PYTHON : SOUTIEN SUR LES LISTES

EXERCICE 1 (Parcours...)

Dans cet exercice, les fonctions seront programmées dans un premier temps à l'aide d'une boucle **for**, puis, dans un second temps, en utilisant les listes par compréhension.

1) Programmer une fonction Python **incremente**(`liste,a`) qui prend en argument une liste numérique et qui renvoie un nouvelle liste qui contient les mêmes éléments incrémentés de `a`.

Par exemple **incremente**($[1,3,2]$, -2) renvoie $[-1,1,0]$.

2) Programmer une fonction Python **somme**(`listeA,listeB`) qui prend en argument deux listes numériques de même longueur et qui renvoie la liste des additions terme à terme.

Par exemple **somme**($[1,3,2]$, $[-1,4,4]$) renvoie $[0,7,6]$.

EXERCICE 2 (Modifications)

1) Programmer une fonction Python **doubleEnPlace**(`liste`) qui prend en argument une liste numérique `liste` et qui modifie la variable pour qu'elle contienne le double de ses valeurs initiales.

Par exemple si au départ `liste`=$[1,3,2]$, alors après avoir exécuté **doubleEnPlace**(`liste`), la variable liste doit valoir $[2,6,4]$.

2) Programmer une fonction Python **substitue**(`listeA,listeB`) qui prend en argument deux listes de même longueur, et qui renvoie une liste qui contient les éléments de `listeA`, sauf s'ils sont pairs, auquel cas, ils sont remplacés par l'élément de même indice dans `listeB`.

Par exemple si `listeA`= $[1,3,2,4,5,0,10]$ et `listeB`= $[-1,4,4,-1,-3,1,0]$, alors la fonction renvoie $[1,3,4,-1,5,1,0]$.

3) Programmer une fonction Python **echange**(`listeA,listeB`) qui prend en argument deux listes de même longueur, et lorsque l'élément dans `listeA` est pair, elle échange cet élément et celui de même indice dans `listeB`.

Par exemple si `listeA`= $[1,3,2,4,5,0,10]$ et `listeB`= $[-1,4,4,-1,-3,1,0]$, alors après exécution de **echange**,

`listeA`= $[1,3,4,-1,5,1,0]$ et `listeB`= $[-1,4,2,4,-3,0,10]$.

EXERCICE 3 (Tests)

1) Créer une fonction Python **contientNul**(`liste`), qui renvoie si 0 appartient à la liste.

2) Créer une fonction Python **nbInf**(`liste,a`), qui prend en argument une liste numérique et un nombre `a`, et qui renvoie le nombre d'éléments de la liste qui sont strictement inférieurs à `a`.

3) On rappelle qu'une médiane d'une liste de nombre est une valeur telle qu'il existe autant d'éléments qui lui soient strictement inférieurs que d'éléments qui lui soient strictement supérieurs.

Ainsi, la médiane n'est pas unique en général.

Programmer une fonction Python **estMediane**(`liste,m`) qui prend en argument une liste numérique et un nombre `m`, et qui renvoie si `m` est une médiane de `liste`.

4) Programmer une fonction Python **estCroissante**(`liste`) qui prend en argument une liste de nombre et qui renvoie si la liste est triée de façon croissante.

5) Programmer une fonction **estNotes**(`liste`) qui vérifie si une liste de nombres n'est bien composée que de nombres compris entre 0 et 20 (inclus).

EXERCICE 4 (Modification sélective)

1) Programmer une fonction Python qui prend en argument une liste de nombre et qui modifie cette liste en conservant les termes impairs et en divisant par deux les termes pairs.

Par exemple si `liste =` $[1,5,4,7,8,3]$, alors elle devient $[1,5,2,7,4,3]$.

Faire une fonction qui crée une nouvelle liste, au lieu de modifier la liste existante.

EXERCICE 5

1) Reprendre l'exercice 7 sur le triangle de Pascal du TD sur les listes.

EXERCICE 6

1) Programmer une fonction Python qui prend en argument une liste numérique et un nombre a et qui renvoie la liste de tous les éléments strictement supérieurs à a.

2) Programmer une fonction sur le modèle de la précédente, mais qui renvoie la liste des éléments dont le précédent est strictement supérieur à a.

Par exemple pour `liste` = $[1,5,3,12,8,14,2,-1,0,7]$, et `a`=4, la fonction renvoie $[3,8,14,2]$.

En effet, les éléments qui précédent 3, 8, 14, et 2 dans la liste, sont tous strictement supérieurs à 4.

EXERCICE 7 (Recherche dans un fichier)

Sur le site, est placé le fichier texte de Du côté de chez Swann.

On pourra s'aider du cours en ligne et utiliser la commande suivante pour ouvrir le fichier (`file` désigne le titre du fichier avec son chemin d'accès. Cela dépend de l'endroit où vous l'enregistrez.).

```
1 fichier = open(file,mode = "r", encoding="utf8")
```

1) À l'aide d'un programme Python chercher le nombre de fois que l'on retrouve le texte `temps` dans le fichier (il n'y a aucun mot coupé).

2) Trouver le nombre de lignes où l'on retrouve la suite de lettres `bcpst` non forcément consécutives, mais dans cet ordre.