Topic	VIRTUAL KEYBOARD & MOUSE	
Class Description	**The student will control the system mouse virtually using hand gestures, line index finger and pinch to interact with system applications by hand gestures virtually using a Python library.**	
Class	**PRO C109**	
Class time	**50 mins**	
Goal	• Learn to use hand gestures to control keyboard to play and pause vrtually using hand gestures • Learn to drag and drop files on the desktop using a virtual pinch gesture.	
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** **10 mins** **25 mins** **05 mins**
Credits	**YouTube Channel: Global Cycling Network** **Video:** **https://www.youtube.com/watch?v=SdbVz-HZ8H8**	

<table>
<tr><th colspan="2">WARM-UP SESSION - 10 mins</th></tr>
<tr><td colspan="2">Teacher Starts Slideshow
Slide 1 to 3
Refer to speaker notes and follow the instructions on each slide.</td></tr>
<tr><th>Teacher Action</th><th>Student Action</th></tr>
<tr><td>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.</td><td>ESR: Hi, thanks!
Yes I am excited about it!

Click on the slide show tab and present the slides</td></tr>
<tr><td colspan="2">WARM-UP QUIZ
Click on In-Class Quiz</td></tr>
<tr><td colspan="2">Continue WARM-UP Session
Slide 4 to 10</td></tr>
<tr><td colspan="2">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.</td></tr>
<tr><td colspan="2">Teacher Ends Slideshow</td></tr>
<tr><th colspan="2">TEACHER-LED ACTIVITY - 10 mins</th></tr>
<tr><td colspan="2">Teacher Initiates Screen Share</td></tr>
</table>

ACTIVITY

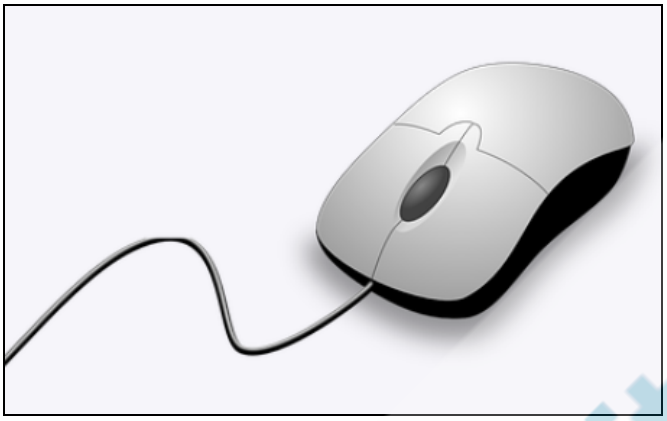
- **Play and Pause a video using virtual controlling spacebar key control using fist gesture**
- **Move a video forward and backward by virtually controlling arrow keys using index finger**

Teacher Action	Student Action
In the last class, we learned how to use the **MediaPipe** library to detect various landmarks on our hands and count the number of fingers. Using the same concepts, today we are going to control some of the applications of our computer using HAND GESTURES VIRTUALLY. **Controlling applications using hand gestures virtually** means, we will not use actual keyboard or mouse to control apps on our computer, instead we will control the keyword using hand gestures like pressing arrow keys by moving index finger or mouse click using a pinch! In today's class we will learn to interact with system applications virtually using hand gestures, like videos or system files/folders. **Control Keyboard Virtually using Hand Gestures**:  We will start by controlling a video, with following features:	

- **Play & Pause a video** using a **fist**(all fingers curled together) gesture.

- **Move a video Forward & Backward** using an **index finger** movement left/right.

Control Mouse Virtually using Hand Gestures:



We will also learn:

- **Drag and Drop system files** using a **pinch gesture**.

But before we start coding, we are going to install two libraries called **pynput** and **PyAutoGUI** using the **pip** command:

pip install <module_name>

```
pip install pynput
pip install pyautogui
```

pynput: Used to observe any input received from the input devices like mouse and keyboards.

PyAutoGUI: Helps to control mouse and keyboards through Python programs.

Once the libraries are installed let's work on the program to control a video using hand gestures. For this we will control the keyboard: • To play and pause, press the spacebar key using a hand fist. • To move forward/backward press arrow keys using the index finger. We will continue after what we did in previous, detecting hand landmarks and counting fingers. *Download the boilerplate code using Teacher Activity 1 code.* ***Note**: The boilerplate is the same as previous class with few additions. Explain the following addition to the previous class code to the student.* Since we want to control the keyboard, we will: • Import the **Key** and **Controller** class from the **pynput.keyboard** module. • Initialize the **Controller** object using a variable called **keyboard**.	

```
import cv2
import mediapipe as mp
from pynput.keyboard import Key, Controller

keyboard = Controller()

cap = cv2.VideoCapture(0)

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

***Note**: Since we are going to control the video using **Spacebar** key(keycode **32**), the control to quit the window has been updated using **Esc** key(keycode **27**).*

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

    # Quit the window on pressing Sapcebar key
    key = cv2.waitKey(1)
    if key == 27:
        break

cv2.destroyAllWindows()
```

Now, we will add the logic to play and pause the video.

PLAY/PAUSE Video By Controlling SPACEBAR Key Using Hand FIST:

The concept is very simple. Based on the number of fingers, we will trigger a keyboard key through our Python script.

For example, if you make a fist then there will be zero fingers, in that case we will want to press the space key.

Video States:

The video will either be playing or it will be paused.

To capture these two states of the video, "**Play**" or "**Pause**", we will use a variable called **state**.

- Create a variable **state** with initial values as **None.**

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

state = None

# 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
```

Now we will update **countFingers()** function to add conditions to control the **state** variable.

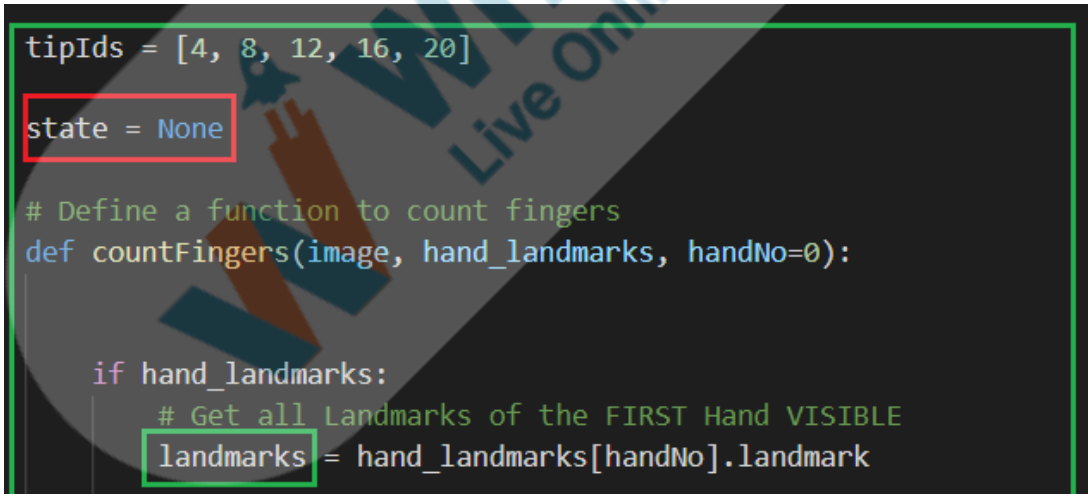
Since we want to use the **state** variable inside the function, we need to make sure the value of the **state** variable is updated every time we open/close our fingers.

Before we can continue, let's scope out variables in Python.

Do you remember what are global and local variables in JavaScript?

ESR: Yes.

- Local variables are defined within the functions and cannot

Amazing! **Global Vs. Local Variables In Python:** Similar to JavaScript, in Python: Variables which are created outside any of the functions are called **global variables**. Variables which are created inside the function are **local variables**. For example, the variable **state**, is global variable and the variable **landmarks**, is local variable(limited to use only inside **countFingers()** function).	be used outside the functions. • Global variables can be used throughout the program within any function.
 	
In Python, if we want to use any variable defined inside a function to be used globally, we can use the **global** keyword before the variable name!	

Since we want to use the **state** variable inside the**countFingers()** function, we use the **global** keyword with the state variable inside the function.

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

state = None

# Define a function to count fingers
def countFingers(image, hand_landmarks, handNo=0):

    global state

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

Now go to the end of function **countFingers()**, where we are counting the total fingers.

We will, **BOTH Play** & **Pause** when **fist**(all fingers closed, finger count will be 0) is **detected**, that means, close fingers to play and close fingers to pause alternatively.

To do this:

- We will keep the **state** variable value as "**Play**", when all fingers are open, that means **totalFingers** is 4.
- Then we will check if the **totalFingers** is 0 and the **state** is "**Play**", we will trigger the **spacebar** key using **press(<keyname>)** method of **pyautogui** library:

  keyboard.press(Key.space)

This will pause the video. We will also change the state to “**Pause**”.

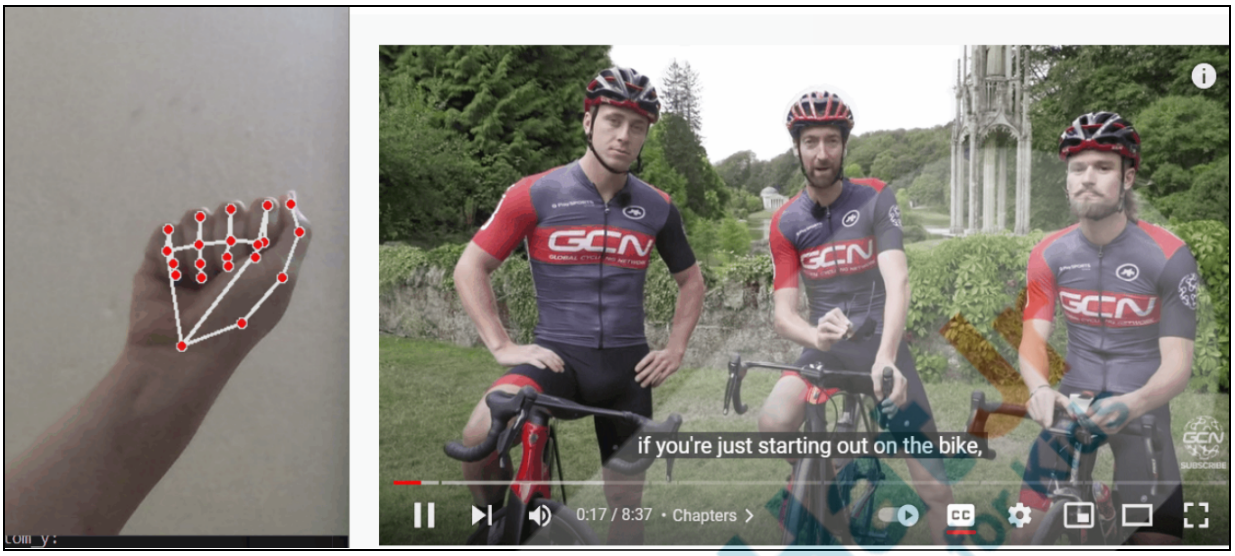
Let’s run the code and check output:

1. Run the code script.
2. Open any YouTube on any browser.
3. Search for a video.
4. Open and Close fingers to check the output.

***Note**: While testing the code, make sure the script is running and the window in which the video is playing is the current active.*

```
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")


totalFingers = fingers.count(1)

# PLAY or PAUSE a Video
if totalFingers == 4:
    state = "Play"

if totalFingers == 0 and state == "Play":
    state = "Pause"
    keyboard.press(Key.space)
```

OUTPUT: Output Video Reference



Move a video FORWARD/BACKWARD By Controlling ARROW Keys Using Hand INDEX FINGER:

Now to move the video in the forward or backward direction, we will use only one finger, the index finger to control the left and right arrow keys.

When we move the index finger towards the right-hand side, it will trigger the right arrow key, which will move the video forward and if we move the index finger to the left it will trigger the left arrow key, which will move the video backwards!

So first we need to check the count of fingers.

- Get the **width** of the output window using int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

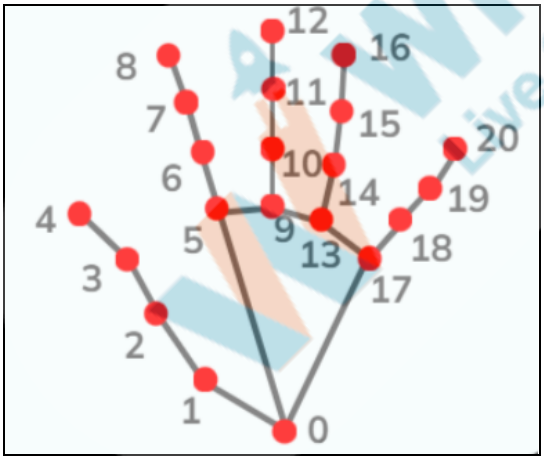
  ***Note 1**: Help the student recollect the OpenCV, CAP_PROP_FRAME_WIDTH and CAP_PROP_FRAME_HEIGHT properties to get the video width and height that we used in the previous classes.*

***Note 2:** The code is given as part Teacher Boilerplate Code.*

```
keyboard = Controller()

cap = cv2.VideoCapture(0)

width  = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height  = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

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

- Get the **x** position of the index fingertip using the fingertip id, 8. Since the landmarks values returned by **process()** method are in decimals, multiply it with output window **width** to make it relative to the output window size.

  finger_tip_x = (landmarks[8].x)*width



- Check if totalFingers == 1:
  - Check if the x position of the tip index finger is less than **width** - 400
    - Press LEFT ARROW Key

- Check if the x position of the finger is greater than **width**-50
  - Press the RIGHT ARROW Key

***Note**: The numbers to subtract from the **width** are selected randomly based on the width of the current output window. In the document, the output has a width of **640px**.*

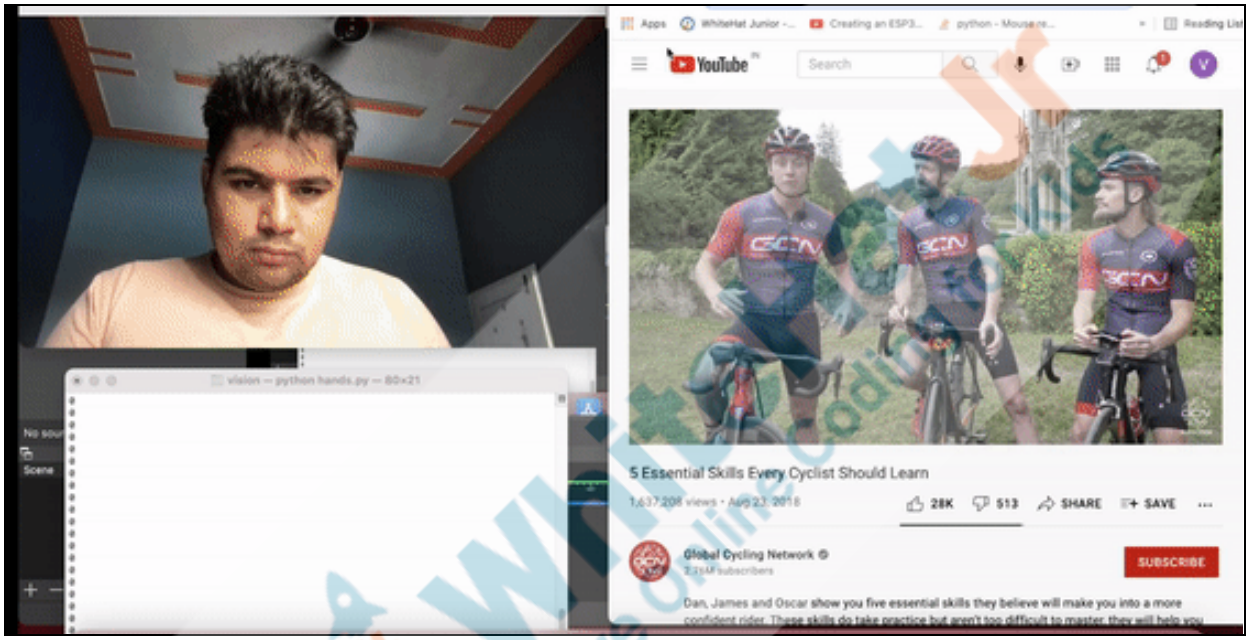
Here, the finger used to check the left, or the right position is the index finger. Even if the user is showing some other finger, we will be able to track all the other fingers as well. So this will work just fine.

```
# PLAY or PAUSE a Video
if totalFingers == 4:
    state = "Play"

if totalFingers == 0 and state == "Play":
    state = "Pause"
    keyboard.press(Key.space)

# Move Video FORWARD & BACKWARDS
finger_tip_x = (landmarks[8].x)*width

if totalFingers == 1:
    if  finger_tip_x < width-400:
        print("Play Backward")
        keyboard.press(Key.left)

    if finger_tip_x > width-50:
        print("Play Forward")
        keyboard.press(Key.right)
```

Let's run the code and check output:

1. Run the code script.
2. Open any YouTube on any browser.
3. Search for a video.
4. Open and Close fingers to check the output.

[Output Video Reference]



We are able to control the videos, but it would be very interesting if we can drag and drop to move the files/folders from one position to another, on the desktop screen using hand gestures.

Are you excited to try it out?

ESR: Yes

Teacher Stops Screen Share

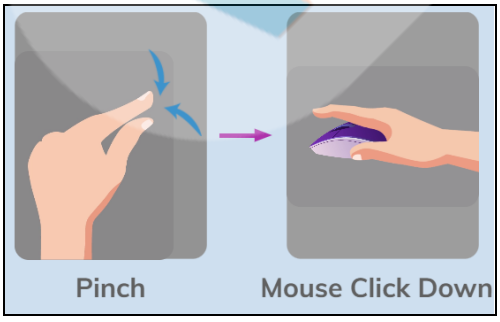
Teacher Starts Slideshow

Slide 11 to 19

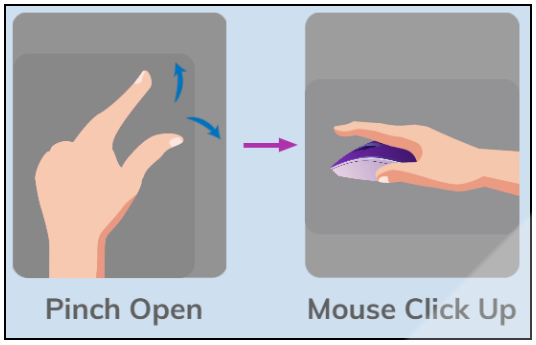
Refer to speaker notes and follow the instructions on each slide.

<table>
<tr><td>We have one more class challenge for you.
Can you solve it?

Let's try. I will guide you through it.</td><td></td></tr>
<tr><td colspan="2">Teacher Ends Slideshow</td></tr>
<tr><td colspan="2">STUDENT-LED ACTIVITY - 20 mins</td></tr>
<tr><td colspan="2">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.</td></tr>
<tr><td colspan="2"><u>ACTIVITY</u>
Drag and Drop files on the desktop using a virtual pinch gesture.</td></tr>
<tr><td>Teacher Action</td><td>Student Action</td></tr>
<tr><td>In the last activity, we have built a program to control keys using our hand gestures.

Now we are going to show you similar concepts to write a program which will enable us to drag and drop files using hand gestures such as:
Pinch to trigger the left mouse click to select the files/folders on the desktop.  </td><td></td></tr>
</table>

- While holding the pinch, to the hold left mouse click, move your hand to move the file/folder.
- Open the pinch, to release the mouse click, to drop the file/folder.



Guide the student to download the boilerplate code using Student Activity 1 code.
***Note**: The boilerplate is the same as previous class with few additions. Explain the following addition to the previous class code to the student.*

But before we start coding, we are going to install two libraries called **pynput** and **PyAutoGUI** using the **pip** command:

pip install <module_name>

```
pip install pynput
pip install pyautogui
```

pynput: Used to observe any input received from the input devices like mouse and keyboards.

PyAutoGUI: Helps to control mouse and keyboards through Python programs.

Since we want to control the mouse, we will: ● Import **math** module, which will be later used to calculate the distance between the tip of the finger and tip of the thumb when the pinch gesture is formed. ● Import the **Button** and **Controller** class from the **pynput.mouse** module. ● Import **pyautogui** module. ● Initialize the **Controller** object using a variable called **mouse**. ● Get the **width** and **height** of the output window using OpenCV CAP_PROP_FRAME_WIDTH and CAP_PROP_FRAME_HEIGHT properties. ● Get the Desktop/Laptop/Computer/System screen size using the **size()** method of the **pyautogui** module into **screen_width** and **screen_height** variables. ● Take a **global** variable **pinch** to detect the pinch formation gesture.	

```
import cv2
import math
import mediapipe as mp
from pynput.mouse import Button, Controller
import pyautogui

mouse=Controller()

cap = cv2.VideoCapture(0)

width  = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height  = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

(screen_width, screen_height) = pyautogui.size()

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)

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

pinch=False

# Define a function to count fingers
def countFingers(image, hand_landmarks, handNo=0):

    global pinch

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

The logic to build this project is very simple, we will follow these steps.

- Draw a line from the points of the tip of the index finger and tip of the thumb.
- Draw a circle in the center of the line to represent the mouse on the output window.
- Calculate the length of this line(distance from the tip

of the index finger to the tip of the thumb).

- If the distance is less than 40(index finger is near the thumb), this will be a pinch gesture, and we will trigger a left mouse click down.
- If the distance is greater than 40, which means pinch is open, and we will trigger a left mouse click up.

```
totalFingers = fingers.count(1)

# PINCH

# Draw a LINE between FINGER TIP and THUMB TIP


# Draw a CIRCLE on CENTER of the LINE between FINGER TIP and THUMB TIP


# Calculate DISTANCE between FINGER TIP and THUMB TIP


# Set Mouse Position on the Screen Relative to the Output Window Size


# Check PINCH Formation Conditions
```

Draw the LINE between TIPS:

Now we will draw a line from the tip of the index finger to the tip of the thumb.
In order to do this we need to get the x, y position of both the tips, and then we will draw a line:
We will use the ID to get the position of the finger points. In our case, the thumb has an ID of 4 and the index finger's ID is 8.

- Get the x position of the finger tip and multiply the value with the output window width to make it

relative to the output window size, into a variable called, **finger_tip_x**.

- Get the y position of the finger tip and multiply the value with the output window width to make it relative to the output window size, into a variable called, **finger_tip_y**.

- Get the x position of the finger tip and multiply the value with the output window width to make it relative to the output window size, into a variable called, **thumb_tip_x**.

- Get the y position of the finger tip and multiply the value with the output window width to make it relative to the output window size, into a variable called, **thumb_tip_y**.

- Draw a line between two tips using using the **cv2.line()** method:

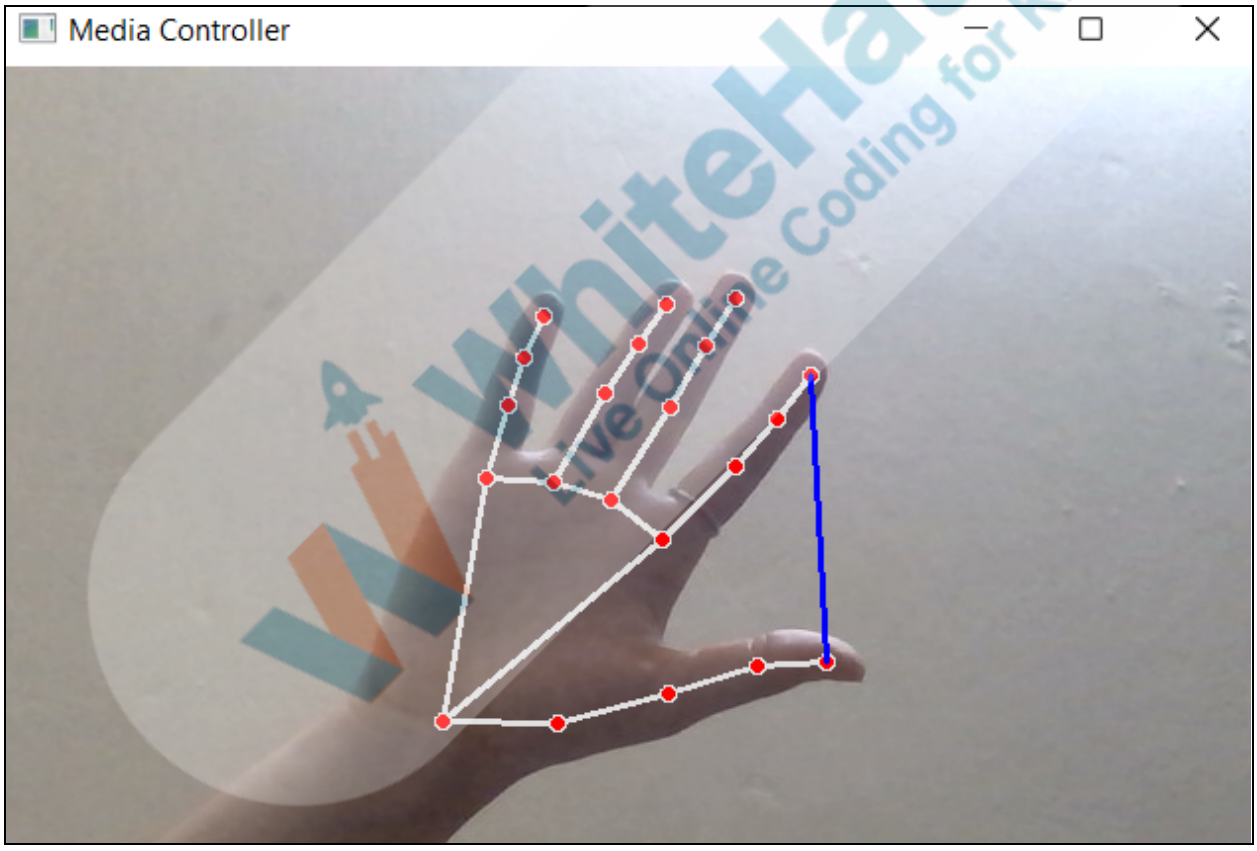
  Syntax:

```
cv2.line(img, pt1, pt2, color,thickness)
```

  - **img**: Image/Video we are working with
  - **pt1**: First Point (x, y) coordinates
  - **pt2**: Second Point (x, y) coordinates
  - **color**: color of the line
  - **thickness**: thickness of the line

```
# PINCH

# Draw a LINE between FINGER TIP and THUMB TIP
finger_tip_x = int((landmarks[8].x)*width)
finger_tip_y = int((landmarks[8].y)*height)

thumb_tip_x = int((landmarks[4].x)*width)
thumb_tip_y = int((landmarks[4].y)*height)

cv2.line(image, (finger_tip_x, finger_tip_y),(thumb_tip_x, thumb_tip_y),(255,0,0),2)
```

OUTPUT: Run the code and check output.



Draw the CENTER on the LINE:

The next step is to draw a center between this line, this will

represent the **virtual mouse** inside the output window that will be used to control the **actual mouse** on the desktop screen!

To do this:

- Get the **center_x** position using half of the sum of the **finger_tip_x** and **thumb_tip_x** position.
- Get the **center_y** position using half of the sum of the **finger_tip_y** and **thumb_tip_y** position.
- Draw a circle at the **center_x** and **center_y** position

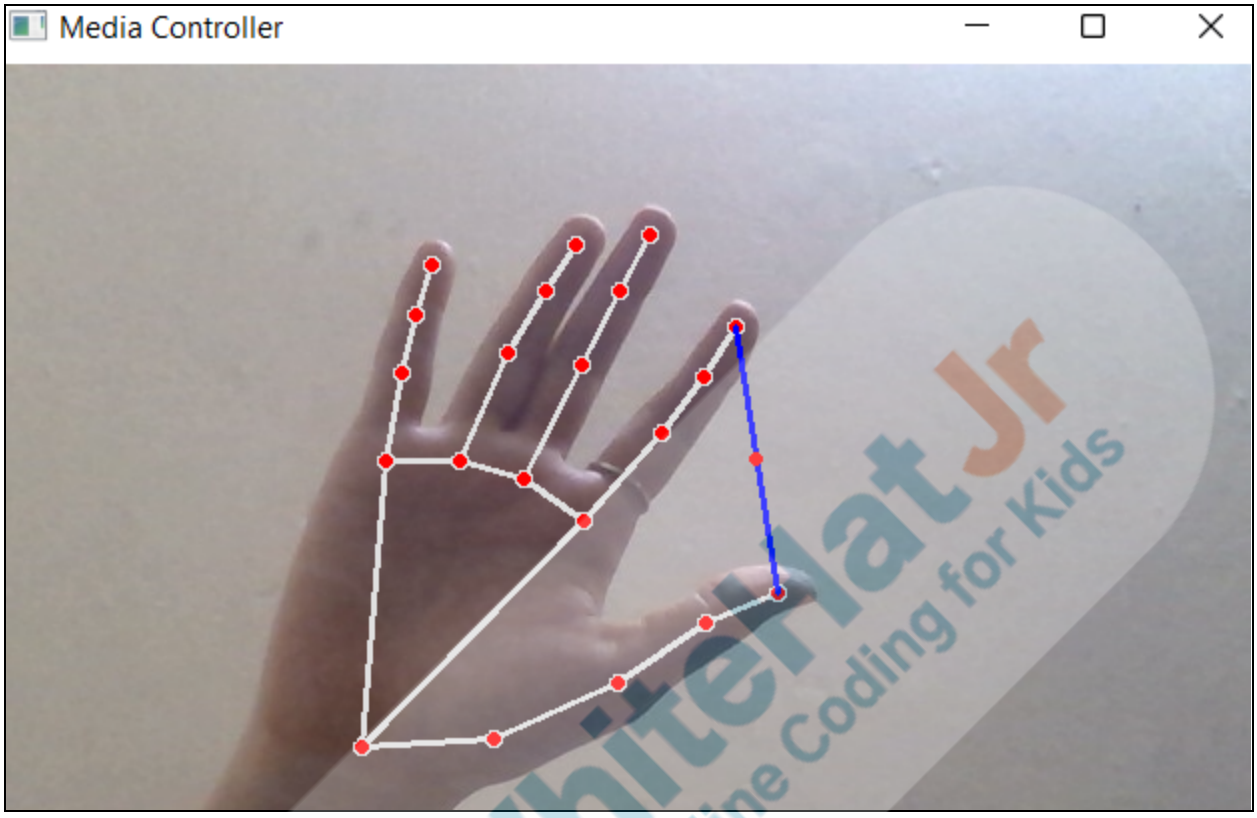
  Syntax:

```
cv2.circle(img, center, radius, color, thickness)
```

  - **img**: Image/Video we are working with
  - **center**: Center Point (x, y) coordinates
  - **radius**: Radius of the circle
  - **color**: color of the circle
  - **thickness**: thickness of the circle

```
# Draw a CIRCLE on CENTER of the LINE between FINGER TIP and THUMB TIP
center_x = int((finger_tip_x + thumb_tip_x )/2)
center_y = int((finger_tip_y + thumb_tip_y )/2)

cv2.circle(image, (center_x, center_y), 2, (0,0,255), 2)
```

OUTPUT: Run the code and check output.



DISTANCE between the TIPS:

Now let's get the **distance** using the Distance Formula we have used in previous classes.

Distance Formula:
To calculate distance between Point 1 with coordinates (x1, y1) and Pont 2 with coordinates (x2, y2)

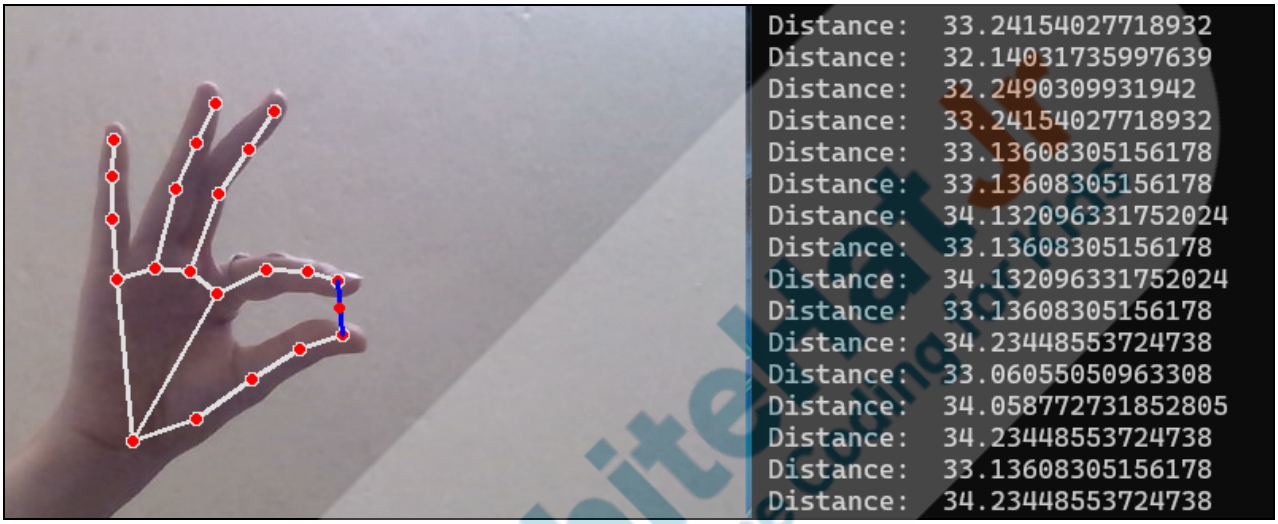
$$D = \sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}$$

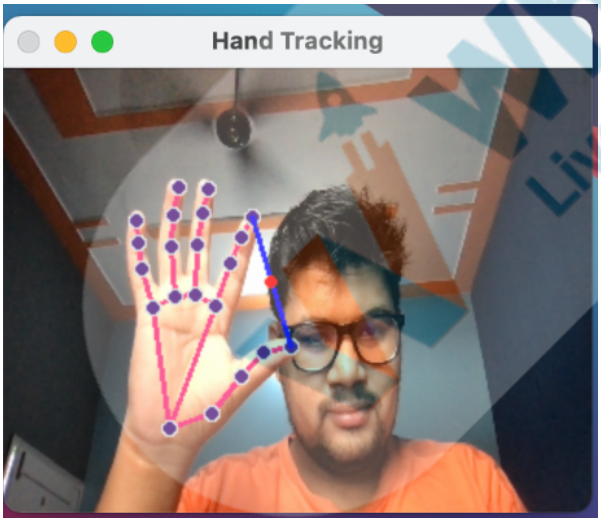
In our program:

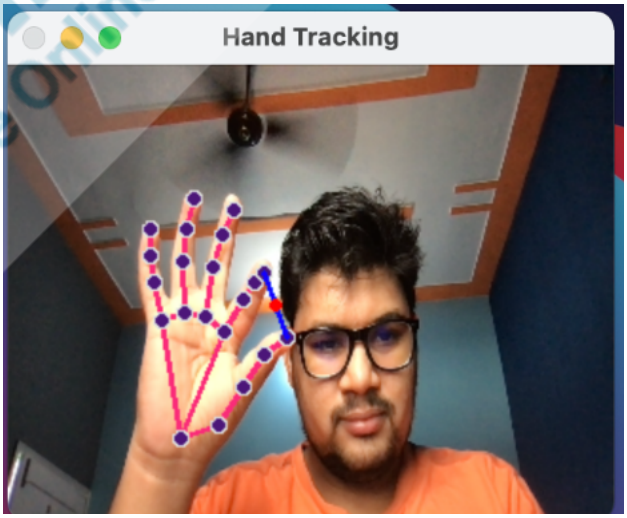
x1: finger_tip_x y1: finger_tip_y	x2: thumb_tip_x y2: thumb_tip_y

```
# Calculate DISTANCE between FINGER TIP and THUMB TIP
distance = math.sqrt(((finger_tip_x - thumb_tip_x)**2) + ((finger_tip_y - thumb_tip_y)**2))

print("Distance: ", distance)
```

OUTPUT: Run the code and check output.







Now we can see a line drawn from the index finger to the thumb, and as we pinch, the line gets smaller. We also have a red circle at the center of our line.

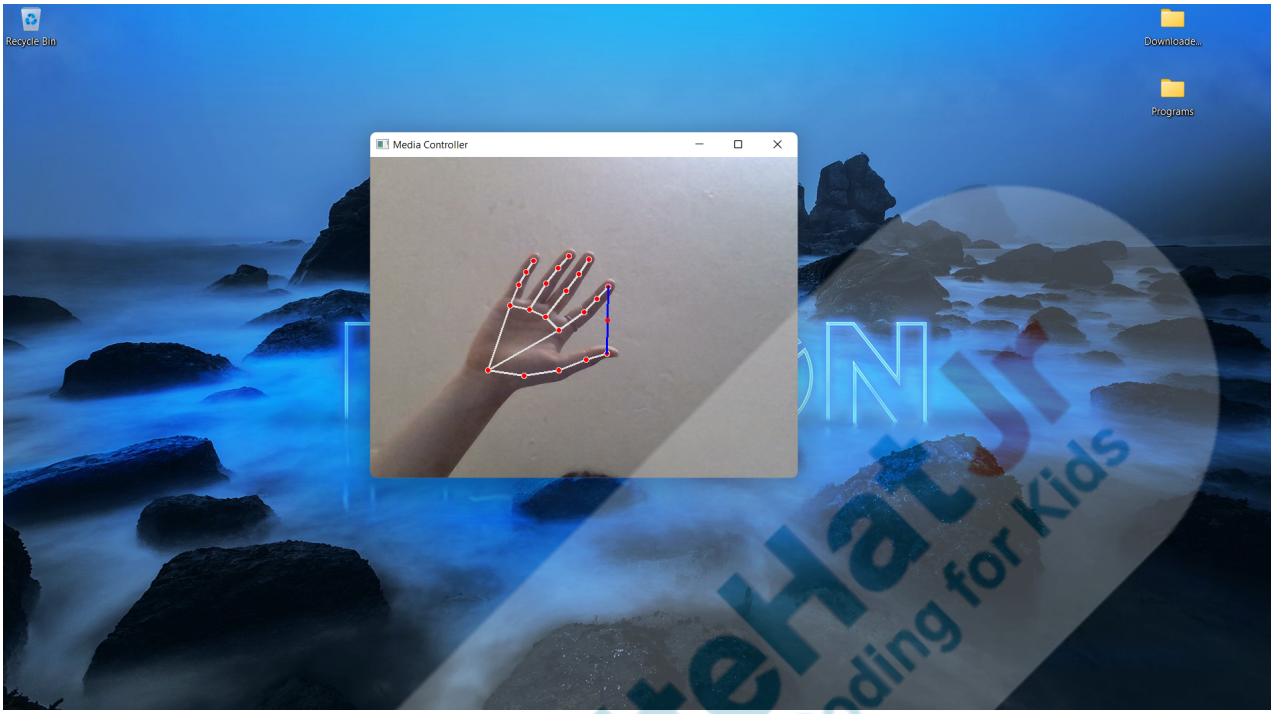
Now we will control the mouse pointer on the screen.

Before we can do that let's print:

- Computer/Desktop/Laptop Screen Size: **screen_width**, **screen_height**
- Output(Webcam) Window Size: **width**, **height**
- Actual Mouse Position:This can be achieved using a property called **position** of the **Controller** object initialized using the **mouse** variable as **mouse.position**.
- Tip Line Center Position: **center_x**, **center_y**

```
print("Distance: ", distance)

print("Computer Screen Size :",screen_width, screen_height, "Output Window size: ", width, height)
print("Mouse Position: ", mouse.position, "Tips Line Centre Position: ", center_x, center_y)
```

OUTPUT: Run the code and check output(only one print output)



As we can see that:

- Actual computer screen size is (1920, 1080) and,
- Output (web camera) window size is (640,480)

***Note**: The size value might vary from system to system.*

With this the hand can be only detected within the (640, 480) dimensions.

Now to click the mouse pointer on the actual screen, the distance moved by the mouse pointer in x and y direction is more than the distance moved by our hand in the output window!

For example, in the output(webcam) window we can only move 640 points in the x direction, but on the computer screen the mouse can move 1920 points in x direction.

Set the VIRTUAL MOUSE(CENTER of LINE) Relative to the ACTUAL MOUSE POINTER:

Now that we have two window sizes that are different, we need to map the virtual mouse position, represented by the center of the line between the tip of the finger and the tip of the thumb, relative to the actual mouse pointer.

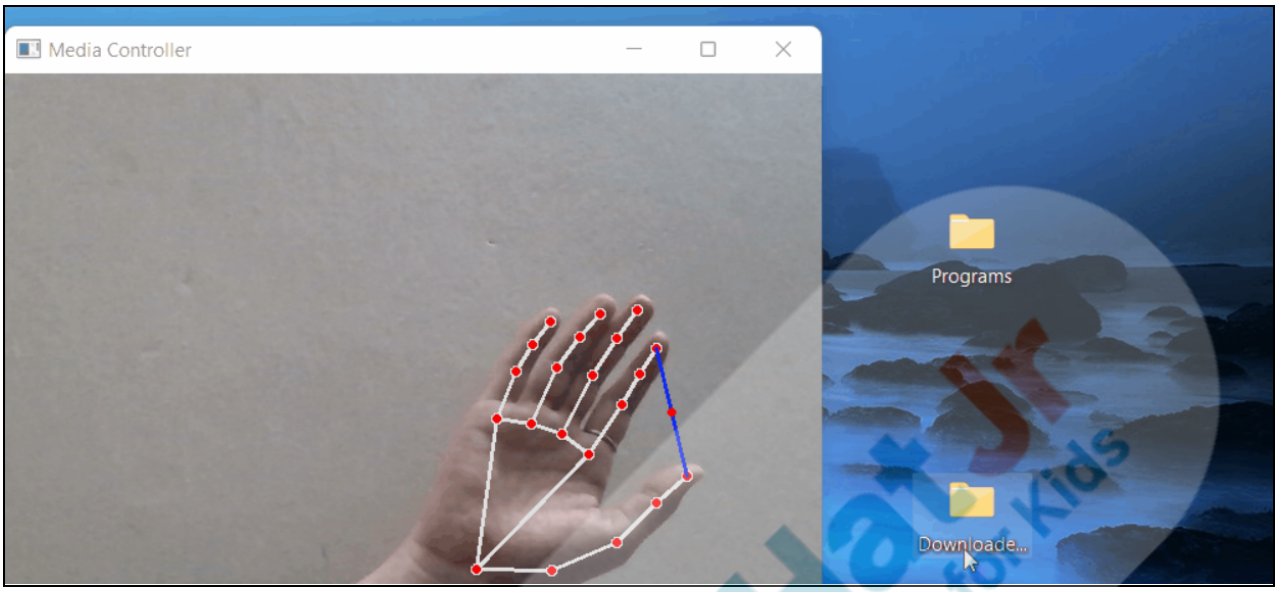
This **cannot** be done by directly equating the **center_x** and **center_y** values of the center points and **mouse.position** as only being able to move in a box of (640, 480).

To map the relative position of the center and actual mouse:

- Get the **relative_mouse_x** position, by taking the ratio of **center_x** to **width** of the output window, then multiply the result by **screen_width**
- Get the **relative_mouse_y** position, by taking the ratio of **center_y** to **height** of the output window, then multiply the result by **screen_height**
- Equate the **(relative_mouse_x,relative_mouse_y)** to the **mouse.position**.

```
# Set Mouse Position on the Screen Relative to the Output Window Size
relative_mouse_x = (center_x/width)*screen_width
relative_mouse_y = (center_y/height)*screen_height

mouse.position = (relative_mouse_x, relative_mouse_y)
```

OUTPUT: Run the code and check output.
Video Reference
***Note**: Once the hand is detected in the webcam output window your mouse control will be relative to the center of the line or virtual mouse. To control the system back with the actual mouse, take your hand from the webcam output window.*



Now you can see we can control the actual mouse on the whole computer screen using the virtual mouse when the hand is detected, let's trigger the mouse click using the pinch gesture on the screen.

Here we will check the value of the **pinch** and **distance** variable, using the **if** condition:

- Check if the distance value is more than 40,
  - Check if **pinch** is **True** and, that means the finger tip and and thumb tip is far from each other,
    - Then assign **pinch = False**
    - And release the mouse click using **mouse.release(Button.left)**

- Check if the distance value is less than 40,
  - Check if **pinch** is **False** and, that means the finger tip and and thumb tip are touching each other,
    - Then assign **pinch = True**
    - And press the mouse click using **mouse.press(Button.left)** method.

```
mouse.position = (relative_mouse_x, relative_mouse_y)

# Check PINCH Formation Conditions
if distance > 40:
    if pinch == True:
        pinch = False
        mouse.release(Button.left)

if distance <= 40 :
    if(pinch==False):
        pinch=True
        mouse.press(Button.left)
```

OUTPUT: Run the code and check output.
Video Reference

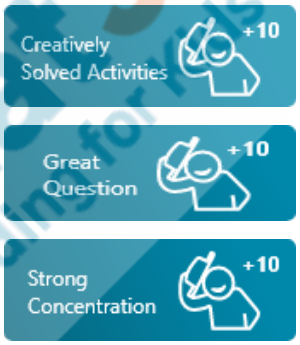
Great! In this class, we have used hand tracking to build two really cool applications.

One is to control the media by triggering keyboard keys, and the second is to control the mouse cursor using our hands. These are really fun projects to build using the OpenCV library. In the upcoming classes, we will understand the basics of machine learning.	
Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 05 mins	
Teacher Starts Slideshow **Slide 20 to 24** 	
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 25 to 30** 	
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

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C109-Teacher-Boilerplate
Teacher Activity 2	Reference Code	https://github.com/procodingclass/PRO-C109-Reference-Code
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C109-Student-Boilerplate
Teacher Reference 1	The pynput Module	https://pypi.org/project/pynput/
Teacher Reference 2	PyAutoGUI Module	https://pyautogui.readthedocs.io/en/latest/
Teacher Reference 3	Project Document	https://s3-whjr-curriculum-uploads.whjr.online/82751ecd-ab94-439a-9442-7e1aad04af6a.pdf
Teacher Reference 4	Project Solution	https://github.com/pro-whitehatjr/Project-solution-C109
Teacher Reference 5	Visual Aid Link	https://s3-whjr-curriculum-uploads.whjr.online/5c45032d-dfeb-47e0-91bd-360f5b8165d2.html
Teacher Reference 6	In Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/735a78a4-ea12-4470-8687-63bd83ff952b.pdf