TD 1
Golang pour DevOps

1 Paramétrage de l'environnement

- Si vous avez Linux, vous pouvez installer golang à partir de votre gestionnaire de paquets. Assurez-vous simplement que la version disponible est supérieure à 1.12, sinon vous ne pourrez pas profiter des gomodules

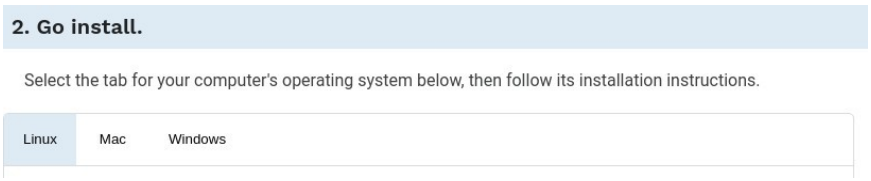
  Si la détection échoue et ne vous propose pas votre OS, rendez-vous sur cette page pour le séléctionner dans la liste des installateurs:

  ```
  https://golang.org/dl/
  ```

- Alternativement, suivez la procédure d'installation de Go ici:

  ```
  https://golang.org/doc/install
  ```

  Pour les instructions d'installation, le site présélectionne la version de votre OS en fonction de ce qu'il détecte, mais vous pouvez la changer en cliquant sur l'un de ces onglets:

- Comme suggéré dans les instructions, vérifiez votre installation en utilisant la commande:

  ```
  $ go version
  ```

2 Compte GitHub

Créez un compte sur GitHub (ou utilisez le vôtre) pour y mettre votre code.

3 Hello World

1. Reprenez l'exemple du cours, et écrivez votre propre "Hello, World!" en Go. Écrivez-le dans un fichier main.go, ou bien remplacez le nom du fichier dans la suite du TD.
2. Testez votre programme:
   - Soit en le compilant puis en le lançant :

     ```
     go build ./...
     ./main.go
     ```

   - Soit à la manière d'un "script":

     ```
     go run main.go
     ```

3. Vous pouvez "installer" votre application sur votre machine avec la com- mande:

   ```
   go install
   ```

   qui va placer votre binaire dans $GOPATH/bin. Une fois cela fait, vous pouvez exécuter votre programme depuis n'importe où.

4 Formater votre code

1. Testez le formateur de code "go fmt" en formatant ce code:

```
    package   main

import   "fmt"

func  main() {
 fmt.Printf(   "Bad ")      ;
      fmt.Println("formating"          )
      }
```

5 La librairie standard

Nous allons nous servir de quelques packages de la librairie standard, dont la documentation est disponible ici:

https://golang.org/pkg/

On va réaliser un petit programme qui compare des images et dit si elles sont identiques.

5.1 Flag

On va utiliser le package https://golang.org/pkg/flag/ pour permettre d'afficher la version de notre programme.

1.1. Définissez une constante dans le fichier main.go, comme ceci:

```
const VERSION = "1.0"
```

1.2. Utilisez flag.Bool() et flag.Parse() pour afficher la version du programme lorsque l'on entre:

```
$ go run main.go -version
```

5.2 Crypto, lecture de fichiers

2.1. Téléchargez les images disponibles sur l'espace de cours: image_1.jpg, image_2.jpg, image_3.jpg

2.2. Créez un projet img_diff, et initialisez-le avec:

```
$ go mod init img_diff
```

2.3. Écrivez une fonction qui permet de lire un fichier complètement, et d'en retourner les octets sous forme de []byte.

Utilisez le package https://golang.org/pkg/io/ioutil/

2.4. Écrivez une fonction qui permet de hasher un fichier donc vous lui passez le path en argument. Cette fonction retourne un "[]byte".

Utilisez le package https://golang.org/pkg/crypto/sha256/

2.5. Comparez les hashes des images, et déterminez celle qui est unique.

Affichez le résultat (le path de l'image unique tel que passé en argument) dans la console.

2.6. Pourriez-vous imaginer une implémentation plus efficace ?

2.7. Implémentez-la en utilisant https://golang.org/pkg/bufio/

5.3 Logs

On va ajouter du logging pour les erreurs. Pour cela, utilisez:
https://golang.org/pkg/log/

3.1. Créez un nouveau logger, et dirigez son output sur os.Stderr (https://golang.org/pkg/os/#pkg-variables)

3.2. Partagez votre logger dans votre programme, et loggez les erreurs éventuelles.

5.4 Depuis internet

On va ajouter une possibilité de récupérer une liste d'images sur internet et de les comparer.

4.1. Ajoutez un flag, comme pour la version, qui permet de passer une chaine de caractères en argument qui est une liste d'URLs séparées par des virgules:

```
"http://url1.tld,http://url2.tld"
```

4.2. Ajoutez un parsing de cette chaîne de caractères en utilisant: https://golang.org/pkg/strings/
Votre parser d'argument retourne un []string, qui est une slice d'URLs.

4.3. Utilisez http.Get() pour télécharger les images. Vous pouvez lire le contenu téléchargé avec:

```
data, err :=

ioutil.ReadAll(resp.Body) if err !=

nil {
    log.Fatal(err)
}

defer resp.Body.Close()
```

4.4. (Optionnel) Vous pouvez écrire les images sur le disque avec os.Create

4.5. Vérifiez le bon fonctionnement de votre programme.

5.5 Time

Ajoutez des logs qui permettent de savoir combien de temps mettent les images téléchargées à être récupérées depuis internet.
Utilisez: https://golang.org/pkg/time/
Et utilisez: https://golang.org/pkg/fmt/ pour les messages.
Formattez avec "%v"