L1 MISPIC
Programmation fonctionnelle
TP 4

Avant de commencer le TP, créez un répertoire `TP4` dans votre arborescence consacrée à OCaml. Allez dans ce répertoire et ouvrez un fichier `tp4.ml` avec *emacs*. Vous taperez vos fonctions dans ce fichier, et les chargerez dans l'interpréteur OCaml avec `#use "tp4.ml";;` Si une fonction est erronée, corrigez-la dans *emacs*, puis rechargez le fichier dans l'interpréteur. Ceci vous permettra de pouvoir sauvegarder votre travail dans le fichier `tp4.ml`. Gardez à l'esprit que même si la définition d'une fonction ne fait pas d'erreur dans l'interpréteur, cela ne signifie pas pour autant qu'elle calcule ce que vous souhaitez. Vous devez pour vous en assurer la tester en l'appelant avec différentes valeurs de paramètres pour être sûr que ce qu'elle vous renvoie est correct.

Notes de musique

Des étudiants mélomanes et fans d'Ocaml, ça existe ! Prenons l'exemple de Joséphine. Très enthousiaste, elle a récemment décidé de combiner ses goûts pour la musique et pour la programmation fonctionnelle, en développant un ensemble de fonctions lui permettant de travailler sur des partitions.

Pour représenter les notes de musique, elle a choisi d'utiliser les entiers : ainsi, 0 désigne le *do*, 1 désigne le *ré*, ..., et 6 désigne le *si*. Elle décide également qu'une partition de musique sera une liste de notes, c'est-à-dire une liste d'entiers. Par exemple, la liste `[0;1;2;0;0;1;2;0;2;3;4]` est le début de Frère Jacques.

1. Écrire une fonction `nb_notes : 'a list -> int` qui prend une partition et retourne le nombre de notes qu'elle contient. Faire 2 versions : une sans accumulateur, et une avec.
2. Écrire une fonction `nb_occurrences : 'a -> 'a list -> int` qui prend une note et une partition, et qui retourne le nombre de fois où la note apparaît dans la partition.
3. Écrire une fonction `la_plus_aigue : 'a list -> 'a` qui prend une partition et renvoie la note la plus aiguë qui s'y trouve.
4. Écrire une fonction `indice_plus_aigue : 'a list -> int` qui prend une partition et renvoie l'indice de la note la plus aiguë (ou de la première occurrence de la note si elle apparaît plusieurs fois). On rappelle que les indices des listes commencent à 0.
5. La tonalité d'un air de musique est (souvent) donnée par la dernière note de sa partition. Écrire une fonction `tonalite : 'a list -> 'a` qui prend une partition et retourne sa tonalité (i.e. sa dernière note).
6. Joséphine trouve que ses partitions sont parfois trop rapides. Pour les ralentir, elle décide de jouer chacune des notes deux fois. Autrement dit, pour Frère Jacques, elle voudrait en fait jouer la partition

`[0;0;1;1;2;2;0;0;0;0;1;1;2;2;0;0;2;2;3;3;4;4]`. Écrire une fonction `ralentir : 'a list -> 'a list` qui retourne la partition dans laquelle toutes les notes ont été doublées.

7. Forte de son expérience, Joséphine se dit qu'elle pourrait aussi accélérer une partition en ne jouant qu'une note sur deux. Écrire une fonction `accelerer : 'a list -> 'a list` qui retourne la partition correspondante. Vérifiez que votre fonction marche correctement, que la partition d'origine ait un nombre pair ou impair de notes.
8. La fonction d'accélération précédente ne donne pas les résultats escomptés, car trop de notes sont supprimées pour que la musique d'origine soit reconnaissable. Néanmoins, par chance, elle permet d'obtenir de la musique contemporaine tout-à-fait intéressante. Joséphine décide donc de faire une nouvelle expérience en inversant deux à deux les notes d'une partition. Écrire une fonction `inverser_2a2 : 'a list -> 'a list` qui réalise cette opération. Assurez-vous que votre fonctionne marche correctement, avec une partition ayant un nombre pair ou impair de notes.
9. Prix de Rome pour ses résultats en théorie de la musique pressée et de la musique inversée, Joséphine devient grâce à OCaml une compositrice de renommée internationale. Mais elle n'a de cesse de vouloir poursuivre ses travaux. Ainsi, elle souhaite désormais adapter sa sonate pour vibraphone pakistanais en une sonate pour cornemuse préparée. Et pour cela, il lui faut transcrire sa partition, en baissant chaque note de deux notes. Cela veut dire qu'un *fa* doit devenir *ré*, un *mi* doit devenir *do*, un *ré* doit devenir *si*, un *do* doit devenir *la*, etc. Sur Frère Jacques, cela donne `[5;6;0;5;5;6;0;5;0;1;2]`. Écrire une fonction `transcrire : int list -> int list` qui permet de faire cette transcription de deux notes.
10. Poursuivant ses activités de recherche en musicologie, Joséphine souhaite savoir si on peut extraire une gamme d'une partition, i.e. si, en effaçant certaines notes de la partition, on peut obtenir la liste `[0;1;2;3;4;5;6]`. Écrire une fonction `gamme : int list -> bool` qui retourne vrai si on peut extraire une gamme de la partition, et faux sinon.
11. Joséphine invente le concept de signature musicale. Pour ce faire, elle cherche à écrire une fonction `signer` qui prend en paramètre une chaîne de caractères, et renvoie une partition dans laquelle la lettre *a* a été codée par un *do* (soit l'entier 0), la lettre *b* par un *ré* (soit l'entier 1), ..., la lettre *g* par un *si* (soit l'entier 6), la lettre *h* par un *do* (soit l'entier 0), etc. Par exemple, `signer "josephine"` renvoie la partition `[2; 0; 4; 4; 1; 0; 1; 6; 4]`. Écrire la fonction `signer : string -> int list` correspondante. On ne considérera que les minuscules, hors accents ou caractères particuliers.

    Remarque :

    ```
    # int_of_char 'a';;
    - : int = 97
    # int_of_char 'j';;
    - : int = 106
    ```
12. Améliorer la fonction précédente pour prendre en compte les majuscules. Ainsi, `signer "Josephine"` renvoie la même chose que précédemment.