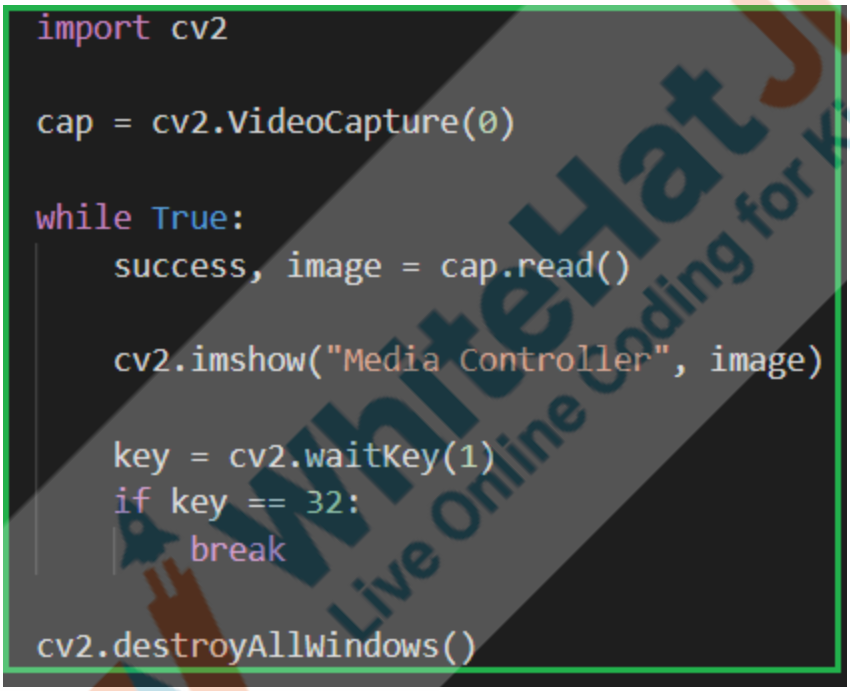
Topic	HAND GESTURE RECOGNITION	
Class Description	**The student will learn to detect human hands using the MediaPipe library and count the number of fingers live using a webcam.**	
Class	**PRO C108**	
Class time	**50 mins**	
Goal	• Detecting the hand points using the **MediaPipe** library.	
Resources Required	• Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone • Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up** **Teacher-Led Activity 1** **Student-Led Activity 1** **Wrap-Up**	**10 mins** **15 mins** **20 mins** **05 mins**
Credits	**MediaPipe Library** **https://google.github.io/mediapipe/**	
WARM-UP SESSION - 10 mins		
Teacher Starts Slideshow 		

Slide 1 to 3 Refer to speaker notes and follow the instructions on each slide.	
Teacher Action	**Student Action**
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! Yes I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session **Slide 4 to 15**	
Following are the session deliverables: • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
Teacher Ends Slideshow	
TEACHER-LED ACTIVITY - 15 mins	
Teacher Initiates Screen Share	
ACTIVITY • **Install the MediaPipe library** • **Detecting hands in a live video**	

Teacher Action	Student Action
Teacher Boilerplate Code(To Access Webcam using OpenCV): ***Note**: Download the boilerplate code, to start the video using a webcam(Teacher Activity 1). This is the same as we did in our previous classes. This code is given as boilerplate for the ease of the teacher.*	
 ``` import cv2 cap = cv2.VideoCapture(0) while True: success, image = cap.read() cv2.imshow("Media Controller", image) key = cv2.waitKey(1) if key == 32: break cv2.destroyAllWindows() ```	
In today's class, we are going to learn advanced concepts of Computer Vision. We will write a program which will: • Detect our hand. • Allow us to drag and drop an icon/file on the screen using our fingers. For this we will be using the **MediaPipe** library. **MediaPipe Library:**	

***Note**: MediaPipe library is developed by Google, and it is free to use.*

Detecting hand gestures using only OpenCV will require writing a lot of code. Even then this may not be very accurate.

MediaPipe has very powerful methods which can easily detect various human body poses, hand gestures and faces with very little code.

MediaPipe provides ready to use **Machine Learning** **Solutions** (that are predefined machine learning algorithms)for live and streaming media.

MediaPipe ML Solutions:

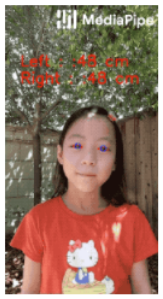
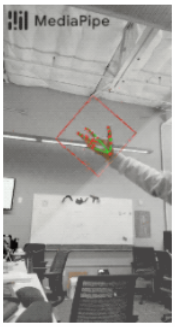
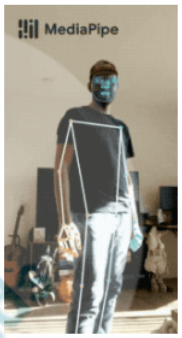
- Face Detection
- Face Mesh
- Iris
- **Hands**
- Pose
- Holistic
- Hair Segmentation
- Object Detection
- Box Tracking
- Instant Motion Tracking
- Objectron
- KNIFT

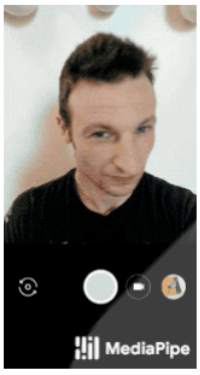
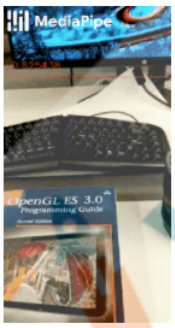
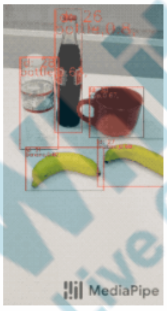
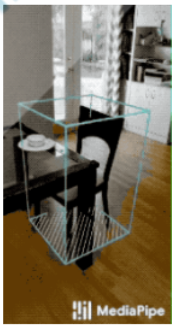
***Note**: We will not deep dive into the working of each MediaPipe ML solution.*

In today's class we will learn to use **Hands solutions** of **MediaPipe** Library.

***Note**: Teachers can open this link to show available solutions in the MediaPipe Library.*

ML solutions in MediaPipe

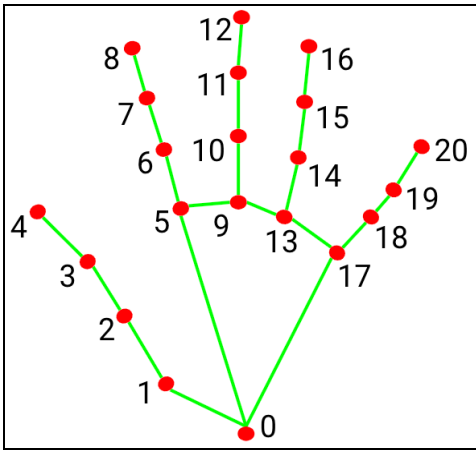
Face Detection	Face Mesh	Iris	Hands	Pose	Holistic
					

Hair Segmentation	Object Detection	Box Tracking	Instant Motion Tracking	Objectron	KNIFT
					

Now let's understand how we can use the MediaPipe Library **Hands Solution** to detect a hand.

The **MediaPipe Hands Solution** tracks 21 points/landmarks on our hand. Each point has a unique ID starting from 0. With this ID we can get the x and y position of this landmark/point on our hand.

***Note**: Open Visual Aids to show hand landmark points to the student.*



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

To start using the **MediaPipe** library, we will have to install the library. To install the MediaPipe library, we are going to run the below command:

***Note**: If there is more than one Python version (Python 2 or Python 3) installed in your system, specify the pip or pip3 respectively.*

```
>> pip install mediapipe
```

***IMP Note**: If the **installation fails**, this could happen as the library version might NOT be compatible with the Python version installed in your system.*

- *Ensure that the 64-bit Python version is installed OR*
- *Try re-installing the Python version between **3.6** to **3.8**. This is valid at the time of creation of this document. Python and other libraries/modules keep getting updated, work around with versions of both to make those compatible with each other.*

Once the installation is complete, we need to import the **MediaPipe** library in the code.

In Python, we can use **alias(nickname)** for libraries with longer names using "**as"** keywords.

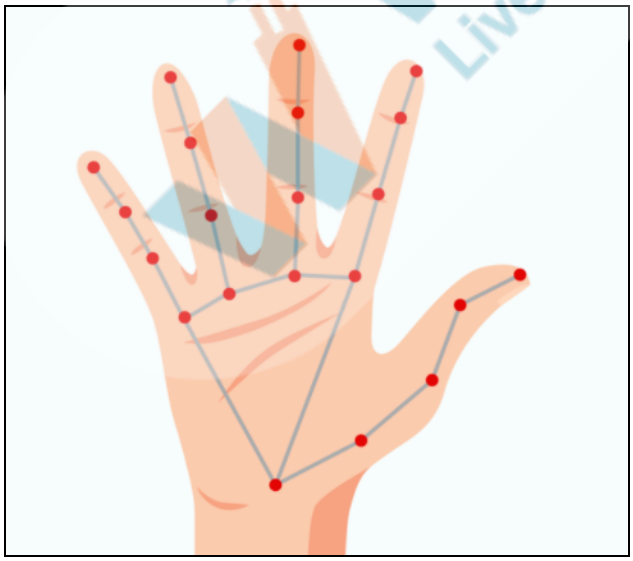
import mediapipe **as** mp

```
import cv2
import mediapipe as mp
```

Now we are going to use the two modules which are part of **mediapie.solutions**:

- **hands** module that defines the algorithm first to detect the palm and later to detect the 21 landmark points over the palm.
- **drawing_utils** module to draw the connecting lines between the 21 landmark points.

***Note**: If we do not draw the connection(gray lines in the image below), then only the red points(in the image below) will be visible on the hands. We will learn about this as we move forward with the class.*



To load the **hands** and the **darwing_utils** module, we will write the following lines of code:

- Take two variables, **mp_hands** and **mp_drawing** a
- Assigning the **mp.solutions.hands** **mp.solutions.darwing_utils**.

```
cap = cv2.VideoCapture(0)

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils


while True:
    success, image = cap.read()
```

Next step is to start using these algorithms on our live video frames being captured from the webcam.

To perform this, we are going to use the **Hands()** method from the hands algorithm we just loaded.

The **Hands()** method will take 2 parameters:

- **min_detection_confidence**
- **min_tracking_confidence**

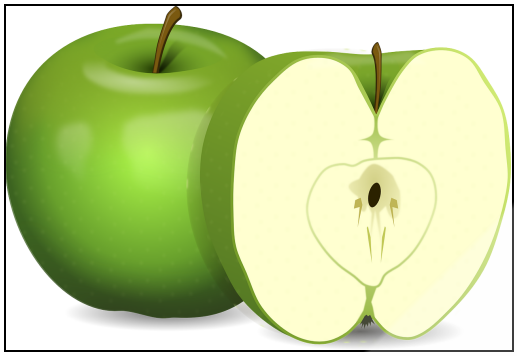
Before we can move forward, let's understand a few terminologies to understand these parameters of the **Hands()** method.

Detection and Prediction:

When any machine learning algorithm **detects** some object in an image or a video, it gives us the **prediction value** about the presence of that object in a particular image/video. **The prediction values represent how**

confident the algorithm is in predicting the result accurately.

For example, if we wrote an algorithm to **detect either an apple or a pear fruit in an image**.



OR

Then if we run the algorithm over a new image of an apple and the algorithm gives us a result by saying that there is a 90%(0.9) chance that this is an apple and 10%(0.1) chance it is a pear.

The **prediction confidence values are represented as decimal point values between the range of 0 to 1**. Hence the 70%,80% or 90% will be represented as 0.7(70/100), 0.8(80/100) and 0.9(90/100).

Now that we understand what the prediction confidence means, we will set the parameters of the **Hands()** method, that sets the value of prediction confidence to consider the **hands** detection results successful.

The **Hands()** method in **hands** algorithm we will set:

- **min_detection_confidence :** First confidence value to detect how sure our algorithm is that it has detected a hand.

  We will use a value of 0.8 which is 80% which means we will only consider the results good when the confidence is 80% or more so while showing the result this hands algorithm will take care of it.

- **min_tracking_confidence :** Second confidence value is for tracking, so we are not only detecting the hands, but we are also tracking them, so we also need to specify the minimum tracking of the confidence which we will keep as 0.5 or 50%.

```
cap = cv2.VideoCapture(0)

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils

hands = mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5)

while True:
    success, image = cap.read()
```

We have defined the parameters of the **Hands()** method of the **hands** algorithm on our video frames, now to detect the hands we only need to call one method:

- **hands.process(image)**

Take a variable result to store the process() method result:

- **result = hands.process(image)**

.

```
hands = mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5)

while True:
    success, image = cap.read()

    # Detect the Hands Landmarks
    results = hands.process(image)

    cv2.imshow("Media Controller", image)
```

Run the code: >> ***python count_fingers.py***

OUTPUT:

We may have detected our hands in the video frame image, but we still won't be able to see them visually in the output whether the hands have been detected or not!

In order to see the detected hands and landmarks on the hand, we need to draw these points and the connection

<table>
<tr><td>between them.

But before we move forward to drawing our hands, have you noticed anything different when we run our code to see the output?

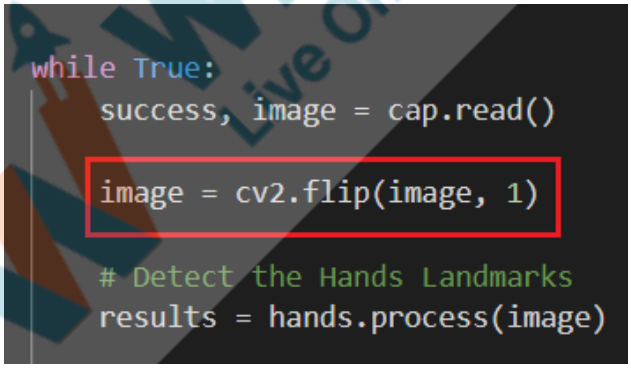
Our video frame image is flipped, i.e. when we lift the left hand it shows on the right side of the output window and vice versa. Although, it does not matter for detection.

But, it feels very weird, right?

So let’s correct it by using the OpenCV flip() method, this method will take two parameters:

● image: image/video frame on which we are working
● direction: in which we want to flip the image
○ 0: To flip vertically
○ 1: To flip horizontally

In our case we want to flip in the horizontal direction, so we will use direction parameter value as 1.</td><td>ESR: Varied.

ESR: Yes</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td>Now let’s draw the connections on the hand landmark points to show the hand detection. For this we will define a function called drawHandLandmarks() function with two parameter:

● image: the image/video frame we are working with</td><td></td></tr>
</table>

- **hand_landmarks**:
  - We will pass the **multi_hand_landmarks** property value returned in the **results** values. The **multi_hand_landmarks** property returns the **x**, **y** and **z** position of the landmark points

    Here, x and y are the normal **x, y** position coordinates of the landmark. Whereas, **z** is the distance of the landmark from the webcam.

```
# Define a function to draw connections
def drawHandLandmarks(image, hand_landmarks):

    # Darw connections between landmark points


while True:
    success, image = cap.read()

    image = cv2.flip(image, 1)

    # Detect the Hands Landmarks
    results = hands.process(image)

    # Get landmark position from the processed result
    hand_landmarks = results.multi_hand_landmarks

    # Draw Landmarks
    drawHandLandmarks(image, hand_landmarks)
```

Define **drawHandLandmarks(image, hand_landmarks)** function:

- First we check whether we are getting the **hand_landmarks** value with the help of **if** condition, which will only be returned when the hands are visible in front of the camera, else the results will be empty.

- We will **loop** through these points to draw connections using the **mp_drawing.draw_landmark()** method with following parameters:

    - **image**: image/video frame we are working with.
    - **landmark positions list**: landmark position values.
    - **HAND_CONNECTIONS** MediaPipe **hands** property.

Now you can run the code to see the output.

```
# Define a function to draw connections
def drawHandLandmarks(image, hand_landmarks):

    # Darw connections between landmark points
    if hand_landmarks:

      for landmarks in hand_landmarks:

        mp_drawing.draw_landmarks(image, landmarks, mp_hands.HAND_CONNECTIONS)
```

OUTPUT:

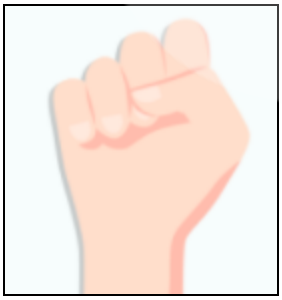
This looks cool, right? So here we are able to detect and draw the landmarks on our hands. Next step is to calculate the number of fingers in the hand and based on that we can trigger different events. Do you want to write the code to calculate the number of fingers? Great! Please share your screen with me.	**ESR:** Yes **ESR:** Yes

Teacher Stops Screen Share

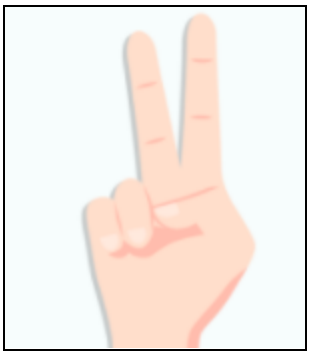
Teacher Starts Slideshow

Slide 16 to 17

Refer to speaker notes and follow the instructions on each slide.

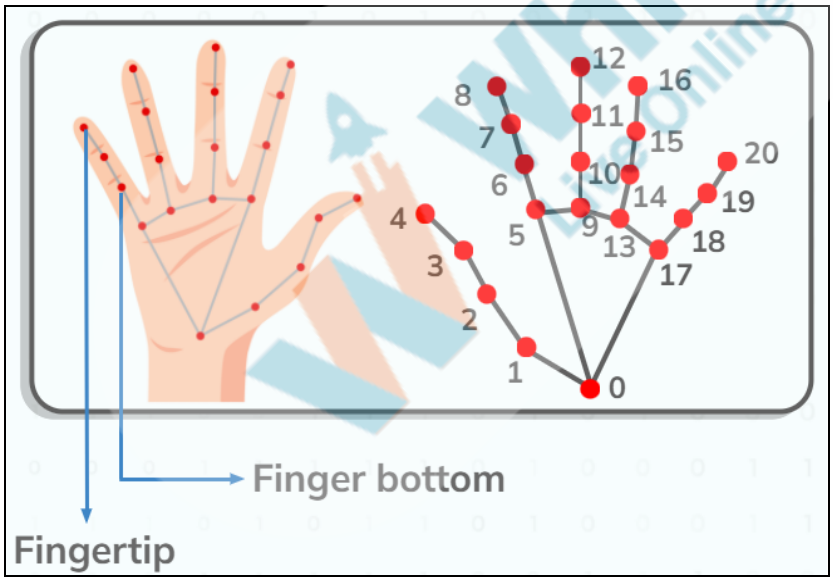
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	
Teacher Ends Slideshow 	
STUDENT-LED ACTIVITY - 20 mins	
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start Screen Share.** • **The teacher gets into Fullscreen.**	
ACTIVITY • **Write a function to count the number of fingers in the camera video.**	
Teacher Action	**Student Action**
Guide the student to download the boilerplate code from Student Activity 1 ***Note**: Before starting to code, make sure that the student has installed the MediaPipe library.* Before starting the code, let's briefly understand the logic to count fingers. When you are making a fist, all of your fingers are curled, and we say that no finger is up. 	*The student downloads the Student Activity 1 code and opens it in the VS Code.*

As you straighten the fingers, we will be able to see the fingers and can count them.



When you make a fist and curl the fingers, what happens to the tips of the fingers and the bottom of your fingers?

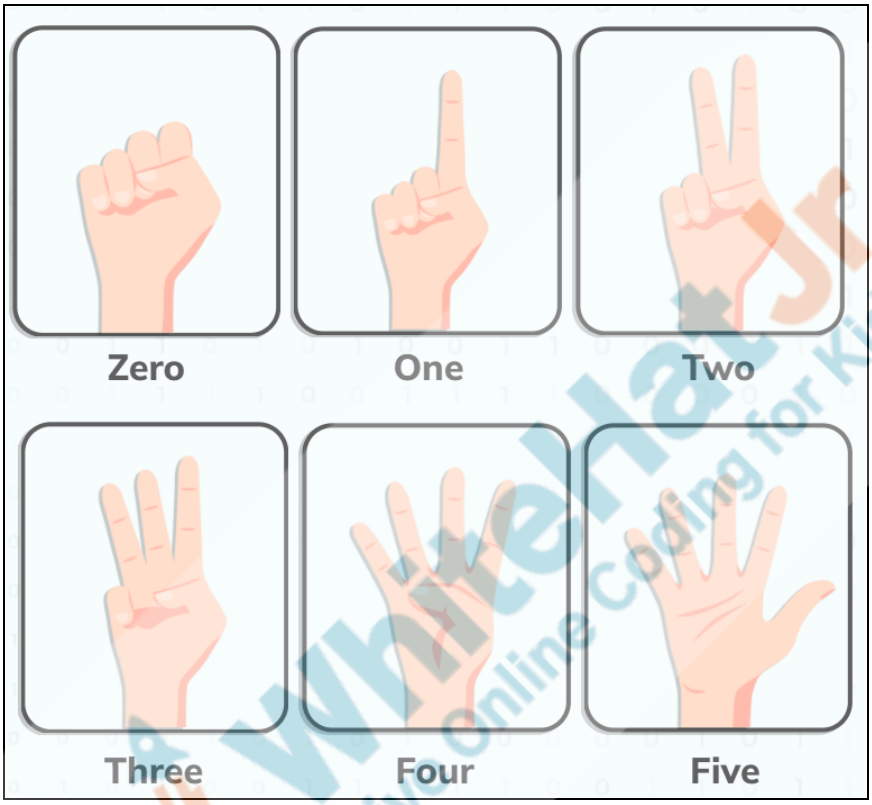
We can say that the position of the fingertips comes closer to the position of the bottom of the fingers.



So to detect the number of fingers, we need to check the position of the fingertips with the position of the finger bottom.

ESR: Varied.

***Note**: We will consider the following positions of hand fingers to count the number of open fingers. Teachers can open Visual Aids to explain the count of fingers to the students.*



So now we have the basic idea of counting fingers after hand detection, we will define a function called **countFingers()**.

***Note**: Explain the following boilerplate code to the student.*

1. Create a list named **tipIds = [4,8,12,16,20]**. These are the IDs of all the fingers and the thumb. Number 4 is the id of the thumb, 8 is the index finger and so on.

2. Define a function named, **countFingers()** outside the **while** loop. This function will need to three

parameters:

a. **image** we are working with
b. **hand_landmarks**: the landmarks obtains from the **process()** method
c. **handNo=0**.This is called a default argument which means we specify the value of this parameter at the time of function declaration so that it is not mandatory to pass this value while calling the function and **handNo = 0** means we are only detecting 1 hand. In the output, you will see the points on both the hands, but it will only consider one hand to get the finger positions.

```
tipIds = [4, 8, 12, 16, 20]

# Define a function to count fingers

def countFingers(image, hand_landmarks, handNo=0):

    ########################################################

        # ADD CODE HERE

    ########################################################
```

3. Check if we received the **hand_landmarks** values or not using the **if** condition.

   if **hand_landmarks:**

4. Create a variable as **landmarks** to store all the landmarks points of the first hand visible in the webcam.This can be by getting landmarks at index

0, stored in the parameter, **handNo**.

5. Print the **landmarks** variable values
   landmarks= hand_landmarks[handNo].landmark

   ***Note**: Indentation is very important in Python, make sure the code is properly indented.*

```
# Define a function to count fingers

def countFingers(image, hand_landmarks, handNo=0):

    if hand_landmarks:
        # Get all Landmarks of the FIRST Hand VISIBLE
        landmarks = hand_landmarks[handNo].landmark

        print(landmarks)
```

Now call the function, **countFingers()** and run the code.

6. Call **countFingers()** inside the **while** loop.
7. Run the code to see the output in the terminal.

```
    # Draw Landmarks
    drawHandLanmarks(image, hand_landmarks)

    # Get Hand Fingers Position
    countFingers(image, hand_landmarks)

    cv2.imshow("Media Controller", image)
```

OUTPUT(In Terminal):

***Note**: The below output image is shortened. The actual output consists of 21 landmark points each with { x:---, y:---, z:---} values combined together in a list.*

```
[x: 0.35317790508270264
y: 0.8584342002868652
z: 5.829742235619051e-07
, x: 0.429390877485275 27
y: 0.8280720710754395
z: -0.039083223789930344
, x: 0.49740302562713623
y: 0.7563502192497253
z: -0.05134163796901703
, x: 0.5484085083007812
y: 0.6893746256828308
z: -0.060729142278432846
```

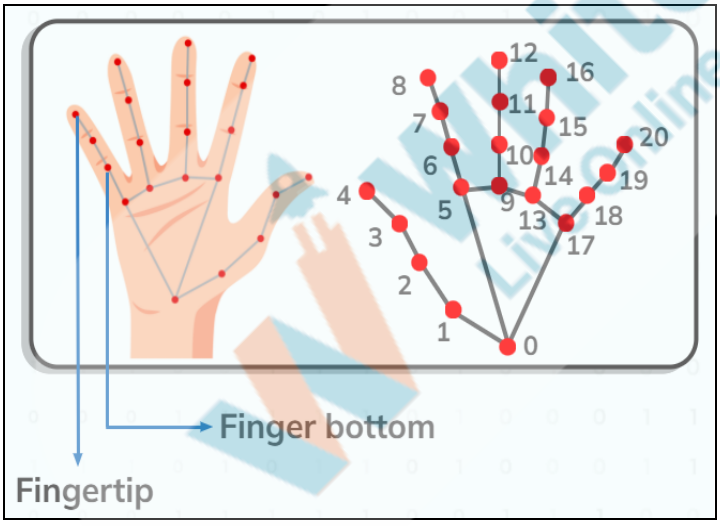
As you can see the **each landmarks values are of type dictionary combined together** in the simple list format:

```
[
    x: —
   y: —
   z: —

  , x: —
  y: —
  z: —
 , ...
 …

 , x: —
 y: —
 z: —
]
```

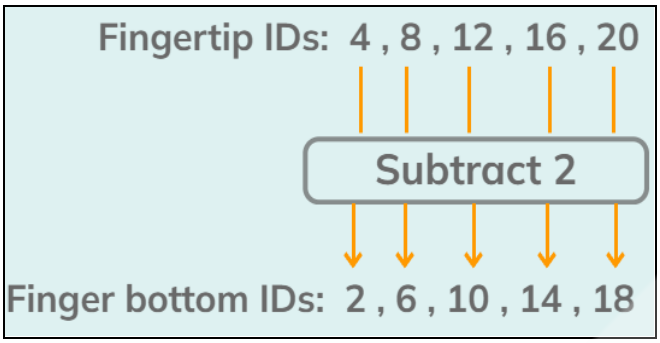
Now, to count the fingers/thumb, we need only x and y positions of each landmark point from the above list.

Can you tell me why?

Counting the fingers/thumb, requires only x and y positions

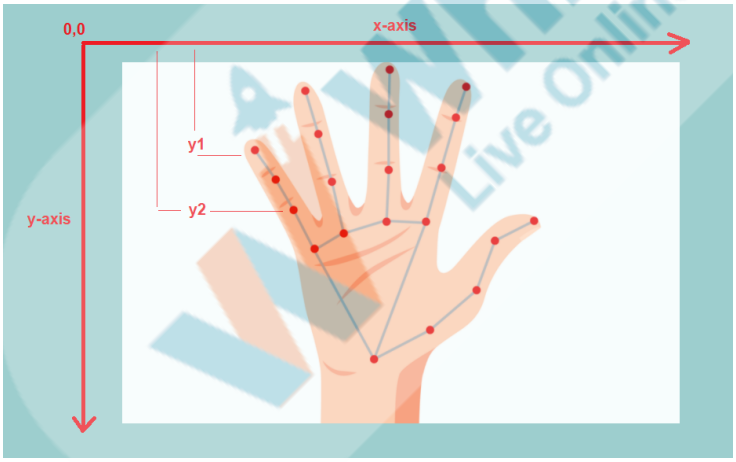
of each landmark point list because the z value represents the distance from the webcam, which is irrelevant to count fingers. Before we continue with code, let's understand the working flow to count the fingers.	**ESR:** Varied.
Countings First Hand FINGERS: Counting all fingers/thumbs will require separate logics for ● Fingers ● Thumb ***Note**: To understand the working of counting **THUMB** implementation refer to **ADDITIONAL ACTIVITY** Section.* **Working Logic to Count 4 Fingers Of Hands**:  1. Get **Fingertip IDs**: `Fingertip IDs: 4 , 8 , 12 , 16 , 20` 2. Get **Finger bottom IDs** by subtracting 2 from form	

Fingertips IDs



3. **Compare** the **y position** of **Fingertips**(y1 in the below image) and **Finger bottom**(y2 in the below image).

   When the fingers are open y position Fingertips will be less than y position of Finger bottom as the y axis increases from top to bottom that is **y1 < y2.**



Now that we have understood the working of how to count fingers, let's implement this using the program:

1. Create an empty list named **fingers**.

2. Loop through **tipIds** using a iteration variable called **lm_index**

    a. Get the y position of the Fingertip landmark point with index value as **lm_index**

    finger_tip_y = landmarks[lm_index].y

    b. Get the y position of the Finger bottom landmark point with index value as **lm_index - 2**

    finger_bottom_y = landmarks[lm_index -2].y

    c. Check if it is not Thumb(Landmark ID of THUMB is 4)

    if lm_index != 4

    d. Compare y position of Fingertip and Finger bottom using if condition:

    if **finger_tip_y < finger_bottom_y**,
    then append the number 1 to the **fingers** listif the finger is open else append 0.

3. Count the number of 1 present in the **fingers** array.
4. Display text on the output window using **putText()** method.

***Note 1**: Help the student recollect this method functionality covered in the previous classes)*

***Note 2**: Indentation is very important in Python, make sure the code is properly indented.*

```
    # Count Fingers
    fingers = []

    for lm_index in tipIds:
            # Get Finger Tip and Bottom y Position Value
            finger_tip_y = landmarks[lm_index].y
            finger_bottom_y = landmarks[lm_index - 2].y

            # Check if ANY FINGER is OPEN or CLOSED
            if lm_index !=4:
                if finger_tip_y < finger_bottom_y:
                    fingers.append(1)
                    print("FINGER with id ",lm_index," is Open")

                if finger_tip_y > finger_bottom_y:
                    fingers.append(0)
                    print("FINGER with id ",lm_index," is Closed")

    # print(fingers)
    totalFingers = fingers.count(1)

    # Display Text
    text = f'Fingers: {totalFingers}'

    cv2.putText(image, text, (50,50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)
```

OUTPUT:

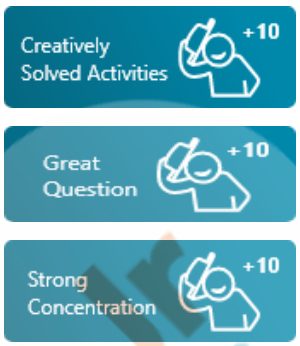
***Note**: Since we are **NOT** counting **THUMB** separately, the count of fingers will not exceed 4 even if all the fingers and thumb are opened together. To achieve this refer to the implementation given in the **ADDITIONAL ACTIVITY** Section.*

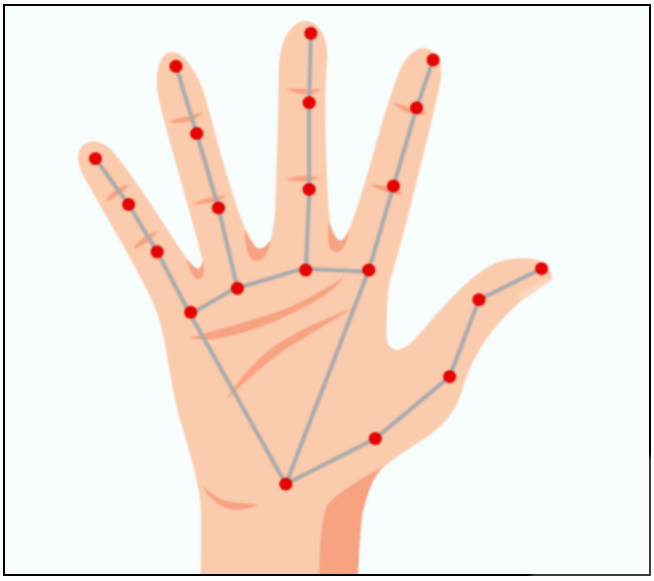
Now we are able to detect the finger count and all the landmarks on the fingers.

In the next class we are going to use these concepts to create a hand gesture controlled media player, where cano drag and drop icons using your virtual hands gestures. How cool is that?

ESR: Varied

Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 05 mins	
Teacher Starts Slideshow **Slide 18 to 23** 	
Activity Details: **Following are the WRAP-UP session deliverables:** • Appreciate the student. • Revise the current class activities. • Discuss the quizzes.	
WRAP-UP QUIZ Click on In-Class Quiz	
Continue WRAP-UP Session **Slide 24 to 29** 	
Activity Details: **Following are the session deliverables:** • Explain the facts and trivia • Next class challenge • Project for the day • Additional Activity (Optional)	
FEEDBACK • **Appreciate the student for his/her efforts in the class.**	
Teacher Action	**Student Action**
You get hats-off for your excellent work!	*Make sure you have given*

	at least 2 Hats Off during the class for: 
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks ✖ End Class	
ADDITIONAL ACTIVITY	
Countings First Hand THUMB on RIGHT SIDE: The Thumb can open towards right side or towards left depending upon the: ● Hand used(left or right hand) ● Side of Hand(Front or Back side of the hand) ***Note**: We will only see how we can include the thumb in the total count of fingers, when the thumb is open towards the right side(as shown in the image below). The student can try to write logic for the thumb opening towards the left side on his own after the class.*	



The process of counting Thumb along with fingers would be almost similar with slight logical difference.

Can you think of how?

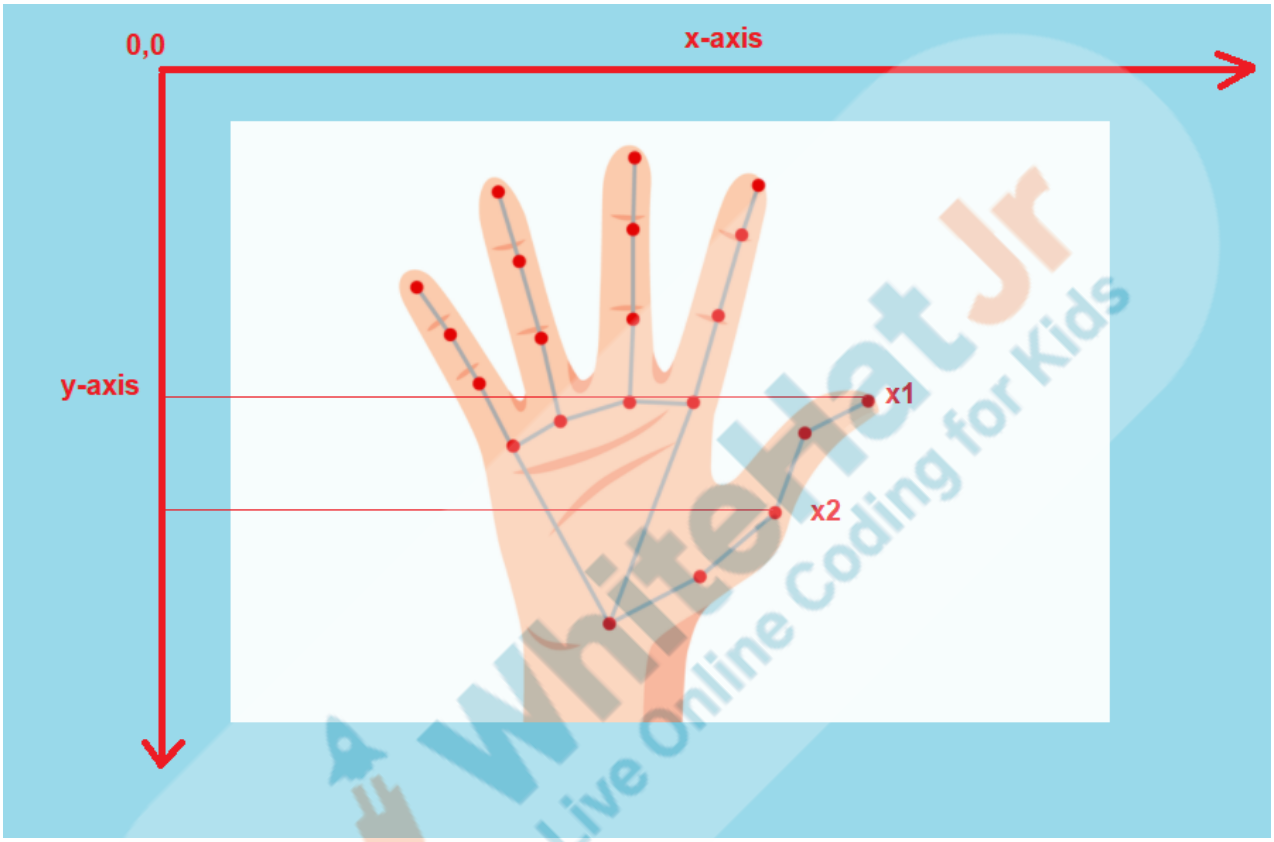
We will need the value of **Fingertip IDs** and **Finger bottom IDs**, as we did to count only fingers.

The only difference after getting the we need to compare x position as the thumb opens and closes in horizontal direction!

Compare the **x position** of **Fingertips**(x1 in the below image) and **Finger bottom**(x2 in the below image).

ESR: Varied.

When the thumb is open towards the **RIGHT SIDE (as shown in the image below)**, **x position Fingertips will be less than x position of Finger bottom** as the x axis increases from left to right that is **x1 > x2.**



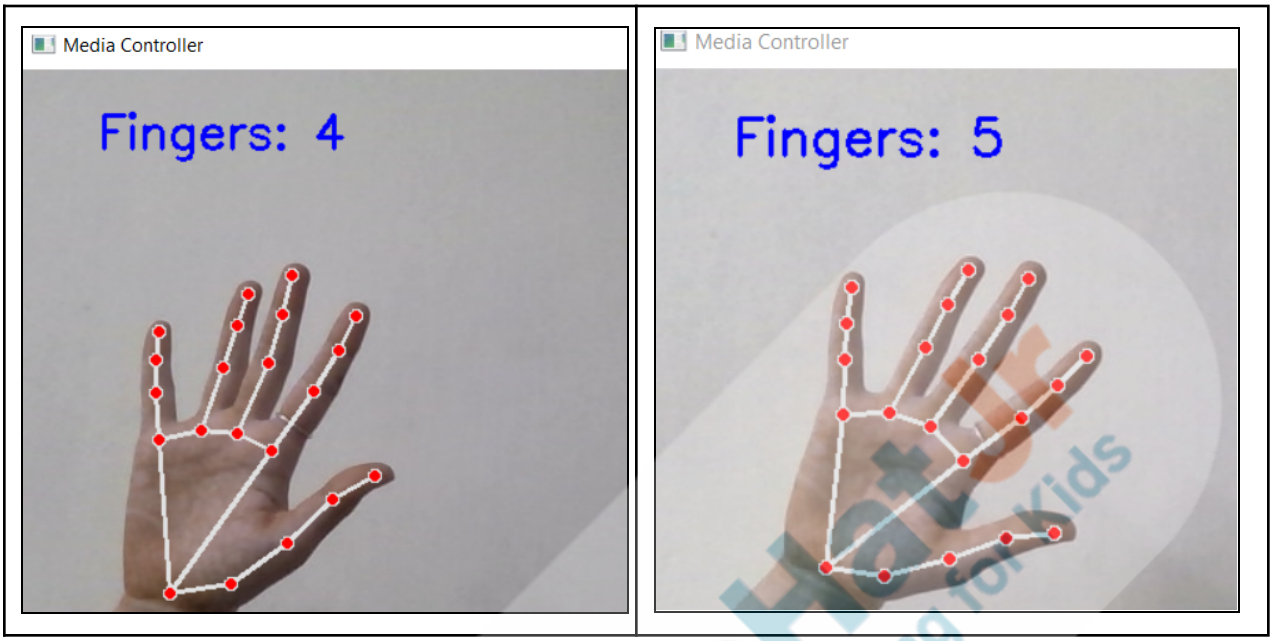
Update the code to add functionality for:

1. Getting x position of the tip and bottom of the thumb
2. Compare the x position of the tip and bottom of the thumb
3. Append 1 to the fingers list if x position of the tip of the thumb > bottom of the thumb(Note: This reversed in case of as the x-axis values increase from left to right)

***Note**: Refer the **count_fingers_with_thumb.py** in the Teacher Activity 2 for the below code.*

```
for lm_index in tipIds:
        # Get Finger Tip and Bottom y Position Value
        finger_tip_y = landmarks[lm_index].y
        finger_bottom_y = landmarks[lm_index - 2].y

        # Get Thumb Tip and Bottom y Position Value
        thumb_tip_x = landmarks[lm_index].x
        thumb_bottom_x = landmarks[lm_index - 2].x

        # Check if ANY FINGER is OPEN or CLOSED
        if lm_index !=4:
            if finger_tip_y < finger_bottom_y:
                fingers.append(1)
                print("FINGER with id ",lm_index," is Open")

            if finger_tip_y > finger_bottom_y:
                fingers.append(0)
                print("FINGER with id ",lm_index," is Closed")
        else:
            if thumb_tip_x > thumb_bottom_x:
                fingers.append(1)
                print("THUMB is Open")

            if thumb_tip_x < thumb_bottom_x:
                fingers.append(0)
                print("THUMB is Closed")
```

OUTPUT: **Before Updating the Code to Count Thumb Separately**	**OUTPUT:** **After Updating the Code to Count Thumb Separately**



We can do a lot using hand detection. Keep Exploring!

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C108-Teacher-Bolierplate
Teacher Activity 2	Reference Code	https://github.com/procodingclass/PRO-C108-Reference-Code
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C108-Student-Boilerplate
Teacher Reference 1	MediaPipe Library	https://google.github.io/mediapipe/
Teacher Reference 2	MediaPipe ML Solutions	https://google.github.io/mediapipe/solutions/solutions.html
Teacher Reference 3	OpenCV Text	https://docs.opencv.org/4.x/d6/d6e/group

		__imgproc__draw.html#ga5126f47f883d730f633d74f07456c576
Teacher Reference 4	Project Document	https://s3-whjr-curriculum-uploads.whjr.online/5da2095f-a30a-4a6f-8103-fc0123e1137d.pdf
Teacher Reference 5	Project Solution	https://github.com/pro-whitehatjr/PROJECT-SOLUTION-C108
Teacher Reference 6	Visual Aid Link	https://s3-whjr-curriculum-uploads.whjr.online/dca8167d-2f8f-4bba-8ebe-bf4816a8b6f1.html
Teacher Reference 7	In Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/7a1db9b8-4a15-49ae-9117-170632282bb7.pdf