03 - TD : listes, récursivité, complexité

Les deux premiers exercices étaient déjà dans les cours précédents, si vous les avez déjà traités passez directement à l'exercice 3.

Exercice 1 : Proposez une fonction récursive qui compte les occurrences d'une lettre dans un mot. Elle prend deux paramètres, la lettre (de type caractère) et le mot (de type chaine de caractère). Elle retourne le nombre de fois où cette lettre apparaît dans le mot.
Cette fonction exploitera l'idée suivante : si le mot est vide, le nombre d'occurrences est nulle. Sinon on pourra examiner si le premier caractère du mot coïncide avec la lettre cherchée, et lancer la recherche sur les lettres suivantes du mot.

Exercice 2 : .

On rappelle que si a et b sont deux entiers naturels, non tous les deux nuls, tels que $a \geq b$, $\text{PGCD}(a,b) = \text{PGCD}(a-b,b)$

1. (Sur une feuille pour comprendre le procédé) En utilisant plusieurs fois de suite cette propriété, calculer $\text{PGCD}(36,24)$, puis $\text{PGCD}(16,9)$. Vous n'avez pas le droit de faire des divisions ou des multiplications. Vous réfléchirez à la condition qui vous a permis, à la fin de votre processus, de conclure.
2. Ecrire une fonction non récursive d'arguments a et b qui calcule le PGCD de a et b.
3. Ecrire une fonction récursive d'arguments a et b qui calcule le PGCD de a et b .

Exercice 3 (*Autour de la recherche du minimum d'une liste*) : .

1. (a) Ecrire une fonction `valeur_min`, ayant pour paramètre d'entrée une liste de nombres, et qui retourne la valeur minimum de cette liste.
   Remarque : Il existe une fonction native dans Python qui réalise cette opération, c'est la fonction `min`.
   (b) Quelle est, en fonction de la taille n de la liste, la complexité de votre fonction ?
2. Ecrire une fonction `indice_min`, ayant pour paramètre d'entrée une liste de nombres, et qui retourne l'indice de la valeur minimum de cette liste. Dans le cas où plusieurs indices correspondraient à cette valeur minimum, vous retournez l'indice le plus petit.
3. Ecrire une fonction `couple_min`, ayant pour paramètre d'entrée une liste de nombres, et qui retourne une liste de deux nombres, composée de l'indice de la valeur minimum de cette liste, et de cette valeur minimum (si plusieurs indices sont possibles, vous indiquerez le plus petit).

Exercice 4 : On dispose de deux listes de même taille :

- Une liste de nombres `L`
- Une liste de booléens `E`, tel que `E[i]` est `True` uniquement si le nombre `L[i]` est à prendre en considération dans les calculs qui vont suivre. (on dit que chaque nombre est "étiqueté" Vrai ou Faux)

1. Ecrire une fonction **`somme`**, ayant pour paramètre d'entrée deux listes `L` et `E` comme décrit ci-dessus, et qui retourne la somme des nombres de la liste `L`, en ne prenant que ceux qui sont étiquetés Vrai. Dans le cas où tous les nombres seraient étiquetés Faux, on retournera le booléen `False`.
2. Ecrire une fonction **`valeur_min2`**, ayant pour paramètre d'entrée deux listes `L` et `E` comme décrit ci-dessus, et qui retourne la valeur minimum de la liste `L` (en ne prenant en compte que les nombres qui sont étiquetés Vrai).Dans le cas où tous les nombres seraient étiquetés Faux, on retournera la valeur $+\infty$. (en Python l'infini est `float('inf')` ou `np.inf` du module `numpy`)

Exercice 5 (*Tri d'une liste par sélection*) : .

Le but de cet exercice est de créer une fonction qui trie une liste de nombres en rangeant ses éléments par ordre croissant.

Voici le principe du tri par sélection : on trie progressivement la liste, en la parcourant de gauche à droite, en déterminant la plus petite des valeurs non encore triées et en la mettant à sa place.

Plus précisément, étant donné une liste `L` de taille n, pour k allant de 0 à $(n-2)$:

- Déterminer un indice j tel que `L[j]` soit la plus petite des valeurs de la portion de la liste `L` entre les indices k et $n-1$ inclus.
- Echanger `L[k]` et `L[j]`

A la fin du processus, la liste est triée.

1. *Création d'une fonction auxiliaire* : Ecrire une fonction **`indice_min3`** recevant comme paramètre une liste `L` de longueur n et deux entiers g et d tels que $0 \leq g \leq d < n$, et renvoyant j tel que L[j] soit la plus petite valeur de la partie de la liste `L` comprise entre les indices g et d inclus.
2. Ecrire une fonction **`tri_selection`** recevant comme paramètre une liste `L` et qui trie les éléments de cette liste dans l'ordre croissant, en suivant le principe expliqué en début d'exercice.
   Cette fonction devra donc modifier la liste `L` (on dit que le tri s'effectue *en place*, aucune nouvelle liste n'est créée), et retourner cette liste `L` triée.
3. Quelle est, en fonction de la taille n de la liste, la complexité de votre fonction **`tri_selection`** ?

Exercice 6 (*Si vous avez le temps : Tri rapide*) : .

Le principe du tri rapide consiste à choisir un élément p du tableau, appelé pivot, puis à trier le tableau en mettant les éléments plus petits que p à gauche de p, et les éléments plus grands que p à droite de p. Ensuite, on recommence le processus sur le tableau de gauche d'une part (c'est à dire les éléments situés à gauche du pivot) et sur le tableau de droite d'autre part.

Pour simplifier les choses, nous choisirons toujours comme pivot le premier élément du tableau.

1. Écrire une fonction récursive **`tri_rapide`** qui implémente le tri rapide d'une liste. Vous pourrez tout d'abord créer une fonction **`partition`** qui à partir d'une liste, retourne un triplet formé du pivot (le premier élément de la liste donc), de la liste des éléments plus petits que le pivot, et de la liste des éléments plus grand que le pivot.
2. Déterminer la complexité du tri rapide.