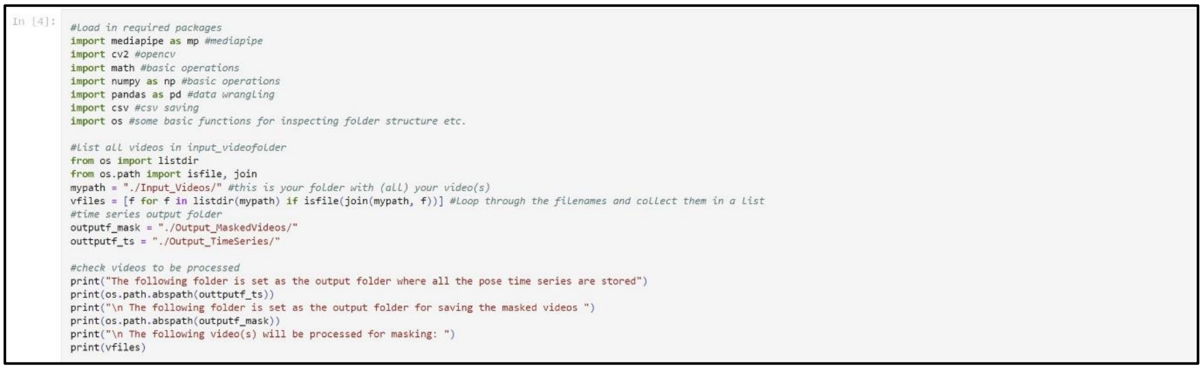
Supplementary material / Additional Information:

Here we describe the steps to setup Masked-Piper and a demo usage

Masked-Piper Setup and Usage

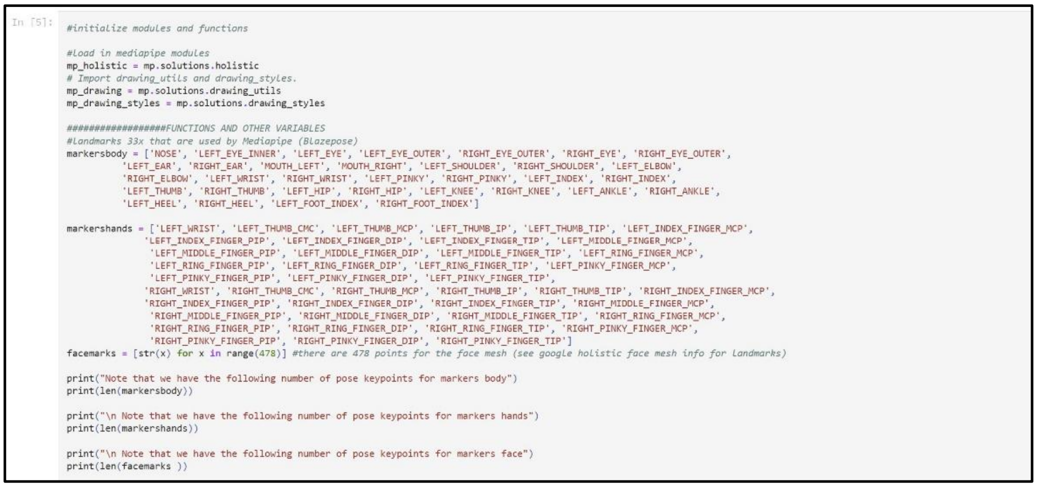
Install all the packages in requirements.txt. Then move the videos that you want to mask into the **input folder**. Then run the code as shown on the GitHub Page with Examples (and shown below). Running this code will loop through all the videos in the input folder and save all the results in the **output folders**.



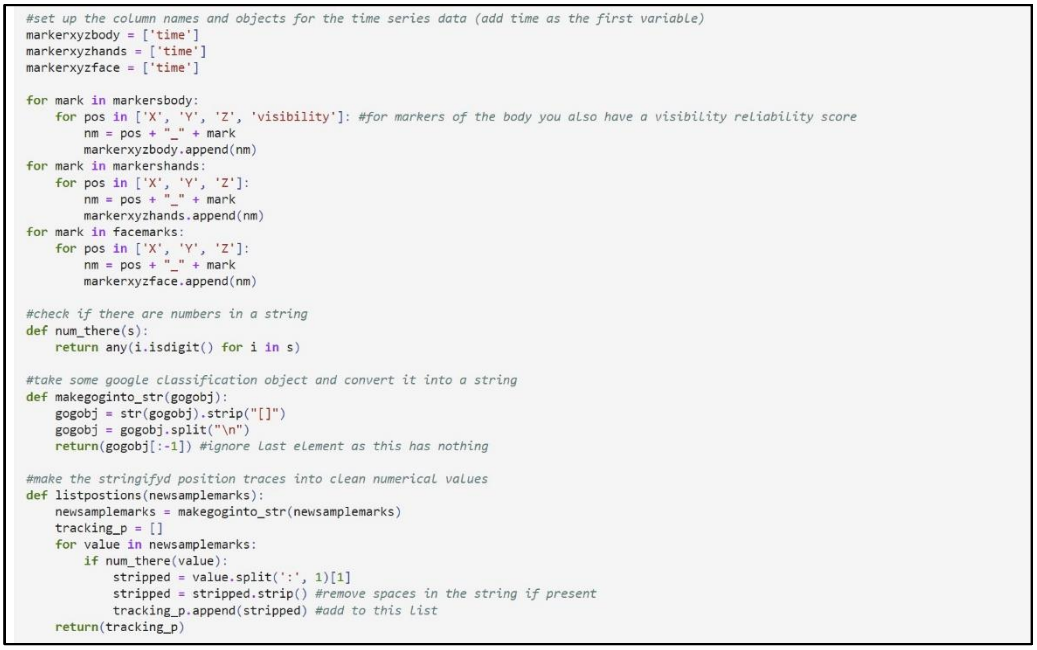
```
#Load in required packages
import mediapipe as mp #mediapipe
import cv2 #opencv
import math #basic operations
import numpy as np #basic operations
import pandas as pd #data wrangling
import csv #csv saving
import os #some basic functions for inspecting folder structure etc.

#List all videos in input_videofolder
from os import listdir
from os.path import isfile, join
mypath = "./Input_Videos/" #this is your folder with (all) your video(s)
vfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))] #Loop through the filenames and collect them in a list
#time series output folder
outputf_mask = "./Output_MaskedVideos/"
outtputf_ts = "./Output_TimeSeries/"

#check videos to be processed
print("The following folder is set as the output folder where all the pose time series are stored")
print(os.path.abspath(outtputf_ts))
print("\n The following folder is set as the output folder for saving the masked videos ")
print(os.path.abspath(outputf_mask))
print("\n The following video(s) will be processed for masking: ")
print(vfiles)
```

```
The following folder is set as the output folder where all the pose time seri
es are stored
D:\TowardsMultimodalOpenScience\Output_TimeSeries
```

```
 The following folder is set as the output folder for saving the masked video
s
D:\TowardsMultimodalOpenScience\Output_MaskedVideos

 The following video(s) will be processed for masking:
['1413451-11105600-11163240_1a1_1.mp4', 'sample.mp4', 'ted_kid.mp4']
```



```
In [5]:
#initialize modules and functions

#load in mediapipe modules
mp_holistic = mp.solutions.holistic
# Import drawing_utils and drawing_styles.
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

##################FUNCTIONS AND OTHER VARIABLES
#landmarks 33x that are used by Mediapipe (Blazepose)
markersbody = ['NOSE', 'LEFT_EYE_INNER', 'LEFT_EYE', 'LEFT_EYE_OUTER', 'RIGHT_EYE_OUTER', 'RIGHT_EYE', 'RIGHT_EYE_OUTER',
          'LEFT_EAR', 'RIGHT_EAR', 'MOUTH_LEFT', 'MOUTH_RIGHT', 'LEFT_SHOULDER', 'RIGHT_SHOULDER', 'LEFT_ELBOW',
          'RIGHT_ELBOW', 'LEFT_WRIST', 'RIGHT_WRIST', 'LEFT_PINKY', 'RIGHT_PINKY', 'LEFT_INDEX', 'RIGHT_INDEX',
          'LEFT_THUMB', 'RIGHT_THUMB', 'LEFT_HIP', 'RIGHT_HIP', 'LEFT_KNEE', 'RIGHT_KNEE', 'LEFT_ANKLE', 'RIGHT_ANKLE',
          'LEFT_HEEL', 'RIGHT_HEEL', 'LEFT_FOOT_INDEX', 'RIGHT_FOOT_INDEX']

markershands = ['LEFT_WRIST', 'LEFT_THUMB_CMC', 'LEFT_THUMB_MCP', 'LEFT_THUMB_IP', 'LEFT_THUMB_TIP', 'LEFT_INDEX_FINGER_MCP',
              'LEFT_INDEX_FINGER_PIP', 'LEFT_INDEX_FINGER_DIP', 'LEFT_INDEX_FINGER_TIP', 'LEFT_MIDDLE_FINGER_MCP',
               'LEFT_MIDDLE_FINGER_PIP', 'LEFT_MIDDLE_FINGER_DIP', 'LEFT_MIDDLE_FINGER_TIP', 'LEFT_RING_FINGER_MCP',
               'LEFT_RING_FINGER_PIP', 'LEFT_RING_FINGER_DIP', 'LEFT_RING_FINGER_TIP', 'LEFT_PINKY_FINGER_MCP',
               'LEFT_PINKY_FINGER_PIP', 'LEFT_PINKY_FINGER_DIP', 'LEFT_PINKY_FINGER_TIP',
              'RIGHT_WRIST', 'RIGHT_THUMB_CMC', 'RIGHT_THUMB_MCP', 'RIGHT_THUMB_IP', 'RIGHT_THUMB_TIP', 'RIGHT_INDEX_FINGER_MCP',
              'RIGHT_INDEX_FINGER_PIP', 'RIGHT_INDEX_FINGER_DIP', 'RIGHT_INDEX_FINGER_TIP', 'RIGHT_MIDDLE_FINGER_MCP',
               'RIGHT_MIDDLE_FINGER_PIP', 'RIGHT_MIDDLE_FINGER_DIP', 'RIGHT_MIDDLE_FINGER_TIP', 'RIGHT_RING_FINGER_MCP',
               'RIGHT_RING_FINGER_PIP', 'RIGHT_RING_FINGER_DIP', 'RIGHT_RING_FINGER_TIP', 'RIGHT_PINKY_FINGER_MCP',
               'RIGHT_PINKY_FINGER_PIP', 'RIGHT_PINKY_FINGER_DIP', 'RIGHT_PINKY_FINGER_TIP']
facemarks = [str(x) for x in range(478)] #there are 478 points for the face mesh (see google holistic face mesh info for landmarks)

print("Note that we have the following number of pose keypoints for markers body")
print(len(markersbody))

print("\n Note that we have the following number of pose keypoints for markers hands")
print(len(markershands))

print("\n Note that we have the following number of pose keypoints for markers face")
print(len(facemarks ))
```

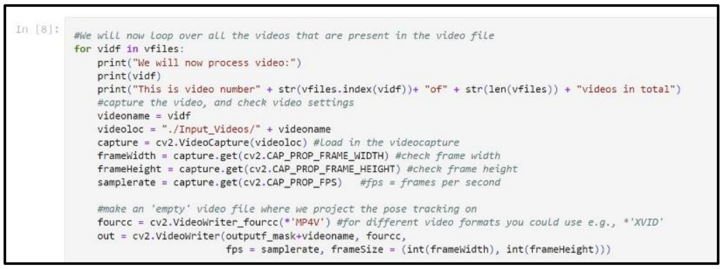


```
#set up the column names and objects for the time series data (add time as the first variable)
markerxyzbody = ['time']
markerxyzhands = ['time']
markerxyzface = ['time']

for mark in markersbody:
    for pos in ['X', 'Y', 'Z', 'visibility']: #for markers of the body you also have a visibility reliability score
        nm = pos + "_" + mark
        markerxyzbody.append(nm)
for mark in markershands:
    for pos in ['X', 'Y', 'Z']:
        nm = pos + "_" + mark
        markerxyzhands.append(nm)
for mark in facemarks:
    for pos in ['X', 'Y', 'Z']:
        nm = pos + "_" + mark
        markerxyzface.append(nm)

#check if there are numbers in a string
def num_there(s):
    return any(i.isdigit() for i in s)

#take some google classification object and convert it into a string
def makegoginto_str(gogobj):
    gogobj = str(gogobj).strip("[]")
    gogobj = gogobj.split("\n")
    return(gogobj[:-1]) #ignore last element as this has nothing

#make the stringifyd position traces into clean numerical values
def listpostions(newsamplemarks):
    newsamplemarks = makegoginto_str(newsamplemarks)
    tracking_p = []
    for value in newsamplemarks:
        if num_there(value):
            stripped = value.split(':', 1)[1]
            stripped = stripped.strip() #remove spaces in the string if present
            tracking_p.append(stripped) #add to this list
    return(tracking_p)
```

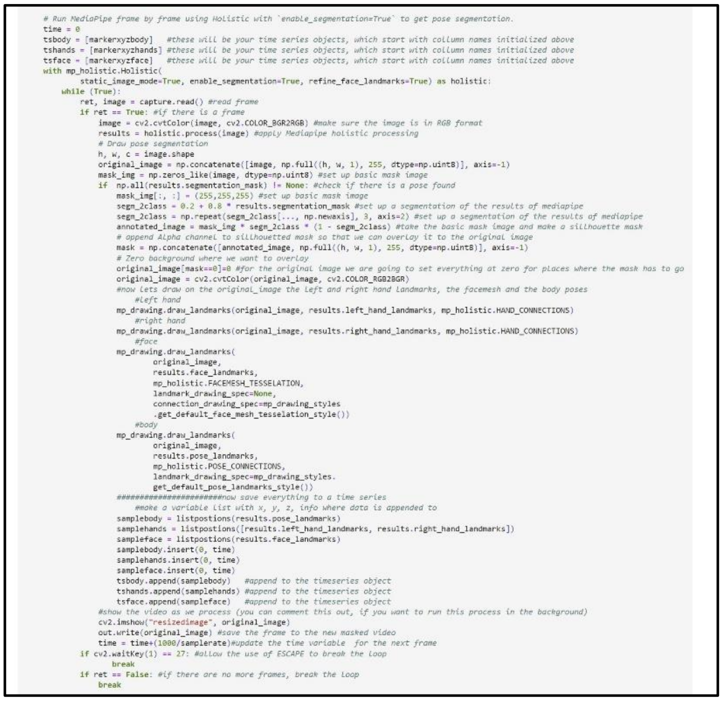
```
Note that we have the following number of pose keypoints for markers body
33

 Note that we have the following number of pose keypoints for markers hands
42

 Note that we have the following number of pose keypoints for markers face
478
```

Main Procedure of Masked-Piper

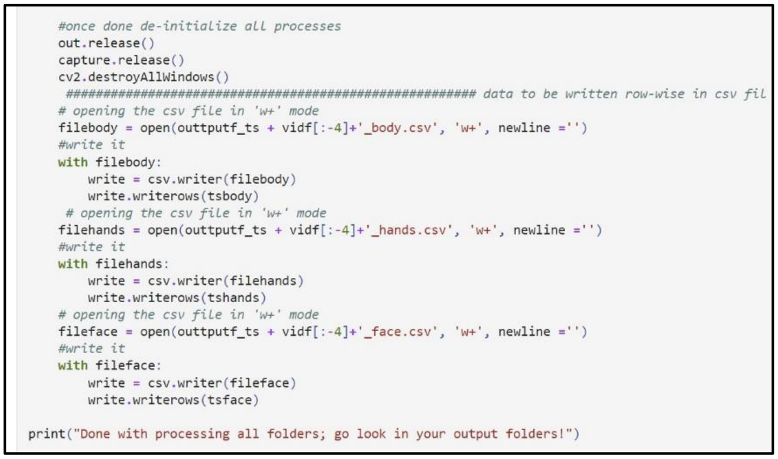
The following chunk of code loops through all the videos loaded into the input folder, assesses each frame for body poses, extracts kinematic info. Next, the code masks the body in a new frame that preserves the background, projecting the kinematic information on the mask. In addition, the code stores the kinematic information for that frame into the time series .csv for the hand + body + face.



```
In [8]:
    #We will now loop over all the videos that are present in the video file
    for vidf in vfiles:
        print("We will now process video:")
        print(vidf)
        print("This is video number" + str(vfiles.index(vidf))+ "of" + str(len(vfiles)) + "videos in total")
        #capture the video, and check video settings
        videoname = vidf
        videoloc = "./Input_Videos/" + videoname
        capture = cv2.VideoCapture(videoloc) #load in the videocapture
        frameWidth = capture.get(cv2.CAP_PROP_FRAME_WIDTH) #check frame width
        frameHeight = capture.get(cv2.CAP_PROP_FRAME_HEIGHT) #check frame height
        samplerate = capture.get(cv2.CAP_PROP_FPS)   #fps = frames per second

        #make an 'empty' video file where we project the pose tracking on
        fourcc = cv2.VideoWriter_fourcc(*'MP4V') #for different video formats you could use e.g., *'XVID'
        out = cv2.VideoWriter(outputf_mask+videoname, fourcc,
                              fps = samplerate, frameSize = (int(frameWidth), int(frameHeight)))
```



```
# Run MediaPipe frame by frame using Holistic with `enable_segmentation=True` to get pose segmentation.
time = 0
tsbody = [markerxyzbody]   #these will be your time series objects, which start with collumn names initialized above
tshands = [markerxyzhands] #these will be your time series objects, which start with collumn names initialized above
tsface = [markerxyzface]   #these will be your time series objects, which start with collumn names initialized above
with mp_holistic.Holistic(
        static_image_mode=True, enable_segmentation=True, refine_face_landmarks=True) as holistic:
    while (True):
        ret, image = capture.read() #read frame
        if ret == True: #if there is a frame
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) #make sure the image is in RGB format
            results = holistic.process(image) #apply Mediapipe holistic processing
            # Draw pose segmentation
            h, w, c = image.shape
            original_image = np.concatenate([image, np.full((h, w, 1), 255, dtype=np.uint8)], axis=-1)
            mask_img = np.zeros_like(image, dtype=np.uint8) #set up basic mask image
            if  np.all(results.segmentation_mask) != None: #check if there is a pose found
                mask_img[:, :] = (255,255,255) #set up basic mask image
                segm_2class = 0.2 + 0.8 * results.segmentation_mask #set up a segmentation of the results of mediapipe
                segm_2class = np.repeat(segm_2class[..., np.newaxis], 3, axis=2) #set up a segmentation of the results of mediapipe
                annotated_image = mask_img * segm_2class * (1 - segm_2class) #take the basic mask image and make a sillhouette mask
                # append Alpha channel to sillhouetted mask so that we can overlay it to the original image
                mask = np.concatenate([annotated_image, np.full((h, w, 1), 255, dtype=np.uint8)], axis=-1)
                # Zero background where we want to overlay
                original_image[mask==0]=0 #for the original image we are going to set everything at zero for places where the mask has to go
                original_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2BGR)
                #now lets draw on the original_image the left and right hand landmarks, the facemesh and the body poses
                    #left hand
                mp_drawing.draw_landmarks(original_image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS)
                    #right hand
                mp_drawing.draw_landmarks(original_image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS)
                    #face
                mp_drawing.draw_landmarks(
                        original_image,
                        results.face_landmarks,
                        mp_holistic.FACEMESH_TESSELATION,
                        landmark_drawing_spec=None,
                        connection_drawing_spec=mp_drawing_styles
                        .get_default_face_mesh_tesselation_style())
                    #body
                mp_drawing.draw_landmarks(
                        original_image,
                        results.pose_landmarks,
                        mp_holistic.POSE_CONNECTIONS,
                        landmark_drawing_spec=mp_drawing_styles.
                        get_default_pose_landmarks_style())
                #######################now save everything to a time series
                    #make a variable list with x, y, z, info where data is appended to
                samplebody = listpostions(results.pose_landmarks)
                samplehands = listpostions([results.left_hand_landmarks, results.right_hand_landmarks])
                sampleface = listpostions(results.face_landmarks)
                samplebody.insert(0, time)
                samplehands.insert(0, time)
                sampleface.insert(0, time)
                tsbody.append(samplebody)   #append to the timeseries object
                tshands.append(samplehands) #append to the timeseries object
                tsface.append(sampleface)   #append to the timeseries object
            #show the video as we process (you can comment this out, if you want to run this process in the background)
            cv2.imshow("resizedimage", original_image)
            out.write(original_image) #save the frame to the new masked video
            time = time+(1000/samplerate)#update the time variable  for the next frame
        if cv2.waitKey(1) == 27: #allow the use of ESCAPE to break the loop
              break
        if ret == False: #if there are no more frames, break the loop
            break
```



```
    #once done de-initialize all processes
    out.release()
    capture.release()
    cv2.destroyAllWindows()
     ######################################################## data to be written row-wise in csv fil
    # opening the csv file in 'w+' mode
    filebody = open(outtputf_ts + vidf[:-4]+'_body.csv', 'w+', newline ='')
    #write it
    with filebody:
        write = csv.writer(filebody)
        write.writerows(tsbody)
     # opening the csv file in 'w+' mode
    filehands = open(outtputf_ts + vidf[:-4]+'_hands.csv', 'w+', newline ='')
    #write it
    with filehands:
        write = csv.writer(filehands)
        write.writerows(tshands)
    # opening the csv file in 'w+' mode
    fileface = open(outtputf_ts + vidf[:-4]+'_face.csv', 'w+', newline ='')
    #write it
    with fileface:
        write = csv.writer(fileface)
        write.writerows(tsface)

print("Done with processing all folders; go look in your output folders!")
```

```
We will now process video:
```

```
1413451-11105600-11163240_1a1_1.mp4
This is video number0of3videos in total
We will now process video:
sample.mp4
This is video number1of3videos in total
We will now process video:
ted_kid.mp4
This is video number2of3videos in total
Done with processing all folders; go look in your output folders!

Done with processing all folders; results are in your output folders!
```