```
#############################################################################################
#                                    Pyramide de nombres                                    #
#############################################################################################

import time as t
import random as rd

p_exemple=[[4,2,8,3,2,9],[1,5,2,6,1],[5,8,9,3],[2,4,6],[7,4],[3]]



## Question 1
# Réponses :
# a- La pyramide est codée sous la forme d'une liste de listes d'entiers

# b- La commande p_exemple[1][3] retourne le nombre 6
#  C'est le nombre du 2nd étage en 4ième position en partant de la gauche

# c- pour avoir les nombres situés sous le nombre p_exemple[i][j] il faut utiliser
#  les commandes p_exemple[i-1][j] (à gauche) et p_exemple[i-1][j+1] (à droite)



## Question 2

def p_a(n):
    return [[rd.randint(0,9) for j in range(n-i)] for i in range(n)]

# Réponse :
# La fonction p_a(n) renvoi une pyramide de n étages avec des nombres
#  choisis aléatoirement entre 0 et 9 inclus



## Question 3

def affiche_p(p):                       # Fonction affichant la pyramide
    nb_eta=len(p)                       # Nombre d'étage de la pyramide
    nb_esp=len(p)-1                     # Nombre d'espacement pour mettre le sommet au centre
    for i in range(nb_eta-1,-1,-1):     # Parcours des étages du sommet (étage n) à la base
        for k in range(nb_esp):         # Placement horizontal du début du ième étage
            print('  ',end='',sep='')   # Affichage d'un espacement sans retour à la ligne
        for nbr in p[i]:                # Parcours d'un niveau de la pyramide
            if nbr<10:                  # Pour un nombre à un seul chiffre
                print(' ',nbr,'  ',end='',sep='') # affiche 1 espace le nombre et 2 espaces
            elif nbr<100:               # Pour un nombre à deux chiffres
                print(' ',nbr,' ',end='',sep='')  # affiche 1 espace le nombre et 1 espace
            else:                       # Pour un nombre à trois chiffres (ou plus)
                print(nbr,' ',end='',sep='')    # affiche le nombre et 1 espace
        print('')               # retour à la ligne
        nb_esp-=1               # Décalage de l'étage inférieur (1 espacement de moins)
```

```
def chemin_et_somme_naif(p):
    n=len(p)                    # Nombre d'étage de la pyramide
    if n==1:                    # Si la pyramide n'a qu'un seul étage (un seul nombre)
        return ([],p[0][0])     #   on renvoie un chemin vide et ce nombre
    else:                       # Sinon
                                # On construit la pyramide inférieure de gauche
        pg=[[p[i][j] for j in range(len(p[i])-1)] for i in range(n-1)]
                                # On construit la pyramide inférieure de droite
        pd=[[p[i][j] for j in range(1,len(p[i]))] for i in range(n-1)]
            # On résout le problème sur la pyramide de gauche
        CG,SG=chemin_et_somme_naif(pg)
            # On résout le problème sur la pyramide de droite
        CD,SD=chemin_et_somme_naif(pd)
            # On choisit la pyramide ayant la somme maximale
        if SG>SD:   # Si c'est celle de gauche qui a la somme maximale
                    # On ajoute 'Gauche' au chemin précédent
                    # Et le nombre au sommet à la somme de gauche
            return (['Gauche']+CG,SG+p[n-1][0])
        else :      # Si c'est celle de droite qui a la somme maximale
                    # On ajoute 'Droite' au chemin précédent
                    # Et le nombre au sommet à la somme de droite
            return (['Droite']+CD,SD+p[n-1][0])

# Réponses :
# a- La fonction chemin_et_somme_naif(p) prend en argument une pyramide de nombre
#  et retourne une liste et un entier. L'entier est la somme maximale qui peut
#  être obtenue en parcourant cette pyramide de son sommet à sa base. Et la liste
#  est celle des directions ('Gauche' ou 'droite') à prendre  dans la pyramide
#  lorsqu'on descend du sommet à la base pour avoir la somme maximale.

# b- L'algorithme de cette fonction est un algorithme récursif

## Question 4

def test_naif(n):
    p=p_a(n)
    print('Voici la pyramide aléatoire de :',n,'étages')
    affiche_p(p)
    t1=t.time()
    C,S=chemin_et_somme_naif(p)
    t2=t.time()
    print('La somme maximale pouvant être obtenue est de :',S)
    print('Les directions à prendre pour obtenir cette somme sont :')
    print(C)
    print('La résolution a été faite en',t2-t1,'secondes')

# Réponses :
# Etant donné qu'à chaque fois que le nombre d'étage est augmenté d'une unité
#  la durée d'exécution de la fonction chemin_et_somme_naif(p) est multiplié
#  par 2, on en déduit que cette fonction est en complexité O(2^n)
# Cet algorithme est trop complexe pour une vingtaine d'étages et même impossible
#  à utiliser au delà d'une trentaine d'étages
```

```
## Question 5
# Code

def pyramide_des_sommes(p):           # Fonction construisant la pyramide des sommes maximales
    n=len(p)
    s=[[p[0][i] for i in range(n)]]     # On recopie la base de la pyramide p
    for i in range(1,n):                # Parcours les étages de p du 2nd au sommet au dernier
        etage=[]                        # On initialise la liste de l'étage
        for j in range(len(p[i])):      # Parcours de l'étage de gauche à droite
                # valeur de la pyramide p + maximum des 2 valeurs inférieures de la pyramide s
            etage.append(p[i][j]+max(s[i-1][j],s[i-1][j+1]))
        s.append(etage)                 # On ajoute la liste de l'étage à la pyramide s
    return s                            # On retourne la pyramide des sommes maximales

## Question 6
# Code

def chemin_et_somme(p):
    n=len(p)
    s=pyramide_des_sommes(p)        # Construction de la pyramide des sommes maximales
    Descente=[]                     # Initialisation de la liste des directions
    position=0                      # Position de la valeur sur le chemin optimal
    for i in range(n-2,-1,-1):      # Parcours du haut vers le bas de la pyramide
        if s[i][position]>s[i][position+1]:     # Si le maximum est en bas à gauche
            Descente.append('Gauche')           # On ajoute Gauche à la liste des direct°
                                    # On ne modifie pas la variable position car l'indice
                                    #  du nombre en bas à gauche est la même
        else:                       # Si le maximum est en bas à droite (pas à gauche)
            Descente.append('Droite')   # On ajoute Droite à la liste des direct°
            position+=1             # L'indice sur le chemin optimal est incrémenté
                                    #  car l'indice du nombre en bas à droite et plus élevé
    return Descente,s[n-1][0]


## Question 7
# Réponse :
# La fonction chemin_et_somme(p) utilise la fonction pyramide_des_sommes(p) laquelle
#  est en complexité O(len(p)^2) (Car deux boucles for imbriquées) puis utilise une
#  boucle for de complexité O(len(p))
# Donc cette fonction est en complexité O(len(p)^2).
# On peut aisément l'utiliser pour une pyramide d'un grand nombre d'étages
```