Fiche d'exercices
TD9 - Boucles imbriquées

Objectifs :

- Mettre en œuvre des parcours de séquences avec des boucles imbriquées
- Être capable de dérouler des programmes contenant des boucles imbriquées
- Savoir choisir le type de boucle le plus pertinent (for ou while)
- Concevoir des algorithmes intégrant plusieurs parcours de séquences

IMPORTANT : Vous ferez chaque exercice dans un fichier .py séparé

Exercice 1 : Lecture de programme

(1.1) Qu'affiche le programme suivant ?

```
1 n = 3
2 m = 4
3 for i in range(n):
4     for j in range(m):
5         print(f"i={i} : j={j}")
6     print()
```

(1.2) Combien de fois la boucle `for` externe (ligne 3) est-elle exécutée ? Combien de fois la boucle `for` interne (ligne 4) est-elle exécutée ?

(1.3) Qu'affiche le programme suivant ?

```
1 n = 5
2 for i in range(1, n+1):
3     for j in range(1, i+1):
4         print(j, end=" ")
5         j += 1
6     print()
```

Exercice 2 : Tables de multiplication

On souhaite afficher les tables de multiplications de 1 à 10 (voir exemple ci-dessous).

$$
\begin{aligned}
1\times1 &= 1\\
1\times2 &= 2\\
&\vdots\\
1\times9 &= 9
\end{aligned}
$$

$$
\begin{aligned}
2\times1 &= 2\\
2\times2 &= 4\\
&\vdots\\
2\times9 &= 18
\end{aligned}
$$

$$
\vdots
$$

$$
\begin{aligned}
9\times1 &= 9\\
&\vdots\\
9\times9 &= 81
\end{aligned}
$$

(2.1) Ecrivez un algorithme en pseudo-code permettant d'afficher ces tables de multiplication

(2.2) Traduire votre algorithme en programme en Python

Exercice 3 : Tri à bulles

Dans un TD précédent, vous avez vu le tri sélection. Dans cet exercice nous allons voir un autre algorithme de tri : le tri à bulles.

Soit une liste d'entiers de longueur n donnée en entrée. L'algorithme de tri à bulles s'effectue en plusieurs «passages», c'est-à-dire en plusieurs examens complets de la liste. Pour chaque passage, on compare successivement tous éléments adjacents et on les échange si le premier élément est supérieur au second. On peut montrer qu'après le premier passage, la plus grande valeur de la liste est située à la dernière position. De même, après le second passage, on peut montrer que la seconde plus grande valeur se retrouve à l'avant-dernière position. On peut ainsi montrer par récurrence qu'après n passages, la liste entière est triée, justifiant la correction de cet algorithme de tri. Un exemple de tri est donné dans la Table 1.

Ce tri doit son nom à l'analogie avec les bulles d'air qui remontent à la surface de l'eau : les grandes valeurs «remontent» vers la droite de la liste.

L'algorithme du tri à bulle (dans sa première version) peut s'écrire de la manière suivante :

6	0	3	5	7	4	2	Liste initiale
6	0	3	5	7	4	2	On compare 6 et 0 : on inverse
0	6	3	5	7	4	2	On compare 6 et 3 : on inverse
0	3	6	5	7	4	2	On compare 6 et 5 : on inverse
0	3	5	6	7	4	2	On compare 6 et 7 : on passe
0	3	5	6	7	4	2	On compare 7 et 4 : on inverse
0	3	5	6	4	7	2	On compare 7 et 2 : on inverse
0	3	5	6	4	2	7	Fin du premier passage
0	3	5	6	4	2	7	On compare 0 et 3 : on passe
0	3	5	6	4	2	7	On compare 3 et 5 : on passe
0	3	5	6	4	2	7	On compare 5 et 6 : on passe
0	3	5	6	4	2	7	On compare 6 et 4 : on inverse
0	3	5	4	6	2	7	On compare 6 et 2 : on inverse
0	3	5	4	2	6	7	On compare 6 et 7 : on passe
0	3	5	4	2	6	7	Fin du second passage

TABLE 1 – Exemple des 2 premiers passage du tri à bulle sur la liste [6, 0, 3, 5, 7, 4, 2].

```
# Soit une liste de taille n
# Pour n répétitions
#    Pour chaque couple d'elements adjacents a et b de la liste
#        Si a > b
#            Echanger les positions des elements a et b
```

(3.1) *Proposez un jeu de tests pour cet algorithme*

(3.2) *Implémentez l'algorithme du tri à bulles en Python, et vérifiez qu'il fonctionne sur votre jeu de tests.*

(3.3) *Modifier votre programme afin de compter le nombre total d'échanges d'éléments de liste effectués lors d'un tri. Pour ce faire, vous pourrez utiliser une variable* `compteur` *initialiée à 0, que vous augmenterez de 1 après chaque échange dans votre code.*

(3.4) *Implémentez la fonction* `liste_aleatoire(n)` *qui renvoie une liste de taille* `n` *d'entiers aléatoires compris entre 0 et 1000. Vous pouvez utiliser la fonction* `randint` *du module* `random`*.*

(3.5) *Utilisez la fonction* `liste_aleatoire` *afin d'afficher le nombre d'échanges effectués pour une liste aléatoire de taille 100. Lancez le code plusieurs fois pour observer les variations du compteur.*

(3.6) *Au lieu de lancer le code à la main plusieurs fois, on veut le faire automatiquement. Modifiez votre code précédent afin de lancer le tri à bulle sur 20 listes aléatoires différentes de taille 100.*

Pour s'entrainer

Exercice 4 : Instruction `if/in`

(4.1) Qu'affiche le programme suivant ? Décrivez en une phrase le but de ce programme

```
1  ma_chaine = "mercredi"
2  voyelles = ['a', 'e', 'i', 'o', 'u', 'y']
3  compteur = 0
4
5  for lettre in ma_chaine:
6      i = 0
7      while i < len(voyelles) and lettre != voyelles[i]:
8          i += 1
9      if i < len(voyelles):
10         compteur += 1
11
12 print(compteur)
```

(4.2) Comparez le programme suivant au programme ci-dessus. Qu'en concluez-vous sur l'instruction `if/in` ?

```
1      ma_chaine = "mercredi"
2      voyelles = ['a', 'e', 'i', 'o', 'u', 'y']
3      compteur = 0
4
5      for lettre in ma_chaine:
6          if lettre in voyelles:
7              compteur += 1
8      print(compteur)
```

Exercice 5 : Suite du tri à bulle

On souhaite maintenant analyser et optimiser le nombre de comparaisons entre deux valeurs de la liste effectuées durant le tri à bulles.

(5.1) En analysant votre code, indiquez combien de comparaisons entre valeurs adjacentes sont effectuées au plus lors du tri à bulles de la question 3.2 ? Attention ce nombre doit dépendre de la taille de la liste n.

On peut montrer par récurrence qu'après i passages du tri à bulles, les i plus grandes valeurs de la liste seront rangées à la fin de la liste. Cette propriété permet d'effectuer des passages plus courts, puisqu'il n'est pas nécessaire d'aller jusqu'à la fin de la liste, sauf pour le premier passage. Par exemple, dans la Table 1, la dernière comparaison du second passage (entre 6 et 7) n'est pas nécessaire.

(5.2) *Modifier votre version du tri à bulle en tenant compte de cette amélioration et comptez le nombre de comparaison effectué.*

(5.3) *En analysant votre code, indiquez combien de comparaisons entre valeurs sont effectuées au plus lors d'un tri à bulles améliorée (question (5.2))? Attention ce nombre doit dépendre de la taille de la liste n.*

Il est possible d'améliorer encore l'efficacité de l'algorithme, en remarquant que si aucun échange n'a été effectué lors d'un passage, alors on peut en conclure... Que la liste est triée ! Il n'est alors plus nécessaire de faire de nouveaux passages.

(5.4) *Modifier votre version précédente du tri à bulle en tenant compte de cette amélioration. Observer à nouveau que le nombre de comparaison diminue. Dans quel(s) cas le nombre de comparaisons de diminue-t-il pas ?*

Notons $f(n)$ le nombre moyen de comparaisons effectuées pour trier une liste de taille n. Ce nombre dépend de la taille de la liste ainsi que de son contenu. Afin de calculer la valeur moyenne de ce nombre de comparaisons il faut générer `nb_listes` listes différentes de taille n, et calculer la moyenne du nombre de comparaisons sur ces listes de même taille.

(5.5) *En utilisant votre code de la question (5.2), créez la fonction `nb_comp_bulle(liste)` qui renvoie le nombre de comparaisons qui ont été faites pour une liste donnée en paramètre.*

(5.6) *En utilisant les fonctions `liste_aleatoire(n)` et `nb_comp_bulle(liste)`, créez la fonction `nb_moyen_comp(n, nb_listes)`, qui renvoie le nombre moyen de comparaisons effectué pour trier `nb_listes` listes de taille `n`.*

(5.7) *En utilisant Matplotlib et votre fonction `nb_moyen_comp(n, nb_listes)`, tracez la courbe du nombre moyen de comparaisons du tri à bulle effectuées en fonction de la taille de la liste*

Exercice 6 : Recherche de motif

Dans la suite, on suppose que les variables `chaine` et `motif` contiennent des chaînes de caractères (String). On dira que `chaine` possède une occurence de `motif` à la position `i` si les lettres de `chaine` à partir de la position `i` coïncident avec celles de `motif`. Par exemple, `aaabaac` possède trois occurences du motif `aa`, en position 0, 1 et 4.
On dira que `motif` est une *sous-chaîne* de `chaine` si celle-ci possède au moins une occurence de `motif`.

(6.1) *Ecrivez une fonction qui teste si `motif` est une sous-chaine de `chaine`. Vous ne pouvez pas utiliser l'opérateur `in`*

(6.2) *Si le motif possède n caractères, et la chaîne en possède m, combien d'opérations élémentaires cette fonction pourra réaliser dans le pire cas ?*

(6.3) *Écrivez une fonction qui calcule le nombre d'occurence de `motif` dans `chaine`*