TP 3 : Simulations de variables aléatoires discrètes (1)

Bibliothèque : `import random as rd`

Les deux syntaxes incontournables pour simuler le hasard sont :
`rd.random()` renvoie un réel entre 0 et 1, et `rd.randint(k,n)` renvoie un entier entre k et n (inclus).

Exercice 1:

1. Simuler un lancer de dé équilibré.
2. Simuler un tirage d'une boule dans une urne contenant n boules numérotées de 1 à n.
3. Simuler un lancer de pièce équilibré.
4. Simuler un lancer de pièce dont la probabilité d'obtenir pile est p.

Exercice 2:

1. Simuler une loi de bernoulli de paramètre p.
2. Simuler une loi binomiale de paramètre n et p.
3. Simuler une loi géométrique de paramètre p.

Exercice 3:

1. On considère une urne contenant b boules blanches et n boules noires, et on effectue 3 tirages avec remise. Ecrire une fonction qui prend en argument b et n et renvoie la liste des couleurs des boules tirées.
   Essayer de proposer 3 syntaxes : avec rd.random(), avec rd.randint(), avec rd.choice()
2. Reprendre la question 1. lorsqu'on effectue 3 tirages sans remise.

Exercice 4:

On considère une urne contenant 20 boules numérotées de 1 à 20. On effectue 5 tirages sans remise dans cette urne. Ecrire une fonction qui renvoie la liste des numéros des boules tirées.
Proposer deux syntaxes : l'une avec rd.randint() et l'autre avec rd.choice().

Exercice 5:

Créer aléatoirement une chaîne de 10 nucléotides. *On proposera deux syntaxes.*

Exercice 6:

Soit une urne contenant p boules numérotées de 0 à $p-1$. On effectue n tirages avec remise.

1. Ecrire une fonction `tirage` de paramètres n, p qui donne en sortie une liste avec les valeurs obtenues.
2. Ecrire une fonction `boule` de paramètres n, p et i un entier, qui renvoie `True` si la boule i a été tirée, et `False` sinon.
3. Ecrire une fonction de paramètres n, p qui donne en sortie le nombre de boules différentes obtenues.
   bonus : trouver une variante à l'aide de la syntaxe `set(L)` où `L` est une liste.

Exercice 7: *pour s'entraîner*

Soit $n \in \mathbb{N}^*$. Une urne U_n contient n boules numérotées de 1 à n. On y effectue une succession de tirages d'une boule, en appliquant la règle suivante : si la boule tirée porte le numéro k, avant de procéder au tirage suivant, on enlève toutes les boules dont le numéro est supérieur ou égal à k.
On note X_n la variable aléatoire égale au nombre de tirages nécessaires pour vider l'urne U_n de toutes ses boules.

1. Compléter le programme suivant afin qu'il simule l'expérience.

```
n=int(input('n=?'))
x=------
while ------ :
    k=------
    n=------
    x=------
print(x)
```

2. Exécuter plusieurs fois le programme avec $n = 100$, puis changer la valeur de n. Que remarquez-vous ?

Exercice 8:

Soit $N \in \mathbb{N}^*$. On considère une urne contenant au départ 1 boule rouge et 1 boule verte. On effectue N tirages successifs de la façon suivante : on tire une boule de l'urne, puis on la remet avec une nouvelle boule de la même couleur. On note X_N le nombre de boules rouges après ces N tirages.
Ecrire une fonction python de paramètre d'entrée N qui renvoie la valeur de X_N.

Exercice 9:

On effectue des lancers successifs d'un dé honnête. Pour tout $n \in \mathbb{N}^*$, on note X_n la variable aléatoire égale au nombre de lancers effectués jusqu'à l'obtention du n^{ie} 6.
Ecrire une fonction d'argument n qui renvoie une réalisation de X_n.

Exercice 10:

On effectue des lancers d'une pièce donnant «Pile» avec la probabilité p, élément de $]0,1[$, et donnant «face» avec la probabilité $q = 1 - p$, les différents lancers étant supposés indépendants.
On note S_k le rang du $k^{\text{ème}}$ pile et T_k le rang d'apparition du dernier «pile» de la première série de k «piles» consécutifs.
Par exemple, si les lancers donnent $F_1, P_2, F_3, P_4, F_5, P_6, P_7, P_8$: S_1 et T_1 prennent la valeur 2, S_2 prend la valeur 4, T_2 prend la valeur 7, S_3 prend la valeur 6, T_3 prend la valeur 8, S_4 prend la valeur 7 et S_5 prend la valeur 8.

1. Compléter les lignes 7 et 9 de la fonction suivante pour qu'elle renvoie la valeur prise par S_k :

```
1. def simu(k,p):
2.     n,c = 0,0
3.    while c<k:
4.          n=n+1
5.          if --- :
6.              c = c+1
7. return ---
```

2. On souhaite que le script précédent affiche la valeur prise par T_k. Remplacer les lignes 5 et 6 par les suivantes, dûment complétées :

```
if ---:
    c = c+1
else:
    ---
```

Exercice 11:

On lance une pièce équilibrée et on note Z la variable aléatoire égale au rang du lancer où l'on obtient le premier "pile". Puis si Z a pris la valeur k ($k \in \mathbb{N}^*$), on remplit une urne de k boules numérotées 1, 2, $\cdots$, k, et on extrait au hasard une boule de cette urne. Soit X la variable aléatoire égale au numéro de la boule tirée.
Ecrire un programme `python` qui simule l'expérience ci-dessus, et affiche les valeurs de Z et X.

Exercice 12: *Marche aléatoire*

Une puce se déplace sur un axe gradué. A l'instant $t = 0$, la puce se trouve à l'origine (point d'abscisse 0). Puis si la puce est à l'abscisse k à l'instant $t = n$, alors à l'instant $t = n+1$, la puce sera à l'abscisse $k-1$ ou $k+1$ avec équiprobabilité. On arrête l'observation à l'instant t=200.

1. Ecrire un programme qui crée une liste permettant de stocker une trajectoire de la puce.
2. Exécuter plusieurs fois le programme précédent et observer les différentes trajectoires : regarder en particulier l'abscisse maximale, minimale etc. La variabilité est-elle grande ?

Exercice 13:

Soit $n \geq 2$ un entier. On dispose de n urnes numérotées de 0 à $n-1$, contenant chacune n boules. On répète n épreuves, chacune consistant à choisir une urne au hasard et à en extraire une boule. Créer une fonction d'argument n qui renvoie la liste contenant le nombre de boules de chaque urne à la fin de l'expérience.

Représentation des lois : diagrammes en bâtons et histogrammes
Bibliothèque : `import matplotlib.pyplot as plt`

A connaitre : soit X et Y deux listes (ou tableaux) de même taille .
Alors la syntaxe `plt.bar(X,Y,width=0.4,color='r')` crée un diagramme en bâtons où les bâtons sont positionnés aux abscisses de X, et sont de hauteur les ordonnées de Y.

Exemple : pour $X = [0,1,2]$ et $Y = [0.5, 1/3, 1/6]$ que donne `plt.bar(X,Y)` ?
Le dessiner sur votre feuille avant de vérifier avec python.

Le **diagramme en bâtons** (théorique) d'une loi est un diagramme en bâtons où X est la liste des valeurs prises, et Y la liste des probabilités d'apparition de chacune des valeurs.
Dans le cas où l'on sait simuler une loi, il peut être intéressant de faire un diagramme en bâtons empirique (empirique = résultant de simulations). Il faut alors commencer par faire beaucoup de simulations : puis on choisit pour X la liste des valeurs prises (en pratique), et pour Y la liste des fréquences d'apparition (ou nombres d'apparition) de chaque valeur prise.
Ces diagrammes en bâtons permettent une "visualisation" de la loi : valeurs les plus souvent prises, répartition des valeurs prises, moyenne, variance etc.

Exercice 14:

Créer le diagramme en bâtons théorique de la loi de bernoulli $\mathcal{B}(1/3)$, puis celui de la loi binomiale $\mathcal{B}(3, 0.5)$.
Commencer par écrire le tableau de ces deux lois sur votre feuille, pour ensuite introduire les X et Y correspondants.

Exercice 15:

1. Créer une liste L qui contient 100 lancers d'un dé équilibré.
2. Compléter la fonction suivante, qui prend en entrée une telle liste L et qui stocke le nombre d'apparitions de chaque valeur $i \in [\![1, 6]\!]$ dans une autre liste :
```
def apparitions(L):
    nbre=6*[0]
    for i in L :
        nbre[...] =......
    return nbre
```
3. Modifier la fonction précédente afin qu'elle renvoie la fréquence d'apparition de chaque valeur $i \in [\![1, 6]\!]$.
4. Créer le diagramme en bâtons empirique correspondant.
5. Superposer le diagramme en bâton théorique de la loi $\mathcal{U}([\![1, 6]\!])$. Constater.
   on pourra refaire plusieurs fois l'expérience, et choisir 1000 ou 10 000 ou ... lancers.
   Remarque : La loi faible des grands nombres permettra de valider mathématiquement(en fin d'année) ce que l'on peut constater informatiquement dans cette question à savoir que lorsque le nombre de simulations tend vers $+\infty$, la fréquence d'apparition (d'un événement) converge vers la probabilité d'apparition de cet événement.

Exercice 16:

Créer le diagramme empirique de la loi $\mathcal{B}(20, 0.6)$ puis de la loi $\mathcal{G}(0.3)$.

$\longrightarrow$ en profiter pour faire varier les paramètres des lois précédentes, puis retenir l'allure de ces lois usuelles (forme de cloche ou non, visualiser la moyenne et la variance).

Remarque : il existe une commande "toute prête" `plt.hist(S,bins=20)` où S est la liste des simulations et bins est le nombre de classes (= bâtons).
Cette commande (qui n'apparait pas dans le mémo pour l'oral) crée directement l'histogramme des simulations S SANS avoir à créer au préalable la liste des fréquences d'apparition : autrement dit, cette commande "fait tout" de manière cachée. Python crée bins rectangles de même base et de hauteur proportionnelle au nombre de simulations qui ont une valeur dans la base correspondante.
Mais comme tout est automatique, les bases choisies ne sont pas toujours en correspondance avec les valeurs prises entières (python prend la plus petite et la plus grande valeur de S, et subdivise cet intervalle en bins intervalle) .

Variante : `plt.hist(S,bins=20,density=True)` permet d'avoir en ordonnées une hauteur reliée aux fréquences d'apparition (pour comparer à la loi théorique, par exemple).

Exemple : Tester cette commande lorsque $S = [2, 1, 1, 1, 1, 2, 3, 1, 1, 2, 2, 3, 1, 2, 1, 2, 2, 1, 1]$.
Ne pas hésiter à tester cette commande dans les deux exercices précédents.

Exercice 17:

On dispose de 2 dés équilibrés, un rouge et un bleu. A chaque tour, on lance les deux dés. On appelle X le nombre de lancers nécessaires du dé rouge pour que le dé rouge amène un "six" et Y le nombre de lancers nécessaires du dé bleu pour que le dé bleu amène un "six". On définit $S = min(X, Y)$, le nombre de tours nécessaires pour que l'un des deux dés amène un "six".

1. Ecrire une fonction Python qui renvoie une réalisation de S.
2. Créer une fonction de paramètre d'entrée N traçant le diagramme en bâtons empirique de S ou l'histogramme de S après N simulations.
3. La loi de S semble-t-elle être usuelle ? Le justifier alors mathématiquement. *On pourra commencer par regarder pour tout $k \in \mathbb{N}$, $P(S > k)$ avant d'en déduire la loi de S.*

Exercice 18:

On effectue des tirages sans remise dans une urne U qui contient initialement $N-1$ (avec $N \geq 3$) boules blanches et 1 boule noire, jusqu'à l'obtention de la boule noire. Soit X_N la var. égale au nombre de tirages effectués.

1. Ecrire une fonction d'en-tête `def tirages(N):` qui renvoie une simulation de X_N.
2. Créer une fonction de paramètres d'entrée N et n traçant le diagramme en bâtons empirique de X_N après n simulations. *Variante :* traçer l'histogramme de X_N après n simulations.
3. Que peut-on conjecturer sur la loi de X_N ? Vérifier alors mathématiquement la conjecture.