TP n°4 - Boucles imbriquées

1. Calculs de sommes doubles

Q1 : **a)** En utilisant Python, déterminer les valeurs des sommes suivantes :

$$\sum_{\substack{1\leq i\leq 3\\ 2\leq j\leq 4}} \frac{i}{j} \qquad \sum_{1\leq i<j\leq 5} (i^2-j)$$

b) Vérifier ces résultats "à la main".

Q2 : **a)** En utilisant Python, écrire des fonctions dépendant d'un argument entier naturel n permettant d'évaluer les sommes suivantes en fonction de n :

$$\sum_{\substack{1\leq i\leq n\\ 1\leq j\leq n}} 1 \qquad \sum_{\substack{1\leq i\leq n\\ 1\leq j\leq n}} i$$

$$\sum_{1\leq i<j\leq n} ij \qquad \sum_{1\leq i<j\leq n} (i+j)$$

b) En fonction de n, déterminer le nombre total d'itérations des boucles imbriquées lors de l'appel de chaque fonction.

c) Pour chaque somme, déterminer une formule mathématique donnant le résultat "immédiatement".

2. Recherches et boucles imbriquées

Q1 : **Recherche de deux valeurs les plus proches dans une liste**

Écrire une fonction `valeurs_proches(liste)` d'argument une liste de nombres flottants de taille au moins 2 et qui renvoie un tuple contenant les deux indices des deux valeurs les plus proches de la liste.

Exemples de comportement attendu :

```
>>> valeurs_proches([49.018, 27.56, 76.763, 66.197, 89.379, 23.617, 66.491, 85.96
    7, 40.191, 65.152])
(3, 6)
```

Q2 : **Recherche dans une liste de listes**

a) Écrire une fonction `tableau_aleatoire(n)` d'argument un entier naturel non nul `n` qui renvoie une liste de `n` listes qui contiennent chacune `n` nombres entiers tirés aléatoirement et uniformément entre `1` et n^2 - *on utilisera pour cela la fonction `randint(a,b)` du module `random`.*

Exemples de comportement attendu :

```
>>> tableau_aleatoire(n)
[[1, 3], [4, 1]]
>>> tableau_aleatoire(5)
[[17, 6, 19, 8, 12], [17, 22, 21, 22, 16], [22, 22, 8, 6, 22], [5, 11, 8, 15
    , 9], [16, 20, 1, 18, 25]]
```

b) Écrire une fonction `recherche_tableau(T,x)` d'arguments une liste `T` de listes de nombres entiers et un entier `x` et qui renvoie `True` si `x` se trouve dans une des sous-listes de `T` et `False` sinon.
Testez votre fonction sur un tableau généré par la fonction `tableau_aleatoire(n)`.

Q3 : Recherche d'un mot dans un texte

a) Écrire une fonction `recherche_mot(texte,mot)` d'arguments deux chaînes de caractères `texte` et `mot` et qui renvoie `True` si `mot` est une sous-chaîne de `texte` et `False` sinon.

Exemples de comportement attendu :

```
>>> recherche_mot("Trop bien l'Informatique avec M. Arnt", 'nul')
False
>>> recherche_mot("Trop bien l'Informatique avec M. Arnt", 'bien')
True
```

b) Écrire une fonction `recherche_indices_mot(texte,mot)` d'arguments deux chaînes de caractères `texte` et `mot` et qui renvoie la liste des indices de chaque occurence de `mot` dans `texte`.

Exemples de comportement attendu :

```
>>>recherche_indices_mot("Trop bien l'Informatique avec M. Arnt, des fois c'
    est nul mais là c'est bien bien bien", 'bien')
[5, 72, 77, 82]
>>>recherche_indices_mot("Trop bien l'Informatique avec M. Arnt, des fois c'
    est nul mais là c'est bien bien bien", 'bof')
[]
```

3. Tri à bulles

Le but de cette partie est de coder l'algorithme de **tri à bulles** qui permet de trier une liste de d'éléments ordonnable. On considère un tableau unidimensionnel `L` de taille `n=len(L)`. Le principe du tri à bulle est le suivant :

i) On parcourt le tableau `L` de gauche à droite avec `i` allant de `0` à `n-2` et on échange l'élément `L[i]` et `L[i+1]` si ces deux éléments sont mal ordonnés i.e. si `L[i]>L[i+1]`.

ii) Une fois le premier parcourt terminé, on recommence le parcourt du tableau du début et on recommence les échanges. Et ainsi de suite, on recommence à parcourir/échanger tant qu'il reste des échanges à effectuer.

iii) On renvoie le tableau : il est triée !

Dans la figure suivante, on peut voir le déroulement de l'algorithme de tri à bulles sur le tableau `L=[1,6,4,2,5,3]`.

1er Parcours						
1	6	4	2	5	3	Pas d'échange
1	6	4	2	5	3	Échange 6 et 4
1	4	6	2	5	3	Échange 6 et 2
1	4	2	6	5	3	Échange 6 et 5
1	4	2	5	6	3	Échange 6 et 3
1	4	2	5	3	6	Fin du 1er Parcours
2eme Parcours						
1	4	2	5	3	6	Pas d'échange
1	4	2	5	3	6	Échange 4 et 2
1	2	4	5	3	6	Pas d'échange
1	2	4	5	3	6	Échange 5 et 3
1	2	4	3	5	6	Pas d'échange
1	2	4	3	5	6	Fin du 2eme Parcours
3eme Parcours						
1	2	4	3	5	6	Pas d'échange
1	2	4	3	5	6	Pas d'échange
1	2	4	3	5	6	Échange 4 et 3
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Fin du 3eme Parcours
4eme Parcours						
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Fin du 4eme Parcours
Aucun échange dans le parcours n°4 : Fin de l'algorithme, on renvoie la liste triée.						

Q1 : Écrire une fonction `parcourt(L)` qui parcourt la liste `L` de gauche à droite et qui effectue les échanges comme indiqué au point i). Cette fonction renvoie `True` si elle a effectué des échanges lors du parcourt et `False` sinon.

Q2 : Écrire une fonction `triabulles(L)` qui effectue l'algorithme de tri à bulles - on se servira bien sûr de la fonction `parcourt(L)`.