

1 Un prédicat

Réalisez un prédicat qui

- prend en paramètre une liste et une fonction de comparaison
- et renvoie `True` si cette liste est triée par l'ordre défini par la fonction de comparaison, et renvoie `False` dans le cas contraire.

```
>>> est_trie([3,1,4,1,5,9,2], compare)
False
>>> est_trie([1,1,2,3,4,5,9], compare)
True
```

2 Ordre sur les dates

On suppose dans cet exercice que les dates sont représentées par des objets d'une classe `Date` possédant trois attributs nommés 'jour', 'mois' et 'annee'. On suppose de plus que les valeurs associées à ces attributs sont des entiers.

On donne la variable `etudiants` contenant des objets de type `Etudiant` possédant quatre attributs : `nip` (un entier), `nom` (une chaîne), `premier` (une chaîne), `date_de_naissance` (de classe `Date`) :

```
etudiants = [Etudiant(99998132, "Calbuth", "Raymond", Date(12, 12, 1987)),
             Etudiant(99994451, "Talon", "Achille", Date(7, 11, 1963)),
             Etudiant(99996348, "Calbuth", "Monique", Date(29, 7, 1987)),
             Etudiant(99995433, "Blanc-Sec", "Adèle", Date(17, 4, 1976)),
             Etudiant(99997674, "Brisefer", "Benoît", Date(15, 12, 1960)),
             Etudiant(99998324, "Lagaffe", "Gaston", Date(28, 2, 1957))]
```

2. Donnez une expression utilisant la fonction `sorted` égale à une nouvelle liste contenant les mêmes données que la liste `etudiants` mais rangées par ordre décroissant de `nip`.
3. Donnez une instruction utilisant la méthode `sort` permettant de modifier la variable `etudiants` afin que la liste des étudiants soit rangée par ordre croissant des `nip`.
4. Donnez une instruction permettant de modifier la variable `etudiants` afin que la liste des étudiants soit rangée par ordre décroissant des `nip`.
5. Donnez une instruction permettant de modifier la variable `etudiants` afin que la liste des étudiants soit rangée par ordre croissant de `nom`.
6. Proposez une fonction de comparaison d'étudiants avec laquelle vous pourrez modifier la variable `etudiants` afin que la liste des étudiants soit rangée par ordre croissant de date de naissance.

3 Ordre sur les chaînes

Dans cet exercice, on souhaite définir une variante de l'ordre des caractères puis étendre cette variante aux chaînes de caractères. On dit que c est une lettre si c est

- soit entre 'A' et 'Z'
- soit entre 'a' et 'z'

1. Réalisez une fonction `compare_car` paramétrée par deux caractères telle que `compare_car(c1, c2)`
 - renvoie soit -1, 0 ou 1
 - avec pour les lettres l'ordre défini par 'aAbBcC...yYzZ'
 - tout caractère non lettre étant plus grand que les lettres, et dans leur ordre habituel.

```
>>> compare_car('a', 'A')
-1
>>> compare_car('A', 'b')
-1
>>> compare_car('@', 'Z')
1
>>> compare_car('9', '@')
-1
```

```
>>> compare_car('@', '@')
```

```
0
```

2. Réalisez une fonction `compare_chaine` permettant de comparer les chaînes en utilisant la fonction `compare_car` précédente, de sorte que

- les chaînes soient rangées par longueur croissante ;
- les chaînes de même longueur soient rangées par ordre lexicographique induit par `compare_car`.

```
>>> compare_chaine('', 'Tim')
```

```
-1
```

```
>>> compare_chaine('Timoleon', 'TiM')
```

```
1
```

```
>>> compare_chaine('Timoleon', 'TimoLeon')
```

```
-1
```

```
>>> compare_chaine('TimoLeon', 'TimoLeon')
```

```
0
```

4 Retour sur les doublons

1. Proposez un algorithme utilisant les tris pour construire la liste des doublons d'une liste donnée.
2. Comparez le nombre de comparaisons d'éléments de la liste effectuées par cet algorithme avec celui de l'algorithme que vous avez proposé dans l'exercice de la feuille précédente.

5 Variante du tri par sélection

Nous avons présenté le tri par sélection du plus petit élément de la tranche restant à trier. Il est possible aussi de faire un tri par sélection du plus grand élément.

1. Donnez l'algorithme de tri par sélection du plus grand élément.
2. Implantez cet algorithme pour réaliser une procédure `tri_sel_max` qui trie de cette manière la liste passée en paramètre.

6 Tri à bulle

L'algorithme ci-dessous est un algorithme de tri dénommé *tri à bulles* qui est une certaine forme de tri par sélection du minimum.

- **entree** t une liste de longueur n .
 - **effet de bord** t une liste triée de longueur n contenant les mêmes éléments.
 - **Pour** i variant de 0 à $n - 2$
 - mettre le plus petit élément de la tranche $t(i..n - 1)$ en position i
 - **Pour** j variant de $n - 1$ à $i + 1$ en décroissant
 - * **Si** $t(j) < t(j - 1)$
 - échanger $t(j)$ et $t(j - 1)$
 - * **Fin si**
 - **Fin Pour**
 - La tranche $t(0..i)$ est triée et ses éléments sont inférieurs ou égaux à tous les autres éléments de t .
 - **Fin Pour**
 - la tranche $t(0..n - 1)$ est triée.
1. Donnez les états successifs de la liste à la fin de chaque itération de la boucle pour la plus interne lorsque $i = 0$ et la liste à trier est

$t =$

0	1	2	3	4	5	6	7
T	I	M	O	L	E	O	N

2. Même question pour la fin de chaque itération de la boucle la plus externe.
3. Combien de comparaisons d'éléments de la liste sont-elles effectuées lors du tri d'une liste de longueur n ?
4. Si lors d'une étape de la boucle pour principale, aucun échange n'est effectué dans la boucle pour interne, c'est que la liste est triée. Il est donc inutile de poursuivre la boucle externe.
Décrivez un algorithme qui tient compte de cette remarque.
5. Combien de comparaisons d'éléments de la liste sont-elles effectuées lors du tri d'une liste de longueur n avec cette variante du tri à bulles ? Décrivez le meilleur et le pire des cas.
6. Réalisez une procédure qui trie la liste passée en paramètre selon cet algorithme.

7 Tri insertion

1. Donnez les états intermédiaires successifs de la liste $[4, 2, 5, 3, 6, 1]$ lors de son tri par insertion. Comptez le nombre de comparaisons d'éléments de la liste effectuées pour ce tri.
2. Donnez un encadrement du nombre de comparaisons d'éléments d'une liste de longueur 6 lors du tri par insertion de cette liste.

8 Une variante du tri par insertion

L'algorithme de tri par insertion vu en cours procède par insertions successives des éléments de la liste dans la tranche de gauche de la liste.

Proposez un algorithme de tri par insertion qui procède par insertions successives dans la tranche de droite.

9 Trier une pile de crêpes



Nous considérons dans cet exercice un problème simple, dont l'intérêt scientifique ne saute sans doute pas aux yeux. Dans sa version la plus imagée, le problème consiste en effet à mettre dans l'ordre une pile de crêpes de différentes tailles. La crêpe de plus grand diamètre doit se retrouver en dessous de pile ; la plus petite au

sommet ; entre les deux, toute crêpe doit reposer sur une crêpe plus large. Pour modifier l'arrangement de la pile de crêpes, vous êtes armé d'une spatule appropriée et contraint à un seul type d'action : insérer la spatule entre deux crêpes et retourner l'ensemble des crêpes du dessus¹.

On représentera les piles de crêpes par des listes des diamètres de ces crêpes (en mm par exemple), le diamètre de la crêpe au sommet de la pile étant le premier élément de la liste.

Il nous faut maintenant une procédure qui simule l'action du retournement d'une partie de la pile de crêpes à l'aide de la spatule. C'est l'objet de la question qui suit.

1. Réalisez une procédure `renverser_sommet` à deux paramètres : une liste ℓ et un entier k compris entre 0 et n , n étant la longueur de la liste, qui renverse l'ordre des k premiers éléments de ℓ . Cette procédure modifie la liste passée en paramètre et ne renvoie rien comme le montre la session qui suit.

```
>>> l = [3,1,4,5,6,2]
>>> renverser_sommet (l,5)
>>> l
[6, 5, 4, 1, 3, 2]
>>> renverser_sommet (l,6)
>>> l
[2, 3, 1, 4, 5, 6]
```

Armé de cette procédure vous allez pouvoir trier des piles de crêpes.

L'idée de cet algorithme consiste à

- repérer où se trouve la plus grande des crêpes ;
- insérer la spatule sous cette crêpe et renverser le sommet de la pile ;
- renverser la totalité de la pile de crêpe.

Ayant accompli ces trois points, la plus grande des crêpes se trouve tout en bas de la pile, et il ne reste plus qu'à trier les crêpes situées au dessus.

2. En considérant la pile de crêpes représentée par la liste $l = [5, 1, 4, 2, 6, 3]$, donnez la série des renversements à effectuer en précisant à côté de chacun d'eux l'état de la pile obtenue.
3. Réalisez une procédure de tri d'une pile de crêpes.

```
>>> l = [35, 12, 44, 50, 61, 25]
>>> tri_pile_crepes (l)
>>> l
[12, 25, 35, 44, 50, 61]
```

10 Tris non en place

La spécification des fonctions de tri vue en cours les définit comme des procédures qui modifient leur paramètre, tout comme la méthode `sort` en python.

Reprogrammez les fonctions de tri par sélection et de tri par insertion, de sorte qu'elles ne modifient pas leur paramètre, mais renvoient une nouvelle liste triée, tout comme la fonction `sorted` en python.

¹D'après un article publié à l'adresse https://interstices.info/jcms/n_52318/genese-dun-algorithme dans lequel on apprend que l'algorithme de cet exercice est utilisé pour résoudre des problèmes de routage dans les réseaux de processeurs qui calculent en parallèle.