C.R. T.P. 4 : tableaux, deuxième partie

1 Tableaux bidimensionnels vus comme listes doubles

Deux questions non traitées dans le corrigé précédent :
f) **Question :** Ecrire une fonction `longmaxLigne` prenant comme argument un tableau et qui renvoie la longueur de la plus grande ligne du tableau.

Réponse : il s'agit d'un brave algorithme de calcul de maximum, mais appliqué à des longueurs de lignes d'un tableau. On reprend donc le schéma de l'algo de calcul de maximum vu au T.P. précédent mais pour les `len(T[i])` si `T` est le tableau bidimensionnel passé en argument à la fonction.

```
def longmaxLigne(T):
    m=len(T[0])
    for i in range(1,len(T)):
        if m<len(T[i]):
            m=len(T[i])
    return m
```

g) En déduire une fonction `transfMatrice` prenant comme argument un tableau et le transformant en matrice ayant toutes ses lignes de même longueur en rajoutant des 0 à la fin des lignes incomplètes.
La fonction `transfMatrice` *doit* transformer *le tableau passé en argument. Elle ne retourne rien.*

```
def transfMatrice(T):
    m=longmaxLigne(T)
    for i in range(len(T)):
            T[i]=T[i]+[0]*(m-len(T[i]))
```

N.B. Via les `T[i]=` il y a vraiment modification du tableau `T` passé en paramètre.

2 Calcul des binomiaux par la formule explicite

a) Ecrire une fonction `factorielle(n)` qui prend en argument un entier `n` et renvoie sa factorielle.

Question alimentaire, à ne pas rater !

```
def factorielle(n):
    F=1
    for i in range(1,n+1):
        F=F*i
    return F
```

b) Ecrire une fonction `binomial(m,n)` qui prend en argument deux entiers `m` et `n` et retourne $\binom{m}{n}$ qui doit être de type *integer*, grâce à la formule explicite.
Réponse de base : L'idée la plus simple, mais pas la plus efficace algorithmiquement !

```
def binomial(m,n):
    return factorielle(m)//(factorielle(n)*factorielle(m-n))
 # avec l'opérateur // de division entière.
```

Le truc à éviter absolument `int(a/b)` qui passerait par le monde de l'approximation des flottants

```
def binomialpourri(m,n):
    return int(factorielle(m)/(factorielle(n)*factorielle(m-n)))
```

Exemple :

```
>>> binomial(1245,23)
48709080146696961862386732116096840650856878287 00

>>> binomialpourri(1245,23)
48709080146696958618875808914105288325208505057 28
```

La fonction `binomialpourri` renvoie un résultat faux dès que l'entier à retourner a plus de seize chiffres : c'est la limitation fondamentale liée aux calculs sur les flottants ! Ne jamais passer par des flottants pour faire des calculs sur les entiers

En oubliant `binomialpourri`, revenons à `binomial` pour dire qu'elle n'est quand même pas efficace :

Pourquoi y-a-t-il plus efficace que `binomial` pour calculer les binomiaux ? Car avec les factorielles, on fait énormément de produits inutiles, à cause de la simplification entre dénominateur et numérateur.

On peut utiliser plutôt la seconde forme de la formule explicite :

$$\binom{m}{n} = \frac{m(m-1)\dots(m-(n-1))}{n!} \quad (*)$$

Il est important de faire les opérations en restant dans le monde des entiers pour ne pas tomber sur des approximations de flottants.

On en déduit alors une fonction `binomialmieux` comme suit :

```
def binomialmieux(m,n):
    num=1 # initialisation du numérateur
    for i in range(n):
        num=num*(m-i)
    return num//fact(n)
```

Comparer les deux fonctions `binomial` et `binomialmieux` avec comme entrée (123456,2345)