**TRAVAIL PRATIQUE – MANIPULATION DE FICHIERS ET GESTION AVEC GIT**

**Nom de l'étudiant** : GOUIFE A DJIBEN ERIC CHRISTIAN Groupe : \_\_\_\_\_\_\_\_\_\_\_

**Consignes générales importantes :**

* **Chaque modification doit être suivie exactement des commandes Git suivantes dans l'ordre :**
  1. git branch nom\_branche
  2. git checkout nom\_branche
  3. git add chemin/fichier\_modifié
  4. git commit -m "Message clair expliquant la modification"
  5. Retour sur main : git checkout main
  6. Fusion de branche : git merge nom\_branche
  7. Suppression branche fusionnée : git branch -d nom\_branche
* Chaque commande Git utilisée doit être clairement inscrite dans git\_commands.txt.
* Tous les scripts Python doivent être dans le dossier scripts/.
* Tous les fichiers de données générés doivent être placés dans le dossier data/.

**1. Travailler avec des fichiers TEXTE (.txt)**

**1.1. INITIALISATION DU PROJET**

**Emplacement :** à la racine de votre espace de travail.

Complétez les commandes suivantes exactement :

➤ Vos commandes Git:

# Écrire ici vos commandes Git :

................................................................................

mkdir TP2

cd TP2

git init

git config –local user.name “Lenrics-01”

git config –local user.email [e2479537@cmaisonneuve.qc.ca](mailto:e2479537@cmaisonneuve.qc.ca)

echo « TP2 » >>README.md

git add README.md

git commit -m « Debut du projet tp2 avec README »

git branch -b main

mkdir data

mkdir scripts

echo « Commandes git utilisées » >>git\_commands.txt

➤ Exemple 1 :

Commande 1 : \_\_\_\_ # Créer un dossier nommé projet\_git\_text (nom explicite et sans espaces)

Commande 2 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Se déplacer dans le dossier nouvellement créé

Commande 3 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Initialiser un dépôt Git local dans ce dossier

Commande 4 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Créer un fichier README.md contenant une courte description du projet

Commande 5 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Ajouter le fichier README.md à la zone de staging (index Git)

Commande 6 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Effectuer un commit initial avec un message clair (ex. : "Initialisation du projet avec README")

➤ Exemple 2 :

* Allez voir exemple.md dans le même dossier ici

**1.2 Écriture sans utiliser with open()**

**Branche Git à créer :** txt-sans-with

**Emplacement :**

* Script : scripts/ecriture\_sans\_with.py
* Fichier : data/journal.txt
* Écrivez dans le fichier (journal.txt) sans utiliser with open() :

Jour 1 : Introduction

Jour 2 : Variables

Jour 3 : Conditions

Jour 4 : Boucles

Jour 5 : Fonctions

Jour 6 : Listes

Jour 7 : Dictionnaires

Jour 8 : Tuples

Jour 9 : Ensembles

Jour 10 : Manipulation de fichiers texte

Jour 11 : Manipulation de fichiers CSV

Jour 12 : Manipulation de fichiers JSON

Jour 13 : NumPy - Sauvegarde de tableaux

Jour 14 : NumPy - Chargement de tableaux

Jour 15 : Graphiques avec matplotlib

Jour 16 : Résolution d'erreurs JSON

Jour 17 : Gestion des exceptions en Python

Jour 18 : Écriture avancée avec format()

Jour 19 : Encodage de fichiers texte

Jour 20 : Conclusion et révision

➤ Votre script python :

# Écrire ici le script Python :

................................................................................ fichier=open("data/journal.txt", "w", encoding="utf-8")

    fichier.write("Jour 1 : Introduction\n")

    fichier.write("Jour 2 : Variables\n")

    fichier.write("Jour 3 : Conditions\n")

    fichier.write("Jour 4 : Boucles\n")

    fichier.write("Jour 5 : Fonctions\n")

    fichier.write("Jour 6 : Listes\n")

    fichier.write("Jour 7 : Dictionnaires\n")

    fichier.write("Jour 8 : Tuples\n")

    fichier.write("Jour 9 : Ensembles\n")

    fichier.write("Jour 10 : Manipulation de fichiers texte\n")

    fichier.write("Jour 11 : Manipulation de fichiers CSV\n")

    fichier.write("Jour 12 : Manipulation de fichiers JSON\n")

    fichier.write("Jour 13 : NumPy - Sauvegarde de tableaux\n")

    fichier.write("Jour 14 : NumPy - Chargement de tableaux\n")

    fichier.write("Jour 15 : Graphiques avec matplotlib\n")

    fichier.write("Jour 16 : Résolution d'erreurs JSON\n")

    fichier.write("Jour 17 : Gestion des exceptions en Python\n")

    fichier.write("Jour 18 : Écriture avancée avec format()\n")

    fichier.write("Jour 19 : Encodage de fichiers texte\n")

    fichier.write("Jour 20 : Conclusion et révision\n")

fichier.close()

*Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

➤ Vos commandes Git:

# Écrire ici vos commandes Git :

................................................................................

git checkout -b txt\_sans\_with

git add scripts/ecriture\_sans\_with.py

git commit -m « Création du fichier ecriture\_sans\_with.py »

python scripts/ecriture\_sans\_with.py

git add data/journal.txt

git commit -m « Création du fichier journal.txt »

git switch main

git merge txt\_sans\_with

git branch -d txt\_sans\_with

**1.3 Réécriture avec utilisation de with open()**

**Branche Git à créer :** txt-avec-with

**Emplacement :**

* Script : scripts/ecriture\_avec\_with.py
* Fichier modifié : data/journal.txt
* Réécrivez les mêmes lignes précédentes avec with open().
* Ajoutez ensuite une ligne en mode "a" : Jour 21 : Examen 1

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

.....................................

with open("data/journal.txt", "w", encoding="utf-8") as fichier:

    fichier.write("Jour 1 : Introduction\n")

    fichier.write("Jour 2 : Variables\n")

    fichier.write("Jour 3 : Conditions\n")

    fichier.write("Jour 4 : Boucles\n")

    fichier.write("Jour 5 : Fonctions\n")

    fichier.write("Jour 6 : Listes\n")

    fichier.write("Jour 7 : Dictionnaires\n")

    fichier.write("Jour 8 : Tuples\n")

    fichier.write("Jour 9 : Ensembles\n")

    fichier.write("Jour 10 : Manipulation de fichiers texte\n")

    fichier.write("Jour 11 : Manipulation de fichiers CSV\n")

    fichier.write("Jour 12 : Manipulation de fichiers JSON\n")

    fichier.write("Jour 13 : NumPy - Sauvegarde de tableaux\n")

    fichier.write("Jour 14 : NumPy - Chargement de tableaux\n")

    fichier.write("Jour 15 : Graphiques avec matplotlib\n")

    fichier.write("Jour 16 : Résolution d'erreurs JSON\n")

    fichier.write("Jour 17 : Gestion des exceptions en Python\n")

    fichier.write("Jour 18 : Écriture avancée avec format()\n")

    fichier.write("Jour 19 : Encodage de fichiers texte\n")

    fichier.write("Jour 20 : Conclusion et révision\n")

with open("data/journal.txt", "a", encoding="utf-8") as fichier:

    fichier.write("Jour 21 : Examen 1 ")

...........................................

et les commandes git sont :

git checkout -b txt\_avec\_with

git add scripts/ecriture\_avec\_with.py

git commit -m « Création du fichier ecriture\_avec\_with.py »

python scripts/ecriture\_avec\_with.py

git add data/journal.txt

git commit -m « ajout dans le fichier journal.txt »

git switch main

git merge txt\_avec\_with

git branch -d txt\_avec\_with

**1.4 Lecture numérotée du fichier texte**

**Branche Git à créer :** txt-lecture

**Emplacement :**

* Script : scripts/lecture\_journal.py
* Affichez chaque ligne de journal.txt avec le numéro de ligne :

Ligne 1 : Jour 1 : Introduction

Ligne 2 : Jour 2 : Variables

Ligne 3 : Jour 3 : Conditions

Ligne 4 : Jour 4 : Boucles

....

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

..........................................

with open("data/journal.txt", "r", encoding="utf-8") as fichier:

    file=fichier.readlines()

for i, ligne in enumerate(file):

     print(f"Ligne {i+1} : {ligne}")

......................................

git checkout -b txt-lecture

git add scripts/lecture\_journal.py

git commit -m « Creation du fichier lecture\_journal.py »

python scripts/lecture\_journal.py

git switch main

git merge txt-lecture

git branch -d txt-lecture

**2. FICHIERS CSV (.csv)**

**2.1 Création CSV avec csv.writer()**

**Branche Git à créer :** csv-creation

**Emplacement :**

* Script : scripts/creation\_csv.py
* Fichier généré : data/notes.csv

Créez un fichier CSV contenant exactement :

Nom,Note

Alice,90

Bob,72

Chloe,88

...

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

.......................................

import csv

donnée=[

   ["Nom","Note"],

   ["Alice",90],

   ["Bob",72],

   ["Chloe",88]

 ]

with open('data/notes.csv', "w", encoding="utf-8", newline='') as f:

  writer=csv.writer(f, delimiter=",")

  writer.writerows(donnée)

.........................................

git checkout -b csv-création

git add scripts/creation\_csv.py

git commit -m « Creation du fichier creation\_csv.py»

python scripts/creation\_csv.py

git add notes.txt

git commit -m “creation du fichier notes.csv

git switch main

git merge csv\_creation

git branch -d csv-creation

**2.2 Lecture brute du CSV (texte brut)**

**Branche Git à créer :** csv-brut

**Emplacement :**

* Script : scripts/lecture\_brute.py
* Lisez et affichez simplement le contenu du fichier CSV comme texte brut avec open().

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

.......................

fichier=open("data/notes.csv", "r")

contenu=fichier.read()

print(contenu)

fichier.close()

...............

..........................................

git checkout -b csv-brut

git add scripts/lecture\_brute.py

git commit -m « Creation du fichier lecture\_brute.py»

python scripts/lecture\_brute.py

git switch main

git merge csv-brut

git branch -d csv-brut

**2.3 Lecture structurée CSV (csv.reader())**

**Branche Git à créer :** csv-reader

**Emplacement :**

* Script : scripts/lecture\_structurée.py
* Lisez le CSV et affichez clairement sous la forme suivante :

Alice a obtenu 90

Bob a obtenu 72

Chloe a obtenu 88

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

...........................................

import csv

with open("data/notes.csv", "r", encoding="utf-8", newline='') as f:

    lecteur=csv.reader(f)

    next(lecteur)

    for ligne in lecteur:

        print(f"{ligne[0]} a obtenu {ligne[1]}")

.....................................

git checkout -b csv-reader

git add scripts/lecture\_structurée.py

git commit -m « Creation du fichier lecture\_structurée.py »

python scripts/lecture\_structurée.py

git add notes.csv

git commit -m “creation du fichier notes.csv

git switch main

git merge csv-reader

git branch -d csv-reader

**2.4 Conflit Git volontaire à créer**

* Branche ajout-etudiant-1 (créée depuis main)  
  Ajoutez David,95 dans notes.csv puis committez.
* Branche ajout-etudiant-2 (créée depuis main)  
  Ajoutez Emma,91 dans notes.csv puis committez.
* Fusionnez ces deux branches dans main pour provoquer un conflit.  
  Résolvez le conflit manuellement et expliquez votre résolution dans analyse.txt.

**Commandes Git à effectuer :** branch, checkout, add, commit, checkout main, merge, résoudre conflit, branch -d

➤ Votre réponse

# Écrire ici le script Python et les commandes Git dans l'ordre :

......................................

git checkout -b ajout-etudiant-1

cd data

echo « David,95 » >>notes.csv

git add data

git commit -m « ajout de donnée dans notes.csv »

git switch main

git checkout -b ajout-etudiant-2

cd data

echo « Emma,91 » >>notes.csv

git add data

git commit -m « ajout de donnée dans notes.csv »

git switch main

git merge ajout-etudiant-1 ajout-etudiant-2

nano notes.csv

git add data

git commit -m « résolution du conflit dans notes.csv »

..........................................

**3. FICHIERS JSON (.json)**

**3.1 Création JSON**

**Branche Git à créer :** json-creation

**Emplacement :**

* Fichier JSON créé : data/produits.json

Créez exactement :

{

"produits": [

{"nom": "Clé USB", "prix": 12.5},

{"nom": "Stylo", "prix": 2.0}

]

}

**Commandes Git à effectuer** (complètes)

➤ Vos Commandes Git à effectuer :

# Écrire ici le script Python et les commandes Git dans l'ordre :

....................................

git checkout -b json-creation

nano data/produits.json

git add data

git commit -m « creation de produits.json »

git switch main

git merge json-creation

git branch -d json-creation

............................................

**3.2 Conflit Git volontaire JSON à créer**

* Branche json-promo (depuis main) : Ajoutez "promo": true à "Clé USB". Committez.
* Branche json-prix (depuis main) : Changez "prix" de "Clé USB" à 10.0. Committez.
* Fusionnez ces deux branches dans main.
* Résolvez le conflit manuellement en conservant les deux changements.
* Documentez clairement votre résolution dans analyse.txt.

**4. FICHIERS NUMPY (.csv avec numpy)**

**4.1 Sauvegarde tableau NumPy**

**Branche Git à créer :** numpy-save

**Emplacement :**

* Script : scripts/save\_numpy.py
* Fichier généré : data/fibonacci.csv
* Sauvegardez le tableau [1, 2, 3, 5, 8] avec np.savetxt().

**4.2 Chargement et analyse tableau NumPy**

**Branche Git à créer :** numpy-load-analyse

**Emplacement :**

* Script : scripts/load\_analyse\_numpy.py
* Chargez fibonacci.csv, puis affichez la moyenne, le minimum, le maximum.

➤ *Votre script python et vos Commandes Git à effectuer :* (branch, checkout, add, commit, retour sur main, merge, branch -d)

# Écrire ici le script Python et les commandes Git dans l'ordre :

import csv

import numpy as np

tab=[1, 2, 3, 5, 8]

with open("data/fibonacci.csv", "w", encoding="utf-8", newline="") as f:

    writer=csv.writer(f,delimiter=",")

    writer.writerow(tab)

    np.savetxt("tes.out", tab, delimiter=",")

.......................................

import numpy as np

tab=np.loadtxt("tes.out", delimiter=",")

print(f"la moyenne est {np.mean(tab)}")

print(f"le minimun est {np.min(tab)}")

print(f"le maximum est {np.max(tab)}")

.........................................

git checkout -b numpy-save

touch scripts/save\_numpy.py

git add scripts

git commit -m « ajout du fichier save\_numpy.py »

python scripts/save\_numpy.py

git add data/fibonacci.csv

git commit -m « création du fichier fibonacci.csv »

git switch main

git merge numpy-save

git branch -d numpy-save

git checkout -b numpy-load-analyse

touch scripts/load\_analyse\_numpy.py

python scripts/load\_analyse\_numpy.py

git add scripts

git commit -m « création du fichier load\_analyse\_numpy.py »

git switch main

git merge numpy-load-analyse

git branch -d b numpy-load-analyse

**5. QUESTIONS FINALES (analyse.txt à compléter précisément)**

1. Pourquoi est-il conseillé d'utiliser with open() plutôt que open()/close() ? Réponse : Parce qu’avec with open() on ne risque pas d’oublier de fermer le fichier après l’avoir ouvert car il ferme le fichier automatiquement.
2. Expliquez précisément comment vous avez résolu les conflits Git créés dans ce TP : Réponse : Pour ressourdre le conflit manuellement j'ai juste eu à supprimer les lignes qui indiquaient les erreurs pour ensuite garder les informations qui ont été ajoutées et supprimer celle que je ne veux pas. tout cela dans le fichier qui se trouve dans la branche \*main\*
3. Donnez la commande Git pour restaurer un fichier supprimé accidentellement : Réponse : git restore

**6. VÉRIFICATION FINALE**

Exécutez cette commande exactement dans votre dépôt à la fin du TP :

git log --oneline --graph

Copiez le résultat exact dans l'espace suivant :

# Collez ici le résultat exact de git log --oneline --graph :

\* 4c9d387 (**HEAD** -> **main**) ajout dans git\_commands.txt

\* 52ace3f creation du fichier load\_analyse\_numpy.py et du fichier tes.out

\* 64d156e creation du fichier fibonacci

\* 1f710bb creation du fichier save\_numpy.py

\* 650d961 creation du fichier save\_numpy.py

\* 9469ade creation du fichier save\_numpy.py

\* eb2e70a (**origin/main**, **origin/HEAD**) ajout de analyse.txt

\* 97f37f5 merge resolu

|\

| \* 58acb3c creation du fichier lecture\_brute.py

\* | ce707cd ajout des modifications

|/

\* 31e720b changement effectué sur ecriture\_sans\_with.py

\* c1e60bf ajout du tp2\_parti\_word

\* 078b777 ajout dans analyse.txt

|\

| \* 5885fda ajout de donnée dans data

\* | 949fa6d ajout de donnée dans data

|/

\* a502b3f ajout de donnée dans data

\* bcd5c91 ajout de donnée dans data

\* d5fa728 ajout dans git\_commands.txt

\* 6ec59b3 ajout de analyse.txt

|**\**

| \* 9244fe6 ajout de donnée dans notes.csv

\* **|** 0fe6cc4 ajout de donnée dans notes.csv

**|/**

\* 6dc250f ajout des history

\* 9b72ced ajout d'un lecteur de fichier csv

\* 02112f8 ajout du fichier csv

\* c5b676a ajout du fichier de création de csv

\* 6064208 (**origin/csv-creation**) nouvelle commande dans git\_commands

\* eaba3f5 nouvelle commande dans data

\* 594a64b nouveau scripts

\* 611094a nouveau scripts

\* 1b33750 nouvelle donnée dans git\_commands

\* e710919 nouvelle donnée dans data

\* 7b6d2f7 nouveau scripts

\* 704e405 ajout du dossier scripts

\* e32929a ajout de git\_commands.txt

\* 2cc815c first commit

**7. QUESTION FINALE – SCÉNARIO RÉALISTE AVEC DONNÉES COMPLEXES**

**7.1. Objectif pédagogique**

Mobiliser vos compétences sur :

* Lecture et filtrage de données CSV avec en-tête
* Traitement de données numériques avec NumPy
* Commandes Git avancées : suppression, restauration, historique, résolution de conflit
* Automatisation d’une analyse avec génération conditionnelle d’un fichier JSON

**7.2. CONTEXTE DU SCÉNARIO :**

Vous travaillez sur un fichier de résultats data/reponses.csv, contenant les réponses à un questionnaire de satisfaction. Il est structuré comme suit :

Nom,Prénom,Note

Garcia,Ana,4

Dubois,Jean,5

Nguyen,Linh,4

Smith,John,3

...

* Le fichier contient **20 lignes de réponses valides**.
* Le fichier complet peut être copié de l'annexe ci-bas (fichier responses.csv)
* Il doit être **commité dans main** au début du projet.

Un autre fichier data/meta.json contient les métadonnées associées :

{

"questionnaire": "Évaluation du cours",

"type": "satisfaction",

"nb\_repondants": 20

}

**7.3. ÉTAPE 1 – Suppression accidentelle d’un fichier critique**

1. À partir de main, créez la branche analyse-1.
2. Dans cette branche, **supprimez** le fichier data/reponses.csv :

rm data/reponses.csv

git rm data/reponses.csv

git commit -m "Suppression accidentelle du fichier de réponses"

**7.4. ÉTAPE 2 – Échec d’analyse dans une autre branche**

1. Revenez dans main, puis créez la branche analyse-2.
2. Dans cette branche, vous tentez de faire une analyse avec ce script :

import numpy as np

data = np.loadtxt("data/reponses.csv", delimiter=",", skiprows=1, usecols=2)

Cela génère une **erreur**, car le fichier a été supprimé dans l’autre branche.

**7.5. ÉTAPE 3 – Restauration correcte avec Git**

1. **Identifiez le commit** dans lequel le fichier existait encore (git log, git checkout, etc.).
2. **Restaurez** le fichier supprimé dans la branche analyse-2, sans le recréer manuellement :

git checkout main data/reponses.csv

1. Commitez cette récupération :

git add data/reponses.csv

git commit -m "Restauration du fichier supprimé depuis l’historique"

**7.6. ÉTAPE 4 – Script d’analyse avancée**

Créez le fichier suivant :  
scripts/analyse\_satisfaction.py

Il doit :

* Lire la colonne Note du fichier CSV (usecols=2)
* Calculer :
  + Moyenne des notes
  + Médiane
  + Pourcentage de notes supérieures ou égales à 4
* Afficher les résultats sous forme lisible
* Si la moyenne est inférieure à 3.0, **générer automatiquement un fichier data/alerte.json** :

{

"alerte": true,

"moyenne": 2.6,

"seuil": 3.0,

"message": "Satisfaction étudiante anormalement basse"

}

Sinon, **ne rien créer**.

**7.7. À REMPLIR DANS VOTRE RAPPORT**

1. Script complet scripts/analyse\_satisfaction.py :

# Script Python :

................................................................................

1. Liste complète des commandes Git utilisées (suppression, checkout, restauration, commit, etc.) :

# Commandes Git :

................................................................................

1. Contenu final du fichier data/reponses.csv (après restauration) :

# Contenu CSV :

................................................................................

1. Si généré, collez le contenu de data/alerte.json :

# JSON :

................................................................................

**Rappels importants :**

* Ne modifiez jamais manuellement les fichiers supprimés : utilisez Git.
* np.loadtxt() ou np.genfromtxt() peuvent tous deux être utilisés.
* N’oubliez pas skip\_header=1 si vous avez une ligne d’en-tête.
* Si vous utilisez csv.reader, veillez à convertir les notes en int ou float.

**8. LIVRABLES À FOURNIR EXACTEMENT**

**RAPPORT À REMETTRE**

Remettez **un seul fichier** nommé :

tp\_fichiers\_NOM\_PRENOM.docx (ou .pdf, .txt ou .py)

Le rapport doit contenir :

* Le **code Python complet** pour chaque étape (copié dans le document)
* Les **commandes Git** associées à chaque script (branch, checkout, add, commit, merge, branch -d)
* Les **résolutions de conflits Git**, avec explication claire
* Les **réponses aux questions finales**
* Le **résultat complet de** :
* git log --oneline --graph

Aucun .zip ou dossier séparé n’est accepté. Le code et les commandes doivent être intégrés dans ce seul fichier.

**Vérifiez soigneusement chaque étape avant remise du TP.**

**Annexe : Fichiers à utiliser dans le TP**

**data/journal.txt**

Jour 1 : Introduction

Jour 2 : Variables

Jour 3 : Conditions

Jour 4 : Boucles

Jour 5 : Fonctions

Jour 6 : Listes

Jour 7 : Dictionnaires

Jour 8 : Tuples

Jour 9 : Ensembles

Jour 10 : Manipulation de fichiers texte

Jour 11 : Manipulation de fichiers CSV

Jour 12 : Manipulation de fichiers JSON

Jour 13 : NumPy - Sauvegarde de tableaux

Jour 14 : NumPy - Chargement de tableaux

Jour 15 : Graphiques avec matplotlib

Jour 16 : Résolution d'erreurs JSON

Jour 17 : Gestion des exceptions en Python

Jour 18 : Écriture avancée avec format()

Jour 19 : Encodage de fichiers texte

Jour 20 : Conclusion et révision

**data/notes.csv**

Nom,Note

Alice,90

Bob,72

Chloe,88

**data/produits.json**

{

"produits": [

{"nom": "Clé USB", "prix": 12.5},

{"nom": "Stylo", "prix": 2.0}

]

}

**data/responses.csv**

Nom,Prénom,Note

Garcia,Ana,4

Dubois,Jean,5

Nguyen,Linh,4

Smith,John,3

Tremblay,Marie,2

Zhao,Yan,1

Roy,Éric,4

Mendes,Carlos,5

Bouchard,Laurie,5

Khan,Imran,3

Ouellet,Lucie,4

Kim,Min,2

Martin,Chloé,1

Benoit,Paul,2

Bergeron,Julie,3

Côté,Alex,5

Lavoie,Noémie,5

Ahmed,Yasmin,4

Gagnon,Sylvain,3

Lemay,Maude,2