```python
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 imag
def js_to_image(js_reply):
  """
  Params:
          js_reply: JavaScript object containing image from web
  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
def bbox_to_bytes(bbox_array):
  """
  Params:
          bbox_array: Numpy array (pixels) containing rectangl
  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(b64encod

  return bbox_bytes
face_cascade = cv2.CascadeClassifier(cv2.samples.findFile(cv2.

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);
```

NameError   NameError ✕ •••

👤 Please explain the error:

```
NameError: name 'video_stream' is n
```

◄ ▬▬▬▬▬▬▬▬▬▬▬▬ ►

✦ # Hypothesis

The error message `NameError: name 'video_stream' is not defined` indicates that the function `video_stream()` is being called before it has been defined or imported. This is likely because the user either forgot to define it, import it from a module, or accidentally misspelled it.

Suggested Changes

The code is attempting to use a function called `video_stream` but it has not been defined or imported from any module. It looks like the user is trying to use the `js2py` library for JavaScript-to-Python interoperability in the context of a web application likely dealing with video streams.

```python
!pip install js2py opencv-python

import cv2
import js2py
import numpy as np

# Assume these are the intended Ja
js_code = """
function video_stream() {
  // Implementation for starting v
  // ... (using browser APIs or ot
}

function video_frame(label_html, b
  // Implementation for fetching v
  // ... (potentially using browse
```

```
    const video = document.createElement('video');        return { "img": "..." }; // Pla
    video.style.display = 'block';                       }
    const stream = await navigator.mediaDevices.getUserMedia

    document.body.appendChild(div);                      function js_to_image(img_data) {
    div.appendChild(video);                                // Implementation for converting
    video.srcObject = stream;                              // ... (using appropriate conve
    await video.play();                                    return new Image(); // Placehol

    // Resize the output to fit the video element.       }
    google.colab.output.setIframeHeight(document.documentEle

    // Wait for Capture to be clicked.                   function bbox_to_bytes(bbox_array
    await new Promise((resolve) => capture.onclick = resolve   // Implementation for converting
                                                           // ... (using appropriate encod:
    const canvas = document.createElement('canvas');       return ""; // Placeholder, adap
    canvas.width = video.videoWidth;                     }
    canvas.height = video.videoHeight;                   """
    canvas.getContext('2d').drawImage(video, 0, 0);
    stream.getVideoTracks()[0].stop();                   # Execute the JavaScript code to
    div.remove();                                        context = js2py.EvalJs()
    return canvas.toDataURL('image/jpeg', quality);      context.execute(js_code)
  }
  ''')                                                   # Access the JavaScript functions
display(js)                                              video_stream = context.video_stre
                                                        video_frame = context.video_frame
# get photo data                                        js_to_image = context.js_to_image
data = eval_js('takePhoto({})'.format(quality))         bbox_to_bytes = context.bbox_to_by
# get OpenCV format image
img = js_to_image(data)                                 # Load the face detection cascade
# grayscale img                                         face_cascade = cv2.CascadeClassif:
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
print(gray.shape)                                       # start streaming video from webca
# get face bounding box coordinates using Haar Cascade  video_stream()
faces = face_cascade.detectMultiScale(gray)             # label for video
# draw face bounding box on image                       label_html = 'Capturing...'
for (x,y,w,h) in faces:                                 # initialze bounding box to empty
    img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2) bbox = ''
# save image
cv2.imwrite(filename, img)                              count = 0

return filename                                         while True:
                                                            js_reply = video_frame(label_t
try:                                                        if not js_reply:
  filename = take_photo('photo.jpg')                            break
  print('Saved to {}'.format(filename))
                                                            # convert JS response to Open(
  # Show the image which was just taken.                    img = js_to_image(js_reply["in
  display(Image(filename))
except Exception as err:                                    # create transparent overlay :
  # Errors will be thrown if the user does not have a webcam o  bbox_array = np.zeros([480,64(
  # grant the page permission to access it.
  print(str(err))

def video_stream():
  js = Javascript('''
```

```
    var video;
    var div = null;
    var stream;
    var captureCanvas;
    var imgElement;
    var labelElement;

    var pendingResolve = null;
    var shutdown = false;

    function removeDom() {
       stream.getVideoTracks()[0].stop();
       video.remove();
       div.remove();
       video = null;
       div = null;
       stream = null;
       imgElement = null;
       captureCanvas = null;
       labelElement = null;
    }

    function onAnimationFrame() {
      if (!shutdown) {
        window.requestAnimationFrame(onAnimationFrame);
      }
      if (pendingResolve) {
        var result = "";
        if (!shutdown) {
          captureCanvas.getContext('2d').drawImage(video, 0, 0
          result = captureCanvas.toDataURL('image/jpeg', 0.8)
        }
        var lp = pendingResolve;
        pendingResolve = null;
        lp(result);
      }
    }

    async function createDom() {
      if (div !== null) {
        return stream;
      }

      div = document.createElement('div');
      div.style.border = '2px solid black';
      div.style.padding = '3px';
      div.style.width = '100%';
      div.style.maxWidth = '600px';
      document.body.appendChild(div);

      const modelOut = document.createElement('div');
      modelOut.innerHTML = "Status:";
      labelElement = document.createElement('span');
      labelElement.innerText = 'No data';
      labelElement.style.fontWeight = 'bold';
      modelOut.appendChild(labelElement);
```

```
# grayscale image for face det
gray = cv2.cvtColor(img, cv2.(

# get face region coordinates
faces = face_cascade.detectMul
# get face bounding box for ov
for (x,y,w,h) in faces:
   bbox_array = cv2.rectangle(l

bbox_array[:,:,3] = (bbox_arra
# convert overlay of bbox into
bbox_bytes = bbox_to_bytes(bbo
# update bbox so next frame ge
bbox = bbox_bytes
```

[Use code with caution](#)

Explanation of Changes:

1. **Install necessary libraries:** `!pip install js2py opencv-python` to install required libraries if not already present.
2. **Import necessary libraries:** `import cv2`, `import js2py`, `import numpy as np` to import the required modules.
3. **Define or import missing functions:** The code snippet introduces placeholders for `video_stream`, `video_frame`, `js_to_image`, and `bbox_to_bytes`. You should replace these placeholders with actual implementations specific to your environment or library usage. We used `js2py` to demonstrate how the user could define these functions using JavaScript syntax and access those functions within Python.
4. **Load face cascade:** The line `face_cascade = cv2.CascadeClassifier(cv2.data. + 'haarcascade_frontalface_defaul` loads the Haar Cascade classifier

```
        div.appendChild(modelOut);

        video = document.createElement('video');
        video.style.display = 'block';
        video.width = div.clientWidth - 6;
        video.setAttribute('playsinline', '');
        video.onclick = () => { shutdown = true; };
        stream = await navigator.mediaDevices.getUserMedia(
            {video: { facingMode: "environment"}});
        div.appendChild(video);

        imgElement = document.createElement('img');
        imgElement.style.position = 'absolute';
        imgElement.style.zIndex = 1;
        imgElement.onclick = () => { shutdown = true; };
        div.appendChild(imgElement);

        const instruction = document.createElement('div');
        instruction.innerHTML =
            '' +
            'When finished, click here or on the video to stop t|
        div.appendChild(instruction);
        instruction.onclick = () => { shutdown = true; };

        video.srcObject = stream;
        await video.play();

        captureCanvas = document.createElement('canvas');
        captureCanvas.width = 640; //video.videoWidth;
        captureCanvas.height = 480; //video.videoHeight;
        window.requestAnimationFrame(onAnimationFrame);

        return stream;
    }
    async function stream_frame(label, imgData) {
        if (shutdown) {
            removeDom();
            shutdown = false;
            return '';
        }

        var preCreate = Date.now();
        stream = await createDom();

        var preShow = Date.now();
        if (label != "") {
            labelElement.innerHTML = label;
        }

        if (imgData != "") {
            var videoRect = video.getClientRects()[0];
            imgElement.style.top = videoRect.top + "px";
            imgElement.style.left = videoRect.left + "px";
            imgElement.style.width = videoRect.width + "px";
            imgElement.style.height = videoRect.height + "px";
            imgElement.src = imgData;
```

for face detection using OpenCV. Make sure the path is correct.

▶ Sources

Rate this answer   👍 👎

```
        imgElement.src = imgData;
      }

      var preCapture = Date.now();
      var result = await new Promise(function(resolve, reject)
        pendingResolve = resolve;
      });
      shutdown = false;

      return {'create': preShow - preCreate,
              'show': preCapture - preShow,
              'capture': Date.now() - preCapture,
              'img': result};
    }
    ''')

  display(js)

def video_frame(label, bbox):
  data = eval_js('stream_frame("{}", "{}")'.format(label, bbox
  return data

video_stream()
# label for video
label_html = 'Capturing...'
# initialze bounding box to empty
bbox = ''
count = 0
while True:
    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break

    # convert JS response to OpenCV Image
    img = js_to_image(js_reply["img"])

    # create transparent overlay for bounding box
    bbox_array = np.zeros([480,640,4], dtype=np.uint8)

    # grayscale image for face detection
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

    # get face region coordinates
    faces = face_cascade.detectMultiScale(gray)
    # get face bounding box for overlay
    for (x,y,w,h) in faces:
      bbox_array = cv2.rectangle(bbox_array,(x,y),(x+w,y+h),(2!

    bbox_array[:,:,3] = (bbox_array.max(axis = 2) > 0 ).astype
    # convert overlay of bbox into bytes
    bbox_bytes = bbox_to_bytes(bbox_array)
    # update bbox so next frame gets new overlay
    bbox = bbox_bytes
```
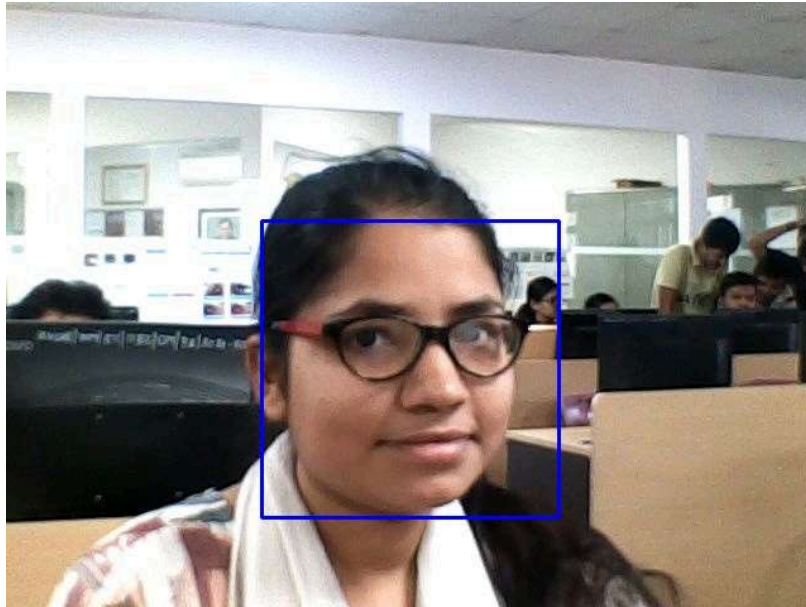
```
(480, 640)
Saved to photo.jpg
```


Enter a prompt here

0 / 2000

Responses may display inaccurate or offensive
information that doesn't represent Google's views.
Learn more