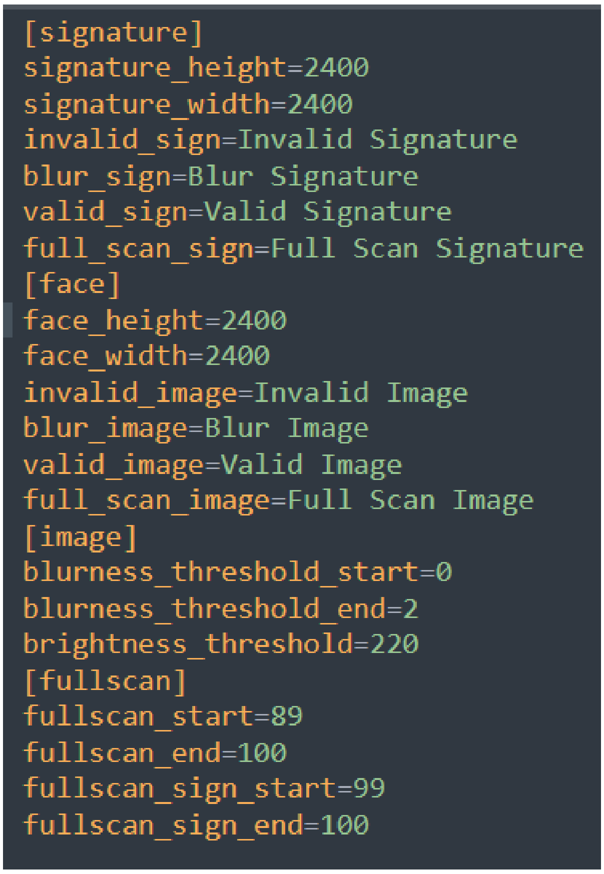
CGG Image Validation API Implementation Details

config.properties

[image] - **Blurriness threshold**	blurness_threshold_start=0
	blurness_threshold_end=2
	brightness_threshold=220

[fullscan] – **Full scan threshold**	fullscan_start=89
	fullscan_end=100
	fullscan_sign_start=99
	fullscan_sign_end=100



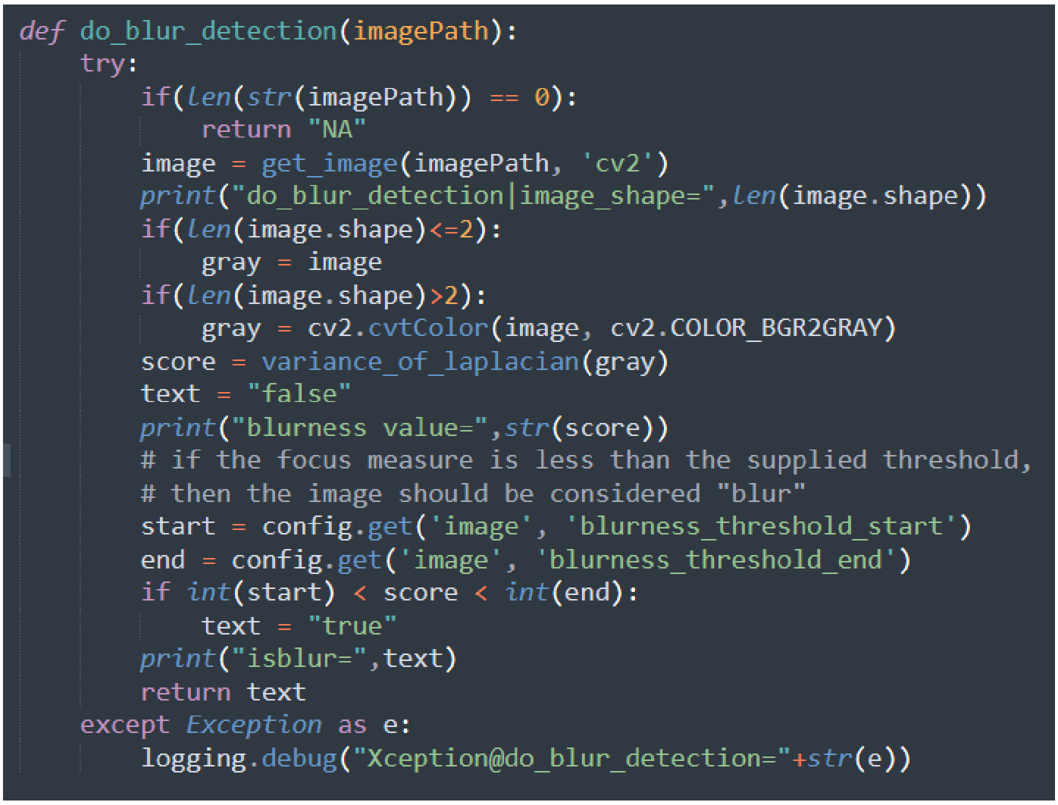
```
[signature]
signature_height=2400
signature_width=2400
invalid_sign=Invalid Signature
blur_sign=Blur Signature
valid_sign=Valid Signature
full_scan_sign=Full Scan Signature
[face]
face_height=2400
face_width=2400
invalid_image=Invalid Image
blur_image=Blur Image
valid_image=Valid Image
full_scan_image=Full Scan Image
[image]
blurness_threshold_start=0
blurness_threshold_end=2
brightness_threshold=220
[fullscan]
fullscan_start=89
fullscan_end=100
fullscan_sign_start=99
fullscan_sign_end=100
```

Blur Detection (face / signature):

A function is said to be a piecewise continuous function if it has a finite number of breaks and it does not blow up to infinity anywhere. Let us assume that the function f(t) is a piecewise continuous function, then f(t) is defined using the Laplace transform. The Laplace transform of a function is represented by L{f(t)} or F(s). Laplace transform helps to solve the differential equations, where it reduces the differential equation into an algebraic problem. It is the integral transform of the given derivative function with real variable t to convert into a complex function with variable s. For t ≥ 0, let f(t) be given and assume the function satisfies certain conditions to be stated later on.

Using open-cv we can calculate Laplace transform as mentioned below.

```
# Blur detection with OpenCV
score = cv2.Laplacian(image, cv2.CV_64F).var()
```



```
def do_blur_detection(imagePath):
    try:
        if(len(str(imagePath)) == 0):
            return "NA"
        image = get_image(imagePath, 'cv2')
        print("do_blur_detection|image_shape=",len(image.shape))
        if(len(image.shape)<=2):
            gray = image
        if(len(image.shape)>2):
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        score = variance_of_laplacian(gray)
        text = "false"
        print("blurness value=",str(score))
        # if the focus measure is less than the supplied threshold,
        # then the image should be considered "blur"
        start = config.get('image', 'blurness_threshold_start')
        end = config.get('image', 'blurness_threshold_end')
        if int(start) < score < int(end):
            text = "true"
        print("isblur=",text)
        return text
    except Exception as e:
        logging.debug("Xception@do_blur_detection="+str(e))
```

If the focus measure of the image falls within the certain threshold(blurness_threshold_start < score <blurness_threshold_end) as mentioned in the above properties, then image is considered as blurry.

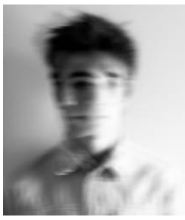


Fig 1. Blurry

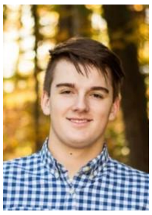


Fig 2. Non-Blurry

Images with high brightness / contrast also categorized as blurry. If the brightness measure is greater than or equal to 220 then that image also considered as blurry.

Brightness detection (face / signature):

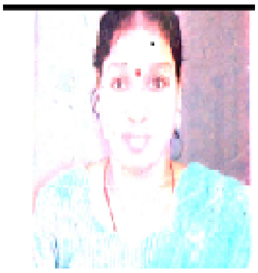
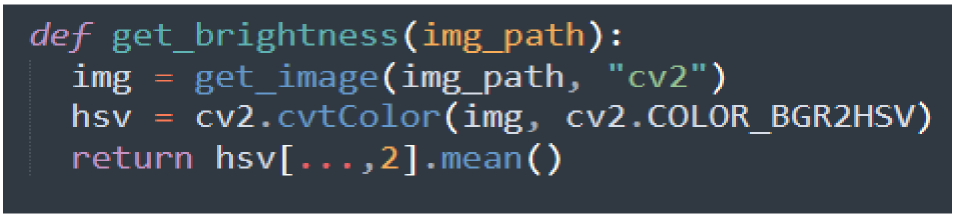
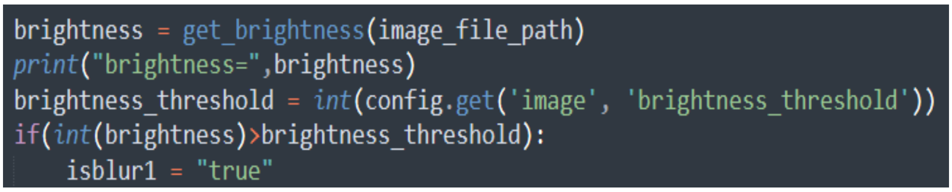


Fig 3. High brightness image

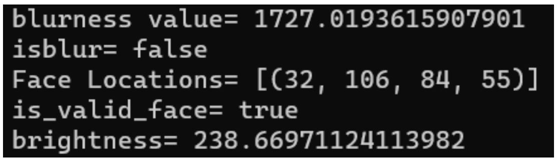


```
def get_brightness(img_path):
    img = get_image(img_path, "cv2")
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    return hsv[...,2].mean()
```



```
brightness = get_brightness(image_file_path)
print("brightness=",brightness)
brightness_threshold = int(config.get('image', 'brightness_threshold'))
if(int(brightness)>brightness_threshold):
    isblur1 = "true"
```

Explanation – Fig 3



```
blurness value= 1727.0193615907901
isblur= false
Face Locations= [(32, 106, 84, 55)]
is_valid_face= true
brightness= 238.66971124113982
```

Blurriness value of Fig 3 is 1727 which is not blurry and brightness is 238, as the brightness value is above 238 > brightness_threshold=220 and is categorized as blurry.

Note if image is blurry face detection will not happen.

Full Scan Image Detection (face / signature):

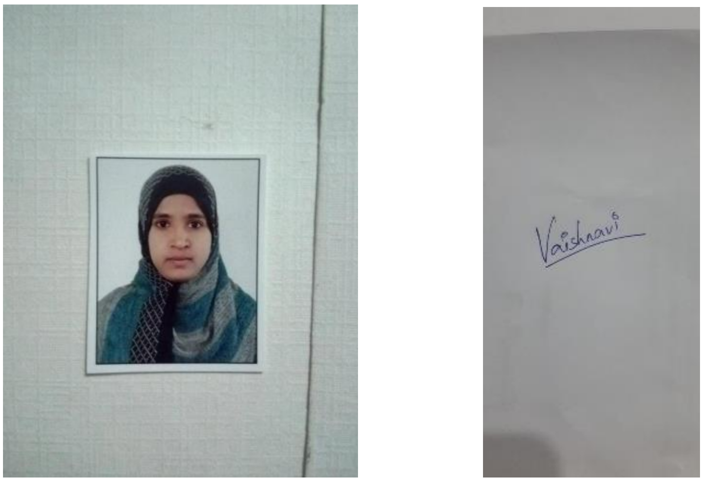
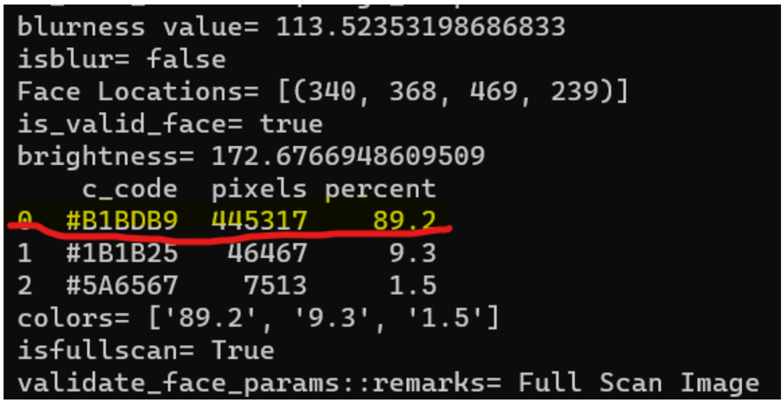
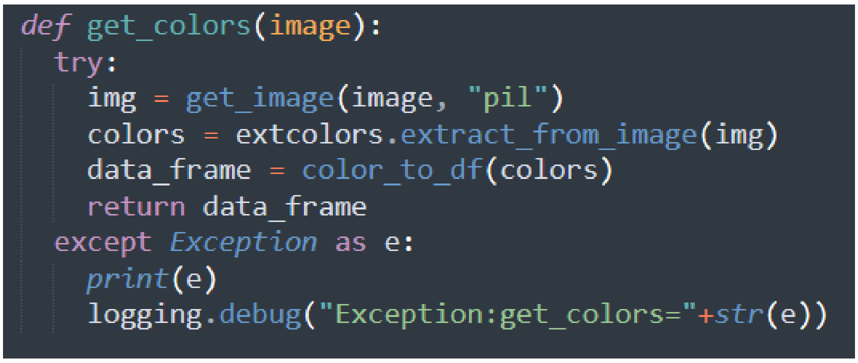


Fig 4. Full scan images sample

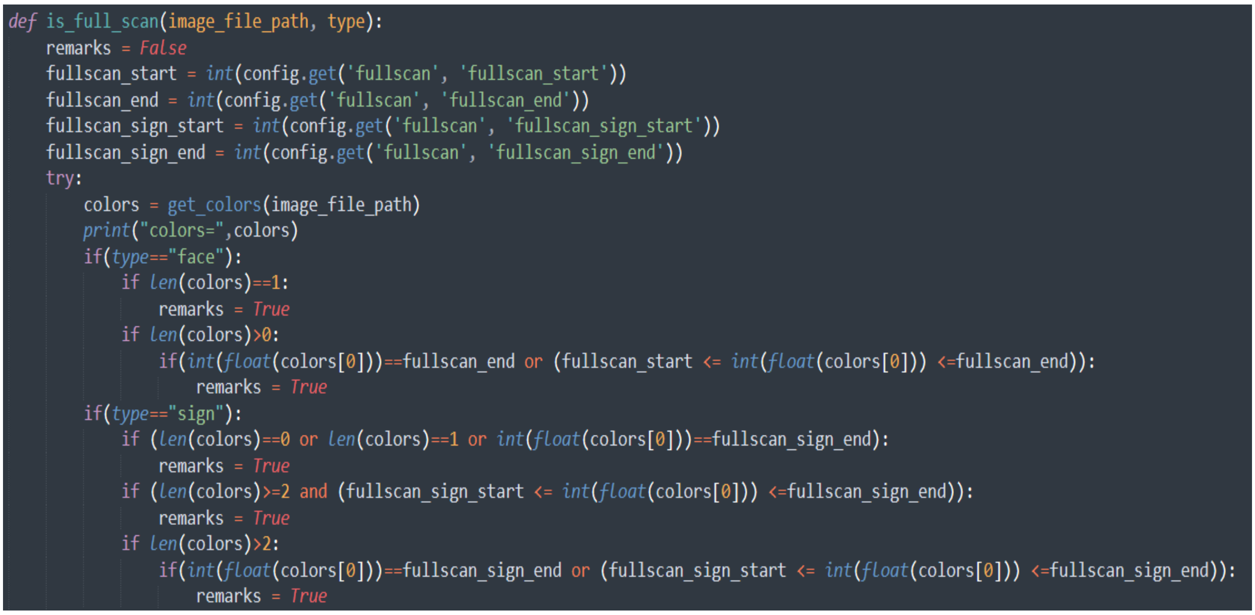
Explanation:



```
blurness value= 113.52353198686833
isblur= false
Face Locations= [(340, 368, 469, 239)]
is_valid_face= true
brightness= 172.6766948609509
    c_code  pixels percent
0  #B1BDB9  445317    89.2
1  #1B1B25   46467     9.3
2  #5A6567    7513     1.5
colors= ['89.2', '9.3', '1.5']
isfullscan= True
validate_face_params::remarks= Full Scan Image
```



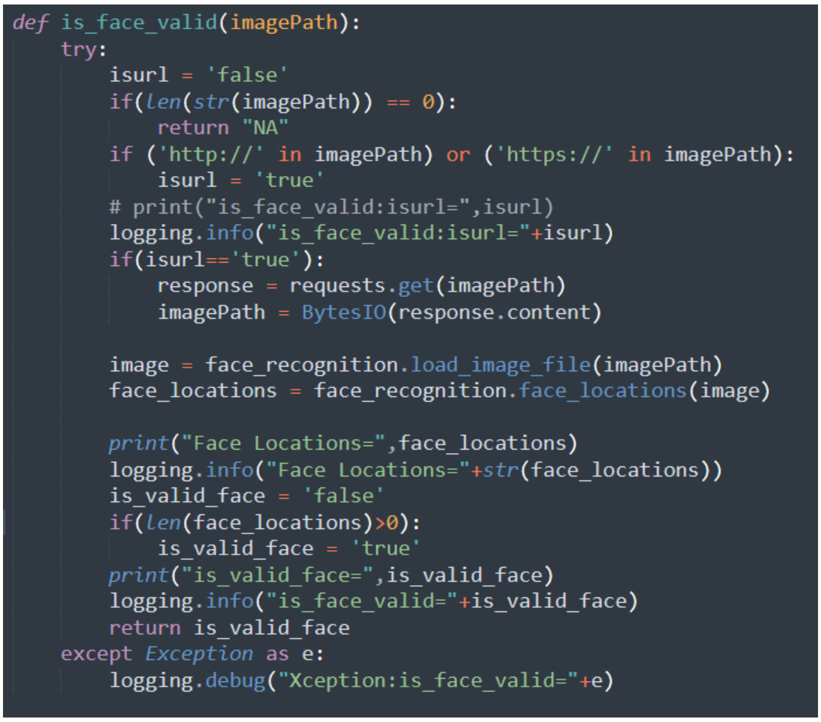
```
def get_colors(image):
    try:
        img = get_image(image, "pil")
        colors = extcolors.extract_from_image(img)
        data_frame = color_to_df(colors)
        return data_frame
    except Exception as e:
        print(e)
        logging.debug("Exception:get_colors="+str(e))
```



```
def is_full_scan(image_file_path, type):
    remarks = False
    fullscan_start = int(config.get('fullscan', 'fullscan_start'))
    fullscan_end = int(config.get('fullscan', 'fullscan_end'))
    fullscan_sign_start = int(config.get('fullscan', 'fullscan_sign_start'))
    fullscan_sign_end = int(config.get('fullscan', 'fullscan_sign_end'))
    try:
        colors = get_colors(image_file_path)
        print("colors=",colors)
        if(type=="face"):
            if len(colors)==1:
                remarks = True
            if len(colors)>0:
                if(int(float(colors[0]))==fullscan_end or (fullscan_start <= int(float(colors[0])) <=fullscan_end)):
                    remarks = True
        if(type=="sign"):
            if (len(colors)==0 or len(colors)==1 or int(float(colors[0]))==fullscan_sign_end):
                remarks = True
            if (len(colors)>=2 and (fullscan_sign_start <= int(float(colors[0])) <=fullscan_sign_end)):
                remarks = True
            if len(colors)>2:
                if(int(float(colors[0]))==fullscan_sign_end or (fullscan_sign_start <= int(float(colors[0])) <=fullscan_sign_end)):
                    remarks = True
```

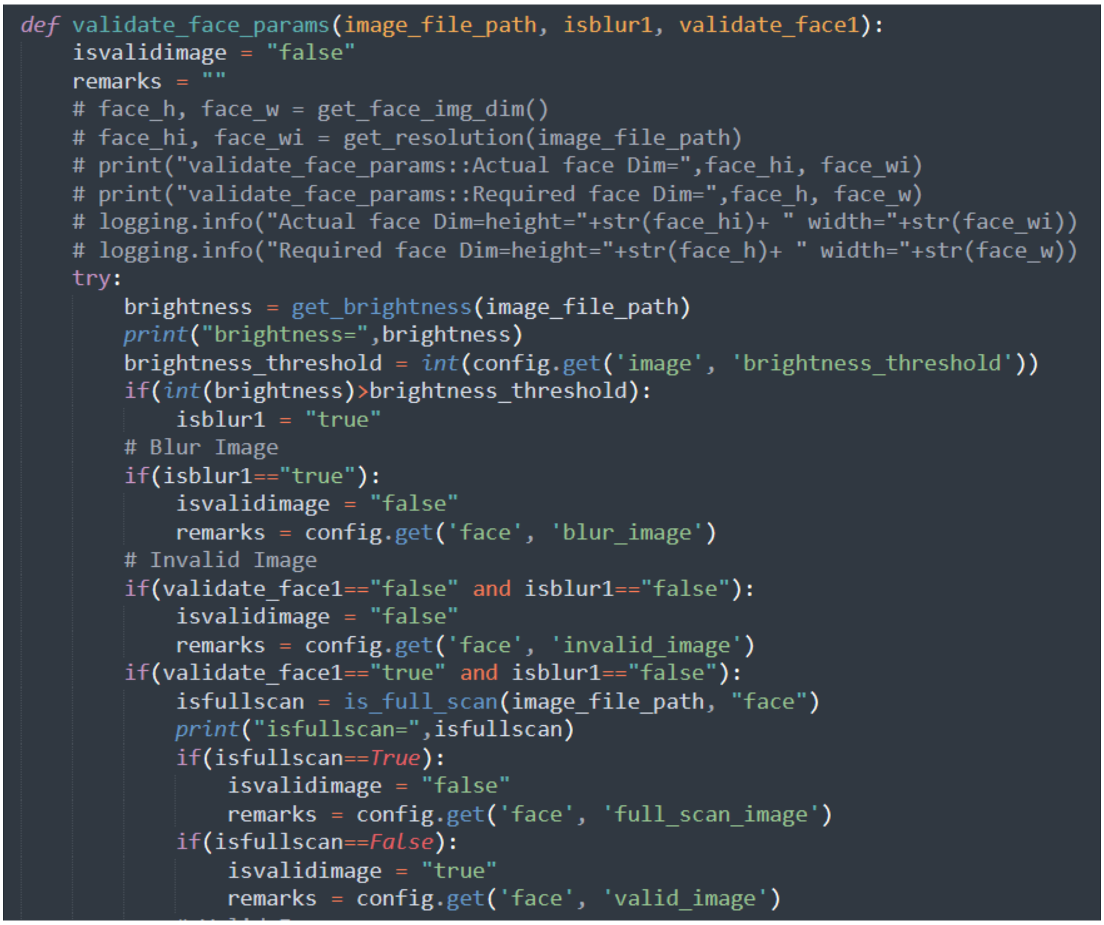
Full scan detection is made by applying color segmentation and sorting all the available colors in the image, if any color in the image is between 89%-100% then that image is considered as full scan image as per above result.

Face Detection:



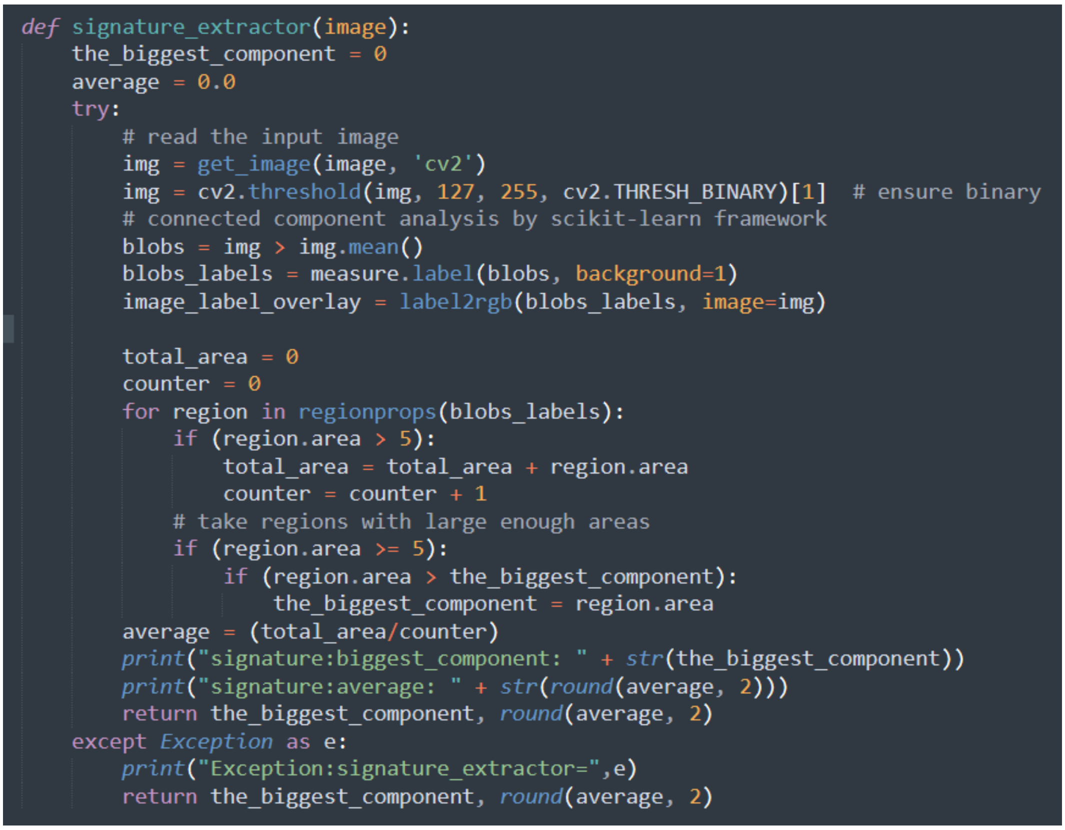
```
def is_face_valid(imagePath):
    try:
        isurl = 'false'
        if(len(str(imagePath)) == 0):
            return "NA"
        if ('http://' in imagePath) or ('https://' in imagePath):
            isurl = 'true'
        # print("is_face_valid:isurl=",isurl)
        logging.info("is_face_valid:isurl="+isurl)
        if(isurl=='true'):
            response = requests.get(imagePath)
            imagePath = BytesIO(response.content)

        image = face_recognition.load_image_file(imagePath)
        face_locations = face_recognition.face_locations(image)

        print("Face Locations=",face_locations)
        logging.info("Face Locations="+str(face_locations))
        is_valid_face = 'false'
        if(len(face_locations)>0):
            is_valid_face = 'true'
        print("is_valid_face=",is_valid_face)
        logging.info("is_face_valid="+is_valid_face)
        return is_valid_face
    except Exception as e:
        logging.debug("Xception:is_face_valid="+e)
```

Validation of face params:



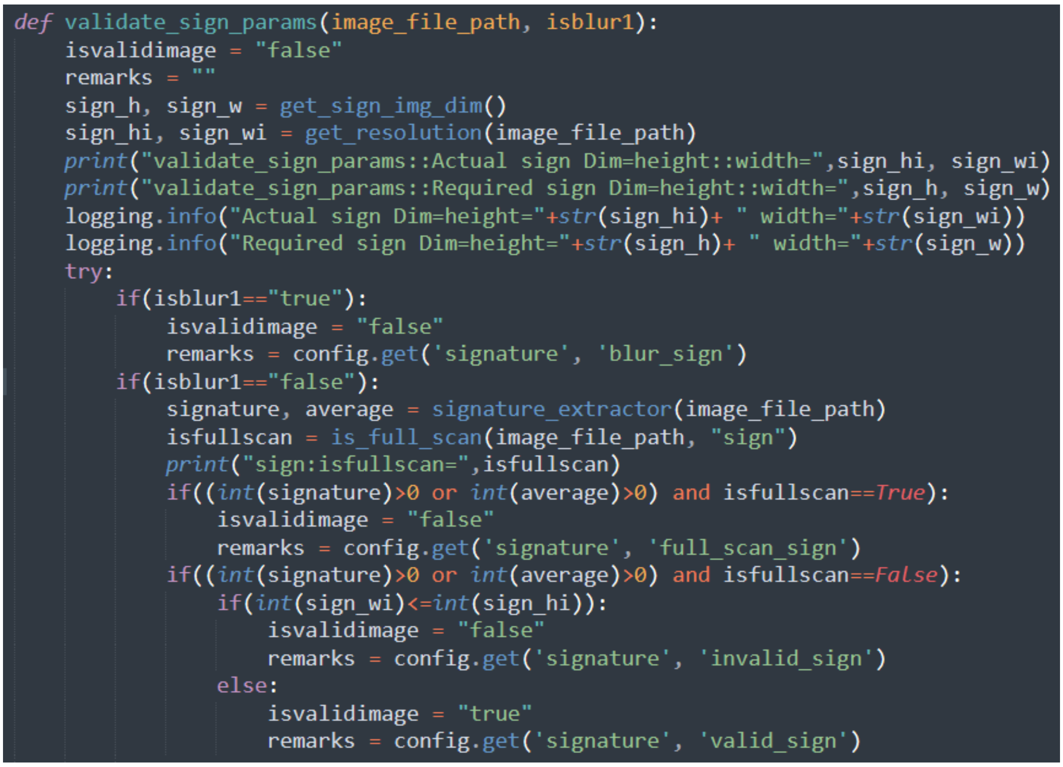
```
def validate_face_params(image_file_path, isblur1, validate_face1):
    isvalidimage = "false"
    remarks = ""
    # face_h, face_w = get_face_img_dim()
    # face_hi, face_wi = get_resolution(image_file_path)
    # print("validate_face_params::Actual face Dim=",face_hi, face_wi)
    # print("validate_face_params::Required face Dim=",face_h, face_w)
    # logging.info("Actual face Dim=height="+str(face_hi)+ " width="+str(face_wi))
    # logging.info("Required face Dim=height="+str(face_h)+ " width="+str(face_w))
    try:
        brightness = get_brightness(image_file_path)
        print("brightness=",brightness)
        brightness_threshold = int(config.get('image', 'brightness_threshold'))
        if(int(brightness)>brightness_threshold):
            isblur1 = "true"
        # Blur Image
        if(isblur1=="true"):
            isvalidimage = "false"
            remarks = config.get('face', 'blur_image')
        # Invalid Image
        if(validate_face1=="false" and isblur1=="false"):
            isvalidimage = "false"
            remarks = config.get('face', 'invalid_image')
        if(validate_face1=="true" and isblur1=="false"):
            isfullscan = is_full_scan(image_file_path, "face")
            print("isfullscan=",isfullscan)
            if(isfullscan==True):
                isvalidimage = "false"
                remarks = config.get('face', 'full_scan_image')
            if(isfullscan==False):
                isvalidimage = "true"
                remarks = config.get('face', 'valid_image')
```

Signature Detection:



```
def signature_extractor(image):
    the_biggest_component = 0
    average = 0.0
    try:
        # read the input image
        img = get_image(image, 'cv2')
        img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)[1]  # ensure binary
        # connected component analysis by scikit-learn framework
        blobs = img > img.mean()
        blobs_labels = measure.label(blobs, background=1)
        image_label_overlay = label2rgb(blobs_labels, image=img)

        total_area = 0
        counter = 0
        for region in regionprops(blobs_labels):
            if (region.area > 5):
                total_area = total_area + region.area
                counter = counter + 1
            # take regions with large enough areas
            if (region.area >= 5):
                if (region.area > the_biggest_component):
                    the_biggest_component = region.area
        average = (total_area/counter)
        print("signature:biggest_component: " + str(the_biggest_component))
        print("signature:average: " + str(round(average, 2)))
        return the_biggest_component, round(average, 2)
    except Exception as e:
        print("Exception:signature_extractor=",e)
        return the_biggest_component, round(average, 2)
```

Validation of signature params.



```
def validate_sign_params(image_file_path, isblur1):
    isvalidimage = "false"
    remarks = ""
    sign_h, sign_w = get_sign_img_dim()
    sign_hi, sign_wi = get_resolution(image_file_path)
    print("validate_sign_params::Actual sign Dim=height::width=",sign_hi, sign_wi)
    print("validate_sign_params::Required sign Dim=height::width=",sign_h, sign_w)
    logging.info("Actual sign Dim=height="+str(sign_hi)+ " width="+str(sign_wi))
    logging.info("Required sign Dim=height="+str(sign_h)+ " width="+str(sign_w))
    try:
        if(isblur1=="true"):
            isvalidimage = "false"
            remarks = config.get('signature', 'blur_sign')
        if(isblur1=="false"):
            signature, average = signature_extractor(image_file_path)
            isfullscan = is_full_scan(image_file_path, "sign")
            print("sign:isfullscan=",isfullscan)
            if((int(signature)>0 or int(average)>0) and isfullscan==True):
                isvalidimage = "false"
                remarks = config.get('signature', 'full_scan_sign')
            if((int(signature)>0 or int(average)>0) and isfullscan==False):
                if(int(sign_wi)<=int(sign_hi)):
                    isvalidimage = "false"
                    remarks = config.get('signature', 'invalid_sign')
                else:
                    isvalidimage = "true"
                    remarks = config.get('signature', 'valid_sign')
```