Go : introduction

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Go

Go ?

- langage de programmation aussi appelé **Golang**
  - compilé et fortement typé
  - à typage statique avec inférence de type
  - orienté simplicité, lisibilité et productivité
  - supporte la programmation impérative, structurée et concurrente
  - sensible à la casse pour les mots-clés, variables, fonctions, types...
- créé chez **Google** par **Robert Griesemer**, **Rob Pike** et **Ken Thompson**
- annoncé publiquement en 2009, puis stabilisé avec **Go 1** en 2012

Go

Histoire de **Go**

- **2007 :** début de la conception de **Go** chez **Google**.
- **2009 :** annonce publique du langage et publication en open source.
- **2012 :** sortie de **Go 1**, première version stable avec une promesse de compatibilité.
- **2018 :** arrivée des **modules Go**, devenus progressivement la manière standard de gérer les dépendances.
- **2022 :** introduction des **génériques** avec **Go 1.18**.
- **Aujourd'hui :** Go est très utilisé pour le backend, le cloud, les outils CLI, les microservices et DevOps.

Go

Go : objectifs de conception

- **Simplicité** : peu de mots-clés, syntaxe volontairement compacte et lisible.
- **Compilation rapide** : cycle modification-compilation-exécution très court.
- **Concurrence intégrée** : goroutines et channels pour écrire des programmes concurrents.
- **Déploiement simple** : production d'exécutables natifs faciles à distribuer.
- **Outillage standardisé** : formatage, tests, documentation et gestion des modules intégrés à l'écosystème.
- ...

Go

Go, pourquoi ?

- Langage de haut niveau, doté de
  - ramasse-miettes : gestion automatique de la mémoire
  - système de packages et de modules intégré
  - bibliothèque standard riche : HTTP, JSON, fichiers, tests, concurrence...
- Très adapté aux programmes serveurs et aux applications réseau.
- Code généralement facile à relire grâce au formatage automatique avec `gofmt`.
- Permettant de développer des programmes :
  - performants et compilés en code natif
  - portables : Windows, macOS, Linux
  - faciles à déployer sous forme d'un exécutable
  - ...

Go

Go : multi-domaine

- **Applications Web et API** : avec la bibliothèque standard `net/http` ou des frameworks comme **Gin**, **Echo**, **Fiber**.
- **Microservices** : langage très présent dans les architectures cloud-native.
- **Outils CLI** : beaucoup d'outils DevOps sont écrits en Go.
- **Cloud et conteneurs** : écosystème très lié à Docker, Kubernetes et Terraform.
- **Réseau** : serveurs HTTP, proxies, clients réseau, services distribués.
- **Systèmes et infrastructure** : outils performants, simples à compiler et à déployer.
- ...

Go

`Hello world` **en Java**

```java
package org.eclipse.test;

public class Main {

  public static void main(String[] args) {
    System.out.println("Hello world");
  }

}
```

L'équivalent en Go

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello world")
}
```

Go

Quelques extensions de fichier utilisées par **Go**

- `.go` : extension standard pour les fichiers de code source **Go**.
- `go.mod` : fichier décrivant le module, son nom et la version minimale de Go utilisée.
- `go.sum` : fichier contenant les sommes de contrôle des dépendances téléchargées.
- `*_test.go` : fichiers contenant les tests unitaires et les benchmarks.
- `vendor/` : dossier optionnel contenant une copie locale des dépendances.
- ...

Quelques versions de **Go**

- **Go** 1.0 (sortie en mars 2012)
- **Go** 1.5 (sortie en août 2015) : compilateur et runtime écrits en Go
- **Go** 1.11 (sortie en août 2018) : modules Go expérimentaux
- **Go** 1.13 (sortie en septembre 2019) : modules et proxy de modules renforcés
- **Go** 1.17 (sortie en août 2021)
- **Go** 1.18 (sortie en mars 2022) : génériques
- **Go** 1.20 (sortie en février 2023)
- **Go** 1.22 (sortie en février 2024)
- **Go** 1.23 (sortie en août 2024)
- **Go** 1.24 (sortie en février 2025)
- **Go** 1.25 (sortie en août 2025)
- **Go** 1.26 (sortie en février 2026)

Introduction

Go : téléchargement

```
https://go.dev/dl/
```

Introduction

Go : téléchargement

`https://go.dev/dl/`

Go : installation

Après installation, vérifiez que le dossier `bin` de Go est bien accessible dans la variable de chemin **PATH**.

Introduction

Ou sous Windows avec la commande

```
winget install -e -id GoLang.Go
```

Introduction

Ou sous Windows avec la commande

```
winget install -e -id GoLang.Go
```

Pour vérifier la version de Go

```
go version
```

Introduction

Go Playground

`https://go.dev/play/`

Introduction

Go Playground

```
https://go.dev/play/
```

Tour of Go

```
https://go.dev/tour/
```

Go

Quel IDE (Environnement de développement intégré) ?

- **Visual Studio Code** (À ne pas confondre avec Visual Studio)
- GoLand
- LiteIDE
- Vim / Neovim
- Eclipse avec plugins adaptés
- ...

Go

Visual Studio Code : téléchargement

`code.visualstudio.com/download`

Go

Visual Studio Code : téléchargement

`code.visualstudio.com/download`

Visual **S**tudio **C**ode (ou **VSC**) , pourquoi ?

- Gratuit et open source
- Multi-langage (**Go**, **JavaScript**, **HTML**, **C++**...)
- Multi-système : **Windows**, **macOS**, **Linux**
- Dispose d'un terminal intégré, d'un contrôle de version **Git**, et d'un débogueur intégré
- Supporte les extensions pour personnaliser son environnement

Go

Quelques raccourcis **VSC**

- Pour activer la sauvegarde automatique : aller dans `File > AutoSave`
- Pour indenter son code : `Alt` `Shift` `f`
- Pour commenter/décommenter : `Ctrl` `:`
- Pour sélectionner toutes les occurrences : `Ctrl` `f2`
- Pour sélectionner l'occurrence suivante : `Ctrl` `d`
- Pour placer le curseur dans plusieurs endroits différents : `Alt`
- Pour ouvrir un terminal intégré : `Ctrl` `ù`
- Pour ajouter une nouvelle colonne : `Ctrl` `*`

Go

Quelques extensions **VSC** pour **Go**

- **Go** : extension officielle pour l'édition, l'auto-complétion, le formatage et les tests.
- **gopls** : serveur de langage utilisé pour l'IntelliSense et la navigation dans le code.
- **Code Runner** : pour lancer rapidement de petits exemples.
- **REST Client** : utile pour tester de petites API HTTP écrites en Go.
- **Docker** : pratique pour les projets backend et les microservices.

Go

Quelques outils de l'écosystème **Go**

- **go**
  - Commande principale : compilation, exécution, tests, modules, documentation...
- **gofmt**
  - Formate automatiquement le code Go selon un style standard.
- **go test** : exécute les tests et les benchmarks.
- **go mod** : initialise et gère les modules et les dépendances.
- **go doc** : consulte la documentation depuis la ligne de commande.
- **go vet** : détecte des erreurs ou incohérences probables dans le code.
- **Delve** : débogueur très utilisé dans l'écosystème Go.
- ...

Go

Conventions et philosophie de **Go**

- **gofmt** impose un formatage homogène du code.
- **Simplicité** : préférer le code explicite au code trop magique.
- **Composition** : on privilégie souvent la composition plutôt que l'héritage classique.
- **Gestion explicite des erreurs** : les erreurs sont généralement retournées et testées.
- **Interfaces implicites** : un type satisfait une interface sans déclaration explicite.
- **Documentation** : les commentaires des éléments exportés commencent généralement par leur nom.

Go

Les règles de nommage en **Go**

- Packages : noms courts, simples, en minuscules, sans underscore si possible.
- Variables et fonctions non exportées : **camelCase**.
- Types et fonctions exportés : **PascalCase**.
- Constantes : **camelCase** ou **PascalCase** selon la visibilité, pas forcément en majuscules.
- Acronymes courants : cohérence recommandée, par exemple `HTTPServer`, `userID`.
- Un identifiant commençant par une majuscule est **exporté** en dehors du package.

Go

Les instructions et le style en **Go**

- Le formatage est automatisé avec `gofmt`.
- Les blocs sont délimités par des accolades `{ ... }`.
- Le point-virgule existe techniquement, mais il est inséré automatiquement dans la plupart des cas.
- Une seule manière idiomatique de structurer beaucoup d'éléments du langage.
- Les imports non utilisés provoquent une erreur de compilation.
- Les variables locales déclarées et non utilisées provoquent aussi une erreur de compilation.

Go

Console et exécution en **Go**

- Go ne propose pas une console interactive standard équivalente à celle de Python.
- On écrit généralement le code dans un fichier `.go`, puis on l'exécute ou on le compile.
- Pour tester rapidement un code, on peut utiliser **Go Playground**.
- Pour un projet, on initialise généralement un module avec `go mod init`.

Introduction

Pour exécuter un fichier Go sans générer explicitement l'exécutable

```
go run main.go
```

Introduction

Pour exécuter un fichier Go sans générer explicitement l'exécutable

```
go run main.go
```

Pour compiler un programme Go

```
go build
```

Go

Remarques

- **Go** est sensible à la casse.
- Un programme exécutable doit généralement contenir un package `main` et une fonction `main()`.
- Les accolades sont obligatoires pour délimiter les blocs.
- Les erreurs doivent souvent être traitées explicitement avec `if err != nil`.
- Les imports et variables inutilisés sont refusés par le compilateur.

Go

Structure minimale d'un programme Go

- La première ligne indique le package du fichier.
- Les imports permettent d'utiliser des packages externes ou de la bibliothèque standard.
- La fonction `main()` est le point d'entrée d'un programme exécutable.

Go

Structure minimale d'un programme Go

- La première ligne indique le package du fichier.
- Les imports permettent d'utiliser des packages externes ou de la bibliothèque standard.
- La fonction `main()` est le point d'entrée d'un programme exécutable.

```
package main

import "fmt"

func main() {
    fmt.Println("Hello world")
}
```

Go

Fichier simple vs module Go

- Un fichier simple peut être exécuté directement avec `go run main.go`.
- Un projet réel est généralement organisé sous forme de **module**.
- Le fichier `go.mod` identifie le module et facilite la gestion des dépendances.
- La commande `go mod init exemple.com/monprojet` initialise un module.
- Les dépendances sont ajoutées automatiquement lors de l'utilisation de `go get` ou `go mod tidy`.

Go

Créer et lancer un petit projet Go

- Initialiser un module.
- Créer un fichier `main.go`.
- Exécuter le programme.

Go

Créer et lancer un petit projet Go

- Initialiser un module.
- Créer un fichier `main.go`.
- Exécuter le programme.

```
mkdir demo-go
cd demo-go
go mod init demo-go
go run main.go
```

Go

En utilisant Go, on peut effectuer des opérations arithmétiques

```
fmt.Println(3 + 2)
// 5
```

Go

En utilisant Go, on peut effectuer des opérations arithmétiques

```
fmt.Println(3 + 2)
// 5
```

Ou des tests logiques

```
fmt.Println(3 + 2 > 1 + 7)
// false
```

Go

À retenir pour démarrer avec **Go**

- Installer Go et vérifier avec `go version`.
- Utiliser un éditeur comme **Visual Studio Code** avec l'extension **Go**.
- Écrire un fichier `main.go` avec `package main` et `func main()`.
- Lancer avec `go run main.go` ou compiler avec `go build`.
- Formater régulièrement avec `gofmt`.