Chapitre 3

Listes et fonctions récursives

1 Listes.

Définition simple de listes :

```
 let l = [ 5;2;1;6];;
val l : int list = [5; 2; 1; 6]
```

Ajout d'un élément en tête d'une liste :

```
let l_2 = 7::l;;
val l_2 : int list = [7; 5; 2; 1; 6]
```

: : est un opérateur de construction de listes. Il faut que l'élément rajouté en tête de la liste ait le même type que les autres éléments de la liste.

Liste vide :

```
[];;
- : 'a list = []
```

La liste vide a un type "polymorphe" : on peut l'utiliser avec plusieurs types différents :

```
let lis2 = 5::[];;
val lis2 : int list = [5]
let lis3 = true::[];;
val lis3 : bool list = [true]
```

Quelques fonctions utiles pour les listes :

- La fonction `hd`, pour l'accès au premier élément d'une liste :

```
List.hd;;
- : 'a list -> 'a = <fun>
let lis = [8;4;3;2];;
val lis : int list = [8; 4; 3; 2]
List.hd(lis);;
- : int = 8
List.hd([]);;
Exception: Failure "hd".
```

- La fonction `tl`, pour l'accès au reste de la liste :

```
List.tl;;
- : 'a list -> 'a list = <fun>
List.tl(lis);;
- : int list = [4; 3; 2]
List.tl([]);;
Exception: Failure "tl".
```

- La fonction `length`, pour l'accès à la taille de la liste :

```
List.length;;
- : 'a list -> int = <fun>
List.length(lis);;
- : int = 4
List.length([]);;
- : int = 0
```

- La fonction **append**, pour la concaténation de deux listes :

```
List.append;;
- : 'a list -> 'a list -> 'a list = <fun>
let lis_1 = [8;4;1;3];;
val lis_1 : int list = [8; 4; 1; 3]
let lis_2 = [7;5];;
val lis_2 : int list = [7; 5]
List.append lis_1 lis_2;;
- : int list = [8; 4; 1; 3; 7; 5]
```

- La fonction `map`, pour la appliquer une fonction aux éléments d'une liste :

```
List.map;;
- : ('a -> 'b) -> 'a list -> 'b list = <fun>
let lis_1 = [8;4;1;3];;
val lis_1 : int list = [8; 4; 1; 3]
List.map (fun x-> 2*x+1) lis_1;;
- : int list = [17; 9; 3; 7]
```

Exercice 1 : Écrire la fonction distance_tete qui prend en argument une liste `l` d'entiers et renvoie une liste composé de l'élément en tête de `l`, suivi des distances entre cet élément et le reste de la liste l :

```
val distance_tete : int list -> int list = <fun>
```

Exemple :

```
distance_tete [ 5;7;2;15;9 ;-3];;
- : int list = [5; 2; 3; 10; 4; 8]
```

2 Définition de fonctions récursives.

Définition 1 :
Une fonction récursive est une fonction qui peut s'utiliser elle-même dans le calcul.

Exemple : Définition de la fonction factorielle :

```
let rec fact n = if n=0  then 1 else n*fact(n-1);;
val fact : int -> int = <fun>
```

Le mot clé rec indique à Ocaml que l'identifiant fact employé dans la définition de la fonction correspond à la fonction que l'on définit.

Propriété 1 :
Syntaxe de la définition d'une fonction récursive :

let rec fonction id1 ... idn = expr ;;
ou
let rec fonction = fun id1 ... idn -> expr ;;

Exercice 2 : On propose la récursive suivante :

```
let rec ma_fonction n_1 n_2 =
    if n_1 > n_2 then []
    else n_1::( ma_fonction (n_1+1) n_2 );;
```

Quelle est la signature de la fonction et que renvoie-t-elle avec les arguments $n_1 = 5$ et $n_2 = 9$?