NSI · PREMIÈRE / COURS

Traiter des données en tables

Astrolab Cours · astrolabcours.fr

01 Le problème

Une ville publie la liste de ses 2 000 arbres : espèce, hauteur, quartier, année de plantation. On veut les platanes de plus de 20 m, ou les arbres triés par hauteur. À la main, c'est des heures ; dans un tableur, c'est possible ; en Python, c'est quelques lignes réutilisables.

01 L'idée : le tableur interrogé en Python

Imagine un tableur dont chaque ligne serait une fiche, avec des étiquettes en haut de colonne. En Python, chaque ligne devient un dictionnaire dont les clés sont ces étiquettes. La table entière est une liste de ces fiches.

01 Le format CSV

Un fichier **CSV** (*comma-separated values*) est un fichier texte : la première ligne donne les noms des colonnes (les **descripteurs**), chaque ligne suivante est un enregistrement, les valeurs sont séparées par une virgule ou un point-virgule.

```
espece;hauteur;quartier
platane;24;Centre
tilleul;12;Nord
platane;18;Sud
```

01 Lire un CSV

```python
import csv

with open("arbres.csv", encoding="utf-8") as f:
    table = list(csv.DictReader(f, delimiter=";"))

table[0]   # {'espece': 'platane', 'hauteur': '24', 'quartier': 'Centre'}
```

Toutes les valeurs lues sont des chaînes : on convertit les nombres.

```python
for ligne in table:
    ligne["hauteur"] = int(ligne["hauteur"])
```

© Astrolab Cours · astrolabcours.fr/nsi/premiere/donnees-en-tables/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits

Sélectionner

```
grands_platanes = [a for a in table
                   if a["espece"] == "platane" and a["hauteur"] > 20]
```

Trier

`sorted` reçoit une fonction `key` qui dit sur quelle colonne trier :

```
par_hauteur = sorted(table, key=lambda a: a["hauteur"], reverse=True)
```

Fusionner deux tables

Si une seconde table donne la population de chaque quartier, on associe chaque arbre à son quartier en comparant la colonne commune `quartier` : c'est une **jointure**, comme en SQL.

Exemple guidé

La table `arbres` est une liste de dictionnaires lue depuis le CSV, avec les champs `espece`, `hauteur`, `quartier`. Trouver les platanes de plus de 20 m, triés du plus grand au plus petit.

```
def hauteur(a):
    return float(a["hauteur"])

grands = [a for a in arbres
          if a["espece"] == "platane"
          and hauteur(a) > 20]
grands.sort(key=hauteur, reverse=True)
```

1. **Sélection** : la liste en compréhension garde les lignes qui vérifient les deux conditions.
2. **Conversion** : dans un CSV, tout est du texte. On convertit la hauteur en nombre avant de comparer, sinon `"9" > "20"` serait vrai.
3. **Tri** : `key` indique sur quel critère trier, `reverse=True` donne l'ordre décroissant.

Retour au problème

Quelques lignes de Python remplacent des heures de tri à la main, et se réutilisent quand la ville met la table à jour.

Les pièges

- Oublier de convertir : `"9" > "12"` est vrai (ordre alphabétique).
- Se tromper de séparateur : avec `;` dans le fichier, préciser `delimiter=";"`.
- Les accents mal lus : préciser `encoding="utf-8"`.

© Astrolab Cours · astrolabcours.fr/nsi/premiere/donnees-en-tables/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits