L'objectif du TP est de présenter quelques exemples d'utilisation de boucles imbriquées.
On parle de boucles imbriquées lorsqu'une boucle (for ou while) se trouve dans le corps d'une autre boucle.
À chaque itération de la boucle extérieure, la boucle intérieure est alors exécutée intégralement. De plus, la boucle intérieure peut dépendre de la variable qui parcourt la boucle extérieure, comme dans l'exemple suivant.

```
for i in range(1,4):
    for j in range(i+1,5):
        print([i,j])
```

L'algorithme précédent affiche `[1, 2]`, `[1, 3]`, `[1, 4]`, `[2, 3]`, `[2, 4]`, `[3, 4]`.

Extrait du programme officiel : Algorithmes opérant sur une structure séquentielle par boucles imbriquées.
Recherche d'un facteur dans un texte. Recherche des deux valeurs les plus proches dans un tableau. Tri à bulles. *Notion de complexité quadratique. On propose des outils pour valider la correction de l'algorithme.*

Partie I : Somme triangulaire

Q.1 Écrire un script qui calcule $\sum_{1\leqslant i\leqslant j\leqslant 100} \frac{i}{j}$ c'est-à-dire qui calcule $\sum_{j=1}^{100}\left(\sum_{i=1}^{j}\frac{i}{j}\right)$.

Q.2 On peut montrer par le calcul que $\sum_{1\leqslant i\leqslant j\leqslant n} \frac{i}{j} = \frac{n(n+3)}{4}$ pour tout $n\in\mathbb{N}^*$. Tester si votre programme renvoie la bonne valeur. *Attention, des erreurs d'arrondie peuvent apparaître avec la manipulation des flottants.*

Partie II : Recherche des deux valeurs les plus proches dans un tableau

Q.3 Écrire une fonction `plus_petite_distance` prenant en argument une liste de nombres `L`, qu'on supposera de longueur au moins 2, et renvoyant la plus petite distance entre deux valeurs de la liste.
On pourra s'appuyer sur l'algorithme suivant.

- Introduire deux variables `n` et `dref` où `n` est la longueur de la liste `L` et `dref` est la distance de référence initialisée à `abs(L[0]-L[1])`.
- Parcourir les couples d'indices (i,j) tels que $0\leq i<j<n$ afin de comparer `abs(L[i]-L[j])` et `dref`. Si `abs(L[i]-L[j])<dref`, alors on actualise `dref` en lui affectant `abs(L[i]-L[j])`.
- À la fin du parcours, on renvoie `dref`.

Ainsi, `plus_petite_distance([-2,1,8,0])` doit renvoyer `1`.
Dans la fonction précédente, la fonction `abs` *est appelée* $\frac{n(n-1)}{2}$ *fois (où n est la taille de la liste). Le terme dominant de* $\frac{n(n-1)}{2}$ *est en n^2. On dit que la fonction est de coût quadratique.*

Q.4 *Pour les plus rapides.* En modifiant le code de la fonction `plus_petite_distance`, écrire une fonction `valeurs_plus_proches` d'argument toujours `L` et renvoyant, sous forme de tableau, les paires dont les valeurs sont les plus proches.
Ainsi, `valeurs_plus_proches([-2,1,6,8,3,-9,-7])` doit renvoyer `[[1,3],[6,8],[-9,-7]]`.
Alors que `valeurs_plus_proches([-2,1,6,8,3,-9,-7,4])` doit renvoyer `[[3,4]]`.

Partie III : Recherche d'un facteur dans un texte

On souhaite déterminer à quelles positions le facteur `f` est présent dans la chaîne de caractères `ch`.
On supposera, pour simplifier, que les chaînes de caractères `ch` et `f` ne sont pas vides, et que la longueur de `f` est inférieure (ou égale) à celle de `ch`.

Q.5 Écrire une fonction `positions_facteurs` prenant en argument les chaînes de caractères `ch` et `f`, et renvoyant les positions du facteur `f` dans `ch`.
On pourra introduire `n` *la longueur de* `f` *et tester si* `ch[0:n]==f`*, puis si,* `ch[1:n+1]==f` ...
Ainsi, `positions_facteurs('blablabla','bla')` doit renvoyer `[0,3,6]`.

Q.6 Dans cette question, on s'interdit d'utiliser la comparaison de sous-chaînes de caractères, écrire une fonction `positions_facteurs_prim` qui réalise la même chose que `positions_facteurs` et qui n'utilise que des comparaisons caractère à caractère, c'est à dire des instructions comme `ch[0]==f[0]`.

Partie IV : Tri à bulles

Nous verrons plus tard dans l'année les différents algorithmes de tri d'une liste dans un TP spécifique. Dans cette partie, nous nous intéressons uniquement au **tri à bulles**.
Le tri à bulles est un algorithme de tri dont le principe est de faire remonter à chaque étape le plus grand élément du tableau à trier, comme les bulles d'air remontent à la surface de l'eau (d'où le nom de l'algorithme).

Commençons par un exemple du fonctionnement de l'algorithme. On souhaite trier la liste de nombres : $[2, 5, 4, 3, 1]$.

- Premier passage :
  $[2, 5, 4, 3, 1]$, on compare 2 et 5 et on ne fait rien car $5 > 2$;
  $[2, 5, 4, 3, 1]$, on compare 5 et 4 et on les échange car $5 < 4$;
  $[2, 4, 5, 3, 1]$, on compare 5 et 3 et on les échange ;
  $[2, 4, 3, 5, 1]$, on compare 5 et 1 et on les échange ;
  $[2, 4, 3, 1, 5]$, fin du premier passage. Le dernier élément est à la bonne place.
- Deuxième passage :
  $[2, 4, 3, 1, 5]$, on compare 2 et 4 et on ne fait rien ; on compare 4 et 3 et on les échange ;
  $[2, 3, 4, 1, 5]$, on compare 4 et 1 et on les échange ;
  $[2, 3, 1, 4, 5]$, fin du deuxième passage. L'avant-dernier élément est à la bonne place.
- Troisième passage :
  $[2, 3, 1, 4, 5]$, on compare 2 et 3 et on ne fait rien ; on compare 3 et 1 et on les échange ;
  $[2, 1, 3, 4, 5]$, fin du troisième passage.
- Quatrième passage :
  $[2, 1, 3, 4, 5]$, on compare 2 et 1 et on les échange ;
  $[1, 2, 3, 4, 5]$, fin du quatrième passage. La liste est triée.

Q.7 Appliquer « à la main » l'algorithme sur la liste $[5, 1, 2, 3, 6, 4]$ afin de vous familiariser avec ce tri.

Q.8 Compléter la fonction `premier_passage` ci-dessous. Celle-ci doit prendre en argument une liste `L` et renvoyer une copie de cette liste ayant subi un passage de l'algorithme du tri à bulles.

```
def premier_passage(L):
    M=L.copy()
    for i in range(...):
        if ... :
            M[i] , M[i+1] = ...
    return M
```

Ainsi, `premier_passage([-2,8,1,0])` doit renvoyer `[-2,1,0,8]`.

Q.9 Écrire une fonction `tri_bulles` implémentant l'algorithme de tri à bulles. *On pourra partir de la fonction précédente et la modifier. On testera ensuite cette fonction avec une liste aléatoire de nombres entiers*[1].

Bonus pour les plus rapides.

Q.10 Déterminer, en fonction de la longueur de la liste en argument, le nombre de comparaisons effectuées par cet algorithme.

Q.11 À partir de l'exemple de la question 7, proposer une optimisation de l'algorithme permettant de gagner du temps dans les meilleurs cas. Implémenter cette optimisation par une fonction `tri_bulles2`.

Q.12 Comparer les temps d'exécution des fonctions `tri_bulles` et `tri_bulles2` sur une liste de 1000 éléments. *Pour calculer le temps d'exécution d'une fonction, on peut utiliser la fonction* `time` *de la bibliothèque* `time`. *Le plus facile étant de créer deux variables* `start=time.time()` *et* `end=time.time()` *entre lesquelles on exécute l'algorithme étudiée, puis d'afficher* `start-end`.

1. Pour générer une liste `L` de `n` nombres entiers aléatoires compris dans l'intervalle d'entiers $[\![a, b]\!]$, on peut écrire `L =[randint(a,b) for i in range(n)]`, sans oublier d'importer la fonction `randint` via `from random import randint`.