```
# -*- coding: utf-8 -*-
"""
Created on Fri Jul  3 16:02:40 2015

@author: arno
"""

# non demandé
def demande_valeur(text):
    x=int(input(text))
    return x


# Q1.
def demande_fenetre():
    Px=demande_valeur('Valeur de Px : ')
    Py=demande_valeur('Valeur de Py : ')
    L=demande_valeur('Valeur de L : ')
    H=demande_valeur('Valeur de H : ')
    return [Px,Py,L,H]

# Q2. fenetre=demande_fenetre()
def verification_fenetre(image_width,image_height,fenetre):
    [Px,Py,L,H]=fenetre
    return 0<=Px and Px+L<=image_width and 0<=Py and Py+H<=image_fenetre

# Q3. SELECT filename FROM instruments JOIN definitions ON instruments.id=defini
tions.mid WHERE name='pince';

# Q4.
def centre(fenetre):
    [Px,Py,L,H]=fenetre
    return [(2*Px+L)//2,(2*Py+H)//2]

# Q5. à chaque pixel est associé un entier entre 0 et 255. Or 2**8=256. Donc cha
que pixel est codé sur 8 bits=1 octet.
# Au total pour l'image: 3*m*n=1 440 000 octets
# soit en gros 1,37 Méga-octets

# Q6.
def grayscale(imagecolor):
    [m,n]=imagecolor.shape[1:3]
    image=[[0 for x in range (m)] for y in range(n)] # avec numpy: image=zeros((
m,n))
    for i in range(m):
        for j in range(n):
            rgb=[imagecolor[0][i][j],imagecolor[1][i][j],imagecolor[2][i][j]]
            image[i][j]=(max(rgb)+min(rgb))//2
    return image

# Q7.
def construction_coordonnees_pts(fenetre,numM,numN):
    [Px,Py,L,H]=fenetre
    pts=[]
    for x in arange(Px,Px+L+1,L//(numM-1)):
        for y in arange(Py,Py+H+1,H//(numN-1)):
            pts.append(x) # x est le même pendant l'exécution de la boucle for y

            pts.append(y)
    return pts

# Q8. Ix = imgI[ux+1,uy] - imgI[ux,uy]
```



```
#     Iy = imgI[ux,uy+1] - imgI[ux,uy]
#     It = ( imgJ[ux,uy] - imgU[ux,uy] ) / float(dt)  en Python 2.7

# Q9.
def creation_patch(P,patch_size): #P=[Px,Py]
    pts=[]
    for x in range(patch_size):
        for y in range(patch_size):
            pts.append(P[0]+x-(patch_size-1)//2) # x est le même pendant l'exécu
tion de la boucle for y
            pts.append(P[1]+y-(patch_size-1)//2)
    return pts

# Q10.
def calc_Ab(imgI,imgJ,patch): #patch_size=sqrt(len(patch)/2)
    A=zeros((len(patch)//2,2))
    b=zeros((len(patch)//2,1))
    for k in len(patch)//2:
        [Ix,Iy,Itdt]=calc_LK_terms([patch[2*k],patch[2*k+1]],imgI,imgJ)
        A[k,0]=Ix
        A[k,1]=Iy
        b[k,0]=-Itdt
    return A,b

# Q11.
def resoud_LK(A,b):
    M=dots(transpose(A),A) #système de Cramer MX=B de taille 2x2
    B=dots(transpose(A),b)
    # algorithme du pivot de Gauss-Jordan
    # recherche d'un pivot sur la colonne 0
    if M[0,0]==0:
        M[1,:],M[0,:] = M[0,:],M[1,:]
    # on effectue une transvection sur la ligne 1
    M[1,:]=M[1,:] - (M[1,0]/float(M[0,0]))*M[0,:] # Python 2.7
    B[1,:]=B[1,:] - (M[1,0]/float(M[0,0]))*B[0,:]
    # calcul de dy puis dx
    dy=B[1,0]/float(M[1,1])
    dx=(B[0,0]-M[0,1]*dy)/float(M[0,0])
    return [dx,dy]

# Q12.
def recherche_points(imgI,imgJ,pts): # pts=construction_coordonnees_pts(fenetre,
numM,numN)
    fpts=[]
    for k in len(pts)//2:
        P=[pts[2*k],pts[2*k+1]]
        patch=creation_patch(P,patch_size) # patch_size=5
        [A,b]=calc_Ab(imgI,imgJ,patch)
        [dx,dy]=resoud_LK(A,b)
        fpts.append(pts[2*k]+int(dx))
        fpts.append(pts[2*k+1]+int(dy))
    return fpts

# Q13. On n'a pas pris en compte le cas où le patch sort de l'image et le cas ou
 transpose(A)*A n'est pas inversible.

# Q14.
fpts=recherche_points(imgI,imgJ,pts)
ffpt=recherche_points(imgJ,imgI,pts)
statut=[]
for k in range(len(ftps)):
    statut.append(fpts[k]==ffpts[k])
```

```
# Q15. la notation A[i:j:k] n'a pas été redéfinie: A[start:stop:step]
# points[0::2] donne les abscisses (indices pairs) et points[1::2] les ordonnées
 (indices impairs)
# on calcule la distance entre un point de référence dans l'image précédente et
dans l'image courante, ceci pour chaque point de référence
# on donne le résultat dans un tableau numpy de flottants de taille 1x(numM*numN
)

# Q16.
def tri_rapide(t):
    if len(t)<=1: # close d'arrêt: tableau vide
        return t  # ou avec un seul élément
         # cette fonction renverra None en sortie
    else:
        pivot=t[0] # choix arbitraire du pivot
        plus_petits=[ x for x in t if x<pivot ]
        plus_grands=[ x for x in t[1:] if x>=pivot ]
        #tri_rapide(plus_petits) # trie la liste plus_petits et renvoie None
        #tri_rapide(plus_grands) # trie la liste plus_grands et renvoie None
        return tri_rapide(plus_petits)+[pivot]+tri_rapide(plus_grands) # concaté
nation et renvoi du résultat

def mediane(a):
    tri_rapide(a)
    n=len(a)
    if n%2==1:
        return a[n//2]
    else:
        return (a[n//2]+a[(n-1)//2])/2.

# Q17. Algorithme de tri rapide. O(n**2) dans le pire des cas et O(n.log(n)) dan
s le meilleur des cas

# Q18.
def verification_pts(pts,fpts,statut):
    nouveaux_pts=[]
    n=len(pts)//2
    critere2=calcul1(pts,fpts)
    med=mediane(res)
    for k in range(n):
        if statut[k]==True and critere2[k]<=med:
            nouveaux_pts.append(pts[2*k])
            nouveaux_pts.append(pts[2*k+1])
    return nouveaux_pts

# Q19.
n=len(pts)//2
critere3=[]
for k in range(n):
    result=cv2.matchTemplate(imgJ,creation_patch([pts[2*k],pts[2*k+1]],patch_siz
e),CV_TM_CCORR_NORMED)
    critere3.append(result)

# Q20. Argh.
```