```python

## Exercice 1
def PGCDv1(a,b): #version non recursive
    '''a et b sont deux entiers naturels, non tous les deux nuls'''
    c=a
    d=b
    while d!=0:
        if c<d:
            (c,d)=(d,c)  #on echange c et d
        (c,d)=(c-d,d) #PGCD(c,d)=PGCD(c-d,c)
    #quand d=0 , c contiendra le PGCD cherche
    return c

def PGCDv2(a,b): #version recursive
    '''a et b sont deux entiers naturels, non tous les deux nuls'''
    if a==0:
        return(b)
    if b==0:
        return(a)
    else:
        if a>=b:
            return PGCDv2(a-b,b)
        else:
            return PGCDv2(a,b-a)


## Exercice 2
#On a besoin des fonctions primitives sur les piles que l'on va d'abord implémenter
def creer_pile():
    return []

def est_vide(p):
    '''teste si une pile est vide'''
    return nb_elements(p)==0

def empiler(p,v):
    '''retourne la pile p auquelle on a ajoute la valeur v en dernier'''
    p.append(v)

def depiler(p):
    '''enleve la derniere valaur de la pile p et la renvoie'''
    assert est_vide(p)==False #on ne peut pas dÃ©piler une pile vide
    return p.pop()

def parenthesage(expr):
    '''teste le bon parenthésage d'une chaine de caractère'''
    p=creer_pile()    #création de la pile
    for x in expr:  #parcours de l'expression
        if x=='(':
           empiler(p,x) # si on rencontre une parenthèse ouvrante on l'empile
        if x==')':          #si on rencontre une parenthèse fermante...
            if est_vide(p):
                return False  #si il n'y a pas déjà de parenthèses ouvrantes, c'est pas bon
            else:
                depiler(p)                          #si il y avait une parenthèse ouvrante dans la pile on l'enlève
    if est_vide(p):
        return True   # à la fin, c'est bon si et seulement si il n'y a plus rien dans la pile
    else:
        return False

## Exercice 3

def voisins(M,i,j):
    '''retourne la liste des coordonnees des voisins de M[i][j]'''
    n=len(M)
    p=len(M[0])
    if i>0 and i<n-1 and j>0 and j<p-1:
        #cas où on n'est pas sur le bord
        liste_voisins=[[i-1,j-1],[i-1,j],[i-1,j+1],[i,j-1],[i,j+1],[i+1,j-1],[i+1,j],[
```

```
            i+1,j+1]]
        if i==0 and j==0:
            liste_voisins=[[0,1],[1,0],[1,1]]
        if i==n-1 and j==0:
            liste_voisins=[[n-2,0],[n-2,1],[n-1,1]]
        if i==0 and j==n-1:
            liste_voisins=[[0,n-2],[1,n-2],[1,n-1]]
        if i==n-1 and j==n-1:
            liste_voisins=[[n-2,n-2],[n-1,n-2],[n-2,n-1]]
        if i==0 and j>0 and j<p-1:
            liste_voisins=[[i,j-1],[i,j+1],[i+1,j-1],[i+1,j],[i+1,j+1]]
        if i==n-1 and j>0 and j<p-1:
            liste_voisins=[[i-1,j-1],[i-1,j],[i-1,j+1],[i,j-1],[i,j+1]]
        if j==0 and i>0 and i<n-1:
             liste_voisins=[[i-1,j],[i-1,j+1],[i,j+1],[i+1,j],[i+1,j+1]]
        if j==p-1 and i>0 and i<n-1:
             liste_voisins=[[i-1,j-1],[i-1,j],[i,j-1],[i+1,j-1],[i+1,j]]
        return liste_voisins

    def composantes_connexes(M):
        '''retourne la liste des composantes connexes rouge dans l'image M'''
        n=len(M)
        p=len(M[0])
        visites=[[False for j in range(p)] for i in range(n)]
        composantes=[]
        for i in range(n):
            for j in range(p):
                if visites[i][j]==False:
                    visites[i][j]=True
                    if M[i][j]=='R':
                        pile=creer_pile()
                        empiler(pile,[i,j])
                        comp=[[i,j]]

                        while est_vide(pile)==False:
                            i0,j0=depiler(pile)
                            #print(i0,j0)
                            liste_voisins=voisins(M,i0,j0)
                            print(liste_voisins)
                            for vois in liste_voisins:

                                i1,j1=vois
                                if visites[i1][j1]==False:
                                    #print(i1,j1)
                                    visites[i1][j1]=True
                                    if M[i1][j1]=='R':
                                        comp.append([i1,j1])
                                        empiler(pile,[i1,j1])
                        composantes.append(comp)
        return composantes
```