TP5 - Recherche séquentielle dans une liste

→ *Q.1) Créez un fichier* `TP5.py` *dans lequel vous pourrez sauvegarder votre travail.*

A) Recherche dans une liste

A.1 Recherche d'un élément

On souhaite déterminer si une liste L contient un élément e donné ou non. Nous allons donc parcourir les éléments de la liste L un par un :

— si l'élément courant est égal à e, on peut arrêter la boucle et renvoyer tout de suite True ;

— sinon, on passe à l'élément suivant.

→ *Q.2) Écrire une fonction* `appartient(L, e)` *qui renvoie* `True` *si* `e` *est un élément de* `L` *et* `False` *sinon.*

→ *Q.3) Modifier la fonction précédente pour qu'elle renvoie aussi le premier indice d'apparition de* `e` *dans* `L`*, ou bien* -1 *si l'élément n'est pas dans la liste.*

→ *Q.4) Modifier la fonction pour quelle renvoie la liste des indices d'apparition de* `e` *dans* `L`.

Python a une instruction qui permet de vérifier si un élément e est dans la liste L : e **in** L qui vaut True si c'est le cas. **Attention** : cette instruction a l'air très rapide à exécuter, mais sa complexité est en fait linéaire en la taille n de la liste L : $O(n)$. Ce n'est pas une opération élémentaire.

B) Recherche du maximum

On considère une liste L de n éléments. On souhaite trouver son maximum. Pour cela, on stocke le premier élément de L dans une variable **max** puis pour chaque élément de L, s'il est supérieur à **max**, on remplace la valeur de **max** par cet élément. La valeur finale de **max** est le maximum de L.

→ *Q.5) Écrire une fonction* `maximum(L)` *qui renvoie le maximum de* `L`.

→ *Q.6) Écrire une fonction* `maximum2(L)` *qui renvoie la position de la première apparition du maximum de* `L`.

→ *Q.7) Écrire une fonction* `maximum3(L)` *qui renvoie la position de la dernière apparition du maximum de* `L`.

→ *Q.8) Écrire une fonction* `maximum4(L)` *qui renvoie le maximum et la liste des positions où apparaît ce maximum.*

Pour tester ces fonctions, nous allons utiliser la fonction random du module random.
La fonction random ne prend pas de paramètre et renvoie un flottant entre 0 et 1 au hasard.

→ *Q.9) Tester les fonctions précédentes avec une liste* `L` *de taille* 10^4 *avec des flottants au hasard entre* 0 *et* 1.

C) Comptage des éléments d'une liste

Pour chercher si un élément e est une clé du dictionnaire d, on dispose de : e **in** d qui vaut True si c'est le cas et False sinon. Contrairement aux listes, cette instruction s'exécute en **temps constant** : son temps d'exécution ne dépend pas de la taille du dictionnaire. On note dans ce cas $O(1)$.
On dispose d'une liste quelconque L et on souhaite en faire un inventaire dans un dictionnaire : les clés de ce dictionnaire seront les valeurs distinctes qui apparaissent dans L, et les valeurs seront le nombre de fois que chaque valeur apparaît.
Pour cela, on parcourt la liste L et pour chaque élément e, si c'est déjà une clé, on augmente la valeur correspondante de 1, sinon, on crée une clé dont la valeur vaut 1.

→ *Q.10) Écrire une fonction* `comptage(L)` *qui renvoie le dictionnaire qui fait l'inventaire de la liste L.*

→ *Q.11) Commenter cette fonction.*

→ *Q.12) Tester cette fonction sur la liste vide, sur une liste de* 10 *zéros.*

→ *Q.13) Écrire une fonction* `liste_alea(n)` *qui renvoie une liste de taille* `n` *composée d'entiers aléatoires entre* 0 *et* 10.

→ *Q.14) Tester la fonction* `comptage` *avec la fonction précédente.*

D) Pour les plus motivés

→ *Q.15) Écrire une fonction qui renvoie les deux plus grandes valeurs d'une liste. Si tous les éléments de la liste ont la même valeur, on ne renvoie que cette valeur. On essaiera de plus de n'effectuer qu'un seul parcours de la liste.*

→ *Q.16) Écrire une fonction* `plus_longue_sous_liste_croissante(L)` *qui prend en argument une liste d'entiers et qui renvoie la longueur de la plus grande sous-liste croissante de* `L` *(les éléments doivent être successifs) ainsi qu'une sous-liste de cette taille.*