Assignment Introduction

In this assignment you will have practiced the use of the libraries behind pillow, tesseract, and opencv in python to build a simple search application. In addition, you will have learned how to manipulate zip files with a new python library.

Part 1:

Importing Modules and Defining Necessary Functions

In [63]:
```
import zipfile
from PIL import Image
import pickle
import pytesseract
import cv2 as cv
import numpy as np
import pickle
import math

#%%
# Function for extracting images out of a given ZIP file and
def create_images_dict_from_zip (zip_dir = None):
    if type(zip_dir) != str: return None
    zf = zipfile.ZipFile(zip_dir, 'r')
    print(zf.namelist())
    images_dict = {}
    for fileName in zf.namelist():
        try:
            tempZip = zf.open(fileName)
            tempIMG = Image.open(tempZip)
            images_dict[fileName] = tempIMG
        except KeyError:
            print ('ERROR: Did not find %s in zip file' % fileName)
        else:
            print (fileName + ' is read successfully.')
    return images_dict

#%%
# Function for Thumbnail Printing
def build_montages(image_list, image_shape = (96, 96), montage_shape = (5, 3)):
    """
    ---------------------------------------------------------------------------
    author: Kyle Hounslow (from "imutils" module)
    ---------------------------------------------------------------------------
    Converts a list of single images into a list of 'montage' images of specifie
    A new montage image is started once rows and columns of montage image is fil
    Empty space of incomplete montage images are filled with black pixels
    ---------------------------------------------------------------------------
    :param image_list: python list of input images
    :param image_shape: tuple, size each image will be resized to for display (w
    :param montage_shape: tuple, shape of image montage (width, height)
    :return: list of montage images in numpy array format
    ---------------------------------------------------------------------------
    example usage:
    # load single image
    img = cv.imread('lena.jpg')
    # duplicate image 25 times
    num_imgs = 25
    img_list = []
    for i in xrange(num_imgs):
        img_list.append(img)
    # convert image list into a montage of 256x256 images tiled in a 5x5 montage
    montages = make_montages_of_images(img_list, (256, 256), (5, 5))
    # iterate through montages and display
    for montage in montages:
        cv.imshow('montage image', montage)
        cv.waitKey(0)
    ------------------------------------------------------------------------------
    """
    if len(image_shape) != 2:
        raise Exception('image shape must be list or tuple of length 2 (rows, co
    if len(montage_shape) != 2:
        raise Exception('montage shape must be list or tuple of length 2 (rows, 
    image_montages = []
    # start with black canvas to draw images onto
    montage_image = np.zeros(shape=(image_shape[1] * (montage_shape[1]), image_s
                          dtype=np.uint8)
    cursor_pos = [0, 0]
    start_new_img = False
    for img in image_list:
        if type(img).__module__ != np.__name__:
            raise Exception('input of type {} is not a valid numpy array'.format
        start_new_img = False
        img = cv.resize(img, image_shape)
        # draw image to black canvas
        montage_image[cursor_pos[1]:cursor_pos[1] + image_shape[1], cursor_pos[0
        cursor_pos[0] += image_shape[0]  # increment cursor x position
        if cursor_pos[0] >= montage_shape[0] * image_shape[0]:
            cursor_pos[1] += image_shape[1]  # increment cursor y position
            cursor_pos[0] = 0
            if cursor_pos[1] >= montage_shape[1] * image_shape[1]:
                cursor_pos = [0, 0]
                image_montages.append(montage_image)
                # reset black canvas
                montage_image = np.zeros(shape=(image_shape[1] * (montage_shape[
                                      dtype=np.uint8)
                start_new_img = True
    if start_new_img is False:
        image_montages.append(montage_image)  # add unfinished montage
    return image_montages

#%%
# Function for inquiring the string to search
def search_inside_newspaper(query_string, images_dict, texts_dict, faces_pos_dic
    nCols_for_montage = 5
    for fileName, text_elem in texts_dict.items():
        if not (query_string in text_elem): continue

        print('Results found in file', fileName)
        image_elem = np.array(images_dict[fileName])
        temp_faces_pos_list = faces_pos_dict[fileName]

        if len(temp_faces_pos_list) == 0: print('But there were no faces in that

        temp_faces_list = []
        for face_vec in temp_faces_pos_list:
            try:
                face_image_crop = image_elem[face_vec[1]:face_vec[1] + face_vec[
                                             face_vec[0]:face_vec[0] + face_vec[
                                             :]
                temp_faces_list.append(face_image_crop)

            except:
                face_image_crop = image_elem[face_vec[1]:face_vec[1] + face_vec[
                                             face_vec[0]:face_vec[0] + face_vec[
                temp_faces_list.append(face_image_crop)

        # print('len(temp_faces_list):', len(temp_faces_list))
        nRows_for_montage = math.ceil(len(temp_faces_list) / nCols_for_montage)
        montage_arrays = build_montages(image_list=temp_faces_list,
                                        montage_shape=(nCols_for_montage, nRows_

        # print('type(montage_arrays):', type(montage_arrays))
        # print('len(montage_arrays):', len(montage_arrays))
        montage_array = montage_arrays[0]
        display(Image.fromarray(montage_array))
        Image.fromarray(montage_array).save(fileName.split('.')[0] + '--Montage.
```

Part 2:

Pre-processing

In [44]:
```
# Performing ZIP file extraction
zip_dir_small_images = 'small_img.zip'
zip_dir_all_images = 'images.zip'
images_dict_small = create_images_dict_from_zip(zip_dir_small_images)
images_dict_all = create_images_dict_from_zip(zip_dir_all_images)
```

```
['a-0.png', 'a-1.png', 'a-2.png', 'a-3.png']
a-0.png is read successfully.
a-1.png is read successfully.
a-2.png is read successfully.
a-3.png is read successfully.
['a-0.png', 'a-1.png', 'a-10.png', 'a-11.png', 'a-12.png', 'a-13.png', 'a-2.pn
g', 'a-3.png', 'a-4.png', 'a-5.png', 'a-6.png', 'a-7.png', 'a-8.png', 'a-9.pn
g']
a-0.png is read successfully.
a-1.png is read successfully.
a-10.png is read successfully.
a-11.png is read successfully.
a-12.png is read successfully.
a-13.png is read successfully.
a-2.png is read successfully.
a-3.png is read successfully.
a-4.png is read successfully.
a-5.png is read successfully.
a-6.png is read successfully.
a-7.png is read successfully.
a-8.png is read successfully.
a-9.png is read successfully.
```

In [49]:
```
# Performing OCR on images
texts_dict_all = {}
texts_dict_small = {}
bin_thresh = 210

for fileName, image_elem in images_dict_all.items():
    bin_thresh, tempIMG = cv.threshold(np.array(image_elem), bin_thresh, 255, cv
    temp_text = pytesseract.image_to_string(np.array(tempIMG))
    # temp_text = pytesseract.image_to_string(np.array(image_elem.convert('L')))
    texts_dict_all[fileName] = temp_text.split()

for fileName, image_elem in images_dict_small.items():
    bin_thresh, tempIMG = cv.threshold(np.array(image_elem.convert('L')),
                                       bin_thresh, 255, cv.THRESH_BINARY)
    temp_text = pytesseract.image_to_string(tempIMG)
    texts_dict_small[fileName] = temp_text.split()
```

In [62]:
```
# Performing Face Detection
face_xml_dir = 'haarcascade_frontalface_default.xml'
face_cascade = cv.CascadeClassifier(face_xml_dir)
faces_pos_dict_all = {}
faces_pos_dict_small = {}
face_scaleFactor = 1.16
face_minNeighbors = 10

for fileName, image_elem in images_dict_all.items():
    temp_faces_pos_list = face_cascade.detectMultiScale(np.array(image_elem),
                                                        scaleFactor = face_scale
                                                        minNeighbors=face_minNei
    faces_pos_dict_all[fileName] = temp_faces_pos_list
for fileName, image_elem in images_dict_small.items():
    temp_faces_pos_list = face_cascade.detectMultiScale(np.array(image_elem),
                                                        scaleFactor = face_scale
                                                        minNeighbors=face_minNei
    faces_pos_dict_small[fileName] = temp_faces_pos_list
```

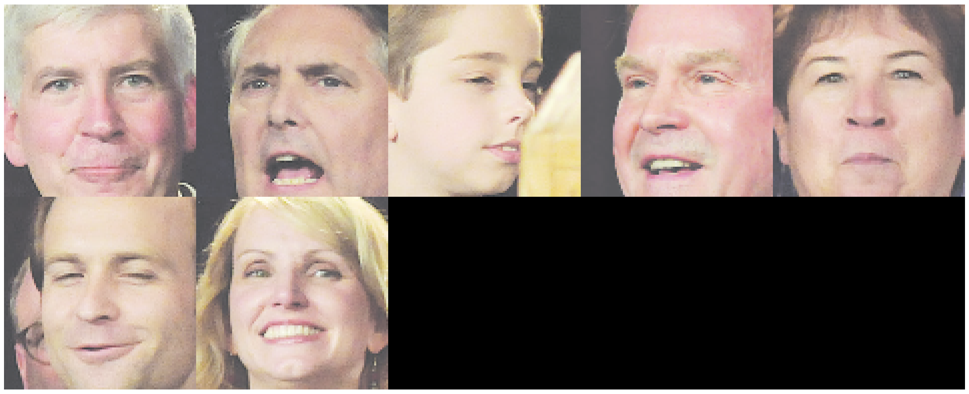
Part 3:

Performing Newspaper Search

1- Using the small_img.zip file, if I search for the string "Christopher" I see the following image:

In [75]:
```python
search_inside_newspaper(query_string='Christopher',
                        images_dict=images_dict_small,
                        texts_dict=texts_dict_small,
                        faces_pos_dict=faces_pos_dict_small)
```

```
Results found in file a-0.png
```

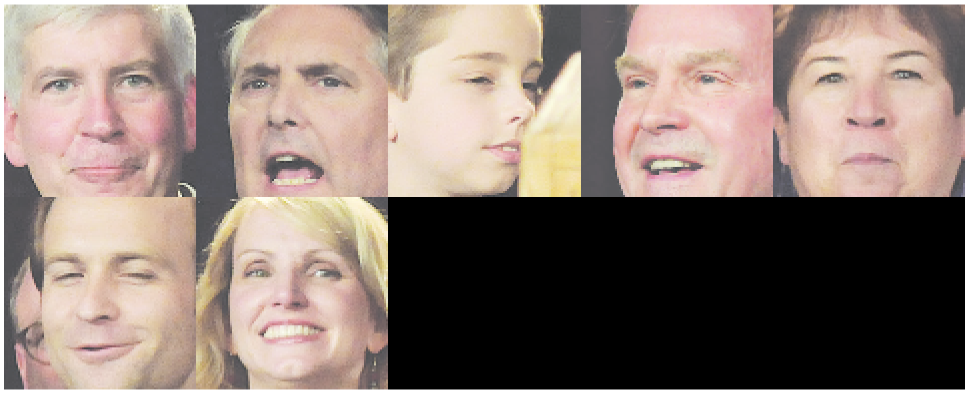


```
Results found in file a-3.png
```

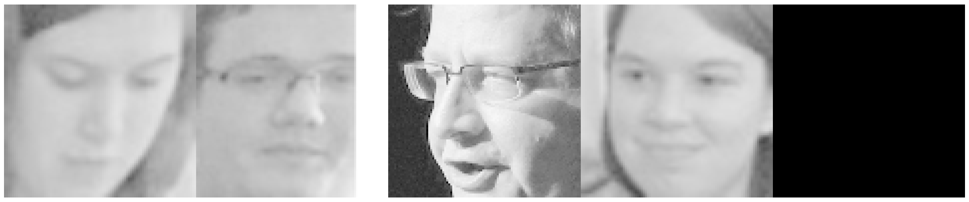
2- If I use the images.zip file and search for "Mark" I would see the following image: (note that there are times when there are no faces on a page, but a word is found!)

```
In [74]: search_inside_newspaper(query_string='Mark',
                                 images_dict=images_dict_all,
                                 texts_dict=texts_dict_all,
                                 faces_pos_dict=faces_pos_dict_all)
```

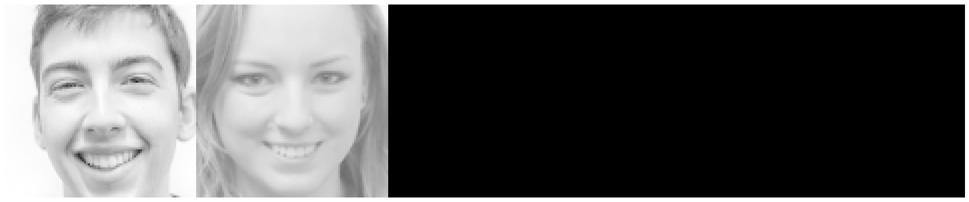
```
Results found in file a-0.png
```



```
Results found in file a-1.png
```



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



```
Results found in file a-3.png
```

```
Results found in file a-8.png
But there were no faces in that file!
```