I Autour des listes

Exercice 1
Ecrire une fonction `maximum(L)` où L est une liste d'entiers et qui retourne la valeur du maximum de L, l'indice de la première occurrence ainsi que le nombre d'occurrences de ce maximum.

Exercice 2
Ecrire une fonction `est_triee(L)` qui renvoie un booléen indiquant si la liste L est triée dans l'ordre croissant.
Question bonus : comment pourrait-on modifier cette fonction pour prendre en compte l'ordre de tri ? On veut obtenir `est_triee(L, ordre)` où `ordre` vaudra 1 pour l'ordre croissant et -1 pour l'ordre décroissant.

Exercice 3 (Bonus)
Ecrire une fonction `distance_max(L)` qui retourne la plus grande distance entre deux éléments de L. La distance entre deux nombres est mesurée par la valeur absolue de leur différence.
Evaluer la complexité.

Exercice 4
Écrire une fonction `maxi_locaux` qui prend une liste `L` de nombres comme argument et renvoie la liste (éventuellement vide) des indices des maxima locaux de `L`.
La première question à se poser est : comment déterminer si la valeur `L[i]` est un maximum local ?

II Arithmétique

Exercice 5
Implémenter la fonction `pgcd(a, b)` qui retourne le pgcd de a et b en utilisant l'algorithme d'Euclide. On pourra en donner au choix une version itérative ou une version récursive.

Exercice 6
Ecrire une fonction `est_premier(n)` qui renvoie le booléen indiquant si n est un entier premier.

Exercice 7
On dit qu'un nombre entier naturel n est *parfait* ssi il est égal à la somme de tous ses diviseurs propres (différents de n). Par exemple 6 est parfait car ses diviseurs propres sont $1, 2$ et 3 et $1 + 2 + 3 = 6$.
Le prochain nombre parfait est 28. Donner la liste de tous les nombres parfaits entre 2 et 10 000

III Algorithmes numériques

Exercice 8
Trouver le plus petit n tel que $\sum_{k=1}^{n} \frac{1}{k} > 10$.

Exercice 9
Etant donné deux listes T et L (de même longueur) qui représentent respectivement des temps et les valeurs d'une certaine fonction f en ces temps, on cherche les points d'annulation de f.
On détecte ces annulations via la méthode suivante. Notons $\mathrm{T} = [t_0, \ldots, t_{n-1}]$. f change de signe entre t_i et t_{i+1} si et seulement si $f(t_i)$ et $f(t_{i+1})$ (deux valeurs consécutives dans L) n'ont pas le même signe. Pour donner une approximation du temps d'annulation, on considère que f est affine sur $[t_i, t_{i+1}]$.
Ecrire une fonction `annulations(T, L)` qui renvoie la liste des temps où f s'annule.
Pour tester, et après les bons imports,

```
T = np.linspace(-np.pi, np.pi)
L = np.cos(T) - T/4
```

et on doit trouver deux points d'annulation.

IV Chaînes de caractère

Exercice 10
Ecrire une fonction `occurences_sous_chaine(motif, chaine)` qui renvoie le nombre d'occurrences de la chaîne `motif` dans `chaine`.
Par exemple pour le motif 'aba' et la chaîne 'aababa' on renvoie 2 (les occurrences n'ont pas à être disjointes).
INDICATION : tranche (ou slice).

Exercice 11
Ecrire une fonction `frequence_max(chaine)` qui renvoie la lettre qui apparaît le plus souvent dans la chaîne donnée. On considère que toutes les lettres sont en minuscule et que la chaîne ne comprend pas de caractère spécial.

Exercice 12
Ecrire une fonction `anagramme(mot1, mot2)` qui renvoie le booléen indiquant si les deux mots sont anagramme l'un de l'autre (sont composés des mêmes lettre, pas forcément dans le même ordre).