TP n°8 – Fonctions en OCaml

1 Définition de fonctions

Exercice 1

1. Définir la fonction `identite : 'a -> 'a` de deux manières différentes : en utilisant le mot-clé `fun` et en utilisant la syntaxe alternative plus simple.
2. Définir, toujours de deux façons différentes, la fonction `successeur : int -> int` qui prend en entier n et s'évalue en l'entier suivant. Vérifier sur plusieurs exemples.
3. Définir, toujours de deux façons différentes, la fonction `ajouter : int -> int -> int` qui réalise la somme de ses deux arguments. Vérifier sur plusieurs exemples.

Exercice 2

1. Définir la fonction $p(x) = 4x^3 - x^2 + 5x - 8$
2. Définir $f(x) = 3 * \ln(x) - 2x$
3. Définir $g(x) = e^{2x} - e^{-x} + 2\sin(x)$

Exercice 3

En définissant $f : x \mapsto \sin(\ln(x))$ comme une fonction locale, implémenter la fonction : $g : x \mapsto \dfrac{f(x)}{1 + (f(x+1))^2}$

Indice (compléter les ...) :

```
let g x =
   let f x = ...
   in  ...;;
```

2 Fonctions récursives

Le mot-clé `#trace` permet de « tracer » le déroulement d'une fonction et est très utile pour visualiser les appels récursifs. Tester les lignes suivantes :

OCaml

```
let rec factorielle n =
  if n = 0 then
    1
  else
    n * factorielle (n - 1)
;;
#trace factorielle;;
factorielle 10;;
#untrace factorielle;;
factorielle 50;;
```

On pourra utiliser la fonction `trace` dans les exercices suivants afin de bien comprendre le comportement des fonctions récursives.

Exercice 7

Écrire une fonction `mult : int -> int -> int` qui calcule le produit des arguments sans utiliser le symbole « `*` ». On rappelle que $a \times b = \sum_{i=0}^{b-1} a$.

Exercice 8 Écrire une fonction `somme_cube : int -> int` qui à n associe $\sum_{k=0}^{n} k^3$.

Exercice 9 Pour certains arguments, une fonction peut ne pas être définie. On peut le signaler en OCaml à l'aide de l'expression `failwith "un message"`, qui va déclencher une exception (ici `Failure "un message"`) et interrompre le programme.

Par exemple si on écrit un code dans lequel a ne doit pas être plus petit que b on écrira :

```
if a<b then failwith "a trop petit"
else ... (*On fait des trucs*);;
```

1. Écrire deux fonctions récursives **quotient** et **reste** qui, à partir de deux entiers positifs a et b, renvoient respectivement le quotient et le reste de la division de a par b, sans utiliser, bien sûr, l'opérateur de division ou l'opérateur **mod**.
2. Écrire une fonction **divmod** qui calcule les deux à la fois et renvoie le couple **(quotient, reste)**.
   Bonus : Écrire une version de **divmod** dans laquelle a et b peuvent être négatifs.

Exercice 10 Moyenne arithmético-géométrique

Pour définir des fonctions mutuellement récursives **f1** et **f2** en Ocaml, on utilise la syntaxe suivante :
```
let rec f1 n = ... and f2 n = ... ;; (*and permet de faire deux liaisons a la fois*)
```

Soit $a, b \in \mathbb{R}_+$, on définit deux suites positives $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$, de premiers termes $u_0 = a$ et $v_0 = b$ et satisfaisant les relations de récurrence (elles sont mutuellement récurrentes) :

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \sqrt{u_n v_n} \quad v_{n+1} = \frac{u_n + vn}{2}$$

1. Définir des fonctions mutuellements récursives **u** et **v** prenant en entrée n et calculant le terme n des suites.
2. Pour $a = 1$ et $b = 2$, calculer u_{10} et v_{10}. A-t-on **(u 10) = (v 10)** ? Expliquer.
3. Écrire une fonction **termes** telle que **termes a b n** renvoie le couple des n-ième termes (u_n, v_n).
4. Calculer u_{10} et v_{10} pour $(a, b) \in \{(1, 1000), (2, 487), (500, 501)\}$.

3 Prendre en entrée et renvoyer des fonctions

En Ocaml on peut prendre en entrée des fonctions et renvoyer des fonctions.
Par exemple la fonction f suivante prend en entrée une constante c et renvoie la fonction $x \mapsto x * c$. Son type est **int -> int -> int**.

```
let f c = fun x -> x*c;;
```

la fonction suivante prend en entrée une fonction f et un nombre x et renvoie $f(2x) + 1$. Son type est **(int -> int) -> int -> int**.

```
let applique f x = (f (2*x))+1;;
```

Exercice 11
Prévoir le type et le retour des fonctions OCaml suivantes. Vérifier à l'aide de OCaml .

OCaml
```
let fonction1 () = fun f -> f 0;;
let fonction2 f = f 0;;
let fonction3 f g = fun x -> (f x) + (g x);;
let fonction4 f g x = (f x) + (g x);;
```

Exercice 12
Proposer une implémentation en OCaml de chacune des fonctions suivantes. Vous pouvez imiter l'exercice précédent) :

- qui à $f : \mathbb{R} \to \mathbb{R}$ associe $\dfrac{f(0) + f(1)}{2}$;
- qui à $f : \mathbb{R} \to \mathbb{R}$ et $x \in \mathbb{R}$ associe $(f(x))^2$;
- qui à $f : \mathbb{R} \to \mathbb{R}$ associe la fonction f^2 ;
- qui à $f : \mathbb{R} \to \mathbb{R}$ associe la fonction $f \circ f$;
- qui à $f : \mathbb{R} \to \mathbb{R}$ associe la fonction $x \mapsto f(x + 1)$ de $\mathbb{R} \to \mathbb{R}$.

Exercice 13 Écrire une fonction **somme_termes** qui, à une fonction $f : \mathbb{N} \to \mathbb{N}$ et à un entier n, associe $\sum_{k=0}^{n} f(k)$.

En utilisant la fonction **somme_termes**, calculer la somme des cubes des entiers naturels inférieurs ou égaux à 100.

4 Un peu d'arithmétique (Exercices plus avancés)

Exercice 14 Écrire une fonction `nb_chiffres : int -> int` qui calcule le nombre de chiffres dans un nombre.

Astuce : diviser par 10.

Exercice 15 Écrire une fonction `nb_div: int -> int` qui calcule le nombre de diviseurs d'un entier x.

Astuce : On peut faire avec une fonction auxiliaire si nécessaire. Imaginez comment vous feriez avec une boucle et transposez.

Exercice 15 Écrire une fonction `est_premier: int -> bool` qui détermine si un entier x est premier ou non.

Exercice 16 Écrire une fonction `pgcd: int -> int -> int` qui calcule le PGCD de 2 nombres.

Astuce : l'algorithme d'Euclide est naturellement un algorithme récursif

Exercice 17 Écrire une fonction `est_parfait : int -> bool` qui détermine si un entier x est parfait, c'est à dire qu'il est égal à la somme de ses diviseurs (en excluant x lui-même).

Astuce : C'est comme `nb_div` sauf qu'il faut garder en mémoire la somme des diviseurs.