Golang I

Présentation du cours

- Durée: 30h
- Evaluation sous forme de Projet

Objectif du cours

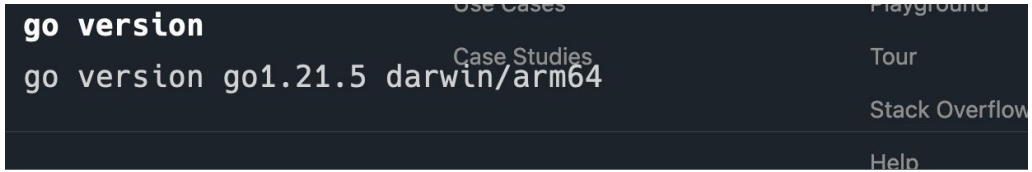
- les bases de la programmation en Go
- les principaux concepts et fonctionnalités du langage

Matériels

- OS: Windows, MacOS, Linux (léger biais), GitPod sinon
- Droits roots ou au moins Go, Git, Docker & Docker-compose installés

Installation

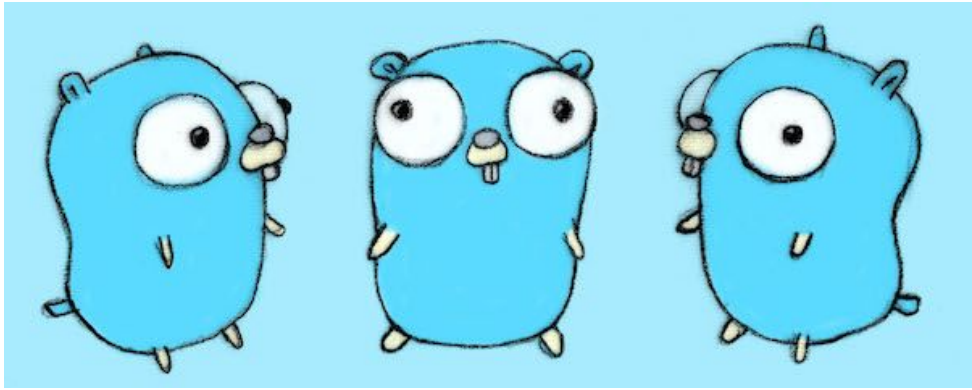
- Windows & Linux:
  https://go.dev/doc/install
- MacOS:
  brew install go

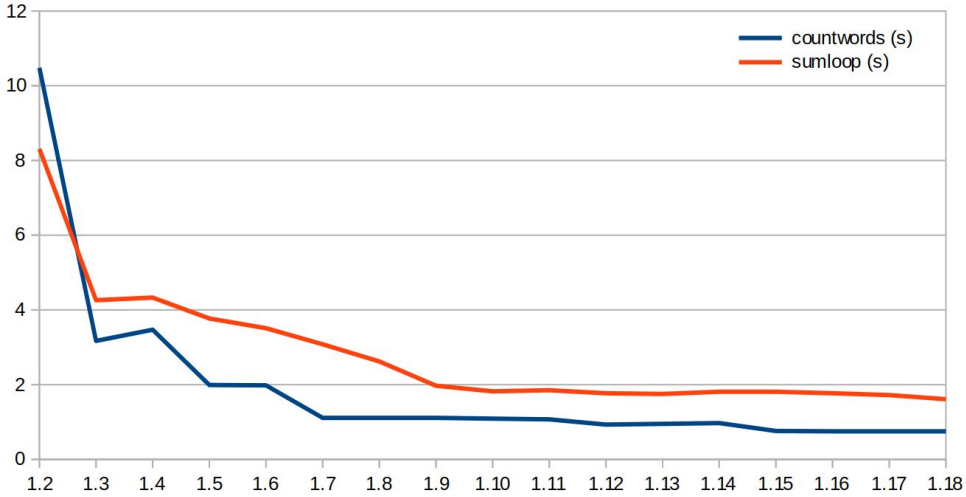


Histoire

Début (2007) : Développé par les ingénieurs de Google Robert Griesemer, Rob Pike et Ken Thompson.

Publication Publique (2009) : Go a été officiellement présenté au public en tant que language open source.





Pourquoi Go ?

Compilation Efficace :

- Processus de compilation efficace qui accélère les cycles de développement.

Support de la Concurrence :

- Support intégré de la concurrence avec les goroutines et les channels.

Syntaxe Claire et Simple :

- Syntaxe minimaliste de Go qui améliore la lisibilité et réduit le code superflu.

Bibliothèque Standard Complète :

- Bibliothèque standard complète pour une variété de tâches de programmation.

Scalabilité et Performance :

- Conçu pour la scalabilité et des performances optimales, particulièrement dans le développement web.

Qui utilise Go ?

Hello World

Let’s Code

Cross compilation:

```
GOOS=linux GOARCH=amd64 go build -o helloworld
GOOS=windows GOARCH=amd64 go build -o helloworld.exe
GOOS=darwin GOARCH=arm go build -o helloworld
GOOS=linux GOARCH=arm GOARM=7 go build -o hellworld.armv7
```

FizzBuzz

Ayant un entier n, retournes un tableau de chaînes de caractères answer (indexé à partir de 1) où :

answer[i] == "FizzBuzz" si i est divisible par 3 et 5.

answer[i] == "Fizz" si i est divisible par 3.

answer[i] == "Buzz" si i est divisible par 5.

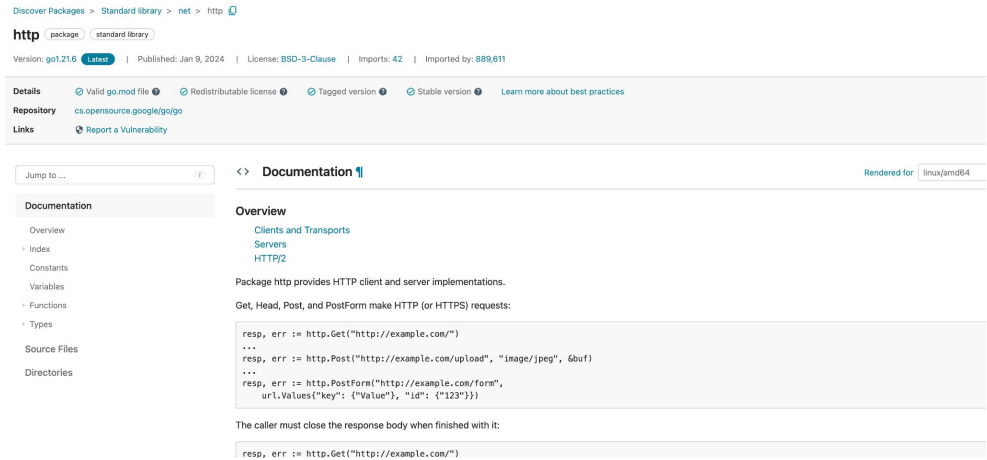
answer[i] == i (sous forme de chaîne de caractères) si aucune des conditions ci-dessus n'est vraie.

n = 5 => ["1","2","Fizz","4","Buzz"]

Documentation:

- https://gobyexample.com/

- https://pkg.go.dev/

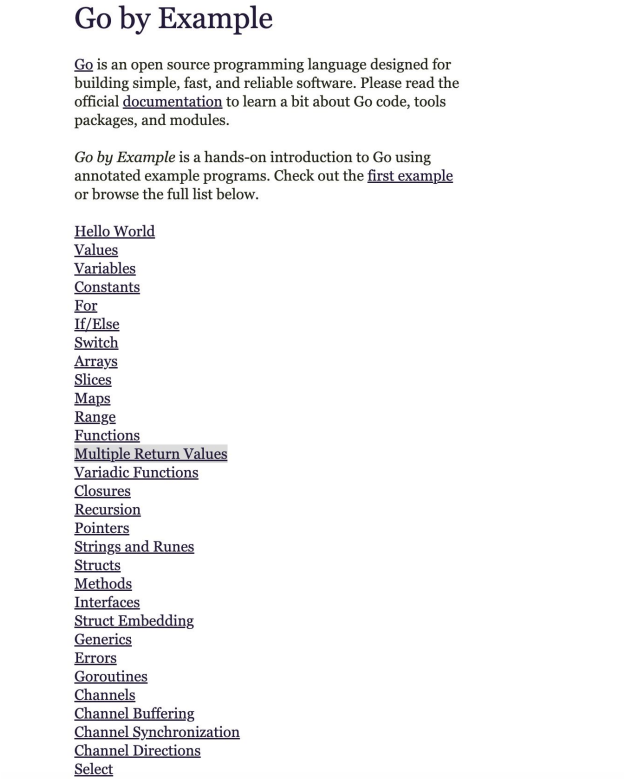


Syntax and Language Features

- Variables:
  ```
  var a int = 1
  a := 1
  ```
- High Order Function
- Pointers
- Multiple Return Values
- Slices
- Structs
- Interface
- Panic

  etc

  https://gobyexample.com



Go by Example

Go is an open source programming language designed for building simple, fast, and reliable software. Please read the official documentation to learn a bit about Go code, tools packages, and modules.

Go by Example is a hands-on introduction to Go using annotated example programs. Check out the first example or browse the full list below.

Hello World
Values
Variables
Constants
For
If/Else
Switch
Arrays
Slices
Maps
Range
Functions
Multiple Return Values
Variadic Functions
Closures
Recursion
Pointers
Strings and Runes
Structs
Methods
Interfaces
Struct Embedding
Generics
Errors
Goroutines
Channels
Channel Buffering
Channel Synchronization
Channel Directions
Select

Go modules

Tutoriel: https://go.dev/blog/using-go-modules

Création d’un module pour fib:

    go mod init <>

Ajout et Upgrade d’une dependencies

    go get -t -u <>

Quid des dependencies

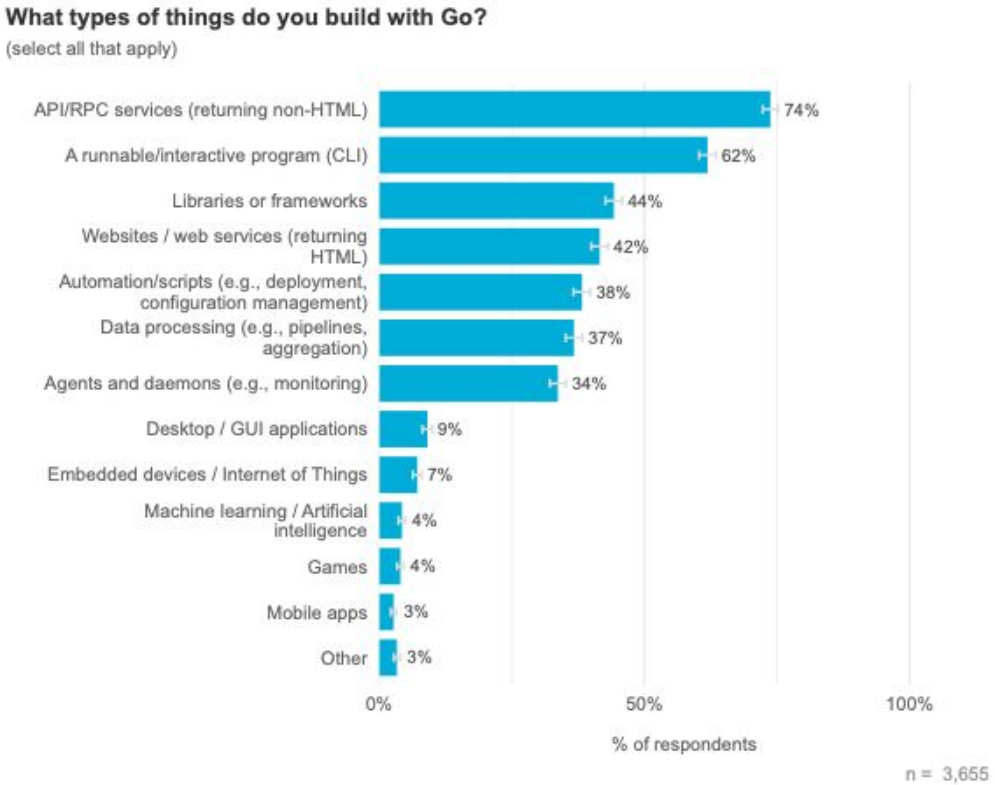
    go mod tidy

Philosophie

- SOLID et RAII
- DRY/KISS à la poubelle
- Importance du naming

  Exemple: connexion à MongoDB

Utilisation souvent en CLI et RPC



ContainsDuplicate

Ayant un tableau d'entiers nums, retournes true si une valeur apparaît au moins deux fois dans le tableau, et retournes false si chaque élément est distinct.

nums = [1,2,3,1] => true
nums = [1,2,3,4] => false
nums = [1,1,1,3,3,4,3,2,4,2] => true

Moving Zeroes

Étant donné un tableau d'entiers nums, déplaces tous les zéros à la fin tout en maintenant l'ordre relatif des éléments non nuls sans faire de copie du tableau.

Exemple:
nombres = [0,1,0,3,12] => [1, 3, 12, 0, 0]
nombres = [0] => [0]

ReverseString

Écris une fonction qui inverse une chaîne de caractères. La chaîne d'entrée est fournie sous la forme d'un tableau de caractères s.

s = ["h","e","l","l","o"] => ["o","l","l","e","h"]

Tests et Benchmarks

```go
func FibonacciLoop(n int) int {
    f := make([]int, n+1, n+2)
    if n < 2 {
        f = f[0:2]
    }
    f[0] = 0
    f[1] = 1
    for i := 2; i <= n; i++ {
        f[i] = f[i-1] + f[i-2]
    }
    return f[n]
}

func FibonacciRecursion(n int) int {
    if n <= 1 {
        return n
    }
    return FibonacciRecursion(n-1) + FibonacciRecursion(n-2)
}
```

FirstBadVersion

Vous êtes un chef de produit et dirigez actuellement une équipe pour développer un nouveau produit. Malheureusement, la dernière version de votre produit ne réussit pas le contrôle qualité. Étant donné que chaque version est développée sur la base de la version précédente, toutes les versions après une version défectueuse sont également défectueuses.

Supposons que vous ayez n versions [1, 2, ..., n] et que vous souhaitiez trouver la première version défectueuse, qui entraîne également la défectuosité de toutes les versions suivantes.

On vous donne une API bool isBadVersion(version) qui retourne si la version est défectueuse. Implémentez une fonction pour trouver la première version défectueuse. Vous devez minimiser le nombre d'appels à l'API.

Implementons: func firstBadVersion(n int) int {}

First Unique Character in a String

Ayant une chaîne de caractères s, trouves le premier caractère non répétitif et retournes son indice. Si cela n'existe pas, retournez -1.

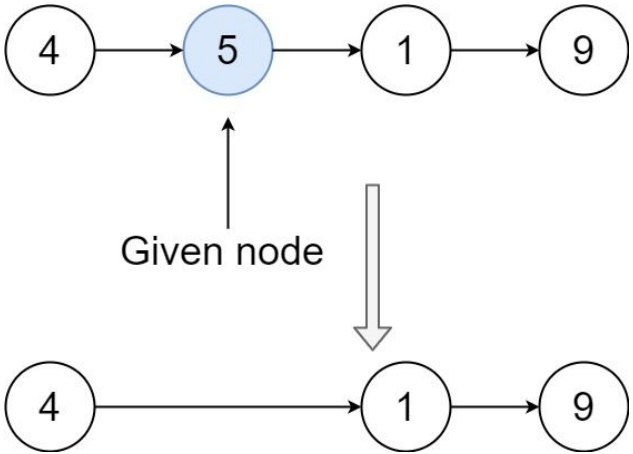
s = "leetcode" => 0

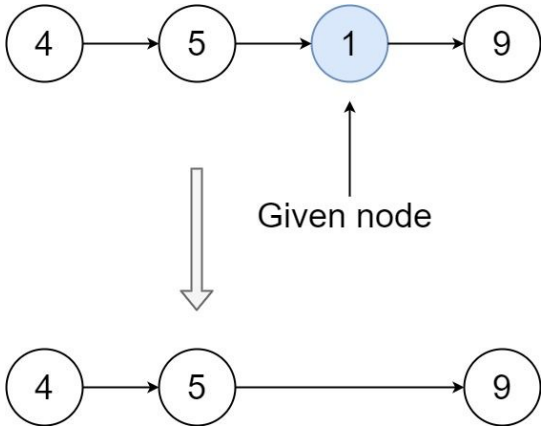
s = "loveleetcode" => 2

s = "aabb" => -1

Delete Node in a Linked List

Il y a une liste chaînée simple avec une tête (head) et nous voulons supprimer un nœud (node) de celle-ci. On vous donne le nœud à supprimer, node. Vous n'aurez pas accès au premier nœud de la tête (head).

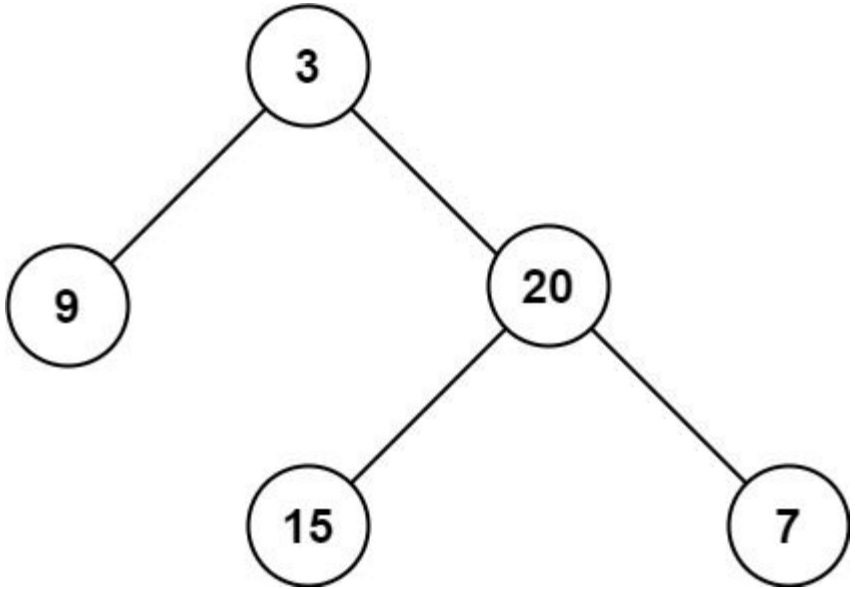




MaxDepthBTree

Étant donné la racine d'un arbre binaire, retournes sa profondeur maximale.

La profondeur maximale d'un arbre binaire est le nombre de nœuds le long du chemin le plus long, de la racine au feuille le plus éloigné.



root = [3,9,20,null,null,15,7] => 3

root = [1,null,2] => 2

Creation un CLI

Create a CLI application for the following operations:

- Print all tasks
- Print a specific task by ID
- Create a new task
- Update an existing task
- Delete a task

Restful API with Chi/Gin

Create routes for the following operations:

- Get all tasks
- Get a specific task by ID
- Create a new task
- Update an existing task
- Delete a task

GRPC with ConnectRPC

Create routes for the following operations:

- Get all tasks
- Get a specific task by ID
- Create a new task
- Update an existing task
- Delete a task

Ajout de base de données

Sur Disk, Redis ou Mongo

Ajout d’un CI

GitlabCI ou GitHub Actions

Exercices Open Data

Elections 2022

https://www.data.gouv.fr/fr/datasets/election-presidentielle-des-10-et-24-avril-2022-resultats-definitifs-du-1er-tour/#/resources

Le nombre de votes exprimés
Le nombre des votes par candidats
Le nombres de votes par candidats par départements
Le palmarès des départements par le nombre de votants

Nouvelles Hospitalisations dues à la COVID

Génération de Graph via https://github.com/gonum/plot

https://www.data.gouv.fr/fr/datasets/donnees-hospitalieres-relatives-a-lepidemie-de-covid-19/

Panic/Recover

Panic:

Un événement grave qui interrompt le programme.

Causée par une erreur grave (division par zéro, etc.)

Récupération:

Intercepte une panique et reprend l'exécution.

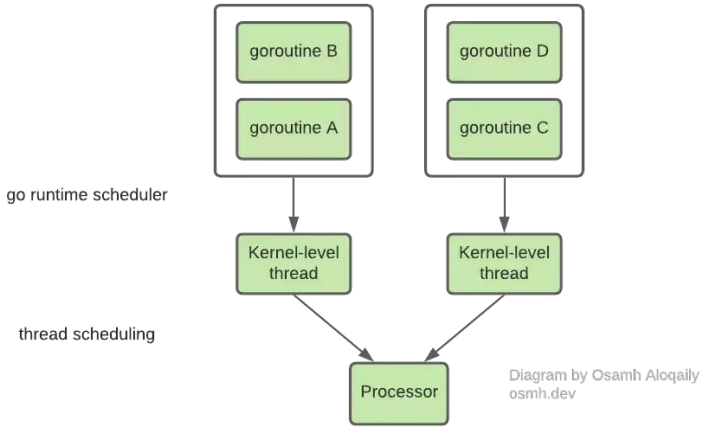
Permet d'effectuer des opérations de nettoyage.

```go
package main

import "fmt"

func mayPanic() {
    panic("a problem")
}

func main() {



    defer func() {
        if r := recover(); r != nil {

            fmt.Println("Recovered. Error:\n", r)
        }
    }()

    mayPanic()

    fmt.Println("After mayPanic()")
}
```

```
$ go run recover.go
Recovered. Error:
 a problem
```

Concurrence et parallelism

Goroutines:
Fonctions légères s'exécutant en parallèle. Permettant d'exploiter la puissance des processeurs multicœurs.




```go
package main

import (
    "fmt"
    "time"
)

func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}

func main() {

    f("direct")


    go f("goroutine")



    go func(msg string) {
        fmt.Println(msg)
    }("going")

    time.Sleep(time.Second)
    fmt.Println("done")
}
```

```
$ go run goroutines.go
direct : 0
direct : 1
direct : 2
goroutine : 0
going
goroutine : 1
goroutine : 2
done
```

Synchronisation

- WaitGroup:

  Attendre la fin d'exécution de plusieurs goroutines

- Channel:

  Permettent la communication entre goroutines de manière synchrone ou asynchrone.

```go
package main

import "fmt"

func main() {

    queue := make(chan string, 2)
    queue <- "one"
    queue <- "two"
    close(queue)

    for elem := range queue {
        fmt.Println(elem)
    }
}
```

```
$ go run range-over-channels.go
one
two
```

Synchronisation

```go
package main

import (
    "fmt"
    "time"
)

func main() {

    c1 := make(chan string)
    c2 := make(chan string)

    go func() {
        time.Sleep(1 * time.Second)
        c1 <- "one"
    }()
    go func() {
        time.Sleep(2 * time.Second)
        c2 <- "two"
    }()

    for i := 0; i < 2; i++ {
        select {
        case msg1 := <-c1:
            fmt.Println("received", msg1)
        case msg2 := <-c2:
            fmt.Println("received", msg2)
        }
    }
}
```

```
$ time go run select.go
received one
received two

real    0m2.245s
```

Projet