Institut national supérieur du professorat et de l'éducation
Académie de Paris

DIU 2nd degré – Mathématiques - Année 2025-2026
Katia.Odiot@inspe-paris.fr

D'après un travail de l'IREM de Paris – Groupe Algorithmique

Introduction à Python – Probabilités et statistiques

Dans ce qui suit, nous utiliserons le type `list` et le module random de Python, ainsi que le module tiers `matplotlib` qui permet de tracer facilement divers types de graphiques.

Partie B: Statistiques simples sur des listes

Dans cet exercice, on travaille sur des fonctions recevant une liste `lst` en paramètre. Ces fonctions exigent chacune que `lst` vérifie certaines conditions, par exemple qu'elle ne soit *pas vide,* ne contienne *que des nombres*, ou que des éléments *comparables deux à deux*.

On considérera qu'il est normal que ces fonctions provoquent des erreurs si elles reçoivent d'autres types de listes. On précisera pour chaque fonction les hypothèses attendues sur la liste `lst`.

1. On considère la fonction suivante, qui permet de calculer le maximum d'une liste non vide d'éléments comparables deux à deux :

```
def maxi(lst):
    res = lst[0]
    for elem in lst[1:len(lst)]:
        if elem > res:
            res = elem
    return res
```

   Recopier cette fonction dans un nouveau fichier Python, et la tester sur quelques listes. Que se passe-t-il si `lst` est vide ? Si c'est une liste de chaînes de caractères ? Si elle contient deux valeurs incomparables entre elles ?

2. En s'inspirant de la fonction `maxi` de la question précédente, écrire une fonction `mini_maxi(lst)` renvoyant le minimum et le maximum de la liste `lst`. Cette fonction ne devra parcourir la liste qu'*une seule fois*.

3. Écrire une fonction `somme(lst)` qui renvoie la somme des éléments de la liste non vide `lst`.

4. À l'aide de la fonction `somme`, écrire une fonction `moyenne(lst)` qui renvoie la moyenne des éléments de `lst`.

5. À l'aide de la fonction `moyenne`, écrire une fonction `ecart_type(lst)` qui renvoie l'écart-type des éléments de `lst`.

Partie C : Pile ou face, moyenne et loi des grands nombres

On souhaite observer expérimentalement le comportement d'une suite de variables (pseudo-aléatoires), sur une expérience très simple : le lancer d'une pièce équilibrée.

1. À l'aide de la fonction `randint(a, b)` du module `random`, qui renvoie un nombre entier pseudo-aléatoire entre `a` et `b` (inclus), écrivez une fonction `pile_ou_face()`, qui simule un lancer (on représentera par l'entier 0 le côté pile, et par 1 le côté face).

2. Écrivez une fonction `moyenne_des_lancers(n)` qui simule n tirages à pile ou face et renvoie la moyenne des résultats obtenus. Testez cette fonction pour quelques grandes valeurs de `n`.

3. En vous inspirant de la fonction précédente, écrivez une fonction `liste_des_moyennes(n)`, qui renvoie la liste des moyennes successives obtenues après chaque lancer.

   Attention : la moyenne doit être re-calculée après chaque nouveau lancer, on ne doit simuler que `n` lancers en tout !

4. À l'aide de la fonction `plot` du module `matplotlib.pyplot`, affichez sous la forme d'un nuage de points la liste des valeurs renvoyées par `liste_des_moyennes(n)` pour différentes valeurs de `n`.

   Une aide sur le module `matplotlib.pyplot` *est en annexe.*

5. On souhaite maintenant observer le phénomène de fluctuation d'échantillonnage en répétant plusieurs simulations de n lancers de pièces et en plaçant sur un nuage de points la fréquence obtenue.

   Écrivez une fonction `repetition_echantillonnage(n, k)` qui simule `k` fois le lancer de `n` pièces équilibrées. Cette fonction doit renvoyer la liste des k moyennes empiriques obtenues, calculées à l'aide de la fonction `moyenne_des_lancers` écrite précédemment.

6. À l'aide de la fonction `plot`, affichez sous la forme d'un nuage de points la liste des moyennes obtenues à l'aide de la fonction précédente.

7. *Optionnel* – voici quelques propositions d'approfondissement :

   a) Matérialisez les bornes de l'intervalle de fluctuation à 95 % sous la forme de lignes horizontales, à l'aide de la fonction `axhline(y)` du module `pyplot`.

   b) Calculez la proportion d'échantillons se situant hors de l'intervalle de fluctuation.

   c) Modifiez l'ensemble du code précédent afin de simuler le lancer d'une pièce biaisée ayant une probabilité `p` de tomber sur le côté pile (on utilisera pour cela la fonction `random` du module `random`).

Partie D : Lancers de dés et visualisation par histogramme

1. Grâce à la fonction `randint`, écrire une fonction `jet_deux_des()` qui renvoie le résultat d'un jet de deux dés à six face (c'est-à-dire la somme des deux dés).

2. Écrire une fonction `histogramme(n)` simulant `n` lancers de deux dés et renvoyant une liste de longueur 13, dont l'élément à chaque position `i` représente le nombre de lancers dont la somme des deux dés a valu `i`.

   Par exemple, `histogramme(4)` pourrait renvoyer la liste

   `[0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0, 0, 1]`

   ce qui signifie que sur les 5 lancers simulés, on a obtenu une fois la somme 3, deux fois la somme 5 et une fois la somme 12.

   Indice : pour initialiser une liste `res` de longueur 13 contenant uniquement des 0, on peut écrire `res = [0] * 13`.

3. En utilisant la fonction `bar` de `pyplot`, écrire un programme affichant l'histogramme pour un grand nombre de lancers.

4. Modifier l'ensemble du code précédent afin de pouvoir réaliser la simulation de `n` lancers de `k` dés à `p` faces et afficher l'histogramme correspondant, pour `n`, `k` et `p` des entiers positifs quelconques.

Partie E : Dés de Sicherman

Les *dés de Sicherman* sont une paire de dés à jouer affichant des nombres entiers différents de ceux de dés ordinaires : les faces du premier dé sont numérotées 1, 2, 2, 3, 3 et 4 et celles de l'autre dé sont numérotées 1, 3, 4, 5, 6 et 8.

On souhaite comparer la loi de probabilité de ces dés à celle de deux dés ordinaires, en calculant de manière exacte la probabilité de chaque issue possible dans chacun des cas.

1. Créer deux listes sicherman1 et sicherman2 contenant les scores possibles sur chacun des deux dés de Sicherman (dans l'ordre croissant).

2. Écrire une fonction issues(de1, de2) recevant deux listes d'entiers décrivant les numéros des faces de deux dés (dés de Sicherman ou dés classiques), et renvoyant une liste de toutes les issues possibles du lancer des deux dés.

3. Écrire une fonction `totaliser(issues)` recevant une liste de scores triée par ordre croissant (contenant des doublons) et renvoyant une liste de couples de la forme `(s, n)`, où `s` est un score et `n` est le nombre d'occurrences de ce score dans la liste issues (la liste doit être triée par scores croissants).

   Par exemple, pour des dés classiques, la liste obtenue devrait commencer par :

   `[(2, 1), (3, 2), (4, 3), ...]`

   car il y a une seule issue de score 2, deux de score 3, trois de score 4, etc.

4. Comparer (grâce à l'opérateur == de Python) les listes de totaux obtenus pour deux dés classiques avec les totaux obtenus pour les dés de Sicherman, et conclure sur la loi de probabilité de ces deux types de dés.

5. À l'aide de la fonction `bar`, afficher un histogramme décrivant le nombre d'occurrences de chaque score pour les deux types de dés (dés de Sicherman et paire de dés ordinaires). *On pourra essayer de disposer les barres de l'histogramme en regard l'une de l'autre pour mieux les comparer.*

6. Les fonctions créées dans cet exercice ont-elles toujours du sens si les listes fournies au départ sont de tailles différentes de 6 (ou différentes entre elles) ?

Partie F : Mélange uniforme de listes

On souhaite écrire une fonction qui mélange une liste de telle sorte que la permutation appliquée à la liste soit tirée de manière (pseudo-)uniforme parmi toutes les permutations possibles.

On commence par écrire la fonction suivante :

```
from random import randint

def melange_faux(lst):
    for i in range(0, len(lst)):
        n = randint(0, len(lst) - 1)
        lst[i], lst[n] = lst[n], lst[i]
```

1. Programmer et tester cette fonction sur quelques exemples.
2. Démontrer que cette fonction ne respecte pas le critère d'uniformité.

   Indication : on pourra montrer que la probabilité qu'apparaisse une permutation donnée est de la forme où est un entier et est la longueur de la liste.
3. Modifier la fonction pour qu'elle respecte le critère d'uniformité.
4. Comment faudrait-il faire pour se convaincre de l'uniformité du tirage pour la fonction corrigée ?