Correction du TP 9

Pour tout le TP, on importe la bibliothèque suivante.

```
import numpy.random as nr
```

Exercice 1 — Loi uniforme

La fonction suivante prend en argument un entier n et simule une variable aléatoire suivant la loi uniforme sur $[\![1;n]\!]$.

```
def uniforme(n):
    """
    Entrée : un entier n.
    Sortie : valeur d'une variable aléatoire
             suivant la loi uniforme sur [[1 ; n]].
    """
    return nr.randint(1, n + 1)
    # On renvoie un entier aléatoire entre 1 et n,
    # choisi selon la loi uniforme.
```

Exercice 2 — Alea jacta est

1. La variable aléatoire donnant le numéro de la face obtenue lors du lancer d'un dé équilibré suit la loi uniforme sur $[\![1;6]\!]$.

```
def de():
    """
    Entrée : aucune.
    Sortie : résultat d'un lancer de dé équilibré à six faces.
    """
    return uniforme(6)
```

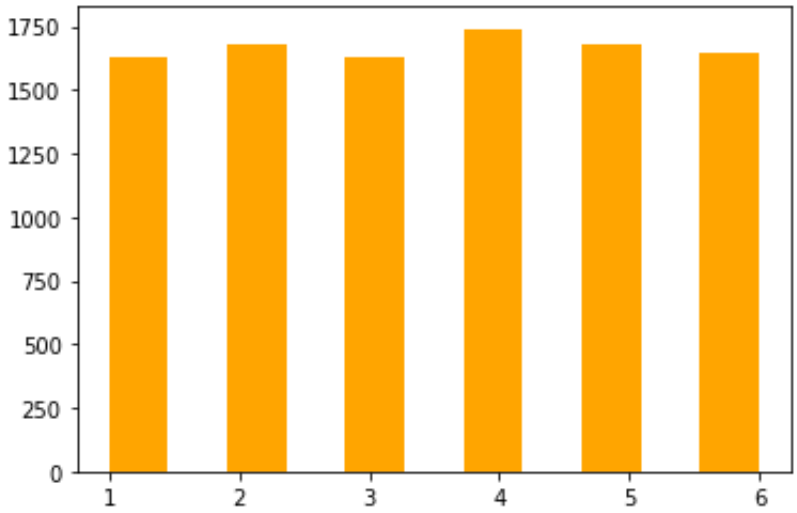
2. (a) La fonction suivante prend en argument un entier N et renvoie la liste des résultats obtenus sur N lancers de dés consécutifs (indépendants).

```
def liste_lancers(N):
    """
    Entrée : un entier N.
    Sortie : la liste des résultats de N lancers d'un dé équilibré.
    """
    L = [] # L sera la liste des résultats obtenus.
    for lancer in range(N):
    # La variable lancer va de 0 à N - 1,
    # donc on fait N tours de boucles, c'est-à-dire N lancers ici.
        L.append(de())
        # On ajoute à la liste le résultat d'un lancer de dé.
    return L
```

(b) Traçons l'histogramme des résultats obtenus sur 10 000 lancers de dé.

```
import matplotlib.pyplot as plt

plt.hist(liste_lancers(10000), 11)
```



Les six faces ont été obtenues à peu près le même nombre de fois sur les 10 000 lancers, ce qui n'est pas étonnant, car le dé équilibré.

3. La fonction suivante prend en argument un entier N et renvoie la moyenne des résultats obtenus sur N lancers d'un dé équilibré.

```
def moyenne_lancers(N):
    """
    Entrée : un entier N.
    Sortie : la moyenne des résultats de N lancers d'un dé équilibré.
    """
    L = liste_lancers(N)
    # On crée la liste des résultats de N lancers de dé,
    # puis on calcule la moyenne des éléments de la liste.
    somme = 0
    for x in L: # x parcourt tous les éléments de la liste L.
        somme += x
    moyenne = somme / N
    return moyenne
```

Exercice 3 — Tirages dans une urne

Une urne contient trois boules rouges, cinq boules orange et 2 boules roses.
Dans cette correction, nous allons représenter chaque configuration de l'urne par la donnée :

- d'une variable `total_urne` qui vaut le nombre de boules total de l'urne ;
- d'un dictionnaire `Urne` dont les clés sont les couleurs des boules et les valeurs associées sont les nombres de boules de chaque couleur.

Ainsi, initialement, l'urne est représentée de la manière suivante :

```
total_urne = 10
# L'urne contient 10 boules en tout.

Urne = {"rouge" : 3, "orange" : 5, "rose" : 2}
# L'urne contient 3 boules rouges, 5 boules orange et 2 boules roses.
```

Cette représentation donne accès facilement à la probabilité de tirer chaque couleur.

1. Pour simuler le tirage d'une boule dans l'urne, on procède de la même manière que dans la section 2.3 du cours.

```
def tirage(Urne, total_urne):
    """
    Entrées : - nombre total_urne de boules dans l'urne ;
              - dictionnaire Urne du nombre de boules
                de chaque couleur dans l'urne.
    Sortie : la couleur d'une boule tirée dans l'urne.
    """
    u = nr.rand() # Nombre aléatoire de [0 ; 1].
    somme = 0
    # La variable sommera au fur et à mesure
    # les probabilités de tirer chaque couleur de boule.
    for couleur in Urne:
    # La variable couleur parcourt les clés du dictionnaire Urne,
    # c'est-à-dire les différentes couleurs des boules de l'urne.
        proba_couleur = Urne[couleur] / total_urne
        # Probabilité de tirer une boule de la couleur couleur.
        somme += proba_couleur
        if u < somme:
            return couleur
```

2. (a) Pour effectuer cinq tirages avec remise dans l'urne, on applique simplement cinq fois la fonction précédente, sans changer l'urne à chaque tirage.

```
def cinq_tirages_avec_remise(Urne, total_urne):
    """
    Entrées : - nombre total_urne de boules dans l'urne ;
              - dictionnaire Urne du nombre de boules
                de chaque couleur dans l'urne.
    Sortie : la liste des résultats de cinq tirages
             avec remise dans l'urne.
    """
    L = [] # L sera la liste des couleurs tirées.
    for t in range(5):
    # Pour t allant de 0 à 4, c'est-à-dire 5 fois :
        # on tire une boule dans l'urne
        # et on ajoute sa couleur à la liste L.
        boule = tirage(Urne, total_urne)
        L.append(boule)
    return L
```

(b) Lorsque les tirages se font sans remise, on doit actualiser le contenu de l'urne après chaque tirage :

- le nombre total de boules de l'urne diminue de 1 ;
- l'urne contient une boule de moins de la couleur tirée.

```
def cinq_tirages_sans_remise(Urne, total_urne):
    """
    Entrées : - nombre total_urne de boules dans l'urne ;
              - dictionnaire Urne du nombre de boules
                de chaque couleur dans l'urne.
    Sortie : la liste des résultats de cinq tirages
             sans remise dans l'urne.
    """
    L = []
    for t in range(5):
    # Cinq fois :
        # on tire une boule dans l'urne :
        boule = tirage(Urne, total_urne)
        L.append(boule)
        # puis on retire une boule de la couleur tirée :
        Urne[boule] = Urne[boule] - 1
        # et on retire une boule au nombre total de l'urne :
        total_urne = total_urne - 1
    return L
```

3. (a) Effectuer deux tirages simultanés dans l'urne revient à faire deux tirages successifs sans remise, et à ne pas tenir compte de l'ordre des tirages dans le résultat.

Pour oublier l'ordre des tirages, deux méthodes sont possibles.

- Une première méthode, dans laquelle le résultat est toujours donné sous forme de liste, consiste à imposer un ordre, par exemple l'ordre alphabétique, sur les deux couleurs de la liste (puisqu'étant donnée une paire de couleurs, il y a une seule manière de l'ordonner dans l'ordre alphabétique).
- Une seconde méthode consiste à renvoyer le résultat sous la forme d'un dictionnaire, dont les clés sont les couleurs et les valeurs associées les nombres de boules de chaque couleur qui ont été tirées.

Nous choisirons cette seconde méthode.

```
def deux_tirages_simultanes(Urne, total_urne):
    """
    Entrées : - nombre total_urne de boules dans l'urne ;
              - dictionnaire Urne du nombre de boules
                de chaque couleur dans l'urne.
    Sortie : le dictionnaire dont les clés sont toutes les
             couleurs des boules de l'urne et les valeurs sont
             les nombres de boules de chaque couleur tirées
             sur deux tirages sans remise dans l'urne.
    """
```

```
    Dico_resultat = {couleur : 0 for couleur in Urne}
    # Initialement, la valeur associée à chaque couleur vaut 0.
    for t in range(2):
    # Deux fois :
        # on tire une boule dans l'urne :
        boule = tirage(Urne, total_urne)
        # on incrémente le nombre de boules tirées
        # de cette couleur dans le dictionnaire-résultat :
        Dico_resultat[boule] += 1
        # puis on met à jour l'urne pour que
        # les tirages se fassent sans remise :
        Urne[boule] = Urne[boule] - 1
        total_urne = total_urne - 1
    return Dico_resultat
```

(b) Pour estimer la probabilité que, lors du tirage simultané de deux boules dans l'urne, les deux boules tirées soient rouges, on réalise un grand nombre de fois l'expérience. Sur ce grand nombre d'expériences, on calcule alors la proportion d'expériences dans lesquelles les deux boules tirées sont rouges.

```
def proba_deux_rouges(Urne, total_urne, N):
    """
    Entrées : - nombre total_urne de boules dans l'urne ;
              - dictionnaire Urne du nombre de boules
                de chaque couleur dans l'urne ;
              - nombre N de fois où l'on répète l'expérience
                de tirer deux boules simultanément.
    Sortie : proportion d'expériences (sur les N effectuées)
             où l'on tire deux boules rouges.
    """
    compteur_succes = 0
    # Compteur d'expériences où l'on tire deux boules rouges.
    for experience in range(N):
    # On répète N fois l'expérience.
        # À chaque nouvelle expérience,
        # on réalise le tirage dans une copie de l'urne,
        # pour ne pas perdre l'urne initiale.
        Urne_copie = Urne.copy()
        total_copie = total_urne
        # On tire deux boules simultanément dans l'urne.
        Resultat = deux_tirages_simultanes(Urne_copie, total_copie)
        if Resultat["rouge"] == 2:
            compteur_succes += 1
        # Si les deux boules tirées sont rouges,
            # on incrémente le compteur.
    return compteur_succes / N
```

Testons sur 10 000 expériences :

```
total_urne = 10
```

```
Urne = {"rouge" : 3, "orange" : 5, "rose" : 2}

proba_deux_rouges(Urne, total_urne, 10000)
```

On obtient que la probabilité de tirer deux boules rouges dans l'urne vaut environ $0,067$. Cette estimation est proche de la valeur exacte, qui — vous le justifierez en exercice ! — vaut :

$$\frac{\binom{2}{3}}{\binom{2}{10}} = \frac{3}{45} = \frac{1}{15} = 0,0\underline{6}...$$