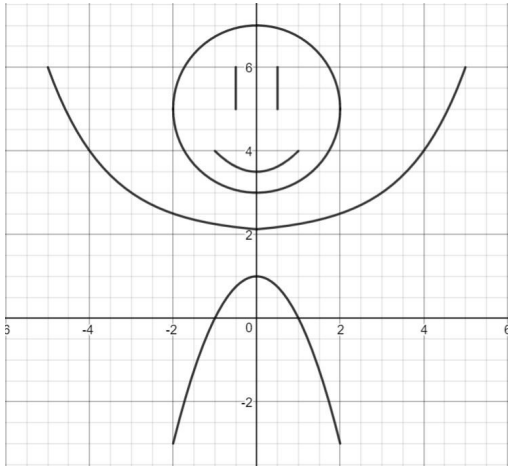
Go
4-Fonction

J. Vlasak

Go 4-Fonction

But du cours

- Savoir manipuler les fonctions en go.

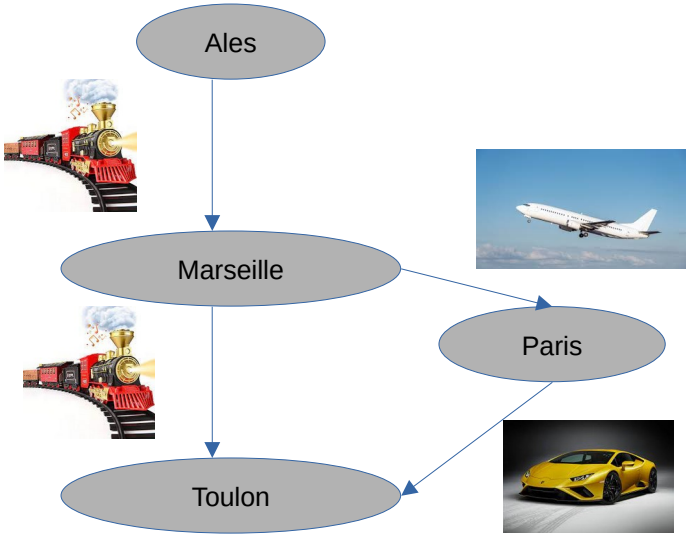


Go 4-Fonction

Souvenez vous de la machine de Turing

- Une fonction ou instruction fera transiter l'automate, c'est à dire changer la valeur des variables

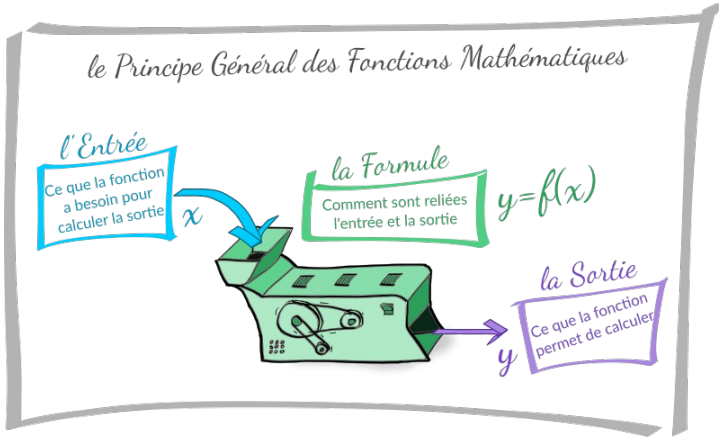
- Ici, par exemple l'état Ales peux se décrire par :
  - Coordonnées 44° 07′ 41″ nord, 4° 04′ 54″ est
  - Altitude Min. 116 m
  - Max. 356 m
  - Superficie 23,16 km2



Go 4-Fonction

Au sens mathématique

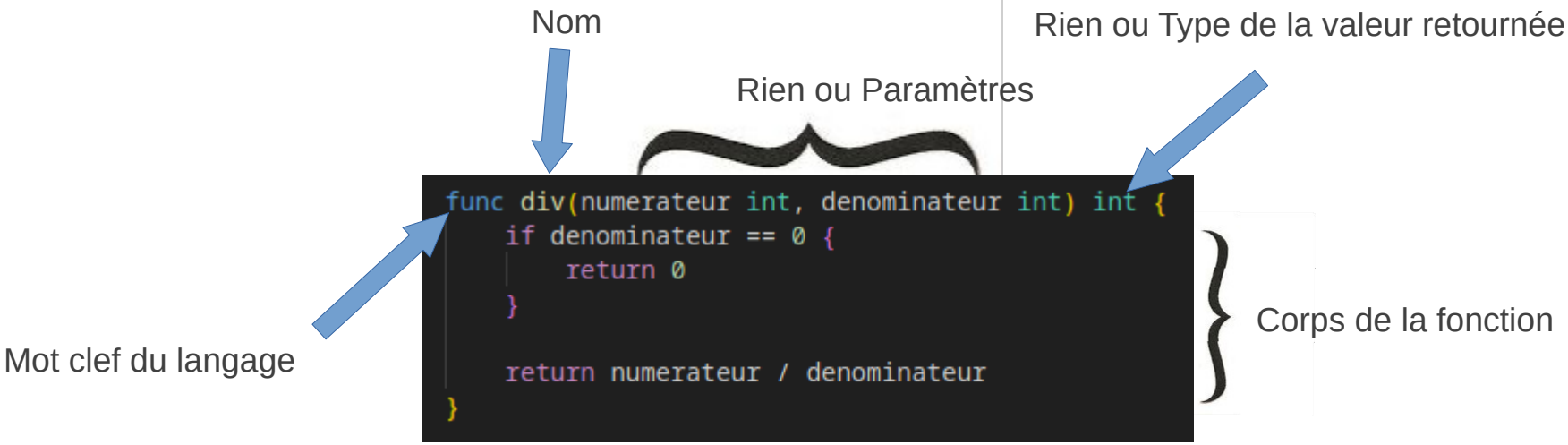
- une fonction permet de définir un résultat pour chaque valeur d’un ensemble appelé domaine.
- une fonction est la description de la méthode pour obtenir le résultat à partir de ses paramètres. Autrement dit une fonction est l'algorithme qui permet de la calculer.



Go 4-Fonction

Définition en Go

- La déclaration d'une fonction (ou signature) est constituée de 4 parties.



- Comme d'autre langage, utilise le mot clef « return »pour retourner une valeur.

Go 4-Fonction

L’utilisation

- Utilise le nom de la fonction suivie des paramètres

```
func main() {

    result := div(8, 3)
    fmt.Println(result)

}
```

Go 4-Fonction

Définition en Go

- Une fonction peut retourner plusieurs valeurs

```go
func divAndRemainder(numerateur int, denominateur int) (int, int) {
    if denominateur == 0 {
        return 0, 0
    }

    return numerateur / denominateur, numerateur % denominateur
}
```

- Ce qui donne dans l'appelant

```go
func main() {

    result, reste := divAndRemainder(8, 3)
    fmt.Println(result, reste)

}
```

Go 4-Fonction

Bizarre !!!

- Possibilité d’oublier les valeurs de retour dans l’appelant

```go
func main() {

    result, _ := divAndRemainder(8, 3)
    _, reste := divAndRemainder(8, 3)

    _, _ = divAndRemainder(8, 3)
    // ou plus simple
    divAndRemainder(8, 3)

    fmt.Println(result, reste)

}
```

Go 4-Fonction

Le traitement des erreurs en go

- Cas de la division par 0 ? Utilise un objet particulier « error » qui est retournée par la fonction.

```go
func divAndRemainder(numerateur int, denominateur int) (int, int, error) {
    if denominateur == 0 {
        return 0, 0, fmt.Errorf("division par 0")
    }

    return numerateur / denominateur, numerateur % denominateur, nil
}
```

- C’est assez lourd … mais on s’y fait !

```go
func main() {

    result, _, err := divAndRemainder(8, 3)
    _, reste, err := divAndRemainder(8, 0)
    _, _, _ = divAndRemainder(8, 3)

    if err != nil {
        fmt.Println("hummm y a un probleme")
        return
    }
    fmt.Println(result, reste, err)

}
```

Une petite pause

- Nous allons coder « divAndRemainder » dans VSC

Go 4-Fonction

A utiliser avec modération

- Nommer les valeurs de retour

```go
func divAndRemainder(numerateur int, denominateur int) (result int, remainder int, err error) {
    if denominateur == 0 {
        return 0, 0, fmt.Errorf("division par 0")
    }

    result, remainder = numerateur/denominateur, numerateur%denominateur
    return result, remainder, err
}
```

- Problème :
  - Peuvent être cache par une variable déclarée localement
  - La valeur affectée par «return» prime sur la valeur locale de «result» et «remainder»

```go
result, remainder = 10,20
return numerateur/denominateur, numerateur%denominateur, err
```

```go
if r, rem, err := divAndRemainder(5, 2); err != nil {
    fmt.Println("hummm y a un probleme")
} else {
    fmt.Println("toujour 2 et 1", r, rem)
}
```

Go 4-Fonction

A éviter……

- Valeur de retour implicite

```go
func divAndRemainder(numerateur int, denominateur int) (result int, remainder int, err error) {
    if denominateur == 0 {
        return 0, 0, fmt.Errorf("division par 0")
    }

    result, remainder = 10, 20
    return
}
```

Go 4-Fonction

Les fonctions comme type

- En Go, les fonctions sont traitées comme un type

- Il est possible :
  - déclaration d'une variable
  - assignation d'une variable
  - valeur de retour avec l'instruction return
  - initialisation d'un tableau
  - paramètre d'une méthode

```go
func add(i, j int) int {
    return i + j
}

func sub(i, j int) int {
    return i - j
}

func main() {

    uneAddition := add
    uneAddition(2,3)

    var uneSoustraction  func (i int, j int) int
    uneSoustraction = sub
    uneSoustraction(3,1)

}
```

Go 4-Fonction

Les fonctions anonymes (ou Lambda)

- Une fonction anonyme est une fonction créé sans nom.
- C’est l’équivalent d’un pointeur de fonction en C, mais avec une syntaxe beaucoup plus simple.
- Peut être directement utilisé (affectation, passage dans une fonction, ….)

```go
func main() {

    add := func(i, j int) int {
        return i + j
    }

    add(3, 2)

}
```

```go
func main() {

    fmt.Println(func(i, j int) int {
        return i + j
    }(2, 3))

}
```

Go 4-Fonction

Les closures

- Une closure est une fonction anonyme qui peut référencer des variables non locales sans restriction

```go
func main() {

    externeLambda := 3
    add := func(i, j int) int {
        fmt.Println(externeLambda)
        return i + j
    }

    add(3, 2)

}
```

Go 4-Fonction

Les closures

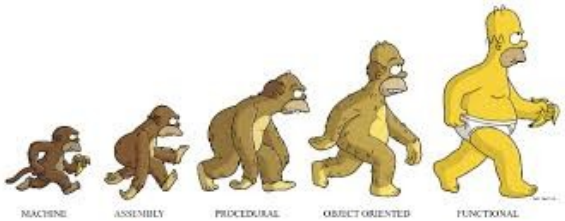
- Un exemple d’utilisation
- Simplifie le code

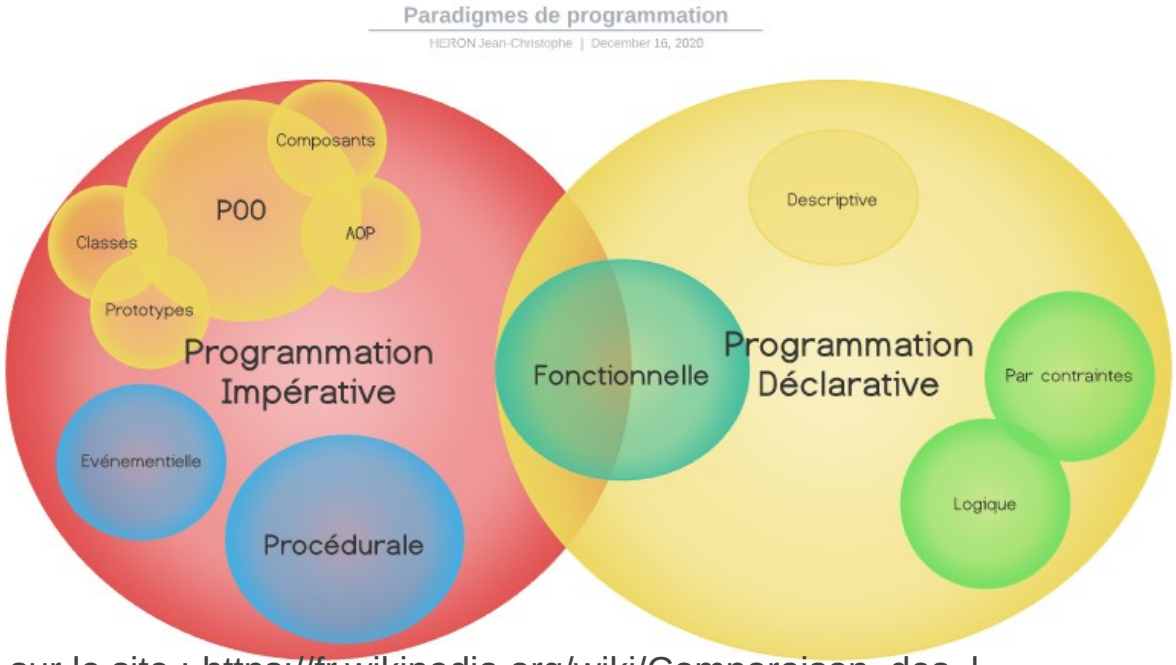
```go
func makeMult(base int) func(int) int {
    return func(factor int) int {
        return base * factor
    }
}

func main() {

    twoBase := makeMult(2)
    threBase := makeMult(3)

    fmt.Println(twoBase(2), threBase(2))

}
```

```go
func makeMult(base int) func(int) int {
    return func(factor int) int {
        return base * factor
    }
}

func operation(op func(int) int, factor int) int {
    return op(factor)
}

func main() {

    twoBase := makeMult(2)
    threBase := makeMult(3)

    fmt.Println(operation(twoBase, 2), operation(threBase, 2))

}
```

Go 4-Pour aller plus loin ...

Les langages modernes implémentent plusieurs paradigme de programmation





Allons voir ces différents paradigme sur le site : https://fr.wikipedia.org/wiki/Comparaison_des_langages_de_programmation_multi-paradigmes

Go 4-Fonction

Pour info

- Il existe des fonctions sans corps
- Le corps est définit ailleurs, par exemple dans un fichier assembleur

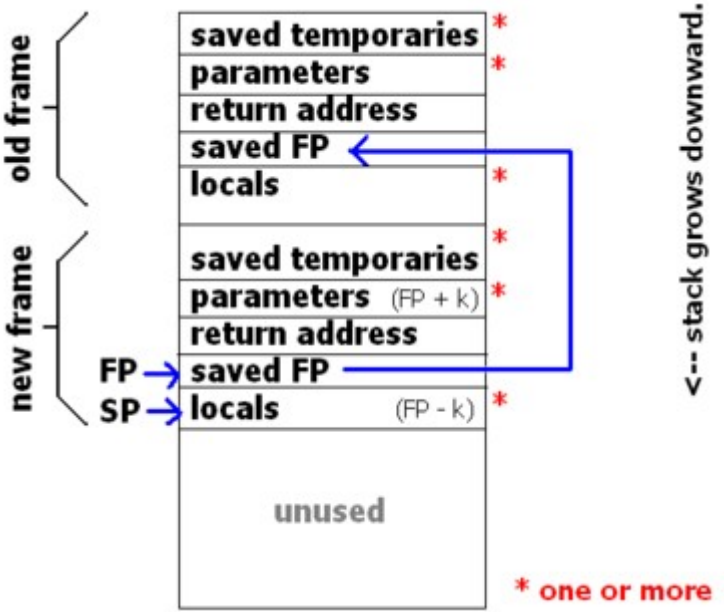
```
func Add(a int64, b int64) int64
```

```
TEXT ·Add(SB),$0-24
        MOVQ a+0(FP), AX
        ADDQ b+8(FP), AX
        MOVQ AX, ret+16(FP)
        RET
```

Go 4-Fonction

Un dernier point ...

- Les paramètres d'une fonction sont passées par valeur
- Pour modifier la valeur d'un paramètre, obligé d'utiliser un pointeur.



Go 4-Fonction