TP7 - Boucles imbriquées

→ *Q.1) Créez un fichier* `TP7.py` *dans lequel vous pourrez sauvegarder votre travail.*

A) Recherche dans un mot dans un texte

A.1 Lecture de fichier texte

Pour manipuler les données stockées dans un fichier texte, on dispose de plusieurs fonctions sous python. Le principe est le suivant :

- On ouvre le fichier : on crée un objet de type **file** grâce à la fonction **open**. On peut voir cet objet comme un curseur qui pointe au début du fichier texte ouvert et qui peut lire les lignes du fichier et les renvoyer sous forme de chaînes de caractères.
- On lit les lignes du fichiers successivement : on appelle la méthode readline au fichier ouvert. Cette méthode lit la ligne, la renvoie sous forme de **str**, puis passe à la ligne suivante. On peut aussi utiliser readlines qui renvoie un itérateur contenant toutes les lignes du fichier, ou bien encore la méthode read qui lit tout le fichier d'un coup.
- On ferme le fichier avec la méthode close.

Dans l'exemple suivant, on ouvre un fichier qui s'appelle test.txt puis on calcule le nombre de mots dans le fichier (un mot étant une chaîne de caractère entourée de deux espaces) :

```
mon_fichier = open('test.txt', 'r')
nombre = 0
for ligne in mon_fichier.readlines():
    mots = ligne.split()
    nombre += len(mots)
mon_fichier.close()
```

La méthode split casse une chaîne de caractère au niveau des espaces et des retours à la ligne et renvoie une liste de chaînes de caractères formée des morceaux obtenus.
La fonction **open** prend en paramètre le nom du fichier et une option pour préciser si le fichier est ouvert en lecture seule ('r' comme read) ou bien en écriture ('w' comme write). Dans l'exemple ci-dessous, on ouvre un fichier (qui est créé s'il n'existe pas encore) et on écrit dans le fichier (en effaçant le contenu précédent le cas échéant) :

```
mon_nouveau_fichier = open('test2.txt', 'w')
mon_nouveau_fichier.write('Bonjour,\nComment allez-vous ?')
mon_nouveau_fichier.close()
```

La méthode write() écrit dans le fichier. Le caractère \n est un retour à la ligne.

→ *Q.2) Créez avec python un fichier* `TP7.txt` *et écrire dans ce fichier trois lignes contenant : votre prénom, votre nom et votre date de naissance.*

→ *Q.3) Ouvrez ce fichier avec un éditeur de texte pour vérifier le résultat.*

A.2 Recherche

On dispose d'un texte dans le fichier roman.txt et on souhaite trouver si un mot (ou une phrase) se trouve dans ce texte.
Autrement dit, on souhaite écrire une fonction recherche_motif(motif, texte) qui détermine la position de la première occurence du motif dans le texte, si elle existe et renvoie None sinon.
Le principe de cet algorithme est le suivant :

- on considère tous les emplacements possibles pour le motif : un indice i varie de 0 à **len**(texte) – **len**(motif) ;
- pour chaque position i, on teste si le motif apparaît avec une seconde boucle qui compare les caractères de motif et de texte un par un : on utilise ici une boucle **while** dans laquelle on utilise une variable j. La boucle s'arrête si j est trop grand ou si deux caractères ne coïncident pas ;
- en sortant de la boucle **while**, l'indice j est égal au nombre de caractères du motifs retrouvés dans le texte : on peut donc tester si on a trouvé le motif ou non.

Voici un exemple simple qui présente la valeur de j après chaque exécution de la boucle **while** : on cherche le motif 'erch' dans 'Recherche' :

i	j	R	e	c	h	e	r	c	h	e
0	0	e	r	c	h					
1	1		e	r	c	h				
2	0			e	r	c	h			
3	0				e	r	c	h		
4	4					e	r	c	h	

→ *Q.4) Écrire une fonction* `recherche_motif(motif, texte)` *qui implémente l'algorithme précédent.*

→ *Q.5) Ouvrir le fichier* `roman.txt` *et chercher si le motif* `"l'ami\nConseil n'a rien à dire"` *s'y trouve.*

→ *Q.6) Quel est le pire des cas pour l'exécution de cet algorithme ?*

→ *Q.7) Quelle est la complexité dans ce cas ?*

B) Tri à bulles

B.1 L'algorithme

On souhaite trier une liste de taille n par ordre croissant. Pour cela,

— on parcourt la liste et on compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, on les échange.

— On répète n fois ce parcours.

→ *Q.8) Écrire une fonction* `echange(L, i)` *qui échange les éléments* `L[i]` *et* `L[i+1]`.

→ *Q.9) Écrire une fonction* `parcours(L)` *qui parcourt une fois* `L` *et fait les échanges nécessaires.*

→ *Q.10) Tester la fonction* `parcours` *sur la liste* `L = [4, 1, 3, 2, 5]`.

→ *Q.11) Montrer qu'après la première exécution de* `parcours`*, l'élément le maximum de* `L` *est en dernière position.*

→ *Q.12) On suppose que les* `p` *derniers éléments de la liste* `L` *sont au bon endroit, c'est-à-dire qu'ils sont triés et plus grand que les* `len(L)-p` *premiers. Que peut-on en déduire si on exécute une fois la fonction* `parcours` *sur* `L` *?*

→ *Q.13) Que peut-on en conclure si on exécute* `len(L)` *fois la fonction* `parcours` *sur* `L` *?*

→ *Q.14) Écrire une fonction* `tri_a_bulles(L)` *qui trie la liste* `L` *en utilisant le tri à bulle.*

B.2 Complexité

→ *Q.15) En utilisant la fonction* `randint` *du module* `random`*, créer une liste* `L` *de 1000 entiers pris au hasard entre* 0 *et* 99.

→ *Q.16) En utilisant la fonction* `process_time` *du module* `time`*, évaluer le temps pris par la fonction* `tri_a_bulles` *pour trier la liste* `L`.

Il y a déjà un tri implémenté sous python pour les liste : l'instruction L.sort() trie la liste.

→ *Q.17) Évaluer le temps d'exécution de cette méthode.*

→ *Q.18) Évaluer la complexité de la fonction* `tri_a_bulles`.

C) Bilan

▷ Lecture d'un fichier texte :
- ouverture : mon_fichier = **open**('nom_du_fichier.txt', 'r').
- lecture d'une ligne : mon_fichier.readline().
- lecture de toutes les lignes (avec itérateur) : mon_fichier.readlines().
- fermeture : mon_fichier.close().
- écriture : mon_fichier.write('...') (si ouvert en mode écriture).

▷ Algorithmes classiques :
- recherche d'un motif (taille m) dans un texte (taille t) : $O(mt)$ dans le pire des cas.
- tri à bulles d'une liste de taille n : $O(n^2)$ dans le pire des cas.