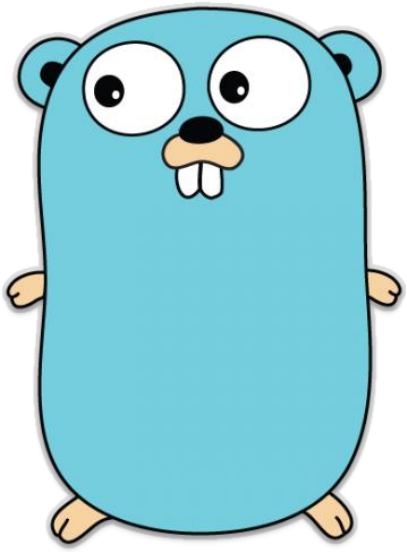

Workshop GO

B-INN-000

Day 01

Les bases

Epitech Toulouse – Florian Davasse & Florent Poinsard

Task 00

Infos utiles

Quelques liens utiles :

- *https://gobyexample.com/*
- *https://golang.org/pkg/*
- *https://gist.github.com/leg0ffant/3bee4829ce2ad8fd026c*
- *https://golang.org/ref/spec*
- https://talks.golang.org/2015/gophercon-goevolution.slide#1

Task 01

my_print_digits

Vous devez écrire une fonction qui doit afficher tous les chiffres.

```
func my_print_digits() {$
```

```
JOK3R@epitech.eu→  ~/projets/GO/Day03 ./my_print_digits
0123456789%
```

Task 02

my_putstr

Vous devez écrire une fonction qui doit afficher une chaine de caractère, caractère par caractère. (Vous ne devez pas afficher la chaine de caractère d'un coup).
Cet exercice a pour but de vous familiariser avec les tableaux en go & les **runes**.

```
func my_putstr(str string) {$
```

```
JOK3R@epitech.eu→  ~/projets/GO/Day03 go run my_putstr.go
Salut%
```

Task 03

my_print_comb

Vous devez écrire une fonction qui doit afficher dans l'ordre croissant tous les nombres possibles de faire avec 3 chiffres différents.

```
func my_print_comb() {$
```

```
JOK3R@epitech.eu→  ~/projets/GO/Day03 ./my_print_comb
012, 013, 014, 015, 016, 017, 018, 019, 023, 024, 025, 026, 027, 028, 029, 034,
035, 036, 037, 038, 039, 045, 046, 047, 048, 049, 056, 057, 058, 059, 067, 068,
069, 078, 079, 089, 123, 124, 125, 126, 127, 128, 129, 134, 135, 136, 137, 138,
139, 145, 146, 147, 148, 149, 156, 157, 158, 159, 167, 168, 169, 178, 179, 189,
234, 235, 236, 237, 238, 239, 245, 246, 247, 248, 249, 256, 257, 258, 259, 267,
268, 269, 278, 279, 289, 345, 346, 347, 348, 349, 356, 357, 358, 359, 367, 368,
369, 378, 379, 389, 456, 457, 458, 459, 467, 468, 469, 478, 479, 489, 567, 568,
569, 578, 579, 589, 678, 679, 689, 789%
```

Task 04

my_print_array

Dans cet exercice vous devez créer un tableau de string et afficher son contenu.
Cet exercice a pour but de vous initier à la création d'**array**.

```
func my_print_array() {}$
```

Task 05

my_print_args

Dans cet exercice vous devez créer une fonction qui va recevoir des arguments et les afficher sur la sortie standard. Pensez à la gestion d'erreur.
Cet exercice à pour but de vous familiariser avec les fonctions de la lib « **os** ».

```
func my_print_args() {}$
```

```
[pflorent@localhost Day_1]$ ./my_print_args "Le" "go" "c'est" "trop" "bien"
Le
go
c'est
trop
bien
[pflorent@localhost Day_1]$
```

Task 06

my_divide

Dans cet exercice vous devez créer une fonction qui prendra 2 arguments et qui doit return le résultat & le reste de la division des deux arguments. Vous vous familiariserez avec les multiples retours de fonction au cours de cet exercice.

```
func myDivide(diviseur, operande int) (int, int) {
```

```
JOK3R@epitech.eu→  ~/projets/GO/Day03 go run my_divide.go
Resultat : 2 Reste : 1
Resultat : 3 Reste : 0
```

Task 07

my_struct

Au cours de cet exercice vous devez créer une structure (par exemple : « fête »), lui donner des attributs (« nombre de personnes » et « lieu »). Puis donner une valeur à ces attributs.
Afficher votre structure sur la sortie standard.
Dans cet exercice vous découvrirez les **struct**.

```
func my_struct() {}
```

Task 08

my_pointers

Les pointeurs... En go aussi il existe des pointeurs, leur utilisation reste assez simple contrairement au C.

Vous devez écrire une fonction qui prend comme argument un pointeur sur un int. Cette fonction devra modifier la valeur du pointeur (par n'importe quelle valeur).

Vous afficherez dans le main la variable avant & après.

```
func modifPtr(ptr *int) {
```

```
JOK3R@epitech.eu→  ~/projets/GO/Day03 go run ptr.go
1
10389876888
```

Task 09

my_gouroutine

Dans cette série d'exercices vous allez découvrir qu'est-ce qu'une **goroutine**, son utilisation et sa puissance.
Créer une simple **goroutine** que votre main appellera. Mettez une boucle qui compte jusqu'à 50 et qui affiche une valeur à chaque tour. Vous devez **attendre la fin de l'affichage pour quitter le programme**.

Vous devez utiliser time.Sleep.

```
func my_goroutine() {}
```

```
JOK3R@epitech.eu→  ~/projets/GO/Day03 go run task09.go
0
1
2
...
49
50
```

Task 10

my_gouroutine

Maintenant que vous arrivez à lancer une goroutine, lancez-en deux en même temps depuis votre main. Mettez une boucle qui affiche des caractères dans chacune des goroutines, et comparer vos outputs !

- Ne vous embêter pas à avoir exactement le même output.

```
func my_goroutine() {}
```

```
JOK3R@epitech.eu→  ~/projets/GO/Day03 go run task09.go
0
1
2
0
3
1
...
50
49
50
```

Task 11

my_channel

Le principe des goroutines étant compris, nous allons maintenant trouver un moyen de **communiquer** entre le processus de notre programme et celui de notre goroutine… Pour se faire, nous allons utiliser les canaux.
Reprenez la Task 9 et au lieu d'utiliser sleep, utiliser les canaux pour attendre la fin de l'exécution de la goroutine.

```
func my_goroutine(wait_chan chan int) {$
```