Go : fondamentaux

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Plan

Go

Rappel

Go est un langage

- compilé
- fortement typé
- à typage statique

Go

Une variable ?

- Zone mémoire permettant de stocker une donnée
- Accessible à travers un identifiant (nom)
- Peut changer de valeur durant l'exécution du programme

Go

Caractéristiques d'une variable

Une variable est caractérisée par :

- son nom dans le programme
- son type
- sa valeur
- son adresse mémoire

Go

Caractéristiques d'une variable

Une variable est caractérisée par :

- son nom dans le programme
- son type
- sa valeur
- son adresse mémoire

Le type d'une variable est connu à la compilation.

Go

Conventions de nommage

- Choisir des noms courts et significatifs.
- Utiliser le camelCase.
- Éviter les caractères spéciaux et les accents.
- Éviter les mots-clés du langage.
- Utiliser des noms explicites :
  - `age`
  - `montantTotal`
  - `estMajeur`
- Les identifiants commençant par une majuscule sont exportés.

Go

Déclaration d'une variable : avec le mot-clé var

```
var age int
age = 40
```

Go

Déclaration d'une variable : avec le mot-clé var

```
var age int
age = 40
```

Ou directement

```
var age int = 40
```

Go

Inférence de type : Le compilateur peut déduire le type

```
var age = 40
var nom = "Achref"
var prix = 19.99
```

Go

Inférence de type : Le compilateur peut déduire le type

```
var age = 40
var nom = "Achref"
var prix = 19.99
```

Le type reste connu à la compilation

```
fmt.Printf("%T\n", age)
```

Go

Inférence de type : Le compilateur peut déduire le type

```
var age = 40
var nom = "Achref"
var prix = 19.99
```

Le type reste connu à la compilation

```
fmt.Printf("%T\n", age)
```

Résultat

```
int
```

Go

Déclaration courte à l'intérieur d'une fonction :

```
age := 40
nom := "Achref"
actif := true
```

Go

Déclaration courte à l'intérieur d'une fonction :

```
age := 40
nom := "Achref"
actif := true
```

Équivalent à :

```
var age int = 40
var nom string = "Achref"
var actif bool = true
```

Go

Déclaration multiple : il est possible de déclarer plusieurs variables.

```
var (
    nom string = "Achref"
    age int = 40
    actif bool = true
)
```

Go

Déclaration multiple : il est possible de déclarer plusieurs variables.

```
var (
    nom string = "Achref"
    age int = 40
    actif bool = true
)
```


Ou

```
nom, age := "Achref", 40
```

Go

Types primitifs

Type	Exemple
bool	true
int	42
int64	1000000
float32	3.14
float64	3.14
string	"bonjour"
rune	'A'
byte	255

Go

Afficher le type d'une variable

```
package main

import "fmt"

func main() {

    age := 40
    prix := 15.5
    nom := "Achref"

    fmt.Printf("%T\n", age)
    fmt.Printf("%T\n", prix)
    fmt.Printf("%T\n", nom)
}
```

Go

Valeur zéro (Zero Value) : toute variable déclarée possède une valeur par défaut.

Type	Valeur zéro
int	0
float64	0.0
bool	false
string	""
pointeur	nil
slice	nil
map	nil

Go

Exemple de valeur zéro

```
package main

import "fmt"

func main() {

    var age int
    var nom string
    var actif bool

    fmt.Println(age)
    fmt.Println(nom)
    fmt.Println(actif)
}
```

Go

Typage statique

- En Go, une variable conserve son type.

  Ceci est correct :

```
age := 40
```

- Ceci provoque une erreur :

```
age = "bonjour"
```

- Le compilateur détecte l'erreur avant l'exécution.

Go

Exercice

Écrire un programme Go qui :

1. Déclare une variable nom contenant votre prénom
2. Déclare une variable age contenant votre âge
3. Déclare une variable actif contenant true
4. Affiche les trois variables

Go

Correction

```
package main

import "fmt"

func main() {

    nom := "Achref"
    age := 40
    actif := true

    fmt.Println(nom)
    fmt.Println(age)
    fmt.Println(actif)
}
```

Go

Opérations et méthodes sur les variables numériques

- + : addition
- - : soustraction
- * : multiplication
- / : division
- % : reste de la division

Go ne possède pas d'opérateur de puissance.

Go

Opérations et méthodes sur les variables Variables numériques

```
fmt.Println(5 + 2)
fmt.Println(5 - 2)
fmt.Println(5 * 2)
fmt.Println(5 / 2)
fmt.Println(5 % 2)
```

Go

Opérations et méthodes sur les variables Variables numériques

```
fmt.Println(5 + 2)
fmt.Println(5 - 2)
fmt.Println(5 * 2)
fmt.Println(5 / 2)
fmt.Println(5 % 2)
```


Résultat

```
7 3 10 2 1
```

Go

Opérations et méthodes sur les variables Variables numériques

```
i += 2
i -= 2
i *= 2
i /= 2
i %= 2
```

Go

Exercice

Écrire un programme **Go** qui permet de permuter le contenu de deux variables sans utiliser de variable temporaire.

Go

Une solution

```
a := 2
b := 3

a, b = b, a

fmt.Println(a, b)
```

Go

Une solution

```
a := 2
b := 3

a, b = b, a

fmt.Println(a, b)
```


Résultat

```
3 2
```

Go

Considérons les deux chaînes suivantes

```
str1 := "bon"
str2 := "jour"
```

Go

Considérons les deux chaînes suivantes

```
str1 := "bon"
str2 := "jour"
```

Utilisons l'opérateur + pour concaténer deux chaînes

```
str3 := str1 + str2
fmt.Println(str3)
```

Go

Considérons les deux chaînes suivantes

```
str1 := "bon"
str2 := "jour"
```

Utilisons l'opérateur + pour concaténer deux chaînes

```
str3 := str1 + str2
fmt.Println(str3)
```


Résultat

```
bonjour
```

Go

Pour répéter une chaîne plusieurs fois

```
import "strings"

fmt.Println(
    strings.Repeat("bonjour", 4),
)
```

Go

Pour répéter une chaîne plusieurs fois

```
import "strings"

fmt.Println(
    strings.Repeat("bonjour", 4),
)
```


Résultat

```
bonjourbonjourbonjourbonjour
```

Go

Pour récupérer la taille d'une chaîne

```
str := "bonjour"

fmt.Println(len(str))
```

Go

Pour récupérer la taille d'une chaîne

```
str := "bonjour"

fmt.Println(len(str))
```


Résultat

```
7
```

Go

Pour remplacer une partie d'une chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.ReplaceAll(
        str,
        "jour",
        "soir",
    ),
)
```

Go

Pour remplacer une partie d'une chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.ReplaceAll(
        str,
        "jour",
        "soir",
    ),
)
```

Résultat

```
bonsoir
```

Go

Pour accéder à un caractère

```
str := "bonjour"

fmt.Printf("%c\n", str[2])
```

Go

Pour accéder à un caractère

```
str := "bonjour"

fmt.Printf("%c\n", str[2])
```


Résultat

```
n
```

Go

Remarque

- Une chaîne de caractères est une séquence d'octets.
- Le premier caractère possède l'indice 0.

Go

Remarque

- Une chaîne de caractères est une séquence d'octets.
- Le premier caractère possède l'indice 0.

b	o	n	j	o	u	r
0	1	2	3	4	5	6

Go

Pour extraire une partie d'une chaîne

```
str := "bonjour"

fmt.Println(str[2:4])
```

Go

Pour extraire une partie d'une chaîne

```
str := "bonjour"

fmt.Println(str[2:4])
```


Résultat

```
nj
```

Go

Pour extraire les caractères à partir d'une position

```
str := "bonjour"

fmt.Println(str[2:])
```

Go

Pour extraire les caractères à partir d'une position

```
str := "bonjour"

fmt.Println(str[2:])
```


Résultat

```
njour
```

Go

Pour extraire les caractères du début jusqu'à une position

```
str := "bonjour"

fmt.Println(str[:4])
```

Go

Pour extraire les caractères du début jusqu'à une position

```
str := "bonjour"

fmt.Println(str[:4])
```


Résultat

```
bonj
```

Go

Quelques fonctions du package strings

- `strings.Contains()`
- `strings.HasPrefix()`
- `strings.HasSuffix()`
- `strings.Index()`
- `strings.Count()`
- `strings.ReplaceAll()`
- `strings.Split()`

Go

Pour rechercher une sous-chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.Contains(
        str,
        "jour",
    ),
)
```

Go

Pour rechercher une sous-chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.Contains(
        str,
        "jour",
    ),
)
```

Résultat

```
true
```

Go

Pour vérifier le début d'une chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.HasPrefix(
        str,
        "bon",
    ),
)
```

Go

Pour vérifier le début d'une chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.HasPrefix(
        str,
        "bon",
    ),
)
```

Résultat

```
true
```

Go

Pour vérifier la fin d'une chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.HasSuffix(
        str,
        "jour",
    ),
)
```

Go

Pour vérifier la fin d'une chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.HasSuffix(
        str,
        "jour",
    ),
)
```

Résultat

```
true
```

Go

Pour récupérer la position d'une sous-chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.Index(
        str,
        "jour",
    ),
)
```

Go

Pour récupérer la position d'une sous-chaîne

```
import "strings"

str := "bonjour"

fmt.Println(
    strings.Index(
        str,
        "jour",
    ),
)
```

Résultat

```
3
```

Go

Pour découper une chaîne selon un séparateur

```
import "strings"

str := "a;b;c;d"

fmt.Println(
    strings.Split(
        str,
        ";",
    ),
)
```

Go

Pour découper une chaîne selon un séparateur

```
import "strings"

str := "a;b;c;d"

fmt.Println(
    strings.Split(
        str,
        ";",
    ),
)
```

Résultat

```
[a b c d]
```

Go

Pour convertir en majuscule

```
import "strings"

fmt.Println(
    strings.ToUpper(
        "bonjour",
    ),
)
```

Go

Pour convertir en majuscule

```
import "strings"

fmt.Println(
    strings.ToUpper(
        "bonjour",
    ),
)
```


Résultat

```
BONJOUR
```

Go

Pour convertir en minuscule

```
import "strings"

fmt.Println(
    strings.ToLower(
        "BONJOUR",
    ),
)
```

Go

Pour convertir en minuscule

```
import "strings"

fmt.Println(
    strings.ToLower(
        "BONJOUR",
    ),
)
```


Résultat

```
bonjour
```

Go

Exercice

Étant donnée la chaîne suivante

```
maChaine := "abcde"
```

Écrire un programme Go qui remplace le caractère `c` par `C`.

Go

Correction

```
import "strings"

maChaine := "abcde"

maChaine =
    strings.ReplaceAll(
        maChaine,
        "c",
        "C",
    )

fmt.Println(maChaine)
```

Go

Correction

```
import "strings"

maChaine := "abcde"

maChaine =
    strings.ReplaceAll(
        maChaine,
        "c",
        "C",
    )

fmt.Println(maChaine)
```

Résultat

```
abCde
```

Go

Rappel

- Go est un langage fortement typé.
- Certaines opérations sont interdites.
- Par exemple :

```
2 + "3"
```

- déclenche une erreur de compilation.

Go

Pour convertir un entier en chaîne

```
import "strconv"

x := 2

str :=
    strconv.Itoa(x)

fmt.Println(str)
```

Go

Pour convertir un entier en chaîne

```
import "strconv"

x := 2

str :=
    strconv.Itoa(x)

fmt.Println(str)
```

Résultat

```
2
```

Go

Pour convertir une chaîne en entier

```
import "strconv"

x, _ :=
    strconv.Atoi("25")

fmt.Println(x)
```

Go

Pour convertir une chaîne en entier

```
import "strconv"

x, _ :=
    strconv.Atoi("25")

fmt.Println(x)
```


Résultat

```
25
```

Go

Pour convertir une chaîne en réel

```
import "strconv"

x, _ :=
    strconv.ParseFloat(
        "12.5",
        64,
    )

fmt.Println(x)
```

Go

Pour convertir une chaîne en réel

```
import "strconv"

x, _ :=
    strconv.ParseFloat(
        "12.5",
        64,
    )

fmt.Println(x)
```


Résultat

```
12.5
```

Go

Pour convertir un booléen en chaîne

```
import "strconv"

fmt.Println(
    strconv.FormatBool(
        true,
    ),
)
```

Go

Pour convertir un booléen en chaîne

```
import "strconv"

fmt.Println(
    strconv.FormatBool(
        true,
    ),
)
```


Résultat

```
true
```

Go

Quelques fonctions de conversion

- `strconv.Atoi()`
- `strconv.Itoa()`
- `strconv.ParseFloat()`
- `strconv.FormatFloat()`
- `strconv.ParseBool()`
- `strconv.FormatBool()`

Go

Pour récupérer le code Unicode d'un caractère

```
fmt.Println(
    int('b'),
)
```

Go

Pour récupérer le code Unicode d'un caractère

```
fmt.Println(
    int('b'),
)
```


Résultat

```
98
```

Go

Pour récupérer le caractère associé à un code Unicode

```
fmt.Printf(
    "%c\n",
    98,
)
```

Go

Pour récupérer le caractère associé à un code Unicode

```
fmt.Printf(
    "%c\n",
    98,
)
```


Résultat

```
b
```

Go

Exemple avec rune

```
var c rune = 'é'

fmt.Println(c)

fmt.Printf(
    "%c\n",
    c,
)
```

Go

Exemple avec rune

```
var c rune = 'é'

fmt.Println(c)

fmt.Printf(
    "%c\n",
    c,
)
```


Résultat

```
233 é
```

Go

Pour lire une saisie clavier

- `fmt.Scan()`
- `fmt.Scanln()`
- `fmt.Scanf()`
- `bufio.NewReader()`

Go

Pour lire une chaîne de caractères

```
var nom string

fmt.Scan(&nom)
```

Go

Exemple

```
var nom string

fmt.Print(
    "Quel est votre nom ? "
)

fmt.Scan(&nom)

fmt.Println(nom)
```

Go

Pour lire un entier

```
var age int

fmt.Scan(&age)
```

Go

Exemple

```
var age int

fmt.Print(
    "Age : "
)

fmt.Scan(&age)

fmt.Println(age)
```

Go

Remarque

- L'opérateur `&` permet de transmettre l'adresse mémoire d'une variable.
- `fmt.Scan()` stocke directement la valeur saisie dans cette variable.

Go

Pour lire une ligne complète

```
import (
    "bufio"
    "os"
)

reader :=
    bufio.NewReader(
        os.Stdin,
    )
```

Go

Exemple

```
reader :=
    bufio.NewReader(
        os.Stdin,
    )

nom, _ :=
    reader.ReadString(
        '\n',
    )

fmt.Println(nom)
```

Go

Pourquoi utiliser `ReadString()` ?

- `fmt.Scan()` s'arrête au premier espace.
- `ReadString()` lit toute la ligne.
- Pratique pour les noms composés.
- Pratique pour les phrases.

Go

Exercice

Écrire un programme Go qui :

1. demande le prénom
2. demande l'âge
3. affiche `Bonjour Achref, vous avez 40 ans`

Go

Correction

```
var nom string
var age int

fmt.Print("Nom : ")
fmt.Scan(&nom)

fmt.Print("Age : ")
fmt.Scan(&age)

fmt.Println(
    "Bonjour",
    nom,
    ", vous avez",
    age,
    "ans",
)
```

Go

Un projet Go contient généralement

- un ou plusieurs fichiers `.go`
- un fichier `go.mod`
- plusieurs `packages`

Go

Extensions courantes

- `.go` : fichier source Go
- `go.mod` : définition du module
- `go.sum` : dépendances téléchargées

Go

Structure minimale

```
package main

import "fmt"

func main() {

    fmt.Println(
        "Hello World",
    )

}
```

Go

Pour créer un projet Go sous Visual Studio Code

1. File > Open Folder...
2. Créer un dossier
3. Ouvrir ce dossier
4. Créer un fichier main.go

Go

Initialiser un module Go

```
go mod init cours_go
```

Go

Initialiser un module Go

```
go mod init cours_go
```

Résultat

```
go: creating new go.mod
```

Go

Contenu du fichier main.go

```
package main

import "fmt"

func main() {

    fmt.Println(
        "Hello World",
    )

}
```

Go

Pour exécuter le programme

```
go run main.go
```

Go

Pour exécuter le programme

```
go run main.go
```

Ou

```
go run .
```

Go

Pour compiler le programme

```
go build
```

Go

Pour compiler le programme

```
go build
```

Résultat

```
cours_go.exe
```

Go

Pour compiler le programme

```
go build
```


Résultat

```
cours_go.exe
```

Sous Linux

```
cours_go
```

Go

Pourquoi compiler ?

- Détection des erreurs avant exécution.
- Exécutable autonome.
- Meilleures performances.
- Déploiement simplifié.

Go

Go est un langage compilé

Code source Go

⇓

Compilation

⇓

Exécutable

⇓

Exécution

Go

Exercice

Écrire un programme Go qui :

1. demande deux nombres
2. affiche : `La somme de 2 et 3 est : 5`

Go

Une première solution

```
var x int
var y int

fmt.Scan(&x)
fmt.Scan(&y)

fmt.Println(
    "La somme de",
    x,
    "et",
    y,
    "est :",
    x+y,
)
```

Go

Remarque

La fonction

```
fmt.Println()
```

ajoute automatiquement un espace entre les paramètres.

Go

Une autre solution consiste à utiliser `fmt.Printf()`

```
fmt.Printf(
    "La somme de %d et %d est : %d",
    x,
    y,
    x+y,
)
```

Go

Quelques spécificateurs

%d	entier
%f	réel
%s	chaîne
%t	booléen
%c	caractère
%T	type
%v	valeur

Go

Exemple

```
nom := "Achref"
age := 40

fmt.Printf(
    "%s a %d ans\n",
    nom,
    age,
)
```

Go

Exemple

```
nom := "Achref"
age := 40

fmt.Printf(
    "%s a %d ans\n",
    nom,
    age,
)
```

Résultat

```
Achref a 40 ans
```

Go

Afficher le type

```
x := 12.5

fmt.Printf(
    "%T\n",
    x,
)
```

Go

Afficher le type

```
x := 12.5

fmt.Printf(
    "%T\n",
    x,
)
```


Résultat

```
float64
```

Go

Afficher un réel avec deux décimales

```
prix := 12.4567

fmt.Printf(
    "%.2f\n",
    prix,
)
```

Go

Afficher un réel avec deux décimales

```
prix := 12.4567

fmt.Printf(
    "%.2f\n",
    prix,
)
```


Résultat

```
12.46
```

Go

Afficher plusieurs informations

```
nom := "Achref"
age := 40
actif := true

fmt.Printf(
    "%s %d %t\n",
    nom,
    age,
    actif,
)
```

Go

Afficher plusieurs informations

```
nom := "Achref"
age := 40
actif := true

fmt.Printf(
    "%s %d %t\n",
    nom,
    age,
    actif,
)
```

Résultat

```
Achref 40 true
```

Go

Exercice

Écrire un programme qui :

1. demande un prénom
2. demande un âge
3. affiche le résultat avec fmt.Printf()

Go

Correction

```
var nom string
var age int

fmt.Scan(&nom)
fmt.Scan(&age)

fmt.Printf(
    "%s a %d ans\n",
    nom,
    age,
)
```

Go

Résumé

- Print : affichage simple.
- Println : affichage avec retour à la ligne.
- Printf : affichage formaté.
- Les spécificateurs commencent par %.

Go

Les commentaires permettent

- d'expliquer le code
- d'améliorer sa lisibilité
- de générer la documentation.

Go

Commentaire sur une ligne

```
// Ceci est un commentaire
fmt.Println("Bonjour")
```

Go

Commentaire sur plusieurs lignes

```
/*
Commentaire
sur plusieurs
lignes
*/
```

Go

Documentation Go

- Les commentaires placés avant :
  - une fonction
  - une structure
  - une interface
  - un package
- peuvent être utilisés par GoDoc.

Go

Exemple

```
// Addition retourne
// la somme de deux nombres.
func Addition(
    a int,
    b int,
) int {

    return a + b
}
```

Go

Une structure conditionnelle permet

- d'exécuter un bloc
- uniquement si une condition est vraie

Go

Syntaxe

```
if condition {

}
```

Go

Exemple

```
age := 20

if age >= 18 {

    fmt.Println(
        "Majeur",
    )

}
```

Go

Remarque

- Les parenthèses sont facultatives.
- Les accolades sont obligatoires.

Go

Syntaxe

```
if condition {

}
else {

}
```

Go

Exemple

```
age := 16

if age >= 18 {

    fmt.Println(
        "Majeur",
    )

} else {

    fmt.Println(
        "Mineur",
    )

}
```

Go

Syntaxe

```
if condition1 {

}
else if condition2 {

}
else {

}
```

Go

Exemple

```
note := 15

if note >= 16 {

    fmt.Println("Très bien")

} else if note >= 14 {

    fmt.Println("Bien")

} else if note >= 10 {

    fmt.Println("Admis")

} else {

    fmt.Println("Ajourné")

}
```

Go

Remarque

- Les conditions sont évaluées dans l'ordre.
- Dès qu'une condition est vraie, les suivantes ne sont plus évaluées.

Go

Une variable peut être déclarée dans le if

```
if age := 20;
    age >= 18 {

    fmt.Println(
        "Majeur",
    )

}
```

Go

Une variable peut être déclarée dans le if

```
if age := 20;
    age >= 18 {

    fmt.Println(
        "Majeur",
    )

}
```


Remarque

La variable déclarée dans le if `age` n'est accessible que dans le bloc conditionnel.

Go

Exercice

Écrire un programme qui :

1. demande une note
2. affiche :
   - Admis si note >= 10
   - Ajourné sinon

Go

Correction

```
var note float64

fmt.Scan(&note)

if note >= 10 {

    fmt.Println(
        "Admis",
    )

} else {

    fmt.Println(
        "Ajourné",
    )

}
```

Go

switch

Le `switch` permet de comparer une valeur à plusieurs cas.

Go

Syntaxe

```
switch variable {

case valeur1:

case valeur2:

default:

}
```

Go

Exemple

```
jour := 2

switch jour {

case 1:
    fmt.Println("Lundi")

case 2:
    fmt.Println("Mardi")

case 3:
    fmt.Println("Mercredi")

default:
    fmt.Println("Inconnu")

}
```

Go

Remarque

Contrairement au langage **C**, `break` est implicite.

Go

Plusieurs valeurs peuvent être regroupées

```
note := 15

switch note {

case 10,11,12:

    fmt.Println("Passable")

case 13,14:

    fmt.Println("Assez bien")

case 15,16:

    fmt.Println("Bien")

}
```

Go

Switch sans variable

```
age := 20

switch {

case age < 18:
    fmt.Println("Mineur")

case age >= 18:
    fmt.Println("Majeur")

}
```

Go

`fallthrough`

Le `fallthrough` force l'exécution du cas suivant.

Go

Exemple

```
x := 1

switch x {

case 1:
    fmt.Println("A")
    fallthrough

case 2:
    fmt.Println("B")

}
```

Go

Exemple

```
x := 1

switch x {

case 1:
    fmt.Println("A")
    fallthrough

case 2:
    fmt.Println("B")

}
```

Résultat

```
A
B
```

Go

Exercice

Écrire un programme qui :

1. demande un nombre entre 1 et 7
2. affiche le jour correspondant.

Go

Correction

```
switch jour {

case 1:
    fmt.Println("Lundi")

case 2:
    fmt.Println("Mardi")

case 3:
    fmt.Println("Mercredi")

case 4:
    fmt.Println("Jeudi")

case 5:
    fmt.Println("Vendredi")

case 6:
    fmt.Println("Samedi")

case 7:
    fmt.Println("Dimanche")

default:
    fmt.Println("Invalide")

}
```

Go

Go ne possède pas :

- de boucle `while`
- de boucle `do ... while`.

Go

Go ne possède pas :

- de boucle `while`
- de boucle `do ... while`.

Conclusion

Toutes les itérations sont réalisées avec `for`

Go

Syntaxe générale

```
for condition {

}
```

Go

Équivalent d'un while

```
i := 1

for i <= 5 {

    fmt.Println(i)

    i++

}
```

Go

Équivalent d'un while

```
i := 1

for i <= 5 {

    fmt.Println(i)

    i++

}
```


Résultat

```
1
2
3
4
5
```

Go

Syntaxe classique

```
for initialisation ;
    condition ;
    incrément {

}
```

Go

Exemple

```
for i := 1;
    i <= 5;
    i++ {

    fmt.Println(i)

}
```

Go

Exemple

```
for i := 1;
    i <= 5;
    i++ {

    fmt.Println(i)

}
```


Résultat

```
1
2
3
4
5
```

Go

Remarques

- L'opérateur `i++` incrémente la variable de 1.
- L'opérateur `i--` décrémente la variable de 1.

Go

Compte à rebours

```
for i := 5;
    i >= 1;
    i-- {

    fmt.Println(i)

}
```

Go

`for range` permet de parcourir

- une chaîne
- un tableau
- une slice
- une map

Exemple avec une chaîne

```
for indice,
    caractere :=
        range "bonjour" {

    fmt.Println(
        indice,
        string(caractere),
    )

}
```

Exemple avec une chaîne

```
for indice,
    caractere :=
        range "bonjour" {

    fmt.Println(
        indice,
        string(caractere),
    )

}
```

Résultat

```
0 b
1 o
2 n
3 j
4 o
5 u
6 r
```

Exemple avec un tableau

```
notes :=
    []int{
        12,
        15,
        18,
    }

for i,v :=
    range notes {

    fmt.Println(i,v)

}
```

Exemple avec un tableau

```
notes :=
    []int{
        12,
        15,
        18,
    }

for i,v :=
    range notes {

    fmt.Println(i,v)

}
```


Résultat

```
0 12
1 15
2 18
```

Go

`break`

Le `break` permet d'interrompre une boucle.

Go

Exemple

```
for i := 1;
    i <= 10;
    i++ {
    if i == 5 {
        break
    }
    fmt.Println(i)
}
```

Go

Exemple

```
for i := 1;
    i <= 10;
    i++ {
    if i == 5 {
        break
    }
    fmt.Println(i)
}
```


Résultat

```
1
2
3
4
```

Go

continue

Le `continue` permet de passer directement à l'itération suivante.

Go

Exemple

```
for i := 1;
    i <= 5;
    i++ {
    if i == 3 {
        continue
    }
    fmt.Println(i)
}
```

Go

Exemple

```
for i := 1;
    i <= 5;
    i++ {
    if i == 3 {
        continue
    }
    fmt.Println(i)
}
```


Résultat

```
1 2 4 5
```

Go

break

Le caractère _ permet d'ignorer une valeur.

Go

Exemple

```
notes :=
    []int{
        12,
        15,
        18,
    }

for _, note :=
    range notes {

    fmt.Println(note)

}
```

Go

Exercice

Écrire un programme qui affiche :

```
1
2
3
4
5
6
7
8
9
10
```

à l'aide d'une boucle.

Go

Correction

```
for i := 1;
    i <= 10;
    i++ {

    fmt.Println(i)

}
```

Go

Exercice

Écrire un programme qui :

1. demande un nombre
2. affiche sa table de multiplication.

Go

Correction

```
var n int

fmt.Scan(&n)

for i := 1;
    i <= 10;
    i++ {

    fmt.Printf(
        "%d x %d = %d\n",
        n,
        i,
        n*i,
    )

}
```

Go

La portée d'une variable représente

- l'endroit où elle est visible
- l'endroit où elle peut être utilisée.

Go

Go distingue principalement

- les variables locales
- les variables de package.

Go

Variable locale

```
func main() {

    age := 40

    fmt.Println(age)

}
```

Go

Remarque

Une variable locale n'est accessible que :

- dans le bloc où elle est déclarée
- dans ses sous-blocs.

Go

Exemple

```
func main() {

    if true {
        x := 10
        fmt.Println(x)
    }
    fmt.Println(x)

}
```

Go

Exemple

```
func main() {

    if true {
        x := 10
        fmt.Println(x)
    }
    fmt.Println(x)

}
```


Résultat

Erreur de compilation : la variable `x` n'existe plus en dehors du bloc.

Go

Variable de package

```
package main

import "fmt"

var message =
    "Bonjour"

func main() {

    fmt.Println(message)

}
```

Go

Les variables de package

- sont déclarées en dehors des fonctions
- sont accessibles dans tout le package
- doivent être utilisées avec modération

Go

Bonnes pratiques

- privilégier les variables locales
- limiter les variables globales
- réduire la portée au strict nécessaire.

Go

Une constante est une valeur :

- nommée
- immuable
- connue à la compilation.

Go

Pour déclarer une constante

```
const TVA = 0.20
```

Go

Pour déclarer une constante

```
const TVA = 0.20
```

Ou

```
const TVA float64 = 0.20
```

Go

Exemple

```
const TVA = 0.20

prixHT := 100.0

prixTTC :=
    prixHT +
    prixHT * TVA

fmt.Println(prixTTC)
```

Go

Remarque

- Une constante ne peut pas être modifiée.
- L'instruction suivante déclenche une erreur :

```
TVA = 0.10
```

Go

Déclaration multiple

```
const (

    TVA = 0.20

    TAUX_REDUIT = 0.055

    PI = 3.14159

)
```

Go

iota

Le `iota` permet de générer automatiquement des constantes numériques.

Go

Exemple

```
const (
    LUNDI = iota
    MARDI
    MERCREDI
)
```

Go

Exemple

```
const (
    LUNDI = iota
    MARDI
    MERCREDI
)
```


Valeurs générées

Constante	Valeur
LUNDI	0
MARDI	1
MERCREDI	2

Go

Exemple complet

```
const (
    JANVIER = iota + 1
    FEVRIER
    MARS
    AVRIL
)
```

Go

Exemple complet

```
const (
    JANVIER = iota + 1
    FEVRIER
    MARS
    AVRIL
)
```


Valeurs générées

Constante	Valeur
JANVIER	1
FEVRIER	2
MARS	3
AVRIL	4

Go

Cas d'utilisation

- jours de la semaine
- mois
- états
- niveaux
- énumérations

Go

Exercice 1

Écrire un programme qui :

1. demande un nombre
2. affiche sa table de multiplication

Go

Exercice 2

Écrire un programme qui :

1. demande un entier positif
2. calcule sa somme :

$$1 + 2 + 3 + ... + n$$

3. affiche le résultat.