
interface{}

Objectifs de l’atelier

Savoir implémenter une interface

Être capable de réaliser sa propre interface

Savoir Quand utiliser des interfaces

interface{} != object()

Problème go est un langage typé !

```
var i int
i = 1
fmt.Printf("%T\n",i) // int
i = 2.34  // constant 2.34 truncated to integer
fmt.Printf("%T\n",i)
i = "ABCD" // cannot use "ABCD" (type string) as type int in assignment
fmt.Printf("%T\n",i)
```

solution : utiliser une interface

```
var i interface{}
i = 1
fmt.Printf("%T\n",i) // int
i = 2.34
fmt.Printf("%T\n",i) // float64
i = "ABCD"
fmt.Printf("%T\n",i) // string
```

interface pour tous types

```
func display(i interface{}){
...
}
```

assertion de type

```
1  var numb int = 10
2  var i interface{} = numb
3  value, ok := i.(int)
4  fmt.Printf("%T - %v - %v", value, value, ok)
5  // output: int - 10 - true
6
7
8
9
```

assertion de type

```
1  func display(i interface{}){
2      switch i.(type){
3          case int :
4              fmt.Println("this is int")
5          case float64 :
6              fmt.Println("this is float64")
7          case string :
8              fmt.Println("this is string")
9      }
10 }
11
```

interface of

contract

Problème : je veux pas afficher les valeurs par défaut

```
type User struct{
   Name, email string
}
```

```
u := User{“Rob Pike”,”r@google.com”}
fmt.Println(u)
// {“Rob Pike”, ”r@google.com”}
```

Solution : implémenter l'interface Stringer

```
type Stringer interface{
   String() string
}
```

```
type User struct{
   Name, email string
}

func (u *User)String()string {
   return u.email
}
```

```
u := User{"Rob Pike","r@google.com"}
fmt.Println(&u)
// ?
```

Solution : implémenter l'interface Stringer

```
type Stringer interface{
   String() string
}
```

```
type User struct{
   Name, email string
}

func (u *User)String()string {
   return u.email
}
```

```
u := User{"Rob Pike","r@google.com"}
fmt.Println(&u)
// r@google.com
```

Solution : oops !

```
type Stringer interface{
   String() string
}
```

```
type User struct{
   Name, email string
}

func (u *User)String()string {
   return u.email
}
```

```
u := User{"Rob Pike","r@google.com"}
fmt.Println(u)
// {"Rob Pike", "r@google.com"}

fmt.Println(&u)
//r@google.com
```

Solution : yes !

```
type Stringer interface{
   String() string
}
```

```
type User struct{
   Name, email string
}

func (u User)String()string {
   return u.email
}
```

```
u := User{"Rob Pike","r@google.com"}
fmt.Println(u) //r@google.com

fmt.Println(&u) //r@google.com
```


Struct
interface
in a

Problème n°1

```
type User struct{
   FirstName, LastName string
}

func (u *User)Walk(s int) {
     fmt.Println("Walk", s, "steps")
}

func (u *User)Run(v float64) {
     fmt.Println("Run at", v, "km/h")
}
```

```
u := User{“Rob”,”Williams”}
go u.Walk(150)  // Walk 150 steps
// et ensuite ?
```

Quand tu cours tu ne marche pas en même temps !

```go
type User struct{
   FirstName, LastName string
}

func (u *User)Walk(s int) {
    fmt.Println("Walk", s, "steps")
}

func (u *User)Run(v float64) {
    fmt.Println("Run at", v, "km/h")
}
```

```go
u := User{“Rob”,”Williams”}
go u.Walk(150)  // Walk 150 steps
go u.Run(11.52) // Run at 11.52 km/h
```

Solution n°1

```
type User struct{
   FirstName, LastName string
   Move                func(int)
}

func Walk(s int) {
    fmt.Println("Walk", s, "steps")
}

func Run(v int) {
    fmt.Println("Run at", v, "km/h")
}
```

```
ua := User{"Bob", "William", Walk()}
ua.Move(150) // Walk 150 steps

ub := User{"Ken", "Thomson", Run()}
// et ensuite ?
```

Une signature de fonction reste une signature de fonction...

```
type User struct{
   FirstName, LastName string
   Move                func(int)
}

func Walk(s int) {
    fmt.Println("Walk", s, "steps")
}

func Run(v int) {
    fmt.Println("Run at", v, "km/h")
}
```

```
ua := User{"Bob", "William", Walk()}
ua.Move(150) // Walk 150 steps

ub := User{"Ken", "Thomson", Run()}
ub.Move(11.52) // error float64 not
int
```

Solution n°2

```
func Walk() func(interface{}) error {
     return func(i interface{}) error {
          res, ok := i.(int)
          if !ok {
               return errors.New("...")
          }
          fmt.Println("Walk",res,"steps")
          return nil
     }
}
```

```
func Run() func(interface{}) error {
     return func(f interface{}) error {
          res, ok := f.(float64)
          if !ok {
               return errors.New("...")
          }
          fmt.Println("Run at",res,"km/h")
          return nil
     }
}
```

Ok ça marche mais c’est compliqué

```
type User struct {
     FirstName, LastName string
     Move  func(interface{}) error
}
```

```
ua := User{"Bob", "William", Walk()}
ua.Move(150) // Walk 150 steps
ua.Move = Run()
```

Solution créer une interface

```
type Mover interface {
    Move()
}
```

Créer des types concrets qui implémentent cette interface

```go
type Walk int

func (w Walk) Move() {
    fmt.Println("Walk at", w)
}
```

```go
type Run float64

func (r Run) Move() {
    fmt.Println("Run at", r)
}
```

Et enfin ajouter l’interface dans User

```
type User struct{
    FirstName, LastName string
    Mover
}
```

```
// User A is Walking
u := User{"Bob", "William",
Walk(12618)}
u.Move()
u.Mover = Run(11.53)
ub.Move()
```

Ressources

https://www.ardanlabs.com/blog/2015/09/composition-with-go.html

https://play.golang.org/p/4PXQ1BCpRA

https://play.golang.org/p/jsjyikdSo_

https://play.golang.org/p/FEbIVcEFsd

https://play.golang.org/p/NPlyu9QKQI