```python

#exo1
def coorVecteur(xA,yA,xB,yB):
    x_AB=xB-xA
    y_AB=yB-yA
    AB=(x_AB,y_AB)
    return AB

#exo2
def coorVecteur(xA,yA,xB,yB):
    x_AB=xB-xA
    y_AB=yB-yA
    AB=(x_AB,y_AB)
    return AB
def multVecteurk(k,xA,yA,xB,yB):
    AB=coorVecteur(xA,yA,xB,yB)
    x_kAB=AB[0]*k
    y_kAB=AB[1]*k
    kAB=(x_kAB,y_kAB)
    return kAB


#exo3
def affichageTuple(monTuple):
    i=0
    for elt in monTuple:
        print(i,elt)
        i=i+1
    return None

def affichageTuple2(monTuple):
    n = len(monTuple)
    for i in range(n):
        print(i," ",monTuple[i])
    return None

#exo4
def longueurTuple(monTuple):
    long=0
    for elt in monTuple:
        long=long+1
    return long


#exo5
def affichagePair(monTuple):
    for elt in monTuple:
        if elt%2==0:
            print(elt,end=" ")
    return None


#exo6
def verifElement(val,monTuple):
    n = len(monTuple)
    for i in range(n):
        if monTuple[i]==val:
            return i
    return False


L = (5,4,8,9,1,4)
L1 = (4 ,3 ,1)
x=9
n=3

print (x in L)
print (x in L1)


print (x not in L)
print (x not in L1)
```

```python

print(len(L))
print(len(L1))

print(L==L1)


print(L[0])
print(L1[0])

print(L[1:4])
print(L1[0:2])

print(L.index(x))
#print(L1.index(x))

print(L.count(x))
print(L1.count(x))

print(L+L1)

print(L*n)

#exo8
def maxiTuple(monTuple):
    maximum = monTuple[0]
    for i in range(1,len(monTuple)):
        if monTuple[i] > maximum:
            maximum = monTuple[i]
    return maximum

#def maxiTuple2(monTuple):
    #return max(monTuple)

#exo9
def comptage(monTuple,valeur):
    compteur = 0
    for elt in monTuple:
        if valeur == elt:
            compteur = compteur + 1
    return compteur

def comptage2(monTuple,n):
    return monTuple.count(n)

#Exo10
def sommePlus(monTuple):
    n=len(monTuple)
    somme=0
    for i in range(n):
        if monTuple[i]>0:
            somme=somme+monTuple[i]
    return somme

def sommePlusbis(monTuple):
    somme = 0
    for elt in monTuple:
        if elt > 0:
            somme = somme + elt
    return somme

#Exo11
def sommePlus2(monTuple):
    n=len(monTuple)
    somme=0
    for i in range(n):
        if monTuple[i]>0 and monTuple[i]%2==0:
            somme=somme+monTuple[i]
    return somme

def sommePlus2bis(monTuple):
    somme = 0
```

```
    for elt in monTuple:
        if (elt > 0) and (elt % 2 == 0):
            somme = somme + elt
    return somme

#exo12
L=[4,2,1,3,4,6]

L.insert(4,5)
[4, 2, 1, 3, 5, 4, 6]

del L[3]
[4, 2, 1, 5, 4, 6]

L.pop(4)
[4, 2, 1, 5, 6]

L.insert(2,15)
[4, 2, 15, 1, 5, 6]


#exo13
L1=[]
n=10000
for i in range(n+1):
    L1.append(i)

L2=[i for i in range(n+1)]

#exo14
n=10000
L3=[i for i in range(0,n+1,2)]

L3bis = [i for i in range(10001) if i%2 == 0]

#exo15
n=10
L4=[2**i for i in range(n+1)]

n=10
L4bis=[]
for i in range(n+1):
    L4bis.append(2**i)

##exo16
from random import *

def liste5(n):
    L5=[]
    L5=[randint(0,20) for i in range(n)]
    return L5



##exo16bis
from random import *
def liste5(n):
    L5 = []
    i=1
    a = randint(0,20)
    while a != 0 and i<=n:
        L5.append(a)
        a = randint(0,20)
        i=i+1
    return L5


def liste5bis(n):
    L5bis=[]
    i=0
    a=randint(0,20)
    while a!=0 and i<n:
```

```
            L5bis.append(a)
            a=randint(0,20)
            i=i+1
        return L5bis

#exo17
L = [[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]]

L[1]
[6, 7, 8, 9, 10]

L[2][3]
14

len(L)
3

len(L[0])
5

#exo18
A = [[1 ,2 ,3] ,[4 ,5 ,6] ,[7,8,9]]

m = len(A)
n = len(A[0])
for j in range(n):
    for i in range(m):
        print(A[i][j], end=" ")

print()

for sousliste in A:
    for element in sousliste:
        print(element,end(" ")

#exo19
B = [[4 ,5] ,[2 ,9] ,[3 ,7]]

B[0].reverse()
B[1].reverse()
B[2].reverse()
print(B)




B1 = [[4 ,5] ,[2 ,9] ,[3 ,7]]
for i in range(3):
    B1[i][0],B1[i][1] = B1[i][1],B1[i][0]

#exo20
C = [[4 ,5] ,[2 ,9] ,[3 ,7]]
somme=0
m=len(C)
n=len(C[0])
for i in range(m):
    for j in range(n):
        somme=somme+C[i][j]
print(somme)

def somme_Termes_Matrice(matrice):
    somme = 0
    for ligne in matrice:
        for elt in ligne:
            somme = somme + elt
    return somme

#exo21
D = [[1 ,4] ,[9 ,5] ,[7 ,2]]

m=len(D)
n=(len(D[0]))
```

```python
max=D[0][0]
for i in range(m):
    for j in range (n):
        if D[i][j]>max:
            max=D[i][j]
print(max)

"""
La méthode pour trouver le maximum est la suivante :
on crée une variable max à laquelle on affecte la valeur du premier élément
 de la liste. Puis on parcourt le tableau et on compare à chaque fois
  l'élément atteint avec max. Si l'élément est supérieur ou égal à max,
   on affecte cet élément à max.
"""

D = [[1 ,4] ,[9 ,5] ,[7 ,2]]
def plus_grande_valeur_matrice(matrice):
    maximum = matrice[0][0]
    for ligne in matrice:
        for elt in ligne:
            if elt > maximum:
                maximum = elt
    return maximum

#exo22
E = [[2 ,1 ,3] ,[4 ,8 ,6] ,[7,5,9]]

m=len(E)
n=(len(E[0]))

for i in range(m):
    for j in range (n):
        if E[i][j]%2==0:
            E[i][j]=True
        else :
            E[i][j]=False
print(E)

def matrice_parite(matrice):
    for i in range(len(matrice)):
        for j in range(len(matrice[i])):
            if matrice[i][j] % 2 == 0:
                matrice[i][j] = True
            else:
                matrice[i][j] = False
    return matrice

def matrice_parite2(matrice):
    M_partite = [[(matrice[i][j] % 2 == 0) == True for j in range(3)] for i in
    range(3)]
    return M_partite

#exo23
def sommeElementCourant(F):
    m=len(F)
    n=(len(F[0]))
    for i in range(m):
        somme=0
        for j in range (n):
            somme=somme+F[i][j]
            F[i][j]=somme
    return F

#exo24
#sol non optimale
def verifMagique(matrice):
    # calcul sommes des lignes
    listeSommeLigne=[]  # liste contenant la somme des lignes [somme ligne 0 , somme
    ligne 1 ,somme ligne2]
    for i in range(3):
        sommeLigne=0    # remise à zéro de la somme à chaque ligne
        for j in range (3):
```

```
359                 sommeLigne=sommeLigne+matrice[i][j]
360             listeSommeLigne.append(sommeLigne)
361         print(listeSommeLigne)
362 
363         # calcul sommes des colonnes
364         listeSommeColonne=[]  # liste contenant la somme des colonnes [somme colonne0 ,
            somme colonne1 ,somme colonne2]
365         for j in range(3):
366             sommeColonne=0    # remise à zéro de la somme à chaque ligne
367             for i in range (3):
368                 sommeColonne=sommeColonne+matrice[i][j]
369             listeSommeColonne.append(sommeColonne)
370         print(listeSommeColonne)
371 
372         # calcul somme diagonale 1 (du haut vers le bas)
373         sommeDiagonale1 =0
374         for k in range(3):
375             sommeDiagonale1=sommeDiagonale1+matrice[k][k]
376         print(sommeDiagonale1)
377 
378         # calcul somme diagonale 2 (du bas vers le haut)
379         sommeDiagonale2 =0
380         for k in range(3):
381             sommeDiagonale2=sommeDiagonale2+matrice[k][2-k]
382         print(sommeDiagonale2)
383 
384         # vérification carré magique ordre 3 (vérif des sommes)
385         return (listeSommeLigne==listeSommeColonne and sommeDiagonale1==sommeDiagonale2)
386 
387     #sol optimale
388     def verifMagique2(matrice):
389         somme_en_liste = []
390         # verif somme des lignes
391         for i in range(3):
392             somme = 0
393             for j in range(3):
394                 somme = somme + matrice[i][j]
395             somme_en_liste.append(somme)
396         # verif somme des colonnes
397         for j in range(3):
398             somme = 0
399             for i in range(3):
400                 somme = somme + matrice[i][j]
401             somme_en_liste.append(somme)
402         # verif somme diagonales
403         somme1 = 0
404         for k in range(3):
405             somme1 = somme1 + matrice[k][k]
406         somme_en_liste.append(somme1)
407         somme2 = 0
408         for k in range(3):
409             somme2 = somme2 + matrice[k][2-k]
410         somme_en_liste.append(somme2)
411 
412         for i in range(len(somme_en_liste)-1):
413             if somme_en_liste[i] != somme_en_liste[i+1]:
414                 return False
415         return True
416 
417     #exo25
418     from random import *
419     def matriceAlea(m,n):
420         L = []                          # Initialisation d'une liste vide
421         for i in range(m):              # Balayage des sous listes
422             SL = []                     # Creation d'une sous liste vide
423             for j in range(n):          # Balayage des elements de la sous liste
424                 SL.append(randint(0,100))       # Ajout d'un elemet a droite de la sous
                    liste
425             L.append(SL)
426         return L
427 
```