



Algorithmique et programmation
Mise en œuvre en Go

Consolidation et structures composées

Les sous-programmes :

retour sur des algorithmes du semestre 1

espaces de noms

portée d'une variable

Retour sur des algorithmes vus au semestre 1

Objectif : revoir les notions de

- Sous-programmes,
- paramètres,
- variables locales,
- appel d'un sous-programme

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

1er type d'algorithme : **saisie avec vérification**

On demande à l'utilisateur de donner une valeur ; cette valeur doit respecter des contraintes.

Exemple : donner une note ; contrainte : la note doit être comprise entre 0 et 20

L'algorithme vérifie la contrainte et redemande à l'utilisateur une valeur tant que la valeur ne respecte pas les contraintes.

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

Fonction demanderNote()

début

variable note réel

afficher("quelle note avez-vous eue à l'UE
MIA0105T ?")

lire(note)

tant que (note < 0 ou note > 20) faire

afficher("erreur, la note doit être comprise entre
0 et 20")

afficher("quelle note avez-vous eue à l'UE
MIA0105T ?")

lire(note)

fin tant que

retourner note

fin

Algorithme principal

variable resultat entier

resultat = demanderNote()

si resultat >= 10 alors

afficher("bravo !")

fin si

```
package main
```

```
import "fmt"
```

```
func demanderNote() float64 {
```

```
    var note float64
```

```
    fmt.Println("quelle note avez-vous eue à l'UE MIA0105T ?")
```

```
    fmt.Scan(&note)
```

```
    for (note < 0 || note > 20) {
```

```
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
```

```
        fmt.Println("quelle note avez-vous eue à l'UE
```

```
        MIA0105T ?")
```

```
        fmt.Scan(&note)
```

```
    }
```

```
    return note
```

```
}
```

```
func main() {
```

```
    var resultat float64
```

```
    resultat = demanderNote()
```

```
    if resultat >= 10 {
```

```
        fmt.Println("bravo !")
```

```
    }
```

```
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

C'est une fonction ==> elle retourne une valeur

Sans paramètre

Avec une variable locale : note

L'appel de la fonction est fait depuis l'algorithme principal :

- demanderNote()
- la fonction est exécutée
- le retour de la fonction est mis dans une variable de l'algorithme principal : resultat = demanderNote()

```
package main

import "fmt"

func demanderNote() float64 {
    var note float64
    fmt.Println("quelle note avez-vous eue à l'UE MIA0105T ?")
    fmt.Scan(&note)
    for (note < 0 || note > 20) {
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
        fmt.Println("quelle note avez-vous eue à l'UE MIA0105T ?")
        fmt.Scan(&note)
    }
    return note
}

func main() {
    var resultat float64
    resultat = demanderNote()
    if resultat >= 10 {
        fmt.Println("bravo !")
    }
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

Fonction demanderNote(message)

début

variable note réel

afficher(message)

lire(note)

tant que (note < 0 ou note > 20) faire

afficher("erreur, la note doit être comprise entre 0 et 20")

afficher(message)

lire(note)

fin tant que

retourner note

fin

Algorithme principal

variable resultat entier

resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")

si resultat >= 10 alors

afficher("bravo !")

fin si

```
package main
```

```
import "fmt"
```

```
func demanderNote(message string) float64 {
```

```
    var note float64
```

```
    fmt.Println(message)
```

```
    fmt.Scan(&note)
```

```
    for (note < 0 || note > 20) {
```

```
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
```

```
        fmt.Println(message)
```

```
        fmt.Scan(&note)
```

```
    }
```

```
    return note
```

```
}
```

```
func main() {
```

```
    var resultat float64
```

```
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
```

```
    if resultat >= 10 {
```

```
        fmt.Println("bravo !")
```

```
    }
```

```
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

C'est une fonction ==> elle retourne une valeur

Avec 1 paramètre : le message à afficher

- paramètre formel
- paramètre effectif

Avec une variable locale : note

L'appel de la fonction est fait depuis l'algorithme principal :

- resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
- la fonction est exécutée
- le retour de la fonction est mis dans une variable de l'algorithme principal : resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")

```
package main
```

```
import "fmt"
```

```
func demanderNote(message string) float64 {  
    var note float64  
    fmt.Println(message)  
    fmt.Scan(&note)  
    for (note < 0 || note > 20) {  
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")  
        fmt.Println(message)  
        fmt.Scan(&note)  
    }  
    return note  
}
```

```
func main() {  
    var resultat float64  
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")  
    if resultat >= 10 {  
        fmt.Println("bravo !")  
    }  
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

Que se passe-t-il à l'exécution ?

- en mémoire centrale
- exécution des instructions

```
package main

import "fmt"

func demanderNote(message string) float64 {
    var note float64
    fmt.Println(message)
    fmt.Scan(&note)
    for (note < 0 || note > 20) {
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
        fmt.Println(message)
        fmt.Scan(&note)
    }
    return note
}

func main() {
    var resultat float64
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
    if resultat >= 10 {
        fmt.Println("bravo !")
    }
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

Fonction demanderNote(message, limiteInf, limiteSup)

début

variable note réel

afficher(message)

lire(note)

tant que (note < limiteInf ou note > limiteSup) faire

afficher("erreur, la note doit être comprise entre",

limiteInf, " et", limiteSup)

afficher(message)

lire(note)

fin tant que

retourner note

fin

Algorithme principal

variable resultat entier

*resultat = demanderNote("quelle note avez-vous eue à l'UE
MIA0105T ? ", 0, 20)*

si resultat >= 10 alors

afficher("bravo !")

fin si

```
package main
```

```
import "fmt"
```

```
func demanderNote(message string, limiteInf float64, limiteSup  
float64) float64 {  
    var note float64  
    fmt.Println(message)  
    fmt.Scan(&note)  
    for (note < limiteInf || note > limiteSup) {  
        fmt.Println("erreur, la note doit être comprise entre ", limiteInf, " et  
", limiteSup)  
        fmt.Println(message)  
        fmt.Scan(&note)  
    }  
    return note  
}
```

```
func main() {  
    var resultat float64  
    resultat = demanderNote("quelle note avez-vous eue à l'UE  
MIA0105T ?", 0, 20)  
    if resultat >= 10 {  
        fmt.Println("bravo !")  
    }  
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

C'est une fonction ==> elle retourne une valeur

Avec 3 paramètres :

- le message à afficher
- la limite inférieure
- la limite supérieure

Avec une variable locale : note

L'appel de la fonction est fait depuis l'algorithme principal :

- resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
- la fonction est exécutée
- le retour de la fonction est mis dans une variable de l'algorithme principal : resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")

```
package main
```

```
import "fmt"
```

```
func demanderNote(message string, limiteInf float64, limiteSup float64) float64 {  
    var note float64  
    fmt.Println(message)  
    fmt.Scan(&note)  
    for (note < limiteInf || note > limiteSup) {  
        fmt.Println("erreur, la note doit être comprise entre ", limiteInf, " et ", limiteSup)  
        fmt.Println(message)  
        fmt.Scan(&note)  
    }  
    return note  
}
```

```
func main() {  
    var resultat float64  
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?", 0, 20)  
    if resultat >= 10 {  
        fmt.Println("bravo !")  
    }  
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

Que se passe-t-il à l'exécution ?

à vous

```
package main

import "fmt"

func demanderNote(message string, limiteInf float64, limiteSup float64) float64 {
    var note float64
    fmt.Println(message)
    fmt.Scan(&note)
    for (note < limiteInf || note > limiteSup) {
        fmt.Println("erreur, la note doit être comprise entre ", limiteInf, " et ", limiteSup)
        fmt.Println(message)
        fmt.Scan(&note)
    }
    return note
}

func main() {
    var resultat float64
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?", 0, 20)
    if resultat >= 10 {
        fmt.Println("bravo !")
    }
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

2ième type d'algorithme : **calculer la somme des n premiers entiers positifs**

Version 1 : calculer la somme des 10 premiers entiers positifs

Version 2 : calculer la somme des n premiers entiers positifs, n est donné à l'algorithme

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

Fonction somme10Premiers()

début

variable somme entier

variable nombre entier

somme = 0

nombre = 1

tant que (nombre <= 10) faire

 somme = somme + nombre

 nombre = nombre + 1

fin tant que

retourner somme

fin

Algorithme principal

variable resultat

resultat = somme10Premiers()

afficher(resultat)

```
package main
```

```
import "fmt"
```

```
func somme10Premiers() int {
```

```
    var (
```

```
        somme = 0
```

```
        nombre = 1
```

```
    )
```

```
    for (nombre <= 10) {
```

```
        somme = somme + nombre
```

```
        nombre = nombre + 1
```

```
    }
```

```
    return somme
```

```
}
```

```
func main() {
```

```
    var resultat int
```

```
    resultat = somme10Premiers()
```

```
    fmt.Println(resultat)
```

```
}
```

Calculer la somme des n premiers entiers positifs

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

fonction ou procédure ?

paramètres ?

variables locales ?

appel ?

```
package main
```

```
import "fmt"
```

```
func somme10Premiers() int {  
    var (  
        somme = 0  
        nombre = 1  
    )  
    for (nombre <= 10) {  
        somme = somme + nombre  
        nombre = nombre + 1  
    }  
    return somme  
}
```

```
func main() {  
    var resultat int  
    resultat = somme10Premiers()  
    fmt.Println(resultat)  
}
```

Calculer la somme des n premiers entiers positifs

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

Fonction sommeNPremiers(limite)

début

variable somme entier

variable nombre entier

somme = 0

nombre = 1

tant que (nombre <= limite) faire

 somme = somme + nombre

 nombre = nombre + 1

fin tant que

retourner somme

fin

Algorithme principal

variable resultat

resultat = sommeNPremiers(5)

afficher(resultat)

```
package main
```

```
import "fmt"
```

```
func sommeNPremiers(limite int) int {
```

```
    var (
```

```
        somme = 0
```

```
        nombre = 1
```

```
    )
```

```
    for (nombre <= limite) {
```

```
        somme = somme + nombre
```

```
        nombre = nombre + 1
```

```
    }
```

```
    return somme
```

```
}
```

```
func main() {
```

```
    var resultat int
```

```
    resultat = sommeNPremiers(5)
```

```
    fmt.Println(resultat)
```

```
}
```

Calculer la somme des n premiers entiers positifs

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

fonction ou procédure ?

paramètres ?

variables locales ?

appel ?

```
package main
```

```
import "fmt"
```

```
func sommeNPreiers(limite int) int {
```

```
    var (
```

```
        somme = 0
```

```
        nombre = 1
```

```
    )
```

```
    for (nombre <= limite) {
```

```
        somme = somme + nombre
```

```
        nombre = nombre + 1
```

```
    }
```

```
    return somme
```

```
}
```

```
func main() {
```

```
    var resultat int
```

```
    resultat = sommeNPreiers(5)
```

```
    fmt.Println(resultat)
```

```
}
```

Calculer la somme des n premiers entiers positifs

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

3ième type d'algorithme : **calculer le minimum de 2 nombres**

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

à vous :

fonction ou procédure ?

paramètres ?

variables locales ?

appel ?

**à vous :
algorithme et code
Go**

Calculer le minimum de 2 nombres

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales (corrigé)

Fonction minimum(x, y)

début

variable resultat réel

si $x < y$ alors

resultat = x

sinon

resultat = y

fin si

retourner resultat

fin

Algorithme principal

variable a, b, mini réel

afficher("donner 2 nombres :")

lire(a)

lire(b)

mini = minimum(a, b)

afficher(mini)

```
package main
```

```
import "fmt"
```

```
func minimum(x, y float64) float64 {
```

```
    var resultat float64
```

```
    if x < y {
```

```
        resultat = x
```

```
    } else {
```

```
        resultat = y
```

```
    }
```

```
    return resultat
```

```
}
```

```
func main() {
```

```
    var a, b, mini float64
```

```
    fmt.Println("donner 2 nombres :")
```

```
    fmt.Scan(&a)
```

```
    fmt.Scan(&b)
```

```
    mini = minimum(a, b)
```

```
    fmt.Println(mini)
```

```
}
```

Calculer le minimum de 2 nombres

Retour sur des algorithmes vus au semestre 1

Sous-programmes, paramètres, variables locales

Que se passe-t-il à l'exécution ?

à vous

```
package main
```

```
import "fmt"
```

```
func minimum(x, y float64) float64 {  
    var resultat float64  
    if x < y {  
        resultat = x  
    } else {  
        resultat = y  
    }  
    return resultat  
}
```

```
func main() {  
    var a, b, mini float64  
    fmt.Println("donner 2 nombres :")  
    fmt.Scan(&a)  
    fmt.Scan(&b)  
    mini = minimum(a, b)  
    fmt.Println(mini)  
}
```

Calculer le minimum de 2 nombres

Les sous-programmes :

retour sur des algorithmes du semestre 1

espaces de noms

portée d'une variable

Les espaces de noms

- Tous les identificateurs de Go vivent dans des ***espaces de noms***.
- Deux identificateurs identiques peuvent cohabiter, pourvu qu'ils vivent dans des espaces de noms différents.
- C'est à la création d'un identificateur (lorsqu'il est déclaré) que Go l'affecte à l'espace de noms où a eu lieu cette création.
- En résumé : *l'endroit où l'on déclare un identificateur dans le code d'un programme détermine son espace de noms (et, comme on le verra, sa **portée**).*

Espaces de noms et sous-programmes

- Les sous-programmes ajoutent leurs propres espaces de noms : par défaut, tout identificateur déclaré dans un sous -programme est associé à l'espace de noms de ce sous-programme, et à aucun autre. Ceci signifie que :

- Les paramètres et les variables déclarées dans un **func** ne sont visibles que par le code situé dans ce **func**. *Ils sont invisibles à l'extérieur de la fonction.*

```
package main

import "fmt"

func sommeNPremiers(limite int) int
{
    var (
        somme = 0
        nombre = 1
    )
    for (nombre <= limite) {
        somme = somme + nombre
        nombre = nombre + 1
    }
    return somme
}
```

```
func main() {
    var resultat int
    resultat = sommeNPremiers(5)
    fmt.Println(resultat)
}
```

**exemple de la somme des N
premiers entiers positifs**

- Les paramètres et les variables déclarées dans un **func** n'entrent pas en conflit avec les variables à l'extérieur du **func** , même si elles portent le même nom, car elles n'appartiennent pas au même espace de noms. **voir diapos suivantes**

Espaces de noms et sous-programmes

Les paramètres et les variables déclarées dans un func ne sont visibles que par le code situé dans ce func. Ils sont invisibles à l'extérieur de la fonction.

Analysez le code ci-contre : que pensez-vous de la visibilité des paramètres et des variables ? **à vous**

```
package main

import "fmt"

func demanderNote(message string) float64 {
    var note float64
    fmt.Println(message)
    fmt.Scan(&note)
    for (note < 0 || note > 20) {
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
        fmt.Println(message)
        fmt.Scan(&note)
    }
    return note
}

func main() {
    var resultat float64
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
    if resultat >= 10 {
        fmt.Println("bravo !")
    }
}
```

Saisie avec vérification

Portée d'une variable

- ***La portée d'une variable est l'endroit où la variable peut être utilisée.***
Elle est toujours déterminée par l'endroit où la variable a été déclarée dans le programme.
- Les variables pouvant être déclarées à des endroits différents, elles ont donc des portées différentes : ***locale*** ou ***globale***.
- Si la variable est déclarée ***dans un sous-programme***, elle est ***locale*** à ce sous-programme.
- Si la variable est déclarée ***à l'extérieur de tout sous-programme***, elle est ***globale*** à tout le programme.

Portée d'une variable/constante et conséquences

- Une variable (ou une constante) **globale** est accessible à partir de n'importe quelle ligne du programme (y compris dans un sous-programme). Si elle commence par une majuscule, elle est également visible dans tous les programmes qui importent le module (par exemple, **Math.Pi**)
- Un sous-programme peut lire et modifier une variable **globale**, mais c'est une pratique **très dangereuse** considérée comme une faute de programmation. On n'a évidemment pas ce problème avec les constantes globales puisqu'elles ne risquent pas d'être modifiées...
- Une variable/constante **locale** n'est accessible que par le sous-programme dans lequel elle est déclarée : seul le code de ce sous-programme y a accès et peut la modifier.
- À un niveau plus fin, on peut aussi déclarer une variable dans un bloc (d'un **for**, d'un **if**). Elle sera alors **locale** à ce bloc et cessera d'exister à la sortie de ce bloc. C'est d'ailleurs ce qui se passe avec les **for** utilisant un compteur ou un **range**...
- En conséquence, un sous-programme a accès à **ses** variables/constantes locales et aux variables/constantes globales. Il n'a pas d'accès aux autres variables/constantes locales

Portée d'une variable : illustration

- Dans le programme suivant, quelles sont les variables locales ? globales ?

```
package main

import "fmt"

var somme int = 100

func sommeNPremiers(lim int) int {
    var (
        somme = 0
        nbre = 1
    )
    for nbre <= lim {
        somme += nbre
        nbre++
    }
    return somme
}

func main() {
    fmt.Println(somme)
    var somme = sommeNPremiers(10)
    fmt.Println(somme)
}
```

Espaces de noms et sous-programmes

Les paramètres et les variables déclarées dans un func n'entrent pas en conflit avec les variables à l'extérieur du func , même si elles portent le même nom, car elles n'appartiennent pas au même espace de noms.

Analysez le code ci-contre. **à vous**

```
package main

import "fmt"

func demanderNote(message string) float64 {
    var note float64
    fmt.Printmessage)
    fmt.Scan(&note)
    for note < 0 || note > 20 {
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
        fmt.Print(message)
        fmt.Scan(&note)
    }
    return note
}

func main() {
    var note float64
    note = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
    if note >= 10 {
        fmt.Println("bravo !")
    }
}
```

Saisie avec vérification

Espaces de noms et sous-programmes

Même travail avec le code ci-contre

```
package main

import "fmt"

func minimum(a float64, b float64) float64 {
    var resultat float64
    if a < b {
        resultat = a
    } else {
        resultat = b
    }
    return resultat
}

func main() {
    var a, b, mini float64
    fmt.Print("donner 2 nombres :")
    fmt.Scan(&a)
    fmt.Scan(&b)
    mini = minimum(a, b)
    fmt.Println(mini)
}
```

Calculer le minimum de 2 nombres

Paramètres d'un sous-programme

Passage par valeur ou par référence

Les paramètres d'un sous-programme

passage de l'appelant à l'appelé :

fonctionnement (rappel)

Dans ce cas, les paramètres sont passés
par valeur (ou recopie) :

la valeur de a est recopiée dans x

et celle de b dans y

```
package main
```

```
import "fmt"
```

```
func minimum(x, y float64) float64 {  
    var resultat float64  
    if x < y {  
        resultat = x  
    } else {  
        resultat = y  
    }  
    return resultat  
}
```

```
func main() {  
    var a, b, mini float64  
    fmt.Print("donner 2 nombres :")  
    fmt.Scan(&a)  
    fmt.Scan(&b)  
    mini = minimum(a, b)  
    fmt.Println(mini)  
}
```

Calcul du minimum de deux nombres

Paramètres d'un sous-programme

Passage par valeur

Un autre exemple :

```
package main

import "fmt"

func echanger(a, b int) {
    var temp = a
    a = b
    b = temp
}

func main() {
    var (
        val1 = 10
        val2 = 100
    )
    fmt.Printf("Avant l'échange : val1 = %v, val2 = %v\n", val1, val2)
    echanger(val1, val2)
    fmt.Printf("Après l'échange : val1 = %v, val2 = %v\n", val1, val2)
}
```

Echange

Paramètres d'un sous-programme

Passage par référence

Il est également possible de passer les paramètres par référence.

Dans ce cas, les "originaux" (val1 et val2 dans l'exemple précédent) sont directement modifiés par le sous-programme.

Nous verrons des exemples plus tard dans le cours.