Informatique : devoir n°3 (non surveillé)

Tous les programmes doivent être écrits dans un unique fichier nommé `DM3_votrenom.py`. Les noms des fonctions doivent être respectés. Les fichiers mal nommés ou non exécutables ne seront pas corrigés. Les noms des variables doivent être bien choisis. On peut utiliser `#` pour mettre des commentaires. On ne confondra pas `print` et `return`.

Exercice 1

1) Écrire une fonction `premiers_elements` qui, recevant une liste de listes, renvoie la liste des premiers éléments de chacune de ces listes.

```
>>> premiers_elements([[2, 3, 5], [4, 5], [0], [7, 8, 9, 9], [45, 6]])
[2, 4, 0, 7, 45]
```

On donnera deux versions : une avec une boucle `for` et des `append`, l'autre avec une liste définie par compréhension.

2) Écrire une fonction `concatener` qui, recevant une liste de listes, renvoie la liste obtenue en concaténant ces listes.

```
>>> concatener([[2, 3, 5], [4, 5], [0], [7, 8, 9, 9], [45, 6]])
[2, 3, 5, 4, 5, 0, 7, 8, 9, 9, 45, 6]
```

3) Écrire une fonction `positions_memes_elements` qui, recevant deux listes de même longueur, renvoie la liste des indices i pour lesquels les i^e éléments des deux listes sont égaux.

```
>>> positions_memes_elements([7, 2, 6, 3, 9, 5], [6, 2, 3, 4, 9, 8])
[1, 4]
```

Exercice 2

Soit $(x_1, \dots, x_n)$ une famille croissante de réels. Sa médiane est x_{p+1} si $n = 2p+1$ et $(x_p + x_{p+1})/2$ si $n = 2p$ (où $p \in \mathbb{N}$). Par exemple, la médiane de la famille $(1,2,3,4,5)$ est 3 et celle de la famille $(1,2,3,4,5,6)$ est $(3+4)/2 = 3{,}5$. Si la famille $(x_1, \dots, x_n)$ n'est pas croissante, on commence par la mettre dans l'ordre.

Écrire une fonction `mediane` qui, recevant une liste de nombres, renvoie sa médiane.

```
>>> mediane([12, 7, 11, 15, 18])
12
>>> mediane([12, 7, 11, 15])
11.5
```

Exercice 3

La suite de Fibonacci est définie par $u_0 = 0$, $u_1 = 1$ et, pour tout $n \in \mathbb{N}$, $u_{n+2} = u_n + u_{n+1}$. Écrire une fonction `fibonacci` qui, recevant un entier $n \geqslant 2$, renvoie la liste des n premiers termes de la suite de Fibonacci.

```
>>> fibonacci(10)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Exercice 4

Soit `L` une liste dont les termes valent `0` ou `1`. Le but de l'exercice est de trouver le nombre maximal de `0` contigus dans `L` (c'est-à-dire figurant dans des cases consécutives). Par exemple, le nombre maximal de `0` contigus dans la liste `[0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1]` est 4.

Écrire une fonction `nombre_zeros_max` prenant en paramètre une liste `L` et renvoyant le nombre maximal de `0` contigus dans `L`.

```
>>> nombre_zeros_max([0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1])
4
```

On testera la fonction sur de nombreux exemples : liste vide, liste à un élément, liste où la plus longue sous-liste de `0` est au début ou à la fin, liste ne contenant que des `0` ou que des `1`, etc.

Exercice 5 - Compression RLE

La compression RLE (run-length encoding) est une méthode de compression de données très simple. On l'utilise lorsque l'objet à compresser contient de longues séries de répétitions d'un même élément (image en noir et blanc par exemple). L'idée est de remplacer chaque série d'un même élément par le couple formé de la longueur de la série et de la valeur de cet élément.

On travaillera ici avec des listes. Par exemple, la liste `[1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 4]` est formée successivement de quatre `1`, de cinq `2`, de trois `1` et d'un `4`. La liste compressée sera alors `[4, 1, 5, 2, 3, 1, 1, 4]`.

1) Écrire une fonction `compresser` qui prend comme argument une liste et renvoie la liste obtenue par la méthode précédente.

```
>>> compresser([1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 4])
[4, 1, 5, 2, 3, 1, 1, 4]
```

On testera la fonction sur de nombreux exemples.

2) Écrire une fonction `decompresser` qui prend comme argument la liste compressée et renvoie la liste d'origine.

```
>>> decompresser([4, 1, 5, 2, 3, 1, 1, 4])
[1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 4]
```