```python
##Exercice 1########☺

#Implementation des fonctions primitives pour travailler avec les piles
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 ajouté la valeur v en dernier'''
    p.append(v)

def depiler(p):
    '''enlève la dernière valaur de la pile p et la renvoie'''
    assert est_vide(p)==False #on ne peut pas dépiler une pile vide
    return p.pop()

def renverse(p):
    '''à partir d'une pile p, renvoie une pile q dont les éléments sont
    ceux de p dans l'ordre inverse'''
    q=creer_pile()
    while est_vide(p)==False:
        a=depiler(p)
        empiler(q,a)
    return q

# l'inconvénient de ce programme est qu'à l'issue de son execution, la pile p est vide

def renverse2(p):
    n=nb_elements(p)
    q=creer_pile()
    pile_aux=creer_pile() #une pile auxiliaire
    for i in range(n):
        a=depiler(p)
        empiler(q,a)
        empiler(pile_aux,a)
    #on va à présent reconstituer p
    for i in range(n):
        a=depiler(pile_aux)
        empiler(p,a)
    return q

##Exercice 2 - Gestiond 'une file de commande
from collection import deque

def annuler(F,commande):
    '''annuler la commande de la file F'''
    F2=deque([])
    while F!=deque([]):
        x=F.popleft()
        if x!=commande:
            F2.append(x)

    while F2!=deque([]):
        x=F2.popleft()
        F.append(x)

def prioriser(F,commande):
    '''mettre une commande en tête de file'''
    F2=deque([commande])
    while F!=deque([]):
        x=F.popleft()
        if x!=commande:
            F2.append(x)

    while F2!=deque([]):
        x=F2.popleft()
        F.append(x)
```

```python
## Exercice 3 - programmation récursive
def somme(n):
    '''calcul de la somme des carrés des n premiers entiers naturels'''
    if n==0 :
        return(0)
    return somme(n-1)+n*n

## Exercice 4
def puiss_v1(x,n):
    '''calcul de x puissance n de manière itérative'''
    p=1
    for i in range(n):
        p=p*x
    return p

def puiss_v2(x,n):
    '''calcul de x puissance n de manière récursive'''
    if n==0:
        return(1)
    else:
        return x*puiss_v2(x,n-1)

## Exercice 5
def inverse(mot):
    '''mot est une chaine de caracteres'''
    n=len(mot)
    if n==0:
        return mot
    else:
        return mot[n-1]+inverse(mot[:n-1])

def palindrome(mot):
    '''teste si un mot est un palindrome'''
    return inverse(mot)==mot

## Exercice 6
def occurrences(lettre,mot):
    '''compte le nombre de fois où une lettre donnée apparait dans un mot'''
    if mot=='':
        return 0
    if mot[0]==lettre:
        return 1+occurrences(lettre,mot[1:])
    else:
        return occurrences(lettre,mot[1:])
```