TP 20 Python – Aléatoire

Python dispose de toute une bibliothèque pour modéliser des expériences aléatoires : `random`.
On commencera donc par importer ce module sous *l'alias* `rd` : `import random as rd`

On a déjà rencontré à plusieurs reprises les deux fonctions essentielles de ce module :

- `rd.random`, qui ne prend pas d'argument et qui renvoie un flottant choisi uniformément entre 0 et 1.
- `rd.randint`, qui prend en argument deux entiers a, b, et qui renvoie un entier choisi uniformément entre a et b (inclus…)

Nous allons nous baser uniquement sur ces deux fonctions pour modéliser n'importe quelle expérience aléatoire sur un univers fini. Nous aurons principalement deux objectifs :

1. Modéliser une loi (une expérience, une variable aléatoire), c'est-à-dire écrire une fonction qui renvoie un **résultat aléatoire** selon cette loi.
   Par exemple, `rd.randint(0,4)` modélise la loi uniforme sur l'ensemble $[\![0, 4]\!]$.
2. Approcher l'espérance (et la variance) d'une variable aléatoire en simulant l'expérience aléatoire un grand nombre de fois, et en calculant les moyennes appropriées.

1. Modéliser une loi

Pour modéliser par exemple un pile ou face, on peut :

```
def pièce():
    k = rd.randint(0,1)
    if k == 0:
        return "P"
    else:
        return "F"
```

```
def pièce_bis():
    p = rd.random()
    if p < 1/2:
        return "P"
    else:
        return "F"
```

Exercice 1 (Loi uniforme sur $[\![a, b]\!]$).

1. Écrire un programme (pas une fonction) qui renvoie entier choisi uniformément entre 1 et 4, **en utilisant seulement la fonction** `rd.random()`.
2. Adapter cette manière de faire pour écrire une fonction `uniforme(a,b)` qui se comporte de la même manière que `rd.randint(a,b`, mais en utilisant uniquement la fonction `rd.random()`
3. Écrire une fonction `trois_dés()` qui simule le lancer de 3 dés, et qui renvoie la somme.
4. On lance un dé équilibré à 20 faces numérotées de 1 à 20.
   (a) Quelle est la probabilité que le nombre obtenu soit un multiple de 3 ?
   (b) Écrire un programme qui modélise 100 lancers de ce dé, et qui affiche la proportion de lancers pour lesquels le résultat obtenu est un multiple de 3. Recommencer en faisant 1000, puis 100000 tirages. Vos résultats sont-ils en accord avec la question précédente ?

Exercice 2 (Tirage dans une urne).

1. On tire dans une urne contenant 4 boules rouges, 3 boules vertes et 7 boules bleues.
   (a) Que fait l'instruction suivante ? `U = ["R"]*4 + ["V"]*3 + ["B"]*7`
   (b) En déduire une manière de simuler un tirage dans cette urne avec la fonction `randint`
   (c) Écrire une fonction `tirages_avec_remise(n)` qui simule n tirages *avec* remise et qui renvoie les résultats successifs dans une liste.
2. Écrire sur le même modèle une fonction `tirageRVB(r, v, b)` qui simule un tirage dans une urne contenant r boules rouges, v boules vertes et b boules bleues.
3. La fonction `choice` du module `random` renvoie un élément choisi uniformément parmi les éléments d'une liste passée en argument.
   Écrire une fonction `tirages_sans_remise(n)` qui qui simule n tirages sans remise dans l'urne de la question 1.
   On rappelle que l'instruction `L.remove(a)`, qui permet de retirer (une fois) l'élément a de la liste L.

Exercice 3 (Loi de Bernoulli)**.**

1. Modéliser le lancer d'une pièce déséquilibrée, pour laquelle on a seulement une chance sur cinq d'obtenir "Pile".
2. Modéliser de même le lancer d'une pièce pour laquelle la probabilité d'obtenir "Face" vaut $\frac{\sqrt{2}}{2}$.
3. Écrire une fonction `Bernoulli(p)` qui modélise une variable aléatoire suivant la loi de Bernoulli de paramètre p, avec $p \in]0,1[$.

Exercice 4 (Loi binomiale)**.** En utilisant le fait qu'une loi binomiale représente une répétition d'expériences de Bernoulli indépendantes, écrire une fonction `Binomiale(n,p)` qui prend en argument un entier $n \geq 1$ et un flottant $p \in]0,1[$, et qui modélise une variable aléatoire suivant la loi $Bin(n,p)$.

Exercice 5 (Autres lois)**.**

1. Écrire une fonction `dé_truqué()` qui ne prend aucun argument et qui renvoie un entier compris entre 1 et 6 , sachant qu'on veut avoir les probabilités d'apparition suivantes :

   - $\mathbb{P}(X=1) = \frac{1}{12}$
   - $\mathbb{P}(X=2) = \frac{1}{3}$
   - $\mathbb{P}(X=3) = \frac{1}{6}$
   - $\mathbb{P}(X=4) = \frac{1}{24}$
   - $\mathbb{P}(X=5) = \frac{1}{8}$

   On utilisera l'instruction `k = rd.randint(1,24)`.
2. Écrire une fonction `dé_truqué_bis()`, qui fait la même chose que la précédent, mais en utilisant uniquement la fonction `random`.
3. De la même manière, simuler en Python la variable aléatoire X donnée par :

x	-2	-1	0	1	2
$\mathbb{P}(X=x)$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	$\frac{1}{3}$	$\frac{1}{15}$

À retenir : **on peut simuler n'importe quelle loi de probabilités finie à l'aide de sa fonction de répartition !**
Si X prend les valeurs $x_1, \ldots, x_n$ avec probabilités respectives $p_1, \ldots, p_n$:

```
def simule():
    p = rd.random()
    if p <= p1:
        return x1
    elif p <= p1 + p2:
        return x2
    elif p <= p1+p2+p3:
        return x3
    # etcaetera !
    else:
        return xn
```

2. Approcher l'espérance (et la variance)

L'espérance en probabilité représente la même chose que la moyenne en statistique. L'idée pour avoir une valeur approchée de l'espérance d'une variable aléatoire et de faire **la moyenne des résultats obtenus lorsqu'on simule un grand nombre de fois cette variable aléatoire.**

Exercice 6 (Avec des dés)**.** Dans tout l'exercice, on comparera la valeur d'espérance approchée par Python à la valeur exacte qu'on calculera mathématiquement.

1. Écrire une fonction `esp_trois_dés(n)` qui renvoie une valeur approchée de l'espérance de la variable aléatoire égale à la somme de trois dés. Le paramètre n représente le nombre de fois que l'on simule la variable aléatoire. On se servira de la fonction `trois_dés()` de l'exercice 1.

2. Écrire une fonction `esp_somme_dés(n,k)` qui donne une valeur approchée de l'ispérance de la variable aléatoire égale à la somme obtenue en lançant k dés.
3. Écrire une fonction `exp_dé_truqué(n)` qui estime expérimentalement la valeur de l'espérance associée au lancer du dé truqué de l'exercice 5.

Exercice 7 (Loi uniforme)**.**
1. Écrire une fonction `esp_unif(n, a, b)` qui renvoie une valeur approchée de l'espérance d'une variable aléatoire suivant la loi uniforme sur $[\![a, b]\!]$.
2. Vérifier sur quelques valeurs de a, b que votre fonction donne des valeurs raisonnables. On essaiera par exemple pour $n = 2$, $n = 10$, $n = 100$, $n = 1000$. Que remarque-t-on ?
3. On rappelle que si $X \hookrightarrow \mathcal{U}([\![a, b]\!])$, alors la valeur exacte de son espérance est $\mathbb{E}(X) = \dfrac{a+b}{2}$.
   Écrire une fonction `auxiliaire(a,b)` qui simule la variable aléatoire $(X - \mathbb{E}(X))^2$ où X suit la loi uniforme sur $[\![a, b]\!]$.
4. En déduire une fonction `variance_unif(n,a,b)` qui renvoie une valeur approchée de la variance d'une variable aléatoire suivant la loi uniforme sur $[\![a, b]\!]$.

Exercice 8. Écrire une fonction `esp_binomiale(n,p,nb_exp)` qui renvoie une valeur approchée de l'espérance d'une variable aléatoire suivant la loi binomiale de paramètres $n \in \mathbb{N}^*$ et $p \in]0, 1[$, obtenue avec `nb_exp` simulations.

3. Exercices de modélisation

Exercice 9. On considère une pièce non truquée. On la lance jusqu'à ce que l'on obtienne Face. On note le nombre de lancers effectués. Écrire une fonction Python qui ne prend aucun argument et qui simule l'expérience décrite.

Exercice 10. On considère une urne contenant N_1 boules rouges, N_2 boules bleues indiscernables au toucher. On tire une à une et sans remise toutes les boules de l'urne. On note r le rang d'apparition de la première boule rouge.
Par exemple,pour $N_1 = 4$ et $N_2 = 7$, si l'on obtient $BBBBBRBBRRR$, on a $r = 6$.
Écrire une fonction `rang_apparition(N1, N2)` qui simule l'expérience décrite précédemment.

Exercice 11 (Marche aléatoire dans Manhattan)**.** Sam se promène dans Manhattan de manière aléatoire : à chaque intersection, elle décide au hasard entre les 4 directions possibles (Nord, Sud, Est, Ouest).
On modélise Manhattan par une grille infinie, où les intersections sont les points $(a, b) \in \mathbb{Z}^2$, et l'origine $(0, 0)$ est la position de départ de Sam.

1. Écrire une fonction `une_étape(position)` qui prend en argument un couple représentant la position de Sam, et qui renvoie un couple correspondant à la position de Sam à l'étape suivante.
2. Écrire une fonction `marche(n)` qui simule le trajet de Sam sur n étapes. On renverra la liste des différentes positions de Sam.
3. En déduire une fonction `trace_marche(n)` qui trace et affiche le trajet de Sam sur n étapes. On utilisera le module `matplotlib.pyplot`.
4. On note D la distance (à vol d'oiseau) entre le point d'arrivée et le point de départ de Sam. On suppose que deux intersections à Manhattan sont toujours séparées de $100m$. Écrire une fonction `distance(n)` qui renvoie cette valeur après n étapes de Sam.
5. Écrire une fonction `esp_dist(n, nb_simul)` qui détermine une valeur approchée de $\mathbb{E}(D)$. `n` est le nombre d'étapes effectuées par Sam, et `nb_simul` désigne le nombre de simulations effectuées pour approcher l'espérance.
6. Écrire de même une fonction `variance_dist(n, nb_simul)` qui donne une valeur approchée de $\mathbb{V}(D)$.
7. (**) Simuler la même marche, **mais** : à chaque intersection, Sam choisit uniformément entre aller tout droit, tourner à gauche ou tourner à droite (elle ne fait jamais demi-tour).