Feuille 4

Exercice 1 Voici les commandes de base pour manipuler des fichiers textes (ou fichiers ascii, en opposition aux fichiers dits binaires). Dans les instructions suivantes, `nom_fichier` est une chaîne de caractères (nom du fichier) et `num_fichier` un entier (numéro logique associé au fichier).

```
num_fichier=open(nom_fichier)          # ouvre le fichier nom_fichier en lecture
                                       # et se place au début
num_fichier=open(nom_fichier,'w')      # ouvre ou crée le fichier  nom_fichier en écriture
                                       # et efface le contenu éventuel
ligne=num_fichier.readline()           # lit la ligne courante du fichier logique num_fichier
                                       # dans la chaine de caractères ligne
num_fichier.write(ligne)               # écrit la chaine de caractères ligne
                                       # dans le fichier logique num_fichier
num_fichier.close()                    # ferme le fichier logique num_fichier
```

1. Dans un script `test_ecriture.py` :
   — créer une liste d'abscisses $x_k = 2\pi k/n$ avec n suffisamment grand (par exemple $n = 50$),
   — créer une liste d'ordonnées $y_k = \cos(x_k)$ (la fonction `cos` se trouve dans le module `math`),
   — afficher la courbe avec la fonction `plot` de la librairie `matplotlib.pylot`,
   — stocker les deux listes dans un fichier "fichier.txt". Pour cela, on écrira une fonction `ecriture_fichier(nom_fichier)` prenant en argument le nom d'un fichier (chaîne de caractères) et écrivant ces deux listes en colonnes dans le fichier.

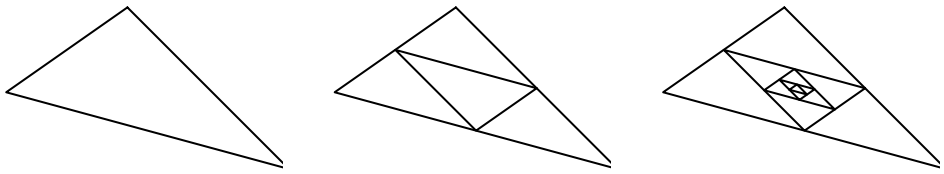
   Écrire un fonction `lecture_fichier` prenant en argument le nom d'un fichier (chaine de caractères) et renvoyant une liste de deux listes de même longueur contenant les deux premières colonnes de nombres stockés dans le fichier.

   Dans un script `test_lecture.py` :
   — Relire les deux listes d'abscisses et ordonnées en appelant la fonction `lecture_fichier`,
   — calculer une liste d'ordonnées $z_k = \sin(x_k)$,
   — afficher les deux courbes sur le même graphique,
   — stocker les trois listes dans le même fichier "fichier.txt" à la place des deux premières listes en utilisant la fonction `ecriture_fichier`.

Exercice 2 Utiliser le module `turtle` pour définir une fonction `triangle` qui prend quatre arguments `(a,b,c,niveau)` où `a`, `b` et `c` sont des listes de deux nombres réels et `niveau` est un entier. Cette fontion
— trace le triangle dont les sommets ont pour coordonnées `a`, `b` et `c` si `niveau==0`,
— trace ce triangle, ainsi que les quatre triangles dont les sommets sont les sommets de ce triangle ou les milieux de ses côtés si `niveau==1`,

— etc

Les figures suivantes correspondent aux cas `niveau==0`, `niveau==1` et `niveau==4`.



Exercice 3 Le module `turtle` permet de réaliser des graphiques en déplaçant une tortue sur l'écran. Par exemple, les instructions suivantes ouvrent une fenêtre graphique et dessinent un rectangle

```
import turtle
turtle.forward(100)
turtle.left(90)
turtle.forward(50)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(50)
```

Afficher la courbe $k \longmapsto v_k$ de l'exercice **??**. Voir la documentation de `turtle`.

Exercice 4 Bien qu'il existe déjà dans Python une méthode associée aux listes permettant de les trier (`Liste.sort()`), les exercices suivant traitent du problème du tri d'une liste. On propose de programmer diverses méthodes de tri. On vérifiera les résultats avec `sort`.

1. Tri par sélection. Écrire une fonction `TriSelection(L)` triant la liste `L` de nombres réels de la façon suivante. On prend le premier élément de la liste et on vérifie s'il en est le plus petit. Si oui, il conserve sa position, sinon on l'échange avec le plus petit. Ainsi le plus petit élément (plus exactement un plus petit élément) se trouve à la première position dans la liste. On met ensuite le deuxième plus petit élément à la deuxième position, ... et le plus grand à la dernière position.
2. Tri par insertion. Écrire une fonction `TriInsertion(L)` triant la liste `L` de la façon suivante. On commence par écrire une fonction `Insertion(T,x)` permettant d'insérer un élément `x` dans une liste triée `T`. On insére le deuxième élément dans la liste formée du premier élément, puis le troisième élément dans la nouvelle liste formée des deux premiers éléments, ...
3. Tri rapide (Quick sort). Écrire une fonction `TriRapide(L)` triant la liste `L` de la façon suivante. On place un élément (par exemple, le premier) qu'on appelle 'pivot', à sa position exacte si la liste avait été triée. C'est-à-dire, que les éléments situés avant (respectivement après) le pivot sont plus petits (respectivement plus grands) que le pivot. Puis

on refait la même chose avec les deux listes ainsi créées.
Écrire une fonction `Partition(L)` permettant de placer `L[0]` à sa positon exacte, puis une fonction `TriRapide`.

4. Comparer les temps d'exécution des trois programmes de tri sur une liste de grande taille. On prendra une liste de la forme `L=range(0,N)` qu'on mélange avec l'instruction `random.shuffle(L)`.

   Remarque : il est déconseillé de manipuler des listes de grande taille. Cet exercice est aussi un pretexte pour utiliser les modules `random` et `time`. Voir la documentation de ces modules.