TP Python 9 : Générer de l'aléa

Partie I : La fonction `rand`

- Le but de ce TP est de découvrir un **module** Python très puissant: le module `random` de la bibliothèque `numpy`. (Un module est une "sous-librairie" d'une librairie "mère", plus générale.)
- Ce module contient des fonctions permettant de **générer des nombres aléatoires**.

Pour traiter un problème de probabilité/générer de l'aléatoire avec Python, on importe donc systématiquement le module `random`.

Entrée[1]:

```
import numpy as np
import numpy.random as rd
from math import floor # on importe la fonction floor depuis la b
```

Découvrons quelques fonctions de bases, disponibles dans la bibliothèque `random`, et **essentielles dans le programme d'ECG**.

1. La fonction `randint` permet de générer un **entier** N aléatoire compris entre deux entiers a et b : $a \leq N < b$.

   Pour appeler cette fonction on utilise la commande `rd.randint(a,b)`.
2. La fonction `random` du module `random` (oui, il y a répétition du nom) génère un flottant compris entre 0 **inclus** et 1 **exlus**.

   Pour appeler cette fonction, on utilise la commande `rd.random()`.

À vous de faire :

1. Construire une liste N contenant 15 nombres entiers tirés au hasard entre -2 et 8, puis l'afficher.
2. Construire une liste X contenant 9 nombres réels tirés au hasard, appartenant à $[0, 1[$, puis l'afficher.

Entrée[44]:

```

```

Sortie[44]: 3

→ Cliquez-ici pour la réponse ←

À vous de faire :

1. Que fait la fonction Python suivante?
2. Que va renvoyer la commande `somysterious(3)` ?

Entrée[ ]:

```python
def somysterious():
    nb = rd.random()
    if nb < 1/3:
        return "P"
    else:
        return "F"

somysterious()
```

→ Cliquez-ici pour la réponse ←

Partie II : Exercices

On va voir dans les trois premiers exercices différentes manières de simuler un dé.

Exercice 1
À l'aide **d'instructions conditionnelles**, écrire une fonction qui simule le lancer d'un dé à quatre faces, tel que la face 1 apparait avec probabilité $\frac{1}{4}$, la face 2 avec probabilité $\frac{1}{2}$, et les faces 3 et 4 avec probabilité $\frac{1}{8}$.

Entrée[ ]:

```python

```

Exercice 2
Sans utiliser `randint`, et à l'aide de la fonction `floor` de la bibliothèque `math`, écrire une fonction qui simule le lancer d'un dé équilibré à 6 faces.

Indication : soit $x \in [0, 1]$ un réel. Dans quel intervalle appartient le réel $a + bx$ avec a, b deux entiers naturels?

Entrée[ ]:

```python

```

Exercice 3

1. À l'aide de la fonction `randint`, écrire une fonction `RollTwoDice` qui simule le lancer de deux dés équilibrés à six faces, et qui renvoie 0 si la somme des deux faces obtenues est différente de 7, et 1 sinon.
2. Construire une autre fonction `MutlipleRolls` qui utilisera `RollDice`, prendra en argument un entier N, réalisera N lancers de deux dés, et renverra la fréquence d'apparition du résultat 7.
3. Que devient cette fréquence sur 100 séries, 1000 séries, ... ?
4. Quelle estimation de la probabilité d'apparition du 7 peut-on proposer ?

Entrée[ ]:
```
1
```

Exercice 4

1. Construire une fonction `piece` qui ne prends rien en argument, et qui code une pièce équilibrée. On renverra 0 pour ``pile'' et $1$ pour ``face''.
2. Constuire une fonction `success` qui prend en argument une liste L de nombres appartenant à $\{0, 1\}$, compte le nombre de 0 dans la liste, et **affiche** la phrase "Gagné!" (resp. ``Perdu!'') lorsque ce nombre est supérieur à 50.
   *Notez que cette fonction affiche des phrases... Elle ne retourne rien. Ne pas oubliez le `return` en fin de fonction qui est **obligatoire**!!!*
3. Construire une liste `lancers` contenant 100 résultats de lancers de pièce, et tester votre fonction `success`.

Entrée[ ]:
```
1
```

Exercice 5

1. Écrire une fonction `ReachEight` tirant des nombres aléatoires entre 0 et 1, jusqu'à ce que la somme des nombres tirés dépasse 8, et qui retourne le nombre X de tirages qui ont été faits.
2. Écrire un programme pour que l'expérience soit faite 10 fois, et affiche à chaque fois le nombre de tirages effectués X pour que la somme dépasse 8.
3. Conjecturer combien de tirages sont nécessaires en moyenne pour que la somme dépasse 8.
4. Écrire une fonction prenant en argument N le nombre de fois où l'expérience est effectuée, et qui renvoie la moyenne du nombre de tirages X. Tester cette fonction pour $N = 10$, $N = 100$, et $N = 1000$.

Entrée[ ]:
```
1
```

Entrée[ ]:
```
1
2
```

<u>Partie III :</u> Diagrammes en bâtons

Réflexe :
Importer le module nécessaire à la représentation graphique en Python.

Entrée[ ]:

```

```

La commande `bar(x, y)` du module `matplotlib.pyplot` permet de représenter un diagramme en bâtons.
Plus précisément, en notant les arguments $x = [x_1, \dots, x_n]$ et $y = [y_1, \dots, y_n]$ (listes de mêmes longueurs) elle permet de positionner un bâton de hauteur y_i en face de la valeur x_i.

Exemple :
Exécuter le programme suivant.

Entrée[ ]:

```python
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()
```

Exercice 6
Représenter le diagramme en bâtons dont les hauteurs des bâtons sont les valeurs **théoriques** de la loi uniforme U(⟦1, N⟧) pour $N = 5$.

Entrée[ ]:

```

```

- Les diagrammes en bâtons sont adaptés au cadre des variables aléatoires dont la loi est finie.
- 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!

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.

Exercice 7

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 $k \in [\![1, n]\!]$, X_k la variable aléatoire qui prend la valeur de la boule obtenue au k-ième tirage.

1. Que fait la fonction `shuffle` du module `random` ? (*Cette fonction est hors programme*).
2. Compléter la fonction python `simul_X` ci-dessous, d'arguments n et k qui renvoie la simulation de X_k.

Entrée[ ]:

```
def simul_X(k, n):
    if k > n:
        print("......")
        .......
    else :
        urne = list(range(1, n + 1))
        rd.shuffle(......)

    return ......


simul_X(3,7)
```

2. On ajoute les commandes suivantes. Expliquer ce qu’elles font. On insistera que la commande de la ligne 8.

Entrée[ ]:

```
1  n = 6
2  k = 4
3
4  echantillon = [simul_X(k,n) for repeat in range(1000)]
5
6  freq = [0] * n
7  for result in echantillon:
8      freq[result - 1] += 1
9
10 freq = [f/1000 for f in freq]
11
12 plt.clf()
13 plt.bar([j for j in range(1, n + 1)], freq, color='pink', edgecol
14 plt.title(f"Fréquence des résultats pour {len(echantillon)} tira
15 plt.xlabel("Valeur de la boule tirée")
16 plt.ylabel("Fréquence")
17 plt.show()
18
```