զինաւորու

Ասողար. Օգեկեղեյթ Բըշ

Les dictionnaires : création

```
var m map[kT]vT
```


Les dictionnaires : création

```
t := make (map [kT] vT)
```

Les dictionnaires : création

```
var m map[int]int  
  
a := new(map[int]int)  
  
b := map[int]int{}  
  
c := make(map[int]int)  
  
d := make(map[int]int, 1000)  
  
e := make(map[int]int, 1000, 10000)  
  
f := map[int]int{0:1,1:2,2:3,3:4,4:5}
```

Les dictionnaires : création

```
var m map[int]int // affecte un type à une variable
❌ a := new(map[int]int) // renvoie *map[int]int
b := map[int]int{} // initialise la map
c := make(map[int]int) // initialise la map
d := make(map[int]int, 1000) // len(c) == 0
❌ e := make(map[int]int, 1000, 10000) // ne compile pas
f := map[int]int{0:1,1:2,2:3,3:4,4:5} // création littéral de map
```

Les dictionnaires : boucles

```
a := map[int]int{0:1, 1:2, 2:3}
for i := 0; i < len(a); i++ {
    print(a[i]) // output ?
}
```

Les dictionnaires : boucles

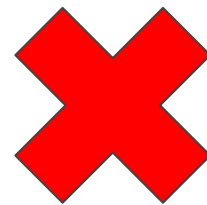
```
a := map[int]int{0:1, 1:2, 2:3}
for i := 0; i < len(a); i++ {
    print(a[i]) // 1 2 3
}
```

Les dictionnaires : boucles

```
a := map[int]int{3:1, 1:2, 2:3}
for i := 0; i < len(a); i++ {
    print(a[i]) // output ?
}
```

Les dictionnaires : boucles

```
a := map[int]int{3:1, 1:2, 2:3}
for i := 0; i < len(a); i++ {
    print(a[i]) // 0 2 3
}
```



Les dictionnaires : boucles

boucle par copie de valeur

```
a := map[int]int{1, 2, 3}
for _, v := range a {
    print(v)
}
// output ?
```

boucle par clé

```
a := map[int]int{1, 2, 3}
for k, _ := range a {
    print(a[k])
}
// output ?
```

boucle par clé simple

```
a := map[int]int{1, 2, 3}
for k := range a {
    print(a[k])
}
// output ?
```


Les dictionnaires : boucles

boucle par copie de valeur

```
a := map[int]int{1, 2, 3}
for _, v := range a {
    print(v)
}
// output :
123 ou 132 ou 213 ou 231
ou 312 ou enfin 321
```

boucle par clé

```
a := map[int]int{1, 2, 3}
for k, _ := range a {
    print(a[k])
}
// output :
123 ou 132 ou 213 ou 231
ou 312 ou enfin 321
```

boucle par clé simple

```
a := map[int]int{1, 2, 3}
for k := range a {
    print(a[k])
}
// output :
123 ou 132 ou 213 ou 231
ou 312 ou enfin 321
```

Les dictionnaires : ordre

```
1  import "sort"
2  ...
3  var m map[int]int{0: 1, 1: 2, 2: 3, 3: 4, 4: 5}
4  var keys []int
5  for k := range m {
6      keys = append(keys, k)
7  }
8  sort.Ints(keys)
9  for _, vk := range keys {
10     fmt.Println("Key:", vk, "Value:", m[vk])
11 }
```

Les dictionnaires : modification

```
1  a := map[int]int{1:1,2:2}
2  fmt.Println(a) // map[1:1 2:2]
3
4  a[1] = 3
5  fmt.Println(a) // map[1:3 2:2]
6
7
8
9
10
11
```

Les dictionnaires : suppression

```
1 m := map[int]int{0:1, 1:2, 2:3, 3:4, 4:5}
2
3 delete(m, 0)
4 fmt.Println(m, ":", "len /", len(m))
5
6 // output : map[3:4 4:5 1:2 2:3] / len : 4
7
8
9
10
11
```

Les dictionnaires : attacher

```
1  a := map[int]int{1:1, 2:2}
2  fmt.Println(a) // map[1:1 2:2]
3
4  b := a
5  fmt.Println(b) // map[1:1 2:2]
6
7  delete(a,1)
8  fmt.Println(b) // map[2:2]
9
10
11
```

Les dictionnaires : valeurs de retour

```
1  a := map[int]int{1:1, 2:2}
2  fmt.Println(a) // map[1:1 2:2]
3
4  delete(a,1)
5
6  fmt.Println(a[1]) // 0
7
8
9
10
11
```

Les dictionnaires : valeurs de retour

```
1  a := map[int]int{1:1, 2:2}
2  fmt.Println(a) // map[1:1 2:2]
3
4  delete(a,1)
5  v, ok := a[1]
6  fmt.Println(v, ok) // 0 false
7
8
9
10
11
```

Les dictionnaires : vider

```
1 m := map[int]int{0:1, 1:2, 2:3, 3:4, 4:5}  
2 m = ?  
3  
4  
5  
6
```


Les dictionnaires : vider

```
1 m := map[int]int{0:1, 1:2, 2:3, 3:4, 4:5}
2 m = nil
3 fmt.Println(m, ":", "len :", len(m))
4 // map[] : len : 0
5
6
```

Les dictionnaires : **vider**

```
1 m := map[int]int{0:1, 1:2, 2:3, 3:4, 4:5}
2 m = map[int]int{}
3 fmt.Println(m, ":", "len :", len(m))
4 // map[] : len : 0
5
6
```

Les dictionnaires : **exercice**

1. Créez et initialisez une map **m** avec en clé des strings et en valeur des ints
2. Créez une variable de type **rune** nommée **letter** et initialisez sa valeur à 'a'
3. Créez une boucle for avec **i** comme itérateur sur 26 éléments
4. Insérer dans **m** la valeur de **letter** après l'avoir casté en string comme clé et comme valeur insérer l'incrément **i** de la boucle for
5. Auto-incrémentez la valeur de **letter**
6. Après la boucle for afficher la valeur de la clé "w" dans la map **m** via la fonction `fmt.Println()`.

Les dictionnaires : conclusion

Type évolué : utilisation avancée parfait pour un gros volume de données

Tout comme le tableau dynamique la taille n'est pas dans le type

Il n'y a pas de capacité mais simplement une Longueur

Partage du tableau sous-jacent par valeur pointeur

Contrairement aux tableaux et tableaux dynamique la zone de mémoire n'est pas contiguë et il n'est pas sûr de récupérer un pointeur d'une des valeurs du tableau.

Liens utiles

Tableaux dynamique

<https://blog.golang.org/slices>

<https://blog.golang.org/go-maps-in-action>

<https://www.goinggo.net/2013/12/three-index-slices-in-go-12.html>

dictionnaires

<https://www.youtube.com/watch?v=TI7mi9QmLns>

<https://blog.golang.org/go-maps-in-action>

Les collection : création

Tableau

```
var t [1]int  
  
a := [2]int{}  
  
b := [2]int{1,2}  
  
c := [2]int{0:1,1:2}  
  
d := [...]int{1,2,3,4,5}  
  
e := [4*MAX]int{}
```

Tableau dynamique

```
var td []int  
  
a := []int{}  
  
b := []int{1,2}  
  
c := []int{0:1,1:2}  
  
d := make([]int,6)  
  
e := make([]int,6,30)
```

Dictionnaire

```
var m map[int]int  
  
a := map[int]int{}  
  
b := make(map[int]int)  
  
c := make(map[int]int, 10)  
  
d := map[int]int{0:1,1:2,2:3,3:4,4:5}
```

Les collection : ajout

Tableau

```
t := [3]string{"A","B","C"}  
  
var t2 [4]int  
  
for i := 0; i < len(t); i++ {  
    t2[i] = t[i]  
}  
t2[3] = "D"
```

Tableau dynamique

```
td := []string{"A","B","C"}  
  
td = append(td, "D")
```

Dictionnaire

```
m := map[int]string{0:"A",1:"B",2:"C"}  
  
m[3] = "D"
```

Les collection : modification

Tableau

```
t := [3]string{"A","B","C"}  
t[1] = "A"
```

Tableau dynamique

```
td := []string{"A","B","C"}  
td[1] = "A"
```

Dictionnaire

```
m := map[int]string{0:"A",1:"B",2:"C"}  
m[1] = "A"
```


Les collection : suppression

Tableau

```
t := [3]string{"A","B","C"}  
  
var t2 [2]string  
  
for i := 0; i < len(t); i++ {  
    if i != 1 {  
        t2[i] = t[i]  
    }  
}
```

Tableau dynamique

```
td := []string{"A","B","C"}  
  
td = append(td[:1], td[2:])
```

Dictionnaire

```
m := map[int]string{0:"A",1:"B",2:"C"}  
  
delete(m, 1)
```

Les collection : destruction

Tableau

```
var t [3]int

// seul moyen

t2 := &[3]int{1, 2, 3}
t2 = nil
```

Tableau dynamique

```
td := []string{"A","B","C"}

td = nil
```

Dictionnaire

```
m := map[int]string{0:"A",1:"B",2:"C"}

m = nil
```