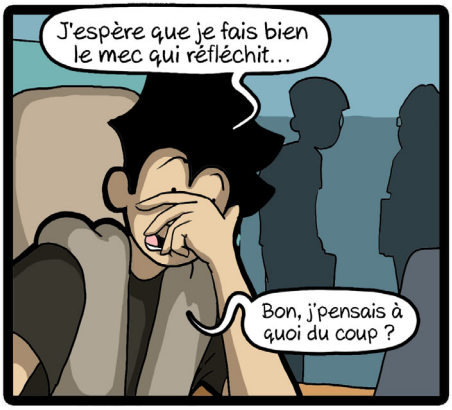
6

Un programme simple



6.1. Objectif de cette étude de cas

Pour cette première étape, notre objectif fonctionnel est une application simple qui calcule le résultat d'une élection. Dans un souci de facilité, il s'agira d'une élection en un seul tour, au suffrage universel direct, comme une élection de délégués de classe, par exemple.

Dans la fonction principale de notre programme (`main`), on instanciera nos données : les politiciens, les votants et chaque vote individuel. Puis, il nous faudra manipuler ces données afin d'obtenir le nombre total de votes reçus par chaque candidat et enfin en déduire le candidat ayant reçu le plus de votes, que l'on affiche à l'écran. Contrainte d'importance : en cas d'égalité, il ne serait pas acceptable d'y avoir deux gagnants.

Note *› Pour récupérer et expérimenter sur le code de cette étude de cas :*

- *Si vous souhaitez exécuter directement le code terminé, téléchargez-le dès maintenant au bon endroit dans votre répertoire GOPATH, à l'aide de la commande* `go get github.com/D-BookeR/LivreGo-sources`*. Vous pouvez itérer sur le code de cette étude de cas en le modifiant et en exécutant* `go run $GOPATH/src/github.com/D-BookeR/LivreGo-sources/etude_1/main.go` *(si vous avez choisi de ne pas définir la variable d'environnement* `$GOPATH` *lors de votre*

installation de Go, n'oubliez pas de la remplacer par le bon répertoire dans la commande ci-dessus, et les suivantes).

- *Plutôt que d'utiliser `go run` (qui compile et exécute à la volée) sur notre code, vous pouvez installer le résultat de manière plus permanente en exécutant `go install github.com/D-BookeR/LivreGo-sources/etude_1`. Un exécutable sans dépendance `etude_1` sera alors généré dans `$GOPATH/bin`. Il pourra être directement appelé en entrant `etude_1` de n'importe où dans votre terminal si vous avez bien mis `$GOPATH/bin` dans votre variable d'environnement `$PATH` en installant Go.*
- *Alternativement, si vous préférez suivre le tutoriel en recréant de zéro l'application dans votre propre compte GitHub, créez-y un nouveau projet `LivreGo-source`, puis créez un répertoire `$GOPATH/src/github.com/votre_username_github/LivreGo-source/etude_1` dans lequel vous mettrez votre version du code. Vous devrez certainement changer certains imports dans le code de ce livre pour vous assurer d'importer vos packages et non les nôtres.*
- *Si vous n'avez pas d'environnement Go installé et que vous ne souhaitez pas l'installer, vous pouvez expérimenter ce projet directement dans votre navigateur via le Go playground, puisqu'il tient en un seul fichier et ne dépend pas d'opérations d'entrée/sortie.*

6.2. Premières lignes

La première ligne d'un programme en Go est habituellement la déclaration du package dans lequel nous souhaitons déclarer ce code. `main` est un package spécial, car sa fonction `main()` sera le point d'entrée de notre script. Pour un script simple qui ne nécessite pas d'espaces de nom, on met généralement tout le code dans le package `main` :

```
package main
```

Typiquement, les quelques lignes suivantes sont dédiées à la déclaration des imports, qui sont les chemins des répertoires sous le `GOPATH` dont on va vouloir utiliser le code. Déclarer un import dans un fichier Go va permettre au code de ce fichier d'avoir visibilité sur les membres publics de cet import, et va aussi permettre au code de l'import d'être compilé avec le reste dans l'exécutable final. Si vous avez installé un éditeur de code et ses extensions Go, vous n'avez sans doute pas besoin d'expliciter ces imports. Ils s'ajouteront automatiquement lorsque vous commencerez à en utiliser les membres en question. Si votre code n'est pas automatiquement formaté (par exemple si vous utilisez le Go playground), vous devez ajouter ces imports à la main :

```
import(
  "fmt"
  "log"
  "errors"
)
```

Le package `"fmt"` permet de formater des chaînes de caractères. Le package `"log"` permet d'afficher des chaînes de caractères dans la sortie standard. Le package `"errors"` permet de déclarer des nouvelles erreurs.

Notez que si vous utilisez un formateur de code, et ajoutez ce code dès maintenant, il sera automatiquement supprimé car vous n'utilisez pas encore ces packages : il considérera donc que vous n'en avez pas besoin. L'approche que suivraient la plupart des développeurs Go consisterait à laisser votre formateur de code les ajouter plus tard automatiquement.

6.3. Structures de données

Avant toute chose, créons les types de données dont nous aurons besoin. Nous savons que nous avons besoin de politiciens et de votants. Disons qu'un votant est défini par son nom et son ID (par exemple, son numéro de carte électorale); et un politicien est défini par son nom, son ID et son parti. À l'image des classes dans la plupart des langages orientés objets, nous allons ici créer des types `struct`. La syntaxe est très similaire, mais souvenez-vous de leurs différences principales : elles sont stockées sur la pile d'exécution et non le tas, et elles ne permettent pas de hiérarchie de type (et donc pas de polymorphisme).

Note *› Pour plus d'informations sur comment les autres langages approchent les problématiques résolues en Go par les structures, référez-vous à la Section 2.4, Objets et gestion d'état. Et pour plus d'informations sur comment les utiliser, référez-vous à leur documentation officielle (en anglais).*

Pour créer un type en Go, la syntaxe est `type nomDuType definition`. Un exemple simple permettant de mieux comprendre consiste à dériver le type `int`, afin de lui attacher d'autres méthodes d'instance plus tard, comme ceci : `type age int`. Pour des structures de données, nous définirons le type de structure par le mot clé `struct` suivi des champs dont nous avons besoin, entre accolades :

```
type politician struct {
 Name  string
 ID    int
 Party string
}

type voter struct {
 Name string
 ID   int
}
```