Réflexe :
Importer les librairies nécessaires au calcul scientifique, à la simulation de l'aléatoire et à la représentation graphique.

Entrée[ ]:
```
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
```

TP Python 16 : Variables aléatoires dicrètes

Correction

Ce TP est un cours à part entière :

- la première et la deuxième parties consistent en des rappels de cours, ainsi que des exercices de bases qu'il convient de maîtriser à 100%.
- La troisième partie est nouvelle et porte sur la simulation des lois discrètes usuelles : elle doit faire l'objet d'un **travail intensif** car elle est au coeur du programme de probabilités en ECG.
- La quatrième partie traite des diagrammes en bâtons : il est très fréquent de voir ce type de diagramme dans les sujets de concours... A bon entendeur!
- La cinquième partie comporte des exercices d'entraînement qu'il sera bon de revoir pour la deuxième année. Ils sont à faire en autonomie et ne sont pas prioritaires.

Partie I - Rappels : simulation d'une variable aléatoire avec `rd.random()`

Commençons par quelques rappels.

Soit $(\Omega, \mathcal{P}(\Omega))$ un espace probabilisable. On appelle variable aléatoire réelle (ou v.a.r) sur $(\Omega, \mathcal{P}(\Omega))$ toute application

$$X : \Omega \to \mathbb{R}.$$

L'ensemble $X(\Omega)$ s'appelle l'univers image de X. C'est l'ensemble des valeurs prises par X.

Une variable aléatoire est caractérisée par sa loi.

- En pratique, dans le cas d'un univers Ω discret (fini ou infini), la loi d'une variable aléatoire sera la donnée de tous les couples $(x, \mathbb{P}(X = x))$ avec $x \in X(\Omega)$.
- La loi sera la principale information dont on disposera sur une variable aléatoire, l'univers Ω étant le plus souvent implicite et non précisé.

Typesetting math: 100%

Avant de simuler une variable aléatoire, il est important d'avoir **compris comment simuler un évènement de probabilité p.**

La commande `rd.random()` renvoie un nombre réel aléatoire entre 0 et 1 suivant la loi uniforme continue $U([0;1])$.

Plus précisément, si $p \in [0,1]$, la probabilité que le nombre renvoyé soit inférieure ou égale à p (ou dans un intervalle de longueur p) vaut exactement p.

On identifie et décide de représenter donc un évènement de probabilité p par le fait que le nombre aléatoire renvoyé soit inférieur ou égal à p (ou dans un intervalle de longueur p).

Exercice 1

Le score d'un joueur joueur lors d'un lancer de fléchette est modélisé par une variable aléatoire X telle que $X(\Omega) = \{0, 2, 5, 10\}$ et

$$\mathbb{P}(X=0) = \frac{1}{5}, \quad \mathbb{P}(X=2) = \frac{1}{2}, \quad \mathbb{P}(X=5) = \frac{1}{5}, \quad \text{et} \quad \mathbb{P}(X=10) = \frac{1}{10}.$$

1. Calculer l'espérance et la variance de X.
2. Découper l'intervalle $[0;1]$ en quatre intervalles où pourrait tomber un nombre aléatoire généré avec `rd.random()` dans le but de simuler X. Écrire alors une fonction `simul_X()` qui renvoie une simulation de la variable £X£.

Entrée[ ]:

```
p0 = 1/5
p2 = 1/2
p5 = 1/5
p10 = 1/10

EX = 0*p0 + 2*p2 + 5*p5 + 10*p10
#on calcule l'esperance de X^2 à l'aide de la formule de transfert
EX2 = 0*p0 + 4*p2 + 25*p5 + 100*p10
# on peut calculer la variance
VX = EX2 -(EX)**2

print("esperance = ",EX,' variance = ', VX)
```

Typesetting math: 100%

Entrée[ ]:

```
def simul_X():

    p = rd.random()

    # 50% des valeurs de [0,1] sont dédiées à l'attribution de 2
    # on dédie donc l'intervalle [0,1/2] à l'attribution de 2
    if p<=1/2 :
        return 2

    # on veut qu'ensuite, 10% des valeurs de [0,1] soient dédiées à l'attr
    # on dédie donc l'intervalle [1/2;1/2+1/10]=[1/2;3/5] à l'attribution
    if 1/2<p <=3/5 :
        return 10

    # on veut qu'ensuite, 20% des valeurs de [0,1] soient dédiées à l'attr
    # on dédie donc l'intervalle [3/5; 3/5+1/5]=[3/5;4/5] à l'attribution
    if 3/5<p<=4/5 :
        return 0

    #enfin, on veut que 20% des valeurs de [0,1] soient dédiées à l'attrib
    # on dédier donc l'intervalle restant, [4/5;1] à l'attribution de 5
    if 4/5<p<=1 :
        return 5


    # On pouvait aussi commencer par attribuer 10, puis 5, puis 2, puis 0,
    # Le choix du découpage revient à vous : tant que l'idée est comprise
    # vous-vous l'appropriez, c'est bon!

simul_X()
```

3. Que fait le programme suivant?

Entrée[ ]:

```
def sample_X(n):

    S = []

    for k in range(n):
        S.append(simul_X())

    return S

print(np.mean(sample_X(1000)))
```

Correction

La fonction `sample_X` prend en argument un entier naturel n. Elle initialise une liste `S` vide, puis effectue une boucle allant de 0 à n : à chaque itération, elle simule une **réalisation** de la variable aléatoire X construite au préalable, et stocke cette réalisation (qui vaut soit $0$0, soit 02, soit 5, soit 10) dans la liste `S` . Enfin, elle renvoie cette liste.

Cette fonction génère donc une liste comportant n réalisations de la variable aléatoire X : on dit qe c'est un n**-échantillon**.

La dernière commande **affiche** la **moyenne** des 1000 valeurs stockées dans la liste obtenue

Typesetting math: 100%

par la commande `sample_X(1000)` .

Un n−échantillon d'une variable aléatoire X est un n−uplet dont chaque composante est une variable aléatoire X_i de même loi que X et qui sont toutes mutuellement indépendantes.

La moyenne empirique des réalisations d'un n−échantillon (que l'on obtient grâce à la commande `np.mean( )`) fournit alors une estimation (valeur approchée) de l'espérance de X (quand celle-ci existe).

Exercice 2
On désigne par n un entier naturel supérieur ou égal à 2. On lance n fois une pièce équilibrée (c'est-à-dire donnant Pile avec la probabilité $1/2$, les lancers étant supposés indépendants.

On note Z la variable aléatoire qui vaut 0 si l'on n'obtient aucun Pile pendant ces n lancers et qui, dans le cas contraire, prend pour valeur le rang du premier pile.

1. Compléter la fonction suivante permettant de simuler la variable aléatoire Z.

Entrée[ ]:
```
def simul_Z(n) :

    for i in range(1,n+1):
        if rd.random()<=1/2 :
            return i

    return 0

# Remarque : lorsque vous testez votre fonction pour n=1, vous êtes en tra
# variable aléatoire X suivant une loi de Bernoulli de paramètre p=0.5.
simul_Z(5)
```

2. Réécrire cette fonction à l'aide d'une boucle `while` ;

Entrée[ ]:
```
def simul_Z_bis(n):

    rangpile = 0

    while rd.random()>=1/2 and rangpile <n:
        rangpile = rangpile + 1

    return rangpile

simul_Z_bis(1)

```

Partie II - Lois usuelles discrètes finies

II.1) Lois uniforme U(⟦m,n⟧)

Typesetting math: 100%

Soit n, m deux entiers naturels, $m<n$. On dit que X suit une loi uniforme sur $ \text{⟦m,n⟧}$, si $X(\Omega) =\text{⟦m,n⟧}$, et que
$$\forall k \in \text{⟦m,n⟧},\quad \mathbb{P}(X=k)=\dfrac{1}{n-m+1}.$$

De plus, $\mathbb{E}[X]=\dfrac{n+m}{2}$ et $\mathbb{V}[X] = \dfrac{(n-m+1)^{2}-1}{12}$.

La commande `rd.randint(m,n+1)` permet de simuler une telle variable.

Exercice 3
On dispose de $n\geq 1$ urnes, numérotées de 1 à n. Pour chaque $k \in \text{⟦1,n⟧}$, l'urne k est composée de boules numérotées de 1 à k. On choisit une urne au hasard puis on tire une boule dans cette urne et on note X_n la variable aléatoire qui prend la valeur du numéro de la boule piochée.

Écrire une fonction `simul_X(n)` qui simule X_n.

Entrée[ ]:
```
def simul_X(n):

    # choix de l'urne
    k = rd.randint(1,n+1)

    # on tire une boule parmis les k boules de l'urne numéro k
    boule = rd.randint(1,k+1)

    return boule
```

II.2) Loi de Bernoulli $\mathcal{B}(p)$

On dit que X suit une loi de Bernoulli de paramètre $p \in [0,1]$, ce qu'on note $X \hookrightarrow \mathcal{B}(p)$, si $X(\Omega) =\{0,1\}$, et que
$$\mathbb{P}(X=1)=p \qquad \text{et}\qquad \mathbb{P}(X=0)=1-p.$$

De plus, $\mathbb{E}[X]=p$ et $\mathbb{V}[X] = p(1-p)$.

Exercice 4
Écrire, à l'aide de la commande `rd.random()` une fonction `Bernoulli(p)` permettant de simuler une variable $X \hookrightarrow \mathcal{B}(p)$.

Typesetting math: 100%

Entrée[ ]:

```
def Bernoulli(p):

    if rd.random()<=p :
        return 1
    else :
        return 0

# on affiche 40 réalisations d'une va suivant une loi de Bernoulli de para
# on veut "vérifier" (à la louche) que 1 apparaît plus souvent que 0...
sample = [Bernoulli(0.7) for k in range(40)]
sample
```

II.3) Loi binomiale $\mathcal{B}(n,p)$

On dit que X suit une loi binomiale de paramètres $(n,p) \in \mathbb{N} \times [0,1]$, ce qu'on note $X \hookrightarrow \mathcal{B}(n,p)$, si $X(\Omega) =\text{⟦0,n⟧}$, et que $$\forall k \in \text{⟦0,n⟧},\quad \mathbb{P}(X=k)=\binom{n}{k}p^{k}(1-p)^{n-k}.$$

De plus, $\mathbb{E}[X]=np$ et $\mathbb{V}[X] = np(1-p)$.

La commande `rd.binomial(n,p)` permet de simuler une telle variable.

Remarque : On peut alors très rapidement simuler une variable aléatoire suivant une loi de Bernoulli de paramètre p à l'aide de la commande `rd.binomial(1,p)` .

Exercice 5
Écrire, à l'aide de la commande `rd.random()` et d'une boucle `for` , une fonction `Binomiale(n,p)` permettant de simuler une variable $X \hookrightarrow \mathcal{B}(n,p)$.

Entrée[ ]:

```
def Binomiale(n,p):

    X = 0

    for k in range(1,n+1):
        if rd.random()<=p:
            X = X + 1
    return X

# on affiche 40 réalisations d'une va suivant une loi Binomiale de paramèt
# cela signifie qu'on a de faible chance de cumuler les succès sur les cin
# on veut "vérifier" (à la louche) qu'on obtient bien des petites sommes (
# A contrario, si l'on changeait pour p = 0.7, on aurait beaucoup de succè
# à obtenir des sommes très proches de 5!

sample = [Binomiale(5,0.2) for k in range(40)]
sample
```

Partie III - Lois usuelles discrètes à support infini

III.1) Loi géométrique $\mathcal{G}(p)$

On dit que X suit une loi géométrique de paramètres $p \in]0,1[$, ce qu'on note $X

Typesetting math: 100%

\hookrightarrow \mathcal{G}(p)$, si $X(\Omega) =\mathbb{N}^*$, et que
$$\forall k \in \mathbb{N}^*,\quad \mathbb{P}(X=k)=(1-p)^{k-1}p.$$

De plus, $\mathbb{E}[X]=\dfrac{1}{p}$ et $\mathbb{V}[X] = \dfrac{1-p}{p^2}$.

Une loi géométrique correspond au **temps d'attente du premier succès** (de probabilité p) lors de répétitions (infinies) **indépendantes** d'une même épreuve de Bernoulli.

La commande `rd.geometric(p)` permet de simuler une telle variable.

Exercice 6
Écrire, à l'aide de la commande `rd.random()` et d'une boucle `while`, une fonction `Geometrique(p)` permettant de simuler une variable $X \hookrightarrow \mathcal{G}(p)$.

Entrée[ ]:
```
def Geometrique(p):

    count = 0

    while rd.random()>p :
        count = count +1

    return count

# On a une faible chance d'obtenir un succès : il va falloir répéter longt
# un premier succès. On relance 40 fois la même expérience pour visualiser
sample_faible = [Geometrique(0.1) for k in range(40)]
print(sample_faible)

# On a une forte chance d'obtenir un succès : on ne répète pas beaucoup l'
# un premier succès. On relance 40 fois la même expérience pour visualiser
sample_fort = [ Geometrique(0.8) for k in range(40)]
print(sample_fort)
```

III.2) Loi de Poisson $\mathcal{P}(\lambda)$

On dit que X suit une loi de Poisson de paramètres $\lambda>0$, ce qu'on note $X \hookrightarrow \mathcal{P}(\lambda)$, si $X(\Omega) =\mathbb{N}$, et que
$$\forall k \in \mathbb{N},\quad \mathbb{P}(X=k)=\dfrac{\lambda^k}{k!}e^{-\lambda}.$$

De plus, $\mathbb{E}[X]=\lambda$ et $\mathbb{V}[X] = \lambda^2$.

La commande `rd.poisson(m)` permet de simuler une variable $X \hookrightarrow \mathcal{P}(m)$, $m>0$.

Exercice 7
Chaque jour, le nombre de personnes subissant un test de dépistage dans une certaine pharmacie est modélisé par une variable X suivant une loi de Poisson de paramètre $m = 5$. Chaque test a une certaine probabilité $p = \dfrac{1}{10}$ d'être positif et les tests sont indépendants les uns des autres. On note N le nombre de tests positifs un jour donné.

1. Écrire une fonction permettant simuler la variable N.
2. Créer un échantillon de taille 1000 de cette variable aléatoire et le comparer, avec un histogramme, à un échantillon de même taille d'une loi de Poisson de paramètre $m\times p = 0.5$. Conjecturer.

On pourra utiliser la fonction `hist()` du module `pyplot` de la bibliothèque

Typesetting math: 100%

matplotlib à l'aide de la commande `plt.hist(....)` .

3. (*Au brouillon*) En commençant par expliciter $\mathbb{P}_{X=n}(N = k)$ (en faisant attention sur le couplage des indices n et k), démontrer le résultat précédemment conjecturé.

Entrée[ ]:

```
def pharmaplus(p):

    # Pour commencer, on a besoin d'avoir le nombre de personnes se rendar
    # ce nombre est la v.a.r X qui suit une loi de Poisson de paramètre 5

    n = rd.poisson(5)

    # on veut compter les tests positifs parmis les n tests réalisés ce jo
    # N suit une loi binomiale de paramètre (n,p) : on compte le nombre de
    # sur n épeuves de Bernoulli (faire un test)
    N = rd.binomial(n,p)

    return N


p = 1/10
m = 5

sample_pharma = [pharmaplus(p) for k in range(1000)]
sample_poisson = [rd.poisson(m*p) for k in range(1000)]

plt.clf()
plt.hist([sample_pharma, sample_poisson], color = ['purple','pink'], densi
plt.legend(["Poisson m conditionnée à binomiale p", "Poisson mxp"])
plt.title("Comparaison de deux échantillons")
plt.show()

```

Partie IV - Diagrammes à bâtons

La commande `bar(x, y)` du package `matplotlib.pyplot` permet de représenter un **diagramme à bâtons**.

Plus précisément, en notant les arguments $x = [x_1, ..., x_n]$ et $y = [y_1, ..., y_n]$ (deux listes de mêmes longueurs) elle permet de positionner un bâton de hauteur y_i en face de la valeur x_i.

À vous de jouer
Exécuter le programme ci-dessous. Commenter chacune des lignes du programme à l'aide du symbole '#'.

Typesetting math: 100%

Entrée[ ]:
```
p=1/2

K = [ k for k in range(1,10) ]
p_th = [ p*(1-p)**(k-1) for k in K ]

plt.clf()
plt.bar(K, p_th )
plt.show()
```

→Cliquez-ici pour la réponse ←

Exercice 8
Représenter le diagramme à bâtons dont les hauteurs des bâtons sont les valeurs théoriques de la loi uniforme $U(\text{⟦1,n⟧})$, $n \in \mathbb{N}^*$.

Entrée[ ]:
```
N = [n for n in range(1,20)]

p_th = [1/n for n in N]

plt.clf()
plt.bar(N,p_th)
plt.title('Valeurs théoriques de la loi uniforme')
plt.xlabel("n")
plt.ylabel("P(X=k) = 1/n, pour tout 1<=k<=n")
plt.show()
```

Attention
On ne fera pas de confusion avec la commande `hist(U)` (du même package) qui représente l'histogramme des valeurs d'une liste `U`.

- Il est souvent très efficace d'utiliser de tels diagrammes pour **conjecturer sur la loi suivie**.

  Par exemple, si tous les bâtons ont à peu près la même hauteur, il est raisonnable de penser que la loi est uniforme!

- Nous verrons en fin d'année, et vous le reverrez l'année prochaine, les diagrammes à bâtons sont aussi pratiques pour **visualiser(ou conjecturer graphiquement) une convergence en loi**.

  Par exemple, les commandes du programme du précédent "À vous de jouer" permettent d'afficher les bâtons dont les hauteurs sont les valeurs théoriques de la loi géométrique de paramètre $p = \dfrac{1}{2}$ (pour les 10 premiers entiers).

L'idée est alors de **créer un n−échantillon** (avec n grand) et de créer un diagramme à bâtons dont la liste x en **abscisses** est celles des **valeurs obtenues par simulation** et la liste y en **ordonnées** celle des **fréquences de chaque valeur dans l'échantillon**.

Typesetting math: 100%

Exercice 9

Une urne contient des boules numérotées de 1 à n, indiscernables au toucher. On effectue alors n tirages sans remise dans l'urne et on note, pour tout $k \in \text{⟦1,n⟧}$, X_k la variable aléatoire qui prend la valeur de la boule obtenue au k−ième tirage.

1. Écrire une fonction Python `def simul_X` qui prend en argument deux entiers naturels non nuls k et n, et qui renvoie une simulation de X_k.

Entrée[ ]:

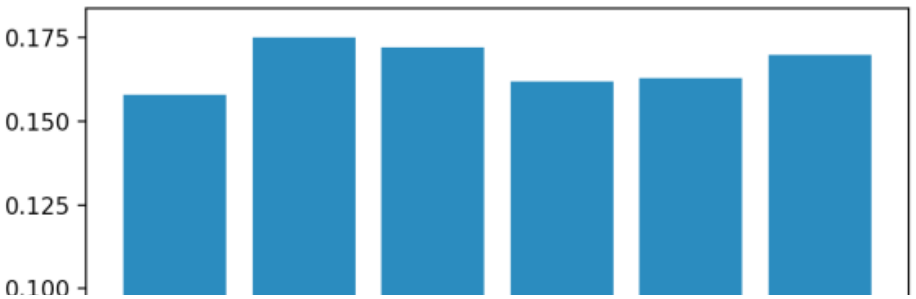
```
1  def simul_X(k,n):
2
3      count  = 0
4      urne = [ i for i in range(1,n+1)]
5
6      while count <k :
7          # print(urne)
8          numero = rd.randint(1,len(urne)+1)
9          boule = urne[numero-1]
10         #print(boule)
11         urne.remove(boule)
12         count = count + 1
13
14     return boule
15
16 simul_X(7,11)
17
```

2. On ajoute les commandes suivantes. Expliquer ce qu'elles font. On insistera que la commande de la ligne 7.

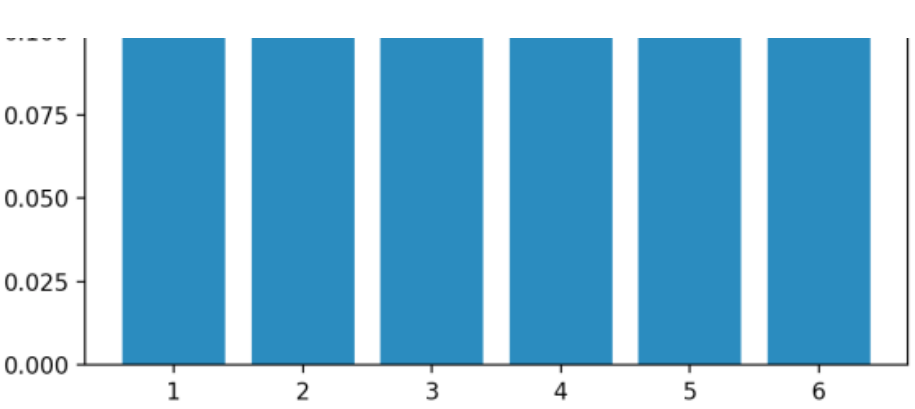
Entrée[ ]:

```
1  n=6
2  k=4
3  echantillon = [simul_X(k,n) for j in range(1000) ]
4  freq = [0]*6
5
6  for j in range(1000):
7      freq[echantillon[j]-1] += 1
8
9  freq = [freq[j]/1000 for j in range(n) ]
10
11 plt.clf()
12 plt.bar([j for j in range(1,n+1)],freq)
13 plt.show()
```

3. Leur exécution permet l'affichage suivant:



Typesetting math: 100%



Émettre une conjecture.

Pour une urne contenant $n=6$ boules, la variable aléatoire X_{4} renvoyant la valeur de la boule au 4-ième tirage semble suivre une loi uniforme de probabilité $$\mathbb{P}(X_{4} =j)=0.1625, \quad \forall j \in \{1,\ldots,6\}.$$
(La valeur 0.165 est obtenue en moyennant 0.150 et 0.175 qui sont les valeurs minimales et maximales des hauteurs affichées dans le diagrammes en bâtons. On vérifie que $6*0.165 =0.99 \simeq 1$. C'est une valeur approchée, mais on doit tout de même obtenir que la somme des $\mathbb{P}(X_{4}=j)$ se rapproche fortement de $$!)

Exercice 10

On lance indéfiniment une pièce donnant Pile avec la probabilité p et Face avec la probabilité $q =1-p$. On suppose que $p \in]0,1[$ et on admet que les lancers sont mutuellement indépendants.

Pour tout entier naturel $k \geq 2$, on dit que le k-ième lancer est un **changement** s'il amène un résultat différent de celui du $(k-1)$-ième lancer.

On note P_k (resp. F_k) l'événement : "on obtient Pile (resp. Face) au k_ième lancer".

Pour tout entier naturel $n \geq 2$, on note X_n la variable aléatoire égale au nombre de changements survenus durant les n premiers lancers.

1. Écrire une fonction d'en-tête `simul_X` prenant en argument un entier $n \geq 2$ et un réel $p \in]0,1[$ et qui renvoie une simulation de X_n.
2. Quelles commandes peut-on ajouter pour représenter le diagramme à bâtons des fréquences des valeurs prises par un 1000−échantillon de X_n ?
3. Représenter le diagramme à bâtons susmentionné dans le cas où $p = \dfrac{1}{2}$, avec $n = 3$ puis avec $n = 4$, et enfin avec $n = 5$. Représenter en parallèle des batôns correspondant aux hauteurs théoriques de la loi $\mathcal{B}(n-1, 0.5)$. Émettre une conjecture.

Typesetting math: 100%

Entrée[ ]:

```
def simul_X(n,p):

    changement = 0
    lancer1 = rd.random()
    lancer2 = 0

    for k in range(n-1):
        lancer2 = rd.random()

        print('On a effectué',k+2,'lancers, avec lancer1=',lancer1,'lancer
        if (lancer1<=p) and (lancer2>p):
            changement = changement +1

        lancer1 = lancer2

    return changement

simul_X(4,0.6)
```

Partie V - Exercices d'entraînement en autonomie

Exercice A
On considère la variable aléatoire X telle que $X(\Omega)= \left\{-1,0,1 \right\}$, et

$$\mathbb{P}(X=-1) = \mathbb{P}(X=1) = \dfrac{1}{4}, \quad \mathbb{P}(X=0) = \dfrac{1}{2}.$$

Écrire une fonction Python permettant de simuler X.

Entrée[ ]:

```
def exoA():
    val = rd.random()
    if val<=0.5:
        return 0
    else:
        if 0.5<val<=0.75:
            return 1
        else:
            return -1

echantillon = [exoA() for k in range(40)]
echantillon
```

Exercice B
Une urne contient 1 boule bleue, 2 boules blanches et 3 boules rouges. On pioche successivement et avec remise 3 boules dans cette urne. Si on obtient les 3 couleurs, on marque 2 points, si on obtient une seule couleur 1 point et 0 point sinon. On note X la variable aléatoire qui prend la valeur du nombre de points marqués sur une partie.

1. Déterminer la loi de X.
2. Écrire une fonction Python qui simule X.
3. Calculer $\mathbb{E}[X]$ puis $\mathbb{V}[X]$.

Typesetting math: 100%

Entrée[ ]:

```
def exoB():
    pbleue = 1/6
    prouge = 3/6
    pblanche = 2/6


    # On effectue trois lancers et on stocke les valeurs obtenue dans
    # la liste "lancer"
    lancer = []
    for k in range(3):
        val = rd.random()
        if val<=pbleue :
            lancer.append('Bleue')
        else :
            if pbleue<val<=pbleue+prouge:
                lancer.append('Rouge')
            else :
                lancer.append('Blanche')

    # on affiche la liste pour vérifier que le code marche
    print(lancer,'\n')


    # Maintenant, on distingue les cas :
    # si on obtient toujours la même couleur, on renvoie 1
    if (lancer[0]==lancer[1]) and (lancer[1]==lancer[2]):
        return 1
    else:
        # si on obtient trois couleurs différentes on renvoie 2
        if (lancer[0]!=lancer[1]) and (lancer[1]!=lancer[2]) and (lancer[0
            return 2
        # sinon, on renvoie 0
        else:
            return 0

    # Note: A!=B!=C ne marche pas en Python pour tester que trois valeurs


exoB()

```

Exercice C

Soit N un entier supérieur ou égal à 3. On considère une urne qui contient $(N-1)$ boules blanches et une seule boule noire.
On effectue des tirages sans remise jusqu'à l'obtention de la boule noire et on note X la variable aléatoire qui prend pour valeur le nombre de tirages nécessaires.

1. Écrire une fonction `simul_X` prenant en argument un entier naturel $N\geq 3$, et qui renvoie une simulation de la variable X.

Typesetting math: 100%

Entrée[ ]:

```
def simul_X(N):
    # On vérifie que N est bien supérieur ou égal à 3
    if N < 3:
        print("N doit être supérieur ou égal à 3")
        return None

    # On crée une liste représentant les boules dans l'urne :
    # 0 = boule blanche, 1 = boule noire
    urne = [0] * (N - 1) + [1]

    # On mélange les boules pour que la position de la noire soit aléatoir
    rd.shuffle(urne)

    # On tire les boules une par une jusqu'à obtenir la noire
    tirages = 0
    for boule in urne:
        tirages += 1
        if boule == 1:
            return tirages  # On retourne le nombre de tirages effectués
```

2. Exécuter le programme ci-dessous. Comparer avec le diagramme théorique de l'exercice 8. Commenter, conjecturer, démontrer.

Entrée[ ]:

```
N = 5 # on fera varier les valeurs de N
freq_val_X = np.zeros(N)

for k in range(10000):
    i = simul_X(N)
    freq_val_X[i-1] += 1


freq_val_X = freq_val_X/10000

plt.clf()
plt.bar(range(1,N+1),freq_val_X)
plt.show()
```

On conjecture au vu du diagramme en bâtons obtenu que la variable X suit une loi uniforme : en effet, les bâtons sont de hauteur quasi égales, et on a bien $5 \times 0.2 = 1$.

Exercice D

On effectue une succession infinie de lancers indépendants d'une pièce donnant Pile avec probabilité p. On note $q = 1 - p$.
On dit que la première série est de longueur $n\geq1$ si les n premiers lancers ont amené le même côté de la pièce et le $(n+1)$-ième l'autre côté. De même la deuxième série commence au lancer suivant la fin de la première série et se termine au lancer précédant un changement de côté.

Par exemple si les lancers donnent les résultats F F P P P P P F F F P P ... alors la première série est de longueur 2 et la deuxième est de longueur 6.

Soient X_1 et X_2 les variables aléatoires égales aux longueurs de la première et

Typesetting math: 100%

deuxième série.

1. Compléter la fonction suivante, prenant en argument la probabilité p d'obtenir Pile et permettant de simuler la variable aléatoire X_1 .

Entrée[ ]:

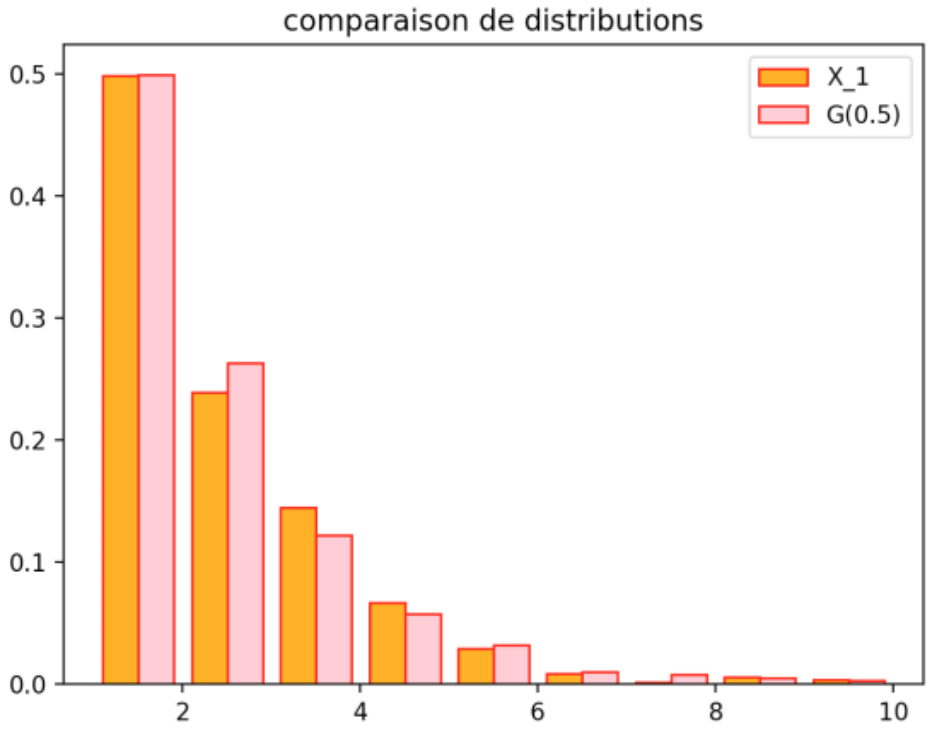
```
def simul_X1(p) :

    X=1
    t_old = rd.binomial(1,p)
    t_new = rd.binomial(1,p)

    while t_old!=t_new :
        X = X+1
        t_old = t_new
        t_new = rd.binomial(1,p)

    return X
```

2. Exécuter le code suivant, qui affiche la figure reproduite ci-après. Interpréter la figure obtenue.



Entrée[ ]:

```
p=0.5
N=1000

x1 = [ simul_X1(p) for k in range(N) ]
x2 = [ rd.geometric(p) for k in range(N) ]

plt.clf()
plt.hist([ x1,x2 ], color = ['orange','pink'] , density = True, label = ['
plt.title('comparaison de distributions')
plt.legend()
plt.show()
```

Typesetting math: 100%

3. Déterminer au brouillon la loi de X_1. Retrouver le résultat conjecturé à la question précédente. Montrer qu'elle admet une espérance que l'on explicitera.
4. Écrire une fonction `simul_X2` prenant en argument un réel $p \in [0,1]$ permettant de simuler la variable X_2 .
5. Déterminer, pour $(k,j) \in \mathbb{N}^{*}\times \mathbb{N}^{*}$, $\mathbb{P}(X_1 = k \cap X_2 = j)$. En déduire, à l'aide de la formule des probabilités totales, la loi de X_2.

La question 3 est laissée en exercice. Attention! Le support de X_1 est infini : pour justifier que X_1 admet une espérance, il faut faire attention...

Entrée[ ]:

```
def simul_X2(p) :

    X= simul_X1(p) + 1
    print("la première série s'est arrêtée au lancer",X-1," on reprends au
    t_old = rd.binomial(1,p)
    t_new = rd.binomial(1,p)

    while t_old!=t_new :
        X = X+1
        t_old = t_new
        t_new = rd.binomial(1,p)


    return X

simul_X2(0.2)
```

La question 5 est laissée en exercice.

Typesetting math: 100%