

# Detector de Sono

# Etapas

1. MediaPipe
2. Mapeando os olhos
3. Identificando a boca
4. Detecção de sono

# 1. MediaPipe

- ✓ Apresentação
- ✓ Preparamos o ambiente
- ✓ Configuramos o ambiente
- ✓ Captura ao vivo
- ✓ MediaPipe Face Mesh

## 2.Mapeamento os olhos

- Coordenadas da Face
- Análise dos olhos #1
- Análise dos olhos #2
- **Cálculo do EAR**
- **Cálculos do Tempo**

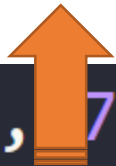
[illegible]

# A Fórmula EAR

```
def calculo_ear(face, p_olho_dir, p_olho_esq):
```

- Definir uma função
  - Define um nome calculo\_ear( )
  - Definimos parâmetros (face, p\_olho\_dir, p\_olho\_esq)

```
p_olho_esq = [385, 380, 387, 73, 362, 263]
```



```
# olho direito
```

```
p_olho_dir = [160, 144, 158, 153, 33, 133]
```

```
p_olhos = p_olho_esq + p_olho_dir
```

```
#FIXME: normalização para pixel
face = face_landmarks.landmark
for id_coord, coord_xyz in enumerate(face):
    if id_coord in p_olhos:
        coord_cv = mp_drawing._normalized_to_pixel_c
        cv2.circle(frame, coord_cv, 2, (255, 0, 0), -1)
#FIXME: chamada do EAR e print
ear = calculo_ear(face, p_olho_dir, p_olho_esq)
```

```
p_olho_dir = [160,144,158,153,33,133]
```



```
#FIXME: chamada do EAR e print  
ear = calculo_ear(face, p_olho_dir, p_olho_esq)
```



```
p_olho_esq = [385,380,387,373,362,263]
```



```
try:
```

```
except:
```

```
    # zerando
```

```
    ear_esq = 0.0
```

```
    ear_dir = 0.0
```

```
face = np.array([[coord.x, coord.y] for coord in face])
```



```
for item in coleccion:  
    print(item)
```

```
face = np.array([[coord.x, coord.y] for coord in face])
```

```
# face_esq  
face_esq = face[p_olho_esq, :]  
# face_dir  
face_dir = face[p_olho_dir, :]
```

# Matemática

- O **Teorema de Pitágoras** relaciona o comprimento dos lados do triângulo retângulo.
- O **Teorema de Pitágoras** relaciona o comprimento dos lados do triângulo retângulo.
- É usado para determinar a medida desconhecida de um lado, uma vez conhecidas as medidas dos outros dois lados.
- O enunciado desse teorema é:
- **“A hipotenusa ao quadrado é igual a soma dos quadrados dos catetos.”**

Em forma de equação, a fórmula do Teorema de Pitágoras é:

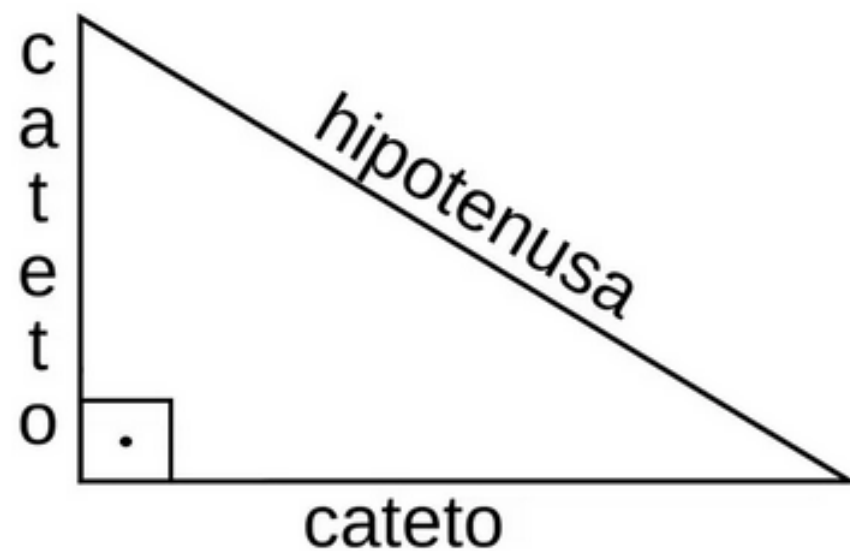
$$a^2 = b^2 + c^2$$

Sendo,

**a:** hipotenusa

**b:** cateto

**c:** cateto



Triângulo retângulo. Os catetos formam 90°.

# Distância Euclidiana

A distância euclidiana é um conceito muito próximo do Teorema de Pitágoras! Assim como no Teorema de Pitágoras, a distância euclidiana mede a "distância direta" entre dois pontos em um espaço. É muito usada em ciência de dados e inteligência artificial, principalmente para calcular a similaridade ou diferença entre vetores de dados.


```
np.linalg.norm(face_esq[0] - face_esq[1])
```

- **Malha Facial 3D**
- **Diferença (subtração) -**

```
ear_esq = (np.linalg.norm(face_esq[0] - face_esq[1])
+ np.linalg.norm(face_esq[2] - face_esq[3]))
```

## 2.1. Description of features

For every video frame, the eye landmarks are detected. The eye aspect ratio (EAR) between height and width of the eye is computed.

$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}, \quad (1)$$


```
/ (2 * (np.linalg.norm(face_esq[4] - face_esq[5])))
```



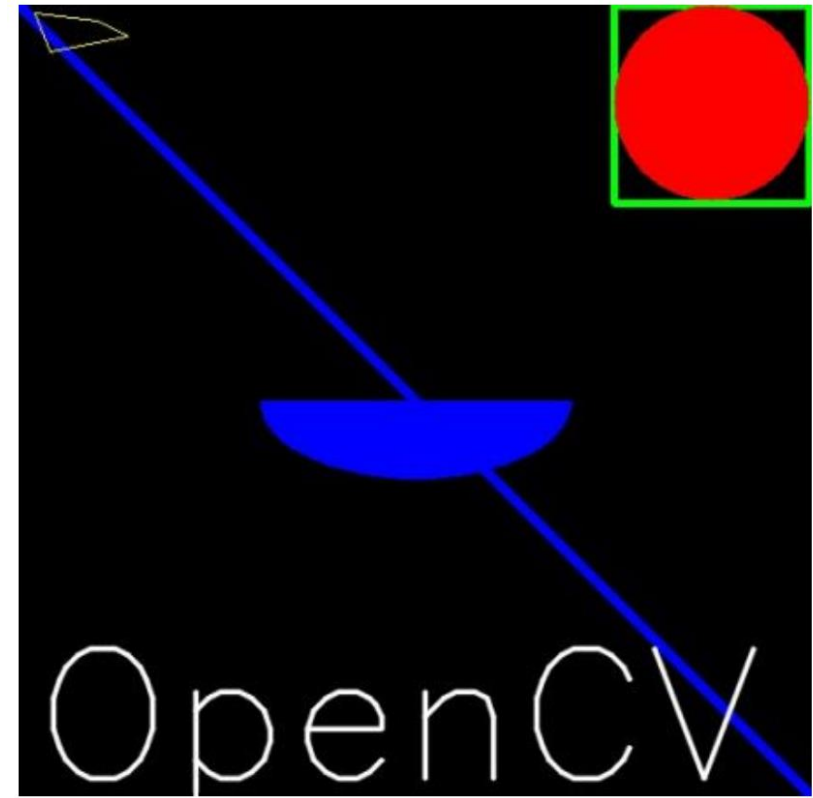
```
#fórmula
```

```
media_ear = (ear_esq + ear_dir) / 2
```

```
return media_ear
```

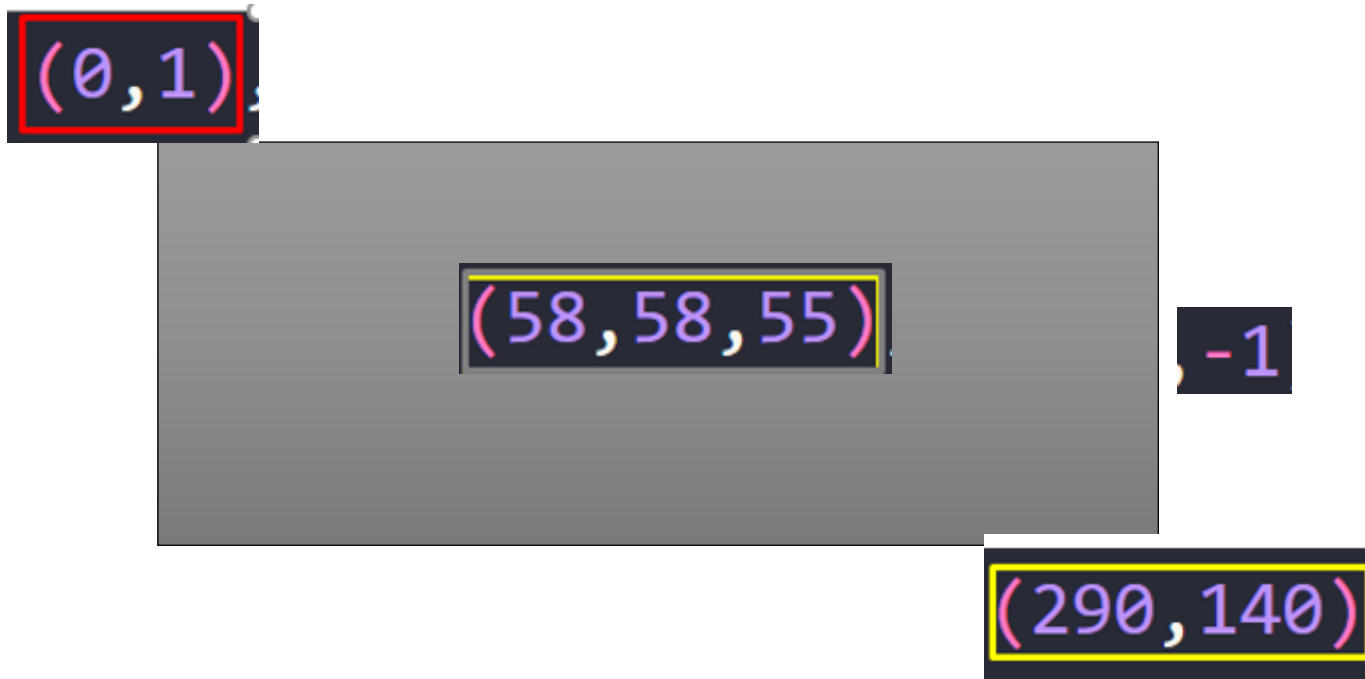
# Desenhando retângulo

- [https://docs.opencv.org/4.x/dc/da5/tutorial\\_py\\_drawing\\_functions.html](https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html)



image

# Desenhando retângulo



```
cv2.rectangle(frame, (0, 1), (290, 140), (58, 58, 55), -1)
```

# cv2.putText(1,2,3,4,5,6,7)

## ◆ putText()

```
void cv::putText ( InputOutputArray img,  
                  const String &    text,  
                  Point              org,  
                  int                fontFace,  
                  double              fontScale,  
                  Scalar            color,  
                  int                thickness = 1 ,  
                  int                lineType = LINE_8 ,  
                  bool               bottomLeftOrigin = false  
                )
```

## Python:

```
cv.putText( img, text, org, fontFace, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]] ) -> img
```

cv2.putText(1,2,3,4,5,6,7)

1. frame

2. texto

3. pontos

4. fonte

5. tamanho da fonte

6. cor

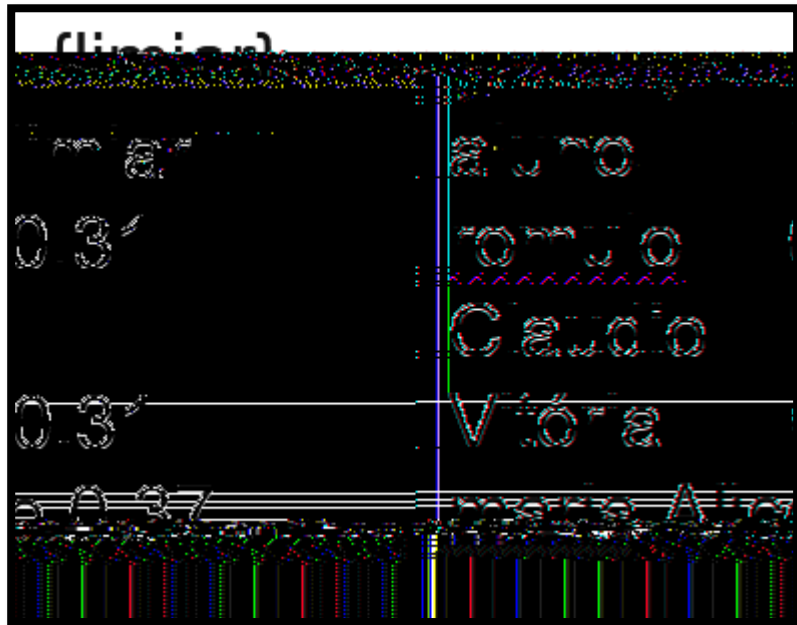
7. espessura do texto

```
cv2.putText(frame, f"EAR: {round(ear, 2)}", (1, 24),  
cv2.FONT_HERSHEY_DUPLEX,  
0.9, (255, 255, 255), 2)
```

**Resultado Final**

# Exercício

- Os alunos deverão entrar no dontpad e anotar o limiar.



Quem já descobriu a limiar?



Camera

— □ ×

FAR: 0.28



Prova





FAR: 0.28

# Passo 3 - Muito tempo com olhos fechados

if tempo&gt;=1.5:

cv2.rectangle(frame, (30, 400), (610, 452), (109, 233, 219), -1)



# Dever de Casa

- Considerando o algoritmo do tempo e a biblioteca time
  - Explique a linha do algoritmo (lógica)
  - Envie para o o email
    - [instrutor.romulo@gmail.com](mailto:instrutor.romulo@gmail.com)
    - “assunto” – algoritmo do tempo (manhã)

# Algoritmo do Tempo

**129 – Se  $ear < ear\_limiar$**

```
128 # Passo 1 - verificação ear < ear_limiar
129 if ear < ear_limiar:
130     t_inicial = time.time() if dormindo == 0 else t_inicial
131     dormindo = 1
132 if dormindo == 1 and ear >= ear_limiar:
133     dormindo = 0
134 t_final = time.time()
135 tempo = (t_final-t_inicial) if dormindo == 1 else 0.0
136
```

```
56 #Iniciaremos setando essa flag em zero.  
57 # 0 zero vai representar "false", isto é, quando a pessoa não está dormindo,  
58 # portanto, não está com o olho fechado.  
59 # Ela nos ajudará a controlar em que momentos calcularemos o tempo.  
60 dormindo = 0
```

flag



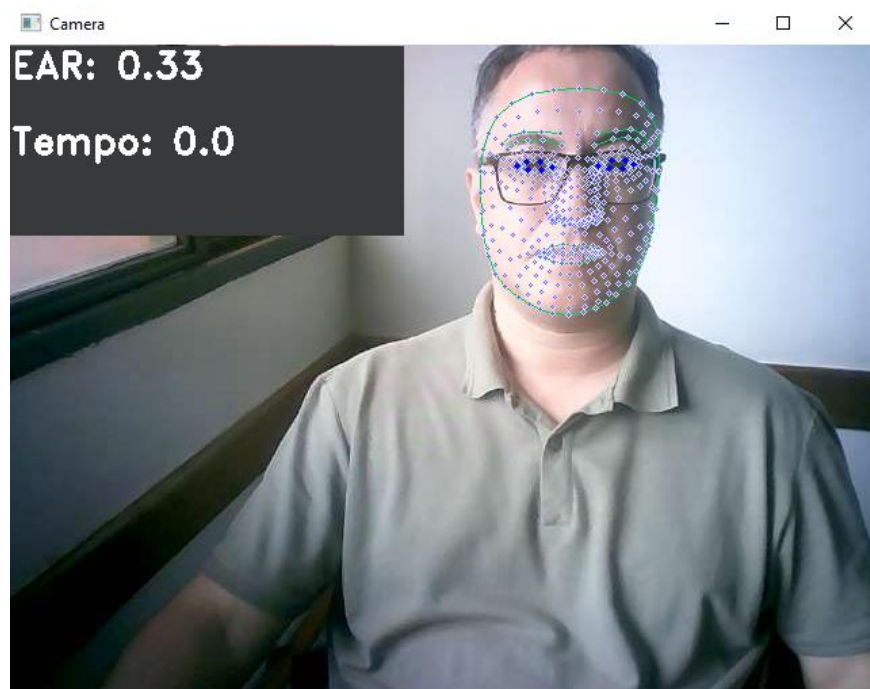


1

```
53 # criando a variável limiar
54 ear_limiar = 0.27
```

2

Olhos abertos

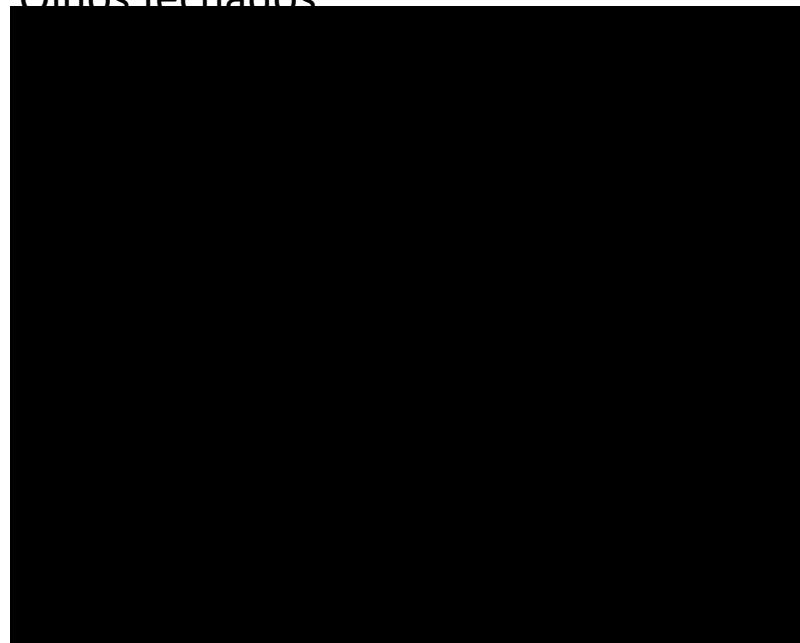


3



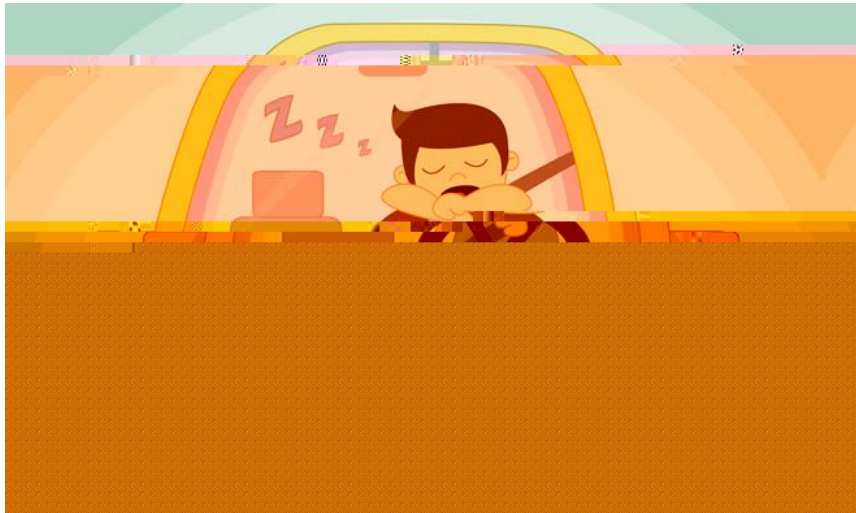
4

Olhos fechados



# Teste de Mesa

- Essa linha verifica se o valor de **ear** é menor do que um valor de limite (**ear\_limiar**).
  - Se isso for verdade, a pessoa estaria "dormindo" ou em um estado de descanso, por exemplo.



```
# Passo 1 - verificação ear < ear_limiar
```

```
if ear < ear_limiar: # dormindo
```

```
    t_inicial = time.time() if dormindo == 0 else t_inicial
```

```
    dormindo = 1
```

```
if dormindo == 1 and ear >= ear_limiar: # acordado
```

```
    dormindo = 0
```

```
t_final = time.time() # tempo
```

```
tempo = (t_final-t_inicial) if dormindo == 1 else 0.0 # calculo do tempo
```



### 3. Identificando a Boca

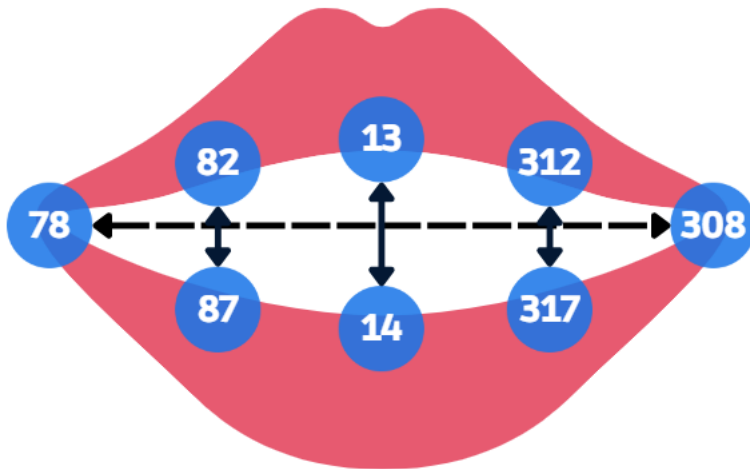
1. Verificar os pontos da boca
2. Cálculo do MAR
3. Interpretar o MAR
4. Verificação da boca

# 1. Verificando os pontos da boca

- Todo projeto é necessário uma análise
- Temos que verificar possíveis problemas e pensar em ajustes para eles
- Rodar o cálculo do EAR , testar e validar faz parte do projeto
- Se eu fecho meus olhos, o tempo passa a ser contado e uma mensagem de alerta aparece.
- Se eu fico muito tempo com olhos fechados eu posso também abrir a boca , dar uma gargalhada entre outras ações.

# 1. Verificando os pontos da boca

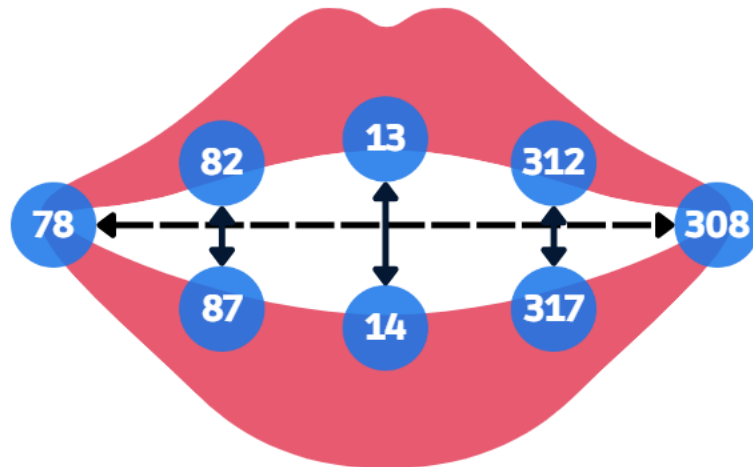
- Precisamos verificar a abertura da boca e seus pontos.
- Temos 8 pontos
- Usaremos o MAR (Mouth Aspect Ratio)
- `p_boca = [82,87,13,14,312,317,78,308]`
- A variável `p_boca` guardará a lista, igual aos valores dos pontos.



```
6  p_olho_esq = [385, 380, 387, 373, 362, 263]
7  p_olho_dir = [160, 144, 158, 153, 33, 133]
8  p_olhos = p_olho_esq + p_olho_dir
9
10 # variáveis da boca
11 p_boca = [82, 87, 13, 14, 312, 317, 78, 308]
```

# 1. Verificando os pontos da boca

- Precisamos verificar a abertura da boca e seus pontos.
- Temos 8 pontos
- Usaremos o MAR (Mouth Aspect Ratio)
- `p_boca = [82,87,13,14,312,317,78,308]`
- A variável `p_boca` guardará a lista, igual aos valores dos pontos.



# Normalização

- O MediaPipe é uma biblioteca desenvolvida pela Google que realiza detecção e rastreamento em tempo real de partes do corpo, mãos, rosto, e outros, usando aprendizado de máquina.
- Ele utiliza um sistema de coordenadas normalizadas, que vai de 0 a 1, em vez de usar pixels diretamente, para garantir que os pontos de referência (landmarks) que ele detecta possam ser facilmente adaptáveis a diferentes resoluções e proporções de imagem.

# \_normalized\_to\_pixel\_coordinates

- Ele recebe o x
- Ele recebe o y
- Ele recebe a largura
- Ele recebe o comprimento

```
if id_coord in p_boca:  
    coord_cv = mp_drawing._normalized_to_pixel_coordinates(coord_xyz.x,  
                                                            coord_xyz.y,  
                                                            largura,  
                                                            comprimento)  
    cv2.circle(frame, coord_cv, 2, (255, 0, 0), -1)
```

# Calculo do MAR

- Rishav Agarwal

```
def calculo_mar(face, p_boca):  
    try:  
        face = np.array([[coord.x, coord.y] for coord in face])  
        face_boca = face[p_boca, :]
```

```
    except:
```

```
        return 0
```



```
face_boca[3]) + np.linalg.norm(face_boca[4] - face_boca[5])) / (2 * (np.linalg.norm(face_boca[6] - face_boca[7])))
```

```
# Chamada do MAR e print  
mar = calculo_mar(faca, n_hora)
```

```
...  
6.9, (255, 255, 255), 7)
```

```
# Limiares
```

```
ear_limiar = 0.27
```

```
mar_limiar = 0.1
```

```
dormindo = 0
```



## 4. Detectar o sono

- Explorar os pontos de melhoria
- Contagem de piscadas
- Ajuste de alerta
- Conclusão (Deploy)

# Avaliação

1. Envio do algoritmo do tempo : 5 pontos  
(instrutor.romulo@gmail.com)
2. Mostrou “aberto” – ao abrir a boca – 4 pontos
3. Mostrou “fechado” – ao fechar a boca 4 pontos
4. Tocou alarme – ao abrir boca – 4 pontos
5. Inseriu mudanças de cores e alinhamentos – 4 pontos
6. **Mostrar o MAR – 4 pontos**