S1J4 : Comprendre les structures de données (listes, dictionnaires, sets).

Objectif :

Maîtriser les structures de données Python (listes, dictionnaires, tuples, ensembles) pour manipuler efficacement les données avant de les charger dans des pipelines ou bases.

---

1. Les Listes.

Les listes sont des collections ordonnées et modifiables. Très utilisées pour stocker des séries de données.

Création de listes.

```
ma_liste = [1, 2, 3, 4, 5]
```

Opérations de base sur les listes.

Ajouter ou supprimer des éléments :

```
ma_liste.append(6)  # Ajouter un élément
ma_liste.remove(3)  # Supprimer un élément
```

Accéder à des éléments par index :

```
print(ma_liste[0])  # Premier élément
print(ma_liste[-1])  # Dernier élément
```

Slicing (sous-listes) :

```
sous_liste = ma_liste[1:4]  # Obtenir les éléments des index 1 à 3
```

Itération sur une liste.

```
for item in ma_liste:
```

```
    print(item)
```

Compréhension de liste.

Une façon élégante de transformer les listes :

```
double = [x * 2 for x in ma_liste]
```

---

2. Les Dictionnaires.

Les dictionnaires permettent de stocker des données sous forme de paires clé-valeur.

Création d'un dictionnaire.

```
mon_dict = {"nom": "Alice", "âge": 30, "ville": "Paris"}
```

Accéder à une valeur par sa clé.

```
print(mon_dict["nom"])
```

Ajouter ou modifier des paires clé-valeur.

```
mon_dict["profession"] = "Data Engineer"
mon_dict["âge"] = 31
```

Supprimer des éléments.

```
del mon_dict["ville"]
```

Itérer sur un dictionnaire.

```
for key, value in mon_dict.items():
    print(f"{key}: {value}")
```

Applications courantes dans l'ingénierie des données.

Stocker des configurations de pipeline.

Mapper des clés pour enrichir des datasets.

---

3. Les Tuples.

Les tuples sont des collections ordonnées et immuables.

Création de tuples.

```
mon_tuple = (1, 2, 3)
```

Avantages des tuples.

Moins gourmand en mémoire.

Utilisés comme clés dans les dictionnaires si immuables.

Déballage de tuples (utile dans les pipelines) :

```
a, b, c = mon_tuple
print(a, b, c)
```

---

4. Les Ensembles.

Les ensembles sont des collections non ordonnées et sans doublons. Très utiles pour des opérations comme les intersections ou les unions.

Création d'un ensemble.

```
mon_set = {1, 2, 3, 4, 5}
```

Opérations de base

Ajouter ou supprimer un élément :

```
mon_set.add(6)
mon_set.remove(3)
```

Vérifier la présence d'un élément :

```
print(2 in mon_set)  # True
```

Intersection, union et différence :

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 & set2)  # Intersection : {3}
print(set1 | set2)  # Union : {1, 2, 3, 4, 5}
print(set1 - set2)  # Différence : {1, 2}
```

---

5. Cas pratiques dans l'ingénierie des données.

Listes : Stocker des enregistrements extraits d'un fichier.

Dictionnaires : Mapper des IDs à des noms ou stocker des métadonnées sur des colonnes d'un dataset.

Tuples : Transmettre des paires de données immuables (ex. : clés primaires).

Ensembles : Identifier des doublons ou des valeurs uniques dans un dataset.

Exercice pratique.

Créez un script qui utilise les structures de données suivantes :

Liste : Stockez une liste de ventes (par exemple `[100, 200, 150, 300, 150]`).

Calculez la somme totale des ventes.

Dictionnaire : Associez un vendeur à ses ventes.

```
ventes = {"Alice": 500, "Bob": 600, "Charlie": 700}
```

Ajoutez un nouveau vendeur et son total.

Trouvez le vendeur ayant le plus de ventes.

Ensembles : Identifiez les produits vendus (sans doublons) à partir d'une liste comme `["Produit A", "Produit B", "Produit A", "Produit C"]` .

Tuple : Stockez un tuple pour représenter une transaction `(vendeur, produit, montant)` et affichez les détails.

Pourquoi c'est important pour un Data Engineer ?

Optimisation des pipelines : Les structures comme les dictionnaires et les ensembles permettent d'accélérer des opérations clés, comme les recherches ou les dédoublonnages.

Manipulation à faible coût : Comprendre quand utiliser une liste ou un tuple peut réduire l'utilisation de mémoire.

Organisation des données : Ces structures aident à organiser efficacement les données brutes avant de les charger dans des bases ou des systèmes.