In [9]:

```
                                        #Author:@Sirish Prabakar

def Signature_Extractor(colour_of_signature,x_coordinate,y_coordinate,signature_image,background_image):


        import numpy as np
        import cv2

        from PIL import Image
        from io import BytesIO
        import base64

        # Convert Image to Base64
        def im_2_b64(image):
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = Image.fromarray(image)
        buff = BytesIO()
        image.save(buff, format="JPEG")
        img_str = base64.b64encode(buff.getvalue())
        return img_str

        # Convert Base64 to Image
        def b64_2_img(data):
        buff = BytesIO(base64.b64decode(data))
        img=Image.open(buff)
        cv_image = cv2.cvtColor(np.asarray(img), cv2.COLOR_RGB2BGR)
        return cv_image

        #trial base 64 inputs: input64 for blue sign and input 65 for black sign, this is for testing only, actu


        #bg and input64 are inputs to this whole program
        #---------------------------------------------------------------------------------
        colour=colour_of_signature
        y_start=y_coordinate
        x_start=x_coordinate
        image = b64_2_img(signature_image)
        bg_img = b64_2_img(background_image)
        #---------------------------------------------------------------------------------
        if colour=='blue':
```

```
        result = image.copy()
        image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        lower = np.array([90, 38, 0])
        upper = np.array([145, 255, 255])
        mask = cv2.inRange(image, lower, upper)

        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
        opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterations=1)
        close = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel, iterations=2)

        cnts = cv2.findContours(close, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if len(cnts) == 2 else cnts[1]

        boxes = []
        for c in cnts:
            (x, y, w, h) = cv2.boundingRect(c)
            boxes.append([x,y, x+w,y+h])

        boxes = np.asarray(boxes)
        left = np.min(boxes[:,0])
        top = np.min(boxes[:,1])
        right = np.max(boxes[:,2])
        bottom = np.max(boxes[:,3])

        result[close==0] = (255,255,255)
        ROI = result[top:bottom, left:right].copy()
        #cv2.rectangle(result, (left,top), (right,bottom), (36, 255, 12), 2)

        hsv = cv2.cvtColor(ROI, cv2.COLOR_BGR2HSV)

        # define range of HSV-color of the signature
        lower_val = np.array([90, 38, 0])
        upper_val = np.array([145, 255, 255])


        # Threshold the HSV image to get a mask that holds the signature area
        mask = cv2.inRange(hsv, lower_val, upper_val)

        mask_inv= cv2.bitwise_not(mask)
        sign_masked = cv2.bitwise_and(ROI,ROI,mask=mask)

        # get the dimensions of the signature
        height, width = ROI.shape[:2]
```

```
#create a subimage of the area where the signature needs to go
placeToPutSign = bg_img[y_start:height+ y_start,x_start:width+x_start]
# exclude signature area
placeToPutSign_masked = cv2.bitwise_and(placeToPutSign, placeToPutSign, mask=mask_inv)
# add signature to subimage
placeToPutSign_joined = cv2.add(placeToPutSign_masked, sign_masked)

# put subimage over main image
bg_img[y_start:height+y_start,x_start:width+x_start] = placeToPutSign_joined
cv2.resize(bg_img,None,fx=0.3,fy=0.3)
# display image
cv2.imshow("result", bg_img)


#-----------------------------------------------------------------------------------
final_64=im_2_b64(bg_img)
print(final_64)
return final_64
#-----------------------------------------------------------------------------------



cv2.waitKey(0)
cv2.destroyAllWindows()

if colour=='black':

import numpy as np
import cv2
# Load image
sign=image
#bg_img.fill(255)
#Convert BGR to HSV
hsv = cv2.cvtColor(sign, cv2.COLOR_BGR2HSV)

# define range of HSV-color of the signature
lower_val = np.array([0,0,0])
upper_val = np.array([179,255,150])

# Threshold the HSV image to get a mask that holds the signature area
mask = cv2.inRange(hsv, lower_val, upper_val)
# create an opposite: a mask that holds the background area
```

```
        mask_inv= cv2.bitwise_not(mask)

        # create an image of the signature with background excluded
        sign_masked = cv2.bitwise_and(sign,sign,mask=mask)

        # get the dimensions of the signature
        height, width = sign.shape[:2]

        # create a subimage of the area where the signature needs to go
        placeToPutSign = bg_img[y_start:height+y_start,x_start:width+x_start]
        # exclude signature area
        placeToPutSign_masked = cv2.bitwise_and(placeToPutSign, placeToPutSign, mask=mask_inv)
        # add signature to subimage
        placeToPutSign_joined = cv2.add(placeToPutSign_masked, sign_masked)

        # put subimage over main image
        bg_img[y_start:height+y_start,x_start:width+x_start] = placeToPutSign_joined
        cv2.resize(bg_img,None,fx=0.3,fy=0.3)
        # display image
        cv2.imshow("result", bg_img)


        #-----------------------------------------------------------------------------------
        final_64=im_2_b64(bg_img)
        print(final_64)
        return final_64
        #-----------------------------------------------------------------------------------




        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

```
b'/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHRofHh0aHBwgJC4nICIsIxwcKDcpLDAxNDQ0Hy
c5PTgyPC4zNDL/2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjL/wA
ARCASwB4ADASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMU
EGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhI
WGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQ
EBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRCh
YkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsr
O0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwD0miiigBjxh+ehqAqVOD1q1SMobqKAK4
FOApSpU0AUASK3Y0+ogKepxQA+iiigApaKKAFApaSnDmgAoK5pacBQBHilp5XNNxigAooooAerY61IKhpytt+lAE1FAIIzS0AIyBvrURBBwa
sUhUMMGgCvS05kkkoAAcdKnR93B4NQ0tAFmlqOO0Ts351LQAVE8QjbcgfA2rhqaigCpS1O8XdrfyqaigCpS1O8XdrfyqaigCpS1O8QbkcGoSMHFABiiiloAcjlDxVhXDiqtOUlTkUAW6WmR
yB/ZqfQAhAYYNV3iKc9RVrFLigCl0keV6elRUAGKVSVORQOV6elRUAGKVSVORQYpaALMcgf2PpUlXWUcP7KVODQA2lRyMGQ1aAAUsEHkUVaeMP9fWq7KVODQA2lRyMGQ1aAAUs
8r+VQUAFPjkKcHlaZRigC2CGGRS1WRyh4qwjhxuWRyh4qwjhxuWRyh4qwjhxuWRyh4qwjhxuWRyh4qwjhxuWRyh4qwjhxuWRyh4qwjhxuWRy
```

In [ ]: