Contrôle de deuxième session

En plus des 2 heures que durent l'épreuve, le site où déposer votre copie vous donne 15 minutes supplémentaires pour prendre en compte le temps de télécharger l'énoncé, de scanner votre copie, et de la déposer sur le site web.

Vous ne devez utiliser l'ordinateur que pour télécharger puis afficher l'énoncé et pour déposer votre copie. En particulier il vous est demandé de ne pas utiliser python pendant l'épreuve. Lorsqu'un programme en python est demandé, l'important n'est pas de savoir s'il s'exécute correctement sans erreur de syntaxe, mais si les opérations effectuées répondent à la question.

Pour la rédaction de votre copie, gardez bien en tête que l'important est de convaincre l'examinateur que vous avez compris pourquoi le résultat est correct (il arrive qu'écrire une demi ligne suffise pour cela, mais le plus souvent écrire uniquement le résultat ne suffit pas).

On rappelle de plus que si `a<b`, alors `range(a,b)` désigne l'intervalle $[\![a,b[\![$.

De plus `range(b)` désigne $[\![0,b[\![$.

Par ailleurs, pour une liste `l`, on rappelle que `l[0]` (resp `l[1]`, etc) désigne le premier (resp le deuxième etc) élément de `l`, et que `l[-1]` (resp `l[-2]`, etc) désigne le dernier (resp l'avant dernier etc) élément de `l`.

1. Triangles définis par récurrence

On s'intéresse dans cet exercice à des généralisations du "triangle de Pascal" (qui donne les coefficients binomiaux).

Plus précisément on considère des fonctions $c : \begin{cases} \mathbb{N}\times\mathbb{Z} & \to \mathbb{Z} \\ (n,p) & \mapsto c_{n,p} \end{cases}$, telles que

- $\forall (n,p) \in \mathbb{N}\times\mathbb{Z}, ((p<0 \text{ ou } p>n) \Rightarrow c_{n,p} = 0)$.
- $\forall n \in \mathbb{N}, c_{n,0} = c_{n,n} = 1$.
- Une relation de récurrence exprime $c_{n,p}$ (où $0<p<n$) en fonction des $(c_{i,j})_{\substack{0<i\leqslant n \\ 0<j\leqslant p \\ (i,j)\neq(n,p)}}$.

On considère les quatre exemples de fonctions ci-dessous :

```
def c0(n,p):
    if  p<0 or p>n : return 0
    if p==0 or p==n: return 1
    else   :         return 3*n+2*p-3*c0(n-1,p-1)+4*c0(n-1,p)

def c1(n,p):
    if  p<0 or p>n : return 0
    r=[0,1] # calcul de c1(p-1,p), c1(p,p), c1(p+1,p), ... c1(n,p)
    for k in range(p+1,n+1):
        r.append(4*k-3*p+5*r[k-p]-2r[k-p-1])
    return r[n-p+1]

def c2(n,p):
    def ligne(i):# calcul de c2(i,0), c2(i,1), ... , c2(i,p)
        if i==0:
            return [1]+[0]*p
        l=ligne(i-1)
        nl=[1]+[2*i-3*k+3*l[k-1] for k in range(1,min(p+1,i))]
        if p<i : return nl
        else   : return nl+[1]+[0]*(p-i)))
    if  p<0 or p>n : return 0
    return ligne(n)[p]

def c3(n,p):
    if  p<0 or p>n : return 0
    for i in range(n+1): # calcul de c3(i,0), c3(i,1), ... , c3(i,i)
        if i==0: ligne=[1]
        else   :
            nl=[1]*(i+1)
            for k in range(1,i): nl[k]=-2*i+4*k-3*nl[k-1]+3*ligne[k]
            ligne=nl
    return ligne[p]
```

Notation :

Pour (n,p) fixé (tel que $0 < p < n$), on désigne par $F^{(0)}_{n,p}$ (resp $F^{(1)}_{n,p}$, etc) le nombre de fois ou le programme (quand on exécute `c0(n,p)` resp `c1(n,p)`, etc) utilise la relation de récurrence pour calculer une valeur $c_{i,j}$. *(On supposera dans la suite que la fonction $F_{n,p}$ indique assez bien le temps de calcul du programme.)*

1. (a) Déterminer la relation de récurrence satisfaite par la fonction `c0(n,p)`.
   (b) Déterminer l'expression explicite de la fonction $F^{(0)}_{n,p}$, ou au minimum une relation de récurrence qui définit $F^{(0)}_{n,p}$.
   (c) Peut-on écrire un autre programme qui calcule la même suite $c_{n,p}$ (c'est à dire qu'il utilise la même relation de récurrence) de telle sorte que le nombre d'étapes (c'est à dire le nombre de fois où la relation de récurrence est utilisée pour calculer une valeur $c_{i,j}$) soit plus petit qu'avec cette fonction `c0`. (Si c'est le cas, écrire explicitement ce programme sur votre copie.)

2. Répondre aux mêmes questions pour le programme `c1(n,p)`.
3. Répondre aux mêmes questions pour le programme `c2(n,p)`.
4. Répondre aux mêmes questions pour le programme `c3(n,p)`.

2. Calcul de probabilité et simulations

La France est découpée en 577 circonscriptions électorales (pour les éections législatives). On choisit 45 français au hasard, en supposant pour simplifier que chacun a autant de chance de venir de chacune des ces circonscriptions, et que leur circonscription d'origine sont indépendantes. On cherche à détmerminer la probabilité que parmi ces 45 français il y en ait au moins deux qui viennent de la même circonscription.

5. Calculer cette probabilité de manière combinatoire (en comptant tous les cas "favorables" et en divisant par le nombre total de cas).
6. Écrire un programme en python qui simule cette expérience et vérifie la valeur de la probabilité trouvée en question 5.