Playing Cards Recognition System

Team

A.V.P. Sewwandi	16001354
K.V.M.S.V. Dissanayake	16000412
W.P.G. Jayasinghe	16000617
H.A.T.Madushika	16020545
A.A.Sathsarani	16001281

INTRODUCTION

Our goal was to build a image recognition system to detect suit and rank of the standard deck of playing cards.

In current society there will be so many major image process techniques such as,

- Character segmentation->thresholding
- Gaussian blur
- Projective Transformation.
- Edge(contour) detection.
- Template matching & etc.

OUR SOLUTION

We used extensive image processing tools.

- OpenCV Library (version 3. 4 .1)
- Python (version 3.6)

Our solution will be involves major functions of the image processing.

- Image Thresholding.
- Contour(edge) Detection.
- Projective transformation.
- Template matching.

Our program will be more accurate with low noisy images although considered the image rotation & scaling.

OUR SOLUTION cont.

Our algorithm for image processing attempt , basically can be divided into two parts.

- Detection
- Identification

Detection involves..

- Filter out & Detecting Playing cards from the background.

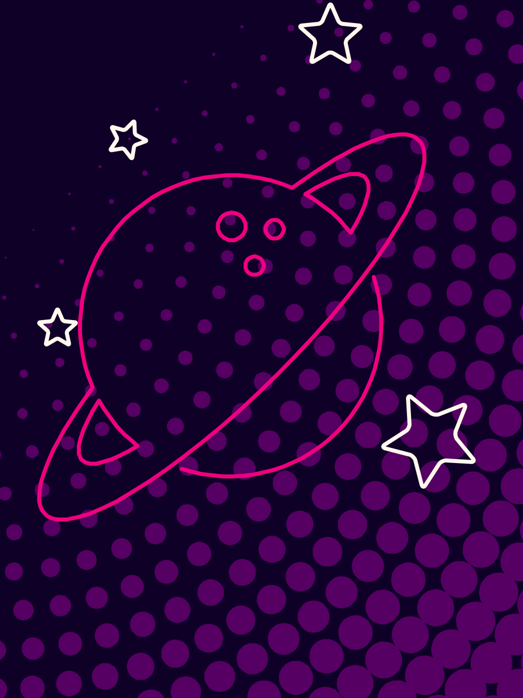
Identification involves..

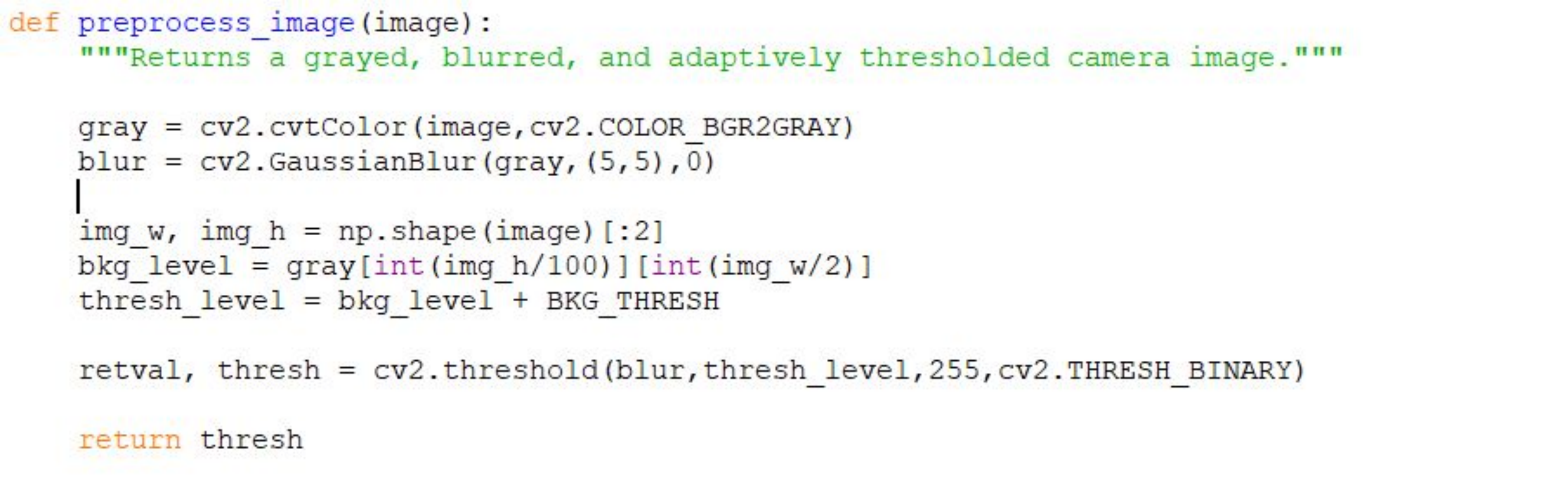
- Identify the rank and the suit of the particular detected card.

Part A - Detection

Basically to detect the playing cards on the video feed ,

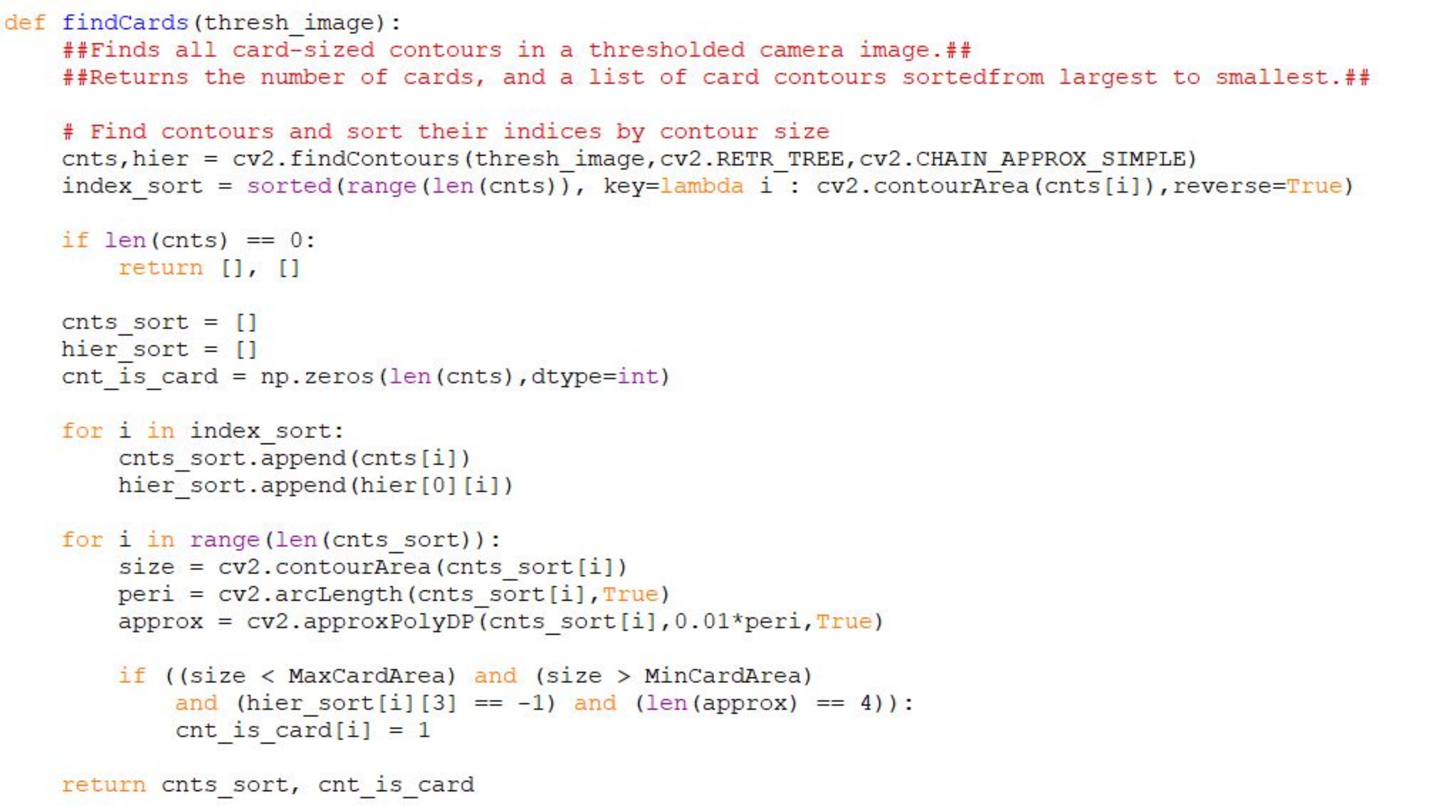
- Converted the video feed to grayscale.
- Blurred the feed that have the playing cards.[Blurred the specific area that have cards on will be enough]
- Thresholded.





```python
def preprocess_image(image):
    """Returns a grayed, blurred, and adaptively thresholded camera image."""

    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),0)

    img_w, img_h = np.shape(image)[:2]
    bkg_level = gray[int(img_h/100)][int(img_w/2)]
    thresh_level = bkg_level + BKG_THRESH

    retval, thresh = cv2.threshold(blur,thresh_level,255,cv2.THRESH_BINARY)

    return thresh
```

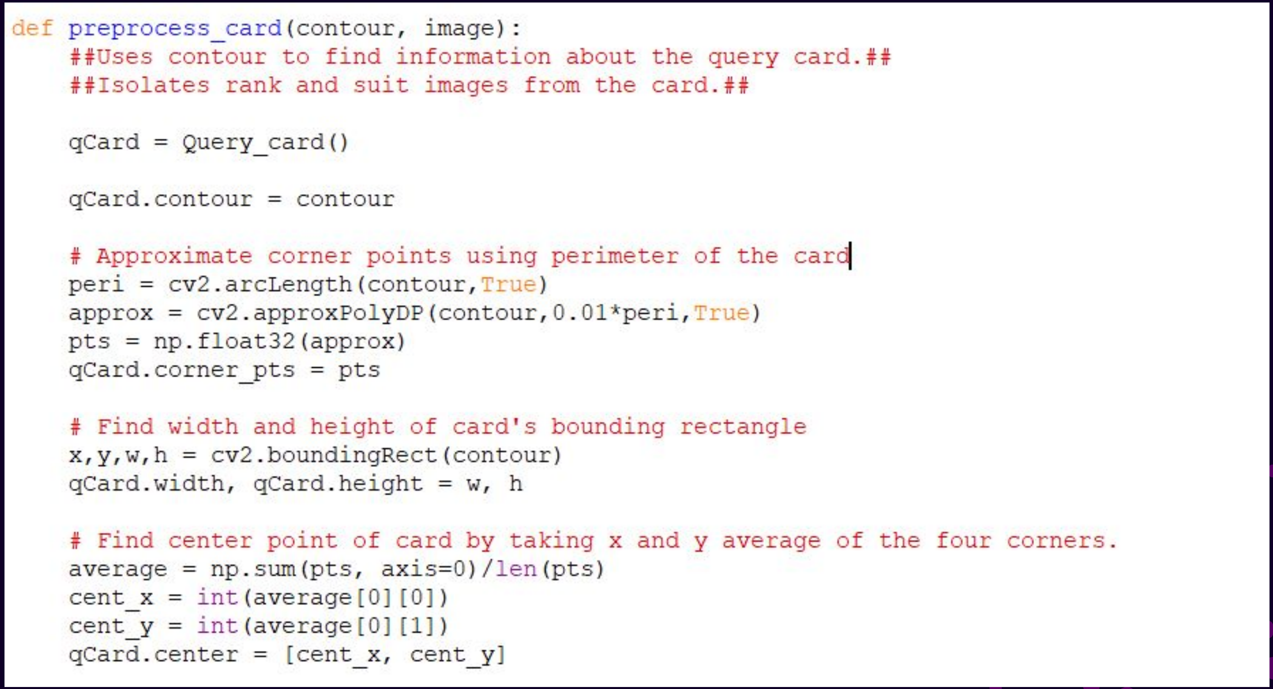
- Then program finds all card-sized contours and returns the number of cards, and a list of card contours sorted from largest to smallest.



```python
def findCards(thresh_image):
    ##Finds all card-sized contours in a thresholded camera image.##
    ##Returns the number of cards, and a list of card contours sortedfrom largest to smallest.##

    # Find contours and sort their indices by contour size
    cnts,hier = cv2.findContours(thresh_image,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    index_sort = sorted(range(len(cnts)), key=lambda i : cv2.contourArea(cnts[i]),reverse=True)

    if len(cnts) == 0:
        return [], []

    cnts_sort = []
    hier_sort = []
    cnt_is_card = np.zeros(len(cnts),dtype=int)

    for i in index_sort:
        cnts_sort.append(cnts[i])
        hier_sort.append(hier[0][i])

    for i in range(len(cnts_sort)):
        size = cv2.contourArea(cnts_sort[i])
        peri = cv2.arcLength(cnts_sort[i],True)
        approx = cv2.approxPolyDP(cnts_sort[i],0.01*peri,True)

        if ((size < MaxCardArea) and (size > MinCardArea)
            and (hier_sort[i][3] == -1) and (len(approx) == 4)):
            cnt_is_card[i] = 1

    return cnts_sort, cnt_is_card
```

Part B - Identification

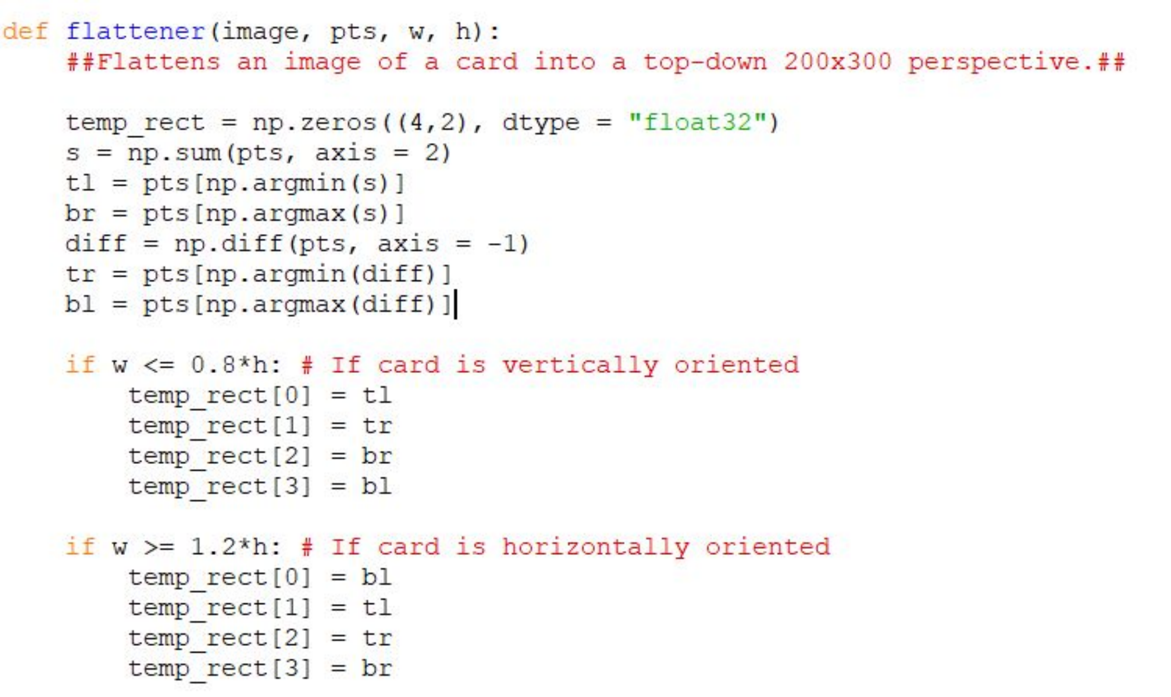
- Program approximates corner points and determines other properties of cards then find the centre point of the input card image.



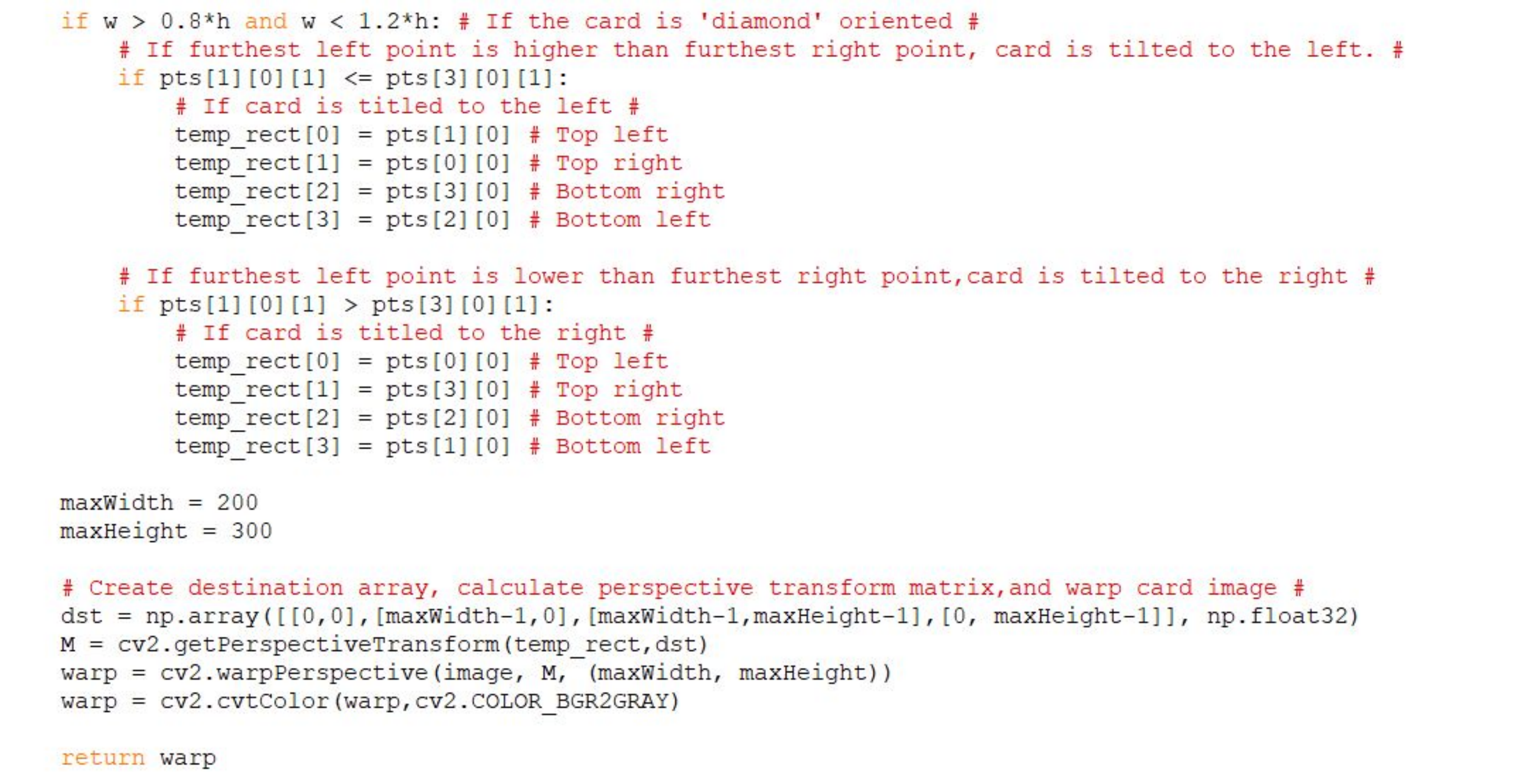
```python
def preprocess_card(contour, image):
    ##Uses contour to find information about the query card.##
    ##Isolates rank and suit images from the card.##

    qCard = Query_card()

    qCard.contour = contour

    # Approximate corner points using perimeter of the card
    peri = cv2.arcLength(contour,True)
    approx = cv2.approxPolyDP(contour,0.01*peri,True)
    pts = np.float32(approx)
    qCard.corner_pts = pts

    # Find width and height of card's bounding rectangle
    x,y,w,h = cv2.boundingRect(contour)
    qCard.width, qCard.height = w, h

    # Find center point of card by taking x and y average of the four corners.
    average = np.sum(pts, axis=0)/len(pts)
    cent_x = int(average[0][0])
    cent_y = int(average[0][1])
    qCard.center = [cent_x, cent_y]
```

Part B - Identification cont.

- Then it will warp the card into 200 by 300 flattened perspective by using image scaling and rotation techniques.



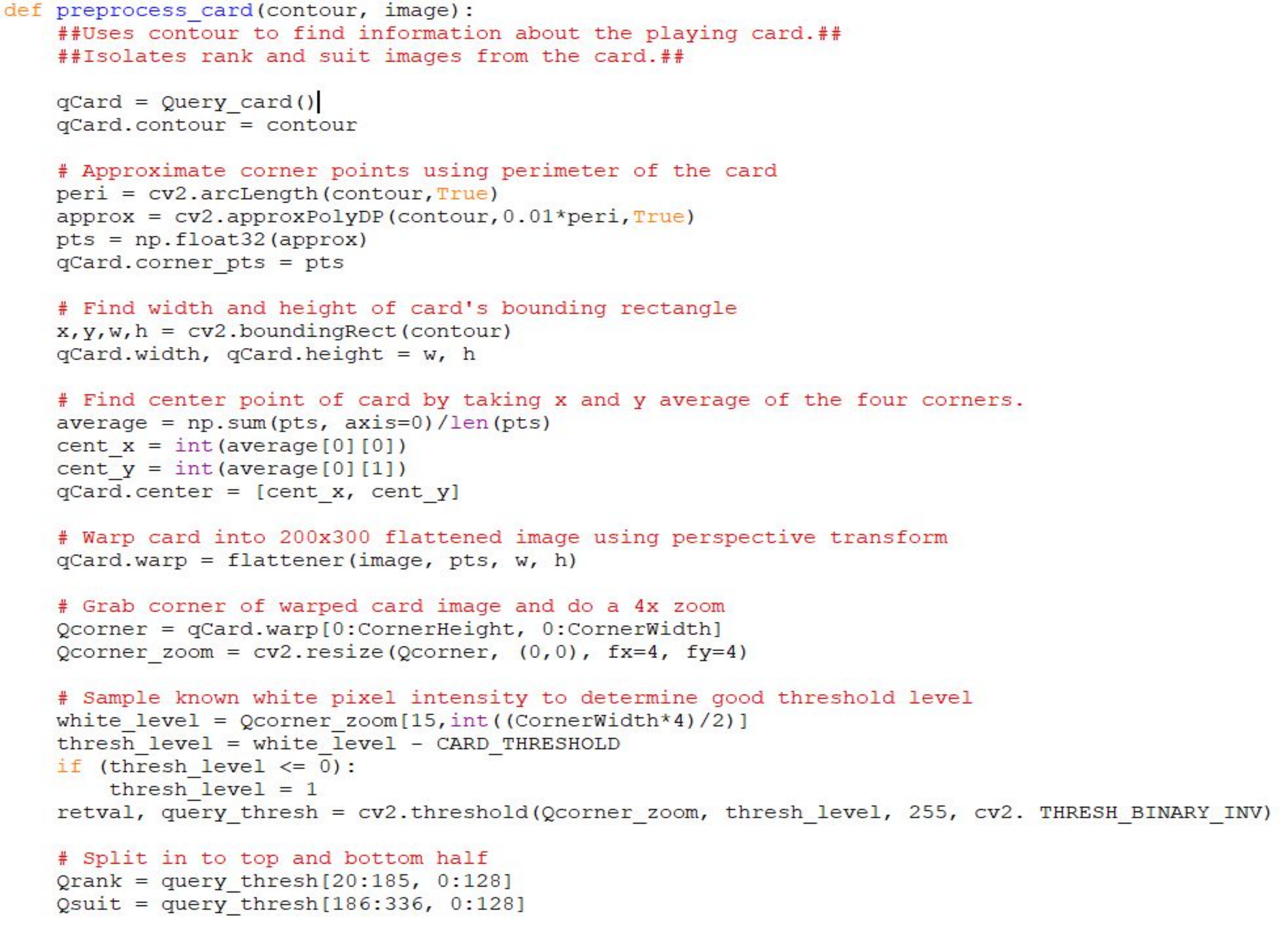
```
def flattener(image, pts, w, h):
    ##Flattens an image of a card into a top-down 200x300 perspective.##

    temp_rect = np.zeros((4,2), dtype = "float32")
    s = np.sum(pts, axis = 2)
    tl = pts[np.argmin(s)]
    br = pts[np.argmax(s)]
    diff = np.diff(pts, axis = -1)
    tr = pts[np.argmin(diff)]
    bl = pts[np.argmax(diff)]

    if w <= 0.8*h: # If card is vertically oriented
        temp_rect[0] = tl
        temp_rect[1] = tr
        temp_rect[2] = br
        temp_rect[3] = bl

    if w >= 1.2*h: # If card is horizontally oriented
        temp_rect[0] = bl
        temp_rect[1] = tl
        temp_rect[2] = tr
        temp_rect[3] = br
```



```
if w > 0.8*h and w < 1.2*h: # If the card is 'diamond' oriented #
    # If furthest left point is higher than furthest right point, card is tilted to the left. #
    if pts[1][0][1] <= pts[3][0][1]:
        # If card is titled to the left #
        temp_rect[0] = pts[1][0] # Top left
        temp_rect[1] = pts[0][0] # Top right
        temp_rect[2] = pts[3][0] # Bottom right
        temp_rect[3] = pts[2][0] # Bottom left

    # If furthest left point is lower than furthest right point,card is tilted to the right #
    if pts[1][0][1] > pts[3][0][1]:
        # If card is titled to the right #
        temp_rect[0] = pts[0][0] # Top left
        temp_rect[1] = pts[3][0] # Top right
        temp_rect[2] = pts[2][0] # Bottom right
        temp_rect[3] = pts[1][0] # Bottom left

maxWidth = 200
maxHeight = 300

# Create destination array, calculate perspective transform matrix,and warp card image #
dst = np.array([[0,0],[maxWidth-1,0],[maxWidth-1,maxHeight-1],[0, maxHeight-1]], np.float32)
M = cv2.getPerspectiveTransform(temp_rect,dst)
warp = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
warp = cv2.cvtColor(warp,cv2.COLOR_BGR2GRAY)

return warp
```

Part B Identification cont.

- After that program grab corner of warped card image.
- Image is zoomed by factor of 4x and calculate a good threshold level using sample white pixel intensity
- Split the thresholded image into top and bottom half.
- Top shows rank and bottom shows suit of the card.

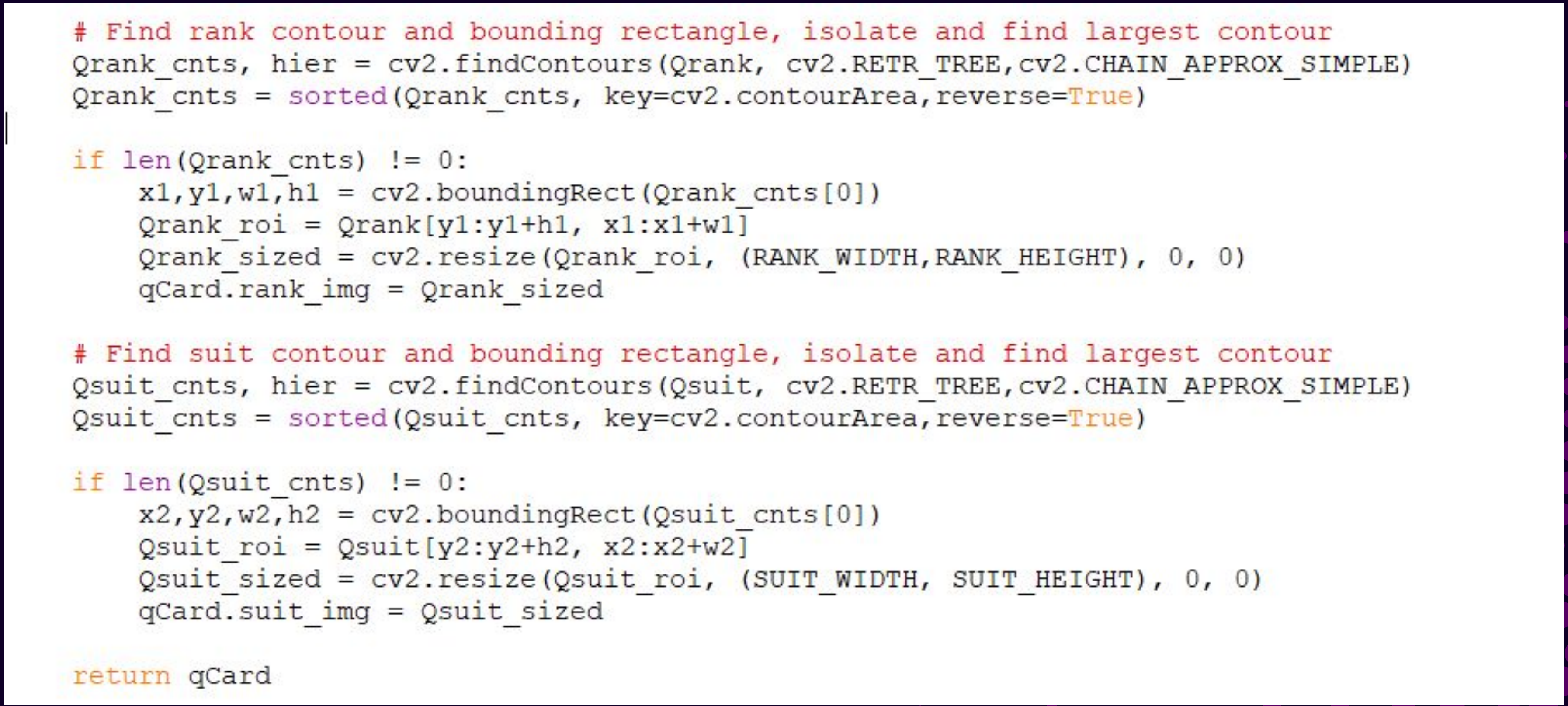


```
def preprocess_card(contour, image):
    ##Uses contour to find information about the playing card.##
    ##Isolates rank and suit images from the card.##

    qCard = Query_card()
    qCard.contour = contour

    # Approximate corner points using perimeter of the card
    peri = cv2.arcLength(contour,True)
    approx = cv2.approxPolyDP(contour,0.01*peri,True)
    pts = np.float32(approx)
    qCard.corner_pts = pts

    # Find width and height of card's bounding rectangle
    x,y,w,h = cv2.boundingRect(contour)
    qCard.width, qCard.height = w, h

    # Find center point of card by taking x and y average of the four corners.
    average = np.sum(pts, axis=0)/len(pts)
    cent_x = int(average[0][0])
    cent_y = int(average[0][1])
    qCard.center = [cent_x, cent_y]

    # Warp card into 200x300 flattened image using perspective transform
    qCard.warp = flattener(image, pts, w, h)

    # Grab corner of warped card image and do a 4x zoom
    Qcorner = qCard.warp[0:CornerHeight, 0:CornerWidth]
    Qcorner_zoom = cv2.resize(Qcorner, (0,0), fx=4, fy=4)

    # Sample known white pixel intensity to determine good threshold level
    white_level = Qcorner_zoom[15,int((CornerWidth*4)/2)]
    thresh_level = white_level - CARD_THRESHOLD
    if (thresh_level <= 0):
        thresh_level = 1
    retval, query_thresh = cv2.threshold(Qcorner_zoom, thresh_level, 255, cv2. THRESH_BINARY_INV)

    # Split in to top and bottom half
    Qrank = query_thresh[20:185, 0:128]
    Qsuit = query_thresh[186:336, 0:128]
```

Part B Identification cont.

- Find contours of splitted suit and rank images, isolate and find largest contour.

- Then find bounding rectangle for largest contour and use it to resize rank and suit images to match dimensions of the trained rank and suit images.



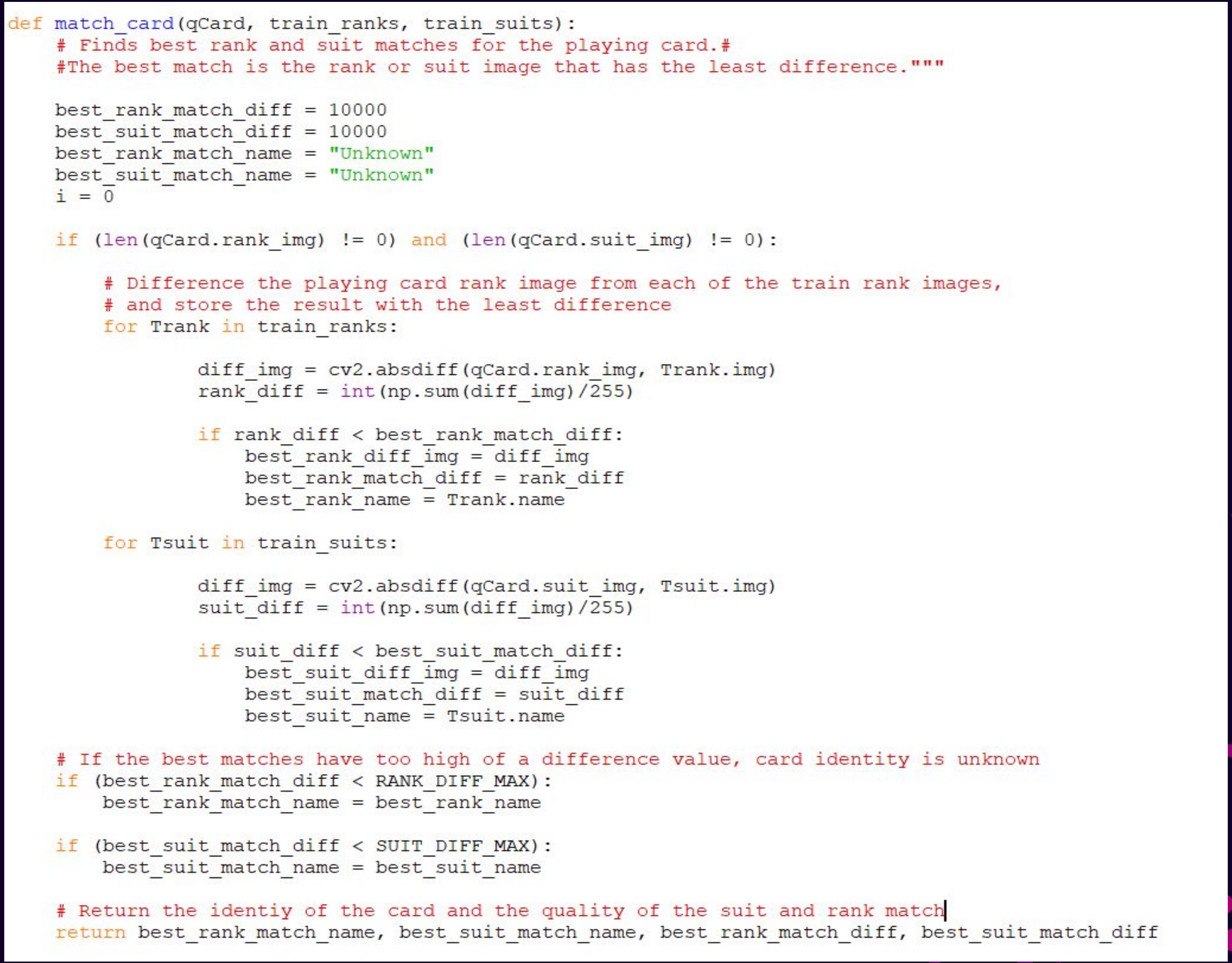
```
# Find rank contour and bounding rectangle, isolate and find largest contour
Qrank_cnts, hier = cv2.findContours(Qrank, cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
Qrank_cnts = sorted(Qrank_cnts, key=cv2.contourArea,reverse=True)

if len(Qrank_cnts) != 0:
    x1,y1,w1,h1 = cv2.boundingRect(Qrank_cnts[0])
    Qrank_roi = Qrank[y1:y1+h1, x1:x1+w1]
    Qrank_sized = cv2.resize(Qrank_roi, (RANK_WIDTH,RANK_HEIGHT), 0, 0)
    qCard.rank_img = Qrank_sized

# Find suit contour and bounding rectangle, isolate and find largest contour
Qsuit_cnts, hier = cv2.findContours(Qsuit, cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
Qsuit_cnts = sorted(Qsuit_cnts, key=cv2.contourArea,reverse=True)

if len(Qsuit_cnts) != 0:
    x2,y2,w2,h2 = cv2.boundingRect(Qsuit_cnts[0])
    Qsuit_roi = Qsuit[y2:y2+h2, x2:x2+w2]
    Qsuit_sized = cv2.resize(Qsuit_roi, (SUIT_WIDTH, SUIT_HEIGHT), 0, 0)
    qCard.suit_img = Qsuit_sized

return qCard
```

Part B Identification cont.

- Then program compares the isolated images of ranks and suits with the trained(predefined) images of ranks and suits.

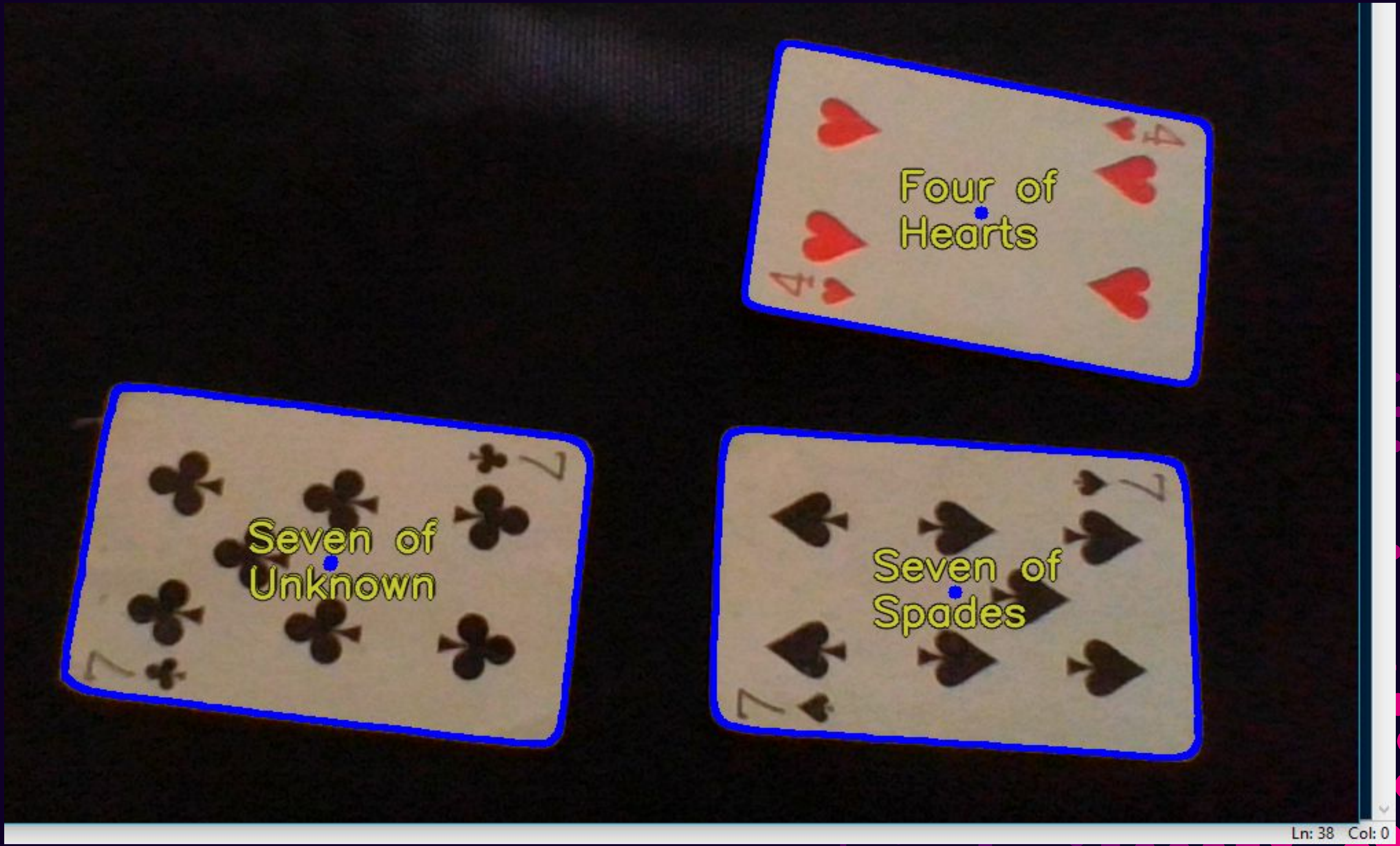
- Get the differenced image between trained images and the isolated image then get the quality of the number by counting the white pixels occur in the difference image.

- After that, minimum number of white pixels occur in the difference image got it as best matched and produced as the suit and rank.



```python
def match_card(qCard, train_ranks, train_suits):
    # Finds best rank and suit matches for the playing card.#
    #The best match is the rank or suit image that has the least difference."""

    best_rank_match_diff = 10000
    best_suit_match_diff = 10000
    best_rank_match_name = "Unknown"
    best_suit_match_name = "Unknown"
    i = 0

    if (len(qCard.rank_img) != 0) and (len(qCard.suit_img) != 0):

        # Difference the playing card rank image from each of the train rank images,
        # and store the result with the least difference
        for Trank in train_ranks:

                diff_img = cv2.absdiff(qCard.rank_img, Trank.img)
                rank_diff = int(np.sum(diff_img)/255)

                if rank_diff < best_rank_match_diff:
                    best_rank_diff_img = diff_img
                    best_rank_match_diff = rank_diff
                    best_rank_name = Trank.name

        for Tsuit in train_suits:

                diff_img = cv2.absdiff(qCard.suit_img, Tsuit.img)
                suit_diff = int(np.sum(diff_img)/255)

                if suit_diff < best_suit_match_diff:
                    best_suit_diff_img = diff_img
                    best_suit_match_diff = suit_diff
                    best_suit_name = Tsuit.name

    # If the best matches have too high of a difference value, card identity is unknown
    if (best_rank_match_diff < RANK_DIFF_MAX):
        best_rank_match_name = best_rank_name

    if (best_suit_match_diff < SUIT_DIFF_MAX):
        best_suit_match_name = best_suit_name

    # Return the identiy of the card and the quality of the suit and rank match
    return best_rank_match_name, best_suit_match_name, best_rank_match_diff, best_suit_match_diff
```

OUTCOMES



THANKS!

Q&A