```
########################################
#                                      #
#          Correction DS 2             #
#                                      #
########################################



# Exercice 1

def moy_ecartype(L):
    s = 0
    for k in L:
        s += k
    moy = s/len(L)
    ectp = 0
    for k in L:
        ectp += (k - moy)**2
    return(moy, (ectp/len(L))**0.5)

def moy_sup10(L):
    s, n = 0, 0
    for k in L:
        if k >= 10:
            s += k
            n += 1
    if n == 0 : return(0)
    else : return(s/n)

def moy_indeximpair(L):
    s = 0
    for k in L[1::2]:
        s += k
    return(s/(int((len(L)-1)/2)))

def moy_pairs(L):
    s,n = 0,0
    for k in L:
        if k%2 == 0:
            s += k
            n += 1
    return(s/n)


#  Exercice 2

minuscules=[chr(k) for k in range(97, 123)]
majuscules = [chr(k) for k in range(65,91)]
alphabet = minuscules+majuscules

phrase = "Ceçi est LA PhrâsE qui nous permet "+
"d'éssayèr les Différentes fonctions"
```

```
def enleve_e_aigu(phrase):
    """
La fonction change tous les 'é' en 'e'
"""
    res =''
    for k in phrase:
        if k == 'é' :
            res += 'e'
        else :
            res += k
    return(res)



def En_minuscule(phrase):
    """
La fonction transforme toutes les majuscules en minuscules
"""
    Maj = [chr(k) for k in range(65,91)]
    # On pouvait aussi ecrire Maj = ABCDEFGHIJKLMNOPQRSTUVWXYZ
    Min = [chr(k) for k in range(97,123)]
    res = ''
    for k in phrase:
        if k in Maj:   # ou écrire une fonction retrouvant l'indice
            res += Min[Maj.index(k)]
        else : res += k
    return(res)




def Enleve_accent_et_cedille(phrase):
    """
Enlève les accents et les cédilles
"""
    accent = 'àâçéèêëîïôù'
    sansaccent = 'aaceeeeiiou'
    res = ''
    for k in phrase:
        if k in accent: res += sansaccent[accent.index(k)]
        else : res += k
    return(res)



def Nombre_e(phrase):
    """
Détermine le nombre de 'ee dans la phrase
"""
    chaine = En_minuscule(phrase)
```

```
    chaine = Enleve_accent_et_cedille(chaine)
# On pourrait écrire : chaine.count('e')
    n = 0
    for k in chaine :
        if k == 'e': n += 1
    return(n)


def Frequence_voyelle(phrase):
    voy ='aeiouy'
    lettre = 'abcdefghijklmnopqrstuvwxyz'
    chaine = En_minuscule(phrase)
    chaine = Enleve_accent_et_cedille(chaine)
    nvoy, nlettre = [0,0,0,0,0,0],0
    for k in chaine :
        if k in voy : nvoy[voy.index(k)] += 1
        if k in lettre : nlettre += 1
    return([k/nlettre for k in nvoy])


# Exercice 3

## Les entiers compris entre 0 et 15 sont, s'ils sont écrits en base 2 :
## [], [1], [0,1], [1,1], [0,0,1], [1,0,1], [0,1,1], [1,1,1],
## [0,0,0,1], [1,0,0,1], [0,1,0,1], [1,1,0,1], [0,0,1,1], [1,0,1,1,], [0,1,1,1],
[1,1,1,1]
## Au total on a besoin de 49 chiffres
##
## En code CLE, ces mêmes entiers sont écrits sous la forme :
## [], [0], [1], [0,1], [2], [0,2], [1,2], [0,1,2], [3], [0,3], [1,3], [0,1,3],
[2,3], [0,2,3], [1,2,3], [0,1,2,3]
## Au total, cela nécessite  32 chiffres


## On note S(p) le nombre de chiffres utilisés pour écrire tous les entiers en
base 2 de 0 jusqu'à 2p - 1
## et T(p) le même nombre de chiffres mais pour l'écriture en code CLE.  On a :
S(p) = 2p * M(p)  et  T(p) = 2p *C(p)
## Pour obtenir le nombre de chiffres nécessaires pour décrire en base 2 tous
les entiers entre 0 et 2^(p+1) - 1,
## il faut d'abord décrire ceux qui sont entre 0 et 2^(p) - 1 (ce qui nécessite
S(p) chiffres)
## puis décrire ceux qui se trouvent entre 2^p et 2^(p+1) - 1 . Or ces derniers
ont tous p+1 chiffres en base 2.
## Ainsi  S(p+1) = S(p) + (p+1)*2^p.   Ainsi  M(p+1) = (M(p) + p + 1) / 2.
## Or :   M(1) = 1/2, M(2) = 5/4 , M(3) = 17/8  donc si on cherche M(p) sous la
forme proposée par l'énoncé, on obtient :
## M(p) =   p - 1 + 1/(2^p)
## Pour obtenir le nombre de chiffres nécessaires pour décrire en code CLE tous
les entiers entre 0 et 2^(p+1) - 1,
## il faut d'abord décrire ceux qui sont entre 0 et 2^(p) - 1  (ce qui nécessite
 T(p) chiffres)
```

```
## puis décrire ceux qui se trouvent entre 2^p et 2^(p+1) - 1 .
## Or ces derniers s'obtiennent en rajoutant aux listes CLE des entiers compris
entre 0 et  2^(p) - 1 le terme "p"
## (ce qui nécessite donc T(p) + 2^p chiffres).Ainsi  T(p+1) = 2*T(p) + 2^p.
## Ainsi  C(p+1) = C(p) +  1 / 2.  Or : C(1) = 1/2, donc on obtient : C(p) = p/2

## On a , pour p >1, C(p) < M(p) :
##    il faut utiliser moins de chiffres en code CLE qu'en binaire direct.
##    Plus précisément, lorsque p tend vers l'infini, le rapport de ces moyennes
tend vers 1/2.
##    Ceci s'explique par le fait, que le code CLE consiste à ne retenir que les
places des 1 en base 2 .




def convbin(N):
    n = N
    res =[]
    while n>0:
        res = [n%2] + res
        n = n//2
    return(res)

def convCLE(N):
    L = convbin(N)[::-1]
    res=[]
    for indice, nb in enumerate(L):
        if nb == 1: res +=[indice]
    return(res)


def deconvCLE(L):
    res = 0
    for k in L:
        res += 2**k
    return(res)


def divparpuis2(L, k):
    if L[0] >= k : return(True)
    return(False)
```