.

Exercices section 9.

Testons l'aléatoire de `numpy.random`

Ouvrir un éditeur de script Python et importer les bibliothèques décrites en section 9.

1. Générer simul : $U \hookrightarrow \mathcal{U}[0;1[$
2. Créer 10 vecteurs lignes $V_1 \dots V_{10}$ composés chacun de 100 valeurs de type simul : $U \hookrightarrow \mathcal{U}[0;1[$
3. Construire une matrice/ un tableau de 10 lignes par 100 colonnes qui synthétise vos données
4. Déterminer les valeurs $\mu_1 \dots \mu_{10}$ des moyennes de chaque séries de valeurs contenues dans les vecteurs $V_1 \dots V_{10}$
   Faire de même avec les écart-types standardisés $\sigma_1 \dots \sigma_{10}$
5. Organiser la synthèse des résultats sous forme d'une matrice à deux lignes (l'une pour les moyennes, l'autre pour les écart-types)
6. Proposez une représentation graphique de vos expériences (libre choix pour tester les commandes proposéees)

Pour créer Dédé

On notera dans cette section DN le résultat d'un dé à N faces équilibrées. Le but de cette partie est de construire une simulation de DN sans avoir recourt à `rd.random` (ce qui reste un attendu des programmes).

1. Testez le programme suivant plusieurs fois (à vous de compléter par l'importation des bibliothèquers nécessaires !) :

```
U = rd.random()
X = 6*U
X = floor(X)
```

   Que renvoie ce programme ?
2. Proposez un script qui prend N en entrées et qui réalise une simulation d'un lancer de DN. Testez ensuite votre programme.
3. Complétez le programme précédant pour qu'il vérifie que $N \in \mathbb{N}^*$ et renvoie un message d'erreur si tel n'était pas le cas.
4. Simulez 100 résultats de lancés de D20 puis conjecturez la moyenne attendue pour cette expérience aléatoire d'après la liste des résultats obtenus.
5. Utiliser Python pour obtenir la moyenne et l'écart-type standardisé résultant des 100 simulations effectuées.
6. Reprenez la démarche avec 250 simulations de $D6 + D12$.

Généralisation : lois $\mathcal{U}[\![a;b]\!]$

On utilisera les bibliothèques `numpy`, `numpy.random` et `matplotlib.pyplot`

1. Programmer une fonction Python qui prend en entrée a et b et qui renvoie une réalisation d'une simulation d'un entier aléatoire compris entre a et b de sorte que chaque valeur possible survienne avec équiprobabilité.
2. Réaliser quelques essais pour vérifier.
3. Vérifier que, sur 1000 lancers, votre fonction renvoie des fréquences sensiblement identiques d'apparition de chaque valeur possible lorsque $a = -2$ et $b = 3$.

En répétant, en répétant, en répétant ...

Cette (sous)-section propose de réaliser des simulations de loi binômiale (voir cours associé)

1. Rappeler le code permetttant de réaliser la simulation d'un dé usuel (D6)
2. Réaliser 100 simulations de dés usuels dans lesquelles on dénombrera le nombre de 6 obtenus.

3. Interpréter le code suivant :

```
p = float(input("fournir un paramètre de [0;1]"))       #un tel p sera fourni
U = rd.random()
if U<= p :
        X=1
else :
        X=0
print(X)
```

En particulier, donner $\mathbb{P}[X = 1]$ et décrire la loi de X.

4. En vous aidant de la question précédante, créer une fonction `binomiale(n,p)` qui simule n variables aléatoires indépendantes $X_1 \dots X_n$ de même loi que X qui précère et renvoie le nombre K d'entre elles qui réalisent la valeur 1.
On fera le lien avec la théorie sur la loi binomiale

Avant les manchots, les chevaux

Auc jeu des petits chevaux, chaque joueur dispose d'une écurie contenant quatre chevaux. Lors de la phase d'initialisation, un joueur lance un dé et peut sortir un cheval de l'écurie si le score obtenu est "six". Sinon, son tour se termine et on passe au joueur suivant.
Nous nous placerons dans une partie à quatre joueurs (dans cet ordre) nommés *Alice*, *Bob*, *Charlie*, *Dina* et désignés plus simplement à l'aide des lettres A, B, C et D respectivement. Puis, nous allons simuler une phase d'initialisation (sortie du premier cheval)

1. Interpréter le code suivant (on complètera les bibliothèques d'importation en conformité) :

```
N=1
While rd.random() > 1/6:
        N=N+1
print("valeur de N :",N)
```

2. On pose $f(p) = p - 4 * \texttt{floor}(p/4)$ pour p entier naturel.
Compléter le tableau des valeurs suivant :

p	2	3	4	5	10	13	16	19
$f(p)$								

Expliquer concrètement l'effet de la fonction f.

3. Compléter le programme suivant afin qu'il détermine le nombre de tours écoulés lors de la sortie du premier cheval et affiche le nom du joueur associé :

```
N=1                                     # premier tour
While ........ :        # sortie écurie ?
        N=N+1
J=f(N)
if J==1:
        print("Alice")
else:
.........................
```

Introduction à `bar` et `histplot`

En SciLab, les commandes `bar` et `histplot` permettent d'afficher des diagrammes en bâtons et des histogrammes. Il s'agit ici de les tester en représentant les lois générées par SciLab et abordées en cours. Aucun compte-rendu n'est attendu pour cette section.

Pour éviter les surcharges graphiques, la commande `scf(k)` (avec k un entier naturel qu'il vous faudra donner explicitement) vous permettra dechanger de graphique sans effacer les autres. Par défaut, SciLab considère `scf(0)` à l'ouverture.

Commençons au `bar`

1. Tester la commande `bar`$([1,-3,5],0.5,'$`yellow`$')$. Réessayer en remplaçant `yellow` par `green`, puis enfin par `blue`.
2. Créer une liste L puis tracer le graphique `bar(L)`. Modifier L puis observez l'influence sur le grahique obtenu.
3. Tester les commandes `bar`$([1,-3,5],[1,-5,6;3,-2,7;4,-3,8])$ -attention aux ; et aux ,
   puis `bar`$([1,-3,5],[1,-5,6;3,-2,7;4,-3,8],'$`stacked`$')$ et comparez les résultats obtenus. Quel est le rôle de `stacked` ? (le sens du mot anglais peut être utile).
4. Déclarer une liste x de n valeurs (idéalement, n entier entre 3 et 5) ainsi qu'une matrice M de taille $n \times n$.
   Afficher le résultat de `bar(x,M)` puis comparer avec `bar(x,M,0.2)` et `bar(x,M,0.8)`. Quel est le rôle de la dernière valeur numérique ?
5. Tester enfin la commande `bar(M)` directement.
   Essayer ensuite avec d'autres tableaux M rectangulaires (nombre de lignes $\neq$ nombre de colonnes)

Passons après à `histplot`

1. Taper `histplot()` tel quel. Ceci vous montre un exemple de ce qui est faisable avec cette commande.
2. Générer une liste Y de 1000 tirages uniformes entiers entre -5 et 5. Tester ensuite `histplot(11,Y)`.
   Pourquoi avoir choisi la valeur 11 ?
3. Générer une liste Z de 1000 tirages selon la loi $\mathcal{B}(12;0.3)$. Tester ensuite `histplot(13,Z)`.
4. Utiliser la commande `histplot` pour réaliser une représentation graphique efficace de la répartition des résultats de 3D6 (on lance trois dés à 6 faces non truqués et on en fait la somme)