**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**

**Subject Name**

**Digital Image Processing**

**Lab Number**

**1**

**SUBMITTED TO:**

**Dr. Asad Khan**

**LE Sundas Ashraf**

**SUBMITTED BY:**

**Student Name**

1. Wahaaj Nasir

**Reg#413238**

**DE- 44 Dept C&SE**

**Objectives:**

Basics of Image Processing in Python

**Related Topic/Chapter in theory class:**

Basics Of Digital Image Processing

**Hardware/Software required:**

Hardware: PC

Software Tool: Pycharm

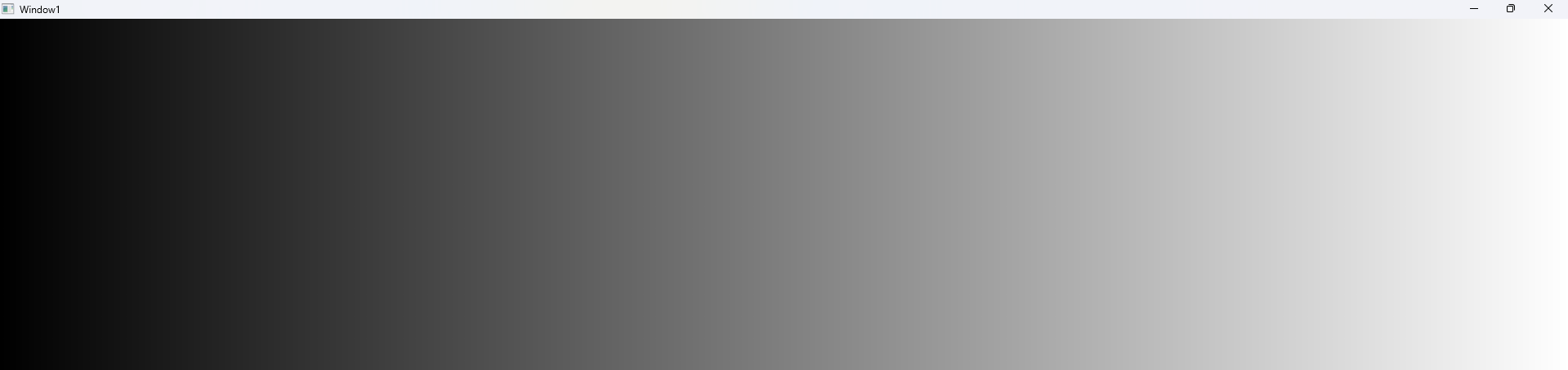
**Task 1:**

**Read a grayscale image (given below) and convert the image to 16 levels, then to 4 levels and finally to 1. Display all four images.**

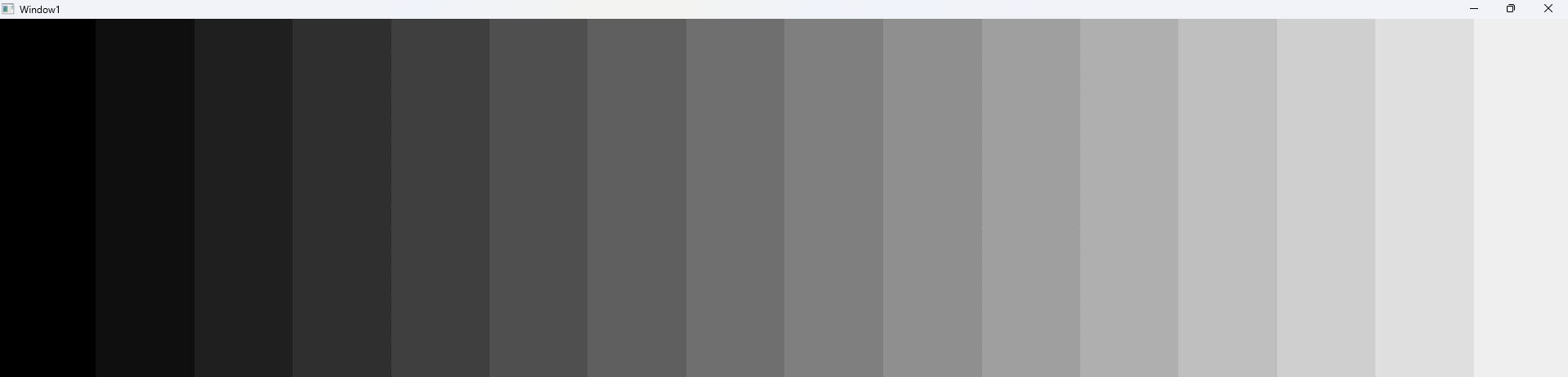
**Solution:**

import numpy as np  
import cv2 as cv  
  
def lower\_by\_16(image):  
 rows, cols = image.shape  
 new\_image = np.ones((rows, cols), dtype = np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 if(image[i, j] >= 0 and image[i, j] <= 15):  
 new\_image[i, j] = 0  
 elif(image[i, j] >= 16 and image[i, j] <= 31):  
 new\_image[i, j] = 15  
 elif(image[i, j] >= 32 and image[i, j] <= 47):  
 new\_image[i, j] = 31  
 elif(image[i, j] >= 48 and image[i, j] <= 63):  
 new\_image[i, j] = 47  
 elif(image[i, j] >= 64 and image[i, j] <= 79):  
 new\_image[i, j] = 63  
 elif(image[i, j] >= 80 and image[i, j] <= 95):  
 new\_image[i, j] = 79  
 elif(image[i, j] >= 96 and image[i, j] <= 111):  
 new\_image[i, j] = 95  
 elif(image[i, j] >= 112 and image[i, j] <= 127):  
 new\_image[i, j] = 111  
 elif(image[i, j] >= 128 and image[i, j] <= 143):  
 new\_image[i, j] = 127  
 elif(image[i, j] >= 144 and image[i, j] <= 159):  
 new\_image[i, j] = 143  
 elif(image[i, j] >= 160 and image[i, j] <= 175):  
 new\_image[i, j] = 159  
 elif(image[i, j] >= 176 and image[i, j] <= 191):  
 new\_image[i, j] = 175  
 elif(image[i, j] >= 192 and image[i, j] <= 207):  
 new\_image[i, j] = 191  
 elif(image[i, j] >= 208 and image[i, j] <= 223):  
 new\_image[i, j] = 207  
 elif(image[i, j] >= 224 and image[i, j] <= 239):  
 new\_image[i, j] = 223  
 elif(image[i, j] >= 240 and image[i, j] <= 255):  
 new\_image[i, j] = 239  
 return new\_image  
  
def lower\_by\_4(image):  
 rows, cols = image.shape  
 new\_image = np.ones((rows, cols), dtype=np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 if(image[i, j] >= 0 and image[i, j] <= 63):  
 new\_image[i, j] = 0  
 elif(image[i, j] >= 64 and image[i, j] <= 127):  
 new\_image[i, j] = 63  
 elif(image[i, j] >= 128 and image[i, j] <= 191):  
 new\_image[i, j] = 127  
 elif(image[i, j] >= 192 and image[i, j] <= 255):  
 new\_image[i, j] = 191  
  
 return new\_image  
  
def lower\_by\_2(image):  
 rows, cols = image.shape  
 new\_image = np.ones((rows, cols), dtype=np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 if (image[i, j] >= 0 and image[i, j] <= 127):  
 new\_image[i, j] = 0  
 elif (image[i, j] >= 128 and image[i, j] <= 255):  
 new\_image[i, j] = 255  
  
 return new\_image  
  
img\_path = "D:/Uni/Semester 6/DIP/Self/Lab/Lab 2/gradient.png"  
orig\_img = cv.imread(img\_path, 0)  
  
down\_16 = lower\_by\_16(orig\_img)  
down\_4 = lower\_by\_4(orig\_img)  
down\_2 = lower\_by\_2(orig\_img)  
  
cv.imshow('Window1', orig\_img)  
cv.waitKey()  
  
cv.imshow('Window1', down\_16)  
cv.waitKey()  
  
cv.imshow('Window1', down\_4)  
cv.waitKey()  
  
cv.imshow('Window1', down\_2)  
cv.waitKey()

**Output:**

****

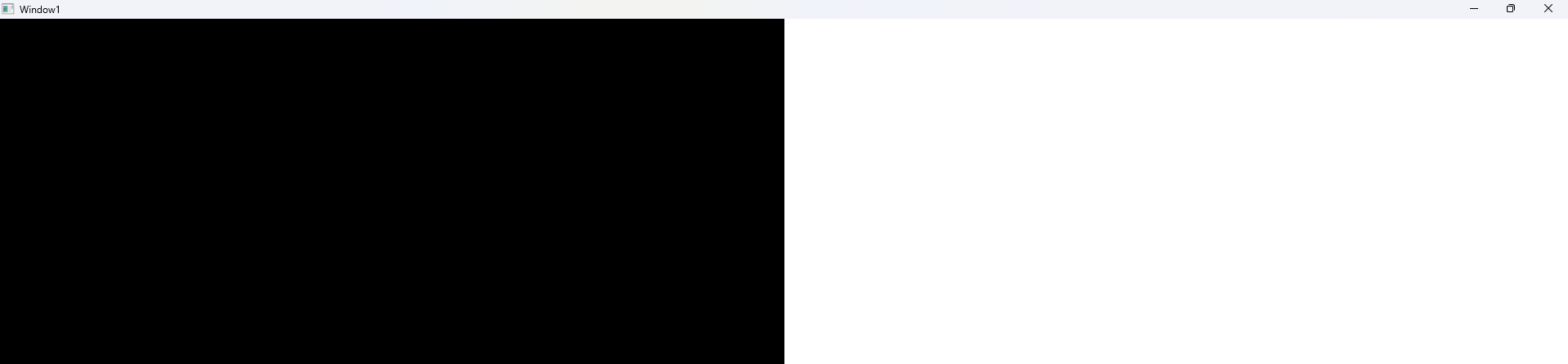
**Original**

****

**16 Values**

****

**4 Values**

****

**2 Values**

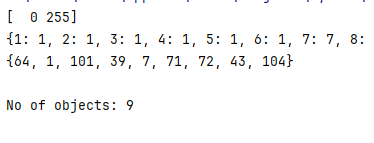
**Task 2:**

**For the images given below (also available with the lab handout), apply the connected component labelling using 4 connectivity and count the total number of objects in the list. (HINT: In the image given here, the background (black portion) has a numeric value of 1 while the white objects have a numeric value of 255.)**

**Solution**

import numpy as np  
import cv2 as cv  
  
def padding(pad, orig):  
 rows, cols = orig.shape  
 padded\_arr = np.ones((rows+ 2 \* pad, cols+ 2 \* pad), dtype = np.uint8)\*0  
  
 for i in range(rows):  
 for j in range(cols):  
 padded\_arr[i+pad][j+pad] = orig[i][j]  
  
 return padded\_arr  
  
def lower\_by\_2(image):  
 rows, cols = image.shape  
 new\_image = np.ones((rows, cols), dtype=np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 if (image[i, j] >= 0 and image[i, j] <= 127):  
 new\_image[i, j] = 0  
 elif (image[i, j] >= 128 and image[i, j] <= 255):  
 new\_image[i, j] = 255  
  
 return new\_image  
  
def cc(orig):  
 rows, cols = orig.shape  
 new\_img = np.zeros((rows, cols), dtype=np.uint8)  
 my\_dict = {}  
 count = 1  
  
 for i in range(1, rows):  
 for j in range(1, cols):  
 if orig[i][j] == 255:  
 if (orig[i-1][j] == 0 and orig[i][j-1] == 0):  
 new\_img[i][j] = count  
 my\_dict[count] = count  
 count += 1  
 elif (orig[i-1][j] == 255 and orig[i][j-1] == 0):  
 new\_img[i][j] = find\_root(my\_dict, new\_img[i-1][j])  
 elif (orig[i-1][j] == 0 and orig[i][j-1] == 255):  
 new\_img[i][j] = find\_root(my\_dict, new\_img[i][j-1])  
 elif (orig[i-1][j] == 255 and orig[i][j-1] == 255):  
 *#This is to determine the smallest of the two values (left and up)* root1 = find\_root(my\_dict, new\_img[i-1][j])  
 root2 = find\_root(my\_dict, new\_img[i][j-1])  
 if root1 != root2:  
 if root1 < root2:  
 my\_dict[root2] = root1  
 else:  
 my\_dict[root1] = root2  
 new\_img[i][j] = min(root1, root2)  
 else:  
 new\_img[i][j] = root1  
  
 for i in range(1, rows):  
 for j in range(1, cols):  
 if new\_img[i][j] != 0:  
 new\_img[i][j] = find\_root(my\_dict, new\_img[i][j])  
  
 return new\_img, my\_dict  
  
*#Back tracks until it finds the root of that pixel i.e if we have a label of 1, 2 and 3 all in the same object, they all need to be corrected to point to 1*def find\_root(my\_dict, x):  
 if x not in my\_dict:  
 my\_dict[x] = x  
 return x  
 while my\_dict[x] != x:  
 my\_dict[x] = my\_dict[my\_dict[x]] *# Path compression* x = my\_dict[x]  
 return x  
  
image = lower\_by\_2(padding(1, (cv.imread("D:/Uni/Semester 6/DIP/Self/Lab/Lab 2/cc.png", 0))))  
cv.imshow('Window1',image)  
cv.waitKey()  
  
print(np.unique(image))  
new\_img, my\_dictionary = cc(image)  
print(my\_dictionary)  
no\_obj = set(my\_dictionary.values())  
print(no\_obj)  
print(f"\nNo of objects: {len(no\_obj)}")  
  
  
cv.imshow('Window1', new\_img)  
cv.waitKey()

**Output:**



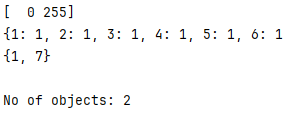
**Error:**

The algorithm sees 9 objects, in place of 8. This is because of the way this object has been created. On smaller size images with clear definition of boundaries, this shows 2 objects.

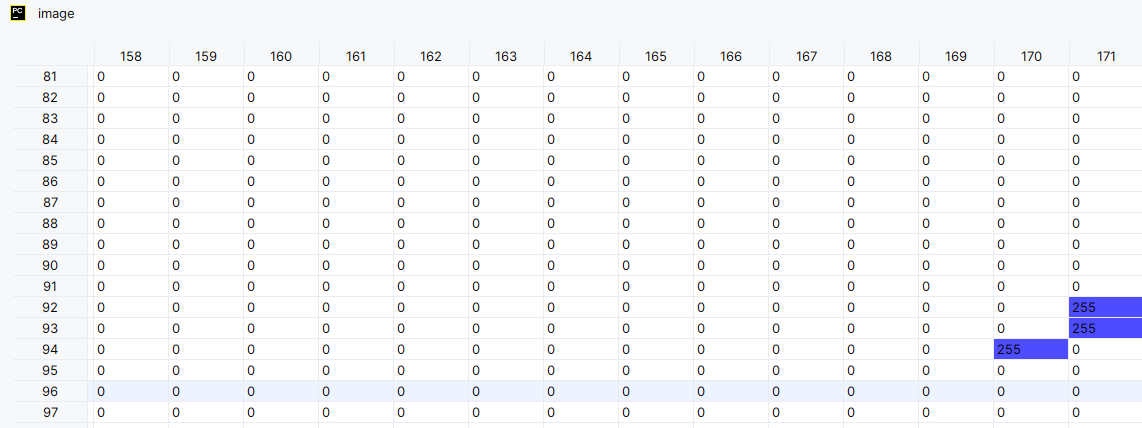
E.g:



This is a 10x10 image, with 2 clear objects. The result my program produces is:



Our provided image has one outlier pixel located at **(95, 170)**.



If you observed the image from the human eye, you could never distinguish this singular pixel. But on observing this pixel as an array, we see that it indeed does not have any adjoint pixels in terms of 4 connectivity. This effectively makes it a separate object in terms of 4 connectivity.