Séance_python_oral 4 : Oral de l'Agro ou de G2E

1) (*Suite de Conway*)

`L` désigne dans ces questions une liste d'entiers de longueur n.

a) Ecrire une fonction `repetition(L)` qui retourne le nombre de répétitions de `L[0]` au début de la liste.

Exemple : L'appel à la fonction `repetition([2,2,2,2,1,1,1,2])` retourne la valeur 4. En effet la première valeur est 2 est elle apparaît 4 fois au début de la liste.

b) Ecrire une fonction Python `repetitionbis(L)` qui retourne la liste `[x,n,y,m]` lorsque :

$$L = [\ \underbrace{x, x, .., x, x, x}_{n \text{ fois } x}, \underbrace{y, y, ..., y, y}_{m \text{ fois } y}, z,\ \cdots\ ]$$

Exemple : L'appel à la fonction `repetitionbis([2,2,2,2,5,5,5,6,4,1,2,3,5,1,1])` renvoie la liste $[2, 4, 5, 3]$. En effet la première valeur est 2 et elle apparait 4 fois au début de la liste.

La suite de Conway est une suite mathématique inventée en 1986 par le mathématicien John Horton Conway, initialement sous le nom de « suite audioactive ». Elle est également connue sous le nom anglais de Look and Say (« regarder et dire »). Dans cette suite, un terme se détermine en annonçant les chiffres formant le terme précédent.

Le premier terme de la suite de Conway est posé comme égal à 1. Chaque terme de la suite se construit en annonçant le terme précédent, c'est-à-dire en indiquant combien de fois chacun de ses chiffres se répète.

Concrètement : $x_0 = 1$, ce terme comporte juste un « 1 ». Par conséquent, le terme suivant est : $x_1 = 11$. $x_1 = 11$ est composé de deux « 1 ». $x_2 = 21$.

En poursuivant le procédé : $x_3 = 1211$, $x_4 = 111221$, $x_5 = 312211$ et ainsi de suite.

c) Donner les trois termes suivants de la suite.

d) Ecrire une fonction Python permettant de construire x_{n+1} à partir de x_n.

2) Compression et décompression.

Le principe de la compression est simple : si la chaîne contient (strictement) plus d'une occurrence successive du même caractère, comme par exemple le caractère `'c'` dans `'abcccd'`, alors on remplace les n occurrences par le nombre n suivi du caractère répété, par exemple 'ab3cd' (car c est répété 3 fois).

On supposera :

- que les chaînes à compresser ne contiennent pas déjà des chiffres.
- le nombre d'occurrences successives ne dépasse jamais 9

a) Ecrire une fonction `compression(T)`

b) Ecrire sa réciproque `decompression(T)`

Par exemple :

Compression :

```
>>> compression('abcccd')
'ab3cd'

>>> compression('abcccddeeeefgh')
'ab3c2d4efgh'

>>> compression('abcdefg')
'abcdefg'
```

Décompression :

```
>>> decompression('ab3cd')
'abcccd'

>>> decompression('ab3c2d4efgh')
'abcccddeeeefgh'

>>> decompression('abcdefg')
'abcdefg'
```