```python


# %% Exercice 1 - programmation récursive de la somme des carrés
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 2 - puissance d'un nombre
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 3 - Inverse d'un mot et palindrome
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 4 - Nombre d'occurrences d'une lettre dans un mot
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:])

# %% Exercice 5  - Inverser une pile

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=[]
    while p!=[]:
        a=p.pop()
        q.append(a)
    return q

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

def renverse2(p):
    q=[]
    pile_aux=[]
    while p!=[]:
        a=p.pop()
        q.append(a)
        pile_aux.append(a)

    #on va à présent reconstituer p
    while pile_aux!=[]:
        a=pile_aux.pop()
        p.appnd(a)
```

```
    return q

# %% Exercice 6 - Gestion d'une file de commande
from collections import deque

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

    #on reconstitue la file F (sans la commande qui a été enlevée)
    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)

    #on reconstitue la file F
    while F2!=deque([]):
        x=F2.popleft()
        F.append(x)

# %% Exercice 7 - PGCD de deux entiers

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 8 - Tous les mots binaires de n chiffres

def mots_binaires(n):
    '''retourne la liste de tous les mots binaires de n lettres'''
    if n==0:
        return ['']
    liste_apres=[]  #la liste avec n lettres , on va la construire
    liste_avant=mots_binaires(n-1) #la liste avec n-1 lettres
    for mots in liste_avant:
        liste_apres.append(mots+'0')
        liste_apres.append(mots+'1')
    return liste_apres
```