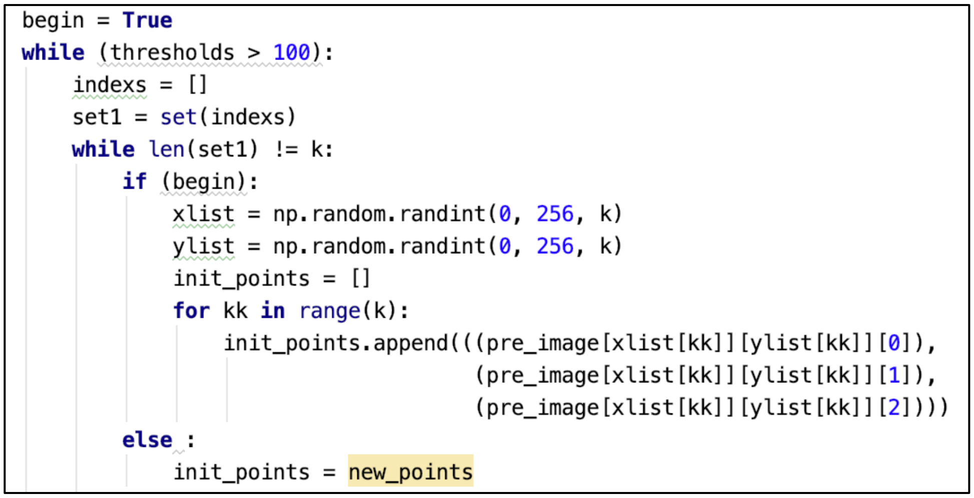
# CS 558 HW2:

## Problem 1 :

1. Randomly pick 10 RGB triplets from the existing pixels as initial seeds.

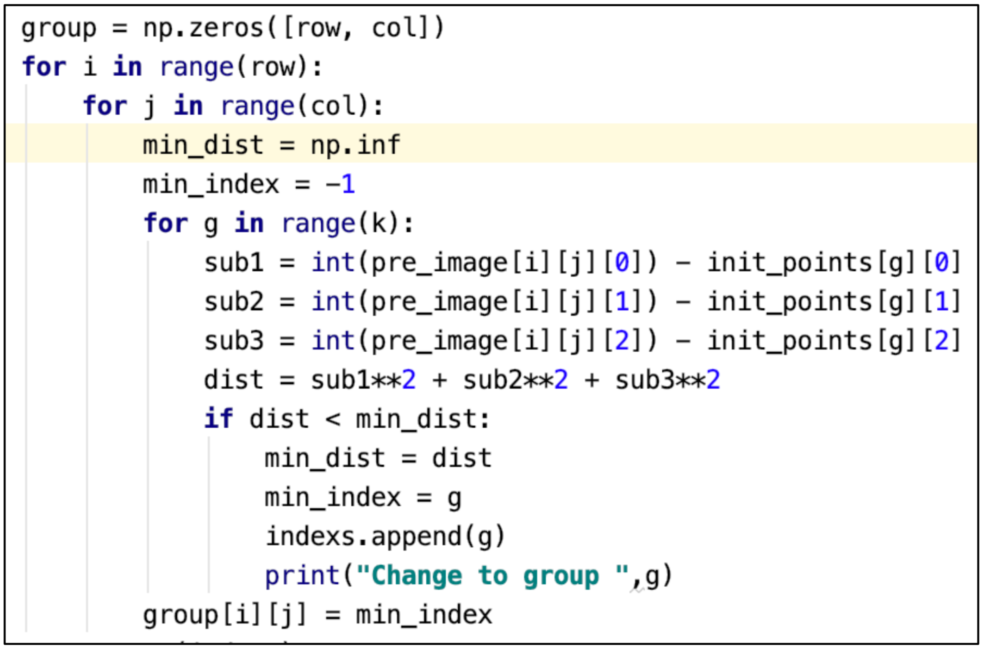
Code:



```
begin = True
while (thresholds > 100):
    indexs = []
    set1 = set(indexs)
    while len(set1) != k:
        if (begin):
            xlist = np.random.randint(0, 256, k)
            ylist = np.random.randint(0, 256, k)
            init_points = []
            for kk in range(k):
                init_points.append(((pre_image[xlist[kk]][ylist[kk]][0]),
                                    (pre_image[xlist[kk]][ylist[kk]][1]),
                                    (pre_image[xlist[kk]][ylist[kk]][2])))
        else :
            init_points = new_points
```

2. The minimum distance is calculated according to k-mean and the cluster is allocated.

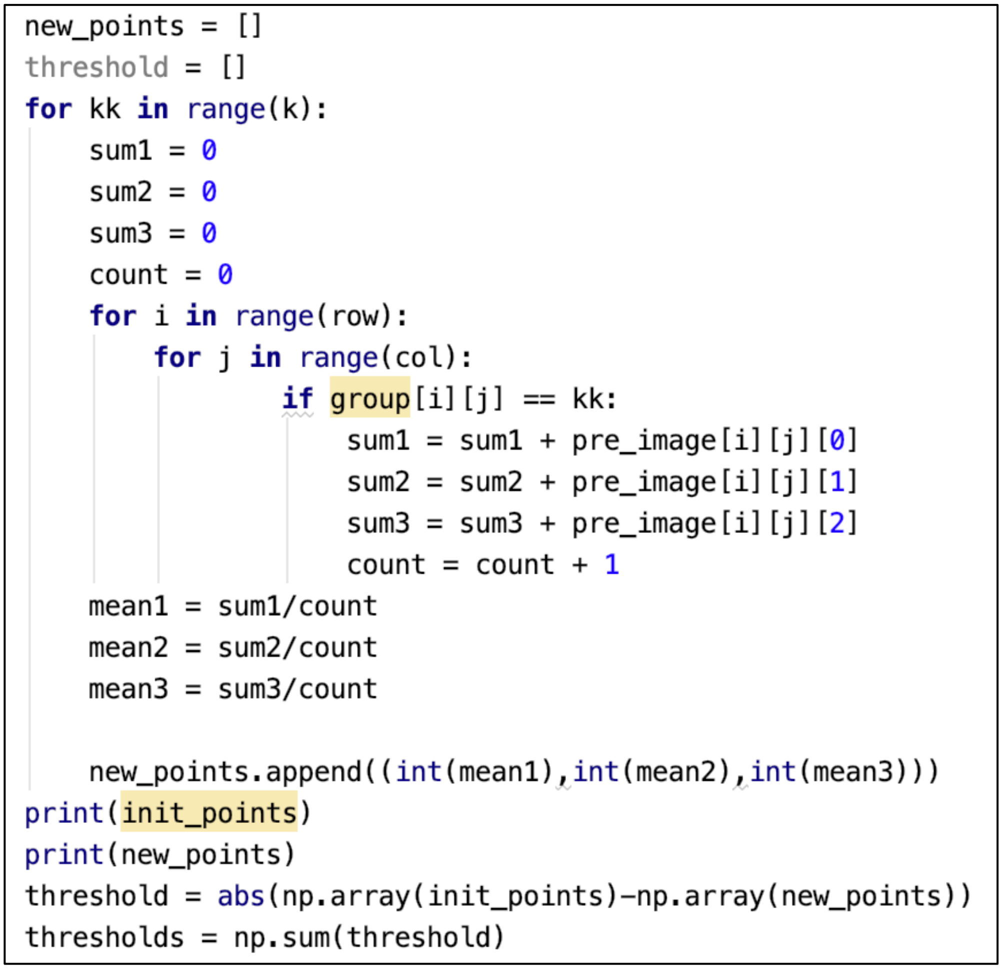
Code:



```
group = np.zeros([row, col])
for i in range(row):
    for j in range(col):
        min_dist = np.inf
        min_index = -1
        for g in range(k):
            sub1 = int(pre_image[i][j][0]) - init_points[g][0]
            sub2 = int(pre_image[i][j][1]) - init_points[g][1]
            sub3 = int(pre_image[i][j][2]) - init_points[g][2]
            dist = sub1**2 + sub2**2 + sub3**2
            if dist < min_dist:
                min_dist = dist
                min_index = g
                indexs.append(g)
                print("Change to group ",g)
        group[i][j] = min_index
```

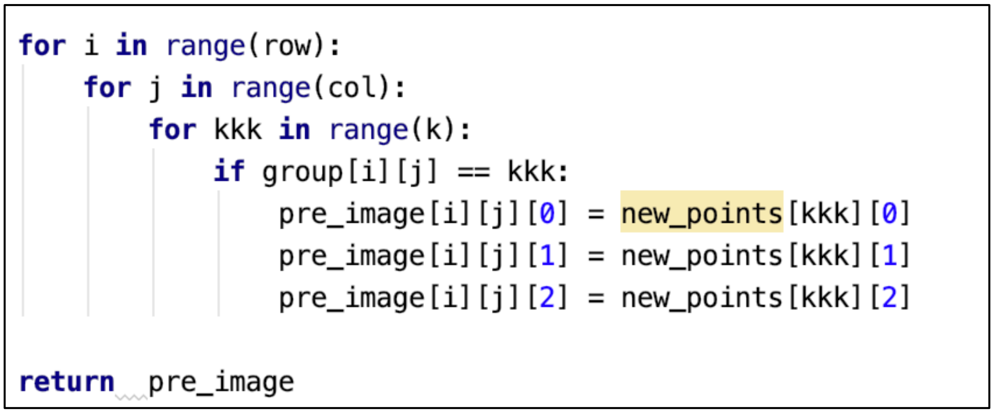
3. Calculate the mean of all the elements in each cluster to get the center of cluster, use it as the new starting point. Go back to the second step and cycle again until the change of center of mass is less than the threshold. (this threshold is defined as threshold here)

Code:



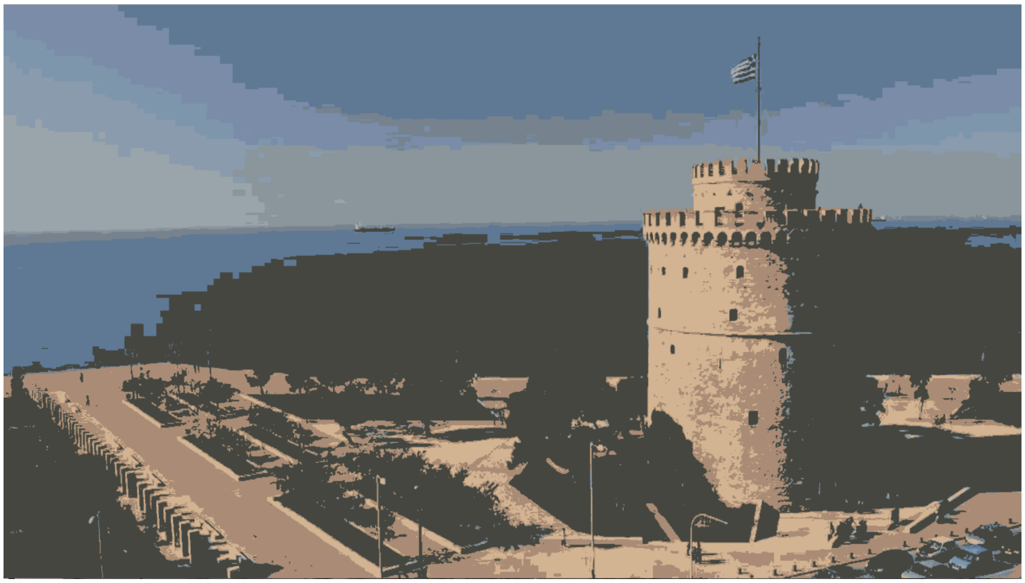
```
new_points = []
threshold = []
for kk in range(k):
    sum1 = 0
    sum2 = 0
    sum3 = 0
    count = 0
    for i in range(row):
        for j in range(col):
                if group[i][j] == kk:
                    sum1 = sum1 + pre_image[i][j][0]
                    sum2 = sum2 + pre_image[i][j][1]
                    sum3 = sum3 + pre_image[i][j][2]
                    count = count + 1
    mean1 = sum1/count
    mean2 = sum2/count
    mean3 = sum3/count

    new_points.append((int(mean1),int(mean2),int(mean3)))
print(init_points)
print(new_points)
threshold = abs(np.array(init_points)-np.array(new_points))
thresholds = np.sum(threshold)
```

4. Represent each cluster with the average RGB value of its members.

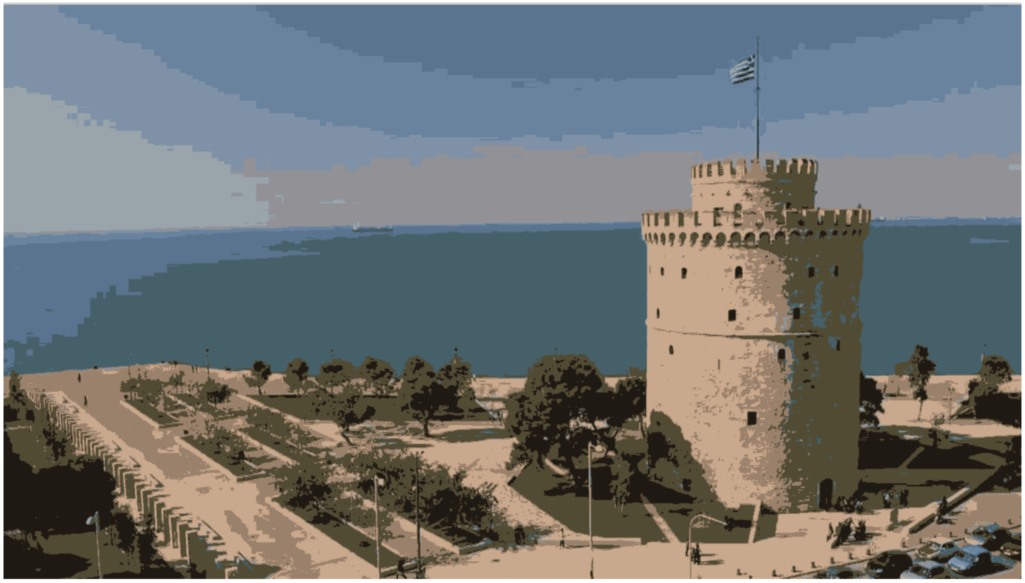


```
for i in range(row):
    for j in range(col):
        for kkk in range(k):
            if group[i][j] == kkk:
                pre_image[i][j][0] = new_points[kkk][0]
                pre_image[i][j][1] = new_points[kkk][1]
                pre_image[i][j][2] = new_points[kkk][2]

return  pre_image
```

Result 1: (threshold = 100)



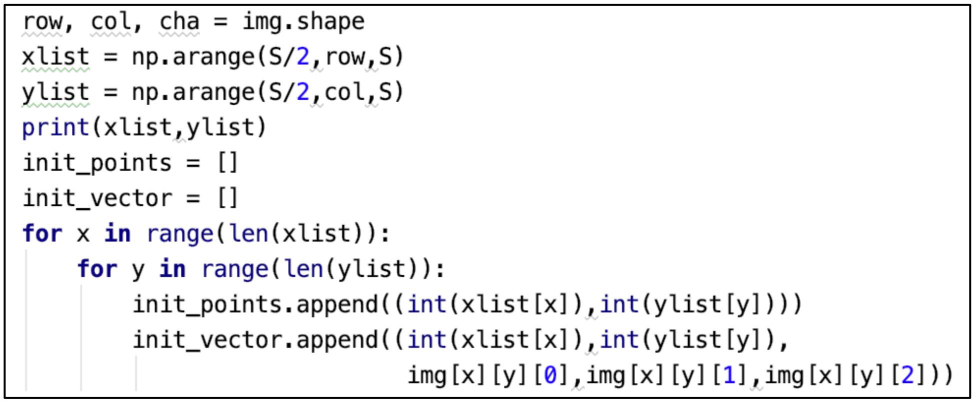
Result 2: (threshold = 10)



**Problem 2:**

1. Divide the image in blocks of 50×50 pixels and initialize a centroid at the center of each block. (S = 50)

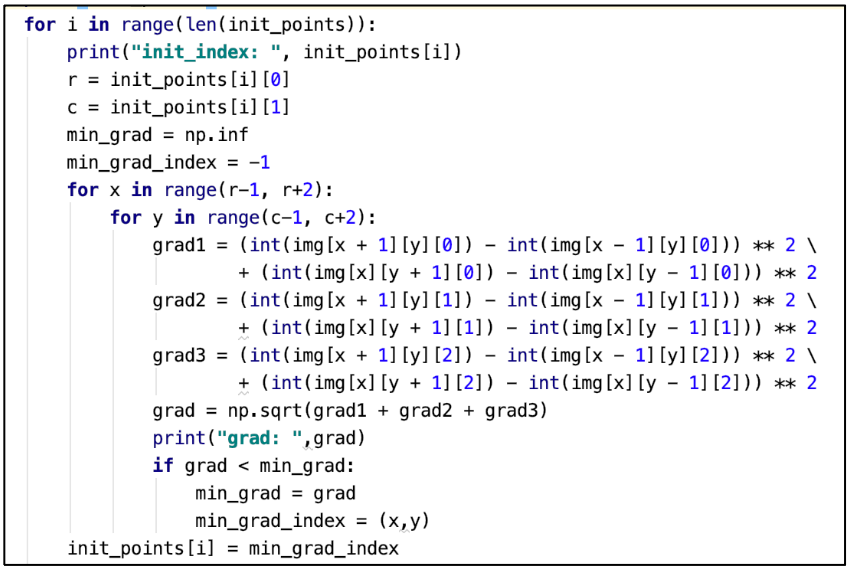
Code:



```
row, col, cha = img.shape
xlist = np.arange(S/2,row,S)
ylist = np.arange(S/2,col,S)
print(xlist,ylist)
init_points = []
init_vector = []
for x in range(len(xlist)):
    for y in range(len(ylist)):
        init_points.append((int(xlist[x]),int(ylist[y])))
        init_vector.append((int(xlist[x]),int(ylist[y]),
                                        img[x][y][0],img[x][y][1],img[x][y][2]))
```

2. Compute the magnitude of the gradient in each of the RGB channels and use the square root of the sum of squares of the three magnitudes as the combined gradient magnitude. Move the centroids to the position with the smallest gradient magnitude in 3×3 windows centered on the initial centroids.

Code:

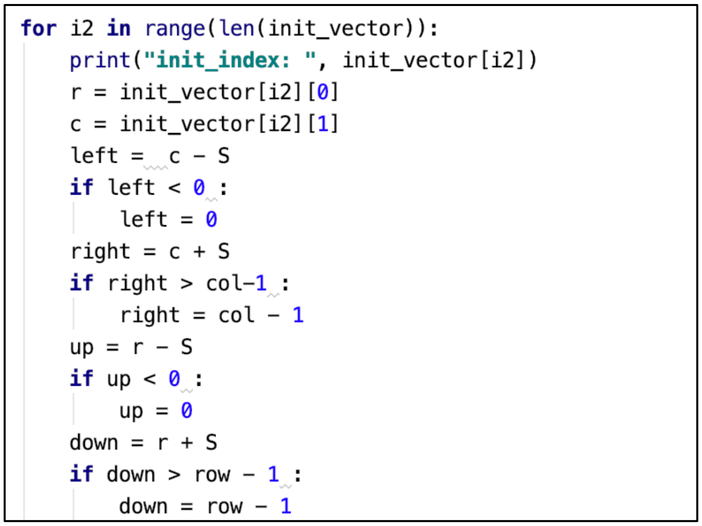


```
for i in range(len(init_points)):
    print("init_index: ", init_points[i])
    r = init_points[i][0]
    c = init_points[i][1]
    min_grad = np.inf
    min_grad_index = -1
    for x in range(r-1, r+2):
        for y in range(c-1, c+2):
            grad1 = (int(img[x + 1][y][0]) - int(img[x - 1][y][0])) ** 2 \
                    + (int(img[x][y + 1][0]) - int(img[x][y - 1][0])) ** 2
            grad2 = (int(img[x + 1][y][1]) - int(img[x - 1][y][1])) ** 2 \
                    + (int(img[x][y + 1][1]) - int(img[x][y - 1][1])) ** 2
            grad3 = (int(img[x + 1][y][2]) - int(img[x - 1][y][2])) ** 2 \
                    + (int(img[x][y + 1][2]) - int(img[x][y - 1][2])) ** 2
            grad = np.sqrt(grad1 + grad2 + grad3)
            print("grad: ",grad)
            if grad < min_grad:
                min_grad = grad
                min_grad_index = (x,y)
    init_points[i] = min_grad_index
```

3. Apply k-means in the 5D space of x, y, R, G, B. Use the Euclidean distance in this space, but divide x and y by 2.

Find the center of all pixels in the 2S range class：

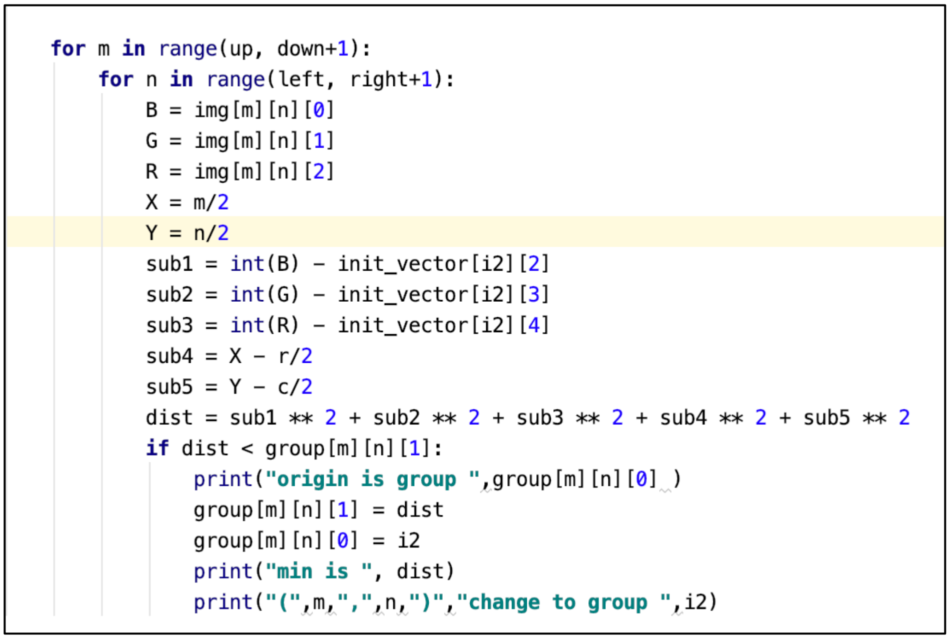
Code：



```
for i2 in range(len(init_vector)):
    print("init_index: ", init_vector[i2])
    r = init_vector[i2][0]
    c = init_vector[i2][1]
    left =  c - S
    if left < 0 :
        left = 0
    right = c + S
    if right > col-1 :
        right = col - 1
    up = r - S
    if up < 0 :
        up = 0
    down = r + S
    if down > row - 1 :
        down = row - 1
```

Calculate the distance from all the points in the 2S range class to the initial center point and assign it to the nearest center point cluster: （divide x and y by 2 ）

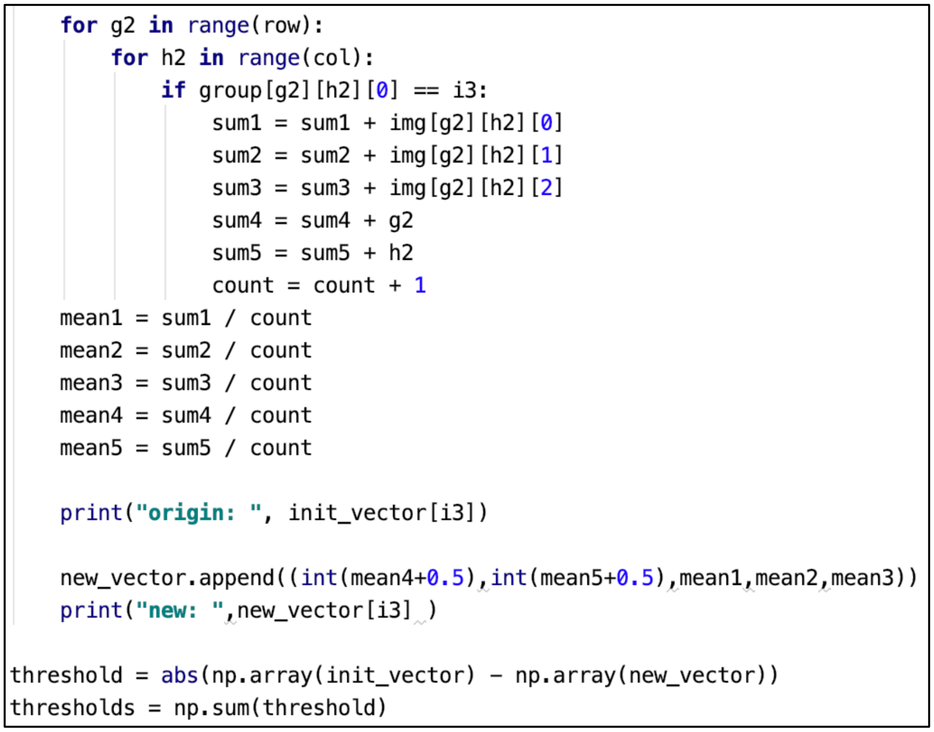
Code：



```
for m in range(up, down+1):
    for n in range(left, right+1):
        B = img[m][n][0]
        G = img[m][n][1]
        R = img[m][n][2]
        X = m/2
        Y = n/2
        sub1 = int(B) - init_vector[i2][2]
        sub2 = int(G) - init_vector[i2][3]
        sub3 = int(R) - init_vector[i2][4]
        sub4 = X - r/2
        sub5 = Y - c/2
        dist = sub1 ** 2 + sub2 ** 2 + sub3 ** 2 + sub4 ** 2 + sub5 ** 2
        if dist < group[m][n][1]:
            print("origin is group ",group[m][n][0] )
            group[m][n][1] = dist
            group[m][n][0] = i2
            print("min is ", dist)
            print("(",m,",",n,")","change to group ",i2)
```

Calculate the average vector of five vectors of each cluster, set the average coordinate as the new center of gravity, and cycle again until the change value of the center of gravity is less than the threshold.

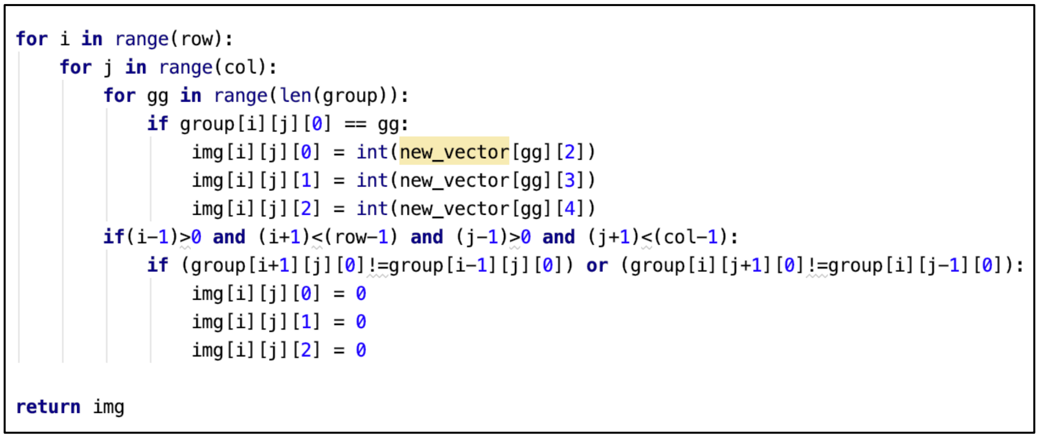
Code:



```
    for g2 in range(row):
        for h2 in range(col):
            if group[g2][h2][0] == i3:
                sum1 = sum1 + img[g2][h2][0]
                sum2 = sum2 + img[g2][h2][1]
                sum3 = sum3 + img[g2][h2][2]
                sum4 = sum4 + g2
                sum5 = sum5 + h2
                count = count + 1
    mean1 = sum1 / count
    mean2 = sum2 / count
    mean3 = sum3 / count
    mean4 = sum4 / count
    mean5 = sum5 / count

    print("origin: ", init_vector[i3])

    new_vector.append((int(mean4+0.5),int(mean5+0.5),mean1,mean2,mean3))
    print("new: ",new_vector[i3] )

threshold = abs(np.array(init_vector) - np.array(new_vector))
thresholds = np.sum(threshold)
```

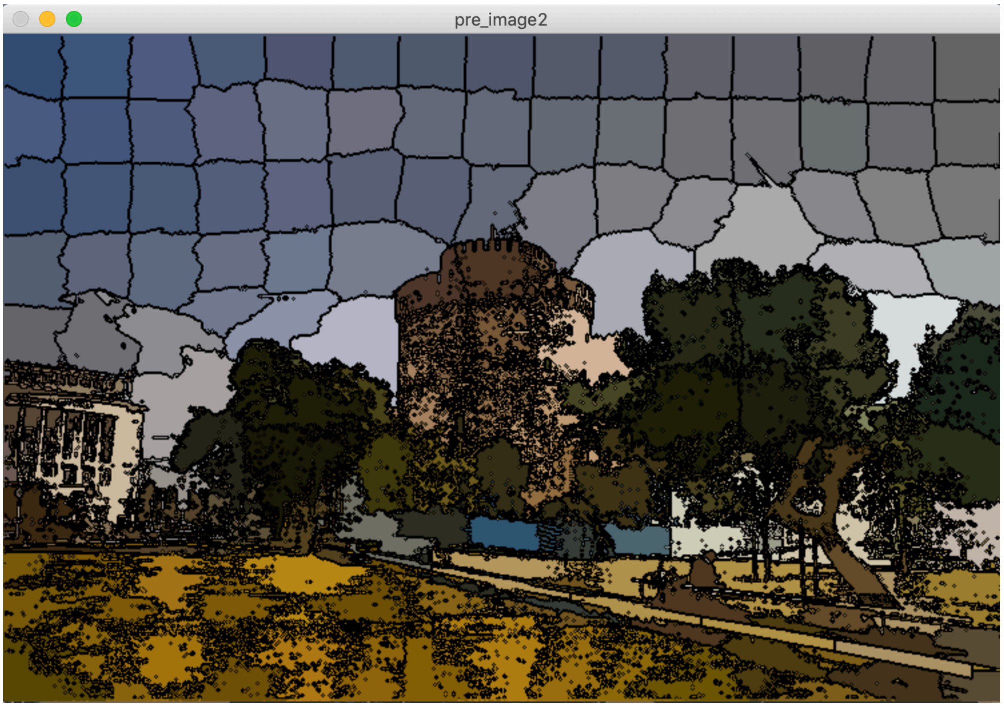
4. After convergence, display the output image, color pixels that touch two different clusters black and the remaining pixels by the average RGB value of their cluster.

Code:

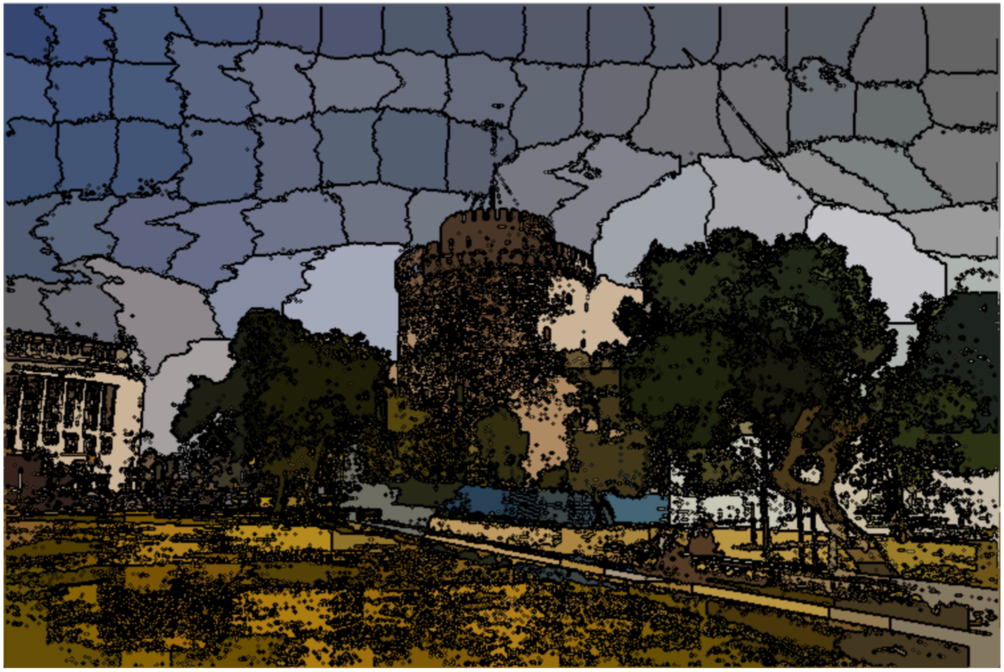


```
for i in range(row):
    for j in range(col):
        for gg in range(len(group)):
            if group[i][j][0] == gg:
                img[i][j][0] = int(new_vector[gg][2])
                img[i][j][1] = int(new_vector[gg][3])
                img[i][j][2] = int(new_vector[gg][4])
        if(i-1)>0 and (i+1)<(row-1) and (j-1)>0 and (j+1)<(col-1):
            if (group[i+1][j][0]!=group[i-1][j][0]) or (group[i][j+1][0]!=group[i][j-1][0]):
                img[i][j][0] = 0
                img[i][j][1] = 0
                img[i][j][2] = 0

return img
```

Result 1: (X and Y are not divided by 2. The threshold is 100)



Result 2: (X and Y divided by 2. The threshold is 100)



The source code:

```
import cv2
import random
import copy
import math
import numpy as np

def do_kmean(pre_image, k):
    row, col, cha = pre_image.shape
    thresholds = np.inf
    begin = True
    while (thresholds > 100):
        indexs = []
        set1 = set(indexs)
        while len(set1) != k:
            if (begin):
                xlist = np.random.randint(0, 256, k)
                ylist = np.random.randint(0, 256, k)
                init_points = []
                for kk in range(k):

init_points.append(((pre_image[xlist[kk]][ylist[kk]][0]),(pre_image[xlist[kk]][
ylist[kk]][1]),(pre_image[xlist[kk]][ylist[kk]][2])))
            else :
                init_points = new_points
            group = np.zeros([row, col])
            for i in range(row):
                for j in range(col):
                    min_dist = np.inf
                    min_index = -1
                    for g in range(k):
                        sub1 = int(pre_image[i][j][0]) - init_points[g][0]
                        sub2 = int(pre_image[i][j][1]) - init_points[g][1]
                        sub3 = int(pre_image[i][j][2]) - init_points[g][2]
                        dist = sub1**2 + sub2**2 + sub3**2
                        if dist < min_dist:
                            min_dist = dist
                            min_index = g
                            indexs.append(g)
                            print("Change to group ",g)
                    group[i][j] = min_index
            set1 = set(indexs)
            print(len(set1))
            if (len(set1)==k) :
                begin = False
        new_points = []
        threshold = []
        for kk in range(k):
            sum1 = 0
            sum2 = 0
            sum3 = 0
            count = 0
            for i in range(row):
                for j in range(col):
                        if group[i][j] == kk:
                            sum1 = sum1 + pre_image[i][j][0]
                            sum2 = sum2 + pre_image[i][j][1]
                            sum3 = sum3 + pre_image[i][j][2]
                            count = count + 1
            mean1 = sum1/count
            mean2 = sum2/count
            mean3 = sum3/count

            new_points.append((int(mean1),int(mean2),int(mean3)))
        print(init_points)
        print(new_points)
        threshold = abs(np.array(init_points)-np.array(new_points))
        thresholds = np.sum(threshold)
        print(thresholds)

    for i in range(row):
        for j in range(col):
            for kkk in range(k):
                if group[i][j] == kkk:
                    pre_image[i][j][0] = new_points[kkk][0]
                    pre_image[i][j][1] = new_points[kkk][1]
                    pre_image[i][j][2] = new_points[kkk][2]

    return  pre_image

def do_SLIC(pre_image, S):
    img = pre_image
    row, col, cha = img.shape
    xlist = np.arange(S/2,row,S)
    ylist = np.arange(S/2,col,S)
    print(xlist,ylist)
    init_points = []
    init_vector = []
    for x in range(len(xlist)):
        for y in range(len(ylist)):
            init_points.append((int(xlist[x]),int(ylist[y])))

init_vector.append((int(xlist[x]),int(ylist[y]),img[x][y][0],img[x][y][1],img[x
][y][2]))
    print(init_points)
    for i in range(len(init_points)):
        print("init_index: ", init_points[i])
        r = init_points[i][0]
        c = init_points[i][1]
        min_grad = np.inf
        min_grad_index = -1
        for x in range(r-1, r+2):
            for y in range(c-1, c+2):
                grad1 = (int(img[x + 1][y][0]) - int(img[x - 1][y][0])) ** 2 +
(int(img[x][y + 1][0]) - int(img[x][y - 1][0])) ** 2
                grad2 = (int(img[x + 1][y][1]) - int(img[x - 1][y][1])) ** 2 +
(int(img[x][y + 1][1]) - int(img[x][y - 1][1])) ** 2
                grad3 = (int(img[x + 1][y][2]) - int(img[x - 1][y][2])) ** 2 +
(int(img[x][y + 1][2]) - int(img[x][y - 1][2])) ** 2
                grad = np.sqrt(grad1 + grad2 + grad3)
                print("grad: ",grad)
                if grad < min_grad:
                    min_grad = grad
                    min_grad_index = (x,y)
        init_points[i] = min_grad_index
        print("min: ",min_grad)
        print("min_index: ",min_grad_index)

    group = np.zeros([row, col, 2])
    min_dist = np.inf
    min_index = -1
    for g in range(row):
```

```
    for h in range(col):
        group[g][h][0] = min_index
        group[g][h][1] = min_dist

thresholds = np.inf
begin = True
while thresholds > 100:
    if begin == False:
        init_vector = new_vector

    begin = False

    for i2 in range(len(init_vector)):
        print("init_index: ", init_vector[i2])
        r = init_vector[i2][0]
        c = init_vector[i2][1]
        left =  c - S
        if left < 0 :
            left = 0
        right = c + S
        if right > col-1 :
            right = col - 1
        up = r - S
        if up < 0 :
            up = 0
        down = r + S
        if down > row - 1 :
            down = row - 1

        for m in range(up, down+1):
            for n in range(left, right+1):
                B = img[m][n][0]
                G = img[m][n][1]
                R = img[m][n][2]
                X = m/2
                Y = n/2
                sub1 = int(B) - init_vector[i2][2]
                sub2 = int(G) - init_vector[i2][3]
                sub3 = int(R) - init_vector[i2][4]
                sub4 = X - r/2
                sub5 = Y - c/2
```

```
                    dist = sub1 ** 2 + sub2 ** 2 + sub3 ** 2 + sub4 ** 2 + sub5 **
2
                    if dist < group[m][n][1]:
                        print("origin is group ",group[m][n][0] )
                        group[m][n][1] = dist
                        group[m][n][0] = i2
                        print("min is ", dist)
                        print("(",m,",",n,")","change to group ",i2)

        new_vector = []
        for i3 in range(len(init_vector)):
            sum1 = 0
            sum2 = 0
            sum3 = 0
            sum4 = 0
            sum5 = 0
            count = 0
            for g2 in range(row):
                for h2 in range(col):
                    if group[g2][h2][0] == i3:
                        sum1 = sum1 + img[g2][h2][0]
                        sum2 = sum2 + img[g2][h2][1]
                        sum3 = sum3 + img[g2][h2][2]
                        sum4 = sum4 + g2
                        sum5 = sum5 + h2
                        count = count + 1
            mean1 = sum1 / count
            mean2 = sum2 / count
            mean3 = sum3 / count
            mean4 = sum4 / count
            mean5 = sum5 / count

            print("origin: ", init_vector[i3])

            new_vector.append((int(mean4+0.5),int(mean5+0.5),mean1,mean2,mean3))
            print("new: ",new_vector[i3] )

        threshold = abs(np.array(init_vector) - np.array(new_vector))
        thresholds = np.sum(threshold)
        print("thresholds: ",thresholds)
```

```
    for i in range(row):
        for j in range(col):
            for gg in range(len(group)):
                if group[i][j][0] == gg:
                    img[i][j][0] = int(new_vector[gg][2])
                    img[i][j][1] = int(new_vector[gg][3])
                    img[i][j][2] = int(new_vector[gg][4])
            if(i-1)>0 and (i+1)<(row-1) and (j-1)>0 and (j+1)<(col-1):
                if (group[i+1][j][0]!=group[i-1][j][0]) or
(group[i][j+1][0]!=group[i][j-1][0]):
                    img[i][j][0] = 0
                    img[i][j][1] = 0
                    img[i][j][2] = 0

    return img


''' Load the image '''
pre_image = cv2.imread("white-tower.png")
row, col, cha = pre_image.shape
print("pre_image:",pre_image.shape)
pre_image = do_kmean(pre_image,10)

pre_image2 = cv2.imread("wt_slic.png")
row2, col2, cha2 = pre_image2.shape
print("pre_image2:",pre_image2.shape)
pre_image2 = do_SLIC(pre_image2,50)

cv2.imshow("pre_image", pre_image)
cv2.imshow("pre_image2", pre_image2)
cv2.waitKey()
cv2.destroyAllWindows()
```