1 Pourquoi trier ?

Réponse simple : **pour faire des recherches plus facilement !**

Organiser les données, en particulier en les triant, est une opération qui nécessite obligatoirement du temps. Mais ce qui pourrait sembler une perte de temps est en réalité un gain de temps énorme pour toutes les opérations de recherche qui pourront avoir lieu ensuite.

À retenir : **on trie une fois, on cherche de multiples fois.**

Il existe de nombreux algorithmes de tri, nous en étudierons 2 en détail cette année : le **tri par insertion** et le **tri par sélection** : voir chap. P7-II.

2 Le tri en Python

La fonction `sorted` de Python transforme une collection de données (ex : p-uplet, liste) en une **liste** de données triées. Cette fonction admet 2 paramètres optionnels :
- reverse : bool : True pour un tri décroissant.
- key : function : fonction qui définit une clef de tri.

Pour trier, il suffit de définir un opérateur de comparaison entre 2 données de même type.

Exemple :
- Les nombres peuvent se comparer par ordre croissant.
- les chaînes de caractères peuvent se comparer par ordre alphabétique.
- mais on peut imaginer des critères plus complexes :
  2 cartes à jouer pourraient se comparer par hauteur puis par couleur par exemple.

3 Exemples

On donne ici quelques exemples pour se remémorer la syntaxe Python pour le tri.

3.1 Tri d'une liste simple

1. Tri d'entier par ordre décroissant :

```
>>> liste = [8, 10, 1, 3]
>>> sorted(liste, reverse=True)
[10, 8, 3, 1]
```

2. Tri de chaînes de caractères par ordre croissant :

```
>>> liste = ["trèfle", "carreau", "coeur", "pique"]
>>> sorted(liste)
['carreau', 'coeur', 'pique', 'trèfle']
```

3.2 Tri d'une liste de tuples

1. Tri par défaut sur le 1er élément des tuples :

```
>>> liste = [(8, 'trèfle'), (10, 'carreau'), (1, 'coeur'), (3, 'pique')]
>>> sorted(liste)
[(1, 'coeur'), (3, 'pique'), (8, 'trèfle'), (10, 'carreau')]
```

2. Tri sur le 2ème élément des tuples :

```
>>> def clef(t):
        """ Fonction de clef de tri.
        Ici on indique qu'on désire trier sur le 2ème élément d'un tuple
        """
        return t[1]

>>> sorted(liste, key=clef)
[(10, 'carreau'), (1, 'coeur'), (3, 'pique'), (8, 'trèfle')]
```

3. Plutôt que de définir une fonction annexe pour la clef de tri, on peut utiliser une **fonction lambda** (ou **fonction anonyme**) directement comme argument du paramètre key.

```
>>> sorted(liste, key=lambda t:t[1])
[(10, 'carreau'), (1, 'coeur'), (3, 'pique'), (8, 'trèfle')]
```

3.3 Tri d'une liste de dictionnaires

1. Tri suivant une des clefs du dictionnaire :

```
>>>liste = [{'nom': '8 de trèfle', 'hauteur': 8, 'couleur': 'trèfle'},
            {'nom': '10 de carreau', 'hauteur': 10, 'couleur': 'carreau'},
            {'nom': 'As de coeur', 'hauteur': 1, 'couleur': 'coeur'},
            {'nom': '3 de pique', 'hauteur': 3, 'couleur': 'pique'}]

>>> sorted(liste, key=lambda d:d['hauteur'])
[{'nom': 'As de coeur', 'hauteur': 1, 'couleur': 'coeur'},
 {'nom': '3 de pique', 'hauteur': 3, 'couleur': 'pique'},
 {'nom': '8 de trèfle', 'hauteur': 8, 'couleur': 'trèfle'},
 {'nom': '10 de carreau', 'hauteur': 10, 'couleur': 'carreau'}]
```

2. Tri par longueur du nom décroissante :

```
>>> sorted(liste, key=lambda d:len(d['nom']), reverse=True)
[{'nom': '10 de carreau', 'hauteur': 10, 'couleur': 'carreau'},
 {'nom': '8 de trèfle', 'hauteur': 8, 'couleur': 'trèfle'},
 {'nom': 'As de coeur', 'hauteur': 1, 'couleur': 'coeur'},
 {'nom': '3 de pique', 'hauteur': 3, 'couleur': 'pique'}]
```