Université d'Orléans	L2 Informatique
Programmation fonctionnelle	2022-2023

TD n°3

Exercice 1

Sans l'utiliser, donner la réponse que donnerait la boucle interactive pour chacune des phrases suivantes, en supposant que celles-ci sont évaluées à la suite. Préciser à chaque fois s'il s'agit d'une expression ou d'une définition.

```
let f = fun x -> if x <= 0 then 0 else 1 + f(x - 1);;
let rec f = fun x -> if x >= 0 then 1 + f(x - 1) else 1 + f(x + 1);;
f 5;;
let rec g = fun x -> g (x - 1);;
g 2;;
let rec h = fun x -> if x <= 0 then 1 else 2 * (h (x-1)) in h 4;;
h 8;;
let rec z = fun l -> if List.length l = 1 then List.hd l
                     else z(List.tl l);;
z ['a'; 'b'; 'c'];;
z [ [1;2;3]; [4;5; 6]; [7] ];;
z [ cos; sin; tan ];;
z [];;
```

Exercice 2

Question 1

Écrire une fonction qui prend en paramètre un entier positif *n*, et qui renvoie la somme des entiers de *0* jusqu'à *n*. Si le paramètre est négatif, la fonction renvoie *0*.

```
# sum (-1);;
- : int = 0
# sum 3;;
- : int = 6
# sum 0;;
- : int = 0
```

Question 2

Écrire une fonction qui prend en paramètre un couple de deux entiers *n* et *m* et qui calcule la somme des entiers entre *n* et *m* inclus. Si *n* est strictement plus grand que *m*, cette fonction doit renvoyer *0*.

```
# sum_from_to (-1, 2);;
- : int = 2
# sum_from_to (1, 3);;
- : int = 6
# sum_from_to (10, 1);;
- : int = 0
```

Exerice 3

Question 1

Écrire une fonction `power` qui prend un couple de deux entiers en arguments *x* et *n*, le second étant supposé positif, et retourne la valeur x^n.

Combien de fois la fonction `power` est-elle appelée *récursivement* (c'est-à-dire en excluant le premier appel) lorsque son second argument est *n* ?

```
# power (2, 0);;
- : int = 1
# power (2, 1);;
- : int = 2
# power (2, 10);;
- : int = 1024
```

Question 2

L'objectif est d'avoir une fonction `power` plus rapide que la fonction précédente en diminuant le nombre d'appels récursifs.

Écrire une nouvelle fonction `power` qui prend un couple de deux entiers en arguments x et n, le second étant supposé positif, et retourne la valeur x^n, mais en prenant en compte le fait que :

- si n est pair, alors $x^n = (x^{\frac{n}{2}})^2$,
- si n est impair alors $x^n = x \times (x^{\frac{n-1}{2}})^2$.

Question optionnelle : Combien de fois la fonction `power` est-elle appelée *récursivement* (c'est-à-dire en excluant le premier appel) lorsque son second argument est n ?

Exercice 4

Ecrire une fonction `repeat` qui qui prend en paramètre un triplet constitué d'un entier `count` et deux chaînes de caractères et qui produit une chaîne qui contient `count` répétitions de la première chaîne, chaque répétition étant séparée de la chaîne précédente par la seconde chaîne.

Si le nombre de répétitions est négatif, la chaîne renvoyée est vide.

```
# repeat ((-1),"Hello", ", ");;
- : string = ""
# repeat (1, "Hello", ", ");;
- : string = "Hello"
# repeat (2, "Hello", ", ");;
- : string = "Hello, Hello"
```

Exercice 5 (Bonus)

Question 1

Écrire une fonction `flatten` ayant le type `'a list list -> 'a list`. Cette fonction doit transformer une liste de listes en liste.

Par exemple :

- `flatten []` donne `[]`
- `flatten [ [] ]` donne `[]`
- `flatten [ [1;2]; [3;4]; []; [5;6] ]` donne `[1; 2; 3; 4; 5; 6]`
- `flatten [ ["a"]; ["b";"c"]; ["d";"e";"f"] ]` donne `["a";"b";"c";"d";"e";"f"]`

Question 2

Donner le type, puis écrire, une fonction `tester` qui applique toutes les fonctions d'une liste de fonctions à toutes les valeurs d'une liste de valeurs et renvoie le résultat sous forme d'une liste.

Par exemple, `tester([abs;fun x->x+1],[-1;0;1])` renvoie `[ 1;0;1;0;1;2 ]`.