TP Chapitre 10 : Traitement de données en tables

Objectifs du TP

Objectifs du TP. À l'issue de cette séance, tu dois être capable de :

- représenter une table en Python par une liste de listes (tableau doublement indexé) et accéder à une case avec `t[i][j]` ;
- importer une table depuis un fichier CSV (lecture, découpage avec `split`, conversion des valeurs) ;
- parcourir une table et calculer sur une colonne (somme, moyenne, maximum) ;
- rechercher les lignes d'une table vérifiant des critères exprimés en logique propositionnelle (`and`, `or`, `not`) ;
- détecter les doublons (`set`) et effectuer des tests de cohérence avant de calculer ;
- trier une table suivant une colonne (`sorted` avec `key`, ou `sort_values` avec pandas) ;
- fusionner deux tables de même structure (concaténation avec `+`, ou `pd.concat` avec pandas).

Consignes générales. Tous les exercices se traitent en Python (console ou fichier). Dans ce TP, une table est une *liste de listes* : chaque ligne est un enregistrement, chaque colonne un champ (descripteur), et `t[i][j]` désigne la case de la ligne i et de la colonne j. Sauf mention contraire, les données sont fournies directement dans le code. Enregistre régulièrement ton travail. Quand un programme ne fonctionne pas, lis le message d'erreur : il indique souvent la ligne et le type d'erreur. La partie B se fait *en autonomie* : tu écris des programmes entiers à partir d'une spécification, aucune ligne de code n'est fournie.

Partie A — Application guidée

Les exercices suivants se programment en Python. Ils sont classés du plus simple au plus difficile. Chaque exercice se termine par un *point de validation* (encadré vert) : le résultat exact que doit produire ton programme pour être juste. Complète les squelettes de code en respectant l'indentation.

Exercice A1 — La caisse du cartable : prix TTC (parcours d'une table)

Une papeterie affiche ses prix hors taxes. Le prix TTC vaut prix HT × 1,2 (TVA à 20 %). La table `articles` contient, pour chaque article, son nom et son prix HT en euros. Compléter le programme pour qu'il affiche, pour chaque article, son prix TTC.

```
articles = [["Cahier", 2.50], ["Stylo", 1.20], ["Trousse", 5.00], ["Sac",
    15.00]]

for article in articles:
    # A COMPLETER : calculer le prix TTC (prix HT * 1.2)
    print(article[0], ":", prix_ttc, "euros")
```

Point de validation

Ton programme doit afficher exactement quatre lignes : `Cahier : 3.0 euros`, `Stylo : 1.44 euros`, `Trousse : 6.0 euros` et `Sac : 18.0 euros`.

Exercice A2 — Le portefeuille d'actions (somme d'une colonne)

Tu possèdes quelques actions en bourse. La table `actions` contient, pour chaque ligne, le nom de l'action, le nombre d'actions possédées et le prix unitaire en euros. La valeur d'une ligne vaut *quantité*

$\times$ *prix*. Compléter le programme pour qu'il affiche la valeur totale de ton portefeuille (la somme des valeurs de toutes les lignes).

```
actions = [["TotalEnergies", 12, 58.40],
           ["Air France-KLM", 30, 8.75],
           ["BNP Paribas", 5, 62.30],
           ["Danone", 10, 55.20]]

total = 0
for action in actions:
    # A COMPLETER : valeur de la ligne (quantite * prix) puis ajout au total

print("Valeur totale du portefeuille :", total, "euros")
```

Point de validation

Ton programme doit afficher `Valeur totale du portefeuille : 1826.8 euros`.

Exercice A3 — Le contrôle des dépenses (recherche dans une table)

Tu surveilles tes dépenses de janvier. La table `depenses` contient, pour chaque ligne, la date, la catégorie et le montant en euros.

```
depenses = [["03/01", "Courses", 54.20],
            ["05/01", "Transport", 12.40],
            ["08/01", "Restaurant", 38.50],
            ["12/01", "Courses", 61.30],
            ["15/01", "Essence", 45.00],
            ["20/01", "Restaurant", 21.80]]
```

1. Compléter le programme suivant pour qu'il affiche uniquement les lignes dont le montant dépasse le seuil de 40 €.

```
seuil = 40
for ligne in depenses:
    # A COMPLETER : afficher la ligne si le montant depasse le seuil
```

2. Compléter le programme suivant pour qu'il affiche uniquement les lignes dont la catégorie est `"Courses"` *et* dont le montant dépasse 50 €. Utiliser l'opérateur `and`.

```
for ligne in depenses:
    # A COMPLETER : deux conditions avec and
```

Point de validation

Question 1 : trois lignes doivent s'afficher : `03/01 Courses 54.2`, `12/01 Courses 61.3` et `15/01 Essence 45.0` (l'ordre peut varier selon l'affichage choisi). Question 2 : deux lignes : `03/01 Courses 54.2` et `12/01 Courses 61.3`.

Exercice A4 — Les catégories de dépenses (recherche de doublons)

On reprend la table `depenses` de l'exercice A3. Certaines catégories reviennent plusieurs fois : ce sont des *doublons*. Compléter le programme pour qu'il affiche la liste des catégories (une par ligne de la table), puis le nombre de catégories *distinctes*.

```python
depenses = [["03/01", "Courses", 54.20],
            ["05/01", "Transport", 12.40],
            ["08/01", "Restaurant", 38.50],
            ["12/01", "Courses", 61.30],
            ["15/01", "Essence", 45.00],
            ["20/01", "Restaurant", 21.80]]

categories = [ligne[1] for ligne in depenses]
print("Categories :", categories)
# A COMPLETER : afficher le nombre de categories distinctes (utiliser set)
```

Point de validation

Ton programme doit afficher `Categories : ['Courses', 'Transport', 'Restaurant', 'Courses', 'Essence', 'Restaurant']` puis `4` (le nombre de catégories distinctes).

Partie B — En autonomie

Tu écris maintenant des programmes *entiers*, sans squelette. Pour chaque exercice : lis la spécification (entrées, traitement, sorties, contraintes), réponds d'abord aux questions de conception de l'encadré orange (sur papier ou à voix haute), puis écris ton programme dans un fichier et teste-le avec les valeurs de l'encadré vert. L'objectif est de concevoir toi-même la solution : ne demande pas de lignes de code toutes faites.

Exercice B1 — Le budget du mois : lire un fichier CSV (import d'une table)

Tu as noté tes dépenses de janvier dans le fichier `depenses.csv`. C'est un fichier texte dont les champs sont séparés par des points-virgules ; la première ligne est l'en-tête (les noms des colonnes).

```
Date;Categorie;Montant
03/01;Courses;54.20
05/01;Transport;12.40
08/01;Restaurant;38.50
12/01;Courses;61.30
15/01;Essence;45.00
20/01;Restaurant;21.80
```

Énoncé. Créer le fichier `depenses.csv` avec ce contenu (dans le même dossier que ton programme), puis écrire un programme qui lit ce fichier ligne par ligne et affiche le total des dépenses et la dépense moyenne.

Spécification.

— *Entrées* : le fichier `depenses.csv` (en-tête + 6 lignes de données).

— *Traitement* : ouvrir le fichier avec `open` (et `with`) ; pour chaque ligne, enlever le retour à la ligne (`strip`), découper les champs (`split(";")`), sauter l'en-tête, convertir le montant (`float`), l'ajouter au total ; compter les dépenses.

— *Sorties* : `Total des depenses : ... euros` puis `Depense moyenne : ... euros`.

— *Contraintes* : le montant est le troisième champ de chaque ligne ; la première ligne (l'en-tête) ne doit pas être comptée comme une dépense ; `encoding="utf-8"` dans `open`.

Pour t'aider — questions de conception (sans donner le code)

- Pourquoi `split(";")` et pas `split(",")` ? (pense aux nombres décimaux français)
- Comment faire pour que l'en-tête `Date;Categorie;Montant` ne soit pas traité comme une dépense ?
- Pourquoi faut-il écrire `float(champs[2])` et pas garder `champs[2]` tel quel ?
- Comment vérifier à la main que le total est juste ? (additionne les six montants)

Point de validation

Ton programme doit afficher un total proche de `233.2` euros (un léger écart d'arrondi est normal) et une moyenne proche de `38.87` euros.

Exercice B2 — Le portefeuille trié : actions au-dessus d'un seuil (recherche et tri)

Tu complètes ton analyse boursière : tu veux ne garder que les actions qui valent vraiment quelque chose, et les classer de la plus grosse à la plus petite.

Énoncé. Écrire un programme qui calcule la valeur de chaque ligne de la table `actions` (quantité × prix), ne garde que les lignes dont la valeur dépasse 300 €, puis affiche ces actions triées par valeur *décroissante*.

Spécification.

— *Entrées* : la table `actions` définie dans le programme :

```
actions = [["TotalEnergies", 12, 58.40],
           ["Air France-KLM", 30, 8.75],
           ["BNP Paribas", 5, 62.30],
           ["Danone", 10, 55.20],
           ["Veolia", 20, 28.10]]
```

— *Traitement* : pour chaque ligne, calculer `valeur = quantite * prix` ; si `valeur > 300`, conserver la ligne ; trier la sélection par valeur décroissante (`sorted` avec `key` et `reverse=True`).

— *Sorties* : une ligne par action conservée : `Nom valeur euros`, de la plus grosse valeur à la plus petite.

— *Contraintes* : la table d'origine ne doit pas être modifiée (`sorted` renvoie une nouvelle liste) ; le tri porte sur la valeur calculée, pas sur le prix unitaire.

Pour t'aider — questions de conception (sans donner le code)

- Comment écrire la clé de tri pour ranger des lignes `[nom, valeur]` selon la valeur ? (rappel : `key=lambda ligne: ligne[1]`)
- Pourquoi `reverse=True` est-il nécessaire ?
- Où construire la liste des actions conservées : avant ou après le calcul de la valeur ?
- Que se passerait-il si l'on triait avant de filtrer ? (le résultat serait-il faux ?)

Point de validation

Ton programme doit afficher, dans cet ordre : `TotalEnergies 700.8 euros`, `Veolia 562.0 euros`, `Danone 552.0 euros`, `BNP Paribas 311.5 euros` (Air France-KLM, 262.5 €, est exclue : elle est sous le seuil).

Exercice B3 — La fusion des ventes : deux canaux, une table (fusion et tri avec pandas)

Un petit commerce vend en ligne et en magasin. Les ventes des deux canaux ont la même structure : un article et son prix. On veut réunir les deux tables en une seule, puis la trier par prix décroissant.

Énoncé. Écrire un programme qui construit deux `DataFrame` pandas à partir des listes `ventes_web` et `ventes_magasin` (colonnes `["Article", "Prix"]`), les fusionne en une seule table, affiche le nombre total de lignes, puis affiche la table triée par prix décroissant.

Spécification.

— *Entrées* : les deux listes de listes :

```
ventes_web = [["Cahier", 2.50], ["Trousse", 5.00], ["Sac", 15.00]]
ventes_magasin = [["Stylo", 1.20], ["Sac", 15.00], ["Cahier", 2.50]]
```

— *Traitement* : `import pandas as pd` ; construire les deux `DataFrame` avec `pd.DataFrame(liste, columns=["Article", "Prix"])` ; fusionner avec `pd.concat` ; trier avec `sort_values("Prix", ascending=False)`.

— *Sorties* : `Nombre de lignes : ...` puis la table fusionnée triée.

— *Contraintes* : les deux tables doivent avoir les mêmes colonnes (même *domaine de valeurs*) ; la fusion ne modifie pas les tables d'origine.

Pour t'aider — questions de conception (sans donner le code)

— Pourquoi les deux tables doivent-elles avoir exactement les mêmes colonnes pour être fusionnées ?

— Que fait `pd.concat` ? À quoi ressemble le résultat quand les deux tables ont une ligne « Sac » chacune ?

— Comment trier par prix *décroissant* avec pandas ?

Point de validation

Ton programme doit afficher `Nombre de lignes : 6` puis une table triée dont la première ligne est `Sac` (prix 15.0) et la dernière `Stylo` (prix 1.2).

Exercice B4 — Le relevé bancaire sous contrôle (test de cohérence)

Ton relevé bancaire contient parfois la mention `"n.c."` (non communiqué) à la place d'un montant. Avant de calculer un total, il faut détecter ces valeurs : c'est un *test de cohérence*.

Énoncé. Écrire un programme qui parcourt la table `releve`, signale chaque ligne dont le montant vaut `"n.c."` (sans la convertir !), compte ces lignes, puis calcule et affiche le total des montants valides (toutes les autres lignes).

Spécification.

— *Entrées* : la table `releve` définie dans le programme :

```
releve = [["03/01", "Salaire", 1850.00],
          ["05/01", "Loyer", -720.00],
          ["08/01", "Courses", "n.c."],
          ["12/01", "Transport", -45.00],
          ["15/01", "Epargne", -200.00]]
```

— *Traitement* : pour chaque ligne, tester si le montant vaut `"n.c."` (comparaison de chaînes) ; si oui, l'afficher et incrémenter un compteur ; sinon, convertir avec `float` et l'ajouter au total.

— *Sorties* : une ligne par valeur manquante (`Valeur manquante : date catégorie`), puis `Lignes incoherentes : ...` et `Total des montants valides : ... euros`.

— *Contraintes* : ne jamais appeler `float` sur une valeur `"n.c."` ; le montant est le troisième champ de chaque ligne.

Pour t'aider — questions de conception (sans donner le code)

— Que se passe-t-il si l'on écrit `float("n.c.")` ? (teste-le en console si besoin)

— Pourquoi faut-il tester la cohérence *avant* de faire le calcul ?

— Un montant négatif est-il une donnée incohérente ici ? (que représente-t-il ?)

Point de validation

Ton programme doit signaler `Valeur manquante : 08/01 Courses`, afficher `Lignes incoherentes : 1` et `Total des montants valides : 885.0 euros`.

Partie C — Exercice de réflexion (à finir à la maison)

À finir à la maison

Mon budget mensuel : analyse complète d'un fichier de transactions.

Contexte. Ton relevé bancaire de janvier est exporté dans le fichier `transactions.csv` (champs séparés par des points-virgules). Une recette (salaire, petit boulot...) a un montant *positif* ; une dépense (loyer, courses...) a un montant *négatif*. Parfois, le montant est marqué `n.c.` (non communiqué) : ce n'est pas un nombre.

```
Date;Categorie;Montant
02/01;Salaire;1850.00
03/01;Loyer;-720.00
05/01;Courses;-86.40
08/01;Transport;-58.20
10/01;Restaurant;-42.50
12/01;Freelance;150.00
15/01;Courses;-91.30
18/01;Essence;-45.00
20/01;Loisirs;-65.00
24/01;Loisirs;n.c.
28/01;Epargne;-150.00
```

Objectif. Écrire un programme qui importe cette table, vérifie sa cohérence, puis répond aux questions : quel est le total des recettes ? le total des dépenses ? le solde du mois (recettes − dépenses) ? quelle est la catégorie de dépense la plus coûteuse ? Le mois est-il équilibré ?

Questions à se poser avant de coder.

— Quelle structure choisir : liste de listes (comme dans la partie B) ou `DataFrame` pandas ? Quels sont les avantages de chacune pour ce problème ?

— Comment repérer les valeurs `n.c.` avant de convertir ? (tu l'as fait dans l'exercice B4)

— Comment distinguer une recette d'une dépense ? (pense au signe du montant)

— Comment regrouper les dépenses par catégorie pour trouver la plus coûteuse ? (accumulateur par catégorie, dictionnaire, ou table à deux colonnes)

— Comment vérifier que le programme est juste ? (recettes 2000 €, dépenses 1258,40 € sur les valeurs valides)

Étapes suggérées.

1. Version 0 : importer le fichier et afficher toutes les lignes (vérifie que les champs sont bien découpés).
2. Version 1 : afficher le total des recettes et le total des dépenses, puis le solde.
3. Version 2 : ajouter le test de cohérence — les lignes `n.c.` sont signalées et exclues des calculs.
4. Version 3 : calculer le total des dépenses par catégorie, puis afficher la catégorie la plus coûteuse.
5. Version 4 : afficher un bilan final lisible : recettes, dépenses, solde, catégorie la plus coûteuse, et une conclusion « mois équilibré » ou « mois déficitaire ».

Contraintes. Données crédibles (le fichier fourni est un exemple, tu peux le compléter avec tes propres dépenses) ; programme testé à chaque version et commenté ; les messages peuvent être écrits sans accents (ex. `Recettes`, `Depenses`) si ton éditeur pose problème. Le plus important est que chaque version fonctionne avant de passer à la suivante.

Fin du TP — la partie C est à finir à la maison.