Python pour économistes

Livret de TD appliqués avec Our World in Data – version révisée

Université de Lille – L2 Économie

Année 2026–2027

Ce livret rassemble les cinq TD. Chaque séance articule apprentissage de Python, statistiques descriptives simples, pseudo-code et applications économiques fondées sur des données Our World in Data.

Obligatoire exercices à rendre et résultats attendus.

Pseudo-code étapes à compléter avant d'écrire le Python.

Pathways chemins de fichiers, dossiers courants et sorties reproductibles.

Compléments entraînement supplémentaire pour consolider la séance.

Objectif : apprendre Python en transformant progressivement des indicateurs économiques OWID en scripts clairs, graphiques et interprétations prudentes.

Mode d'emploi du livret

Comment travailler une séance

1. Lire la question fil rouge, la source OWID et les livrables avant de coder.
2. Compléter le pseudo-code : données, étapes Python, sortie statistique, contrôle.
3. Écrire le script par petits blocs et l'exécuter souvent.
4. Vérifier les chemins de fichiers, surtout le dossier courant et le dossier `out`.
5. Quand une erreur apparaît, lire la dernière ligne, identifier le type d'erreur, puis revenir à la ligne indiquée.
6. Terminer par une interprétation courte, prudente et reliée aux résultats économiques.

Table des matières

TD1 – Premiers scripts et statistiques descriptives simples

Python pour économistes – Université de Lille, L2 Économie

Question fil rouge. Comment écrire un premier script Python pour décrire quelques indicateurs économiques et sociaux issus d'Our World in Data ?

Livrables. Un fichier `td1_nom_prenom.py` qui s'exécute sans erreur, affiche des statistiques simples, crée `esperance_vie_td1.png` et contient trois phrases d'interprétation.

Objectifs du TD

— Créer, sauvegarder et exécuter un fichier Python `.py`.

— Utiliser `print()`, des variables, des types simples et des listes.

— Produire un premier graphique très guidé avec `matplotlib`.

— Lire un premier message d'erreur et formuler une correction.

— Produire une statistique descriptive simple, un graphique et une courte interprétation.

Progression du TD

Les exercices 1 à 12 sont obligatoires. Le but principal est Python : variables, types, listes, calculs, affichage lisible, commentaires, premier graphique et premier débogage guidé. La statistique sert seulement de support.

Source des données

Les valeurs utilisées dans ce TD sont des extraits pédagogiques arrondis de graphiques Our World in Data. Pour l'instant, elles sont écrites directement dans le script afin de ne pas dépendre de pandas.

```
https://ourworldindata.org/grapher/life-expectancy.csv
https://ourworldindata.org/grapher/population.csv
```

Logique 1 – Décrire une petite liste d'indicateurs PSEUDO-CODE

Compléter la logique avant d'écrire le code.

```
# Objectif : decrire une liste de valeurs OWID
# Entree : une liste de pays et une liste de valeurs
# 1. Afficher les donnees de depart
# 2. Calculer le nombre d'observations
# 3. Calculer la moyenne
# 4. Reperer la valeur minimale et maximale
# 5. Afficher une phrase avec une unite
# Sortie : statistiques + interpretation courte
```

A. Dossier, script, exécution

Exercice 1 – Préparer un dossier de travail

OBLIGATOIRE

Créer un dossier `TD1` contenant un script `td1_nom_prenom.py`. Le nom doit se terminer exactement par `.py` : c'est l'extension qui indique que le fichier contient du code Python.

```
mkdir TD1
cd TD1
pwd
python3 td1_nom_prenom.py
```

Procéder dans cet ordre :

a. créer le fichier `td1_nom_prenom.py` ;

b. écrire `print("test TD1")` ;

c. sauvegarder le fichier ;

d. exécuter le fichier avec la commande ci-dessus ;

e. modifier le message, sauvegarder, puis relancer.

Dans le script, ajouter un commentaire qui explique la différence entre le dossier courant et le fichier exécuté. Si le terminal répond que le fichier est introuvable, vérifier d'abord le dossier courant avec `pwd` et le nom exact du fichier.

Exercice 2 – Premières variables économiques

OBLIGATOIRE

Dans le script, créer les variables suivantes puis afficher leur valeur et leur type.

```
pays = "France"
annee = 2021
esperance_vie = 82.5
population_millions = 67.7
```

Afficher ensuite une phrase lisible avec une f-string :

```
print(type(pays))
print(type(annee))
print(type(esperance_vie))
print(f"En {annee}, {pays} compte environ {population_millions} millions d'habitants.")
```

Exercice 3 – Prédire puis modifier une affectation

OBLIGATOIRE

Lire le fragment suivant sans l'exécuter.

```
population_millions = 67.7
population_millions = 68.0
print(population_millions)
```

Écrire en commentaire la valeur qui sera affichée, puis exécuter pour vérifier. Modifier ensuite seulement la deuxième ligne pour afficher `68.2`. Cet exercice sert à comprendre qu'une nouvelle affectation remplace l'ancienne valeur.

Exercice 4 – Premier débogage guidé

OBLIGATOIRE

Copier chaque ligne fautive une par une dans le script, exécuter, lire le message, puis corriger. Ne pas laisser de ligne fautive dans le rendu final.

```
print("Bonjour"
print(pay)
print("Population : " + population_millions)
```

Pour chaque erreur, noter en commentaire :

a. le type d'erreur affiché par Python ;

b. la ligne indiquée ;

c. l'hypothèse de correction.

Indice : les trois problèmes portent sur une parenthèse, un nom de variable et un mélange entre texte et nombre.

B. Listes et statistiques simples

Exercice 5 – Créer un extrait OWID simplifié

OBLIGATOIRE

Créer deux listes alignées :

```
pays = ["France", "Germany", "Spain", "Italy", "Poland", "Sweden"]
esperance_vie = [82.5, 80.6, 83.0, 82.9, 76.5, 83.1]
```

Les valeurs sont arrondies et exprimées en années. Afficher la première valeur, la dernière valeur et le nombre de pays. Avant d'exécuter, prédire ce que renvoient `pays[0]`, `esperance_vie[-1]` et `esperance_vie[10]`. Tester ensuite les deux premières expressions. Pour la troisième, expliquer en commentaire pourquoi elle produirait une `IndexError`.

Exercice 6 – Modifier une liste sans la désaligner

OBLIGATOIRE

Ajouter `"Japan"` à la liste `pays` et `84.5` à la liste `esperance_vie`. Afficher ensuite :

a. le nombre de pays ;

b. le nombre de valeurs d'espérance de vie ;

c. la phrase `Les deux listes ont la même longueur :` `True` si les deux longueurs sont identiques.

Supprimer temporairement la valeur `84.5`, relancer, puis expliquer en commentaire pourquoi deux listes alignées doivent toujours avoir la même longueur.

Exercice 7 – Moyenne, minimum, maximum

OBLIGATOIRE

À partir de la liste `esperance_vie`, calculer :

a. le nombre d'observations avec `len(...)` ;

b. la moyenne avec `sum(...) / len(...)` ;

c. le minimum avec `min(...)` ;

d. le maximum avec `max(...)`.

Afficher chaque résultat avec une unité claire : années.

Logique 2 – Passer d'une liste à un graphique PSEUDO-CODE

```
# Objectif : representer des valeurs par pays
# Entree : liste de pays, liste de valeurs
# 1. Creer une figure
# 2. Tracer une barre par pays
# 3. Ajouter un titre et une unite sur l'axe vertical
# 4. Sauvegarder la figure dans un fichier png
# Sortie : graphique lisible
```

Exercice 8 – Premier graphique avec matplotlib

OBLIGATOIRE

On trace un graphique volontairement simple. Copier le code, l'exécuter, puis modifier le titre.

```python
import matplotlib.pyplot as plt

plt.figure()
plt.bar(pays, esperance_vie)
plt.ylabel("Années")
plt.title("Espérance de vie - extrait OWID")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.savefig("esperance_vie_td1.png", dpi=300)
```

Si Python affiche `ModuleNotFoundError: No module named 'matplotlib'`, le script est lisible mais la bibliothèque manque dans l'environnement utilisé.

Exercice 9 – Modifier le graphique sans changer les données

OBLIGATOIRE

Reprendre le graphique précédent et modifier seulement :

a. le titre ;

b. le nom de l'axe vertical ;

c. le nom du fichier sauvegardé, par exemple `esperance_vie_td1_version2.png`.

Avant de relancer, sauvegarder le script. Vérifier ensuite que deux fichiers `.png` distincts existent dans le dossier du TD.

Exercice 10 – Pourcentage d'évolution

OBLIGATOIRE

Pour la France, on donne un extrait simplifié de l'espérance de vie :

```python
france_2000 = 79.1
france_2021 = 82.5
```

Calculer la variation en années, puis le taux d'évolution en pourcentage :

$$\frac{\text{valeur finale} - \text{valeur initiale}}{\text{valeur initiale}} \times 100.$$

Afficher le résultat avec deux décimales.

C. Interpréter sans surcharger

Exercice 11 – Phrase d'interprétation

OBLIGATOIRE

Ajouter trois commentaires à la fin du script :

a. une phrase qui donne la moyenne de l'extrait ;

b. une phrase qui compare la France à cette moyenne ;

c. une limite : pourquoi six pays ne suffisent pas à décrire le monde ?

Exercice 12 – Contrôle final du script

OBLIGATOIRE

Le script rendu doit contenir au minimum : une variable `str`, une variable `int`, une variable `float`, une liste, une moyenne, un graphique `matplotlib`, un pourcentage d'évolution, une f-string et trois phrases d'interprétation.

D. Entraînement facultatif

Pour aller plus loin 1 – Dix petits entraînements OWID

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

a. Ajouter `"Japan"` et une valeur d'espérance de vie dans les deux listes.

b. Créer une liste de populations en millions pour les mêmes pays.

c. Calculer la population totale de l'extrait.

d. Calculer la part de la France dans cette population totale.

e. Afficher `pays[0]` et `esperance_vie[0]` dans une même phrase.

f. Tester `esperance_vie[0] > 80` et interpréter le booléen obtenu.

g. Remplacer une valeur par une valeur manquante fictive `0`, puis expliquer pourquoi cela fausse la moyenne.

h. Créer une variable `source = "Our World in Data"` et l'afficher dans les résultats.

i. Arrondir la moyenne avec `round(moyenne, 1)`.

j. Écrire une phrase expliquant pourquoi les unités sont importantes.

À retenir

— Un script Python est un fichier sauvegardé, exécuté depuis un dossier courant.

— Une variable associe un nom à une valeur ; son type détermine les opérations possibles.

— Une liste ordonne plusieurs valeurs et s'utilise avec des indices.

— `matplotlib` permet de transformer une liste de valeurs en figure sauvegardée.

— Un message d'erreur se lit en partant du type d'erreur, puis de la ligne indiquée.

TD2 – Boucles, conditions et proportions avec données OWID

Python pour économistes – Université de Lille, L2 Économie

Question fil rouge. Comment utiliser des boucles et des conditions pour classer des observations économiques simples ?

Livrables. Un fichier `td2_nom_prenom.py` contenant une boucle, une condition, une proportion, un graphique `co2_td2.png` et une interprétation économique courte.

Objectifs du TD

— Lire une liste de dictionnaires représentant des observations.

— Utiliser une boucle `for` pour répéter une opération.

— Écrire une condition `if/elif/else` et compter une proportion.

— Construire un graphique simple avec `matplotlib` à partir d'une boucle.

— Repérer les erreurs fréquentes de boucle, compteur et indentation.

Progression du TD

Les exercices 1 à 13 sont obligatoires. La notion statistique principale est la proportion. Les seuils utilisés sont pédagogiques : ils servent à apprendre `if`, `for` et les compteurs, sans aller trop vite vers un programme complet.

Source des données

Les valeurs de CO_2 par habitant sont des extraits pédagogiques arrondis issus d'Our World in Data.

```
https://ourworldindata.org/grapher/co-emissions-per-capita.csv
```

Logique 1 – Classer des observations PSEUDO-CODE

```
# Objectif : classer des pays selon un indicateur
# Entree : liste de pays et valeurs de CO2 par habitant
# Pour chaque pays :
#     lire la valeur
#     comparer la valeur a un seuil
#     afficher une categorie
# Compter combien de pays sont dans chaque categorie
# Sortie : effectifs, proportions, interpretation
```

A. Représenter les données

Exercice 1 – Observer un dictionnaire seul

OBLIGATOIRE

Avant de manipuler plusieurs observations, lire une seule observation.

```
obs = {"pays": "France", "co2_pc": 4.7}
```

```
2
3 print(obs["pays"])
4 print(obs["co2_pc"])
```

Avant d'exécuter, prédire les deux affichages. Modifier ensuite la valeur de co2_pc à 5.1 et vérifier que seul le résultat affiché change.

Exercice 2 – Identifier une clé de dictionnaire

OBLIGATOIRE

Lire le fragment suivant avant de l'exécuter.

```
1 obs = {"pays": "France", "co2_pc": 4.7}
2
3 print(obs["pays"])
4 print(obs["population"])
```

Prédire quelle ligne fonctionne et quelle ligne produit une erreur. Exécuter ensuite le fragment, noter le type d'erreur en commentaire, puis remplacer "population" par une clé réellement présente dans le dictionnaire.

Exercice 3 – Créer une liste de dictionnaires

OBLIGATOIRE

Créer la liste suivante. Chaque dictionnaire représente une observation.

```
1  donnees = [
2      {"pays": "France", "co2_pc": 4.7},
3      {"pays": "Germany", "co2_pc": 7.9},
4      {"pays": "Spain", "co2_pc": 5.0},
5      {"pays": "Poland", "co2_pc": 8.4},
6      {"pays": "United States", "co2_pc": 14.9},
7      {"pays": "India", "co2_pc": 2.0},
8      {"pays": "China", "co2_pc": 8.0},
9      {"pays": "World", "co2_pc": 4.7}
10 ]
```

Afficher le nombre d'observations et la première observation.

Exercice 4 – Ajouter une observation courte

OBLIGATOIRE

Ajouter à la fin de donnees une observation pour "Brazil" avec co2_pc = 2.3. Relancer ensuite seulement deux contrôles :

a. `print(len(donnees))` ;

b. `print(donnees[-1])`.

Vérifier que le reste du script pourra utiliser cette nouvelle observation sans modifier les boucles.

Exercice 5 – Parcourir les observations

OBLIGATOIRE

Avec une boucle for, afficher une phrase par pays :

```
1 for obs in donnees:
2     print(f"{obs['pays']} : {obs['co2_pc']} tonnes par habitant")
```

Observer que obs prend successivement la valeur de chaque dictionnaire de la liste. Modifier ensuite l'affichage

pour arrondir à une décimale.

B. Conditions et proportions

Exercice 6 – Classer avec une condition

OBLIGATOIRE

On utilise trois catégories pédagogiques :

— `faible` si `co2_pc < 5` ;

— `intermediaire` si `5 <= co2_pc < 10` ;

— `eleve` si `co2_pc >= 10`.

Compléter la boucle suivante pour afficher la catégorie de chaque pays.

```
for obs in donnees:
    pays = obs["pays"]
    co2 = obs["co2_pc"]
    if co2 < 5:
        categorie = ...
    elif ...:
        categorie = ...
    else:
        categorie = ...
    print(pays, categorie)
```

Exercice 7 – Tester les seuils avant la boucle complète

OBLIGATOIRE

Avant d'appliquer le classement à tous les pays, tester quatre valeurs isolées : `4.9`, `5.0`, `9.9` et `10.0`. Pour chacune, écrire en commentaire la catégorie attendue, puis vérifier avec quelques lignes utilisant `if/elif/else`.

Expliquer en une phrase pourquoi le cas `5.0` doit être classé `intermediaire` et non `faible`.

Exercice 8 – Compter une proportion

OBLIGATOIRE

Compter le nombre de pays dont les émissions dépassent 5 tonnes par habitant. Calculer ensuite la proportion :

$$\frac{\text{nombre de pays au-dessus du seuil}}{\text{nombre total de pays}} \times 100.$$

Compléter d'abord ce squelette :

```
compteur = 0
for obs in donnees:
    if obs["co2_pc"] > 5:
        compteur = compteur + 1

proportion = ...
```

Afficher une phrase du type : `62.5 % des observations sont au-dessus du seuil.`

Logique 2 – Préparer des listes pour un graphique PSEUDO-CODE

```
# Objectif : tracer une valeur par pays
# Entree : liste de dictionnaires
# 1. Creer une liste vide pour les noms de pays
# 2. Creer une liste vide pour les valeurs de CO2
# 3. Parcourir les observations une par une
# 4. Ajouter le pays et la valeur dans les deux listes
# 5. Tracer puis sauvegarder un graphique
# Sortie : figure png
```

Exercice 9 – Graphique des émissions par habitant

OBLIGATOIRE

Construire deux listes avec une boucle, puis tracer un graphique en barres.

```python
import matplotlib.pyplot as plt

noms_pays = []
valeurs_co2 = []

for obs in donnees:
    noms_pays.append(obs["pays"])
    valeurs_co2.append(obs["co2_pc"])

plt.figure()
plt.bar(noms_pays, valeurs_co2)
plt.ylabel("Tonnes de CO2 par habitant")
plt.title("Emissions de CO2 par habitant - extrait OWID")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.savefig("co2_td2.png", dpi=300)
```

Exercice 10 – Créer une fonction de classement

OBLIGATOIRE

Compléter la fonction `classe_co2(valeur)`. Elle doit renvoyer `"faible"`, `"intermediaire"` ou `"eleve"`.

```python
def classe_co2(valeur):
    if valeur < 5:
        return ...
    elif valeur < 10:
        return ...
    else:
        return ...
```

La tester d'abord avec `classe_co2(4.7)`, `classe_co2(7.9)` et `classe_co2(14.9)`, puis l'utiliser dans une boucle pour afficher le classement de tous les pays.

Exercice 11 – Débogage : boucle, compteur, indentation

OBLIGATOIRE

Les trois fragments suivants contiennent une erreur fréquente. Pour chacun, identifier le type d'erreur probable ou le résultat logique faux, puis proposer une correction.

```python
# Fragment A
for obs in donnees:
print(obs["pays"])
```

```
4
5  # Fragment B
6  compteur = compteur + 1
7
8  # Fragment C
9  compteur = 0
10 for obs in donnees:
11     if obs["co2_pc"] > 5:
12     compteur = compteur + 1
```

Ne pas intégrer ces fragments fautifs au script final.

C. Interpréter

Exercice 12 – Question économique

OBLIGATOIRE

Ajouter quatre commentaires à la fin du script :

a. quel pays de l'extrait a la valeur la plus élevée ?

b. la France est-elle au-dessus ou au-dessous de la valeur mondiale ?

c. quelle proportion de l'extrait dépasse 5 tonnes par habitant ?

d. pourquoi ne faut-il pas confondre CO_2 par habitant et CO_2 total ?

Exercice 13 – Contrôle final

OBLIGATOIRE

Le script doit contenir au minimum : une liste de dictionnaires, une boucle `for`, une condition `if/elif/else`, un compteur, une proportion, une fonction, un graphique `matplotlib` et une interprétation.

D. Entraînement facultatif

Pour aller plus loin 1 – Dix petits entraînements

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

a. Changer le seuil de 5 à 7 tonnes et recalculer la proportion.

b. Créer une liste contenant seulement les pays classés `eleve`.

c. Calculer la moyenne de co2_pc avec une boucle, sans `sum()`.

d. Ajouter un pays et vérifier que le code fonctionne encore.

e. Afficher les pays au-dessous de la valeur mondiale.

f. Compter séparément les pays `faible`, `intermediaire` et `eleve`.

g. Écrire une fonction `au_dessus_seuil(valeur, seuil)`.

h. Trier manuellement deux pays : lequel a la valeur la plus élevée ?

i. Écrire une phrase sur la limite d'un classement par seuil.

j. Expliquer pourquoi une proportion dépend de l'échantillon choisi.

À retenir

— Une boucle `for` permet d'appliquer la même logique à chaque observation.
— Une condition transforme une comparaison booléenne en décision.
— Un compteur doit être initialisé avant la boucle, puis modifié au bon niveau d'indentation.
— Un graphique peut se préparer en construisant explicitement les listes à afficher.
— Une proportion dépend toujours de l'ensemble d'observations choisi.

TD3 – Fonctions et comparaison de groupes

Python pour économistes – Université de Lille, L2 Économie

Question fil rouge. Comment écrire des fonctions Python pour comparer des groupes de pays à partir d'indicateurs OWID ?

Livrables. Un fichier `td3_nom_prenom.py`, un tableau texte, un graphique `comparaison_pib_td3.png` et une interprétation de cinq lignes.

Objectifs du TD

— Distinguer paramètre, argument, affichage et valeur de retour.

— Écrire et tester des fonctions simples sur des listes numériques.

— Comparer deux groupes avec moyenne, médiane, minimum et maximum.

— Réutiliser une fonction pour produire un graphique `matplotlib`.

— Corriger des erreurs liées aux noms de variables et à `return`.

Progression du TD

Les exercices 1 à 13 sont obligatoires. Les notions statistiques obligatoires sont : moyenne, médiane, minimum, maximum et proportion. Les quartiles et l'écart-type restent facultatifs. Le point Python central est la fonction : paramètre, appel, `return` et test.

Source des données

Les valeurs de PIB par habitant sont des extraits pédagogiques arrondis d'Our World in Data.

```
https://ourworldindata.org/grapher/gdp-per-capita-worldbank.csv
```

Logique 1 – Automatiser une statistique PSEUDO-CODE

```
# Objectif : comparer deux listes de valeurs
# Entrees : liste_groupe_a, liste_groupe_b
# Definir une fonction moyenne(liste)
# Definir une fonction mediane(liste)
# Appliquer les fonctions aux deux groupes
# Comparer les resultats
# Sortie : tableau + phrase d'interpretation
```

A. Données et rappels

Exercice 1 – Créer deux groupes de pays

OBLIGATOIRE

Créer deux listes de PIB par habitant, exprimées en milliers de dollars internationaux et arrondies pour l'exercice.

```
groupe_europe = [46.0, 54.0, 34.0, 38.0, 22.0, 61.0]
```

```
groupe_monde = [46.0, 76.0, 19.0, 10.0, 7.0, 3.0]
```

Ajouter deux listes de noms pour documenter les valeurs :

```
pays_europe = ["France", "Germany", "Spain", "Italy", "Poland", "Sweden"]
pays_monde = ["France", "United States", "China", "Brazil", "India", "Ethiopia"]
```

Exercice 2 – Prédire quelques opérations sur les listes

OBLIGATOIRE

Sans écrire de fonction, prédire puis vérifier :

a. `len(groupe_europe)` ;

b. `sum(groupe_monde)` ;

c. `sorted(groupe_monde)` ;

d. `groupe_europe[0] > groupe_monde[0]`.

Ajouter une phrase qui explique pourquoi ces petits contrôles sont utiles avant de définir une fonction.

Exercice 3 – Rappel intuitif

OBLIGATOIRE

Ajouter dans le script trois commentaires :

a. la moyenne résume un niveau général ;

b. la médiane est la valeur centrale après tri ;

c. min et max donnent l'étendue des valeurs observées.

B. Fonctions

Exercice 4 – Lire une fonction avant d'en écrire une

OBLIGATOIRE

Avant d'écrire des fonctions statistiques, lire ce petit exemple.

```
def ajouter_trois(x):
    resultat = x + 3
    return resultat

print(ajouter_trois(10))
```

Répondre en commentaires :

a. quel est le paramètre de la fonction ?

b. quel est l'argument utilisé lors de l'appel ?

c. quelle ligne renvoie une valeur ?

d. quelle ligne affiche une valeur ?

Exercice 5 – Modifier une fonction courte

OBLIGATOIRE

Compléter puis modifier cette fonction.

```
def convertir_en_dollars(valeur_milliers):
    return ...

print(convertir_en_dollars(46.0))
```

La fonction doit transformer `46.0` en `46000.0`. Modifier ensuite son nom ou son paramètre, puis vérifier que l'appel utilise exactement le même nom. L'objectif est de repérer les erreurs de nom avant les fonctions statistiques.

Exercice 6 – Fonction moyenne

OBLIGATOIRE

Écrire une fonction `moyenne(valeurs)` qui renvoie la moyenne d'une liste. La tester sur les deux groupes.

```
def moyenne(valeurs):
    return sum(valeurs) / len(valeurs)
```

La tester d'abord sur `[10, 12, 14]` pour vérifier que le résultat attendu est `12.0`. Afficher ensuite les résultats des deux groupes avec une unité : `milliers de dollars par habitant`.

Exercice 7 – Fonction médiane guidée

OBLIGATOIRE

Pour une liste de longueur paire, on peut prendre la moyenne des deux valeurs centrales. Compléter la fonction :

```
def mediane_paire(valeurs):
    valeurs_triees = sorted(valeurs)
    n = len(valeurs_triees)
    milieu_droit = ...
    milieu_gauche = ...
    return ...
```

Tester d'abord la fonction sur `[8, 10, 12, 14]` : la médiane attendue est `11.0`. Tester ensuite la fonction sur les deux groupes.

Exercice 8 – Débogage de fonction

OBLIGATOIRE

Les deux fonctions suivantes ressemblent à des solutions, mais elles posent problème. Identifier l'erreur, formuler une hypothèse, puis écrire la correction.

```
def moyenne_bug(valeurs):
    resultat = sum(valeur) / len(valeurs)
    return resultat

def moyenne_sans_return(valeurs):
    print(sum(valeurs) / len(valeurs))

m = moyenne_sans_return([10, 12, 14])
print(m + 1)
```

Indice : le premier cas concerne un nom de variable ; le second permet de distinguer afficher une valeur et renvoyer une valeur.

Exercice 9 – Min, max et proportion

OBLIGATOIRE

Pour chaque groupe, calculer :

a. la valeur minimale ;

b. la valeur maximale ;

c. la proportion de pays au-dessus de 30 milliers de dollars par habitant.

Conseil : utiliser un compteur et une boucle.

C. Comparer et interpréter

Logique 2 – Comparer deux groupes par un graphique PSEUDO-CODE

```
# Objectif : comparer visuellement deux resultats
# Entree : deux listes de valeurs et une fonction moyenne
# 1. Calculer la moyenne du groupe Europe
# 2. Calculer la moyenne du groupe Monde
# 3. Placer les deux resultats dans une liste
# 4. Tracer une barre par groupe
# 5. Sauvegarder la figure
# Sortie : comparaison lisible
```

Exercice 10 – Graphique de comparaison

OBLIGATOIRE

Utiliser les fonctions écrites plus haut pour construire un graphique simple.

```python
import matplotlib.pyplot as plt

groupes = ["Europe", "Monde"]
moyennes = [moyenne(groupe_europe), moyenne(groupe_monde)]

plt.figure()
plt.bar(groupes, moyennes)
plt.ylabel("Milliers de dollars par habitant")
plt.title("PIB par habitant moyen - extrait OWID")
plt.tight_layout()
plt.savefig("comparaison_pib_td3.png", dpi=300)
```

Modifier ensuite le code pour tracer les médianes à la place des moyennes.

Exercice 11 – Déboguer un graphique

OBLIGATOIRE

Voici deux fragments qui ressemblent au code précédent mais posent problème. Identifier l'erreur probable, puis proposer une correction.

```python
# Fragment A
groupes = ["Europe", "Monde"]
moyennes = [moyenne(groupe_europe)]
plt.bar(groupes, moyennes)

# Fragment B
plt.bar(groupes, medianes)
```

Dans le fragment A, comparer la longueur des deux listes. Dans le fragment B, vérifier si la variable `medianes` a bien été créée avant son utilisation.

Exercice 12 – Tableau texte

OBLIGATOIRE

Afficher un petit tableau texte. Il doit contenir les colonnes `groupe`, `moyenne`, `mediane`, `min`, `max` et `part_au_dessus_30`. L'objectif est la lisibilité, pas la perfection typographique.

Exercice 13 – Question économique

OBLIGATOIRE

Ajouter cinq commentaires à la fin du script :

a. quel groupe a la moyenne la plus élevée ?

b. quel groupe a la médiane la plus élevée ?

c. pourquoi moyenne et médiane peuvent-elles raconter deux choses différentes ?

d. pourquoi le choix des pays influence-t-il fortement le résultat ?

e. quelle limite voyez-vous à travailler sur seulement six pays par groupe ?

D. Entraînement facultatif

Pour aller plus loin 1 – Statistiques supplémentaires sans pression

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

a. Ajouter un pays dans chaque groupe et vérifier que les fonctions marchent encore.

b. Calculer l'étendue : `max - min`.

c. Écrire une fonction `proportion_au_dessus(valeurs, seuil)`.

d. Tester le seuil de 50 au lieu de 30.

e. Écrire une fonction `resume(valeurs)` qui affiche moyenne, médiane, min et max.

f. Comparer la moyenne avant et après ajout d'une valeur très élevée.

g. Calculer un premier quartile avec une méthode simple de votre choix.

h. Calculer l'écart-type à partir d'une formule fournie par l'enseignant.

i. Écrire une phrase expliquant pourquoi l'écart-type mesure la dispersion.

j. Transformer les valeurs en dollars au lieu de milliers de dollars.

À retenir

— Une fonction sert à nommer une suite d'opérations que l'on veut réutiliser.
— Le paramètre est écrit dans la définition ; l'argument est la valeur fournie lors de l'appel.
— `return` renvoie une valeur réutilisable, alors que `print()` affiche seulement.
— Une fonction testée peut servir ensuite dans un tableau ou un graphique.
— Tester une fonction sur un cas simple permet de détecter une erreur avant de l'appliquer aux données.

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.

TD5 – Mini-projet pandas avec Our World in Data

Python pour économistes – Université de Lille, L2 Économie

Question fil rouge. Comment importer un CSV OWID, produire des statistiques descriptives simples, tracer un graphique et rédiger une interprétation économique ?

Livrables. Un fichier `td5_nom_prenom.py`, une figure `out/gdp_top10.png`, un fichier `out/gdp_selection.csv` et une interprétation de six à huit lignes.

Objectifs du TD

— Importer un fichier CSV avec pandas et inspecter sa structure.

— Filtrer un tableau, calculer des statistiques descriptives et exporter un résultat.

— Produire un graphique lisible à partir d'un `DataFrame`.

— Diagnostiquer des erreurs courantes de bibliothèque, chemin, colonne et filtre vide.

Progression du TD

Les exercices 1 à 13 sont obligatoires. Ce TD est le mini-projet final : il combine chemins de fichiers, pandas, statistiques descriptives simples, graphique et interprétation. Les premiers exercices restent guidés ; les derniers demandent davantage d'autonomie.

Pathways : travailler avec un CSV OWID

Le TD lit directement un CSV publié par Our World in Data. Si la connexion internet ne fonctionne pas en salle, télécharger le CSV avec un navigateur et placer le fichier dans un dossier `data`.

```
https://ourworldindata.org/grapher/gdp-per-capita-worldbank.csv
```

Logique 1 – Pipeline pandas final PSEUDO-CODE

```
# Objectif : mini-projet descriptif avec un CSV OWID
# 1. Lire le CSV
# 2. Inspecter lignes, colonnes et types
# 3. Renommer les colonnes utiles
# 4. Filtrer une annee et quelques pays
# 5. Calculer moyenne, mediane, min, max
# 6. Tracer un graphique lisible
# 7. Exporter un tableau propre
# 8. Rediger une interpretation prudente
```

A. Importer et inspecter

Exercice 1 – Préparer le script

OBLIGATOIRE

Créer l'arborescence suivante :

```
TD5/
  td5_nom_prenom.py
```

```
3   out/
```

Dans le script, créer le dossier `out`.

```python
from pathlib import Path

out_dir = Path("out")
out_dir.mkdir(exist_ok=True)
```

`Path("out")` désigne un dossier nommé `out` dans le dossier depuis lequel le script est exécuté. Si les fichiers produits ne sont pas retrouvés, vérifier le dossier courant et relancer après sauvegarde du script.

Exercice 2 – Vérifier le dossier courant

OBLIGATOIRE

Ajouter temporairement ces deux lignes au début du script, puis exécuter.

```python
print("Dossier courant :", Path.cwd())
print("Le dossier out existe :", out_dir.exists())
```

Comparer le chemin affiché avec l'endroit où le fichier `td5_nom_prenom.py` a été enregistré. Supprimer ensuite ces affichages si le dossier est correct, ou les garder en commentaire si cela aide à comprendre le problème.

Exercice 3 – Lire le CSV OWID

OBLIGATOIRE

Lire le fichier depuis l'URL OWID.

```python
import pandas as pd

url = "https://ourworldindata.org/grapher/gdp-per-capita-worldbank.csv"
df = pd.read_csv(url)

print(df.head())
print(df.shape)
print(df.columns)
```

Ajouter un commentaire indiquant le nombre de lignes et de colonnes. Si la connexion internet ne fonctionne pas, télécharger le CSV dans un dossier `data`, puis remplacer temporairement la ligne de lecture par :

```python
df = pd.read_csv("data/gdp-per-capita-worldbank.csv")
```

Exercice 4 – Repérer la colonne de l'indicateur

OBLIGATOIRE

Avant de renommer les colonnes, afficher seulement les noms de colonnes.

```python
print(list(df.columns))
```

Écrire en commentaire le nom exact de la colonne qui contient le PIB par habitant. Vérifier ensuite que `df.columns[-1]` correspond bien à cette colonne dans le fichier utilisé.

Exercice 5 – Renommer proprement

OBLIGATOIRE

Le nom de la colonne de valeur peut être long. On le récupère automatiquement après avoir regardé `df.columns`.

```python
value_col = df.columns[-1]

df = df.rename(columns={
    "Entity": "country",
    "Code": "code",
    "Year": "year",
    value_col: "gdp_pc"
})
```

Afficher `df[["country", "year", "gdp_pc"]].head()`. **Pourquoi `df.columns[-1]` ?** Les fichiers OWID placent souvent l'indicateur dans la dernière colonne. Ici, `-1` signifie : prendre le dernier nom de colonne. Ce n'est pas une règle universelle ; c'est une vérification pratique après inspection.

Exercice 6 – Débogage pandas guidé

OBLIGATOIRE

Pour chaque situation, identifier l'hypothèse la plus probable et le premier contrôle à effectuer.

a. `ModuleNotFoundError: No module named 'pandas'` ;

b. `FileNotFoundError: data/gdp-per-capita-worldbank.csv` ;

c. `KeyError: 'gdp_pc'` après le renommage ;

d. le script s'exécute, mais `selection` est vide.

Indices : environnement Python, dossier courant, nom exact des colonnes, orthographe exacte des pays.

B. Filtrer et décrire

Exercice 7 – Sélectionner une année récente

OBLIGATOIRE

Créer un tableau `latest` avec la dernière année disponible dans le fichier, puis retirer les valeurs manquantes.

```python
latest_year = df["year"].max()
latest = df.loc[df["year"] == latest_year].copy()
latest = latest.dropna(subset=["gdp_pc"])
print(latest_year)
print(latest.shape)
```

Exercice 8 – Sélectionner quelques pays

OBLIGATOIRE

Créer une liste de pays et filtrer le tableau.

```python
pays_selection = [
    "France", "Germany", "Spain", "Poland",
    "United States", "China", "India", "Brazil"
]
selection = latest.loc[latest["country"].isin(pays_selection)].copy()
print(selection[["country", "gdp_pc"]])
```

Exercice 9 – Diagnostiquer une sélection vide

OBLIGATOIRE

Modifier temporairement un nom de pays, par exemple "Germany" en "Germanie", puis relancer le filtre. Observer ce qui disparaît du tableau.

Ajouter ensuite ces deux contrôles :

```
print(len(selection))
print(sorted(latest["country"].unique())[:10])
```

Remettre le nom correct avant de continuer. L'objectif est de comprendre qu'un filtre peut fonctionner techniquement tout en produisant une sélection incomplète.

Exercice 10 – Statistiques descriptives simples

OBLIGATOIRE

Sur `selection`, calculer :

a. moyenne ;

b. médiane ;

c. minimum ;

d. maximum ;

e. proportion de pays au-dessus de la moyenne de la sélection.

On peut utiliser `mean()`, `median()`, `min()` et `max()`.

C. Graphique et export

Logique 2 – Passer du tableau au graphique PSEUDO-CODE

```
# Objectif : representer les dix valeurs les plus elevees
# Entree : tableau latest sans valeurs manquantes
# 1. Trier le tableau par PIB par habitant decroissant
# 2. Garder les dix premieres lignes
# 3. Tracer une barre par pays
# 4. Ajouter titre, axes et rotation des etiquettes
# 5. Sauvegarder la figure dans le dossier out
# Sortie : fichier gdp_top10.png
```

Exercice 11 – Graphique du top 10

OBLIGATOIRE

Créer un graphique en barres des dix pays les plus élevés dans `latest`.

```
import matplotlib.pyplot as plt

top10 = latest.sort_values("gdp_pc", ascending=False).head(10)
ax = top10.plot(kind="bar", x="country", y="gdp_pc", legend=False)
ax.set_xlabel("Pays")
ax.set_ylabel("PIB par habitant")
ax.set_title(f"Top 10 du PIB par habitant - {latest_year}")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.savefig(out_dir / "gdp_top10.png", dpi=300)
```

Exercice 12 – Exporter un tableau propre

OBLIGATOIRE

Exporter `selection` dans `out/gdp_selection.csv` avec seulement `country`, `code`, `year`, `gdp_pc`.

```
selection[["country", "code", "year", "gdp_pc"]].to_csv(
    out_dir / "gdp_selection.csv",
    index=False
)
```

Exercice 13 – Interprétation finale

OBLIGATOIRE

Ajouter six à huit lignes de commentaire :

a. année étudiée ;

b. pays le plus élevé dans la sélection ;

c. pays le plus faible dans la sélection ;

d. moyenne et médiane de la sélection ;

e. comparaison de la France à la moyenne ;

f. limite : choix des pays ;

g. limite : le PIB par habitant ne mesure pas toutes les dimensions du développement.

D. Extension facultative

Pour aller plus loin 1 – Pour les étudiants plus rapides

Choisir au moins deux questions si le mini-projet est terminé.

a. Tracer uniquement les pays de la sélection.

b. Comparer la France à l'Allemagne, aux États-Unis et à la Chine.

c. Créer une colonne `above_selection_mean`.

d. Exporter le `top10` dans `out/gdp_top10.csv`.

e. Tracer l'évolution du PIB par habitant de la France depuis 1990.

f. Ajouter le monde `"World"` si présent dans le fichier.

g. Remplacer `latest_year` par une année choisie manuellement.

h. Calculer le rapport entre le maximum et le minimum de la sélection.

i. Importer aussi l'espérance de vie depuis le fichier CSV OWID `life-expectancy.csv`.

j. Extension avancée : fusionner PIB par habitant et espérance de vie pour une même année.

À retenir

— Avant de traiter un CSV, il faut inspecter ses colonnes, ses dimensions et quelques lignes.

— Un `DataFrame` se filtre avec une condition booléenne et se copie quand on crée un extrait de travail.

— Les chemins de sortie doivent être construits et vérifiés pour retrouver les fichiers produits.

— Une interprétation descriptive doit citer l'année, la sélection et les limites du résultat.