Séance du 26.03. - Trier un tableau - GrB

A faire pour mardi 26.03.

Exercice 45.
Écrire en pseudo code et en Python un algorithme qui construit un tableau `fibo` de taille 20 et dont :

- l'élément d'indice 0 vaut 5,
  l'élément d'indice 1 vaut 2,
- les autres éléments sont égaux à la somme des deux éléments qui les précèdent.

Le tableau sera affiché à la fin de l'algorithme.

```
###Exercice 45 - Zaina corrigé

def fibo():
    fib = [5, 2]
    while len(fib) < 20 :
        fib = fib + [fib[-1] + fib[-2]]
    return(fib)

print(fibo())
```

Exercice 46 (Tri à bulles).
Considérons un tableau de taille n constitué de nombres entiers. Nous allons trier ces entiers dans l'ordre croissant selon la méthode appelée tri à bulles. Le tri à bulles consiste à répéter $n-1$ fois l'étape suivante : parcourir le tableau en comparant les éléments consécutifs et les échanger s'ils ne sont pas dans l'ordre croissant.

Par exemple si le tableau est `[7;1;5;4;3]` :

- Au cours du premier parcours, 7 est échangé avec 1, puis avec 5, puis avec 4 puis avec 3. Le tableau devient `[1;5;4;3;7]`
  Au cours du second parcours, 1 et 5 ne sont pas échangés car ils sont dans le bon ordre. En revanche, 5 et 4 sont échangés (le tableau devient `[1;4;5;3;7]`), puis 5 et 3 (`[1;4;3;5;7]`). Enfin 5 et 7 ne sont pas échangés.
  Au cours du troisième parcours, 4 et 3 sont échangés, et le tableau obtenu est trié.

1) Écrire en pseudo-code puis en Python une fonction `tribulles` qui prend en paramètre un tableau et trie ce tableau selon le tri à bulles. On pourra s'aider de la fonction précédente.
2) En réalité, le tableau peut être trié avant que les $n-1$ parcours aient été effectués. On peut en effet s'arrêter dès lors qu'aucun n'échange n'est fait pendant un parcours. En s'aidant de cela, modifier la fonction précédente pour limiter le nombre de parcours.

```
###Exercice 46 - Antonin corrigé

def tribulles(tab):
    n = len(tab)
    for i in range(n-1):
        for j in range(0, n-i-1):
            if tab[j] > tab[j+1]:
                prov = tab[j]
                tab[j] = tab[j+1]
                tab[j+1] = prov

    return(tab)
tableau = [5,4,8,4,1,2,9,15,4,7]
print (tribulles(tableau))
```

Exercice 47 (Bogo tri).
On désire trier un tableau en utilisant la méthode appelée "bogo-tri" (ou tri stupide). Pour effectuer ce tri, on répète, jusqu'à ce que le tableau soit trié, les deux étapes suivantes :

- On vérifie si les éléments sont classés par ordre croissant.
- Si ils ne le sont pas, on mélange aléatoirement les éléments du tableau.

1) Écrire en pseudo-code et en Python une fonction `ordre` qui prend en argument un tableau et retourne Vrai si les éléments du tableau sont triés dans l'ordre croissant, et Faux sinon.

2) On considère la fonction `F` suivante :

```
1  Fonction F(tab)
2  Variables locales : ...
3  DEBUT FONCTION
4      bogotab ← []
5      Tant Que longueur(tab)>0:
6          i ← randint(0, longueur(tab)-1)
7          bogotab ← bogotab + [tab[i]]
8          tab ← tab[0:(i-1)] + tab[(i+1):(longueur(tab)-1)]
9      Fin Tant Que
10     Retourner : bogotab
11 FIN FONCTION
```

   a) Lister les variables utilisées dans la fonction F, ainsi que leur type.

   b) À quoi sert la ligne 8 de la fonction ? Que se passe-t-il sans cette ligne ?

   c) Que fait la fonction F ?

   d) Traduire cette fonction en Python.

3) Écrire en pseudo-code et en Python une fonction `tri` qui prend en paramètre un tableau et retourne le tableau trié grâce au bogo-tri.
Cette fonction pourra utiliser les fonctions `F` et `ordre`.
Attention à ne tester que sur de petits tableaux.

4) Pour que le temps de tri soit raisonnable, on décide de limiter le nombre d'itérations du bogo-tri à 100 000. Si au bout de 100 000 itérations, le tableau n'est toujours pas trié, on affichera un message d'échec.
Écrire en Python une fonction `tri2` reprenant la fonction `tri` mais avec la limite d'itérations.

1)

```python
def ordre(tab):          # retourne vraie si le tableau
    i = 0                # est trié croissant, faux sinon
    REP = True
    while i < len(tab)-1 and REP == True:
        if tab[i] > tab[i+1]:
            REP = False
        i = i + 1
    return(REP)
```

A terminer pour la semaine prochaine