

Liste de liste [ [],[],…] avec

Juste une ligne d’entête [clé1, clé2 ..] …

Liste de dictionnaire [ {clé1 : valeur}, {clé2 : }, {},…] avec

(Name, …

[

    ['Name', 'Age', 'Job'],

    ['Alice', '30', 'Engineer'],

    ['Bob', '35', 'Doctor'],

    ['Moussa', '45', 'Prof'],

    ['Jo', '20', 'Farmer']

]

[

    {'Name': 'Alice', 'Age': '30', 'Job': 'Engineer'},

    {'Name': 'Bob', 'Age': '35', 'Job': 'Doctor'},

    {'Name': 'Moussa', 'Age': '45', 'Job': 'Prof'},

    {'Name': 'Jo', 'Age': '20', 'Job': 'Farmer'}

]

csv.DictReader(file)

csv.reader(file)

1. **Avantages d'une liste de dictionnaires**
2. **1. Lisibilité et clarté du code**

* Chaque ligne est un dictionnaire où les colonnes sont accessibles via des **clés descriptives** (les noms des colonnes).
* Cela rend le code plus intuitif. Par exemple :

python

Copier le code

ligne['nom'] # Accès clair à la colonne "nom"

* Avec un tableau 2D, on utiliserait des indices numériques :

python

Copier le code

ligne[0] # Moins explicite, nécessite de connaître l'ordre des colonnes

1. **2. Flexibilité pour gérer les colonnes**

* Les dictionnaires ne dépendent pas de l'ordre des colonnes. Si les colonnes changent d'ordre dans le fichier CSV, le code continue de fonctionner correctement.
* Avec un tableau 2D, si l'ordre des colonnes change, le code basé sur des indices peut devenir incorrect.

1. **3. Ajout ou suppression de colonnes**

* Une liste de dictionnaires permet d'ajouter ou de supprimer des colonnes facilement en modifiant les clés des dictionnaires.
* Dans un tableau 2D, cela nécessiterait de restructurer tout le tableau.

1. **4. Recherche et manipulation aisées**

* Avec des dictionnaires, il est facile de rechercher des lignes ou de manipuler des colonnes grâce aux clés :

python

Copier le code

[ligne for ligne in table if ligne['ville'] == 'Paris']

* Dans un tableau 2D, il faudrait utiliser des indices, rendant le code plus complexe et moins lisible.

1. **5. Compatibilité avec les formats tabulaires courants**

* Les formats de fichiers comme CSV ou JSON sont souvent bien adaptés aux dictionnaires. En Python, les modules standard comme csv.DictReader ou json utilisent des dictionnaires pour représenter des structures tabulaires.

Le choix entre une **liste de dictionnaires** et un **tableau 2D indexé par des indices** dépend des besoins en termes de lisibilité, de flexibilité, et des opérations à effectuer sur les données. Voici les raisons principales pour lesquelles une liste de dictionnaires est souvent préférée pour représenter des tables en Python :

1. **Avantages d'une liste de dictionnaires**
2. **1. Lisibilité et clarté du code**

* Chaque ligne est un dictionnaire où les colonnes sont accessibles via des **clés descriptives** (les noms des colonnes).
* Cela rend le code plus intuitif. Par exemple :

python

Copier le code

ligne['nom'] # Accès clair à la colonne "nom"

* Avec un tableau 2D, on utiliserait des indices numériques :

python

Copier le code

ligne[0] # Moins explicite, nécessite de connaître l'ordre des colonnes

1. **2. Flexibilité pour gérer les colonnes**

* Les dictionnaires ne dépendent pas de l'ordre des colonnes. Si les colonnes changent d'ordre dans le fichier CSV, le code continue de fonctionner correctement.
* Avec un tableau 2D, si l'ordre des colonnes change, le code basé sur des indices peut devenir incorrect.

1. **3. Ajout ou suppression de colonnes**

* Une liste de dictionnaires permet d'ajouter ou de supprimer des colonnes facilement en modifiant les clés des dictionnaires.
* Dans un tableau 2D, cela nécessiterait de restructurer tout le tableau.

1. **4. Recherche et manipulation aisées**

* Avec des dictionnaires, il est facile de rechercher des lignes ou de manipuler des colonnes grâce aux clés :

python

Copier le code

[ligne for ligne in table if ligne['ville'] == 'Paris']

* Dans un tableau 2D, il faudrait utiliser des indices, rendant le code plus complexe et moins lisible.

1. **5. Compatibilité avec les formats tabulaires courants**

* Les formats de fichiers comme CSV ou JSON sont souvent bien adaptés aux dictionnaires. En Python, les modules standard comme csv.DictReader ou json utilisent des dictionnaires pour représenter des structures tabulaires.

1. **Quand préférer un tableau 2D avec indices ?**
2. **1. Performance**

* Les tableaux 2D (listes imbriquées) peuvent être légèrement plus rapides à manipuler dans des cas où les indices sont suffisants pour représenter les colonnes.
* Si le fichier est très grand et les colonnes peu nombreuses, l'usage des indices peut réduire la consommation de mémoire.

1. **2. Applications spécifiques**

* Pour des calculs mathématiques ou des opérations matricielles (par exemple, en algèbre linéaire), les tableaux 2D sont plus adaptés et peuvent être directement manipulés avec des bibliothèques comme **NumPy**.

1. **3. Ordre strict des colonnes**

* Si l'ordre des colonnes est critique et ne doit pas changer, un tableau 2D peut être suffisant.

1. **Résumé comparatif**

| **Critère** | **Liste de dictionnaires** | **Tableau 2D (indices)** |
| --- | --- | --- |
| **Lisibilité** | Clés descriptives rendent le code clair | Indices numériques, moins intuitif |
| **Flexibilité** | Facile à gérer avec des colonnes dynamiques | Plus rigide |
| **Performance** | Suffisant pour des données de taille modérée | Optimisé pour des données massives |
| **Complexité du fichier** | Gère bien des colonnes nombreuses | Gère bien des colonnes fixes et simples |
| **Utilisation typique** | Manipulation de données structurées (CSV/JSON) | Calculs matriciels, algorithmes basés sur indices |