Terminez le TP2 sur la correction partielle du DS numéro 1 en utilisant OCaml.

Listes et tableaux : structures de données de bases en OCaml

Comme vu en cours mardi, les deux structures de données de base en OCaml sont les listes et les tableaux. Elles sont l'implémentation concrètes de structures de données abstraites que sont les listes simplement chaînées et les tableaux.

— Les listes s'écrivent `[x1; x2; ...; xn]` (avec n valeurs), ou `tete :: queue` (avec `tete` une valeur et `queue` une liste), et elles sont de types homogènes (les valeurs ont le même type). Le module `List` contient toutes les fonctions utiles pour les listes, ex. `List.hd` renvoie la tête d'une liste et `List.tl` renvoie la queue de la liste. En OCaml, généralement on parcourt la liste dans une fonction *récursive*, et sans utiliser `List.nth li n` qui donne le `n`-ième élément en temps linéaire en `nn` (car il faut reparcourir toute la liste chaînée). La documentation est en ligne sur `https://ocaml.org/api/List.html`.

— Les tableaux s'écrivent `[|x1; x2; ...; xn|]` et sont aussi de types homogènes. Le module `Array` contient toutes les fonctions utiles opérant sur les tableaux, comme par exemple `Array.length` qui renvoie la longueur d'un tableau. En général on utilise les tableaux dans une fonction *impérative*, quand on sait que l'on va devoir accéder souvent à des cases quelconque du tableau et modifier leur contenu. C'est typiquement le cas des algorithmes de tri. La documentation est sur `https://ocaml.org/api/Array.html`.

Réimplémentation des fonctions sur les listes

Pour chacune des fonctions suivantes qui sont dans le module `List`, écrivez votre propre version en suivant la spécification, c'est à dire son typage et sa documentation) :

— `val nth : 'a list -> int -> 'a` : *Return the n-th element of the given list. The first element (head of the list) is at position 0.*

— `val rev : 'a list -> 'a list` : *List reversal.*

— `val append : 'a list -> 'a list -> 'a list` : *Concatenate two lists.*

— `val iter : ('a -> unit) -> 'a list -> unit` : *iter f [a1 ; ... ; an] applies function f in turn to a1 ; ... ; an. It is equivalent to begin f a1 ; f a2 ; ... ; f an ; () end.*

Implémentation de quelques fonctions sur les tableaux

Pour chacune des fonctions suivantes (qui elles, ne sont pas dans le module `Array`), écrivez votre propre version en suivant la spécification (typage et documentation) :

— `val max : 'a array -> 'a` : le maximum du tableau, supposé non vide ;

— `val moyenne_arithmetique : float array -> float` : la moyenne arithmétique du tableau, supposé non vide (ie. $1/n * \sum_{i=0}^{n-1} \text{tab}[i]$) ;

— `val moyenne_geometrique : float array -> float` : la moyenne géométrique du tableau, supposé non vide (ie. $\prod_{i=0}^{n-1} \text{tab}[i]^{1/n}$) ;

— `val moyenne_harmonique : float array -> float` : la moyenne harmonique du tableau, supposé non vide et avec des valeurs non nulles (ie. $n/(\sum_{i=0}^{n-1}(1/\text{tab}[i]))$).

