The Project

1. This is a project with minimal scaffolding. Expect to use the the discussion forums to gain insights! It's not cheating to ask others for opinions or perspectives!
2. Be inquisitive, try out new things.
3. Use the previous modules for insights into how to complete the functions! You'll have to combine Pillow, OpenCV, and Pytesseract
4. There are hints provided in Coursera, feel free to explore the hints if needed. Each hint provide progressively more details on how to solve the issue. This project is intended to be comprehensive and difficult if you do it without the hints.

The Assignment

Take a ZIP file) of images and process them, using a library built into python that you need to learn how to use. A ZIP file takes several different files and compresses them, thus saving space, into one single file. The files in the ZIP file we provide are newspaper images (like you saw in week 3). Your task is to write python code which allows one to search through the images looking for the occurrences of keywords and faces. E.g. if you search for "pizza" it will return a contact sheet of all of the faces which were located on the newspaper page which mentions "pizza". This will test your ability to learn a new (library), your ability to use OpenCV to detect faces, your ability to use tesseract to do optical character recognition, and your ability to use PIL to composite images together into contact sheets.

Each page of the newspapers is saved as a single PNG image in a file called images.zip. These newspapers are in english, and contain a variety of stories, advertisements and images. Note: This file is fairly large (~200 MB) and may take some time to work with, I would encourage you to use small_img.zip for testing.

Here's an example of the output expected. Using the small_img.zip file, if I search for the string "Christopher" I should see the following image:

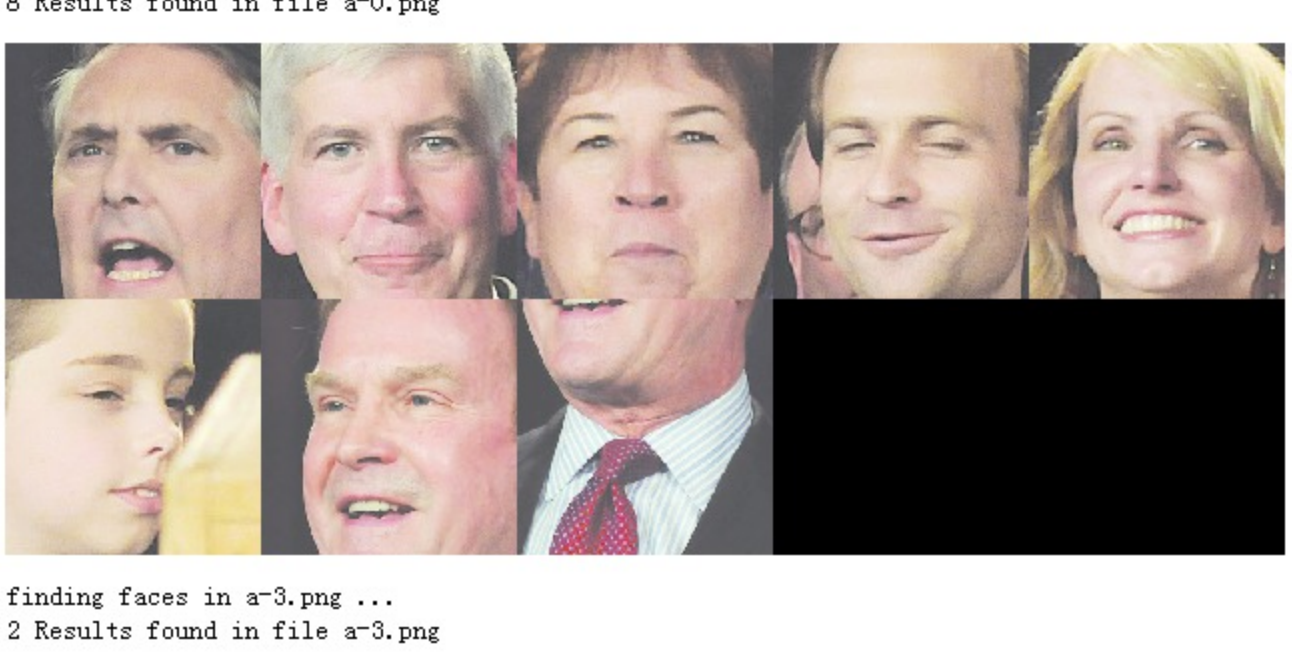
Christopher Search
If I were to use the images.zip file and search for "Mark" I should see the following image (note that there are times when there are no faces on a page, but a word is found!):

Mark Search

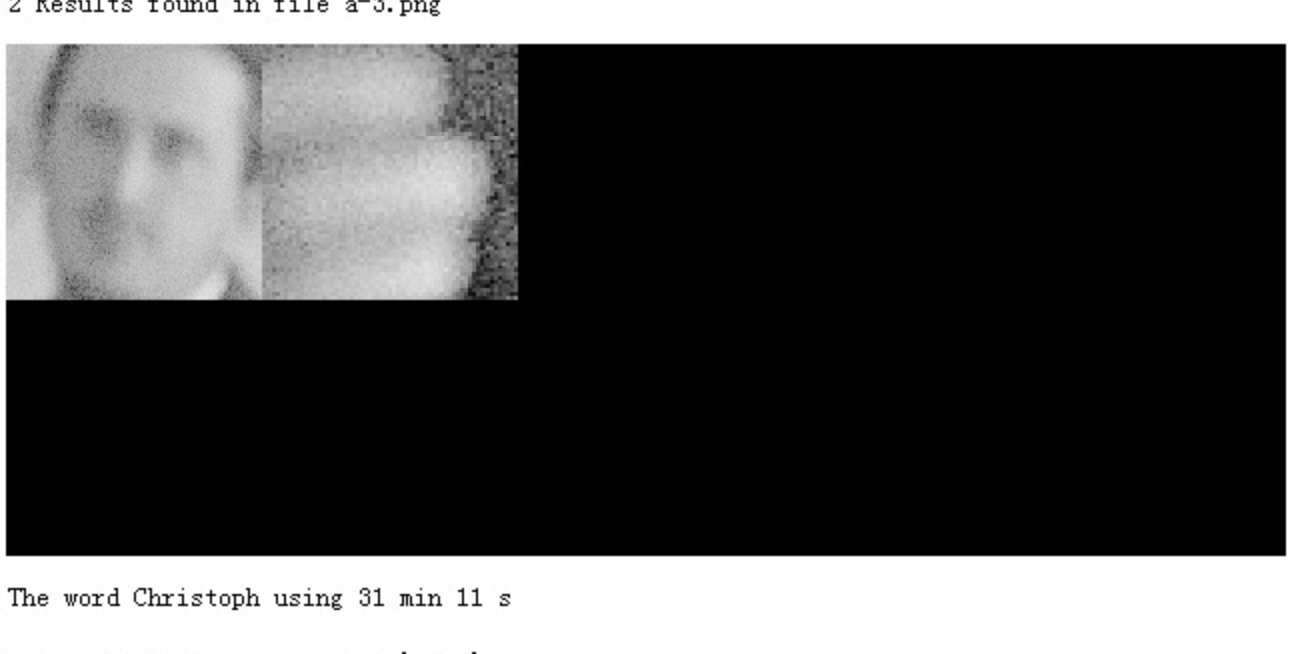
Note: That big file can take some time to process - for me it took nearly ten minutes! Use the small one for testing.

[1]:
```python
print('Checking that Jupyter environment is online...')
import zipfile
from zipfile import ZipFile
import time
import PIL
from PIL import Image
import pytesseract
import cv2 as cv
import numpy as np
from PIL import ImageDraw

T0 = time.time()
# loading the word and filedirs
target_word = ['Christoph','Mark','Michigan','serving','celebrating']
tw = ''
for i in range(len(target_word)-1):
    tw = tw + '\'' + target_word[i] + '\', '
print('We choose the words of '+ tw +'and \''+target_word[-1]+'\' as our target words\n')
test_dir = 'readonly/small_img.zip'
file_dir = 'readonly/images.zip'
# file_dir = 'readonly/small_img.zip'


for tarword in target_word:
    T1 = time.time()

    print('The target word we chose is \'' + tarword + '\'')


    # loading the face detection classifier
    face_cascade = cv.CascadeClassifier('readonly/haarcascade_frontalface_default.xml')

    # define a function to take in an image and a threshold value
    def binarize(image_to_transform, threshold):
        # now, lets convert that image to a single greyscale image using convert()
        output_image=image_to_transform.convert("L")
        # the threshold value is usually provided as a number between 0 and 255, which
        # is the number of bits in a byte.
        # the algorithm for the binarization is pretty simple, go through every pixel in the
        # image and, if it's greater than the threshold, turn it all the way up (255), and
        # if it's lower than the threshold, turn it all the way down (0).
        # so lets write this in code. First, we need to iterate over all of the pixels in the
        # image we want to work with
        for x in range(output_image.width):
            for y in range(output_image.height):
                # for the given pixel at w,h, lets check its value against the threshold
                if output_image.getpixel((x,y))< threshold: #note that the first parameter is actually a tuple object
                    # lets set this to zero
                    output_image.putpixel( (x,y), 0 )
                else:
                    # otherwise lets set this to 255
                    output_image.putpixel( (x,y), 255 )
        #now we just return the new image
        return output_image

    # This is for test
    #word_dict = {#'a-1.png':'Christoph',#'a-0.png':'Christoph','a-2.png':'Christoph','a-3.png':'Christoph',
    #             'a-4.png':'Christoph','a-5.png':'Christoph','a-6.png':'Christoph','a-7.png':'Christoph',
    #             'a-8.png':'Christoph','a-9.png':'Christoph','a-10.png':'Christoph','a-11.png':'Christoph',
    #             'a-12.png':'Christoph','a-13.png':'Christoph'}

    # if the type is test, then we use the saved data to run the program (this is just for programing process)
    if 'word_dict' not in dir():
        word_dict = {}
        thresh = 230
        with ZipFile(file_dir) as myzip:
            for each in myzip.infolist():
                print('recognizing the words in',each.filename)
                with myzip.open(each.filename) as myfile:
                    im=Image.open(myfile)
                    img = im.convert('L')
                    text = pytesseract.image_to_string(binarize(img, thresh))
                    word_dict[each.filename] = text
                print(each.filename,'over')

    # Defining the face bounding box (only for programming process)

#     if 'face_dir' not in dir():
#         face_dict = {}
#         with ZipFile(file_dir) as myzip:
#             for each in word_dict:
#                 with myzip.open(each) as myfile:
#                     print('finding faces in',each)
#                     pil_img = Image.open(myfile)
#                     face_cascade = cv.CascadeClassifier('readonly/haarcascade_frontalface_default.xml')
#                     img = np.array(pil_img)
#                     gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
#                     faces = face_cascade.detectMultiScale(gray,minNeighbors = 15)
#                     face_dict[each] = faces.tolist()
#                 print(each,'over')
#         print(face_dict)

    with ZipFile(file_dir) as myzip:
        for each in word_dict:
            if tarword.lower() in word_dict[each].lower():
                with myzip.open(each) as myfile:
                    print('finding faces in',each,'...')
                    pil_img = Image.open(myfile)
                    face_cascade = cv.CascadeClassifier('readonly/haarcascade_frontalface_default.xml')
                    img = np.array(pil_img)
                    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
                    faces = face_cascade.detectMultiScale(gray,minNeighbors = 9)
                    # print(faces, type(faces))

                    #facelist = faces.tolist()

                    # create a contact sheet from different brightnesses
                    # facelist = face_dict[each]
                    size = 256
                    if len(faces) == 0:
                        print('But there is no faces in that file!')
                    else:
                        facelist = faces.tolist()
                        rec = facelist[0]
                        im = pil_img.crop((rec[0],rec[1],rec[0]+rec[2],rec[1]+rec[3]))
                        first_image = im.resize((size,size))
                        contact_sheet = Image.new(first_image.mode, (size*5,size*2))
                        x=0
                        y=0
                        print(len(facelist),'Results found in file',each)
                        for rec in facelist:
                            # resize the images and display
                            im = pil_img.crop((rec[0],rec[1],rec[0]+rec[2],rec[1]+rec[3]))
                            imrsz = im.resize((size,size))
                            contact_sheet.paste(imrsz, (x, y) )
                            # Now we update our X position. If it is going to be the width of the image, then we set it to 0
                            # and update Y as well to point to the next "line" of the contact sheet.
                            if x+ size >= contact_sheet.width:
                                x=0
                                y=y + size
                            else:
                                x=x + size

                        # resize and display the contact sheet

                        contact_sheet = contact_sheet.resize((int(contact_sheet.width/2),int(contact_sheet.height/2) ))

                        display(contact_sheet)

    T2 = time.time()
    print('The word', tarword, 'using',int((T2-T1)/60), 'min' , int((T2-T1) % 60), 's\n')

T00 = time.time()
print('All finished, ', 'using',int((T00-T0)/60), 'min' , int((T2-T1) % 60), 's')
```

```
Checking that Jupyter environment is online...
We choose the words of 'Christoph', 'Mark', 'Michigan', 'serving', and 'celebrating' as our target words

The target word we chose is 'Christoph'
recognizing the words in a-0.png
a-0.png over
recognizing the words in a-1.png
a-1.png over
recognizing the words in a-10.png
a-10.png over
recognizing the words in a-11.png
a-11.png over
recognizing the words in a-12.png
a-12.png over
recognizing the words in a-13.png
a-13.png over
recognizing the words in a-2.png
a-2.png over
recognizing the words in a-3.png
a-3.png over
recognizing the words in a-4.png
a-4.png over
recognizing the words in a-5.png
a-5.png over
recognizing the words in a-6.png
a-6.png over
recognizing the words in a-7.png
a-7.png over
recognizing the words in a-8.png
a-8.png over
recognizing the words in a-9.png
a-9.png over
finding faces in a-0.png ...
8 Results found in file a-0.png
```



```
finding faces in a-3.png ...
2 Results found in file a-3.png
```



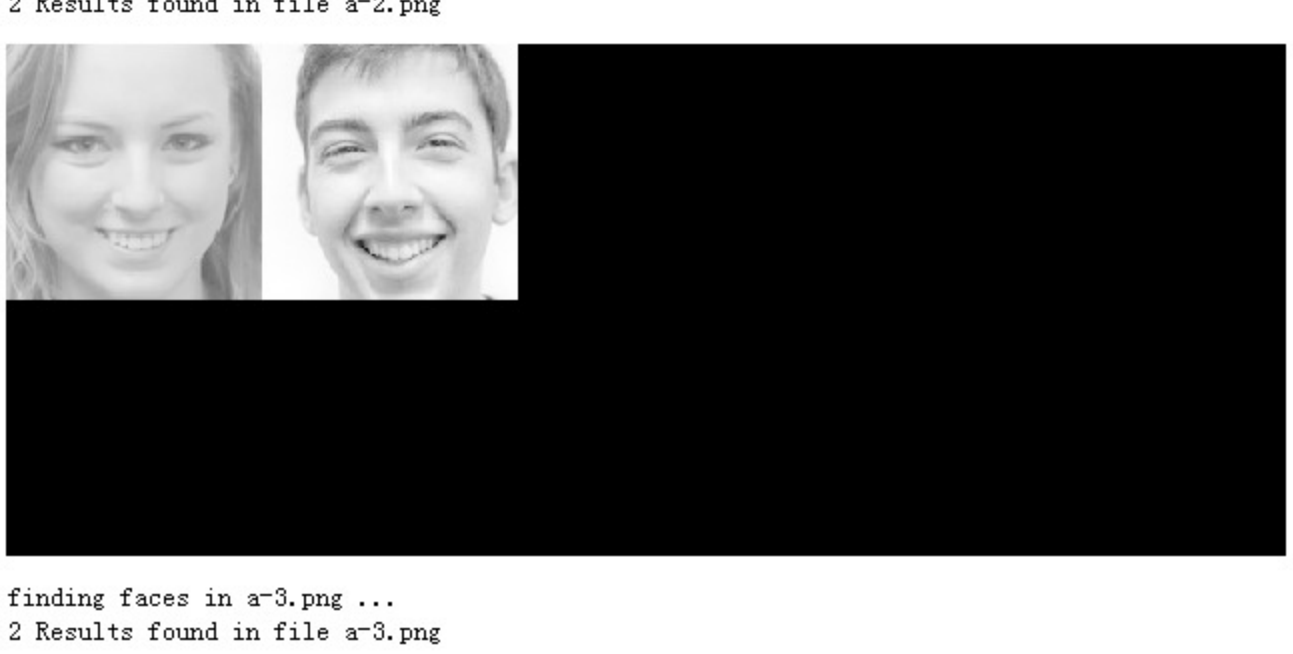
```
The word Christoph using 31 min 11 s

The target word we chose is 'Mark'
finding faces in a-0.png ...
8 Results found in file a-0.png
```

```
finding faces in a-1.png ...
5 Results found in file a-1.png
```

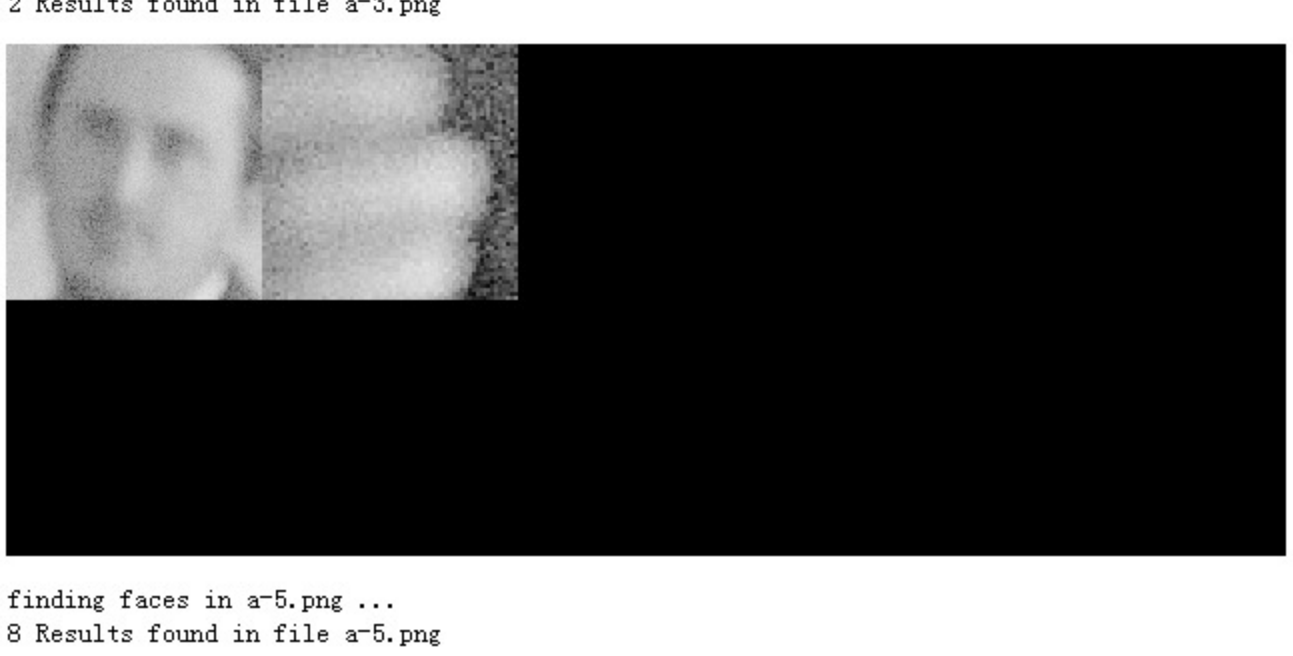
```
finding faces in a-10.png ...
1 Results found in file a-10.png
```

```
finding faces in a-13.png ...
5 Results found in file a-13.png
```

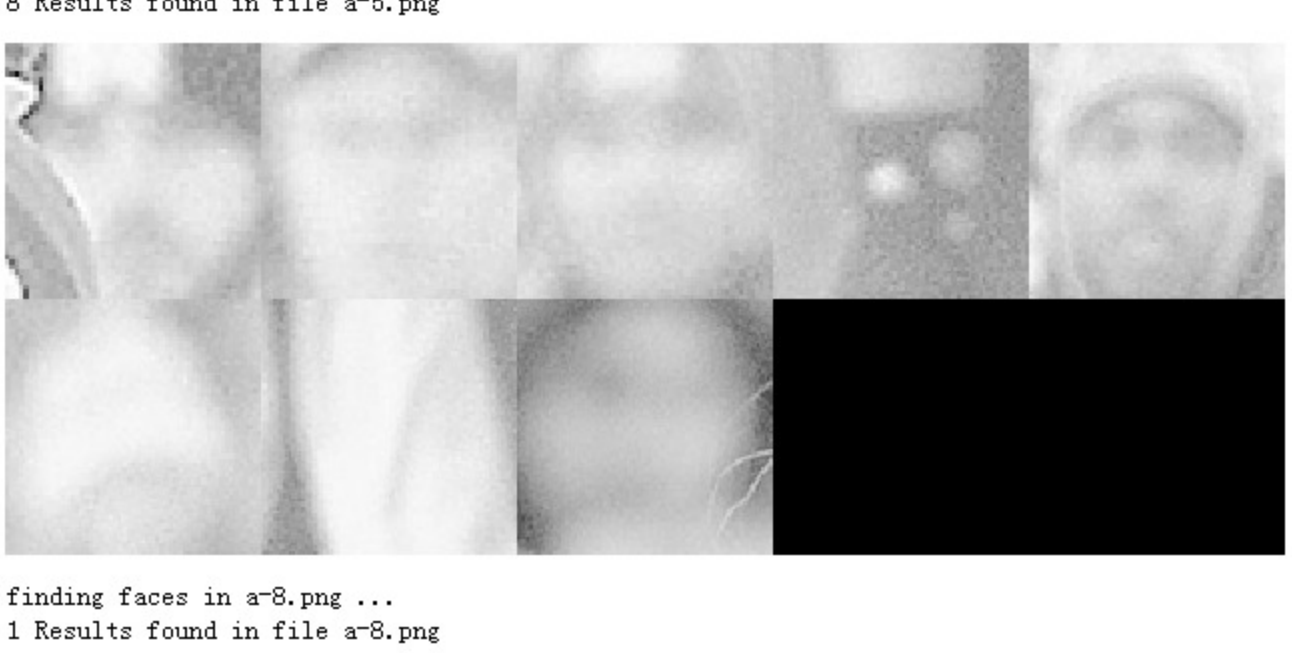
```
finding faces in a-2.png ...
2 Results found in file a-2.png
```



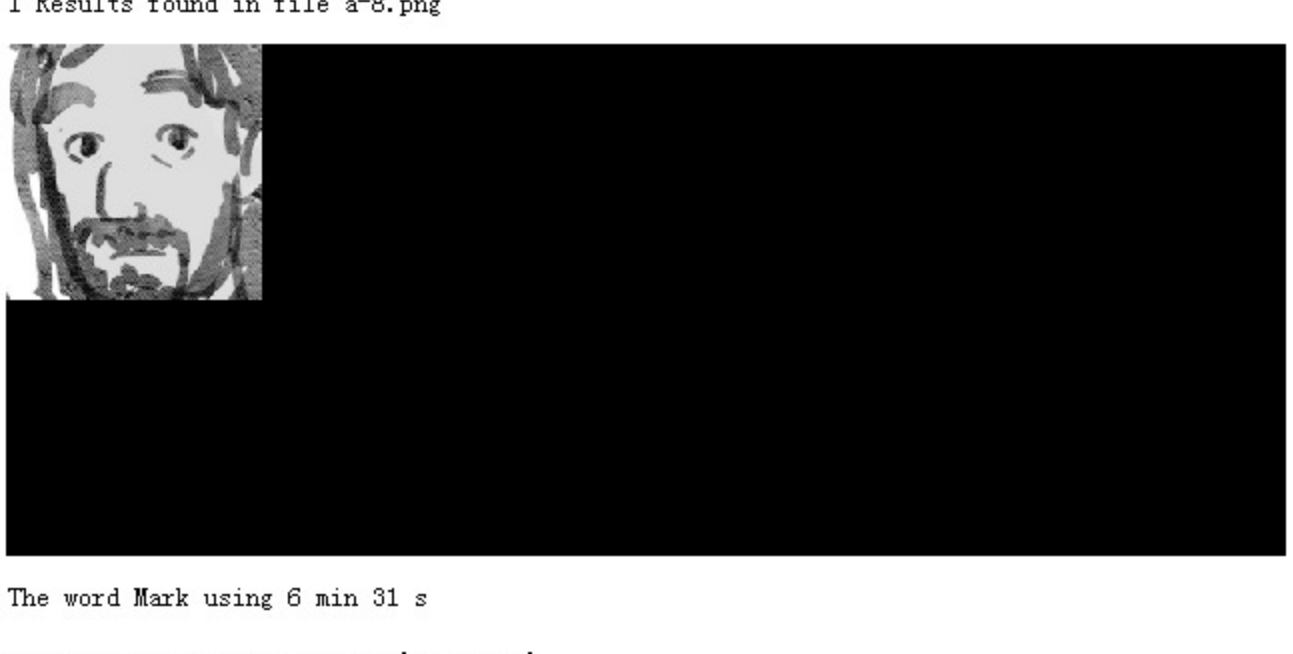
```
finding faces in a-3.png ...
2 Results found in file a-3.png
```



```
finding faces in a-5.png ...
8 Results found in file a-5.png
```



```
finding faces in a-8.png ...
1 Results found in file a-8.png
```

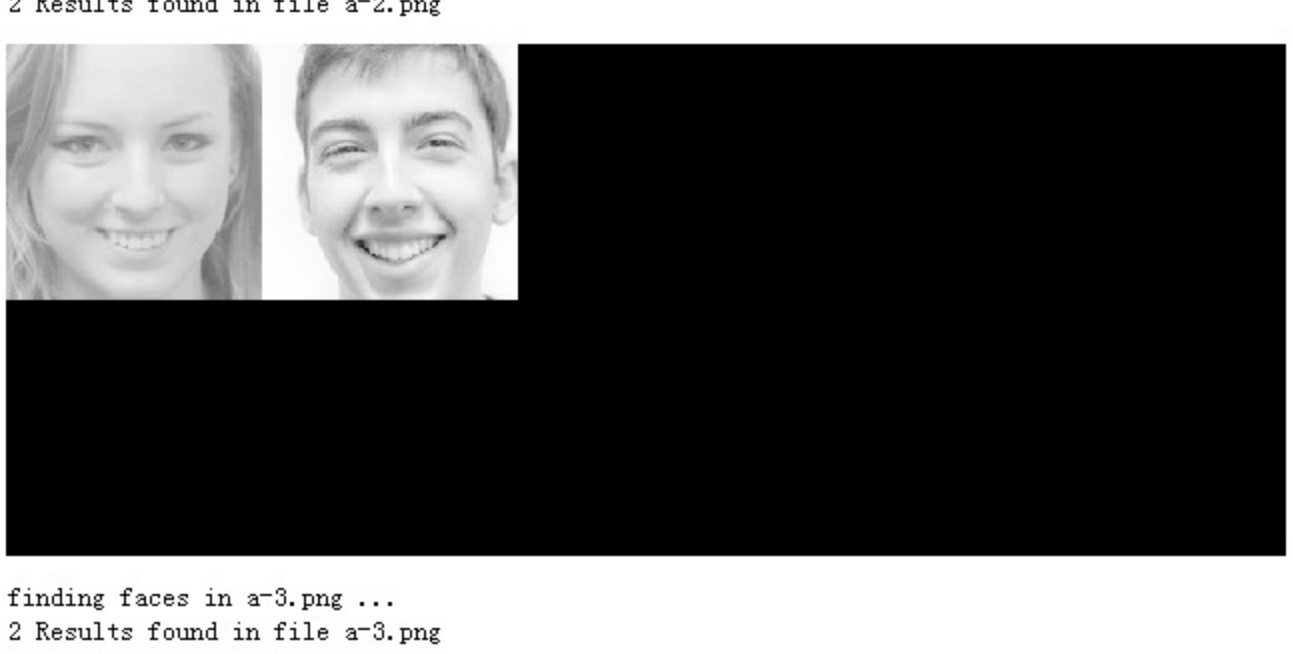


```
The word Mark using 6 min 31 s

The target word we chose is 'Michigan'
finding faces in a-0.png ...
8 Results found in file a-0.png
```

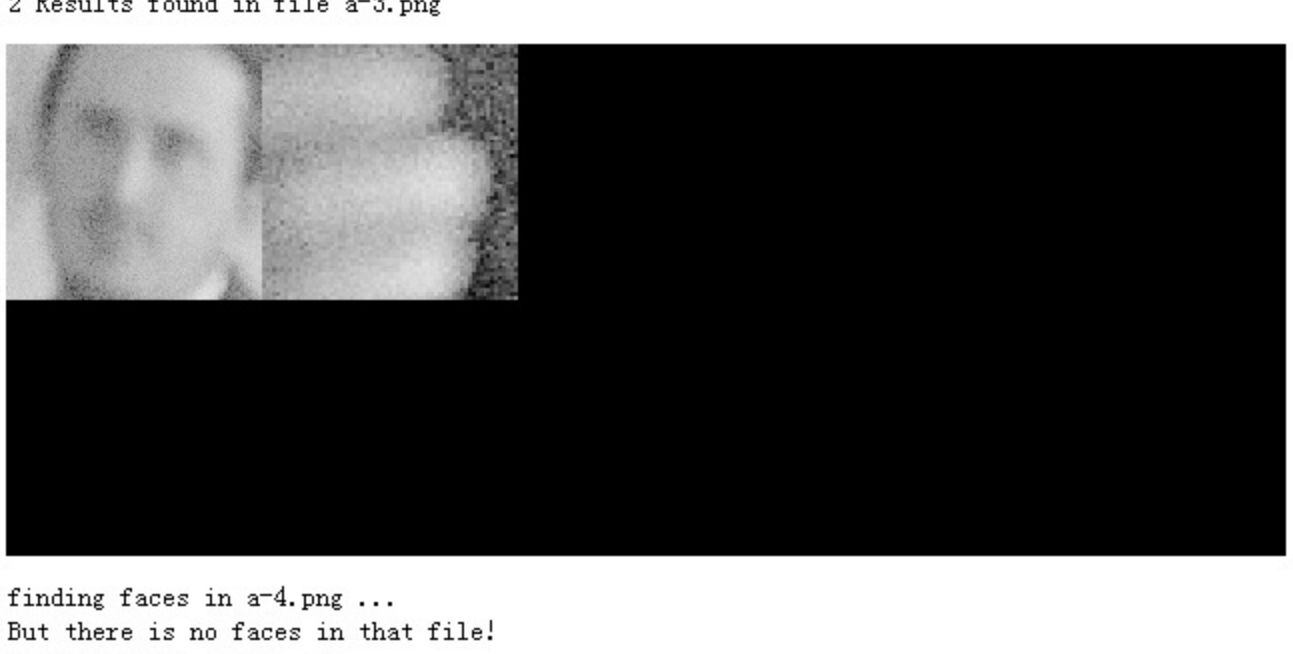
```
finding faces in a-1.png ...
5 Results found in file a-1.png
```

```
finding faces in a-10.png ...
1 Results found in file a-10.png
```

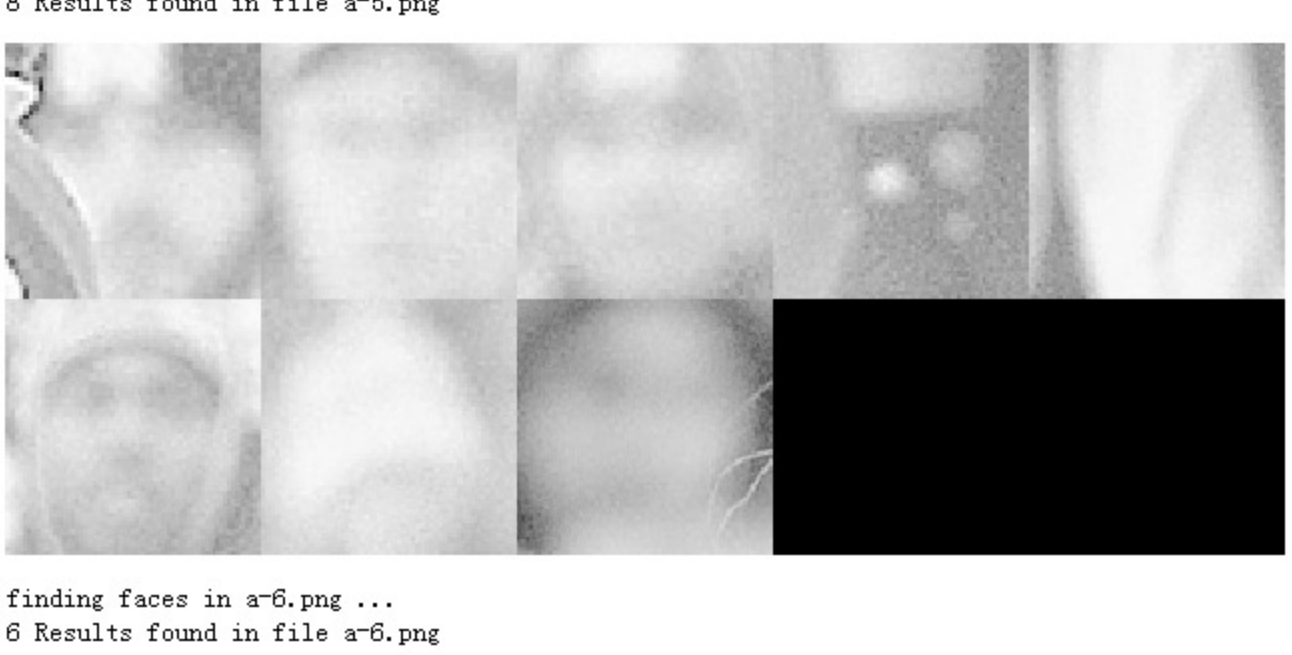
```
finding faces in a-11.png ...
But there is no faces in that file!
finding faces in a-2.png ...
2 Results found in file a-2.png
```



```
finding faces in a-3.png ...
2 Results found in file a-3.png
```



```
finding faces in a-4.png ...
But there is no faces in that file!
finding faces in a-5.png ...
8 Results found in file a-5.png
```

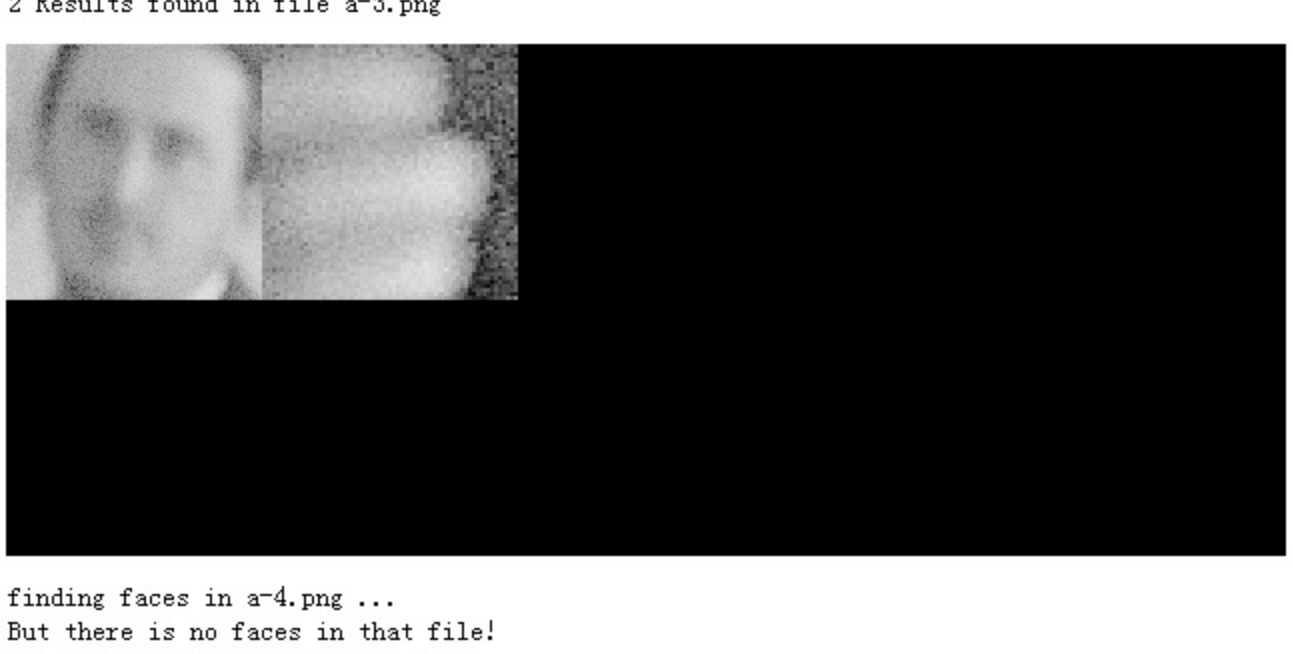


```
finding faces in a-6.png ...
6 Results found in file a-6.png
```

```
finding faces in a-7.png ...
1 Results found in file a-7.png
```

```
finding faces in a-8.png ...
1 Results found in file a-8.png
```

```
The word Michigan using 8 min 12 s

The target word we chose is 'serving'
finding faces in a-3.png ...
2 Results found in file a-3.png
```



```
finding faces in a-4.png ...
But there is no faces in that file!
The word serving using 2 min 27 s

The target word we chose is 'celebrating'
finding faces in a-0.png ...
8 Results found in file a-0.png
```

```
The word celebrating using 0 min 35 s

All finished,  using 48 min 35 s
```

In [ ]:

In [ ]: