# Spécialité NSI en terminale
# Exercices 2

## 1 Exercice 1

On programme le tri par insertion de manière récursive.
Le tri insertion est étudié en classe de première.

1. Écrire une fonction récursive `insertion` qui prend trois paramètres, une liste, un élément de la liste `x` et son indice `n`. Si `n` est strictement positif, on suppose les éléments d'indice 0 à `n-1` triés et la fonction insère l'élément `x` à la bonne place.
2. Écrire une fonction récursive `tri_insertion` qui prend en paramètres une liste et la longueur de la liste et qui trie la liste.

## 2 Exercice 2

Le valeur du pgcd (plus grand commun diviseur) de deux entiers naturels $a$ et $b$ est calculable à l'aide de l'algorithme d'Euclide : pgcd(a, b) = pgcd(b, r) ou r est le reste dans la division euclidienne de a par b. En voici une version itérative :

```
def pgcd(a, b):
    if b == 0:
        return a
    while b != 0:
        a, b = b, a % b
    return a
```

Écrire une version récursive de cette fonction.

## 3 Exercice 3

Écrire une fonction récursive `inverse` qui prend en paramètre une chaîne de caractères `ch` et renvoie la chaîne obtenue en inversant l'ordre des caractères.

Par exemple, `inverse("azerty")` a pour valeur la chaîne `"ytreza"`.

## 4 Exercice 4

On dispose d'une fonction `nettoie` qui prend en paramètre une liste triée et élimine les éléments identiques. Par exemple, si la liste est `liste=[1,1,2,6,6,6,8,8,9,10]`, après l'instruction `nettoie(liste)`, cette liste a pour valeur `[1, 2, 6, 8, 9, 10]`.

```
def nettoie(L):
    n = len(L)
    k = 0
    while k < n - 1:
        if L[k] != L[k+1]:
            k = k + 1
        else:
            del L[k]
            n = len(L)
```

Écrire une version récursive de la fonction `nettoie`.

## 5 Exercice 5

Utilisation du module Turtle
Tester le programme ci-dessous puis en écrire une version récursive.

```
from turtle import *

couleurs = ['blue', 'green', 'yellow', 'orange', 'red', 'purple']

bgcolor('black')

def dessin():
    for i in range(180):
        color(couleurs[i%6])
        forward(i)
        right(59)

dessin()
```

## 6 Exercice 6

On considère la fonction `mystere` définie ci-dessous.

```
def mystere(n, liste, g, d):
    """ la liste est triée """
    if g == d - 1:
        if n == liste[g]:
            return g
        return False
    else:
        m = (g + d) // 2
        if n == liste[m]:
            return m
        elif n < liste[m]:
            return mystere(n, liste, g, m)
        else:
            return mystere(n, liste, m, d)
```

1. On définit une liste par : `liste = [2, 5, 7, 8, 10, 13, 15, 16, 17, 20, 23, 24]`.
   On écrit `mystere(10, liste, 0, len(liste))`.
   Détailler l'exécution de la fonction en précisant les valeurs prises successivement par les trois variables `g`, `d` et `m`.
2. On conserve la même liste et on écrit `mystere(18, liste, 0, len(liste))`. Expliquer avec précision l'exécution de la fonction.