TP3 : Récursivité et chaînes récursives

Module

L'objectif de ce TP est triple :

1. Vous permettre d'écrire des fonctions récursives dans un cadre contraint (les fonctions externes utilisables sont imposées) ;
2. De définir vous même les fonctions externes support (nil(), head(), tail()) ;
3. De vous familiariser avant les prochains tests, examens et TPs, à la lecture et à l'écriture dans un fichier.

et, comme la semaine dernière, de pratiquer un peu plus `git`, Visual Studio Code, Python3 et `venv`.

★ **Exercice 1:** En utilisant des listes python, écrivez les fonctions python suivantes qui vous serviront dans le reste du TP

$\mathbf{nil}(\mathbf{l} : \mathbf{list}) -> \mathit{bool}$	La liste vide
$\mathbf{head}(\mathbf{l} : \mathbf{list}) -> \mathit{char}$	Récupère le premier caractère de la liste (pas défini si list est la chaîne vide)
$\mathbf{tail}(\mathbf{l} : \mathbf{list}) -> \mathbf{list}$	Retourne la liste amputée de son premier élément
$\mathbf{add}(\mathbf{c} : \mathbf{char}, \mathbf{l} : \mathbf{list}) -> \mathbf{list}$	Ajoute le caractère en tête de liste

Vous pouvez utiliser dans la suite du TP le type List de Python, mais vous ne pouvez utiliser que les opérations ci-dessus pour vos fonctions récursives. Pour chaque question, vous créerez un fichier python qui porte le même nom (avec l'extension `.py` en plus que la fonction à implémenter. Pour tester vos fonctions, un fichier nommé `Tx.txt` ou x représente le numéro de la question est à votre disposition. Le contenu de chaque fichier est détaillé sous chaque question. Pour chaque question votre programme devra créer un fichier `Rx.txt` dans lequel vous écrirez les résultats de l'application des données de `Tx.txt` à vos fonctions.
Écrire les fonctions suivantes :

▷ **Question 1:** *longueur* : `List[Char]` $\mapsto$ `Int`
retourne le nombre de lettres composant la chaîne

Pour tester votre fonction, vous avez à votre disposition le fichier `T1.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)
Lignes 2 à n+1 une chaîne de caractères par ligne

Votre fichier réponse `R1.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction (ici un entier).

▷ **Question 2:** *est_membre* : `List[Char]` $\times$ `Char` $\mapsto$ `Boolean`
retourne `true` ssi le caractère fait partie de la chaîne

Pour tester votre fonction, vous avez à votre disposition le fichier `T2.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)
Lignes 2 à n+1 sur chaque ligne, le caractère recherché puis une chaîne de caractères séparés par une virgule

Votre fichier réponse `R2.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction (ici True ou False).

▷ **Question 3:** *occurence* : `Char` $\times$ `List[Char]` $\mapsto$ `Int`
retourne le nombre d'occurences du caractère dans la chaîne

Pour tester votre fonction, vous avez à votre disposition le fichier `T3.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)
Lignes 2 à n+1 sur chaque ligne, le caractère recherché puis une chaîne de caractères séparés par une virgule

Votre fichier réponse `R3.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 4:** *tous_differents* : `List[Char]` $\mapsto$ `Boolean`
retourne `true` ssi tous les membres de la chaine sont différents

Pour tester votre fonction, vous avez à votre disposition le fichier `T4.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 la chaîne de caractères à explorer

Votre fichier réponse `R4.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 5:** *supprime* : $\begin{cases} \texttt{Char} \times \texttt{List[Char]} \mapsto \texttt{List[Char]} \\ \text{retourne la chaine privée de toutes les occurences du caractère.} \end{cases}$

Si le caractère ne fait pas partie de la chaîne, celle-ci est inchangée.

Pour tester votre fonction, vous avez à votre disposition le fichier `T5.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , le caractère à supprimer suivi de la chaîne de caractères à explorer, les deux sont séparés par une virgule.

Votre fichier réponse `R5.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 6:** *deuxieme* : $\begin{cases} \texttt{List[Char]} \mapsto \texttt{Char} \\ \text{retourne le deuxième caractère de la chaîne} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T6.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 la chaîne de caractères à explorer

Votre fichier réponse `R6.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 7:** *dernier* : $\begin{cases} \texttt{List[Char]} \mapsto \texttt{Char} \\ \text{retourne le dernier caractère de la chaîne} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T7.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 la chaîne de caractères à explorer

Votre fichier réponse `R7.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 8:** *saufdernier* : $\begin{cases} \texttt{List[Char]} \mapsto \texttt{List[Char]} \\ \text{retourne la chaine privée de son dernier caractère} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T8.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 la chaîne de caractères à explorer

Votre fichier réponse `R8.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 9:** *nieme* : $\begin{cases} \texttt{List[Char]} \times \texttt{Int} \mapsto \texttt{Char} \\ \text{retourne le nieme caractère de la chaîne} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T9.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , le rang du caractère à supprimer suivi de la chaîne de caractères à explorer, les deux séparés par une virgule.

Votre fichier réponse `R9.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 10:** *npremiers* : $\begin{cases} \texttt{List[Char]} \times \texttt{Int} \mapsto \texttt{List[Char]} \\ \text{retourne les n premiers caractères de la chaîne} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T10.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , le nombre de caractères à garder dans la chaîne de caractères suivi de la chaîne de caractères à explorer, les deux séparés par une virgule.

Votre fichier réponse `R10.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 11:** *nderniers* : $\begin{cases} \texttt{List[Char]} \times \texttt{Int} \mapsto \texttt{List[Char]} \\ \text{retourne les n derniers caractères de la chaîne} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T11.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , le nombre de caractères à garder dans la chaîne de caractères suivi de la chaîne de caractères à explorer, les deux séparés par une virgule.

Votre fichier réponse `R11.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 12:** *retourne* : $\begin{cases} \texttt{List[Char]} \mapsto \texttt{List[Char]} \\ \text{retourne la chaine lue en sens inverse} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T12.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , pour chaque cas de test, la chaîne à retourner.

Votre fichier réponse `R12.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 13:** *concat* : $\begin{cases} \texttt{List[Char]} \times \texttt{List[Char]} \mapsto \texttt{List[Char]} \\ \text{retourne les deux chaines concaténées} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T13.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , pour chaque cas de test, les deux chaînes de caractères séparées par une virgule.

Votre fichier réponse `R13.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 14:** *min_ch* : $\begin{cases} \texttt{List[Char]} \mapsto \texttt{Char} \\ \text{retourne le caractère le plus petit de la chaîne} \end{cases}$

On considère l'ordre lexicographique, et on suppose l'existance d'une fonction min(a,b).

Pour tester votre fonction, vous avez à votre disposition le fichier `T14.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , pour chaque cas de test, la chaîne de caractères à traiter

Votre fichier réponse `R14.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 15:** *croissante* : $\begin{cases} \texttt{List[Char]} \mapsto \textit{booléen} \\ \text{retourne si la chaine est croissante (dans l'ordre lexicographique)} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T15.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , pour chaque cas de test, la chaîne de caractères à traiter

Votre fichier réponse `R15.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 16:** *nnaturels* : $\begin{cases} \texttt{Int} \mapsto \texttt{List[Int]} \\ \text{retourne une chaine formée des n premiers entiers naturels} \end{cases}$

Dans un premier temps, on construira $\{n, n-1, n-2, \ldots, 3, 2, 1\}$ avant de construire $\{1, 2, 3, \ldots, n\}$.

Pour tester votre fonction, vous avez à votre disposition le fichier `T16.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , pour chaque cas de test, le nombre d'entiers naturels souhaités

Votre fichier réponse `R16.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction. Pour des besoins de lisibilité, séparez les entiers dans la chaîne résultante par une virgule suivie d'un espace " ".

▷ **Question 17:** *palindrome* : $\begin{cases} \texttt{List[Char]} \mapsto \textit{booléen} \\ \text{retourne VRAI si la chaine est un palindrome} \end{cases}$

Un palindrome se lit indifféremment de droite à gauche ou de gauche à droite. Exemple : « Esope reste et se repose ». On peut ignorer les espaces.

Pour tester votre fonction, vous avez à votre disposition le fichier `T17.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , pour chaque cas de test, la chaîne de caractères à traiter

Avant d'effectuer les tests 5 à 8, il vous faudra éliminer de la chaîne de tests, les caractères spéciaux suivants : espaces, ".", ",", " ?", " !" et transformer les majuscules en minuscules (sur le cas)

Votre fichier réponse `R17.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction (ici un booléen TRUE ou FALSE).

▷ **Question 18:** *anagramme* : $\begin{cases} \texttt{List[Char]} \times \texttt{List[Char]} \mapsto \textit{booléen} \\ \text{retourne VRAI si les chaines sont des anagrammes l'une de l'autre} \end{cases}$

Une anagramme d'un mot est un autre mot obtenu en permutant les lettres. Exemples : «chien» et «niche» ; «baignade» et «badinage» ; «Séduction», «éconduits» et «on discute».

Pour tester votre fonction, vous avez à votre disposition le fichier `T18.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , pour chaque cas de test, les deux chaînes de caractères à traiter, séparées par une virgule.

Votre fichier réponse `R18.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction (ici un booléen TRUE ou FALSE).

▷ **Question 19:** *union* : $\begin{cases} \texttt{List[Char]} \times \texttt{List[Char]} \mapsto \texttt{List[Char]} \\ \text{retourne une chaîne formée de toutes les lettres de ch1 et ch2, sans doublons} \end{cases}$

On peut supposer dans un premier temps que ch1 et ch2 ne contiennent pas de doublons.

Pour tester votre fonction, vous avez à votre disposition le fichier `T19.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , pour chaque cas de test, les deux chaînes de caractères à traiter, séparées par une virgule.

Votre fichier réponse `R19.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

▷ **Question 20:** $difference$: $\begin{cases} \texttt{List[Char]} \times \texttt{List[Char]} \mapsto \texttt{List[Char]} \\ \text{retourne toutes les lettres de ch1 ne faisant pas partie de ch2} \end{cases}$

Pour tester votre fonction, vous avez à votre disposition le fichier `T20.txt`. Celui-ci est organisé comme suit :

Ligne 1 : nombre de cas de tests dans le fichier (n)

Lignes 2 à n+1 , pour chaque cas de test, les deux chaînes de caractères à traiter, séparées par une virgule.

Votre fichier réponse `R20.txt` devra comporter une ligne pour chaque test réalisé et cette ligne devra contenir le résultat de votre fonction.

★ **Exercice 2:** Les élèves ayant terminé les 20 questions précédentes sont invité à implémenter une fonction simple (force brute) qui, étant données de chaînes de caractères $c1$ et $c2$, renvoit VRAI si $c1$ est une sous-chaîne de $c2$, FAUX sinon. En cas de réponse positive, la fonction devra également renvoyer l'indice de début de la sous-chaîne dans la chaîne.

★ **Exercice 3:** En 1970 James Morris et Vaughan Pratt ont proposé une amélioration de l'algorithme naïf (ou force brute). L'algorithme est décrit à l'adresse suivante :

`https://www-igm.univ-mlv.fr/~lecroq/string/node7.html`

Implémentez le et comparez sur un ensemble de chaînes la différence de performances avec l'algorithme "force brute".