1BCPST2 2BCPST1 **Comment simuler des tirages avec ou sans remise ?**

1 Tirages avec remise

On considère une urne contenant p boules blanches et q boules noires dans laquelle on tire n boules **avec** remise.
On propose trois fonctions `avecRemise1(p,q,n)`, `avecRemise2(p,q,n)` et `avecRemise3(p,q,n)` qui simulent cette expérience.

La première fonction renvoie une liste de 0 (noir) et de 1 (blanc) dont l'ordre correspond à l'ordre des tirages. Cette fonction simule des tirages avec remise dans une urne virtuelle, elle est donc plus "proche" de l'expérience concrète mais son emploi est déconseillé si $p+q$ est très "grand".

La deuxième fonction renvoie la même liste que la fonction précédente mais n'utilise pas d'urne virtuelle. Cette fonction est aussi complète que la première en terme d'information recueillie et elle est plus rapide à exécuter si $p+q$ est très "grand".

La troisième fonction retourne seulement le nombre de boules blanches obtenues et n'utilise ni urne virtuelle ni liste des résultats. Cette fonction qui simule une loi binomiale de paramètres $\left(n, \frac{p}{p+q}\right)$ peut suffire si on ne se préoccupe pas de l'ordre de sortie et elle est plus rapide à exécuter que les deux précédentes si n est très "grand".

```
import random as rd
def avecRemise1(p,q,n):
    urne = [1]*p + [0]*q
    resultats = []
    for _ in range(n):
        i = rd.randint(0, p+q-1)
        tirage = urne[i]
        resultats.append(tirage)
    return resultats
```

```
def avecRemise1bis(p,q,n):
    urne = [1]*p+[0]*q
    resultats = []
    for _ in range(n):
        b = rd.choice(urne)
        resultats.append(b)
    return resultats
```

```
def avecRemise2(p,q,n):
    resultats = []
    for _ in range(n):
        if rd.random() < p/(p+q):
            tirage = 1
        else:
            tirage = 0
        resultats.append(tirage)
    return resultats
```

```
def avecRemise3(p,q,n):
    cptB = 0 # compteur des boules blanches tirées
    for _ in range(n):
        cptB += rd.random() < p/(p+q)
    return cptB
```

On en déduit le code le plus compact pour simuler $\mathscr{B}(n,p)$:

```
def binomiale(n,p):
    cptS = 0 # compteur des succès
    for _ in range(n):
        cptS += rd.random() < p
    return cptS
```

2 Tirages sans remise

On considère une urne contenant p boules blanches et q boules noires dans laquelle on tire n boules **sans** remise.
On propose trois fonctions `sansRemise1(p,q,n)`, `sansRemise2(p,q,n)` et `sansRemise3(p,q,n)` qui simulent cette expérience.

La première fonction renvoie une liste de 0 (noir) et de 1 (blanc) dont l'ordre correspond à l'ordre des tirages. Cette fonction simule des tirages sans remise dans une urne virtuelle, elle est donc plus "proche" de l'expérience concrète mais son emploi est déconseillé si $p+q$ est très "grand".

La deuxième fonction renvoie la même liste que la fonction précédente et elle a remplacé l'urne virtuelle par deux compteurs. Cette fonction est aussi complète que la première en terme d'information recueillie et elle est plus rapide à exécuter si $p+q$ est très "grand".

La troisième fonction retourne seulement le nombre de boules blanches obtenues et n'utilise ni urne virtuelle ni liste des résultats. Cette fonction qui simule une loi hypergéométrique de paramètres $\left(p+q, n, \frac{p}{p+q}\right)$ peut suffire si on ne se préoccupe pas de l'ordre de sortie et elle est plus rapide à exécuter que les deux précédentes si n est très "grand".

```
import random as rd
def sansRemise1(p,q,n):
    urne = [1]*p + [0]*q
    resultats = []
    for _ in range(n):
        m = len(urne)
        i = rd.randint(0, m - 1)
        tirage = urne.pop(i)
        resultats.append(tirage)
    return resultats
```

```
def sansRemise1bis(p,q,n):
    urne = [1]*p+[0]*q
    resultats = []
    for _ in range(n):
        b = rd.choice(urne)
        urne.remove(b)
        resultats.append(b)
    return resultats
```

```
def sansRemise2(p,q,n):
    resultats = []
    cptBN = p+q # compteur des boules blanches et noires de l'urne
    cptB = p # compteur des boules blanches de l'urne
    for _ in range(n):
        if rd.random() < cptB/cptBN:
            tirage = 1
            cptB -= 1
        else:
            tirage = 0
        cptBN -= 1
        resultats.append(tirage)
    return resultats
```

```
def sansRemise3(p,q,n):
    cptBN = p+q # compteur des boules blanches et noires de l'urne
    cptB = p # compteur des boules blanches de l'urne
    for _ in range(n):
        cptB -= rd.random() < cptB/cptBN
        cptBN -= 1
    return p - cptB
```

On peut contracter le code de *sansRemise3* en utilisant le compteur de la boucle *for* à la place du compteur *cptBN* :

```
def sansRemise4(p,q,n):
    cptB = p # compteur des boules blanches de l'urne
    for k in range(n):
        cptB -= rd.random() < cptB/(p+q-k)
    return p - cptB
```