# TP d'informatique n°10
## (Python Maths)

# Manipulations de Tableaux bidimensionnels Complexité

L'objectif du TP est d'utiliser les structures de listes Python pour manipuler des tableaux bidimensionnels

## I Manipulations avancées

1. **a) C**réer un tableau bidimensionnel (une liste de listes…) rectangulaire à 8 lignes et 7 colonnes constitué de 0

   **b) C**réer un tableau bidimensionnel rectangulaire à 8 lignes et 7 colonnes et dont le p-ème élément de la n-ème ligne est $2n + p^2$.

   **c) C**réer un tableau bidimensionnel triangulaire à 8 lignes et dont la p-ième ligne possède p fois le nombre 1

   **d) C**réer une fonction prenant comme argument un entier naturel n non nul et qui retourne le tableau bidimensionnel triangulaire correspondant au triangle de Pascal à n lignes

2. **E**crire une fonction prenant comme argument une liste, et retournant la plus grande sous-suite croissante constituée de termes consécutifs de la liste. :

```
>>> plus_long_bloc_croissant([2, 10, 1, 3, 5, 7, 6, 8, 9])
[1, 3, 5, 7]
```

   On pourra faire quelques tests sur des listes aléatoires :

```
from random import randint
resultats = []
for i in range(100) :
        t = [randint(0, 10**4) for j in range(10**3)]
        resultats.append( len( plus_long_bloc_croissant(t) ) )

>>> moyenne(resultats) , maximum(resultats)
```

3. **S**ans utiliser la méthode sort, écrire une fonction qui prend comme argument une liste L de nombres entiers ou flottants et qui retourne la liste des éléments de L rangés dans l'ordre croissant. Pour cela on pourra commencer par chercher le minimum a de la liste L, mettre ce terme a dans la liste résultat et réitérer le processus sur la liste L privée de la première occurrence de a. Comparer la vitesse d'exécution de ce tri avec la méthode sort.

4. **A** l'aide de la fonction clock (ou la fonction time) du module time et de la fonction deepcopy du module copy, tester les différentes durées de création ou de copy de listes par les instructions :

```
from time import clock, time
from copy import deepcopy
```

```
debut = clock( )
T1 = tuple(" ")
for k in range(10**4) :
        T1 += (k,)
L0 = list(T1)
fin = clock( )
print(fin – debut)
```

```
debut = clock( )
L1 = [ ]
for k in range(10**4) :
        L1 += [k]
fin = clock( )
print(fin – debut)
```

```
debut = clock( )
L2 = [ ]
for k in range(10**4) :
        L2.append(k)
fin = clock( )
print(fin – debut)
```

```
debut = clock( )
L3 = [k for k in range(10**4) ]
fin = clock( )
print(fin – debut)
```

```
debut = clock( )
L4 = [0] * (10**4)
for k in range(10**4) :
        L4[k] = k
fin = clock( )
print(fin – debut)
```

Sous le même schéma, tester les durées de plusieurs méthodes de copie de listes, d'ajout d'une centaine de termes dans une longue liste sous différentes formes, de l'effet d'un changement de termes sur une copie d'une liste donnée…

# II Etude de la complexité d'une fonction inconnue

**S**oit la fonction suivante qui prend comme argument 2 entiers naturels non tous nuls

```
def fonction_1(a0, b0) :
        a, b = max(abs(a0), abs(b0)) , min(abs(a0), abs(b0))
        while b > 0:
                a, b = b, a%b
        return(a)
```

**5. a) Q**ue fait fonction_1(a0, b0) ?

**b)** Soit n un entier. On pose $u_n$ le nombre d'itérations dans la boucle while effectuées lors de l'appel à fonction_1(a0, b0) avec a0 et b_0 deux entiers positifs inférieurs ou égaux à $2^n$.
**M**ontrer que : $u_{n+1} = u_n + c_n$ avec $c_n = 0$, 1 ou 2
**E**n déduire que : $u_n \leqslant 2\,n$
Calculer la complexité de cette fonction lorsqu'on effectue l'appel fonction_1(a0, b0) en fonction de :

- la valeur maximale N de $a_0$ et $b_0$
- la longueur maximale L du nombre de chiffres (en base 10) de $a_0$ et $b_0$

**6) M**êmes questions avec fonction_2(a0, b0) ?

```
def fonction_2(a0, b0) :
        a, b = max(abs(a0), abs(b0)) , min(abs(a0), abs(b0))
        while b > 0:
                if a<2*b:
                        a, b = b, a-b
                else :
                        a, b = a-b, b
        return(a)
```