

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Question-1

a) For the image shown in a), calculate the centroids of each object and display them in red color. Note: you cannot use Image Moments to find centroids

1: Define the structuring element for dilation and erosion as a 3x3 matrix filled with ones using NumPy's `np.ones()` function.

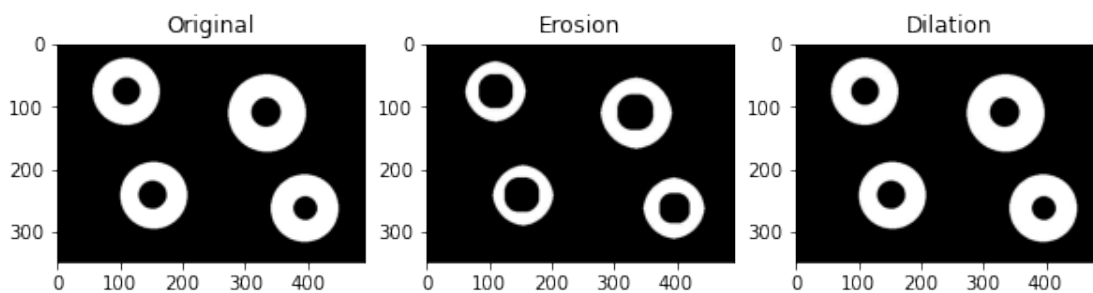
2: Apply erosion operation on the input image `img` using the `cv2.erode()` function with the defined kernel and 5 iterations, and store the result in the `img_erosion` variable.

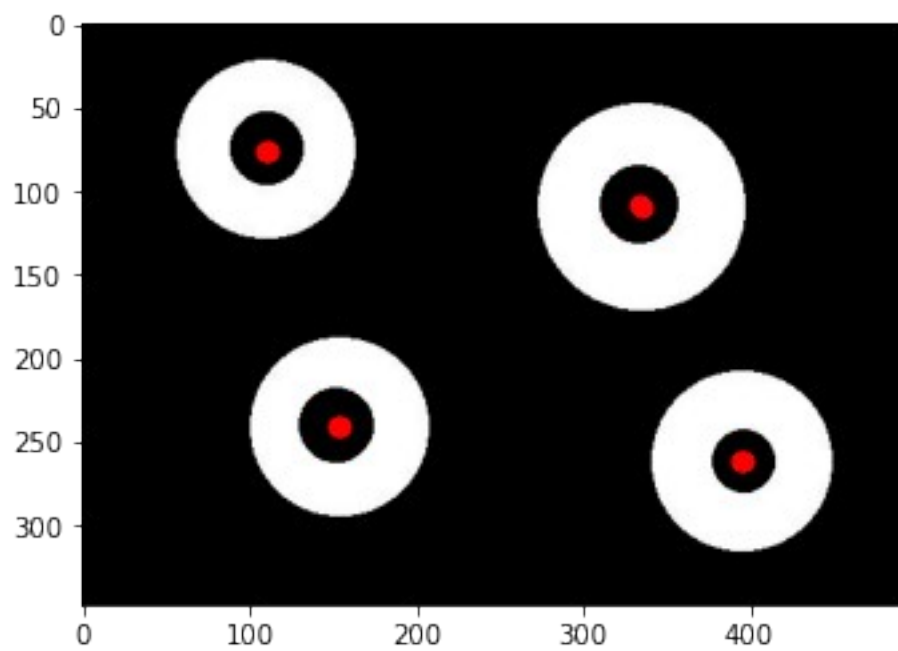
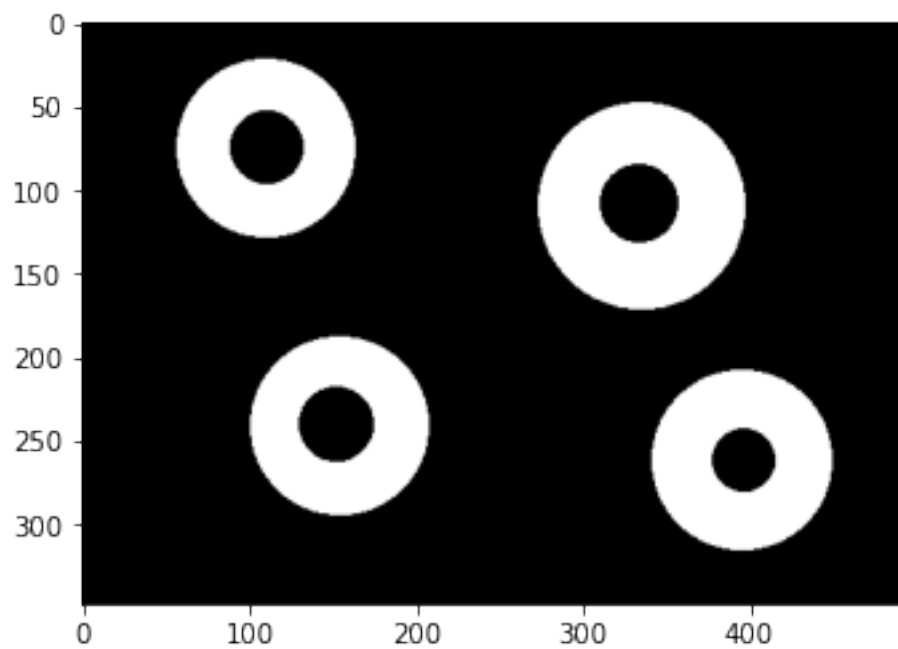
3: Apply dilation operation on the eroded image `img_erosion` using the `cv2.dilate()` function with the same kernel and iterations, and store the result in the `img_dilation` variable.

4: Threshold the dilated image `img_dilation` using the `cv2.threshold()` function to convert it to a binary image.

5: Find contours in the thresholded image using the `cv2.findContours()` function, and store the contours and hierarchy in the `contours` and `hierarchy` variables, respectively.

6: Loop over each contour in `contours`, calculate its centroid using the `cv2.boundingRect()` function, and append it to the `centroids` list.



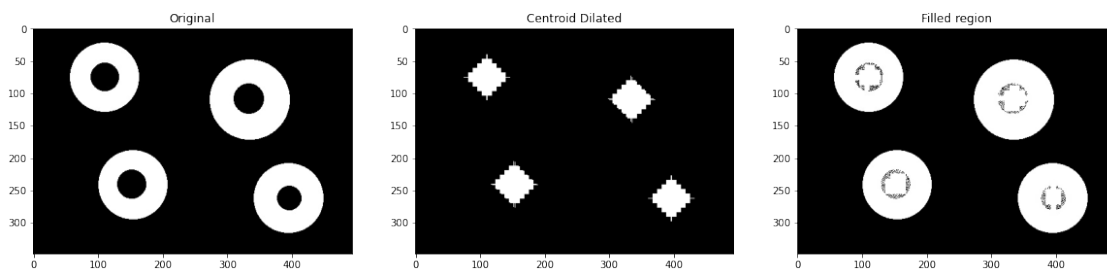


b) Fill the holes of these objects (figure-1 a) using Morphological Dilation as discussed in class. You can use the centroids calculated in a) as marker points for dilation to fill the centers.

1- Define a 15x15 cross-shaped kernel for dilation

2- Create a marker image with the centroids in white color. The marker image is initialized with all zeros using `np.zeros_like()` function and then the centroids are set to 255 (white).

3- Apply dilation to the marker image with the kernel defined in step 2 to fill the centers using `cv2.dilate()` function and store the result in `dilated_img`



c) Calculate the diameter of the biggest hole (figure-1 a) and display its diameter above that circle in green color.

1- Threshold the input image to create a binary image where white represents the region of interest.

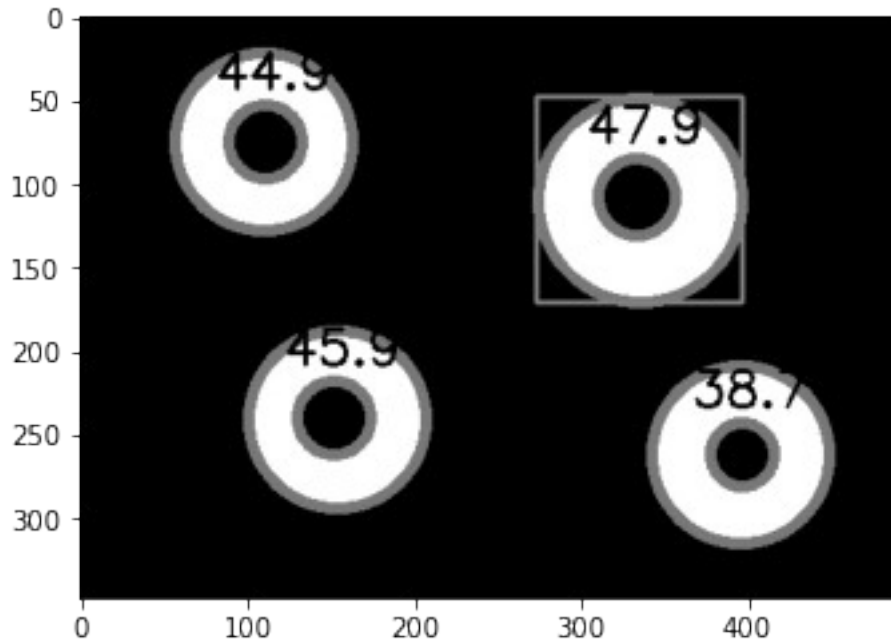
2- Find contours in the binary image using `cv2.findContours()` function.

3- Create a copy of the input image to draw the diameter of the largest circle on.

4- Keep track of the contour with the largest diameter found so far.

5- Draw a circle with the diameter on the copy of the input image and display the diameter value on the image.

6- Draw a rectangle with the largest diameter on the copy of the input image.



d) count the number of objects with holes and without holes and display the count

1- Thresholding method is applied to create a binary image, where the background is black and the objects are white.

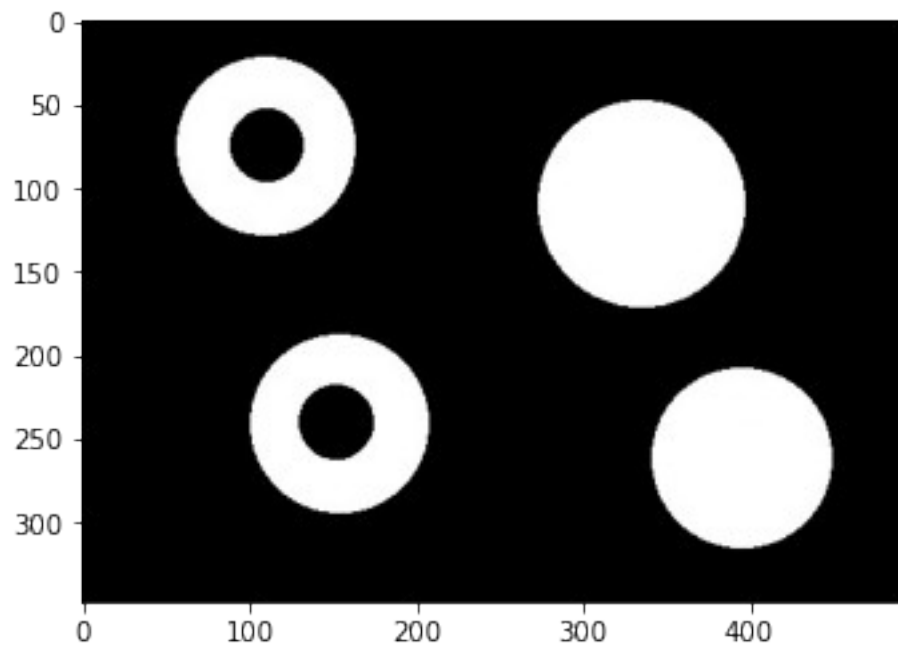
2- The `findContours()` function from OpenCV is used to find contours in the binary image.

3- Two counters are initialized to keep track of the number of circles with and without holes.

4- The code loops over each contour found in the image.

5- For each contour, the area is calculated using the `contourArea()` function.

6- The code checks if the area of the contour is within a certain range to filter out noise.



Circles with holes: 2
Circles without holes: 2

Question 2.

For the image shown in Figure 2, find the number of persons in the image. Draw a bounding box around each person and display the image.

1- Apply bilateral filtering on the input image `img` with a filter size of `15x15` and sigma values of `15`.

2- Threshold the filtered image using a threshold value of `140`, and set all pixel values below this threshold to `0`. The thresholding mode used here is `cv2.THRESH_TOZERO`, which means all pixel values above the threshold are kept as is, while pixel values below the threshold are set to `0`.

3- Define a `5x5` kernel of ones to use for morphological operations.

4- Perform erosion operation on the thresholded image to remove small objects and noise.

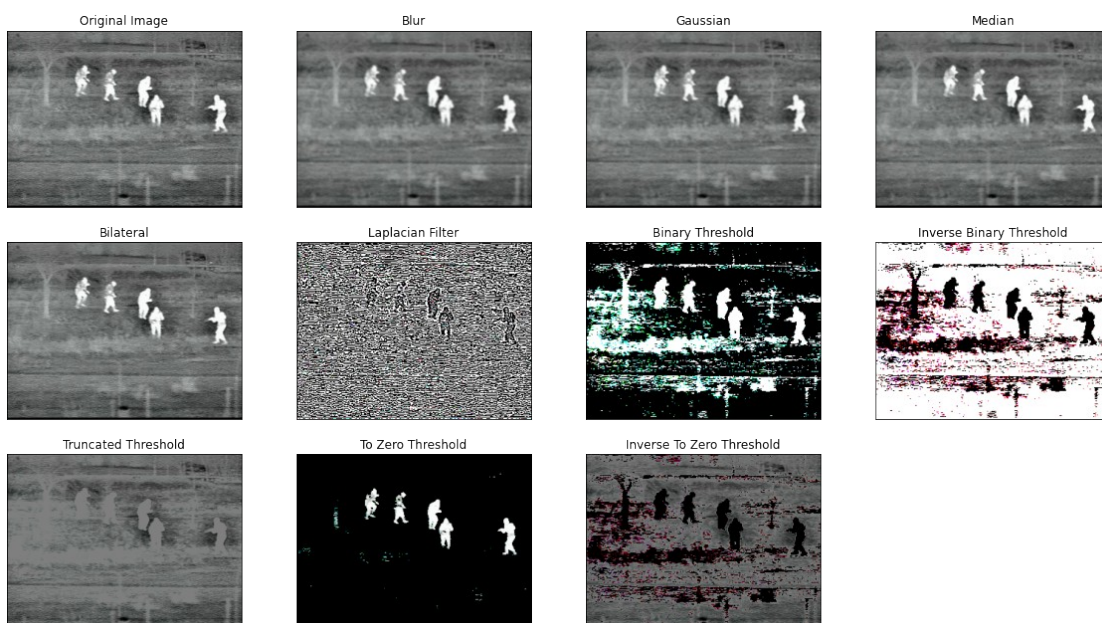
5- Perform dilation operation on the eroded image to fill in gaps and expand the remaining objects.

6- Find contours in the dilated image using `cv2.findContours` function. The `cv2.RETR_EXTERNAL` flag retrieves only the external contours and `cv2.CHAIN_APPROX_SIMPLE` compresses horizontal, vertical, and diagonal segments and leaves only their end points.

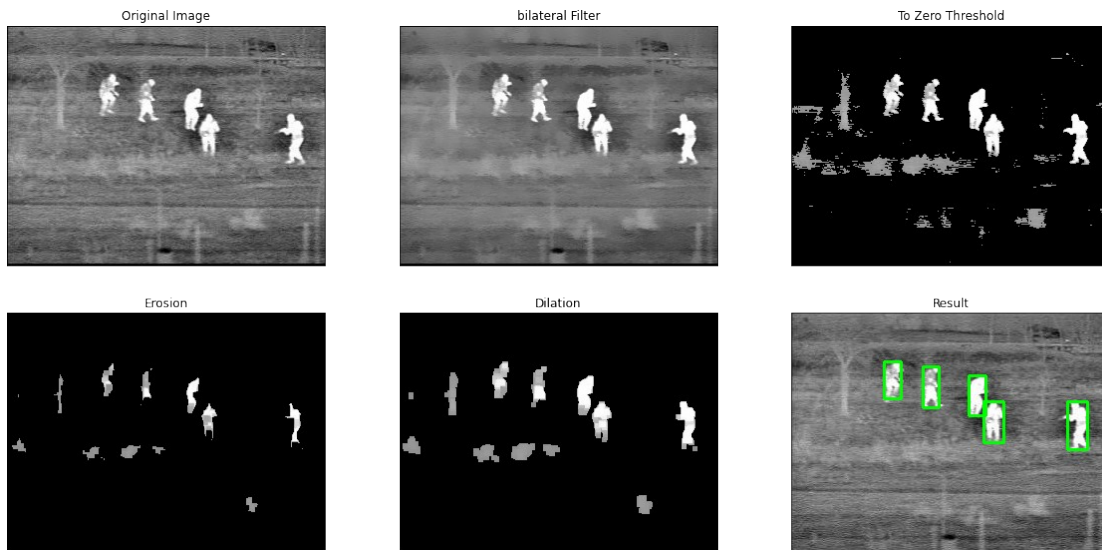
7- Define minimum and maximum area thresholds for the detected contours.

8- Draw a green bounding box around each remaining contour in the `img_bbox` image.

Clipping input data to the valid range for `imshow` with RGB data (`[0..1]` for floats or `[0..255]` for integers).



Number of people in the image: 5



Question 3.

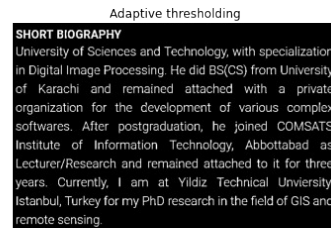
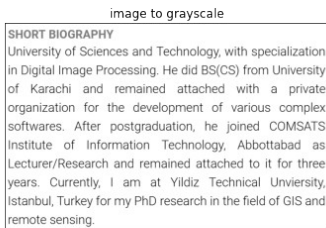
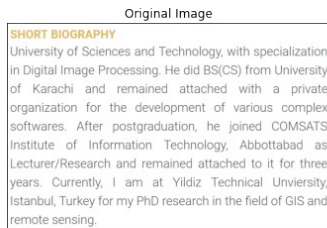
a) For the image shown in Figure 3, count the number of text lines in the image

1- Apply adaptive thresholding to create a binary image with the text in black and background in white

2- Apply morphological operations dilation fill gaps in the text

3- connected component analysis on the binary image dilation, which means that it identifies distinct regions of the image that are connected together.

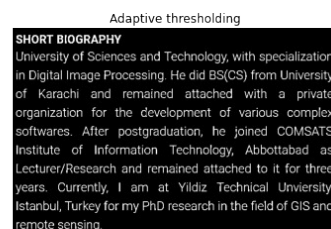
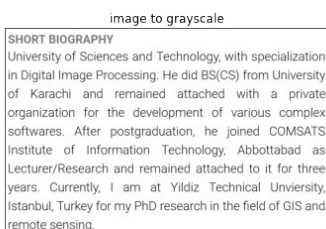
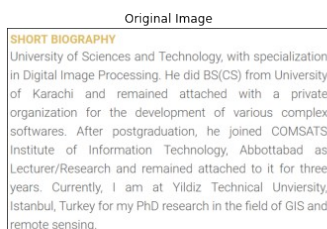
Number of text lines: 11



b) Count the number of words in the given image

- 1- Apply adaptive thresholding to create a binary image with the text in black and background in white
- 2- Apply morphological operations dilation fill gaps in the word
- 3- connected component analysis on the binary image dilation, which means that it identifies distinct regions of the image that are connected together.

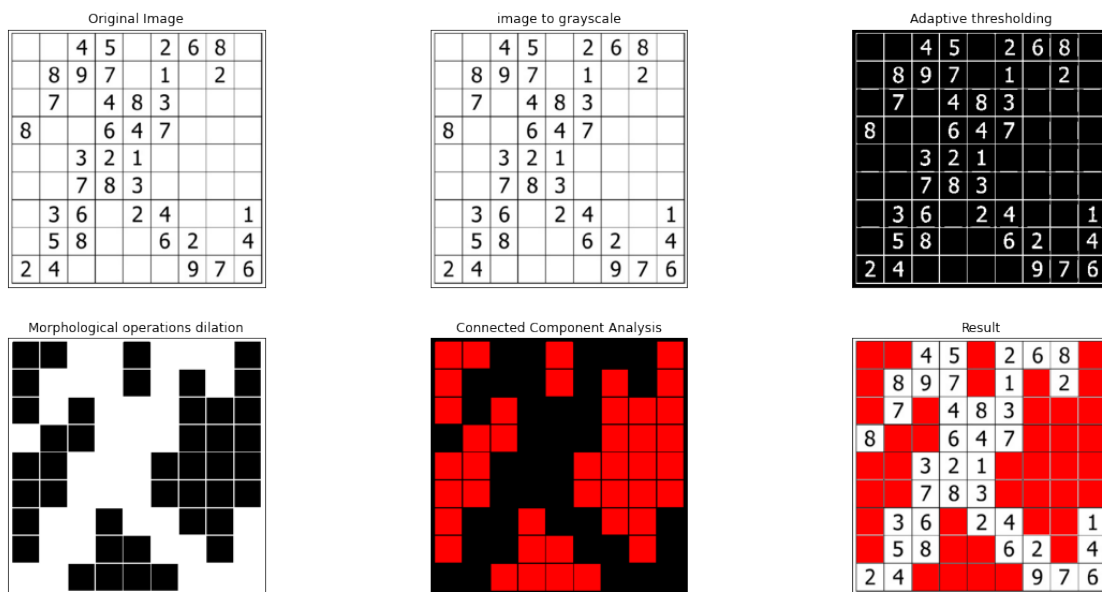
Number of text word: 75



Consider the Sudoku game image shown in Figure 4. Only identify the empty squares, color them red, and display the output image. The rest of the squares will remain unchanged.

2- Apply morphological operations to remove noise and fill gaps

4- Compare the original image with the processed image and replace the pixel values accordingly



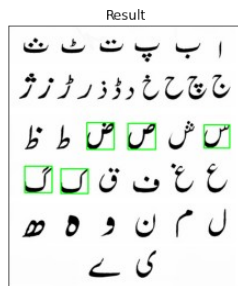
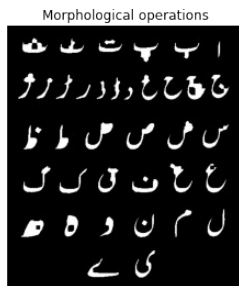
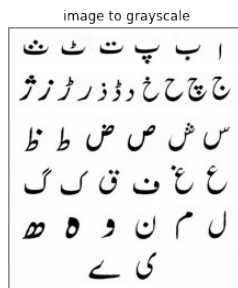
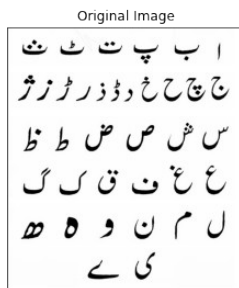
Question 5.

Figure 5 shows an image of the Urdu alphabet. All these characters have different widths and heights. For instance, Alif is thin but tall. Similarly, Pay has a more extended width than height and three associated dots. a) Write a program that displays all characters whose width exceeds their height.





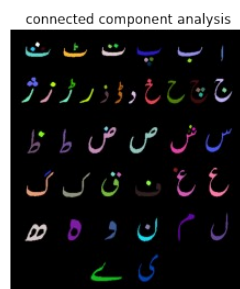
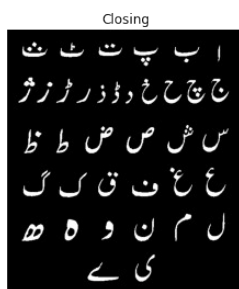
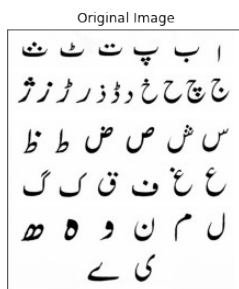
b) Write a program that displays all the alphabets that have almost the same width and height

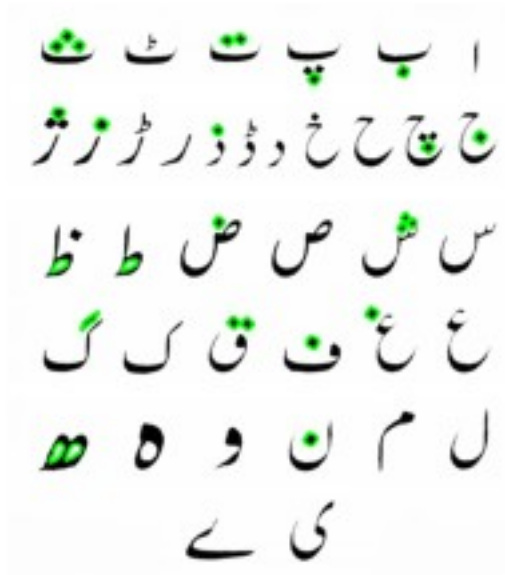




c) Write a program that displays all alphabets that have dots (Nukta).

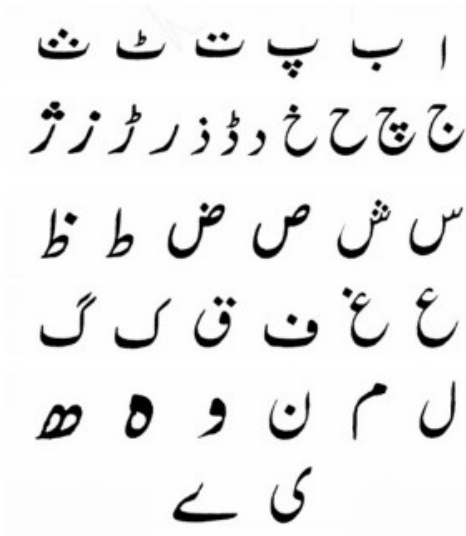
Labels 69



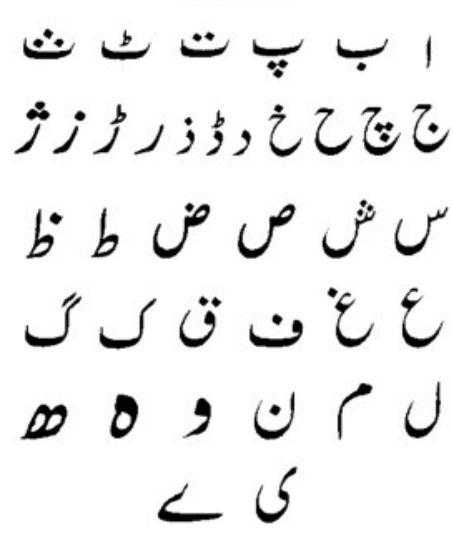


Write a program to display all alphabets which are composed of four components, e.g. It has one body and three dots

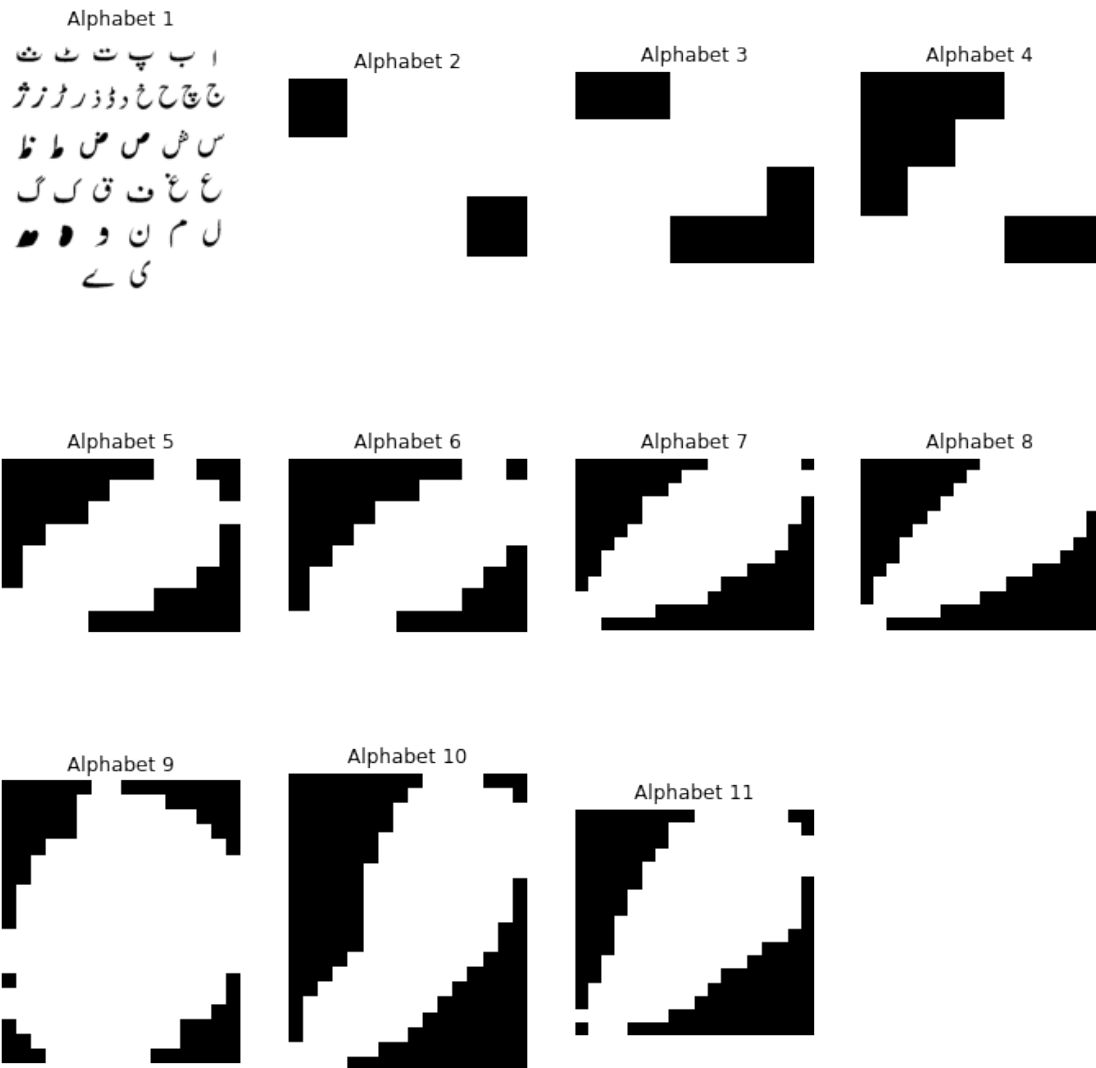
Original Image



Binary Image



Alphabets with Four Components



Write a code to display all alphabets which have holes in them. E.g.,
has two holes, has one hole

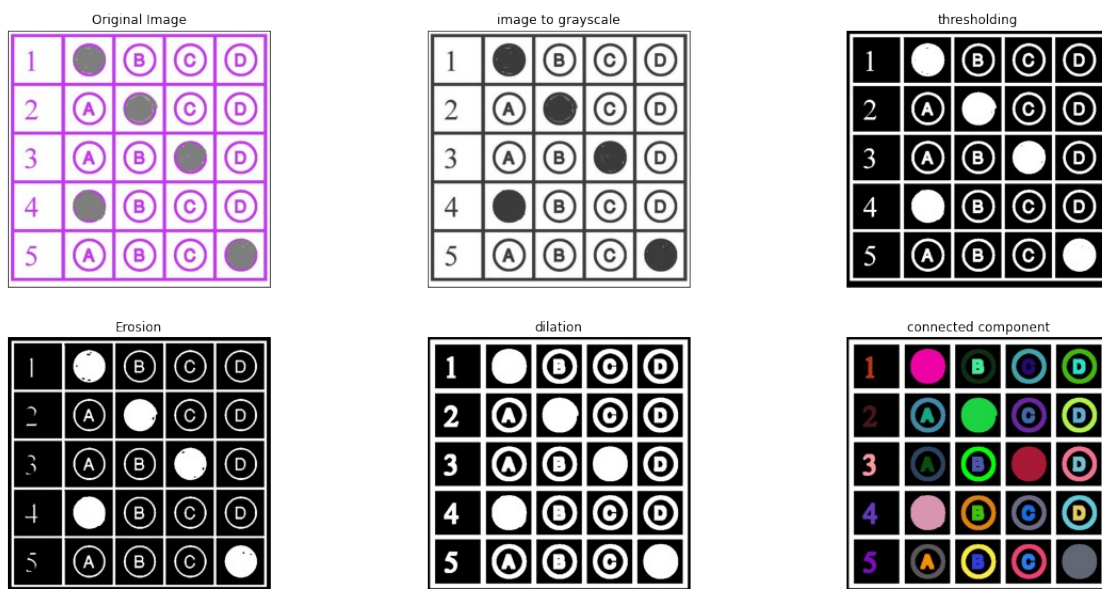
Question 6.

Figure 6 shows an image of a bubble sheet filled for an exam. There are five questions, and for each question, there are four choices.

Write a program that applies DIP techniques to identify the candidate's answer for each question. Display the output in the format as shown below:

1:A, 2:B, 3:C, 4: A, 5:D

Labels 42



1- Apply binary thresholding on the grayscale image using OpenCV's `cv2.threshold()` function. This creates a binary image where all pixels above a certain threshold value are set to white (255), and all pixels below the threshold are set to black (0). The thresholded image is stored in the `thresh` variable.

2- Find contours in the thresholded image using OpenCV's `cv2.findContours()` function. Contours are the boundaries of objects in an image. The contours and hierarchy information are stored in the `contours` and `hierarchy` variables, respectively.

3- Filter the contours based on their area. The code removes all contours with an area less than 10000 or greater than 15000 pixels. The filtered contours are stored in the `filtered_contours` list.

4- Sort the filtered contours based on their y-axis position using the `sorted()` function and the key parameter with the `cv2.boundingRect()` function.

5- Group the filtered contours that have y-coordinates within a certain range. The code creates a list of lists, where each sublist contains contours that are close together on the y-axis.

6- Sort the contours in each group based on their x-axis position using the `sort()` function and the key parameter with the `cv2.boundingRect()` function.

7- Find the contour with the maximum filled area within each group. The code iterates through each group and checks each contour to see if it is filled. If a contour is more than 65% filled, it is added to the `filled_contours` list, and its corresponding option is added to the `correct_options` list.

8- Draw the filled contours on a copy of the original image using OpenCV's `cv2.drawContours()` function. The copy of the image with the contours drawn on it is stored in the `img_copy` variable.

9- Print the correct option for each filled bubble found by iterating through the `correct_options` list and using the `alphabats` string to map the option number to a letter. The output is printed to the console.

Original Image





















1		(B)	(C)	(D)
2	(A)		(C)	(D)
3	(A)	(B)		(D)
4		(B)	(C)	(D)
5	(A)	(B)	(C)	

image to grayscale

1		(B)	(C)	(D)
2	(A)		(C)	(D)
3	(A)	(B)		(D)
4		(B)	(C)	(D)
5	(A)	(B)	(C)	

thresholding

1		(B)	(C)	(D)
2	(A)		(C)	(D)
3	(A)	(B)		(D)
4		(B)	(C)	(D)
5	(A)	(B)	(C)	

1		(B)	(C)	(D)
2	(A)		(C)	(D)
3	(A)	(B)		(D)
4		(B)	(C)	(D)
5	(A)	(B)	(C)	

1 : A
 2 : B
 3 : C
 4 : A
 5 : D