```
"""Correction CCINP TSI 2025 :  Conduite autonome"""
# Q1                                                        13h30
dico = {'adulte':[],'enfant':[],'animaux':[],'véhicule':[],'adulte':[],'indéterminé':[]}
# Q2
Lm = [175,110,50,200,50*0.8/1.2]
i = 0
for e in dico :
    dico[e].append(Lm[i]*0.8)
    dico[e].append(Lm[i]*1.2)
    i = i+1
dico['indéterminé'][0]=0
# Q3
def obstacles_rencontres(dico,hauteur):
    s = []
    for e in dico:
        if dico[e][0] <= hauteur <= dico[e][1]:
            s.append(e)
    return s
# Q4
def focale(faces_detec,Dr,Hr):
    f=[]
    n = len (faces_detec)
    for i in range(n):
        f.append(faces_detec[i][2]/Hr*Dr)
    return f
# Q5
def moy(L):
    s = 0
    n = len(L)
    for i in range(n):
        s = s + L[i]
    return s/n
# Q6
"""1pixel = 3 couleurs * 1o/couleur = 3o
1 image de 48Mpix --> 144Mo"""
# Q7
def Clinear(C):
    C1 = C/255
    if C1 <= 0.04045:
        Clin = C1/12.92
    else :
        Clin = ((C1+0.055)/1.055 )**2.4
    return Clin
# Q8
def Intensite(pix):
    Ylin = 0.2126*Clinear(pix[0])+0.7152*Clinear(pix[1])+0.0722*Clinear(pix[2])
    if Ylin <= 0.0031308:
        Y = int(255*12.92*Ylin)
    else :
        Y=int(255*( 1.055*Ylin**(1/2.4)-0.055) )
    return Y
# Q9
def init(h,w):
    I = []
    for i in range(h):
        I.append([0]*w)
    return I
# Q10
def Niveau_Gris(I):
    nl = len(I)
    nc = len(I[0])
```

```
    Ig = init(nl,nc)
    for i in range(nl):
        for j in range(nc):
            Ig[i][j] = Intensite(I[i][j])
    return Ig
# Q11
"""Complexité de Niveau_Gris en o(h*w) car Intensite en o(1)."""
# Q12
"""La moyenne de luminance est calculé sur un tiers des lignes len(image) et des colonnes
len(image[0]) de l'image. """
# Q13
def coef_integral(M,x,y):
    s=0
    for j in range(x+1):
        for k in range(y+1):
            s = s + M[j][k]
    return s
# Q14
n=len(M)
N=len(M[0])
M_int=init(n,N)
for i in range(n):
    for j in range(N):
        M_int[i][j] = coef_integral(M,i,j)
# Q15                                                    15h
def lumi_section(M_int,x,y,n,p):
    return (M_int[x+n][y+p]-M_int[x+n][y]-M_int[x][y+p]+M_int[x][y])/(n*p)
# Q16
def detection(M_int,n,p,C_ref,seuil):
    Ls=[]
    nl=len(M_int)
    nc=len(M_int[0])
    i,j = 0,0
    while (i+1)*n <nl and (j+1)*p<nc:
        mg = lumi_section(M_int,i*n,j*p,n,p//2)
        md = lumi_section(M_int,i*n,j*p+p//2,n,p//2)
        C_obs = max(md,mg)-min(md,mg)
        if C_obs/C_ref > seuil :
            Ls.append([(i+1)*n,(j+1)*p])
        i = i + 1
        j = j + 1
    return Ls
# Q17
def min_max(X) :
    mini = X[0]
    maxi = X[0]
    n = len(X)
    for i in range(n):
        if maxi < X[i]:
            maxi = X[i]
        elif X[i] < mini :
            mini = X[i]
    return (mini, maxi)
# Q18
def coeff_normalise(data) :
    data_nom = data[:,:]
    n = len(data)
    N = len(data[0])
    for i in range(n):
        for j in range(N):
            m,M = min_max(data[i])
```

```
            if M!=m:
                data_nom[i,j] = (data[i,j]-m)/(M-m)
    return data_nom
# Q19
def distance(Z,data):
    N = len(Z)
    n=len(data)
    d = np.zeros(n,float)
    for i in range(n):
        s = 0
        for j in range(N):
            s = s + (Z[j]-data[i,j])**2
        d[i] = s ** 0.5
    return d
# Q20
def tri (T):
    if len(T) <= 1:
        return T
    else:
        m= len(T) //2
        L1 = []
        for x in range(m):
            L1.append(T[x])
        L2 = []
        for x in range(m, len(T)):
                L2.append(T[x])
        return fusion ( tri (L1) , tri (L2))
def fusion (T1,T2):
    if T1 == []:
        return T2
    if T2 == []:
        return T1
    if T1[0][0] < T2[0][0]:
        return [T1[0]]+fusion (T1[1:] ,T2)
    else:
        return [T2[0]]+fusion (T1 ,T2[1:])
# Q21
"""C(n) = o(n) + c(n-1) = o(n²) plus lent qu'un tri fusion logarithmique en
o(n.log(n))"""
# Q22
"""1) calcul des distances, entre z et les autres points, classées par ordre croissant
2) ajout dans select du nombre d'apparitions respectifs des nb types pour les K plus
proches voisins de z
3) détermination du type dont le nombre d'apparition dans select est le plus grand"""
# Q23                                                    17h
"""Matrice de confusion
Diagonale = nombre de prédiction correcte
Sur la première ligne des 'adultes' l'algorithme a identifié par erreur 4 enfants, 3
animaux, 5 véhicules, 2 indéterminés
Taux de réussite = (27+18+40+36+19)/200 = 140/200 = 70%"""
# Q24
La requête liste les jours distincts où le bus 3 a été en panne.
# Q25
SELECT Date_jour, COUNT(*) FROM Passagers WHERE numero_bus == 1 GROUP BY Date_jour;
# Q26
SELECT date_jour, numero_bus, capteurs
FROM Acquisition JOIN Passagers ON id_bus == id1
WHERE defaillance == 'Perte de signal';
# Q27
SELECT id_bus, SUM(reparations) AS total FROM Acquisition
HAVING total > 10 ORDER BY total DESC                                    17h20
```