L2 MIASHS – Informatique S3
TD2 : Ensembles, dictionnaires, et tri

2024

Les exercices 1 à 4 de cette feuille doivent être maîtrisés. Le petit contrôle de vendredi prochain portera sur ces exercices ainsi que sur le cours, y compris celui de mercredi prochain.

1 Optimisation avec des dictionnaires imbriqués

Problème : Vous êtes en charge de comparer différents produits électroniques pour un site de vente en ligne. Chaque produit est évalué selon son prix et sa note qualité attribuée par les utilisateurs. Vous devez recommander le produit offrant le meilleur rapport qualité/prix, c'est-à-dire le prix le plus bas pour la meilleure note.
Écrivez une fonction `meilleur_produit` qui renvoie le produit avec le meilleur rapport qualité/prix (prix le plus bas pour la meilleure note).
Exemple d'utilisation :

```
produits = {
    "ordinateur": {"prix": 1000, "note": 4.5},
    "tablette": {"prix": 500, "note": 4.2},
    "smartphone": {"prix": 800, "note": 4.8}
}
print(meilleur_produit(produits))
# Résultat : "smartphone"
```

2 Fusion d'ensembles avec filtrage

Problème : Imaginez que vous gérez une base de données de clients inscrits à plusieurs événements. Vous souhaitez extraire les clients ayant participé à certains événements en filtrant ceux qui répondent à un critère précis. Par exemple, vous devez identifier les participants dont les numéros de ticket sont des nombres pairs.
Écrivez une fonction `filtrer_paires` qui renvoie un ensemble avec uniquement les nombres pairs de l'union des deux ensembles triés.
Exemple d'utilisation :

```
ensemble1 = {1, 2, 3, 8}
ensemble2 = {3, 4, 5, 6}
print(filtrer_paires(ensemble1, ensemble2))
```

```
# Résultat : {2, 4, 6, 8}
```

3 Algorithme de tri dans un dictionnaire

Problème : Vous devez organiser les résultats d'une évaluation pour plusieurs élèves, les noms étant stockés dans un dictionnaire en tant que clés, et leurs notes en tant que valeurs. Vous devez classer ces élèves selon leurs notes, de la plus basse à la plus haute.
Écrivez une fonction `trier_notes` qui renvoie une liste des noms triés selon leurs notes.
Exemple d'utilisation :

```
notes = {"Alice": 15, "Bob": 12, "Charlie": 9}
print(trier_notes(notes))
# Résultat : ["Charlie", "Bob", "Alice"]
```

Réflexion : En Python, il est important de comprendre qu'un dictionnaire n'est pas une structure ordonnée. Cela signifie que vous ne pouvez pas directement trier un dictionnaire. En revanche, vous pouvez trier ses éléments (clés, valeurs, ou les deux) en les stockant dans une nouvelle structure, comme une liste de tuples. Réfléchissez à pourquoi Python impose cette restriction et comment cela impacte l'efficacité du traitement des données.
Quelques informations sur le tri : Le tri des données est une opération essentielle dans de nombreux contextes informatiques. En Python, il est courant d'utiliser les fonctions `sort()` et `sorted()` pour trier les éléments. Cherchez la différence entre les deux. Il est aussi important de savoir ré-écrire un algorithme de tri simple (exercice suivant).
Exemple d'utilisation de la fonction `sorted()` :

```
d = {"Alice": 15, "Bob": 12, "Charlie": 9}
sorted_d = sorted(d.items(), key=lambda item: item[1])
# Résultat : [('Charlie', 9), ('Bob', 12), ('Alice', 15)]
```

4 Tri par permutation

L'algorithme :

- **Première boucle (externe) :** Parcourez chaque élément `x` de la liste, de gauche à droite.
    1. **Deuxième boucle (interne) :** Pour chaque élément `x` , parcourez les éléments `y` qui le suivent dans la liste,
    2. Comparez l'élément `x` avec l'élément `y`.
    3. Si `x` est plus grand que `y`, échangez-les.
- Répétez ce processus pour tous les éléments de la liste.
- À la fin la liste est triée

Faites fonctionner cet algorithme sur papier sur un petit exemple, pour comprendre comment il trie.

Cet algorithme est simple à implémenter, mais il est inefficace pour les grandes listes car il nécessite un grand nombre de comparaisons.

Écrivez une fonction `tri_par_permutation` qui implémente cet algorithme et trie une liste de nombres.
Exemple d'utilisation :

```
nombres = [3, 1, 4, 1, 5, 9, 2, 6, 5]
print(tri_par_permutation(nombres))
# Résultat : [1, 1, 2, 3, 4, 5, 5, 6, 9]
```