OMR Sheet Scanner

AUTHORS

Department of Computer Science and Engineering ,Amrita Vishwa Vidyapeetham Coimbatore, 641112 ,India

Amidela Anil Kumar (cb.en.u4cse20206@cb.students.amrita.edu)

Manaswini Kar (cb.en.u4cse20236@cb.students.amrita.edu)

Dhanush Penugonda(cb.en.u4cse20216@cb.students.amrita.edu)

Gutha Venkat Sai (cb.en.u4cse20221@cb.students.amrita.edu)

Problem Statement

To take an OMR sheet image as an input and preprocess it to select the answer section in and evaluate it to give out the results.

Implementation

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

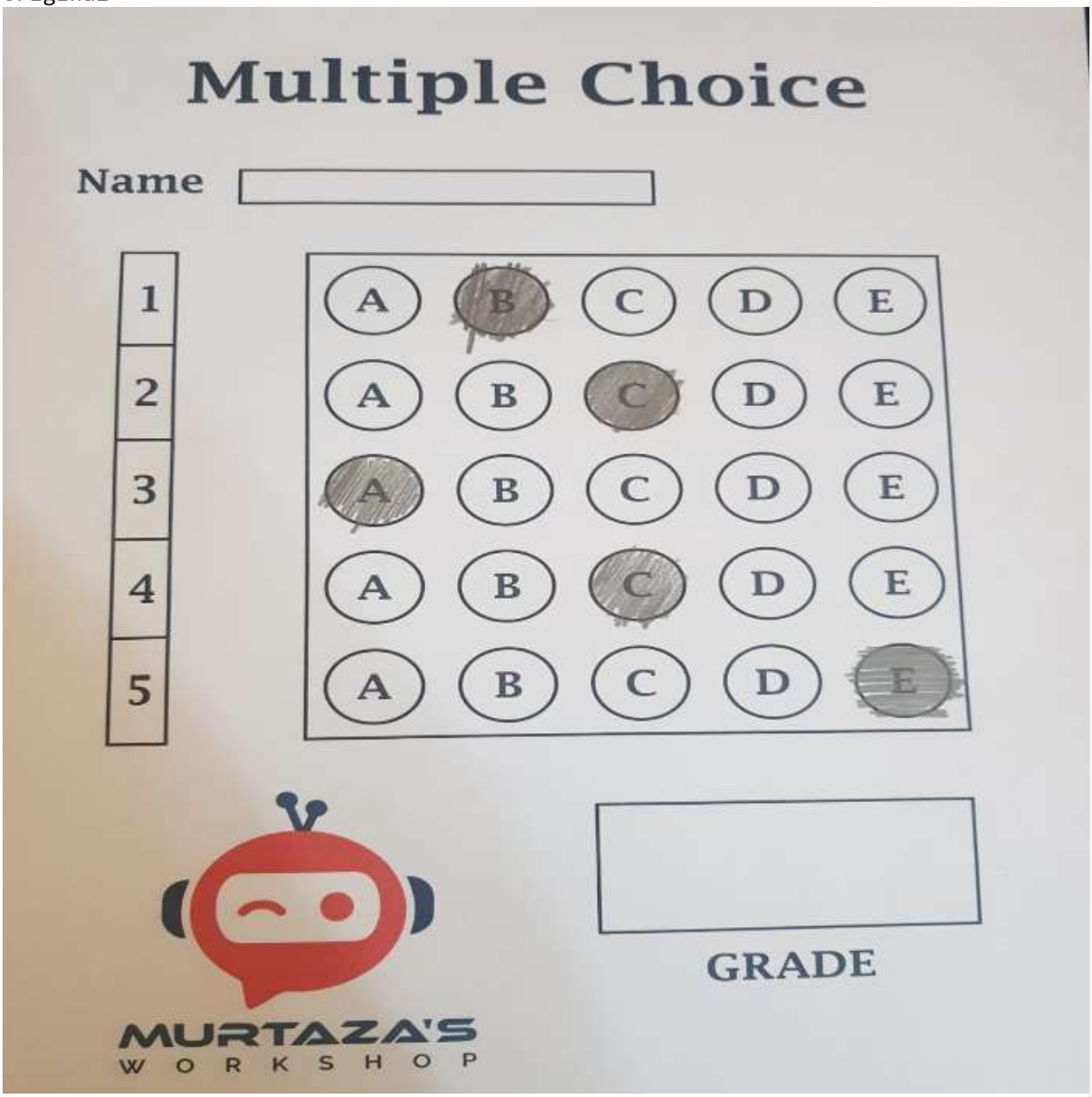
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import cv2
from google.colab.patches import cv2_imshow
import math
import random
```

Preprocessing

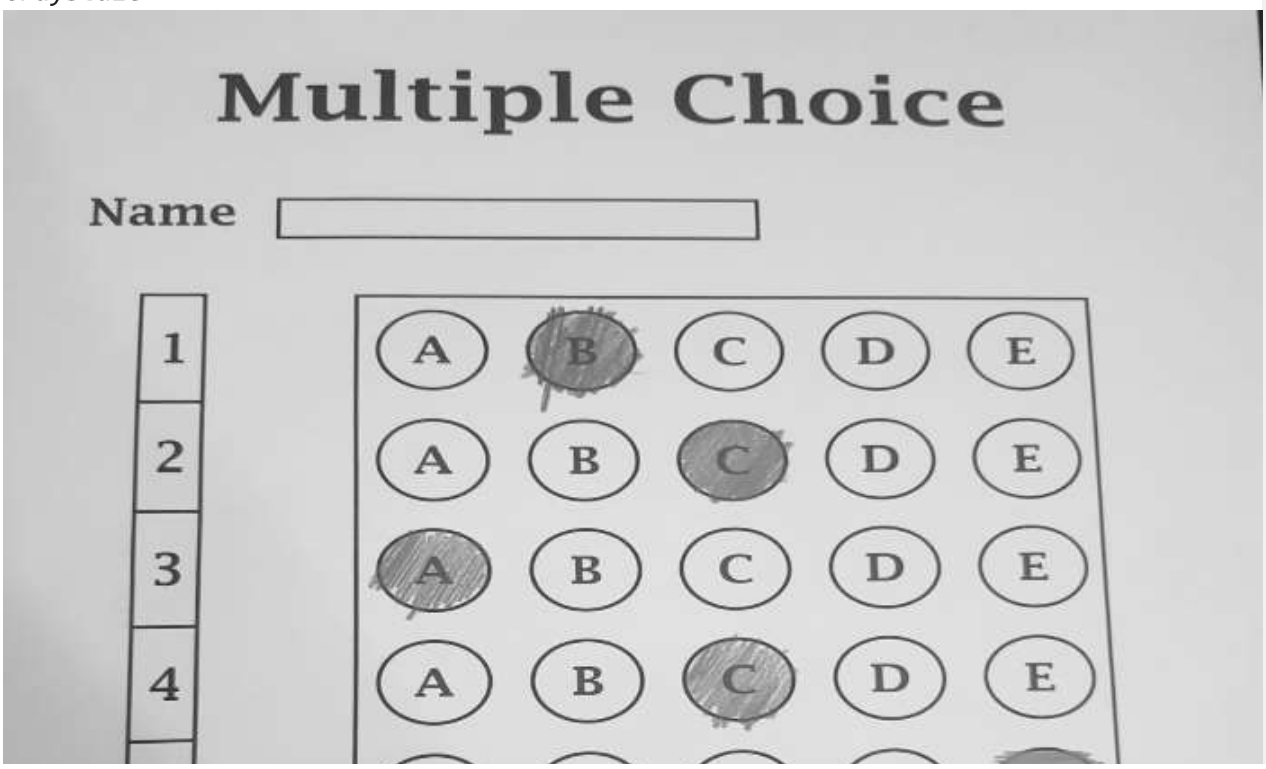
```
heightImg = 700
widthImg  = 700
questions = 5
```

```
choices = 5
solutions = [1,0,0,3,1]

imgColored = cv2.imread("/content/drive/MyDrive/Colab Notebooks/DIP/images/test-8.png")
imgColored = cv2.resize(imgColored, (widthImg, heightImg))
print("Original")
cv2_imshow(imgColored)
img = cv2.cvtColor(imgColored, cv2.COLOR_BGR2GRAY)
print("GrayScale")
cv2_imshow(img)
imgBlur = cv2.GaussianBlur(img, (5, 5), 1)
print("Blurred")
cv2_imshow(imgBlur)
imgCanny = cv2.Canny(imgBlur,10,70)
print("Canny (Edges Detected)")
cv2_imshow(imgCanny)
```

Original

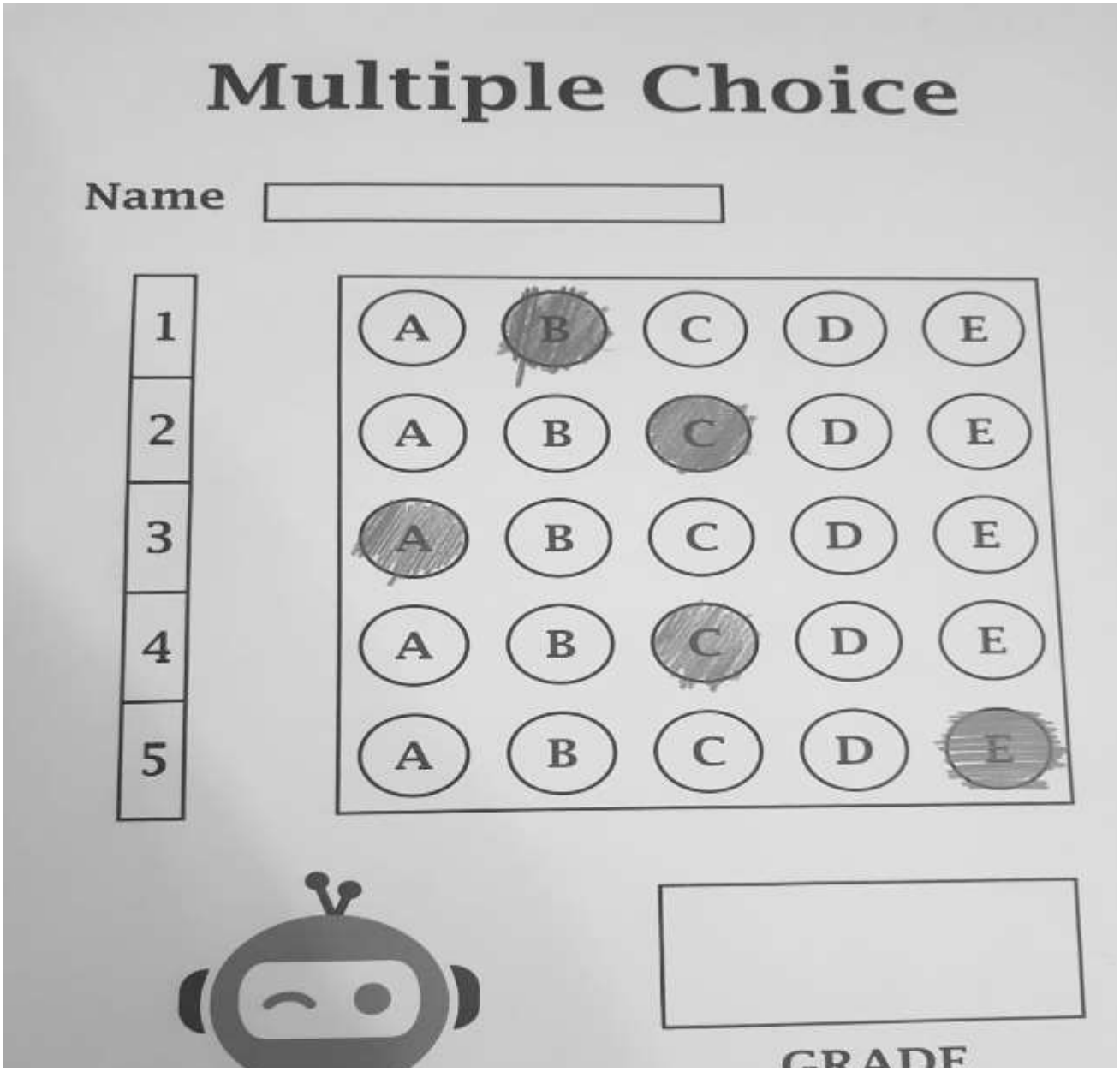


GrayScale

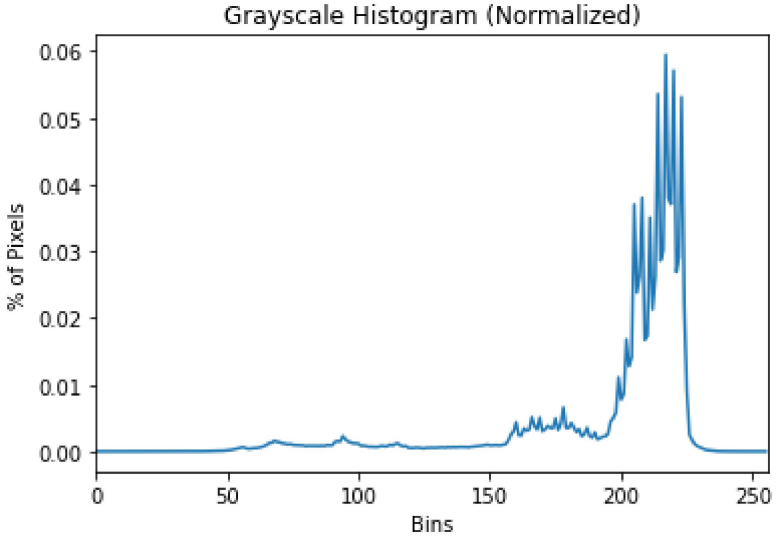


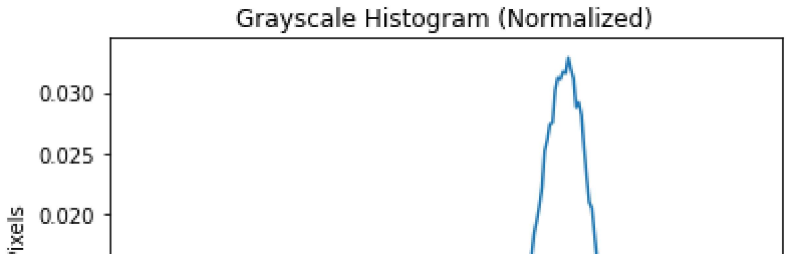
Noise 1

```
noise=np.zeros(img.shape,dtype=np.uint8)
cv2.randn(noise,128,20)
noise=(noise*0.45).astype(np.uint8)
imgNoisy= 0.55*img+noise
imgNoisy = imgNoisy.astype(np.uint8)
cv2_imshow(img)
cv2_imshow(imgNoisy)
```



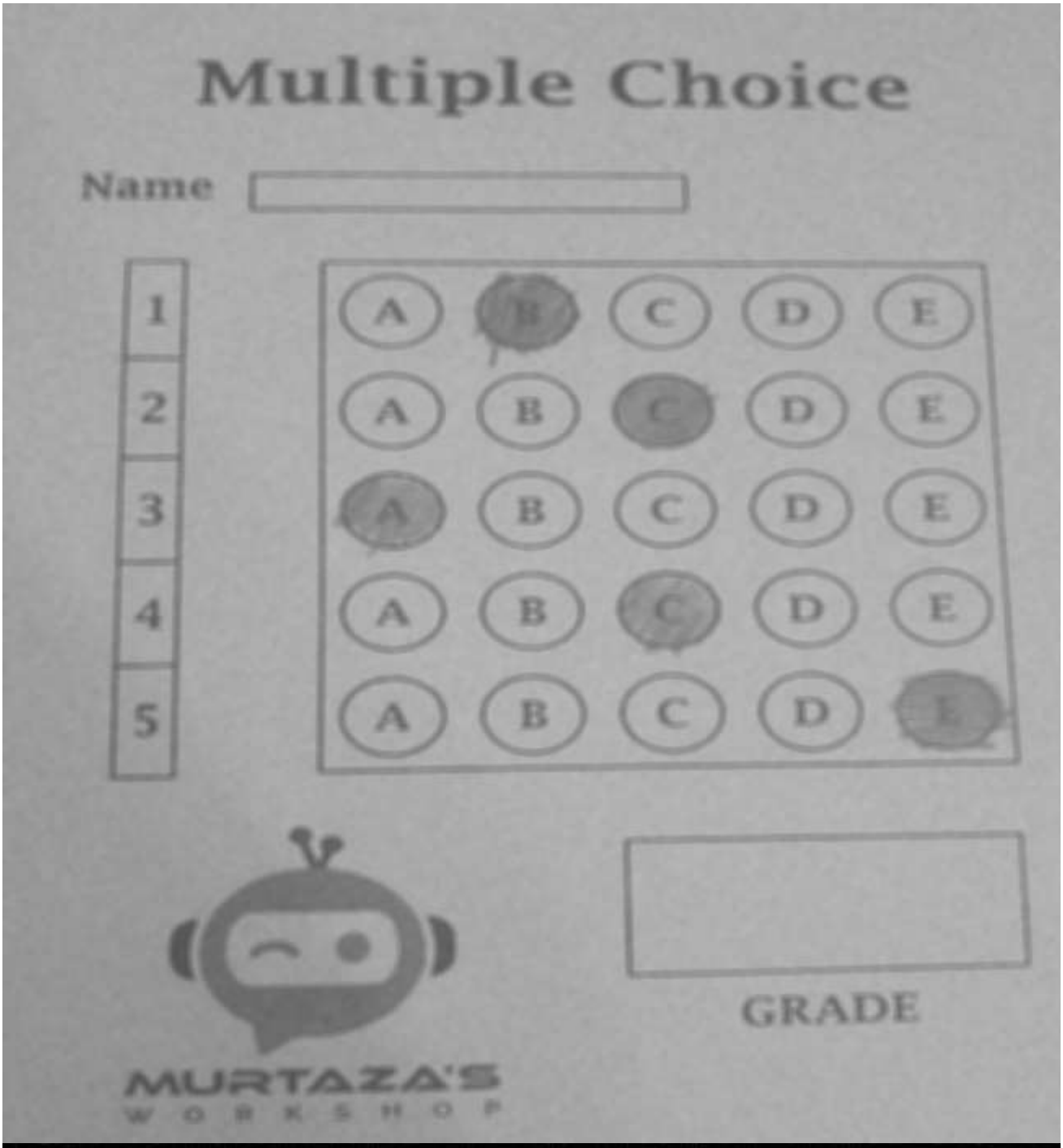

```
def histo(img):
    hist=cv2.calcHist([img], [0], None, [256], [0, 256])
    hist /= hist.sum()
    plt.figure()
    plt.title("Grayscale Histogram (Normalized)")
    plt.xlabel("Bins")
    plt.ylabel("% of Pixels")
    plt.plot(hist)
    plt.xlim([0, 256])
    plt.show()
histo(img)
histo(imgNoisy)
```

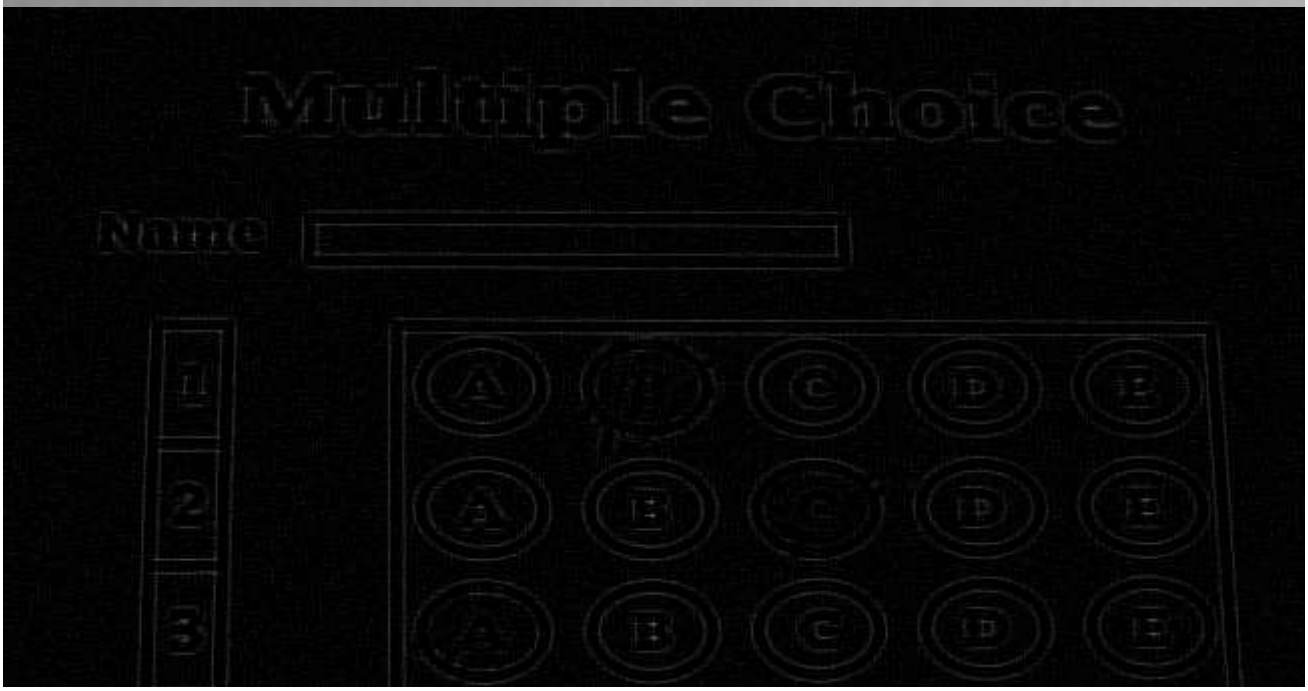




3x3 Mean and Laplacian filters

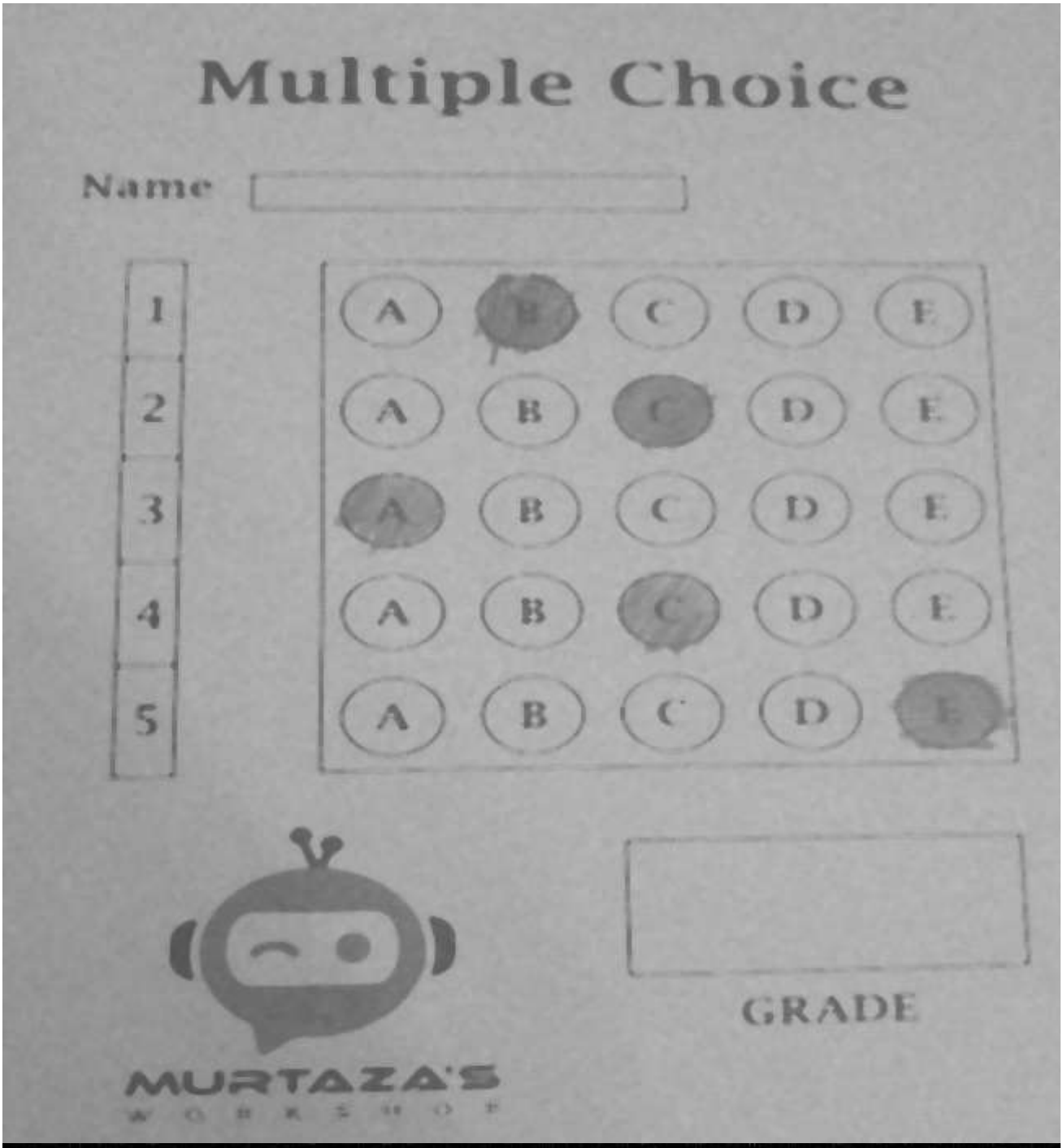
```
mean = np.ones((5, 5), np.float32)/25
imgNew = cv2.filter2D(src=imgNoisy, ddepth=-1, kernel=mean)
cv2_imshow(imgNew)
lapl = np.array([[-1, -1, -1],[-1, 8, -1],[-1, -1, -1]])
laplacian = cv2.filter2D(src=imgNew, ddepth=-1, kernel=lapl)
cv2_imshow(laplacian)
imgNew = 0.7*imgNew+0.3*laplacian
imgNew = imgNew.astype(np.uint8)
cv2_imshow(imgNew)
sum = 0.0
for x in range(widthImg):
   for y in range(heightImg):
      difference = (int(img[x,y]) - int(imgNew[x,y]))
      sum = sum + difference*difference
mean_mse = sum /(widthImg*heightImg)
print("The mean square error is:",mean_mse)
```

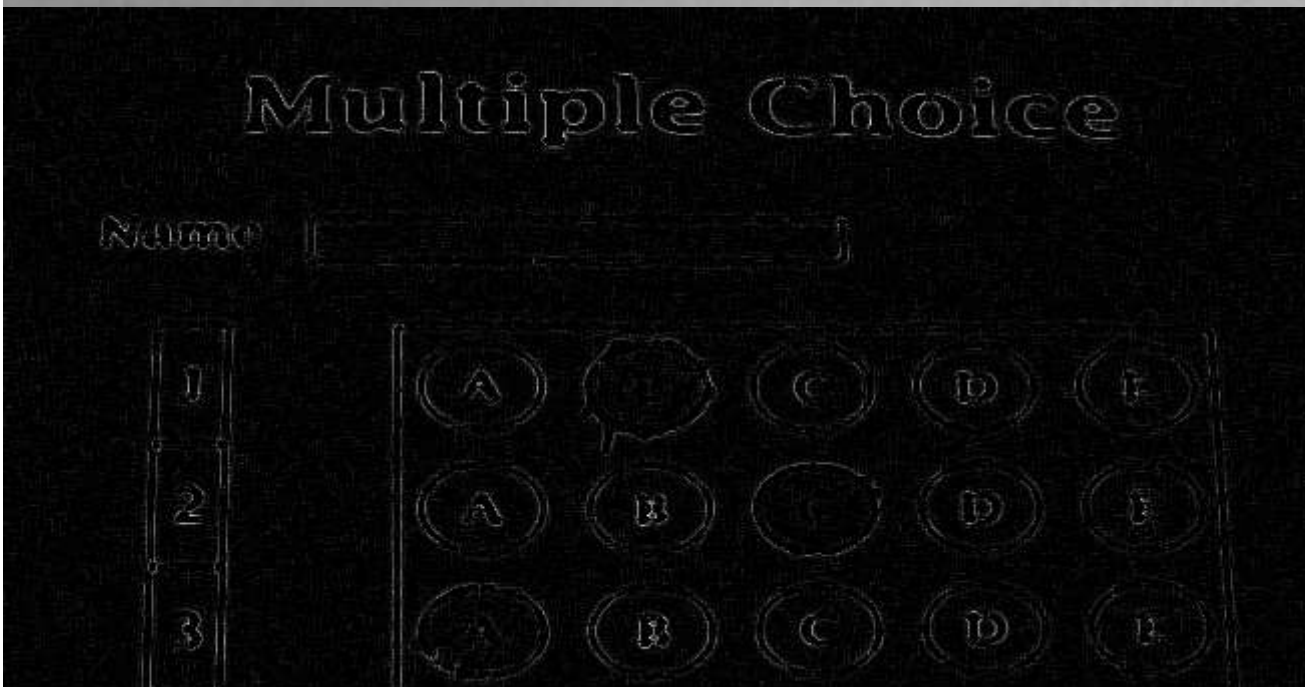

Multiple Choice
Name
1
2
3
4
5
A B C D E
A B C D E
A B C D E
A B C D E
A B C D
GRADE
MURTAZA'S
WORKSHOP

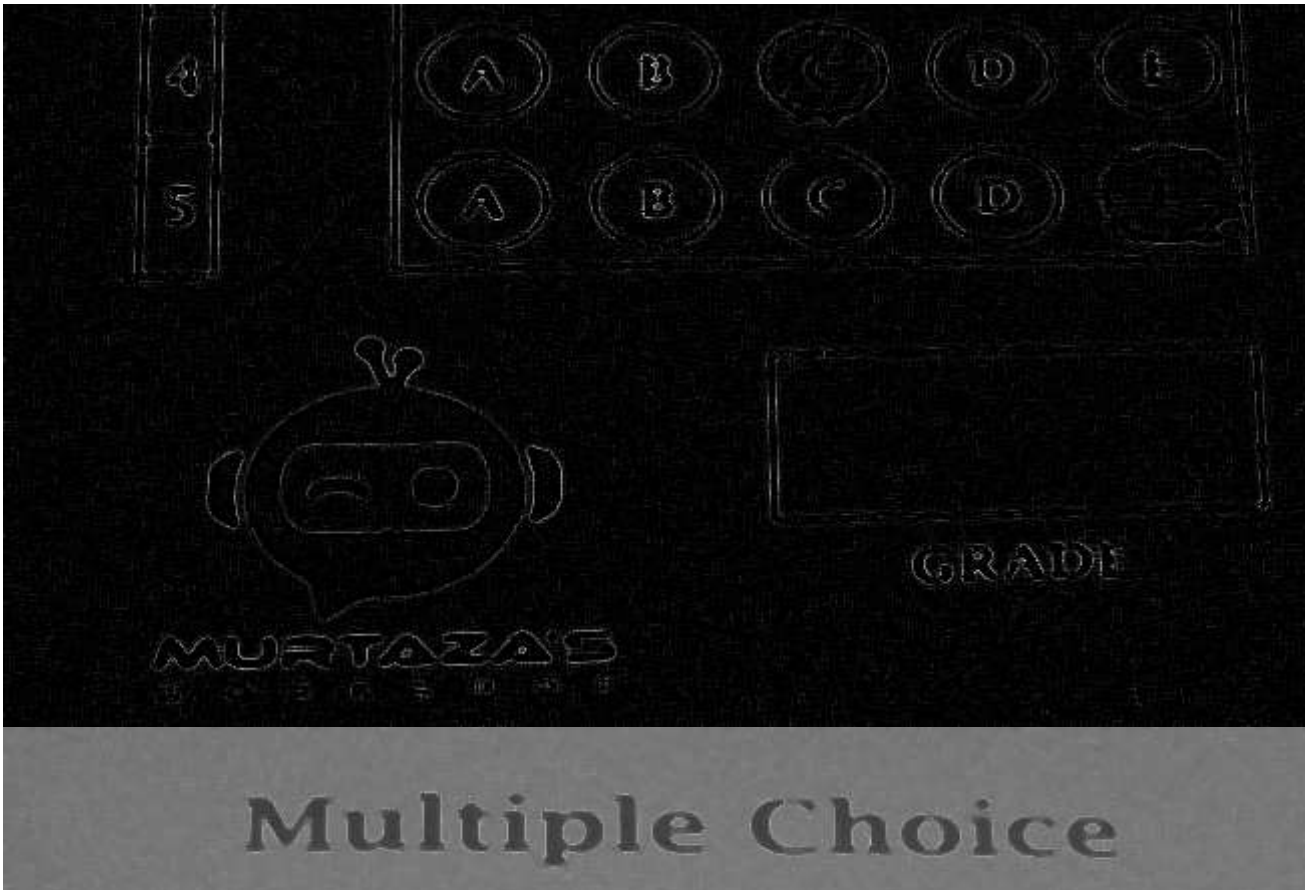

Multiple Choice
Name
1
2
3

3x3 Median and Laplacian filters

```
imgNew = cv2.medianBlur(imgNoisy,5)
cv2_imshow(imgNew)
lapl = np.array([[-1, -1, -1],[-1, 8, -1],[-1, -1, -1]])
laplacian = cv2.filter2D(src=imgNew, ddepth=-1, kernel=lapl)
cv2_imshow(laplacian)
imgNew = 0.7*imgNew+0.3*laplacian
imgNew = imgNew.astype(np.uint8)
cv2_imshow(imgNew)
sum = 0.0
for x in range(widthImg):
   for y in range(heightImg):
      difference = (int(img[x,y]) - int(imgNew[x,y]))
      sum = sum + difference*difference
median_mse = sum /(widthImg*heightImg)
print("The mean square error is:",median_mse)
```


Multiple Choice
Name
1
2
3
4
5
A B C D E
MURTAZA'S
WORKSHOP
GRADE

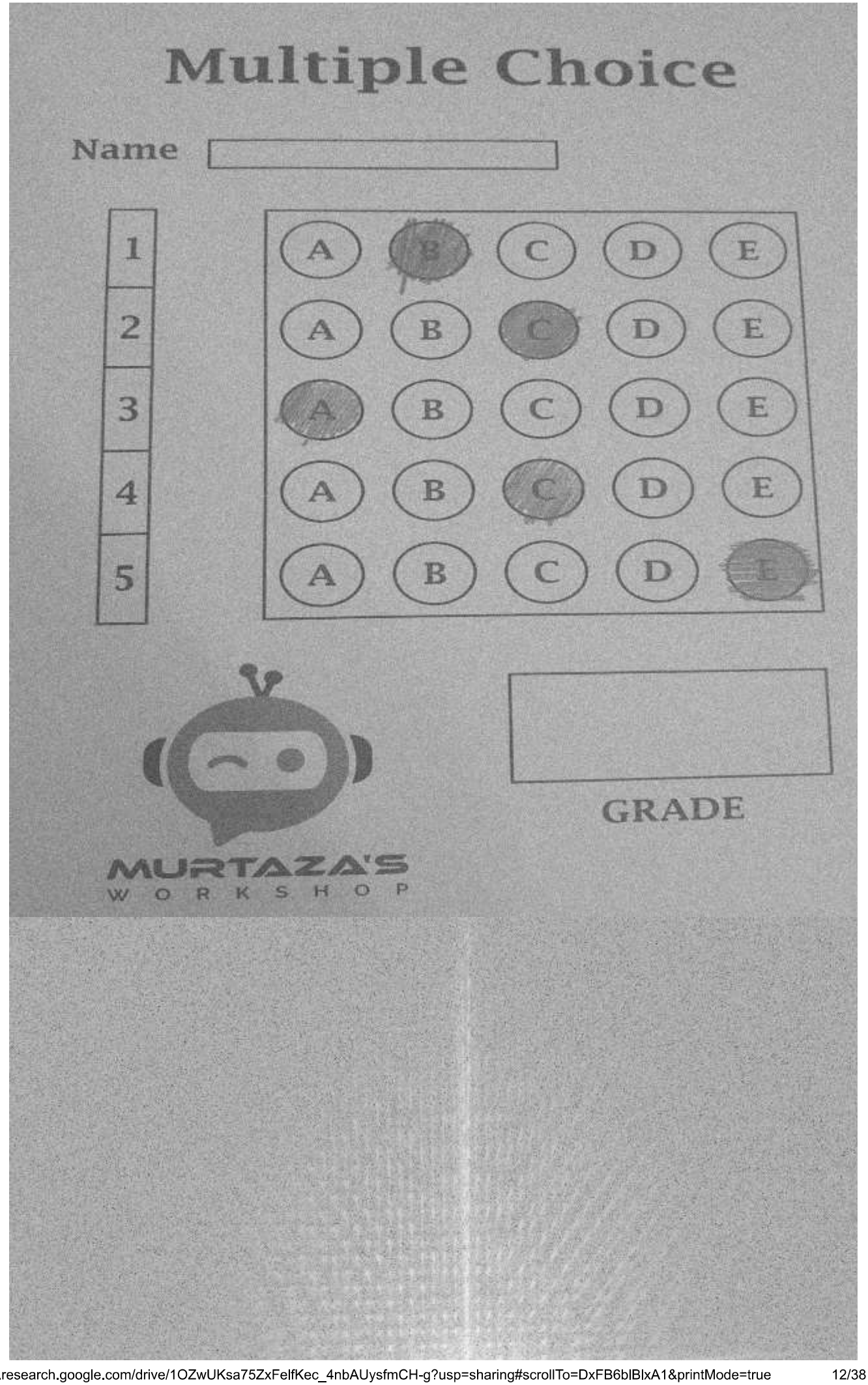

Multiple Choice
Name
1
2
3



Ideal Low Pass Filter

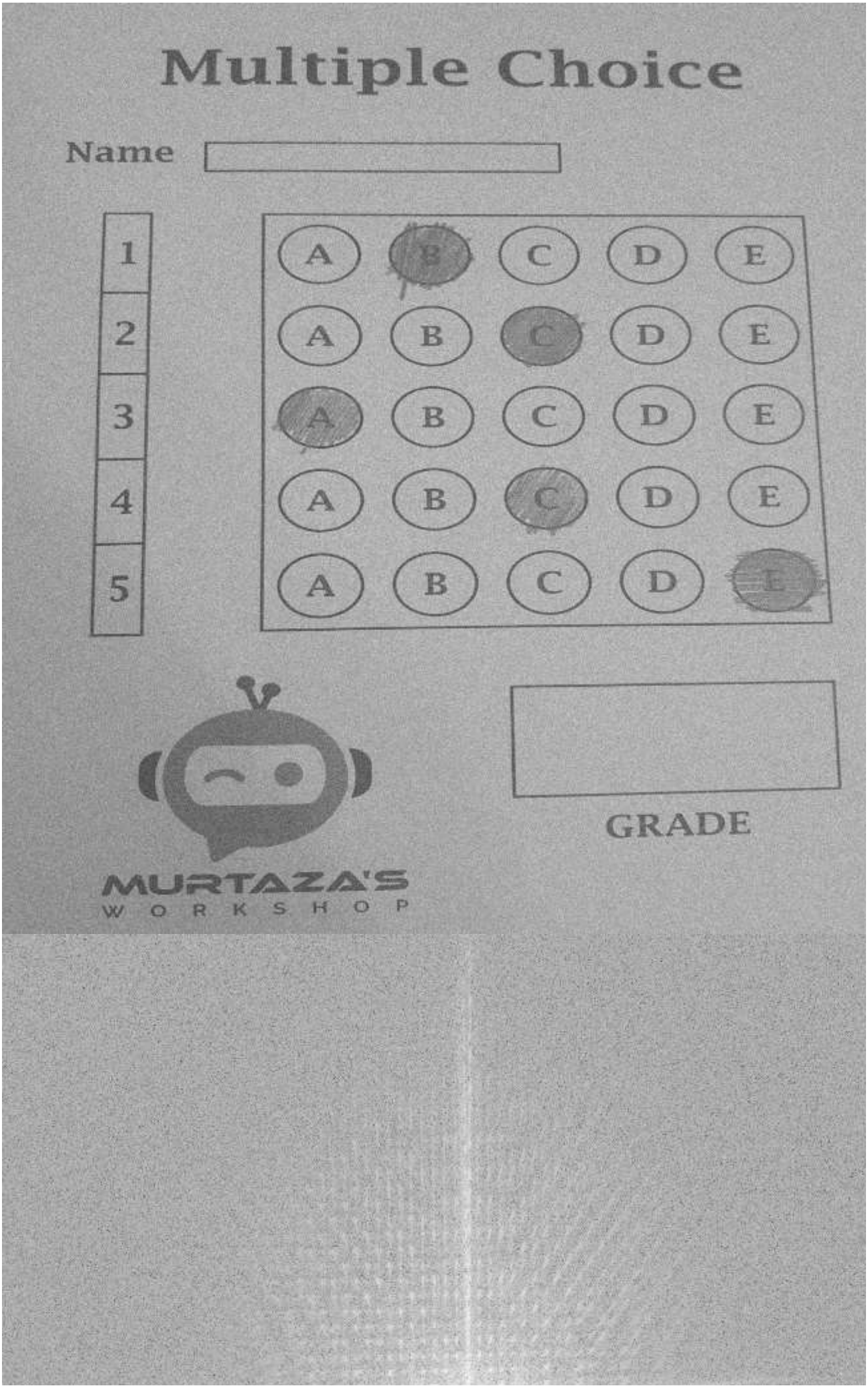
```
dft = cv2.dft(np.float32(imgNoisy),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

cv2_imshow(imgNoisy)
cv2_imshow(magnitude_spectrum)

rows, cols = img.shape
crow,ccol = rows//2 , cols//2
mask = np.zeros((rows,cols,2),np.float32)

d0=150

for i in range(rows):
  for j in range(cols):
    duv=((crow-i)**2+(ccol-j)**2)**0.5
    if duv<=d0:
      mask[i][j]=0
    else:
      mask[i][j]=1
# print(mask)
mask = 1 - mask
fshift = dft_shift*mask

cv2_imshow(20*np.log(cv2.magnitude(fshift[:,:,0],fshift[:,:,1])))

f_ishift = np.fft.ifftshift(fshift)
```

```
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
img_back = img_back*255 // img_back.max()
img_back = img_back.astype(np.uint8)

cv2_imshow(img_back)
sum = 0.0
for x in range(widthImg):
   for y in range(heightImg):
      difference = (int(img[x,y]) - int(img_back[x,y]))
      sum = sum + difference*difference
ideal_mse = sum /(widthImg*heightImg)
print("The mean square error is:",ideal_mse)
```


Multiple Choice
Name
1
2
3
4
5
A B C D E
A B C D E
A B C D E
A B C D E
A B C D E
MURTAZA'S
WORKSHOP
GRADE

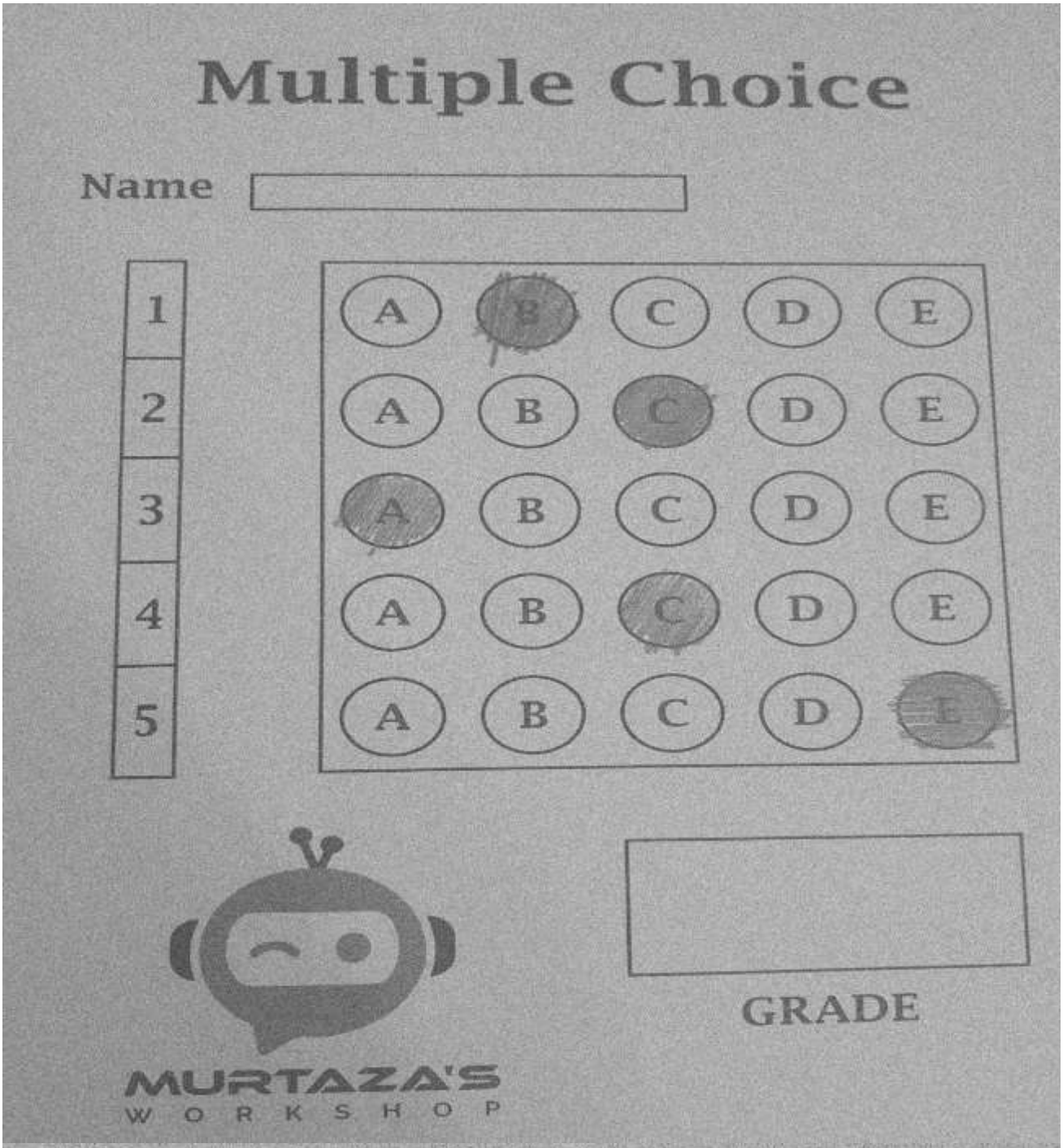
Butterworth Low Pass Filter

```
dft = cv2.dft(np.float32(imgNoisy),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

cv2_imshow(imgNoisy)
cv2_imshow(magnitude_spectrum)

rows, cols = img.shape
crow,ccol = rows//2 , cols//2
mask = np.zeros((rows,cols,2),np.float32)

d0,n=150,1

for i in range(rows):
  for j in range(cols):
    mask[i][j]=(1+(((crow-i)**2+(ccol-j)**2)**0.5/d0)**(2*n))**-0.5
# mask = 1 - mask
fshift = dft_shift*mask

cv2_imshow(20*np.log(cv2.magnitude(fshift[:,:,0],fshift[:,:,1])))

f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
img_back = img_back*255 // img_back.max()
img_back = img_back.astype(np.uint8)

cv2_imshow(img_back)
sum = 0.0
for x in range(widthImg):
   for y in range(heightImg):
      difference = (int(img[x,y]) - int(img_back[x,y]))
      sum = sum + difference*difference
butter_mse = sum /(widthImg*heightImg)
print("The mean square error is:",butter_mse)
```


Multiple Choice
Name
1
2
3
4
5
A B C D E
A B C D E
A B C D E
A B C D E
A B C D E
GRADE
MURTAZA'S
WORKSHOP

Gaussian Low Pass Filter

```
dft = cv2.dft(np.float32(imgNoisy),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

cv2_imshow(imgNoisy)
cv2_imshow(magnitude_spectrum)

rows, cols = img.shape
crow,ccol = rows//2 , cols//2
mask = np.zeros((rows,cols,2),np.float32)

sigma=100

for i in range(rows):
  for j in range(cols):
    mask[i][j]=math.exp(-((crow-i)**2+(ccol-j)**2)/(2*sigma**2))
# mask = 1 - mask
# print(mask)
fshift = dft_shift*mask

cv2_imshow(20*np.log(cv2.magnitude(fshift[:,:,0],fshift[:,:,1])))

f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
img_back = img_back*255 // img_back.max()
img_back = img_back.astype(np.uint8)

cv2_imshow(img_back)
sum = 0.0
for x in range(widthImg):
   for y in range(heightImg):
      difference = (int(img[x,y]) - int(img_back[x,y]))
      sum = sum + difference*difference
gaussian_mse = sum /(widthImg*heightImg)
print("The mean square error is:",gaussian_mse)
```


Multiple Choice
Name
1
2
3
4
5
A B C D E
A B C D E
A B C D E
A B C D E
A B C D E
GRADE
MURTAZA'S
WORKSHOP

```
errors = [["Mean filter",mean_mse],["Median filter",median_mse],["Ideal Low Pass filter",i
analysis = pd.DataFrame(errors, columns = ["Filter","Mean Squared Error"])
analysis
```

	Filter	Mean Squared Error
0	Mean filter	7352.656024
1	Median filter	7162.338406
2	Ideal Low Pass filter	267.888506
3	Butterworth Low Pass filter	346.193647
4	Gaussian Low Pass filter	923.533957

Noise 2

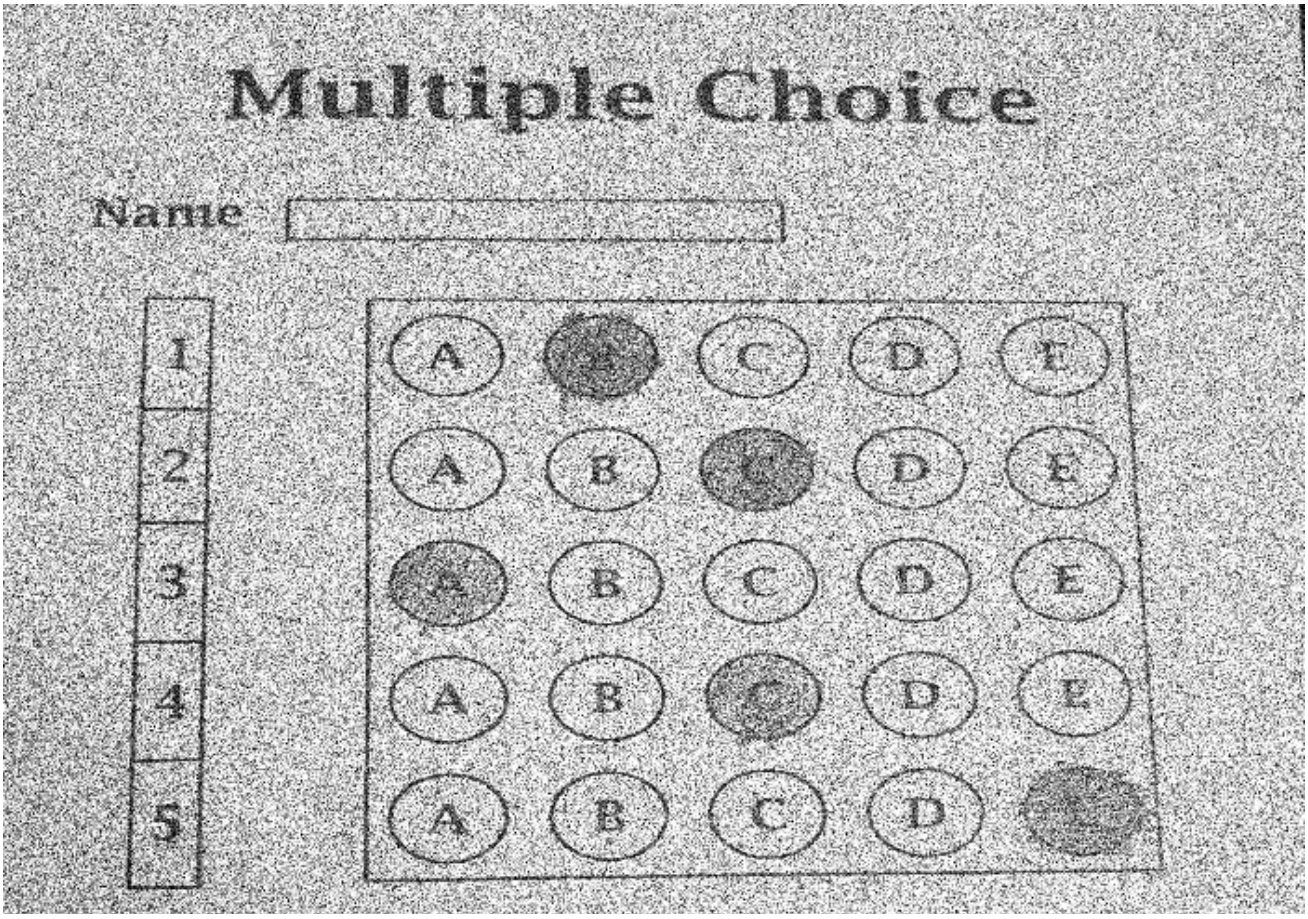
[ ] ↳ *14 cells hidden*

Noise 3

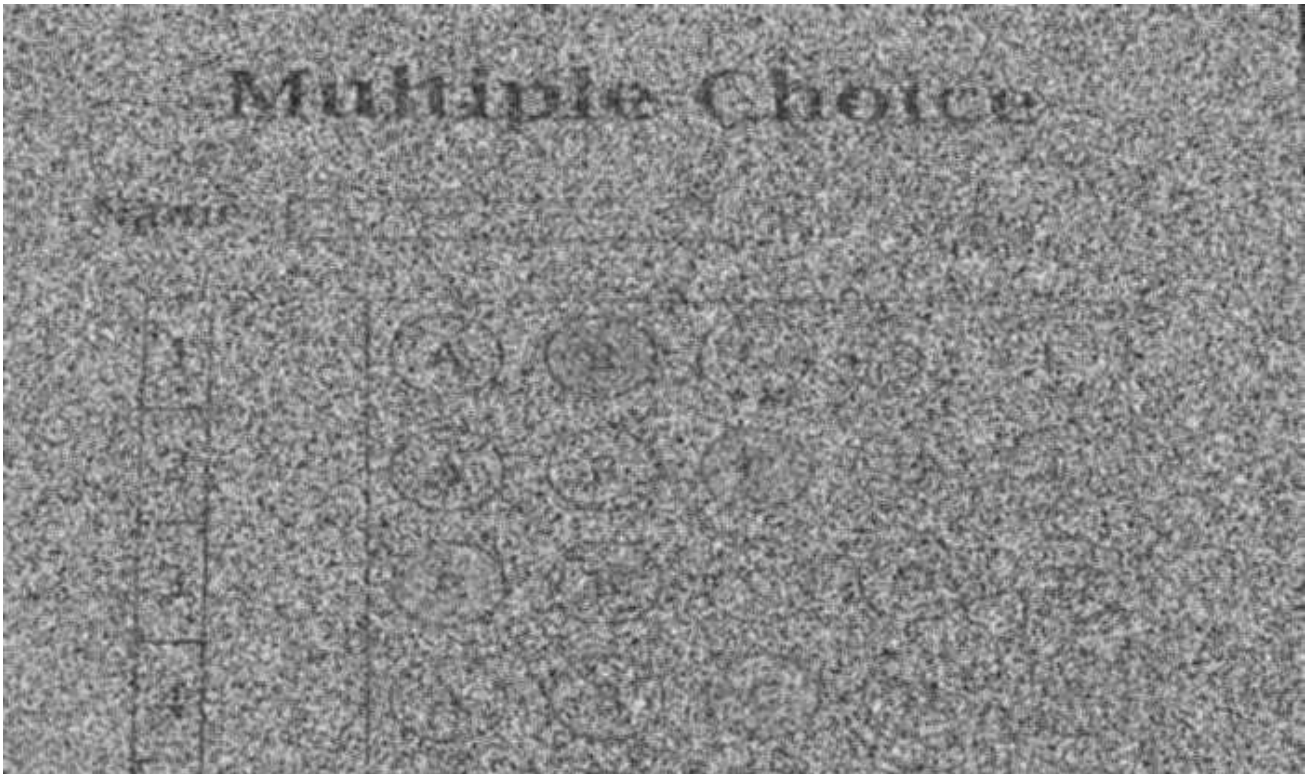
[ ] ↳ *13 cells hidden*

Noise 4

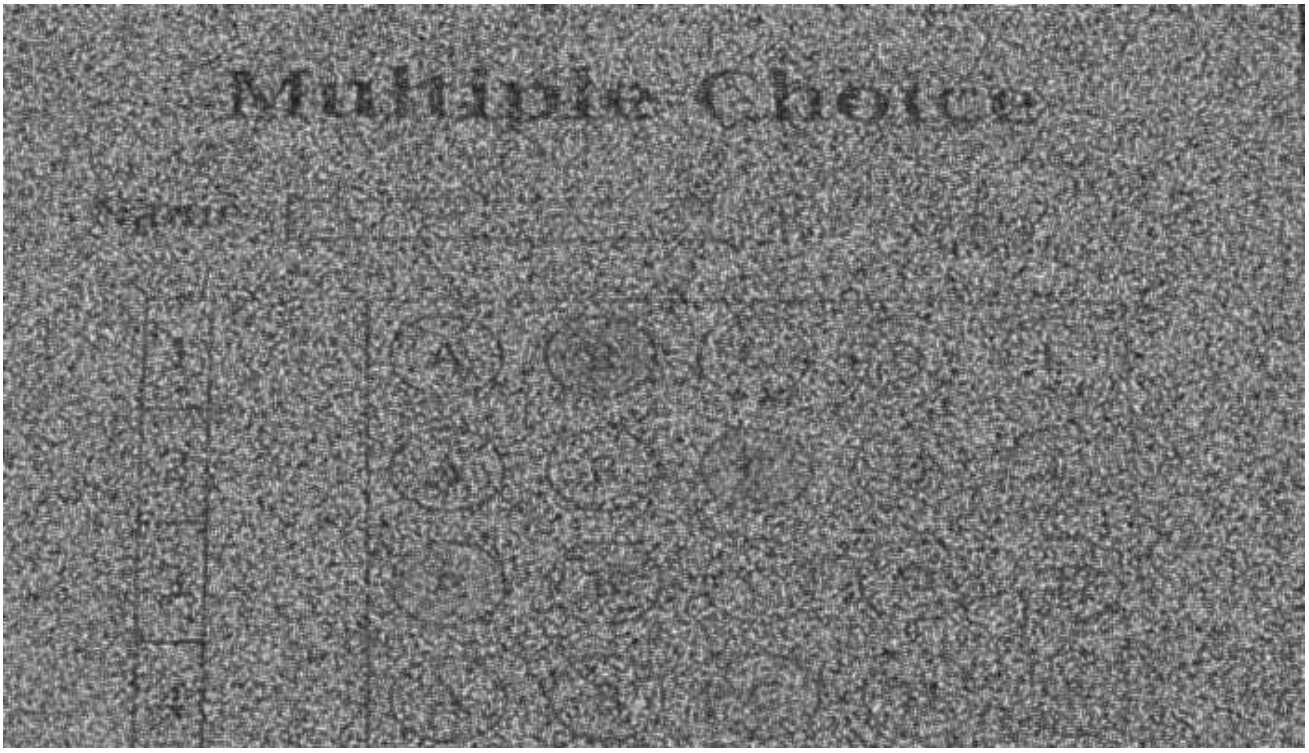
```
# -- sqrt(mu) * normal(0,1) --
poisson_noise = np.sqrt(img) * np.random.normal(0, 5, img.shape)

# Add the noise to the mu values
noisy_image = img + poisson_noise

cv2_imshow(noisy_image)
noisy_image = noisy_image.astype(np.uint8)
```



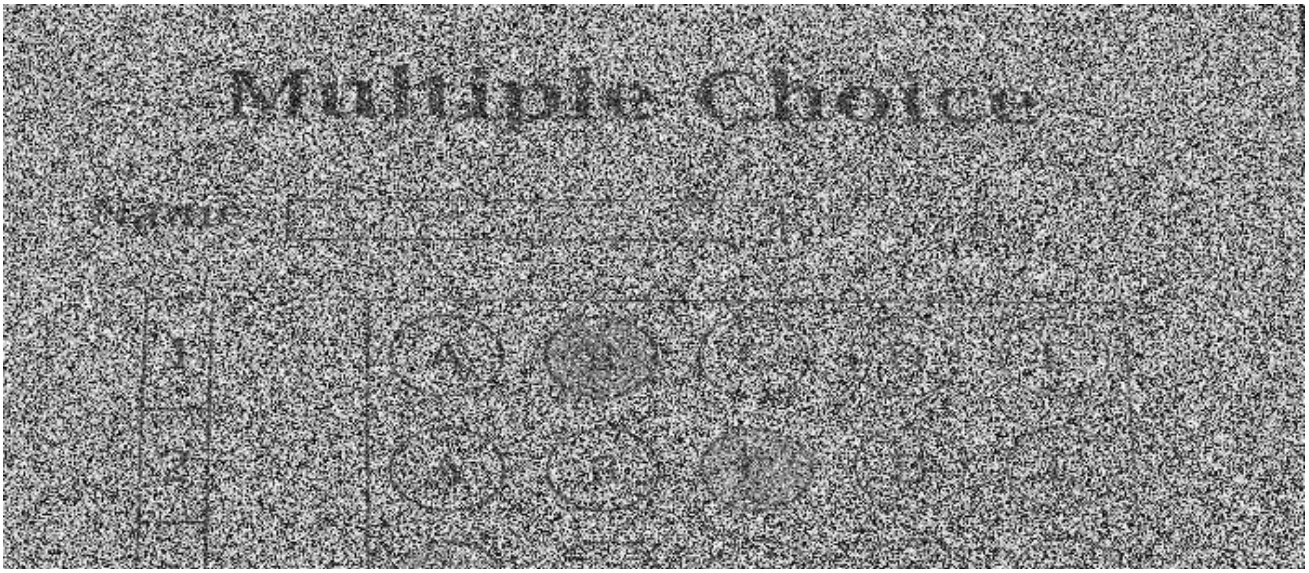

```
# Apply a Gaussian blur
blurred_image = cv2.GaussianBlur(noisy_image, (3, 3), 2)
cv2_imshow(blurred_image)

sum = 0.0
for x in range(widthImg):
   for y in range(heightImg):
      difference = (int(img[x,y]) - int(blurred_image[x,y]))
      sum = sum + difference*difference
gaussian_mse = sum /(widthImg*heightImg)
print("The mean square error is:",gaussian_mse)
```




```
lapl = np.array([[-1, -1, -1],[-1, 8, -1],[-1, -1, -1]])
laplacian = cv2.filter2D(src=blurred_image, ddepth=-1, kernel=lapl)

blurred_image = 0.8*blurred_image+0.2*laplacian
blurred_image = blurred_image.astype(np.uint8)
cv2_imshow(blurred_image)

sum = 0.0
for x in range(widthImg):
   for y in range(heightImg):
      difference = (int(img[x,y]) - int(blurred_image[x,y]))
      sum = sum + difference*difference
laplacian_mse = sum /(widthImg*heightImg)
print("The mean square error is:",laplacian_mse)
```




```
bilateral = cv2.bilateralFilter(noisy_image, 1, 1, 1)
cv2_imshow(bilateral)

sum = 0.0
for x in range(widthImg):
   for y in range(heightImg):
      difference = (int(img[x,y]) - int(bilateral[x,y]))
      sum = sum + difference*difference
bilateral_mse = sum /(widthImg*heightImg)
print("The mean square error is:",bilateral_mse)
```




```
errors = [["Gaussian filter",gaussian_mse],["Laplacian filter",laplacian_mse],["Bilateral
analysis = pd.DataFrame(errors, columns = ["Filter","Mean Squared Error"])
analysis
```

	Filter	Mean Squared Error
0	Gaussian filter	4993.274076
1	Laplacian filter	7942.905553
2	Bilateral filter	9733.442061

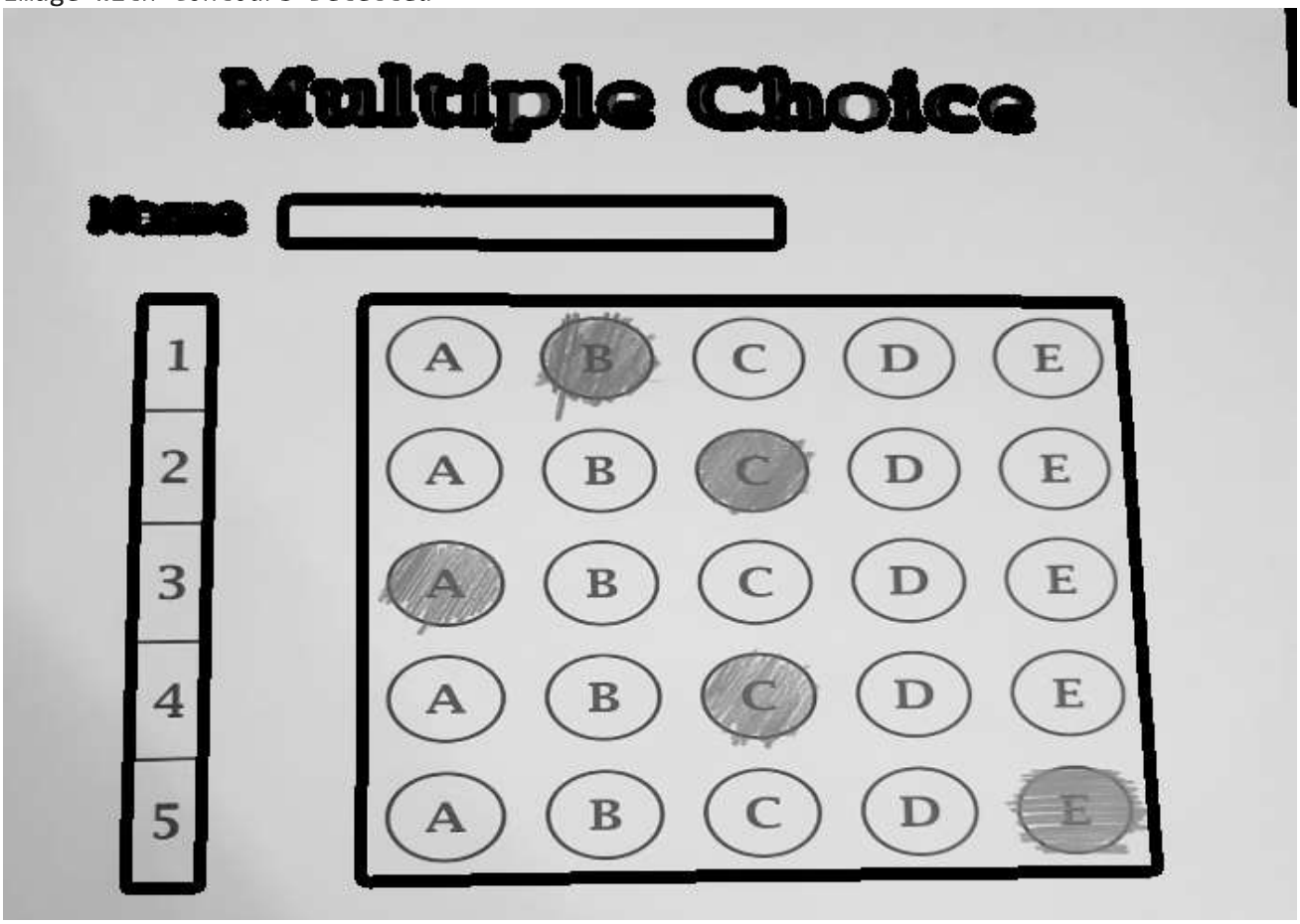
Segementation-1

Contours

```
The mean square error is: 9733.44206122449
```

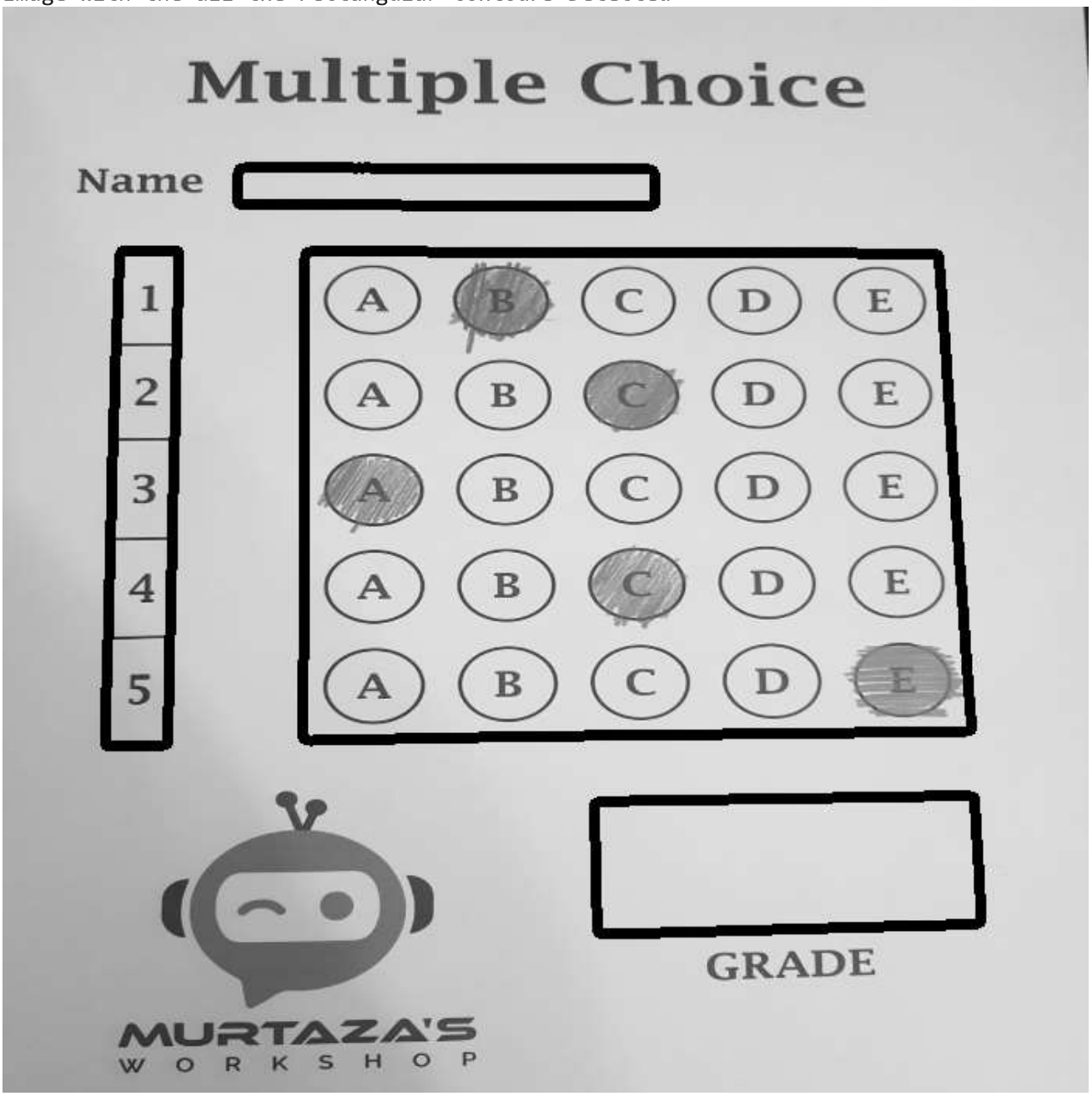
```
imgContours = img.copy()
contours, hierarchy = cv2.findContours(imgCanny, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
print("Image with Contours Detected")
cv2_imshow(cv2.drawContours(imgContours, contours, -1, (0, 0, 0), 5))
```

Image with Contours Detected




```
def rectContour(contours):
    rectCon = []
    max_area = 0
    for i in contours:
        area = cv2.contourArea(i)
        if area > 50:
            peri = cv2.arcLength(i, True)
            approx = cv2.approxPolyDP(i, 0.02 * peri, True)
            if len(approx) == 4:
                rectCon.append(i)
    rectCon = sorted(rectCon, key=cv2.contourArea,reverse=True)
    return rectCon

def getCornerPoints(cont):
    peri = cv2.arcLength(cont, True)
    approx = cv2.approxPolyDP(cont, 0.02 * peri, True)
    return approx


rectCon = rectContour(contours)
imgRectCon = img.copy()
print("Image with the all the rectangular contours Detected")
cv2_imshow(cv2.drawContours(imgRectCon, rectCon, -1, (0, 0, 0), 5))
biggestPoints= getCornerPoints(rectCon[0])
gradePoints = getCornerPoints(rectCon[1])
```

Image with the all the rectangular contours Detected

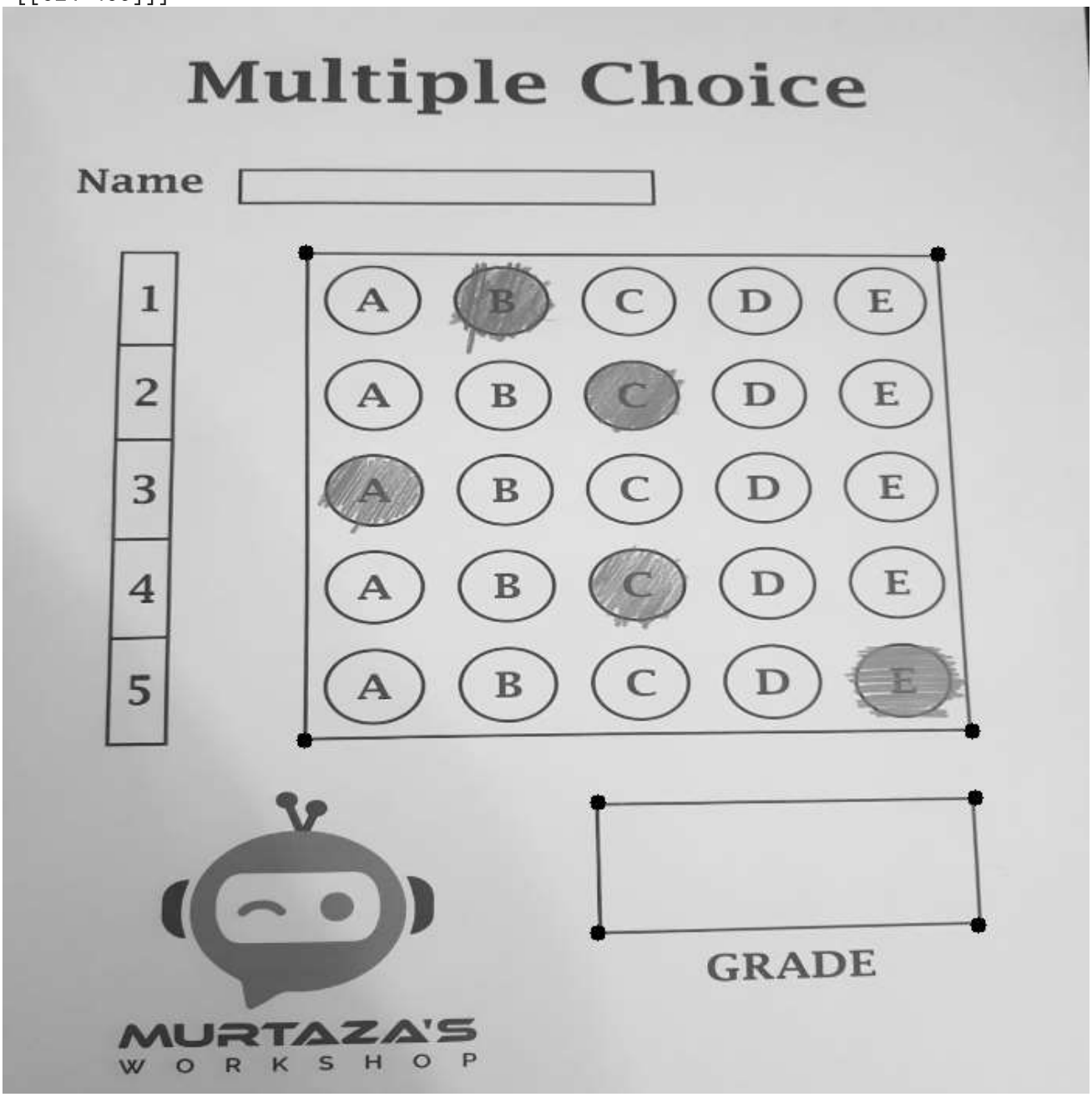


Corner Points

```
def reorder(myPoints):

    myPoints = myPoints.reshape((4, 2))
    myPointsNew = np.zeros((4, 1, 2), np.int32)
    add = myPoints.sum(1)
    myPointsNew[0] = myPoints[np.argmin(add)]
    myPointsNew[3] =myPoints[np.argmax(add)]
    diff = np.diff(myPoints, axis=1)
    myPointsNew[1] =myPoints[np.argmin(diff)]
    myPointsNew[2] = myPoints[np.argmax(diff)]

    return myPointsNew


biggestPoints = reorder(biggestPoints)
```

```
gradePoints = reorder(gradePoints)
imgBigContour = img.copy()
print("Image with the corner points of anwers section in the OMR detected")
print(biggestPoints)
cv2.drawContours(imgBigContour, biggestPoints, -1, (0, 0, 0), 10)
cv2.drawContours(imgBigContour, gradePoints, -1, (0, 0, 0), 10)
cv2_imshow(imgBigContour)
```

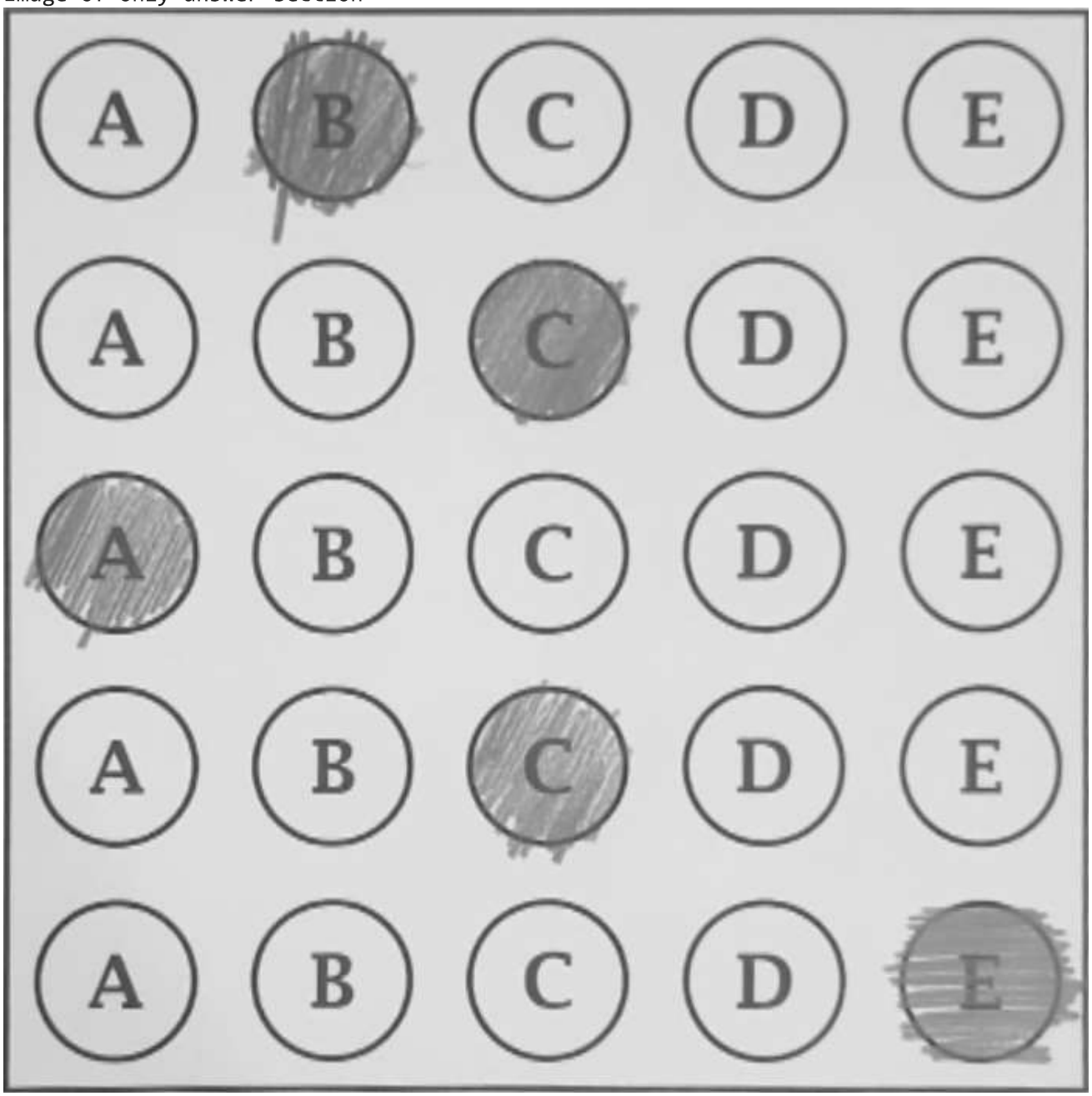
```
Image with the corner points of anwers section in the OMR detected
[[[195 158]]

 [[602 159]]

 [[194 472]]

 [[624 466]]]
```



Answer section

```
pts1 = np.float32(biggestPoints)
pts2 = np.float32([[0, 0],[widthImg, 0], [0, heightImg],[widthImg, heightImg]])
matrix = cv2.getPerspectiveTransform(pts1, pts2)
imgWarpColored = cv2.warpPerspective(imgColored, matrix, (widthImg*choices//5, heightImg*q
imgWarp = cv2.warpPerspective(img, matrix, (widthImg, heightImg))
print("Image of only answer section")
cv2_imshow(imgWarp)
```

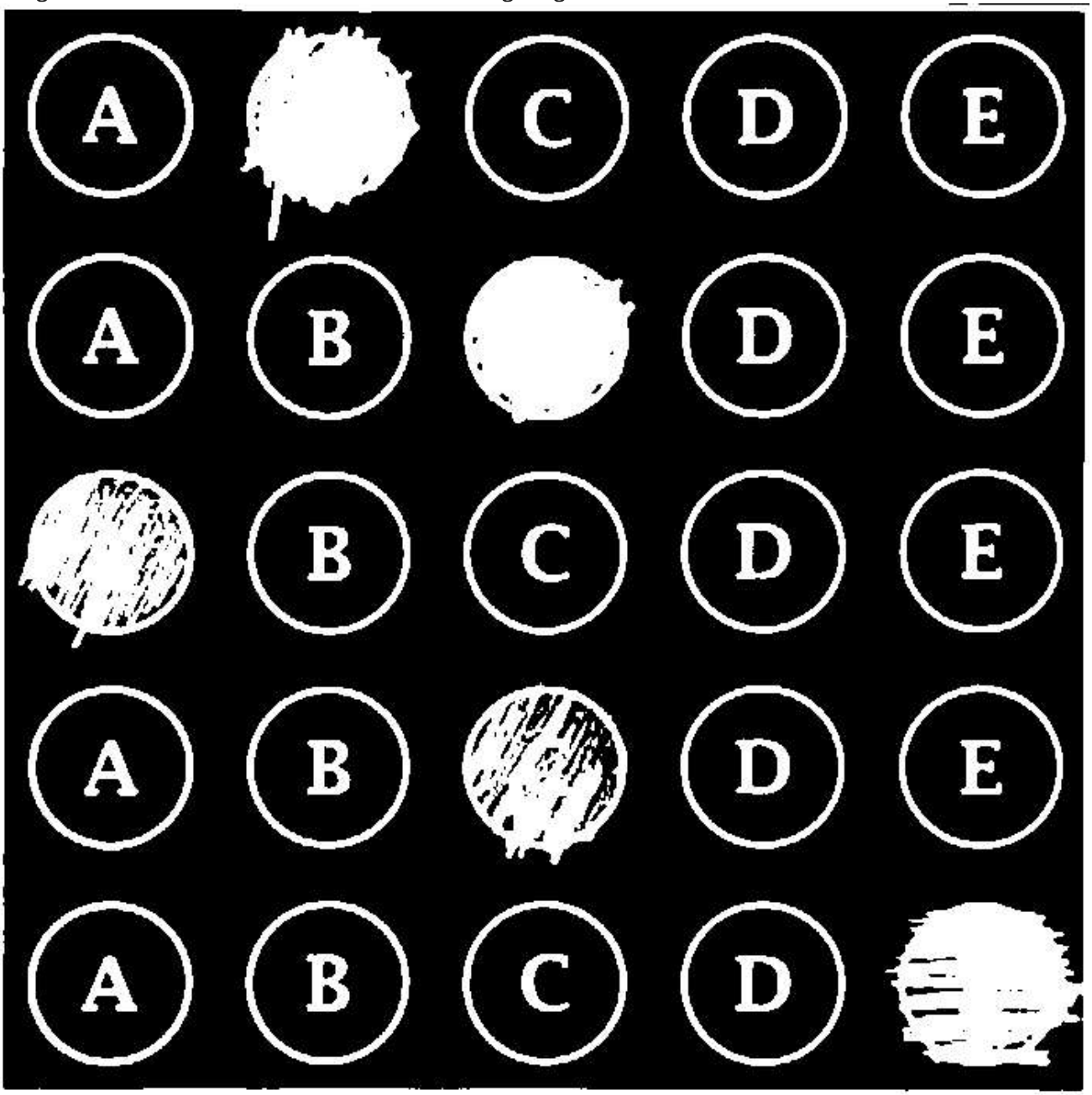
```
Image of only answer section
```




```
imgThresh = cv2.threshold(imgWarp, 0, 255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)[1]
print("Image is thresholded and inverted to highlight the bubbled answers")
```

```
# print(len(imgThresh))
cv2_imshow(imgThresh)
```

Image is thresholded and inverted to highlight the bubbled answers

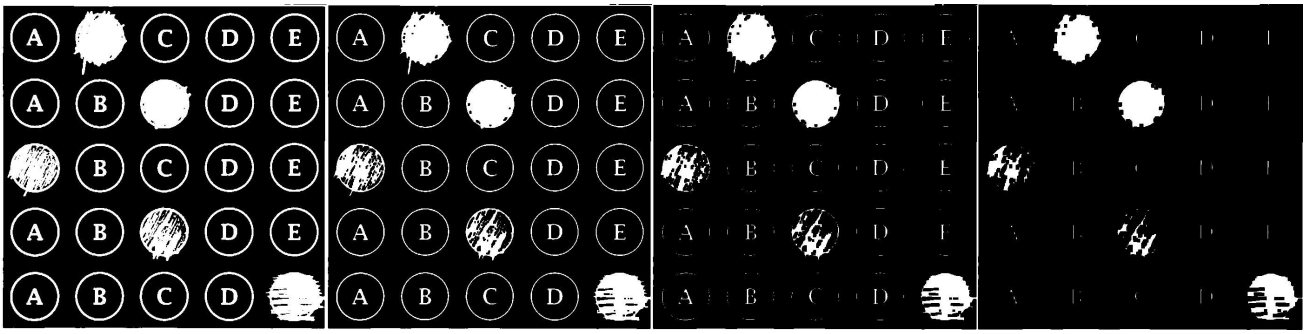


Morphological Processing

Erosion

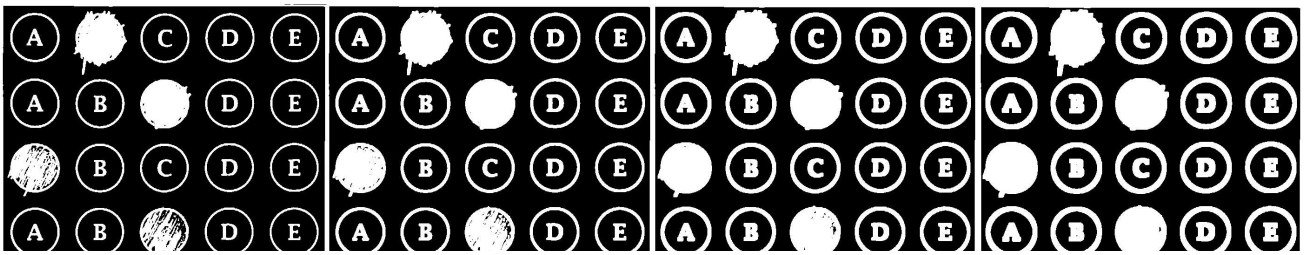
```
kernel1 = np.ones((3, 3), np.uint8)
erosion1 = cv2.erode(imgThresh, kernel1, iterations=1)

kernel2 = np.ones((5, 5), np.uint8)
erosion2 = cv2.erode(imgThresh, kernel2, iterations=1)
```

```
kernel3 = np.ones((7, 7), np.uint8)
erosion3 = cv2.erode(imgThresh, kernel3, iterations=1)

#cv2_imshow(erosion)
cv2_imshow(np.hstack((imgThresh, erosion1, erosion2, erosion3)))
```



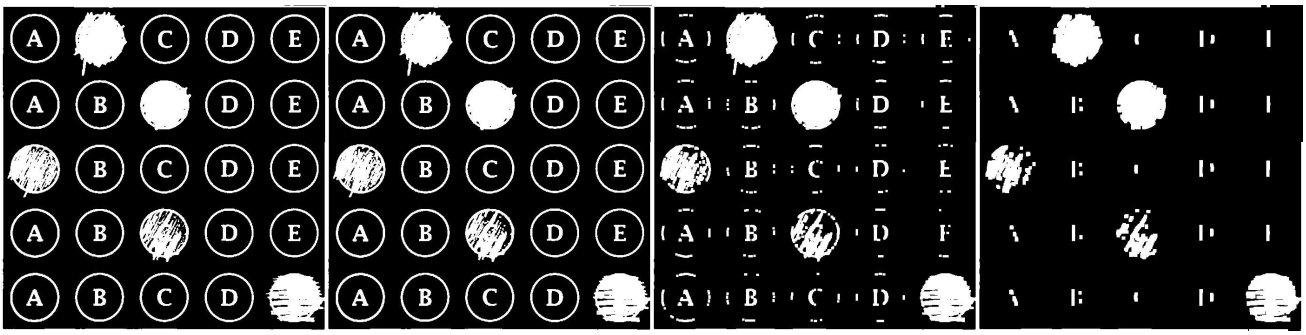
Dilation

```
kernel1 = np.ones((3, 3), np.uint8)
dilation1 = cv2.dilate(imgThresh, kernel1, iterations=1)

kernel2 = np.ones((5, 5), np.uint8)
dilation2 = cv2.dilate(imgThresh, kernel2, iterations=1)

kernel3 = np.ones((7, 7), np.uint8)
dilation3 = cv2.dilate(imgThresh, kernel3, iterations=1)

cv2_imshow(np.hstack((imgThresh, dilation1, dilation2, dilation3)))
```



Opening

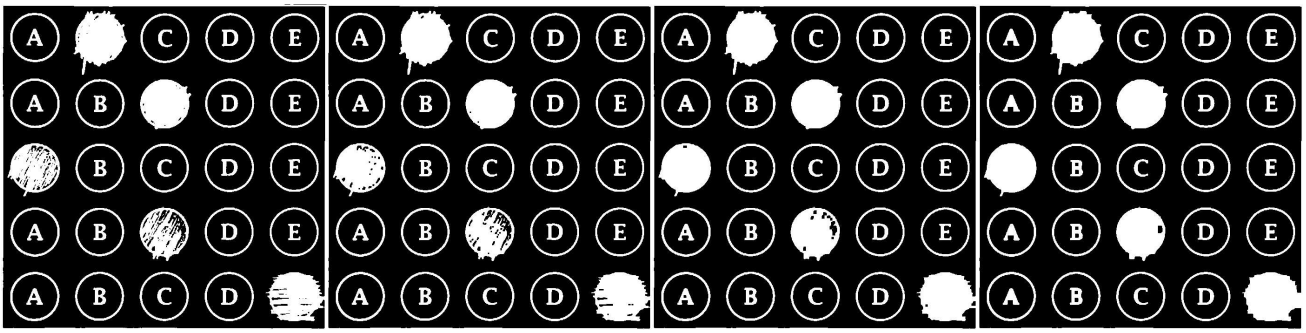
```
kernel1 = np.ones((3, 3), np.uint8)
opening1 = cv2.morphologyEx(imgThresh, cv2.MORPH_OPEN, kernel1, iterations=1)

kernel2 = np.ones((5, 5), np.uint8)
opening2 = cv2.morphologyEx(imgThresh, cv2.MORPH_OPEN, kernel2, iterations=1)

kernel3 = np.ones((7, 7), np.uint8)
opening3 = cv2.morphologyEx(imgThresh, cv2.MORPH_OPEN, kernel3, iterations=1)

cv2_imshow(np.hstack((imgThresh, opening1, opening2, opening3)))
```



Closing

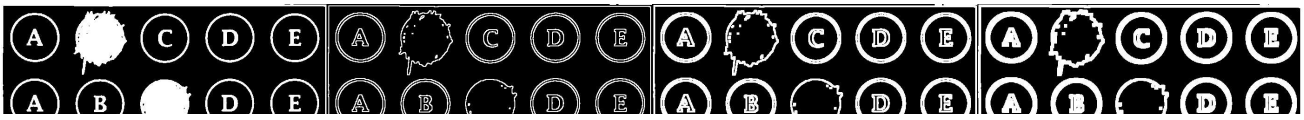
```
kernel1 = np.ones((3, 3), np.uint8)
closing1 = cv2.morphologyEx(imgThresh, cv2.MORPH_CLOSE, kernel1, iterations=1)

kernel2 = np.ones((5, 5), np.uint8)
closing2 = cv2.morphologyEx(imgThresh, cv2.MORPH_CLOSE, kernel2, iterations=1)

kernel3 = np.ones((7, 7), np.uint8)
closing3 = cv2.morphologyEx(imgThresh, cv2.MORPH_CLOSE, kernel3, iterations=1)
```

```
# print the output
cv2_imshow(np.hstack((imgThresh, closing1, closing2, closing3)))
```



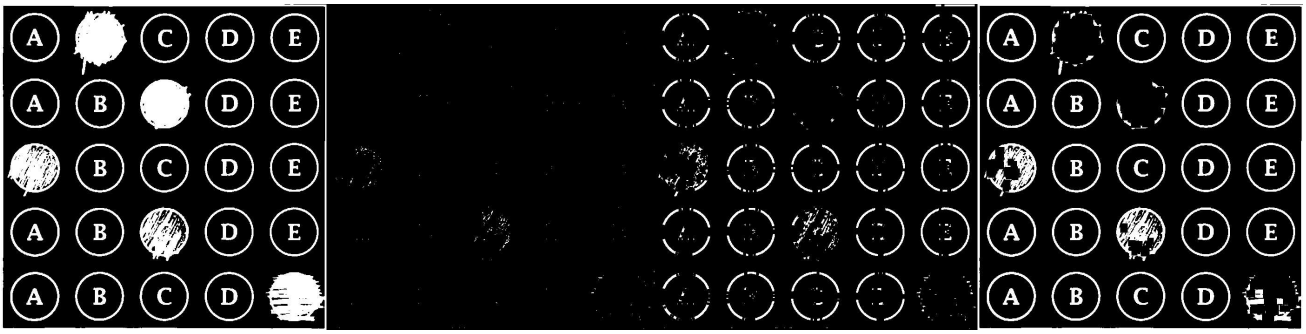
Morphological Gradient

```
kernel1 = np.ones((3, 3), np.uint8)
morph_gradient1 = cv2.morphologyEx(imgThresh, cv2.MORPH_GRADIENT, kernel1)

kernel2 = np.ones((5, 5), np.uint8)
morph_gradient2 = cv2.morphologyEx(imgThresh, cv2.MORPH_GRADIENT, kernel2)

kernel3 = np.ones((7, 7), np.uint8)
morph_gradient3 = cv2.morphologyEx(imgThresh, cv2.MORPH_GRADIENT, kernel3)

cv2_imshow(np.hstack((imgThresh, morph_gradient1, morph_gradient2, morph_gradient3)))
```



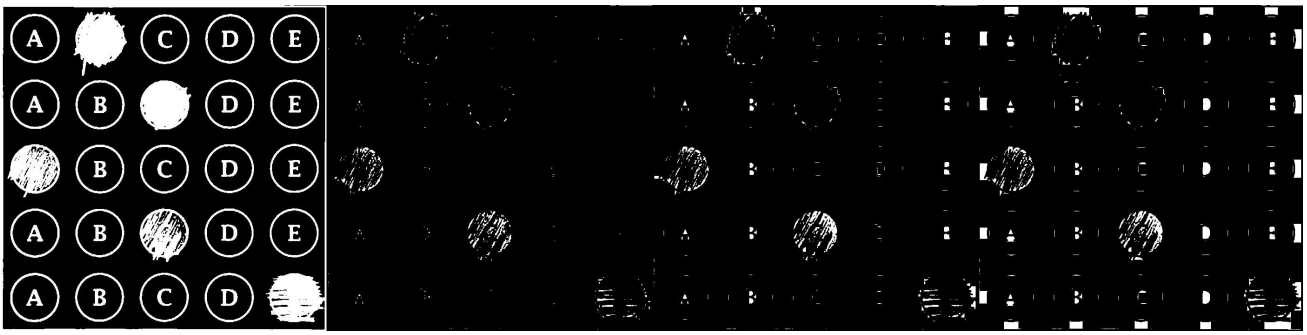
Top Hat

```
kernel1 = np.ones((3, 3), np.uint8)
th1 = cv2.morphologyEx(imgThresh, cv2.MORPH_TOPHAT, kernel1)

kernel2 = np.ones((5, 5), np.uint8)
th2 = cv2.morphologyEx(imgThresh, cv2.MORPH_TOPHAT, kernel2)

kernel3 = np.ones((13, 13), np.uint8)
th3 = cv2.morphologyEx(imgThresh, cv2.MORPH_TOPHAT, kernel3)

cv2_imshow(np.hstack((imgThresh, th1, th2, th3)))
```



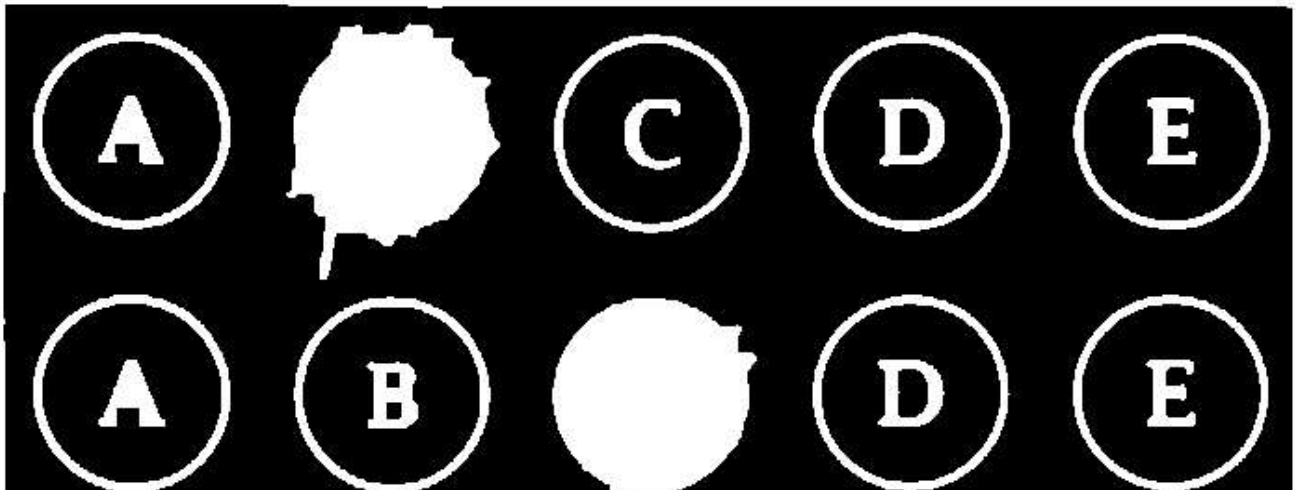
Black Hat

```
kernel1 = np.ones((5, 5), np.uint8)
bh1 = cv2.morphologyEx(imgThresh, cv2.MORPH_BLACKHAT, kernel1)

kernel2 = np.ones((11, 11), np.uint8)
bh2 = cv2.morphologyEx(imgThresh, cv2.MORPH_BLACKHAT, kernel2)

kernel3 = np.ones((17, 17), np.uint8)
bh3 = cv2.morphologyEx(imgThresh, cv2.MORPH_BLACKHAT, kernel3)

# print the output
cv2_imshow(np.hstack((imgThresh, bh1, bh2, bh3)))
```

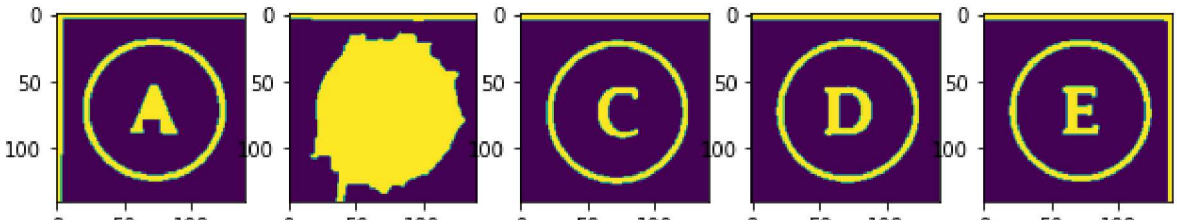


Options

```
kernel = np.ones((7, 7), np.uint8)
imgThresh = cv2.morphologyEx(imgThresh, cv2.MORPH_CLOSE, kernel, iterations=1)
cv2_imshow(imgThresh)
```




```
def splitBoxes(img):
    rows = np.vsplit(img,questions)
    boxes=[]
    for r in rows:
        cols= np.hsplit(r,choices)
        temp=[]
        for box in cols:
            temp.append(box)
        boxes.append(temp)
    return boxes
boxes = splitBoxes(imgThresh)
fig, ax = plt.subplots(questions,choices, figsize=(10, 10))
for i in range(questions):
    for j in range(choices):
        ax[i][j].imshow(boxes[i][j])
```



```
white_pixels = np.zeros((questions,choices))
def count_pixels(image):
    count=0
    for i in range(len(image)):
        for j in range(len(image[0])):
            if image[i][j]==255:
                count+=1
    return count
for x in range(questions):
    for y in range(choices):
        white_pixels[x][y] = count_pixels(boxes[x][y])
print(white_pixels)
```

```
 [[ 3445. 10573.  2703.  2957.  3199.]
  [ 2874.  2293.  8910.  2168.  2423.]
  [ 9729.  2178.  1946.  2150.  2503.]
  [ 2899.  2163.  8754.  2093.  2485.]
  [ 3494.  2915.  2594.  2700. 11193.]]
```

```
answers = []
for x in range(questions):
    arr = white_pixels[x]
    myIndexVal = [[-1]] if np.amax(arr)-np.amin(arr)<1500 else np.where(arr == np.amax(arr
    answers.append(myIndexVal[0][0])
answers
```

```
 [1, 2, 0, 2, 4]
```

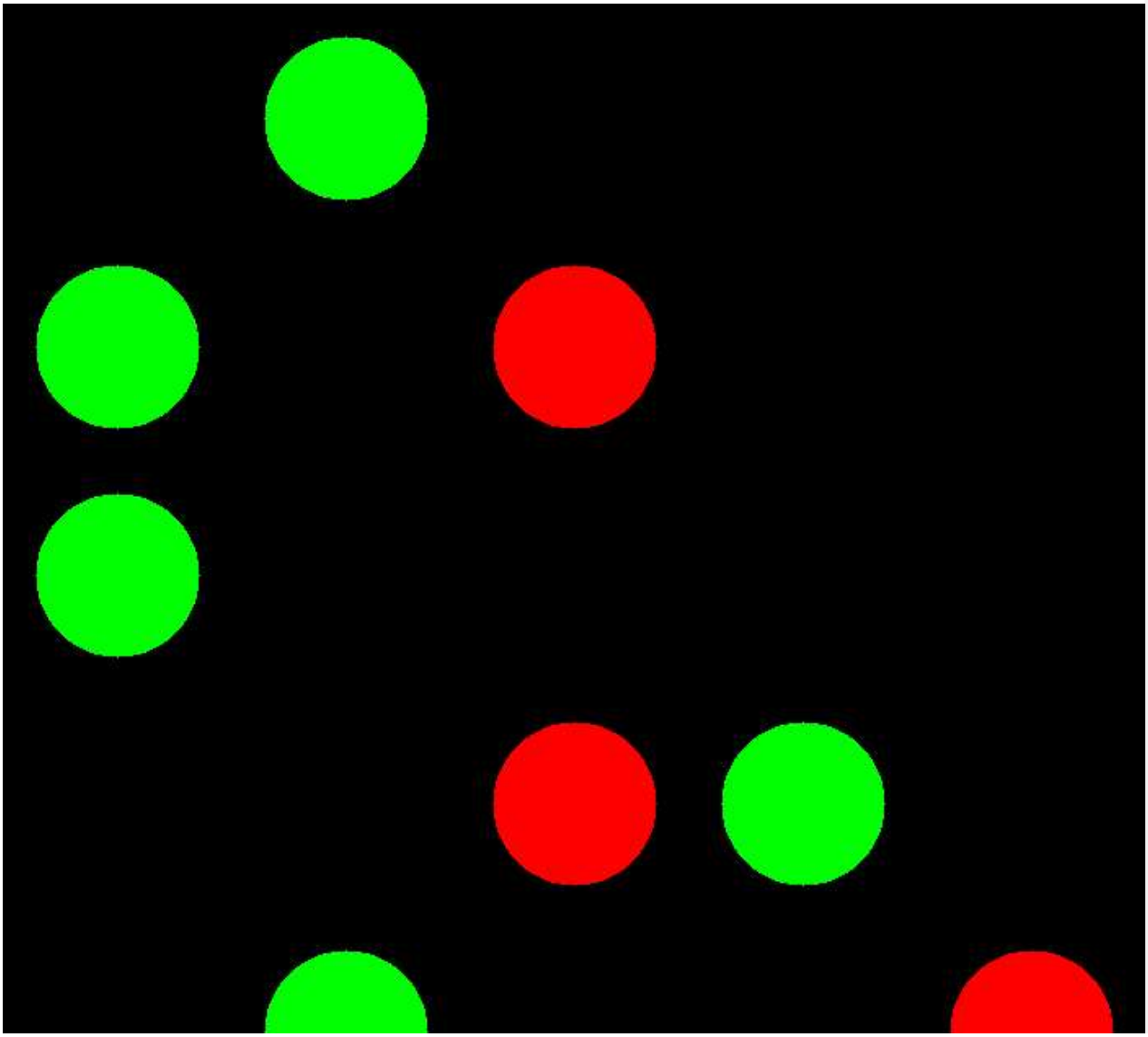
```
grading=[]
sum = 0
for x in range(questions):
    if solutions[x] == answers[x]:
        grading.append(1)
        sum += 1
    else:
        grading.append(0)
# print("GRADING",grading)
score = (sum/questions)*100
score
```

```
 40.0
```

```
def showAnswers(img,myIndex,grading,ans,questions,choices):
    secW = int(img.shape[1]/questions)
    secH = int(img.shape[0]/choices)
```

```
        for x in range(0,questions):
            myAns = myIndex[x]
            cX = (myAns * secW) + secW // 2
            cY = (x * secH) + secH // 2
            if grading[x]==1:
               myColor = (0,255,0)
               #cv2.rectangle(img,(myAns*secW,x*secH),((myAns*secW)+secW,(x*secH)+secH),myCol
               cv2.circle(img,(cX,cY),50,myColor,cv2.FILLED)
            else:
               myColor = (0,0,255)
               #cv2.rectangle(img, (myAns * secW, x * secH), ((myAns * secW) + secW, (x * sec
               cv2.circle(img, (cX, cY), 50, myColor, cv2.FILLED)

               # CORRECT ANSWER
               myColor = (0, 255, 0)
               correctAns = ans[x]
               cv2.circle(img,((correctAns * secW)+secW//2, (x * secH)+secH//2),50,myColor,cv

    showAnswers(imgWarpColored,answers,grading,solutions,questions,choices)
    cv2_imshow(imgWarpColored)
```

```
imgRawDrawings = np.zeros_like(imgWarpColored)
showAnswers(imgRawDrawings, answers, grading, solutions,questions,choices)
cv2_imshow(imgRawDrawings)
invMatrix = cv2.getPerspectiveTransform(pts2, pts1)
imgInvWarp = cv2.warpPerspective(imgRawDrawings, invMatrix, (widthImg, heightImg))
cv2_imshow(imgInvWarp)
```



Segmentaion-2

Grade Points

```
ptsG1 = np.float32(gradePoints)
ptsG2 = np.float32([[0, 0], [325, 0], [0, 150], [325, 150]])
matrixG = cv2.getPerspectiveTransform(ptsG1, ptsG2)
imgGradeDisplay = cv2.warpPerspective(imgColored, matrixG, (325, 150))
cv2_imshow(imgGradeDisplay)
```

```
imgRawGrade = np.zeros_like(imgGradeDisplay,np.uint8)
print(score)
cv2.putText(imgRawGrade,str(int(score))+"%",(70,100),cv2.FONT_HERSHEY_COMPLEX,3,(255,255,0
cv2_imshow(imgRawGrade)
```

```
40.0
```


```
invMatrixG = cv2.getPerspectiveTransform(ptsG2, ptsG1)
imgInvGradeDisplay = cv2.warpPerspective(imgRawGrade, invMatrixG, (widthImg, heightImg))
cv2_imshow(imgInvGradeDisplay)
```

Final Result

```
imgFinal = imgColored.copy()
imgFinal = cv2.addWeighted(imgFinal, 1, imgInvWarp, 1,0)
imgFinal = cv2.addWeighted(imgFinal, 1, imgInvGradeDisplay, 1,0)
cv2_imshow(imgFinal)
```