Pour simuler des expériences aléatoires discrètes, on utilise le module `numpy.random` qu'il faut importer.

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

Python

- La commande `"rd.rand()"` renvoie un nombre réel aléatoire entre 0 et 1.
- La commande `"rd.rand(n,m)"` renvoie une matrice $n \times m$ dont chaque terme est un réel aléatoire entre 0 et 1.
- Soit p un réel fixé entre 0 et 1.

La commande `"rd.rand() < p"` simule un évènement de probabilité p.

- EXERCICE 1 (LOIS USUELLES DISCRÈTES) -

1. **Écrire une fonction `"Bernoulli(p)"` prenant comme paramètre un réel $p \in [0;1]$ et simulant une loi de Bernoulli de paramètre p.**

```
def Bernoulli(p) :
    if rd.rand() < p :
        return(1)  # On passe ici avec probabilité p (donc succès)
    else :
        return(0)  # On passe ici avec probabilité 1-p (donc échec)
```

2. **Écrire une fonction `"Binomiale(n,p)"` prenant comme paramètre un entier n et un réel $p \in [0;1]$ et simulant une loi binomiale de paramètres n et p.**

```
def Binomiale(n,p) :
    S = 0  # compte le nombre de succès
    for k in range(n) : # on fait n expériences
        if rd.rand()<p :  # succès dans une épreuve de Bernoulli
            S = S+1 # on augmente le nombre de succès
    return(S)
```

3. **Écrire une fonction `"Geometrique(p)"` prenant comme paramètre un réel $p \in [0;1]$ et simulant une loi géométrique de paramètre p.**

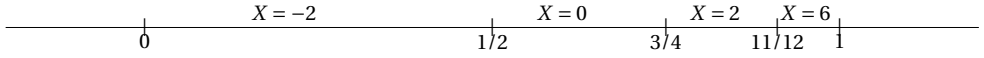
```
def Geometrique(p) :
    c = 1  # compte le nombre d'expérience (1 au départ)
    while rd.rand()>p : # tant qu'on a des échecs on continue
        c = c+1 # on augmente le nombre d'expériences
    return(c)
```

- EXERCICE 2 -

Le gain d'un joueur sur un jeu de grattage (prix du billet inclus) est modélisé par une variable aléatoire X dont la loi est donnée par : $X(\Omega) = \{-2,0,2,6\}$ **et** $P(X=-2)=\frac{1}{2}$, $P(X=0)=\frac{1}{4}$, $P(X=2)=\frac{1}{6}$, $P(X=6)=\frac{1}{12}$.

1. (a) **En utilisant judicieusement la commande `"rd.rand()"`, écrire une fonction `"SimuX()"` qui renvoie une simulation de la variable X.**

On peut modéliser les probas sur le segment [0, 1] comme suit :




```
def SimuX() :
    r = rd.rand()
    if r<1/2 :
        return(-2)
    elif r<3/4 :  # 3/4=1/2+1/4
        return(0)
    elif r<11/12 : # 11/12=1/2+1/4+1/6
        return(2)
    else :
        return(6)
```

(b) **Quelle valeur est affichée par ce programme?**

```
S = 0
for k in range(1000) :
    S += SimuX()
print( S/1000 )
```

Ce programme calcule la moyenne de X sur 1000 réalisations ce qui donne une valeur approchée de l'espérance de X.

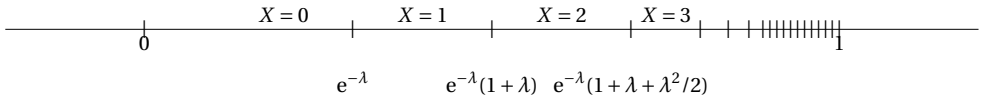
Or, $X(\Omega) = \{-2,0,2,6\}$ est fini donc $E(x)$ existe et on a : $E(X) = -2\frac{1}{2} + 0\frac{1}{4} + 2\frac{1}{6} + 6\frac{1}{12} = -\frac{1}{6}$.

Le programme renverra environ $-\frac{1}{6}$.

2. **S'inspirer de l'exemple précédent pour simuler une loi de Poisson de paramètre λ.**

Comme précédemment il faut découper l'intervalle [0, 1] avec les probabilités de la loi de poisson : $P(X=k) = \frac{e^{-\lambda}\lambda^k}{k!}$.

On tire ensuite un nombre au hasard entre 0 et 1 et on cherche dans quel segment il tombe.




```
import numpy as np, numpy.random as rd, math as m
def SimuPoisson(lbd) :
    r = rd.rand()
    k = 0
    S = np.exp(-lbd)*(lbd**k)/(m.factorial(k))
    while S < r :
        k = k+1
        S = S +  np.exp(-lbd)*(lbd**k)/(m.factorial(k))
    return k
```

Python

Soit X une variable aléatoire suivant la loi binomiale de paramètres n et p.

- La commande `"rd.binomial(n, p)"` renvoie une réalisation de X.
- La commande `"rd.binomial(n, p, k)"` renvoie un vecteur contenant `"k"` réalisations de X.

Soit Y une variable aléatoire suivant la loi géométrique de paramètre p.

- La commande `"rd.geometric(p)"` renvoie une réalisation de Y.
- La commande `"rd.geometric(p, k)"` renvoie un vecteur contenant `"k"` réalisations de Y.

Soit Z une variable aléatoire suivant la loi de Poisson de paramètre λ (noté `lbd`).

- La commande `"rd.poisson(lbd)"` renvoie une réalisation de Z.

- La commande `"rd.poisson(lbd, k)"` renvoie un vecteur contenant `"k"` réalisations de Z.

- EXERCICE 3 -

1. **La commande `"np.mean(L)"` renvoie la moyenne empirique de la liste `L`. Donner une approximation de la valeur affichée par le programme suivant.**

```
np.mean(rd.geometric(0.25 ,10000))
```

On simule un vecteur contenant 10000 réalisations d'une variable aléatoire X suivant une loi géométrique de paramètre 0.25 puis on calcule sa moyenne (avec `"np.mean"`). Ainsi, on obtient une approximation de l'espérance $E(X) = \dfrac{1}{0.25} = 4$.

2. **La commande `"np.count_nonzero(condition)"` compte le nombre de fois que la "condition" est vraie dans une liste. Quelle est la variable aléatoire simulée par la variable `"X"` dans le programme suivant ?**

```
R = rd.rand(1,20)
X = np.count_nonzero(R<0.25)
```

La variable `"R"` est un vecteur contenant 20 nombres aléatoires entre 0 et 1.
La variable `"X"` compte le nombre d'éléments de `"R"` inférieurs à 0.25 donc le nombre de fois qu'une expérience de Bernoulli de paramètre 0.25 (`"rand()<0.25"`) est un succès.
Ainsi, la variable `"X"` simule une loi binomiale $\mathscr{B}(20, 0.25)$.

- EXERCICE 4 -

On lance n fois une pièce équilibrée ($n \geqslant 2$), 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 prend pour valeur le rang du premier pile dans le cas contraire.

1. **Compléter la fonction suivante permettant de simuler la variable aléatoire Z, le nombre n de lancers étant donné en paramètre.**

```
def SimuZ(n) :
    for i in range(1, n+1) :
        if rd.rand()<1/2 : # on simule l'obtention d'un pile (proba=1/2)
            return(i)  # on a obtenu un pile au tirage i
    return(0)    # aucun pile n'a été obtenu sur les n tirages
```

2. **Écrire un programme renvoyant une approximation de l'espérance de Z.**
Au choix :

```
S = 0
for k in range(1000) :
    S += SimuZ()
print( S/1000 )
```

```
S = [SimuZ() for k in range(1000)]
print( np.mean(S) )
```

- EXERCICE 5 -

Deux individus A et B s'affrontent dans un jeu de Pile ou Face dont les règles sont les suivantes :

- **le joueur A dispose d'une pièce amenant Pile avec la probabilité $\dfrac{2}{3}$ et lance cette pièce jusqu'à l'obtention du deuxième Pile ; on note X la variable aléatoire prenant la valeur du nombre de Face alors obtenus ;**
- **le joueur B dispose d'une autre pièce amenant Pile avec la probabilité $p \in]0,1[$ et lance cette pièce jusqu'à l'obtention d'un Pile ; on note Y la variable aléatoire prenant la valeur du nombre de Face alors obtenus ;**
- **Le joueur A gagne si son nombre de Face obtenus est inférieur ou égal à celui de B ; sinon c'est le joueur B qui gagne.**

On dit que le jeu est équilibré lorsque les joueurs A et B ont la même probabilité de gagner.

1. **Écrire une fonction Python `"simule_X()"` qui simule la variable aléatoire X.**

```
def simule_X() :
    Pile = 0
    Face = 0
    while Pile < 2 : # on continue le jeu tant qu'on n'a pas 2 piles
        if rd.rand() < 2/3 : # on obtient pile
            Pile = Pile + 1
        else :
            Face = Face + 1
    return(Face)
```

2. **Écrire une fonction Python `"simule_Y(p)"` qui prend en argument un réel p de $]0;1[$ et simule la variable aléatoire Y.**

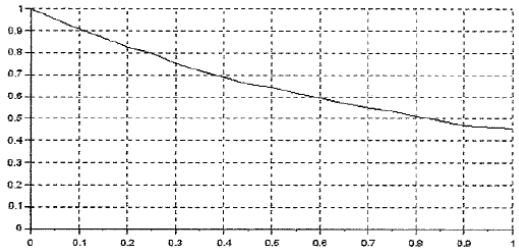
```
def simule_Y(p) :
    Face = 0
    while rd.rand() < 1-p : # on simule un face (proba 1-p)
                            # jusqu'à obtention d'un pile
        Face = Face + 1
    return(Face)
```

3. **Expliquer ce que renvoie la fonction suivante :**

```
def mystere(p) :
    r = 0
    N = 10**4
    for k in range(1,N+1) :
        x = simule_X()
        y = simule_Y(p)
        if x <= y :
            r = r + 1/N
    return( r )
```

On fait 10000 simulations de X et de Y et on regarde si $X \leqslant Y$ donc on regarde si A gagne.
Ainsi, on compte le nombre de fois que A gagne sur 10000 expériences. En fait, on calcule plutôt la fréquence de victoire de A (`"r = r + 1/N"` calcule une fréquence (puisqu'on divise par le nombre d'expériences).

4. **On trace, en fonction de p, une estimation de la probabilité que A gagne et on obtient le graphe suivant :**



À la vue de ce graphe, conjecturer une valeur de p pour lequel le jeu serait équilibré.
On cherche la valeur de p (en abscisses) pour laquelle $P(\text{A gagne})$ (en ordonnées) vaut 1/2. On obtient : $p \approx 0.8$.

Python 1

Soit X une variable aléatoire suivant la loi uniforme discrète sur $[\![m;n]\!]$: $X \hookrightarrow \mathscr{U}([\![m;n]\!])$.
La commande `"rd.randint(m, n+1)"` renvoie une simulation de X.

- EXERCICE 6 -

On dispose de n urnes, numérotées de 1 à n. Pour chaque $k \in [\![1;n]\!]$, l'urne k contient k boules numérotées de 1 à k. On choisit une urne au hasard puis on tire une boule dans cette urne. On note X_n la variable aléatoire qui prend la valeur du numéro de la boule piochée.
Écrire une fonction `"simuX(n)"` qui simule X_n pour une valeur de n donnée en paramètre.

```
def simuX(n) :
    u = rd.randint(1, n+1) # on choisir l'urne
    X = rd.randint(1, u+1) # on pioche une boule dans l'urne
                           #(il y a u boules)
    return(X)
```

- EXERCICE 7 -

Soit N un entier supérieur ou égal à 3. On considère une urne qui contient une seule boule noire numérotée 1 et $(N-1)$ boules blanches numérotées de 2 à N et .
On effectue des tirages sans remise jusqu'à l'obtention de la boule noire et on note X la variable aléatoire qui donne le nombre de tirages nécessaires.

1. **Compléter la fonction suivante pour qu'elle simule X pour une valeur de N donnée en paramètre.**

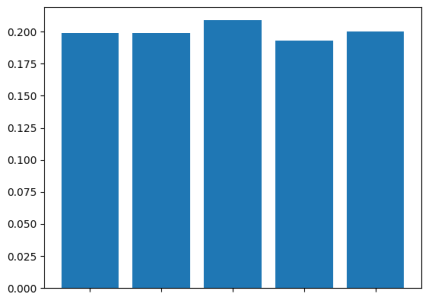
```
def simuX(N) :
    nbr_boules = N
    c = 0
    for i in range(1,N+1) :
        tirage = rd.randint(1,nbr_boules+1)  # on pioche une boule
        c = c+1   # un tirage de plus
        if tirage == 1 :  # on a obtenu la noire
            return(c)  # on renvoie le numero du tirage
        else :
            nbr_boules = nbr_boules-1  # il n'y a pas remise
                                       # donc 1 boule de moins
```

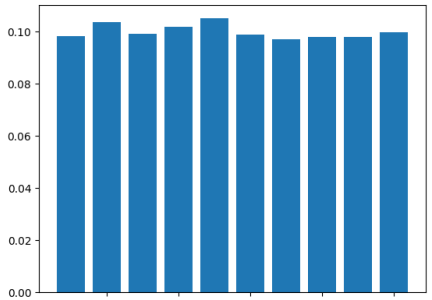
2. **Compléter le programme suivant pour qu'il trace le diagramme en bâton de X, à partir de ses fréquences.**

```
import matplotlib . pyplot as plt

N = 5
freqX = np.zeros(N)  # on construit un vecteur qui contiendra les fréquences
for k in range (10000) :
    i = simuX(N)      # L'évènement (X=i) est réalisé
    freqX[i-1] += 1  # on augmente le nombre d'obtention de (X=i)
freqX =  freqX /10000
# diagramme en batons
plt.bar( range(1, N+1),  freqX )
plt.show ()
```

3. **On obtient les diagrammes en bâtons ci-dessous pour $N = 5$ et $N = 10$. Que peut-on conjecturer sur la loi de X ?**





On remarque une situation d'équiprobabilité ($P(X = k)$ est à peu près constant) ce qui permet de conjecturer que X suit la loi uniforme sur $[\![1;N]\!]$.

- EXERCICE 8 -

Une urne contient initialement 1 boule rouge et 1 boule blanche. On effectue une succession d'épreuves selon le protocole suivant.

- **On pioche une boule au hasard dans l'urne. On replace la boule tirée dans l'urne puis on rajoute une boule de la même couleur que celle qui vient d'être piochée.**

Pour tout $n \in \mathbb{N}^*$, on note X_n le nombre de boules rouges qui ont été ajoutées dans l'urne (par rapport à la composition initiale) à l'issue des n premières épreuves.

1. (a) **Écrire une fonction `"tirage(x,y)"` qui simule le tirage d'une boule dans une urne contenant `"x"` boules rouges et `"y"` boules blanches et qui retourne la valeur 0 si la boule est rouge et 1 si elle est blanche.**

```
def tirage(x,y) :
    r = rd.rand()
    if r < x/(x+y) :
        return(0)
    else :
        return(1)
```

(b) **Compléter la fonction suivante, qui simule n tirages successifs dans une urne contenant initialement 1 boule rouge et 1 boule blanche (selon le protocole décrit ci-dessus) et qui retourne la valeur de X_n :**

```
def experience(n) :
    x = 1
    y = 1
    for k in range(1,n+1) :
        r = tirage(x,y)
        if r == 0 : # on obtient une boule rouge
            x = x+1
        else :      # on obtient une boule blanche
            y = y+1
    return(x-1)
```

(c) **Donner l'ensemble $X_n(\Omega)$.**
$X_n(\Omega) = [\![0; n]\!]$.

(d) **On souhaite estimer la loi de X_n. Pour cela, on va construire un vecteur nommé loi contenant les approximations de $P(X_n = 0)$, $P(X_n = 1), \dots, P(X_n = n)$. Pour obtenir ces approximations, on va simuler m fois (m grand) l'expérience grâce à la fonction `"experience"` précédente.**
Compléter la fonction suivante pour que la variable loi contienne en sortie les approximations de $P(X_n = 0)$, $P(X_n = 1), \dots, P(X_n = n)$.

```
def  simulation(n,m) :
    loi = np.zeros(n+1)
    for k in range(1,m+1) :
        Xn = experience(n)  # l'expérience a retourné la valeur Xn
        loi[Xn] = loi[Xn] + 1 # on augmente nbr d'obtention de Xn
    loi = loi / m # On calcule la fréquence (approx. des probas)
    return(loi)
```

2. (a) **On considère les instructions suivantes et leurs sorties.**
Pour $n = 2$ et $m = 10000$:

```
>>> simulation(2,10000)
array([0.3331, 0.3317, 0.3352])
```

Pour $n = 4$ et $m = 10000$:

```
>>> simulation(4,10000)
array([0.2051, 0.1897, 0.2053, 0.2029, 0.197 ])
```


Que peut-on conjecturer sur la loi de X_n ?
Il semblerait que X_n suive une loi uniforme (probas sensiblement égales). Plus précisément, on peut conjecturer que X_n suit une loi uniforme sur $[\![0; n]\!]$ puisque chaque probabilité vaut environ $\frac{1}{n+1}$.

(b) **Déterminer la loi de X_1 et vérifier votre conjecture lorsque $n = 1$.**

- $X_1(\Omega) = [\![0; 1]\!]$.
- En notant B_i (resp. R_i) l'évènement "tirer une boule blanche (resp. rouge) au tirage numéro i, on a :

$$P(X_1 = 0) = P(B_1) = \frac{1}{2} \quad \text{et} \quad P(X_1 = 1) = P(R_1) = \frac{1}{2}$$

La conjecture est donc vérifiée : X_1 suit une loi uniforme sur $[\![0; 1]\!]$

(c) **En utilisant la formule des probabilités totales, déterminer la loi de X_2 et vérifier votre conjecture lorsque $n = 2$.**

- $X_2(\Omega) = [\![0; 2]\!]$.
- D'après la FPT avec le SCE $((X_1 = 0), (X_1 = 1))$, on a :

$$\begin{aligned} P(X_2 = 0) &= P((X_1 = 0) \cap (X_2 = 0)) + P((X_1 = 1) \cap (X_2 = 0)) \\ &= P(B_1 \cap B_2) + 0 \\ &= P(B_1) P_{B_1}(B_2) = \frac{1}{2}\frac{2}{3} = \frac{1}{3} \end{aligned}$$

En effet, $P_{B_1}(B_2) = \frac{2}{3}$ car cela signifie qu'on a tiré une blanche dans une urne contenant 1 rouge et 2 blanches.
Et $P((X_1 = 1) \cap (X_2 = 0)) = 0$ car cet évènement est impossible.

- De même, on obtient :

$$P(X_2 = 1) = \frac{1}{3} \quad \text{et} \quad P(X_2 = 2) = \frac{1}{3}$$

La conjecture est donc vérifiée.