TD4 – Graphiques et simulation simple à partir d'OWID

Python pour économistes – Université de Lille, L2 Économie

Question fil rouge. Comment utiliser Python pour visualiser une distribution et comprendre intuitivement la stabilisation d'une moyenne ?

Livrables. Un fichier `td4_nom_prenom.py`, deux figures `.png` et une interprétation courte.

Objectifs du TD

— Construire un graphique simple avec `matplotlib`.

— Utiliser `random.choice` pour simuler des tirages.

— Observer comment une moyenne simulée évolue quand le nombre de tirages augmente.

— Identifier une `SyntaxError` et une erreur d'indentation dans une boucle.

Progression du TD

Les exercices 1 à 11 sont obligatoires. La simulation est volontairement simple : elle sert à apprendre les boucles, `random.choice` et les graphiques, pas à faire de statistique avancée.

Source des données

Les taux de fécondité sont des extraits pédagogiques arrondis d'Our World in Data.

```
https://ourworldindata.org/grapher/children-per-woman-un.csv
```

Bibliothèques utilisées

`matplotlib` sert à tracer les graphiques. `random` sert à tirer des valeurs au hasard. Si un import échoue, le problème vient probablement de l'environnement Python ou de la bibliothèque installée, pas de la logique de votre script.

Logique 1 – Visualiser puis simuler PSEUDO-CODE

```
# Objectif : observer une distribution puis simuler des tirages
# Entree : liste de taux de fecondite
# 1. Calculer la moyenne observee
# 2. Tracer un graphique en barres
# 3. Tirer au hasard une valeur dans la liste
# 4. Repeter le tirage plusieurs fois
# 5. Calculer la moyenne des tirages
# 6. Observer si la moyenne se stabilise quand n augmente
```

A. Préparer les données et un graphique

Exercice 1 – Créer l'extrait OWID

OBLIGATOIRE

Créer un dictionnaire où les clés sont des pays ou régions et les valeurs des taux de fécondité arrondis.

```python
fertilite = {
    "World": 2.3,
    "France": 1.8,
    "United States": 1.6,
    "Japan": 1.2,
    "India": 2.0,
    "Nigeria": 4.5,
    "Niger": 6.1,
    "Brazil": 1.6,
    "Ethiopia": 3.9,
    "China": 1.0
}
```

Créer ensuite la liste des valeurs et calculer la moyenne initiale. Ce nom sera réutilisé dans la simulation.

```python
valeurs = list(fertilite.values())
moyenne_initiale = sum(valeurs) / len(valeurs)
```

Afficher le nombre d'observations et la moyenne simple de ces valeurs.

Exercice 2 – Observer clés et valeurs

OBLIGATOIRE

Avant le graphique, vérifier ce que renvoient les conversions suivantes.

```python
pays = list(fertilite.keys())
valeurs = list(fertilite.values())

print(pays[:3])
print(valeurs[:3])
print(len(pays) == len(valeurs))
```

Écrire en commentaire pourquoi le graphique en barres a besoin de deux listes de même longueur.

Exercice 3 – Graphique en barres

OBLIGATOIRE

Avec `matplotlib`, tracer un graphique en barres des taux de fécondité.

```python
import matplotlib.pyplot as plt

pays = list(fertilite.keys())
valeurs = list(fertilite.values())

plt.figure()
plt.bar(pays, valeurs)
plt.xticks(rotation=45, ha="right")
plt.ylabel("Enfants par femme")
plt.title("Taux de fecondite - extrait OWID")
plt.tight_layout()
plt.savefig("fertilite_barres.png", dpi=300)
```

Exercice 4 – Modifier un paramètre graphique

OBLIGATOIRE

Reprendre le graphique en barres et faire deux modifications seulement :

a. changer le titre pour indiquer qu'il s'agit d'un extrait pédagogique ;

b. sauvegarder la figure sous le nom `fertilite_barres_version2.png`.

Vérifier que l'ancien fichier et le nouveau fichier existent tous les deux. Si un seul fichier existe, vérifier le nom passé à `plt.savefig(...)`.

Exercice 5 – Interpréter le graphique

OBLIGATOIRE

Ajouter trois commentaires :

a. quel pays a le taux le plus élevé dans l'extrait ?

b. quel pays a le taux le plus faible ?

c. pourquoi ce graphique est-il plus lisible qu'une liste brute ?

B. Simulation très guidée

Exercice 6 – Un tirage aléatoire

OBLIGATOIRE

Importer `random`, puis tirer une valeur au hasard dans la liste des taux.

```
import random

tirage = random.choice(valeurs)
print(tirage)
```

Exécuter plusieurs fois le script : le résultat est-il toujours le même ?

Exercice 7 – Prédire le rôle de range

OBLIGATOIRE

Avant de simuler 20 tirages, prédire combien de lignes seront affichées par chaque boucle.

```
for i in range(3):
    print(i)

for i in range(1):
    print("tirage")
```

Exécuter ensuite les deux fragments. Ajouter une phrase qui explique pourquoi `range(20)` permet de répéter exactement 20 tirages.

Exercice 8 – Moyenne de plusieurs tirages

OBLIGATOIRE

Compléter le code suivant pour calculer la moyenne de 20 tirages.

```
tirages = []
for i in range(20):
    tirages.append(random.choice(valeurs))

moyenne_tirages = sum(tirages) / len(tirages)
print(moyenne_tirages)
```

Comparer cette moyenne à la moyenne de la liste initiale. **Mini-débogage.** Avant de continuer, expliquer

pourquoi `for i in range(20)` sans deux-points produit une `SyntaxError`, et pourquoi une ligne non indentée après `for` produit une erreur d'indentation.

Exercice 9 – Stabilisation de la moyenne

OBLIGATOIRE

Calculer la moyenne des tirages pour plusieurs tailles : 5, 20, 100, 1000. Afficher les résultats dans une phrase.

```
tailles = [5, 20, 100, 1000]
for n in tailles:
    tirages = []
    for i in range(n):
        tirages.append(random.choice(valeurs))
    print(n, sum(tirages) / len(tirages))
```

C. Visualiser la simulation

Logique 2 – Passer des tirages à une courbe **PSEUDO-CODE**

```
# Objectif : voir si la moyenne simulee se stabilise
# Entree : plusieurs tailles de simulation
# 1. Creer une liste vide pour les moyennes
# 2. Pour chaque taille n :
#       faire n tirages
#       calculer la moyenne des tirages
#       ajouter cette moyenne a la liste
# 3. Tracer tailles en abscisse et moyennes en ordonnee
# Sortie : graphique de stabilisation
```

Exercice 10 – Graphique de stabilisation

OBLIGATOIRE

Créer deux listes : `tailles` et `moyennes_simulees`. Compléter d'abord le squelette, puis tracer le graphique.

```
tailles = [5, 20, 100, 1000]
moyennes_simulees = []

for n in tailles:
    tirages = []
    for i in range(n):
        tirages.append(random.choice(valeurs))
    moyenne_n = ...
    moyennes_simulees.append(moyenne_n)

plt.figure()
plt.plot(tailles, moyennes_simulees, marker="o")
plt.axhline(moyenne_initiale, linestyle="--")
plt.xlabel("Nombre de tirages")
plt.ylabel("Moyenne simulée")
plt.tight_layout()
plt.savefig("simulation_moyenne.png", dpi=300)
```

La variable `moyenne_initiale` doit avoir été calculée avant le graphique.

Exercice 11 – Interprétation économique

OBLIGATOIRE

Ajouter quatre commentaires :

a. que représente la moyenne de l'extrait ?

b. que montre la simulation quand le nombre de tirages augmente ?

c. pourquoi ce résultat ne décrit-il pas directement la population mondiale ?

d. pourquoi la sélection des pays est-elle importante ?

D. Entraînement facultatif

Pour aller plus loin 1 – Pour aller plus loin, sans obligation

Choisir au moins deux questions si le parcours obligatoire est terminé.

a. Fixer la graine avec `random.seed(123)` et observer l'effet.

b. Faire 10 simulations de taille 20 et comparer les moyennes obtenues.

c. Tracer un histogramme des 1000 tirages.

d. Supprimer `Niger` de la liste et recalculer la moyenne.

e. Ajouter une ligne pour la médiane de la liste initiale.

f. Écrire une fonction `moyenne_tirages(n)`.

g. Tester des tailles `10`, `50`, `500`, `5000`.

h. Calculer l'écart entre moyenne simulée et moyenne initiale.

i. Lire intuitivement la loi des grands nombres : que veut dire "stabiliser" ?

j. Extension avancée : construire un intervalle `moyenne +/- 2 * ecart_type / racine(n)` avec une formule donnée.

À retenir

— Un graphique rend une comparaison plus lisible qu'une liste brute de valeurs.
— Un import donne accès à une bibliothèque ou à un module non disponible par défaut.
— Une simulation répète une opération contrôlée pour observer un comportement.
— Dans une boucle, les deux-points et l'indentation font partie de la logique du programme.