# import dependencies

from IPython.display import display, Javascript, Image

from google.colab.output import eval\_js

from base64 import b64decode, b64encode

import cv2

import numpy as np

import PIL

import io

import html

import time

#......................................

# function to convert the JavaScript object into an OpenCV image

def js\_to\_image(js\_reply):

  """

  Params:

          js\_reply: JavaScript object containing image from webcam

  Returns:

          img: OpenCV BGR image

  """

  # decode base64 image

  image\_bytes = b64decode(js\_reply.split(',')[1])

  # convert bytes to numpy array

  jpg\_as\_np = np.frombuffer(image\_bytes, dtype=np.uint8)

  # decode numpy array into OpenCV BGR image

  img = cv2.imdecode(jpg\_as\_np, flags=1)

  return img

# function to convert OpenCV Rectangle bounding box image into base64 byte string to be overlayed on video stream

def bbox\_to\_bytes(bbox\_array):

  """

  Params:

          bbox\_array: Numpy array (pixels) containing rectangle to overlay on video stream.

  Returns:

        bytes: Base64 image byte string

  """

  # convert array into PIL image

  bbox\_PIL = PIL.Image.fromarray(bbox\_array, 'RGBA')

  iobuf = io.BytesIO()

  # format bbox into png for return

  bbox\_PIL.save(iobuf, format='png')

  # format return string

  bbox\_bytes = 'data:image/png;base64,{}'.format((str(b64encode(iobuf.getvalue()), 'utf-8')))

  return bbox\_bytes

#................................................

# initialize the Haar Cascade face detection model

face\_cascade = cv2.CascadeClassifier(cv2.samples.findFile(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml'))

#.................................................

def take\_photo(filename='photo.jpg', quality=0.8):

  js = Javascript('''

    async function takePhoto(quality) {

      const div = document.createElement('div');

      const capture = document.createElement('button');

      capture.textContent = 'Capture';

      div.appendChild(capture);

      const video = document.createElement('video');

      video.style.display = 'block';

      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);

      div.appendChild(video);

      video.srcObject = stream;

      await video.play();

      // Resize the output to fit the video element.

      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.

      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');

      canvas.width = video.videoWidth;

      canvas.height = video.videoHeight;

      canvas.getContext('2d').drawImage(video, 0, 0);

      stream.getVideoTracks()[0].stop();

      div.remove();

      return canvas.toDataURL('image/jpeg', quality);

    }

    ''')

  display(js)

  # get photo data

  data = eval\_js('takePhoto({})'.format(quality))

  # get OpenCV format image

  img = js\_to\_image(data)

  # grayscale img

  gray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)

  print(gray.shape)

  # get face bounding box coordinates using Haar Cascade

  faces = face\_cascade.detectMultiScale(gray)

  # draw face bounding box on image and blur the face image

  for (x,y,w,h) in faces:

      img[y:y+h, x:x+w] = cv2.GaussianBlur(img[y:y+h, x:x+w],(15,15),cv2.BORDER\_DEFAULT)

      cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

  # save image

  cv2.imwrite(filename, img)

  return filename

#..................................................

#Take picture 3 time

for x in range(3):

  try:

    filename = take\_photo('photo.jpg')

    print('Saved to {}'.format(filename))

    # Show the image which was just taken.

    display(Image(filename))

  except Exception as err:

    # Errors will be thrown if the user does not have a webcam or if they do not

    # grant the page permission to access it.

    print(str(err))

RESULT:





