TP-1 : parcours séquentiels et comptage

1 Calcul de la moyenne d'une liste de nombres.

On suppose que l'on a une liste L, de longueur $n = len(L) > 0$, de nombres.

1. Écrire une fonction `moyenne` qui prend en argument une telle liste et retourne, calcule sa moyenne $m = \frac{1}{n}\sum_{i=0}^{n-1} L[i]$ et la tester sur quelques exemples.
   (évidemment, Python a une fonction qui calcule la somme des éléments d'une liste et aussi une fonction qui calcule directement la moyenne d'un tableau de nombres mais on s'interdit de les utiliser ici)
   Préciser le nombre d'opérations (additions et division) nécessaires pour faire ce calcul.
2. On rappelle que la variance de la liste L considérée comme une série statistique est la moyenne des carrés des écarts à la moyenne, c'est-à-dire $V = \frac{1}{n}\sum_{i=0}^{n-1}(L[i] - m)^2$. Un théorème dit que V est en fait égal à la moyenne des carrés des éléments de L moins le carré de la moyenne de L. En supposant que la fonction `moyenne` est déjà écrite donc utilisable, écrire deux fonctions `variance1` et `variance2` qui pour une liste de nombres retourne sa variance en utilisant chacune des deux caractérisations de la variance données ci-dessus.
   Préciser le nombre d'opérations élémentaires nécessaires pour une liste de longueur n pour chacune des deux fonctions.

2 Recherche d'un élément dans une liste.

On suppose que l'on a un objet itérable (voire indexable), par exemple ici une liste `L`, mais cela pourrait être une chaîne de caractère ou un dictionnaire, et un objet `a` et l'on veut savoir si a est élément de la liste L. Il y a bien sûr une syntaxe Python qui fait cela, c'est `a in L` qui retourne le booléen `True` si a est dans L et `False` sinon. On va ici implémenter nous-mêmes cette fonction, que l'on pourra nommer `Est_élément` (on peut supprimer les accents) qui prend deux paramètres, dans cet ordre, une liste `L` et un objet `a`. Pour cela il suffit de créer une boucle qui prend les éléments de L les uns après les autres et s'arrête en retournant True dès que on a rencontré a dans L. Sinon, on retourne False. On parle de *recherche séquentielle*.

1. Implémenter cela dans un script et tester la fonction avec quelques exemples, qui explorent de manière pertinente tous les cas qui peuvent se présenter. Il s'agit de voir si d'une part l'algorithme tourne bien (il termine sans sortie intempestive à cause d'une erreur de programmation) et d'autre part s'il fait bien ce que l'on attend (on dit qu'il est conforme à sa *spécification*).
2. Pour un algorithme s'appliquant à une donnée de taille n (ici, on va prendre n, la longueur de la liste) on veut évaluer le nombre d'opérations élémentaires pertinentes (ici on va prendre la comparaison « == ») et on étudie son comportement asymptotique, c'est-à-dire ce qui se passe quand n devient grand. On appelle cela la *complexité* de l'algorithme.
   Pour une liste de longueur n, quel est le nombre minimal d'instructions pertinentes exécutées dans la fonction `Est_élément` ? Préciser pour quelle genre de liste on obtient ce résultat.
   On parle de complexité dans le meilleur des cas et ici on dit qu'elle est constante (*coût constant*).
   Pour une liste de longeur n, quel est le nombre maximal d'instructions pertinentes exécutées ?
   On a là la complexité dans le pire des cas. Justifier la qualificatif de linéaire (*coût linéaire*).
3. Écrire une fonction qui prend en argument une chaîne de caractères et un caractère 'c' et vérifie la présence ou pas du caractère dans la chaîne de caractères (et la tester).
4. Adapter, modifier, la première question en demandant en plus, en cas de présence dans la donnée indexable, de retourner un indice correspondant à l'élément cherché (puis la liste des indices correspondants).

3 Recherche d'un extremum dans une liste de nombres.

1. On suppose que la liste est non vide d'objets que l'on peut comparer. Pour déterminer le maximum des éléments de la liste on peut procéder par recherche séquentielle, en créant par exemple une variable `maxi` initialisée au premier élément de la liste, puis en parcourant toute la liste, si on rencontre un élément e strictement supérieur à la valeur actuelle de $maxi$, on affecte le e à $maxi$. Comme on est sûr de parcourir toute la liste, le `for` s'impose.

   Implémenter une fonction `maximum` qui prend en argument une liste d'entiers (pour faire simple) et qui retourne la maximum de la liste. Écrire une variante qui retourne en plus du maximum, la liste des indices où cette valeur est atteinte.

2. On cherche maintenant le *deuxième maximum*, c'est-à-dire, ici, la valeur maximale, une fois que l'on a éliminé les éléments égaux à la valeur maximale entendue comme ci-dessus. On suppose que la liste contient au moins deux valeurs distinctes.

   Écrire une première fonction de son choix `deux_maximum` qui, pour une telle liste L, retourne le couple $(max1, max2)$ tel que $max1$ et $max2$ soient éléments de L, et tel que pour tout élément e dans L, on ait $e \leqslant max1$ et ($e < max1$ entraine $e \leqslant max2$).

3. Écrire (si distincte de la première) une deuxième version `deux_maximum2` qui, pour une telle liste L, crée le couple initialisé à `res = [L[0], L[1]]` lorsque $L[0] \geqslant L[1]$ et à `[L[1],L[0]]` dans le cas contraire. Ensuite, on parcourt la liste et on actualise au fur et à mesure de l'avance de manière à ce que au moment où on a examiné les k premiers éléments de la liste L, le couple $(res[0], res[1])$ soit le premier maximum suivi du second maximum de la liste extraite `L[0:k-1]` dès qu'il y a au moins deux valeurs distinctes dans cette tranche.

4. Étudier et comparer les complexités respectives pour les deux versions de `deux_maximum` pour une liste de longueur n. (On pourra compter les comparaisons et les affectations)

4 Comptage des éléments d'une liste

On considère une liste L de longueur $n = len(L)$ non nulle. On voudrait préciser ses éléments et le nombre de représentants par élément. Par exemple pour `L = [1, 0, 4, 2, 1, 0, 4, 1, 0]`, on a trois fois le 0, trois fois le 1, une fois le 2 et deux fois le 4. Le mieux est d'utiliser un dictionnaire qui prendra en clés les valeurs présentes dans la liste L et pour chacune de ces valeurs le nombre d'occurrences de la valeur dans la liste L.

1. Implémenter une fonction `compte` qui prend en argument une liste et, avec un parcours séquentiel, retourne le dictionnaire décrit ci-dessus (on testera la fonction bien sûr). On pourra partir du cadre qui suit.

```
def compte(liste):
    d = {} # crée un dictionnaire vide
    for e in liste:
        ... # si e n'était pas encore une clé de d, on la fait apparaître
                              en tant que telle avec la valeur 1
        ... # si e est déjà présente comme clé dans d, sa valeur est incrémentée de 1
    return d
```

2. Lorsque la liste L est une liste d'entiers naturels, on peut créer la liste des entiers compris entre 0 et $p = \max(L)$, considérer la liste `T = [0]*(p+1)` (pourquoi $p+1$?) et parcourir L en incrémentant $T[i]$ de 1 à chaque fois que i apparaît dans L.

   Implémenter une fonction qui fait cela.