L1 MISPIC
Programmation fonctionnelle
TP 3

Avant de commencer le TP, créez un répertoire `TP3` dans votre arborescence consacrée à OCaml. Allez dans ce répertoire et ouvrez un fichier `tp3.ml` avec *emacs*. Vous taperez vos fonctions dans ce fichier, et les chargerez dans l'interpréteur OCaml avec `#use "tp3.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 `tp3.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.

1 Pour commencer

1. Écrire une fonction `existe_g_d : char -> string -> bool` qui renvoie vrai si le caractère apparaît dans la chaîne, et faux sinon. Le parcours de la chaîne se fera de gauche à droite (du début vers la fin).
2. Écrire une fonction `existe_d_g : char -> string -> bool` qui fait la même vérification que la fonction précédente, mais cette fois-ci le parcours de la chaîne se fera de droite à gauche (de la fin vers le début).
3. Écrire une fonction `nb_occurences_g_d : char -> string -> int` qui compte combien de fois le caractère apparaît dans la chaîne, sans utiliser d'accumulateur. Le parcours de la chaîne se fera de gauche à droite.
4. Écrire une fonction `nb_occurences_d_g : char -> string -> int` qui fait le même calcul que la fonction précédente, mais cette fois-ci le parcours de la chaîne se fera de droite à gauche, et on utilisera un accumulateur.

2 La disparition

Un lipogramme est un texte d'où certaines lettres ont été exclues.

1. Écrire une fonction `est_lipogramme : string -> char -> bool` qui renvoie vrai si la chaîne est un lipogramme selon le caractère donné en paramètre, et faux sinon.
2. Écrire une fonction `lipogramme : string -> char -> string` qui prend une chaîne quelconque et la transforme en lipogramme selon le caractère donné en paramètre. Par exemple, `lipogramme "le chat et la mouette" 'e';;` renvoie `"l chat t la moutt"`. Vous ferez 2 versions (parcours de la chaîne de gauche à droite et de droite à gauche).

   Remarque :

   ```
   # String.make 1 'd';;
   - : string = "d"
   ```

3 Jeux de mots

3.1 Version classique

1. Un palindrome est un mot qui peut se lire dans les deux sens, i.e. l'ordre des lettres est le même qu'il soit lu de gauche à droite ou de droite à gauche. Ainsi, `sagas` est un palindrome. Écrire une fonction `est_palindrome : string -> bool` qui renvoie vrai si la chaîne est un palindrome, et faux sinon. Par exemple, `est_palindrome "engagelejeuquejelegagne"` renvoie vrai, et `est_palindrome "engage le jeu que je le gagne"` renvoie faux, à cause des espaces.
2. Écrire une fonction `est_prefixe : string -> string -> bool` qui renvoie vrai si la première chaîne est un préfixe de la seconde (i.e. il s'agit de son début).
3. Écrire une fonction `est_suffixe : string -> string -> bool` qui renvoie vrai si la première chaîne est un suffixe de la seconde (i.e. il s'agit de sa fin).
4. Écrire une fonction `est_facteur : string -> string -> bool` qui renvoie vrai si la première chaîne est un facteur de la seconde (i.e. il s'agit d'un morceau de la chaîne). Par exemple, `est_facteur "ab" "acabd"` renvoie vrai, mais `est_facteur "ac" "agcabd"` renvoie faux.
5. Écrire une fonction `est_sous_mot : string -> string -> bool` qui renvoie vrai si la première chaîne est un sous-mot de la seconde (i.e. une suite de lettres, non forcément consécutives, mais dans l'ordre de lecture). Par exemple, `est_sous_mot "ab" "meatdbac"` renvoie vrai, mais `est_sous_mot "ab" "cboac"` renvoie faux.

3.2 Version améliorée

1. On souhaite que les fonctions précédentes marchent sans tenir compte des espaces dans la chaîne. Pour cela, il n'est pas nécessaire de les réécrire entièrement, mais simplement d'ajouter un pré-traitement au bon endroit...

   Remarque :

   ```
   # String.make 1 'd';;
   - : string = "d"
   ```

3.3 Mots entremêlés

Soient deux mots. On peut les entremêler pour former un nouveau mot en prenant tour à tour une lettre dans chaque mot. Par exemple, `"shoe"` et `"cold"` s'entremêlent pour former `"schooled"`.

1. Écrire une fonction `mots_meles : string -> string -> string` qui prend 2 chaînes de même longueur et renvoie le mot mêlé correspondant.
2. Si les deux chaînes ne sont pas de même longueur, on souhaite que la fin de la chaîne la plus longue soit ajoutée à la fin du mot mêlé. Par exemple, `"shoeabc"` et `"cold"` forment `"schooledabc"`, et `"shoe"` et `"coldxyz"` forment `"schooledxyz"`. Proposer une seconde version `mots_meles2 : string -> string -> string` qui tient compte de cela.

   Remarque :

   ```
   # String.sub "bonjour a tous" 4 7;;
   - : string = "our a t"
   ```