```
#
# Emily Schultz
# ess2183
# COMS 4735, Assignment 1: Hand Gesture Combination Lock
# Due: 2/19/2013
#

import Image
import ImageFilter
import ImageEnhance as ie
import ImageMath as im
import ImageOps as io
import scipy
from scipy import ndimage
import numpy as np

def main():
    '''
    This method mostly interacts with the user, getting the input, passing that
    to the program, then getting the combination YES/NO displaying it to user.
    '''
    print 'Welcome to Hand Gesture Combination Lock'
    sequence = map(int, raw_input('Please enter hand combination (photo numbers '
                                  'separated by spaces):').split(' '))
    result = check_combination(sequence)
    if result == 0:
        print 'Sorry, you have not entered the correct combination.'
    elif result == 1:
        print 'Congrats, you have entered the correct combination.'
    else:
        print 'The program is unsure about one or more images from the input sequence'
        print 'and therefore will not be opening the lock'

def check_combination(seq):
    '''
    The main algorithmic portion of the program. Takes the png file, makes it a ppm
    file. Uses the pixel data of that smaller, better ppm file and converts it to
    binary 0's and 1's. Using that image, we compute the center of mass and bounding
    box. The ratio of these two tells us whether the image is splayed vs. fist.
    Then in the creativity portion, this is extended. We look at the number of
    connected components, as well as the rectangularness of the bounding box to
    determine the gesture being formed.
    '''
    #past gestures and unlock y/n
    history = ('','')
    result = 0

    for num in seq:
        # convert png files to ppm
        filename = 'images/h' + str(num) + '.png'
        outname =  'images/hh' + str(num) + '.ppm'
        imconvert(filename, outname)

        # convert to binary 0/1 file
        im = Image.open(outname)

        # calculate center of mass and bounding box for skin
        com = centermass(im)
        loc = im.getbbox() #left, upper, right, lower
        bbox_area = (loc[2] - loc[0])*(loc[3] - loc[1])

        # calculate ratio of bounding box area to actual skin area
        bc_ratio = bbox_area/float(com[0])

        #calclate bounding box width to length ratio
        bbox_width = (loc[2] - loc[0])
        bbox_height = (loc[3] - loc[1])
        bb_wh_ratio = bbox_width / float(bbox_height)

        #calculate the number of connected components
        cc = conn_comp(outname)

        #FROM ORIGINAL, not CREATIVITY:
        #if bc_ratio < 1.45:
        #    #FIST
        #        # UPPER LEFT                    #UPPER RIGHT
        #    if (com[0] < 40 and com[1] < 33) or (com[0] < 40 and com[1] > 45)
        #        or (com[0] > 60 and com[1] > 45) or (com[0] > 60 and com[1] < 33):
        #        # LOWER RIGHT                       #LOWER LEFT
        #        if(hist == "SPLAY_CEN"):
        #            return 1
        #        else
        #            hist == "FIST_COR"
        #elif bc_ratio > 2.0:
        #    #SPLAY
        #    if com[0] > 35 and com[0] < 65 and com[1] > 25 and com[1] < 53:
        #        #CENTER
        #        hist == "SPLAY_CEN"
        #else:
        #    #UNKNOWN
        #    hist = "?"

        #determine which gesture it is, and proper response based on history
        #CORRECT:
        cur = ""
        if cc == 2 and bb_wh_ratio < 1.2 and bb_wh_ratio > .7:
            # Circle "O" Symbol
            cur = "O"
        elif cc == 2 and bb_wh_ratio > 1.5 and bb_wh_ratio < 3.0:
            #"OK" Symbol
            cur = "OK"
        elif cc == 1 and bb_wh_ratio < 1.2 and bb_wh_ratio > 0.7:
            #Fist
            # automatic cancel if fist in lower right corner
            if com[0] > 60 and com[1] > 45:
                return 0;
            cur = "FIST"
        elif cc == 1 and bb_wh_ratio > 1.5 and bb_wh_ratio < 3.0:
            #splay palm
            cur = "SPLAY"
        elif cc == 1 and bb_wh_ratio > 4.0:
            #horizontal chop
            cur = "HCHOP"
        elif cc == 1 and bb_wh_ratio < 0.5:
            #vertical chop
            cur = "VCHOP"
        else:
            #UH-OH! Unknown - pretend it's another symbol not part of passcode
            cur = "?"
        #check if combo is good
        if history[1] == "VCHOP" and history[0] == "O":
            result = 1
        else:
        #update history
            history[1] = history[0]
            history[0] = cur

    return result

def imconvert(infile, outfile):
    '''
    This method takes two filenames, one for in and one for out, then opens the infile
    resizes it to a more reasonable size, smooths for inconsistencies, then saves it
    as outfile.
    '''
    im = Image.open(infile)
    new = im.resize((100,75))
    new = new.filter(ImageFilter.SMOOTH)
    new = binarize(new)
    try:
        new.save(outfile)
    except IOError:
        print "cannot convert ", infile


def conn_comp(outname):
    '''
    Returns the number of connected components. Has to invert the file first since
    we want the number of black components (assumes white components is always 1)
    then uses scipy library to get number of disconnected components.
    '''
    #first invert the image
    im = Image.open(outname)
    inverted = io.invert(im)
    inverted.save('inverted.ppm')

    image = ndimage.imread('inverted.ppm')
    a, num_comp = ndimage.measurements.label(image)
    return num_comp

def binarize(image):
    '''
    Given an image, it loads the image into a list of RGB values then, accessing each
    pixel at a time row-wise and column-wise, determines if the average pixel value is
    above 180. If it is, then it is a hand pixel and set to 255, 255, 255. Otherwise,
    it is a background pixel and set to 0,0,0.
    '''
    pix = image.load()

    for r in range (0,image.size()[0]):
        for c in range(0, image.size()[1]):
            curpix = pix[r,c]
            avg = int((curpix[0] + curpix[1] + curpix[2])/3.0)
            if(avg > 180):
                pix[r,c] = (255,255,255)
            else:
                pix[r,c] = (0,0,0)
    return image

def centermass(image):
    '''
    Given a binary image, it loads the image as a list, the accessing each value in
    turn. Whenever it accesses a 'skin' value (ie an all 255 pixel), it adds to the
    count of skin pixels total as well as the location of the pixel in x and y.
    It returns a list of the form (area, center of mass x, center of mass y).
    '''
    pix = image.load()
    count = 0
    xsum = 0
    ysum = 0
    for r in range(0,image.size()[0]):
        for c in range(0, image.size()[1]):
            curpix = pix[r,c]
            if curpix[0] == 255:
                xsum = xsum + r
                ysum = ysum + c
                count = count + 1
    xavg = xsum / (count * 1.0)
    yavg = ysum / (count * 1.0)
    return (area, xavg, yavg)

#call main() method
if __name__ == "__main__":
    main()
```