Algorithmique avancée

—o000o– TD1 / TP1 –o000o—

On s'intéresse tout d'abord à différents traitements sur des tableaux / listes d'entiers.

1. écrire une fonction `generation(nbval, min, max)` qui retourne un tableau d'entiers avec `nbval` valeurs contenues entre les valeurs `min` et `max` incluses ; en Python 3, il faut importer le module `random` à l'aide de la commande `import random` et ensuite d'appeler la fonction `random.random()` pour générer un réel appartenant à $[0, 1[$. On pourra également utiliser les fonctions `math.floor()` pour déterminer le plus grand entier inférieur à un réel ou bien la fonction de transtypage `int()` pour générer des entiers.

2. **en TP** : reprendre cette question et essayer de générer une seconde solution purement Python avec des "list comprehension". Ensuite, utiliser le module `time.time()` pour comparer les temps d'exécution des deux fonctions de génération pour un nombre d'éléments dans les listes allant de 10 à 10 000. Dans la suite, utiliser le même mécanisme pour comparer vos solutions et évaluer les temps de calculs.

3. écrire une fonction `inverse(tab)` qui prend en argument une liste et retourne une nouvelle liste contenant les éléments de `tab` dans l'ordre inverse (le premier est en dernier etc.) ; proposer deux solutions différentes en Python.

4. écrire une fonction `unSurN(tab, n)` qui prend en argument une liste d'entiers ainsi qu'un entier `n` strictement positif et retourne une liste constituée uniquement d'un élément tous les `n` parmi ceux de la liste `tab` ; proposer deux solutions différentes en Python.

5. écrire une fonction `maxDes2(tab1, tab2)` qui prend en argument deux tableaux de nombres `tab1` et `tab2` de même longueur et retourne un tableau formé des valeurs maximales observées pour chaque indice entre les tableaux `tab1` et `tab2`. Par exemple, `maxDes2({1, 4, 5}, {2, 2, 3})` retourne le tableau `{2, 4, 5}` ;

6. écrire une fonction `myzip(tab1, tab2)` qui retourne une liste dont chaque élément d'indice `i` est lui-même une liste possédant deux valeurs issues des listes `tab1` et `tab2` à l'indice `i`. Par exemple, `myzip({1, 4, 5}, {2, 2, 3})` retourne la liste `{{1, 2}, {4, 2}, {5, 3}}` ; comparer votre solution à la fonction `zip()` de Python.

On se propose maintenant de programmer des opérations basiques sur des matrices de nombres réels. Il est possible de créer des tableaux à 2 dimensions en Python en générant des listes de listes (comme en Java). On utilise pour cela la notation suivante :

```
matrix = [[]]    # creation d'une matrice vide
matrix2 = [[random.random() for col in range(4)]
                  for row in range(6)]
        # 6 lignes et 4 colonnes de valeurs aleatoires
```

7. **en TP** : écrire une fonction `genMat(row, col, mini, maxi)` qui construit une liste de liste contenant `row` lignes et `col` colonnes et dont les valeurs sont comprises entre `mini` et `maxi` ;

8. écrire une fonction `diagonale(mat)` qui prend en argument une matrice de réels `mat` et retourne une liste contenant les éléments de sa diagonale ;

9. écrire une fonction `trace(mat)` qui prend en argument une matrice de réels `mat` et retourne la somme de ses éléments diagonaux ;

10. écrire une fonction `somme(mat1, mat2)` qui prend en argument deux matrices de réels `mat1` et `mat2` et retourne la matrice somme de ces deux matrices ;

11. écrire une fonction `produit(mat1, mat2)` qui prend en argument deux matrices de réels `mat1` de dimension $(n \times k)$ et `mat2` de dimension $(k \times m)$ et retourne la matrice produit de ces deux matrices de dimension $(n \times m)$. On rappelle que les éléments de la matrice résultat M sont définis comme suit :

$$\forall i \in [1, n], \forall j \in [1, m], M(i, j) = \sum_{l=1}^{k} mat_1(i, l) * mat_2(l, j)$$

On s'intéresse enfin à l'écriture de fonctions récursives.

12. écrire une fonction récursive `estDivisible(n, m)` qui retourne `true` si et seulement si `m` divise `n` et `false` sinon. Votre fonction ne doit pas utiliser les opérateurs de division \ ou le modulo `%` ;

13. écrire une fonction récursive `palindrome(tab)` qui indique si la liste `tab` passé en argument est symétrique ou pas. On dira que la liste est un palindrome si elle est vide, ne possède qu'un élément symétrique et si son premier élément est égal au dernier ;

14. écrire une fonction récursive `longueur(n)` qui prend en argument un entier naturel et retourne le nombre de chiffres qui le compose ;

15. écrire une fonction récursive `combienInf4(n)` qui prend en argument un entier naturel et retourne le nombre de chiffres qui le compose qui sont strictement inférieurs à 4 ;