Lorsqu'on manipule des données tabulées, il est souvent utile de pouvoir les organiser selon différents critères. Par exemple, on peut trier une liste d'élève par ordre alphabétique, ou par note croissante, etc. Nous allons voir comment faire cela.

1 Les fonctions `sort` et `sorted`

Python dispose de deux fonctions natives permettant de trier des tableaux : `sort` et `sorted`.

La différence entre les deux consiste dans la manière de les appeler et dans l'effet produit. Considérons un tableau `t` (`list`) d'entiers (`int`).

— La commande `t2 = sorted(t)` crée une copie triée de `t` et la stocke dans le tableau `t2`.

— La commande `t.sort()` trie le tableau `t` directement, on parle de **tri sur place**.

Par défaut, le tri sera effectué **par ordre croissant**.

Ces fonction acceptent de trier des tableaux :

— d'entiers (`int`) ;

— de flottants (`float`) ;

— de chaînes de caractères (`str`) et dans ce cas c'est l'ordre alphabétique qui est utilisé ;

— des tableaux et des n-uplets ne contenant que de telles valeurs (on commence par trier selon la première coordonnée, puis la deuxième, etc.)

Exercice : Tester les fonctions `sort` et `sorted` sur différents types de tableaux (`int`, `float`, `str`, `list`, `tuple`).

Exercice : Vérifier que ces fonctions renvoient une erreur sur des tableaux de dictionnaires (`dict`).

2 Ordre lexicographique et stabilité

Comme on l'a vu plus haut, les fonction de tri de Python autorisent le tri de tableaux ou de $n-uplet$ à condition que chaque coordonnée soient nativement comparables (nombres ou chaîne de caractère).

Pour ce faire, elles comparent les valeurs selon la procédure suivante :

— Elles comparent toujours les valeurs de la première coordonnée de chaque case `t[i][0]` ;

— En cas d'égalité de ces valeurs dans, elles regargent comparent les valeurs de la deuxième coordonnée `t[i][1]` ;

— etc.

En ce basant sur cette façon de comparer chaque données entre elles et à l'aide d'un algorithme de tri quelconque, on trie les éléments par **ordre lexicographique**.

Exercice : Trier les élèves de la classe par date de naissance (ordre lexicographique année, mois, jour).
Une autre manière d'obtenir un ordre lexicographique est d'utiliser plusieurs occurrences d'un tri dit stable.

Un algorithme de tri est dit **stable** si deux valeurs possédant la même clé de tri sont toujours positionnée dans le tableau dans le même ordre relatif avant et après le tri.

Un exemple : On considère la liste suivante :

1, 8, 8, 2, 2, 4, 4, 1

En respectant les couleurs, et puisque chaque nombre rouge est à l'origine positionné avant son homologue bleu, un tri stable devra donner la liste suivante :

1, 1, 2, 2, 4, 4, 8, 8,

Ainsi, pour obtenir un ordre lexicographique, on peut imaginer l'algorithme suivant :

— Pour chaque coordonnées des tuples du tableau en partant de la dernière :
  — trier le tableau selon cette coordonnée **à l'aide un tri stable**.

Exercice : trier les élèves de la classe par date de naissance.

3 Tri d'une table, clé de tri

Considérons une table Python, c'est à dire pour nous un tableau (`list`) de dictionnaires (`dict`).

On rappelle que :

— `table[0]`, `table[1]`, ..., `table[n-1]` nous permet d'accéder aux lignes de la table ;

— `table[i][att]` nous permet d'accéder à l'attribut de nom `att` (`str`) de la ligne `i`.

Par exemple, considérons le tableau `eleves` suivant :

nom	annee	mois	jour
Toto	2004	10	22
Titi	2004	11	3
Tutu	2003	7	15
Tata	2005	1	12

Exercice :

— Écrire la table (tableau de dictionnaires) correspondante.

— Que renvoie `eleve[2]` ?

— Que renvoie `eleve[3]["annee"]` ?

— Comment accéder au jour de naissance de Toto (on sait qu'il est à la première ligne) ?

Comme on l'a vu, les fonction `sort` et `sorted` ne fonctionne pas directement sur un tableau tel que `eleves`. Pour qu'elles fonctionnent, il faut leur préciser un argument d'habitude optionnel nommé `key`. Cet argument permet de préciser la **clé de tri**, c'est à dire le critère que l'on veut utiliser pour effectuer le tri.

Cette clé est une **fonction** prenant en argument une donnée du même type que les élément du tableau et renvoyant une donnée que Python est capable de trier par défaut (`int, float, str`).

Par exemple, si on veut trier notre table d'élèves par ordre lexicographique selon le nom des élèves, il faut le préciser à `sorted` ou `sort` :

```
def nom(ligne):
    return ligne["prenom"]

eleves.sort(key = nom)
#ou
eleves_tries = sorted(eleves, key = nom)
```

Exercice : tester ce code et constater que la table `eleves` est bien triée par ordre alphabétique.

Le fait de devoir définir une fonction à chaque fois qu'on veut effectuer un tri n'est pas très pratique. À la place, on peut utiliser une syntaxe permettant de définir une fonction de manière simplifiée, en une seule ligne.

Exercice : Tester les commandes suivantes et analyser leur fonctionnement :

— `f = lambda x : x**2`

— `print(f(2), f(3), f(4))`

— `(lambda x : x-1)(7)`

— `(lambda x : 2*x+1)(8)`

L'écriture (`lambda x : formule dépendant de x`) permet de définir rapidement une fonction sans passer par la syntaxe **def** habituelle.

Cette écriture nous permet de condenser nos appels aux fonctions de tri :

```
eleves.sort(key = lambda ligne : ligne["nom"])
#ou
eleves_tries = sorted(eleves, key = lambda ligne : ligne["nom"])
```