HE VD
IG

Paradigmes et Langages de Programmation

Haute École d'Ingénierie et de Gestion du Canton de Vaud

3. Haskell / Fonctions d'ordre supérieur

2022

Exercice 1

Montrer comment la liste en compréhension *[f x | x <- xs, p x]* peut être exprimée à partir des fonctions d'ordre supérieur *map* et *filter*.

Exercice 2

Définir la fonction *check* qui permet de construire la liste :

check f [(x_1*,* y_1*) ... (*x_n*,* y_n*)]-> [(*x_1*,* y_1*,* $f\,x_1\,y_1$*) ... (*x_n*,* y_n*,* $f\,x_n\,y_n$*)].*

Exercice 3

Écrire une fonction *fmap* prenant une liste de fonctions et une valeur, et qui applique chaque fonction à la valeur passée comme paramètre :

```
Prelude> fmap [succ, pred] 2
[3, 1]
```

Exercice 4

Écrire une fonction qui transforme une liste d'éléments en liste de listes :

```
Prelude> listes [1, 2, 3]
[[1], [2], [3]]
```

Pour la fonction précédente, vous avez probablement défini une fonction anonyme qui transforme un élément en liste. Instancier partiellement *(:)* pour définir *listes* sans cette fonction anonyme.

Exercice 5

Écrire une fonction qui calcule la liste des carrés d'une liste de nombres :

carres [x_1*,* x_2 *...* x_n*]->* $[x_1^2, x_2^2 \ldots x_n^2]$

Exercice 6

Écrire la fonction précédente sans fonction auxiliaire (utiliser l'opérateur qui élève à la puissance et *flip*).

Exercice 7

Écrire une fonction *positiveAttitude* qui prend la valeur absolue des nombres d'une liste :

```
Prelude> positiveAttitude [-1,2,5,-3,0]
[1,2,5,3,0]
```

Exercice 8

Écrire une fonction qui prend une liste de chaînes de caractères et ne retient que les 5 premiers caractères de chaque chaîne.

Exercice 9

Écrire une fonction qui transforme tous les caractères d'une chaîne en minuscules, sauf le premier :

```
Prelude> minuscules "Ceci N'est Pas Un Titre."
"Ceci n'est pas un titre."
```

Exercice 10

Écrire une fonction qui filtre les éléments pairs ou impairs d'une liste.

Exercice 11

Écrire une fonction qui filtre les éléments d'une liste qui sont supérieurs ou inférieurs à une valeur donnée.

Exercice 12

Écrire une fonction qui cherche le premier élément dans une liste qui satisfait un prédicat :

```
Prelude> cherche (>3) [1,2,3,4,5,6]
4
```

Exercice 13

Utiliser un pliage pour écrire la fonction *existe* qui correspond au quantificateur existentiel :

```
existe :: ... => [t] -> (t -> Bool) -> Bool
```

Cette fonction retourne *True* si au moins un élément de la liste satisfait le prédicat.

Exercice 14

Utiliser un pliage pour écrire la fonction *tous* qui correspond au quantificateur universel :

```
tous :: ... => [t] -> (t -> Bool) -> Bool
```

Cette fonction retourne *True* si tous les éléments de la liste satisfont le prédicat.

Exercice 15

Donner une version alternative des fonctions des deux derniers exercices sans définir de fonction auxiliaire pour le pliage, en utilisant *map* pour transformer la liste *[t]* en *[Bool]*.

Exercice 16

Une chaîne de Collatz est la liste de nombres $[x_i, x_i + 1, x_i + 2, \dots, 1]$ tels que :

- $x_i + 1 = x_i * 3 + 1$ si x_i impair
- $x_i + 1 = x_i/2$ si x_i pair

Exemple : La chaîne de Collatz de 3 est : [3, 10, 5, 16, 8, 4, 2, 1].

Écrire une fonction pour calculer ces chaînes. Quelle est la longueur de la chaîne de Collatz de 156159 ?

Exercice 17

La mémoïsation consiste à optimiser le calcul d'une fonction récursive à l'aide d'un tableau qui mémorise les valeurs calculées précédemment.

Écrire une fonction qui permet de calculer ces valeurs avec une fonction récursive. Appliquer la mémoïsation au calcul de la série de Fibonacci. Comparer les performances de ces deux solutions.

Challenge : Appliquer le principe de la mémoïsation au calcul de la fonction d'Ackermann.

Bon travail !