Feuille d'exercices 3

Table des matières

Exercice 1

Dans les exercices, nous avons vu que nous devions souvent demander une saisie à l'utilisateur et que cette saisie devait s'accompagner d'un message pour lui indiquer ce qu'on attendait de lui. Généralement, on écrivait donc ces lignes en Go :

```
var nom string
fmt.Print("Bonjour, comment t'appelles-tu ? ")
fmt.Scan(&nom)
```

En Python, on dispose de la fonction input qui fait tout cela à la fois :

```
nom = input("Bonjour, comment t'appelles-tu ? ")
```

> L'appel input("un message") affiche un message à l'écran et attend que l'utilisateur saisisse quelque chose au clavier puis appuie sur la touche Entrée.
>
> Ce qui a été saisi est alors **renvoyé sous forme de chaîne** (dans notre exemple, la variable nom recevrait donc cette chaîne).

- Écrire cette fonction input en Go pour qu'elle se comporte exactement comme la fonction input de Python (on supposera qu'on ne saisit pas de chaîne avec des espaces…)
- Testez cette fonction à l'aide d'un programme principal qui demande à l'utilisateur de saisir son nom, puis qui affiche un message de bienvenue, du type Bonjour Toto (où Toto est le nom qui a été saisi par l'utilisateur).

> - C'est l'exception qui confirme la règle : **les sous-programmes ne doivent**

pas faire de saisies, ni afficher quoi que ce soit, sauf si l'on vous demande clairement d'écrire un sous-programme de saisie, comme ici…

ATTENTION :

- L'**affichage d'une valeur** avec un Print (ou autre sous-programme d'affichage) n'est pas du tout la même chose que **renvoyer une valeur** !!!
- En programmation comme en Mathématiques, les fonctions doivent renvoyer des valeurs qui sont ensuite récupérées par le code qui les appelle. Ce "transfert" de la fonction vers le code appelant ne s'effectue que par un appel à return.
- Ce code appelant peut ensuite, s'il le souhaite, afficher la valeur renvoyée par la fonction, la tester, l'utiliser dans un calcul, etc.
- Ce n'est pas à la fonction de décider comment sera utilisé son résultat : c'est au code qui appelle la fonction.
- Certains sous-programmes ne renvoient rien : on ne les appelle donc pas "fonctions", mais "procédures". Leur utilisation doit être exceptionnelle et peut-être considérée comme désuette.

Exercice 2

La fonction input que vous venez d'écrire renvoie **toujours une chaîne**.

En vous aidant de ce qui a été présenté en cours, modifiez votre programme principal (**sans toucher à la fonction** input **que vous venez d'écrire**) pour, qu'en plus du nom, il demande la saisie de l'âge (un entier) et de la taille (un réel).

Votre programme devrait maintenant pouvoir afficher un message du type Bonjour Toto, tu as 20 ans et tu mesures 1.85m.

Pour l'instant, ignorez les erreurs éventuelles de conversion

Exercice 3

Reprenez votre programme principal (sans toucher à la fonction input) pour, cette fois-ci, tenir compte des erreurs de saisie éventuelles (saisie de "titi" pour l'âge ou pour la taille, par exemple).

Réfléchissez au comportement que devrait avoir votre programme dans de tels cas et faites en sorte qu'il l'ait.

> Aidez vous du programme d'exemple

Exercice 4

Les **diviseurs propres** d'un nombre entier sont tous les diviseurs de ce nombre, sauf lui-même. Par exemple, les diviseurs propres de 8 sont 1, 2 et 4. Le seul diviseur propre de 7 est 1.

> 1 fait toujours partie des diviseurs propres.

On veut calculer la somme des diviseurs propres d'un nombre entier (qu'on supposera supérieur ou égal à 2). Pour ce faire, vous devez écrire la fonction sommeDivPropres qui, étant donné un nombre, renvoie la somme de ses diviseurs propres.

Le programme principal demandera à l'utilisateur de saisir un nombre entier supérieur ou égal à 2 (en utilisant la fonction input écrite précédemment) et affichera la somme de ses diviseurs propres.

> Pour trouver les diviseurs propres d'un nombre N, le plus rapide consiste à tester chaque nombre de 2 à la racine carrée de N. Si ce nombre (appelons-le div) est un diviseur de N alors il faut l'ajouter au résultat, **ainsi que N/div**). On n'oubliera pas d'ajouter 1 au résultat puisque 1 fait toujours partie des diviseurs propres.
>
> Si N vaut 26, on parcourera tous les nombres de 2 à 5 (qui est l'entier le plus proche par défaut de la racine carrée de 26). Le résultat final sera donc la somme de 1 et de 2 (et donc de 13). Ce qui donnera un résultat final de 16.
>
> Pour obtenir la racine carrée d'un nombre en Go, vous disposez de la fonction math.Sqrt (il faut donc importer le module math).
>
> La lecture de sa documentation (avec la commande go doc math.Sqrt) vous montrera que cette fonction attend un réel en paramètre et renvoie un réel :

vous devrez donc effectuer des conversions entre entiers et réels.

Exercice 5

Un **nombre parfait** est égal à la somme de ses diviseurs propres : par exemple, 6 est un nombre parfait.

Écrire un programme demandant une limite >= 2 à l'utilisateur, puis affichant tous les nombres parfaits inférieurs à cette limite.

Voici un exemple d'exécution de ce programme :

```
Entrez une limite >= 2 : 1000
6 28 496
```

Last updated 2023-01-31 13:14:45 +0100