func and methods

Thèmes de l’atelier

Les différentes signatures de fonction

Les fonction anonymes

Création de méthodes sur une structure

Signature de fonction

fonction : signature

```go
func Hello (n string) string {
     return “Salut ” + n
}
```

fonction : signature

```
func Hello (n string) (string, error) {
     if len(n) == 0{
          return “”, errors.New(“no name”)
     }
     return “Salut ” + n, nil
}
```

fonction : signature

```go
func Hello (n string) (res string, e error) {
    if len(n) == 0{
        e = errors.New(“no name”)
    }
    res = “Salut ” + n
    return
}
```

fonction : signature

```
func Hello (n, ln string) (string, error) {
    if len(n) == 0 || len(ln) == 0 {
        return "", errors.New("no name")
    }
    return "Salut " + n + " " + ln, nil
}
```

fonction : signature

```go
func Hello (n ...string) string {
     var res string
     for k := range n{
          res += n[k] + " "
     }
     return "Salut " + res
}

```

fonction : anonyme

```go
f := func(i int) int {
    return i / 2
}
fmt.Println(f(1))  // 0
fmt.Println(f(10)) // 5
```

fonction : closure

```go
func fibo() func() int {
    a, b := 0, 1
    return func() int {
        a, b = b, a + b
        return b
    }
}

```

Méthode : attacher à une structure

```
func (u *User) SayHi() string {
    return “Hello ” + u.FirstName
}
…
u := User{“Bob”}
fmt.Println(u.SayHi()) // output : Hello Bob
```

Méthode : attacher à une structure

```
func (u User) SayHi() string {
    return “Hello ” + u.FirstName
}
…
u := User{“Bob”}
hi := u.SayHi()
fmt.Println(hi) // output : Hello Bob

```

Méthode : création d’une structure

```go
func NewUser(n string) *User {
     return &User{Name: n}
}
…
u := NewUser(“Bob”)
fmt.Printf(“%T : %v”, u, u)
// output : *main.User : &User{Bob}

```

Méthode : arguments optionnels

```go
type User struct {
  Name string
}
```

```go
type User struct {
  FirstName           string
  LastName            string
  DateOfBirth         time.Time
  DateOfLastConnected time.Time
  IsActive            bool
  IsADummyUser        bool
}
```

Méthode : arguments optionnels

```
func NewUser(firstNane, lastName string, dateOfBirth , dateOfLastConnected time.Time, is
isADummyUser bool ) *User {
    return &User{FirstName: firstName, LastName: lastName, DateOfBirth: dateOfBirth,
DateOfLastConnected: dateOfLastConnected, IsActive: isActive, IsADummyUser: isADummyUser
}
…
u := NewUser("Bob", "Wilson", time.Now(), time.Now(), true, true)
fmt.Printf("%v", u)
// output : *main.User : &User{"Bob", "Wilson", 2009-11-10 23:00:00 +0000 UTC, 2009-11-10 2
+0000 UTC, true, true}
```

Méthode : arguments optionnels

```go
type UserConfig struct{
    dateOfBirth, dateOfLastConnected  time.Time
    isActive, isADummyUser            bool
}

func NewUser(fn, ln string, uc *UserConfig) *User {
     u := User{
          FirstName: fn,
          LastName:  ln,
     }
     if uc != nil {
          u.DateOfBirth = uc.DateOfBirth
          u.DateOfLastConnected = uc.DateOfLastConnected
          u.IsActive = uc.IsActive
          u.IsADummyUser = uc.IsADummyUser
     }
     return &u

}
```

Méthode : arguments optionnels

```
uc := UserConfig{time.Now(),time.Now(),true,true}
u := NewUser("Bob", "Wilson", &uc)
fmt.Printf("%T : %v", u, u)
// output : *main.User : &User{"Bob", "Wilson", 2009-11-10 23:00:00 +0000
UTC, 2009-11-10 23:00:00 +0000 UTC, true, true}

u2 := NewUser("Daves", "Riss", nil)
// output : *main.User : &{Daves Riss {0001-01-01 00:00:00 +0000 UTC
0001-01-01 00:00:00 +0000 UTC false false}}
```

links

http://dave.cheney.net/2014/10/17/functional-options-for-friendly-apis
https://commandcenter.blogspot.fr/2014/01/self-referential-functions-and-design.html
https://dave.cheney.net/2016/11/13/do-not-fear-first-class-functions