0. Install and Import Dependencies

In [ ]:
```
!pip install mediapipe opencv-python
```

In [2]:
```
import cv2
import mediapipe as mp
import numpy as np
mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose
```

In [3]:
```
# VIDEO FEED
cap = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
    cv2.imshow('Mediapipe Feed', frame)

    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

1. Make Detections

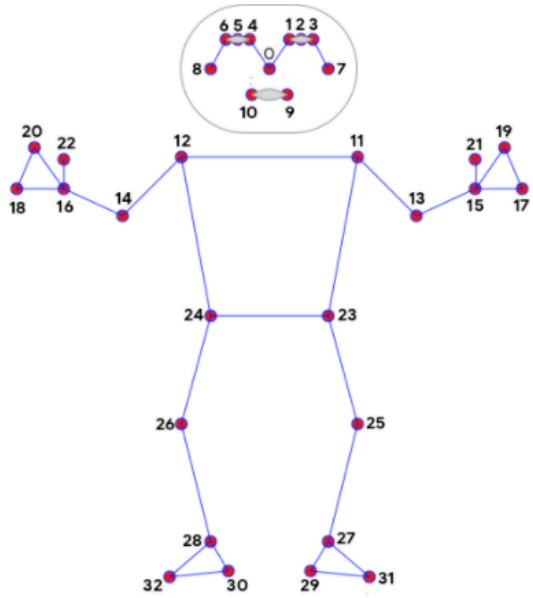
In [4]:
```
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as
    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CO
                                  mp_drawing.DrawingSpec(color=(245,117,66), thick
                                  mp_drawing.DrawingSpec(color=(245,66,230), thick
                                   )

        cv2.imshow('Mediapipe Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
```

```
    cap.release()
    cv2.destroyAllWindows()
```

In [5]:
```
mp_drawing.DrawingSpec??
```

```
Init signature:
mp_drawing.DrawingSpec(
    color: Tuple[int, int, int] = (224, 224, 224),
    thickness: int = 2,
    circle_radius: int = 2,
) -> None
Docstring:      DrawingSpec(color: Tuple[int, int, int] = (224, 224, 224), thickn
ess: int = 2, circle_radius: int = 2)
Source:
@dataclasses.dataclass
class DrawingSpec:
  # Color for drawing the annotation. Default to the white color.
  color: Tuple[int, int, int] = WHITE_COLOR
  # Thickness for drawing the annotation. Default to 2 pixels.
  thickness: int = 2
  # Circle radius. Default to 2 pixels.
  circle_radius: int = 2
File:           c:\users\ponar\anaconda3\lib\site-packages\mediapipe\python\solut
ions\drawing_utils.py
Type:           type
Subclasses:
```

2. Determining Joints



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

In [ ]:
```
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as
    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False
```

```
        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Extract landmarks
        try:
            landmarks = results.pose_landmarks.landmark
            print(landmarks)
        except:
            pass


        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CO
                                  mp_drawing.DrawingSpec(color=(245,117,66), thick
                                  mp_drawing.DrawingSpec(color=(245,66,230), thick
                                   )

        cv2.imshow('Mediapipe Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

In [7]:
```
len(landmarks)
```

Out[7]:
```
33
```

In [ ]:
```
for lndmrk in mp_pose.PoseLandmark:
    print(lndmrk)
```

In [9]:
```
landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].visibility
```

Out[9]:
```
0.9100753664970398
```

In [10]:
```
landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value]
```

Out[10]:
```
x: 1.0849497
y: 1.1214637
z: -0.6799235
visibility: 0.48577008
```

In [11]:
```
landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value]
```

Out[11]:
```
x: 0.95796424
y: 0.5965014
z: -0.79124755
visibility: 0.7861945
```

3. Calculate Angles

In [12]:
```
def calculate_angle(a,b,c):
    a = np.array(a) # First
    b = np.array(b) # Mid
    c = np.array(c) # End

    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) - np.arctan2(a[1]-b[1], a[0]-b[0]
    angle = np.abs(radians*180.0/np.pi)

    if angle >180.0:
        angle = 360-angle

    return angle
```

In [13]:
```
shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmarks[mp_p
elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmarks[mp_pose.Po
wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmarks[mp_pose.Po
```

In [14]:
```
shoulder, elbow, wrist
```

Out[14]:
```
([0.8539964556694031, 0.8562468886375427],
 [1.0849497318267822, 1.121463656425476],
 [0.9579642415046692, 0.5965014100074768])
```

In [15]:
```
calculate_angle(shoulder, elbow, wrist)
```

Out[15]:
```
27.451331966211058
```

In [16]:
```
tuple(np.multiply(elbow, [640, 480]).astype(int))
```

Out[16]:
```
(694, 538)
```

In [17]:
```
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as
    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Extract landmarks
        try:
            landmarks = results.pose_landmarks.landmark

            # Get coordinates
            shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,la
            elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmark
            wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmark

            # Calculate angle
```

```
            angle = calculate_angle(shoulder, elbow, wrist)

            # Visualize angle
            cv2.putText(image, str(angle),
                           tuple(np.multiply(elbow, [640, 480]).astype(int)),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv
                                )

        except:
            pass


        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CO
                                  mp_drawing.DrawingSpec(color=(245,117,66), thick
                                  mp_drawing.DrawingSpec(color=(245,66,230), thick
                                   )

        cv2.imshow('Mediapipe Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

4. Curl Counter

In [19]:
```
cap = cv2.VideoCapture(0)

# Curl counter variables
counter = 0
stage = None

## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as
    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Extract landmarks
        try:
            landmarks = results.pose_landmarks.landmark

            # Get coordinates
            shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,la
            elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmark
            wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmark
```

```
            # Calculate angle
            angle = calculate_angle(shoulder, elbow, wrist)

            # Visualize angle
            cv2.putText(image, str(angle),
                           tuple(np.multiply(elbow, [640, 480]).astype(int)),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv
                                )

            # Curl counter logic
            if angle > 160:
                stage = "down"
            if angle < 30 and stage =='down':
                stage="up"
                counter +=1
                print(counter)

        except:
            pass

        # Render curl counter
        # Setup status box
        cv2.rectangle(image, (0,0), (225,73), (245,117,16), -1)

        # Rep data
        cv2.putText(image, 'REPS', (15,12),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 1, cv2.LINE_AA)
        cv2.putText(image, str(counter),
                    (10,60),
                    cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)

        # Stage data
        cv2.putText(image, 'STAGE', (65,12),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 1, cv2.LINE_AA)
        cv2.putText(image, stage,
                    (60,60),
                    cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)


        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CO
                                  mp_drawing.DrawingSpec(color=(245,117,66), thick
                                  mp_drawing.DrawingSpec(color=(245,66,230), thick
                                   )

        cv2.imshow('Mediapipe Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

In [ ]: