Python 2
Simulations de variables aléatoires discrètes

L'objectif est l'utilisation de **Python** dans la modélisation d'expériences et variables aléatoires. Nous aurons besoin de la bibliothèque **numpy.random** à importer ainsi :

```
import numpy.random as rd
```

Nous aurons également besoin, pour les graphiques, d'importer :

```
import matplotlib.pyplot as plt
```

Pour représenter un histogramme de données contenues dans la liste **Lvaleurs** en fonction de la listes de bornes **Lbornes**, on utilise :

```
plt.hist(Lvaleurs,Lbornes)
```

Après la création de l'histogramme, il faut rajouter la ligne **plt.show()** pour le représenter.
Nous aurons besoin de la commande **rd.random()** qui renvoie un réel aléatoire de $]0;1[$

Petite remarque
Par défaut, il s'agit d'un histogramme d'effectifs. Pour obtenir un histogramme de fréquences, on ajoute **density=True** en option dans la commande **plt.hist()**.

I Simuler les lois binomiale et géométrique avec RANDOM

Commençons par des rappels du cours de probabilités en complétant le tableau ci-dessous :

Nom & notation	$X(\Omega)$	Loi de probabilité	Contexte
uniforme sur $[\![1;n]\!]$ $n \in \mathbb{N}^*$ $X \hookrightarrow \mathscr{U}([\![1;n]\!])$			
de Bernoulli de paramètre p $p \in]0;1[$ $X \hookrightarrow \mathscr{B}(p)$			
binomiale de paramètres n et p $n \in \mathbb{N}^*$ et $p \in]0;1[$ $X \hookrightarrow \mathscr{B}(n;p)$			
géométrique de paramètre p $p \in]0;1[$ $X \hookrightarrow \mathscr{G}(p)$			
de Poisson de paramètre λ $\lambda > 0$ $X \hookrightarrow \mathscr{P}(\lambda)$			

Que permet de faire le programme suivant ?

```python
import numpy.random as rd

def bernoulli(p):
    r=rd.random()
    if r<p:
        return 1
    else :
        return 0
```

I.1 Loi binomiale

1. En utilisant la fonction donnée ci-dessus, écrire une fonction **liste_bernoulli** prenant en arguments d'entrée un entier naturel n et un réel p de $]0;1[$ et renvoyant une liste de n réalisations indépendantes d'une variable aléatoire suivant une loi $\mathscr{B}(p)$.
2. En déduire une fonction **binomiale** prenant en arguments d'entrée un entier naturel n et un réel p de $]0;1[$ et renvoyant une réalisation d'une variable aléatoire suivant la loi $\mathscr{B}(n;p)$.
3. Proposer une autre fonction permettant de simuler une réalisation d'une variable aléatoire suivant la loi $\mathscr{B}(n;p)$ qui n'utilise pas les fonctions précédentes.

```python
import numpy.random as rd
import matplotlib.pyplot as plt

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

def liste_bernoulli(n,p):
    return [bernoulli(p) for k in range(n)]

def binomiale(n,p):
    return sum(liste_bernoulli(n,p))

def binomiale_bis(n,p):
    nb_succes=0
    for k in range(n): #ou range(0,n)
        if rd.random()<p:
            nb_succes+=1 #ou nb_succes=nb_succes+1
    return nb_succes
```

I.2 Loi géométrique

4. En utilisant la fonction **bernoulli**, écrire une fonction **geometrique** prenant en argument d'entrée un réel p de $]0;1[$ et renvoyant une réalisation d'une variable aléatoire suivant la loi $\mathscr{G}(p)$.
5. Proposer une autre fonction permettant de simuler une réalisation d'une variable aléatoire suivant la loi $\mathscr{G}(p)$ qui n'utilise pas la fonction **bernoulli**.

```python
import numpy.random as rd

def bernoulli(p):
    r=rd.random()
    if r<p:
        return 1
    else :
```

```
        return 0

def geometrique(p):
    rang=1
    while bernoulli(p)==0:
        rang+=1
    return rang

def geometrique_bis(p):
    rang=1
    while rd.random()<1-p:
        rang+=1
    return rang
```

II Simuler les lois usuelles avec les commandes existantes

Des commandes à connaître :

- `rd.random(N)` renvoie N réels aléatoires de $]0;1[$ dans un tableau 1 ligne $\times$ N colonnes
- `rd.randint(a,b)` renvoie un entier aléatoire de $[\![a;b[\![$
- `rd.randint(a,b,N)` renvoie N entiers aléatoires de $[\![a;b[\![$ dans un tableau 1 ligne $\times$ N colonnes
- `rd.binomial(n,p)` renvoie une réalisation d'une variable aléatoire suivant la loi $\mathscr{B}(n;p)$
- `rd.binomial(n,p,N)` renvoie N réalisations d'une variable aléatoire suivant la loi $\mathscr{B}(n;p)$ dans un tableau 1 ligne $\times$ N colonnes
- `rd.geometric(p)` renvoie une réalisation d'une variable aléatoire suivant la loi $\mathscr{G}(p)$
- `rd.geometric(p,N)` renvoie N réalisations d'une variable aléatoire suivant la loi $\mathscr{G}(p)$ dans un tableau 1 ligne $\times$ N colonnes
- `rd.poisson(lam)` renvoie une réalisation d'une variable aléatoire suivant la loi $\mathscr{P}(\lambda)$, avec $\lambda =$`lam`
- `rd.poisson(lam,N)` renvoie N réalisations d'une variable aléatoire suivant la loi $\mathscr{P}(\lambda)$, avec $\lambda =$`lam`, dans un tableau 1 ligne $\times$ N colonnes

✘ Attention !
Comme pour la commande `range(a,b)`, la borne de droite est exclue !

6. 6.a. Représenter l'histogramme des fréquences de 10000 réalisations indépendantes d'une variable aléatoire suivant la loi binomiale de paramètres $n = 10$ et $p = 0,3$.
   6.b. Sur le même graphique, faire apparaître les valeurs des probabilités d'une telle loi.

```
import numpy.random as rd
import matplotlib.pyplot as plt
import math

n=10
p=0.3
Labs=[-0.5+k for k in range(0,n+2)]
plt.hist(rd.binomial(n,p,1000),Labs,density=True,edgecolor='b')

Lproba=[math.comb(n,k)*p**k*(1-p)**(n-k) for k in range(0,n+1)]
plt.plot(range(0,n+1),Lproba,"r+")

plt.show()
```

Aide
Après avoir importé la bibliothèque `math`, la commande `math.comb(n,k)` renvoie la valeur de $\binom{n}{k}$.

Petite remarque
Le fait que la distribution empirique observée sur un échantillon de grande taille se rapproche de la distribution théorique est assez naturel. Nous le justifierons mathématiquement dans le chapitre 11 !

7. 7.a. Représenter l'histogramme des fréquences de 10000 réalisations indépendantes d'une variable aléatoire suivant la loi géométrique de paramètre $p = 0,5$.
   7.b. Sur le même graphique, faire apparaître les valeurs des probabilités d'une telle loi.

```
import numpy.random as rd
import matplotlib.pyplot as plt

p=0.5
L=[rd.geometric(p) for k in range(10000)]
m=max(L)
Labs=[-0.5+k for k in range(1,m)]
```

```
plt.hist(L,Labs,density=True,edgecolor='b')

Lproba=[p*(1-p)**(k-1) for k in range(1,m+1)]
plt.plot(range(1,m+1),Lproba,"r+")

plt.show()
```

8. 8.a. Représenter l'histogramme des fréquences de 10000 réalisations indépendantes d'une variable aléatoire suivant la loi de Poisson de paramètre $\lambda = 3$.
   8.b. Sur le même graphique, faire apparaître les valeurs des probabilités d'une telle loi.
   8.c. Comparer le graphique obtenu avec celui de la question 6....

```
import numpy.random as rd
import matplotlib.pyplot as plt
import numpy as np
import math

lam=3
p=0.3
L=[rd.poisson(lam) for k in range(10000)]
m=max(L)
Labs=[-0.5+k for k in range(0,m+2)]
plt.hist(L,Labs,density=True,edgecolor='b')

Lproba=[lam**k*np.exp(-lam)/math.factorial(k) for k in range(0,m+1)]
plt.plot(range(0,m+1),Lproba,"r+")

plt.show()
```

III Quelques autres lois...

III.1 Géométrique tronquée

On lance successivement et de façon indépendante une pièce donnant PILE avec la probabilité 0,3 jusqu'à obtenir le premier PILE, et on s'arrête dans tous les cas après 10 lancers. On note X la variable aléatoire égale au nombre de lancers effectués.

9. 9.a. Écrire une fonction telle que l'exécution de `simul_X()` renvoie une réalisation de la variable aléatoire X.
   9.b. Représenter l'histogramme des fréquences de 10000 réalisations indépendantes de X.

```
import numpy.random as rd
import matplotlib.pyplot as plt

def simul_X():
    n=1
    while n<10:
        if rd.random()<0.7:
            n=n+1
        else :
            return n
    return n

def simul_Xbis():
    n=1
    while n<10 and rd.random()<0.7:
        n=n+1
    return n

def simul_Xter():
    for n in range(1,11):
        if rd.random()<0.3:
            return n
    return 10

L=[simul_Xter() for k in range(10000)]
Labs=[-0.5+k for k in range(1,12)]
plt.hist(L,Labs,density=True,edgecolor='k')
plt.show()
```

9.c. Donner une valeur approchée de $\mathbb{P}([X = 10])$ et calculer sa valeur exacte.

III.2 Maximum de deux VA

Soit $n \in [\![2; +\infty[\![$. On considère une urne composée de n boules, numérotées de 1 à n, indiscernables au toucher. On tire simultanément deux boules dans cette urne. On note X_n la variable aléatoire égale au plus grand des deux nombres obtenus.

10. 10.a. Écrire une fonction telle que l'exécution de `simul_X(n)` renvoie une réalisation de la variable aléatoire X_n.

    10.b. Écrire un programme permettant d'obtenir une valeur approchée de $\mathbb{E}(X_5)$.

```
import numpy.random as rd
def simul_X(n):
    a=rd.randint(1,n+1)
    b=rd.randint(1,n+1)
    while b==a:
        b=rd.randint(1,n+1)
    return max(a,b)

n=5
L=[simul_X(n) for k in range(10000)]
E=sum(L)/len(L)
print(E)
```

Petite remarque

L'interprétation de la moyenne empirique comme valeur approchée de l'espérance est basée sur la loi faible des grands nombres, que nous verrons dans le chapitre 11.

Pour s'entraîner :

Déterminer la loi de X_n ainsi que son espérance et sa variance.
Si besoin : ECG1A - Chapitre 12 - Exercice 17.

III.3 Loi uniforme ?

Soit $n \in [\![3; +\infty[\![$. On dispose d'une urne contenant $(n-1)$ balles blanches et une balle noire. On effectue des tirages sans remise dans l'urne, jusqu'à l'obtention de la balle noire. On note X_n la variable aléatoire égale au rang d'apparition de la balle noire.

11. 11.a. Écrire une fonction `Python` telle que l'exécution de la commande `simul_X(n)` renvoie une réalisation de la variable aléatoire X_n.

    11.b. Représenter l'histogramme des fréquences de 10000 réalisations indépendantes de X_n dans les cas $n = 5$, puis $n = 10$.
    Que peut-on conjecturer ?

```
import numpy.random as rd
import matplotlib.pyplot as plt

def simul_X(n):
    nb=n #nb de balles restantes
    X=1
    while rd.randint(1,nb+1)!=1: #on code par 1 la balle noire
        nb=nb-1
        X=X+1
    return X

def simul_Xbis(n):
    nb=n
    X=1
    while rd.random()>1/nb:
        nb=nb-1
        X=X+1
    return X

n=10
L=[simul_X(n) for k in range(10000)]
```

```
Labs=[-0.5+k for k in range(1,n+2)]
plt.hist(L,Labs,edgecolor='k',density=True)
plt.show()
```

11.c. Démontrer cette conjecture.