Topic	FACE DETECTION	
Class Description	**Students will learn the basics of face detection using Haar Cascade Classifiers. Students will apply the learnings to detect faces on images and videos.**	
Class	**PRO C106**	
Class time	**45 mins**	
Goal	• Learn the basics of face detection using Haar feature-based Cascade Classifiers • Write a code to detect face on images • Write a code to detect faces on each frame of videos from the webcam.	

Resources Required	• Teacher Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Visual Studio Code
	• Student Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Visual Studio Code	

Class structure	**Warm-Up** **Teacher-led Activity 1** **Student-led Activity 1** **Wrap-Up**	**05 mins** **15 mins** **20 mins** **05 mins**
Credit	Object Detection using Haar feature-based cascade classifiers is an effective method proposed by Paul Viola and Michael Jones in the 2001 paper, "Rapid Object Detection using a Boosted Cascade of	

	Simple Features". All GIFs used in this class are created in house. The images used in face_detect are licensed copies.	

WARM-UP SESSION - 05 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 12

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 • **Detect Faces Using Haar-Cascade**	
Teacher Action	**Student Action**
Before, I will get into details of today's class. Let me show you something. *The teacher should download Teacher Activity 1. Unzip the folder and save it as C106. Open in VSC. Run the code in the Terminal using* ***python face_detect.py*** ***The code uses a webcam. Make sure to turn off your camera in the class, so that the code can use it.*** *The webcam will show the teacher's face highlighted with a rectangle, the teacher can move slowly from left to right or up and down simply to show that the rectangle follows the teacher's face.* Can you tell me what is happening? The code I have written is able to detect my face and is highlighting it with a rectangle. Have you seen any other device that can recognize or detect your face?	**ESR:** Varied **ESR:** Smartphones. **ESR:** Varied

Today, we are going to write a code to detect a face from an image. **Face Detection** detects the presence of a face in an image. However if you want to identify the face with pre-stored images then we would write a code that compares the detected image of a face. This process is known as face recognition. Today we will learn how to detect the face using OpenCV and Python in an image. There are a couple of methods for implementing face detection using Python. Are you excited to try out?	**ESR**: Yes
When we try to detect any particular feature in an image, we basically **classify** or distinguish different parts of an image. We identify trees, houses, mountains, vehicles, humans, and so on. For computers to detect the face of humans, we need to create an algorithm that will classify the image into two parts, face and rest of image. **Algorithms** are a set of instructions given to the computer to perform a specific task. OpenCV has face detection classifiers data stored in the form **XML** file format, which can be readily used in a program. This classifier is known as the **Haar Cascade Classifier**.	

There are multiple Haar Cascade Classifiers available in the OpenCV library. These classifiers **XML** files get downloaded along with **opencv-python** installation using **pip** to download the **cv2** package.

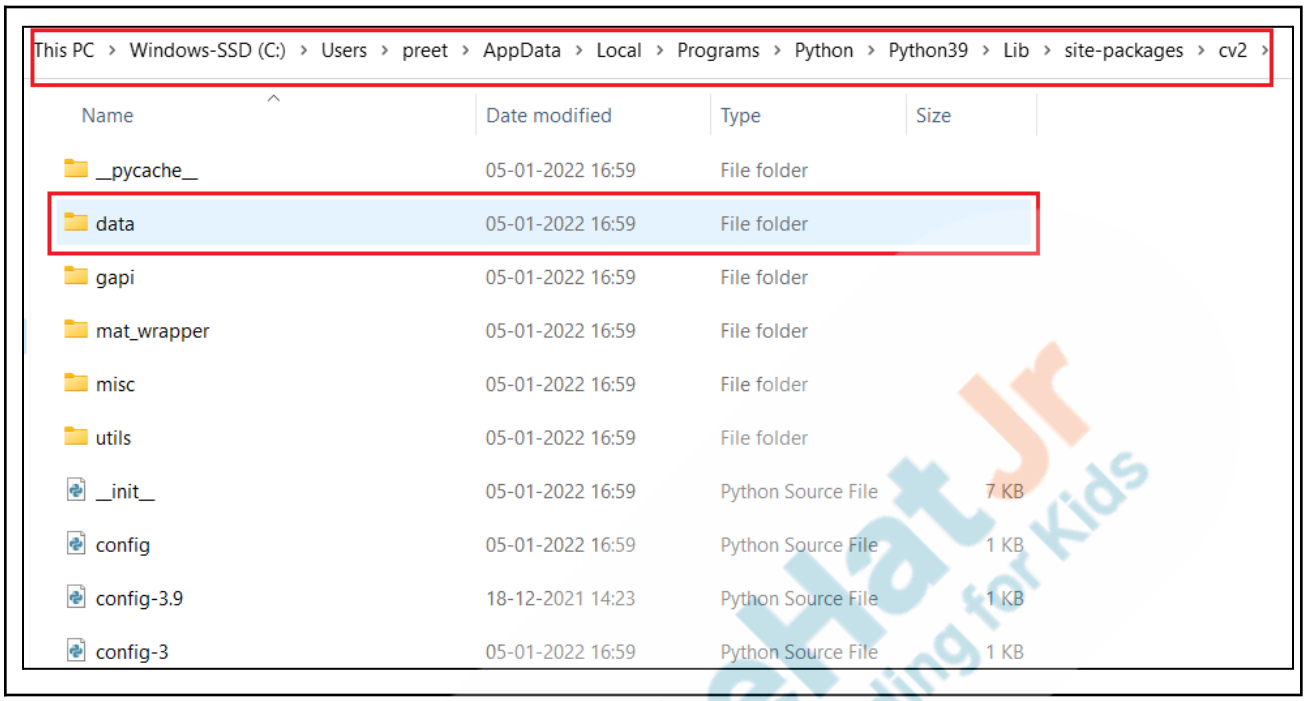
All the Haar Cascade files are located in the "**data**" folder where **cv2** is installed in your system.

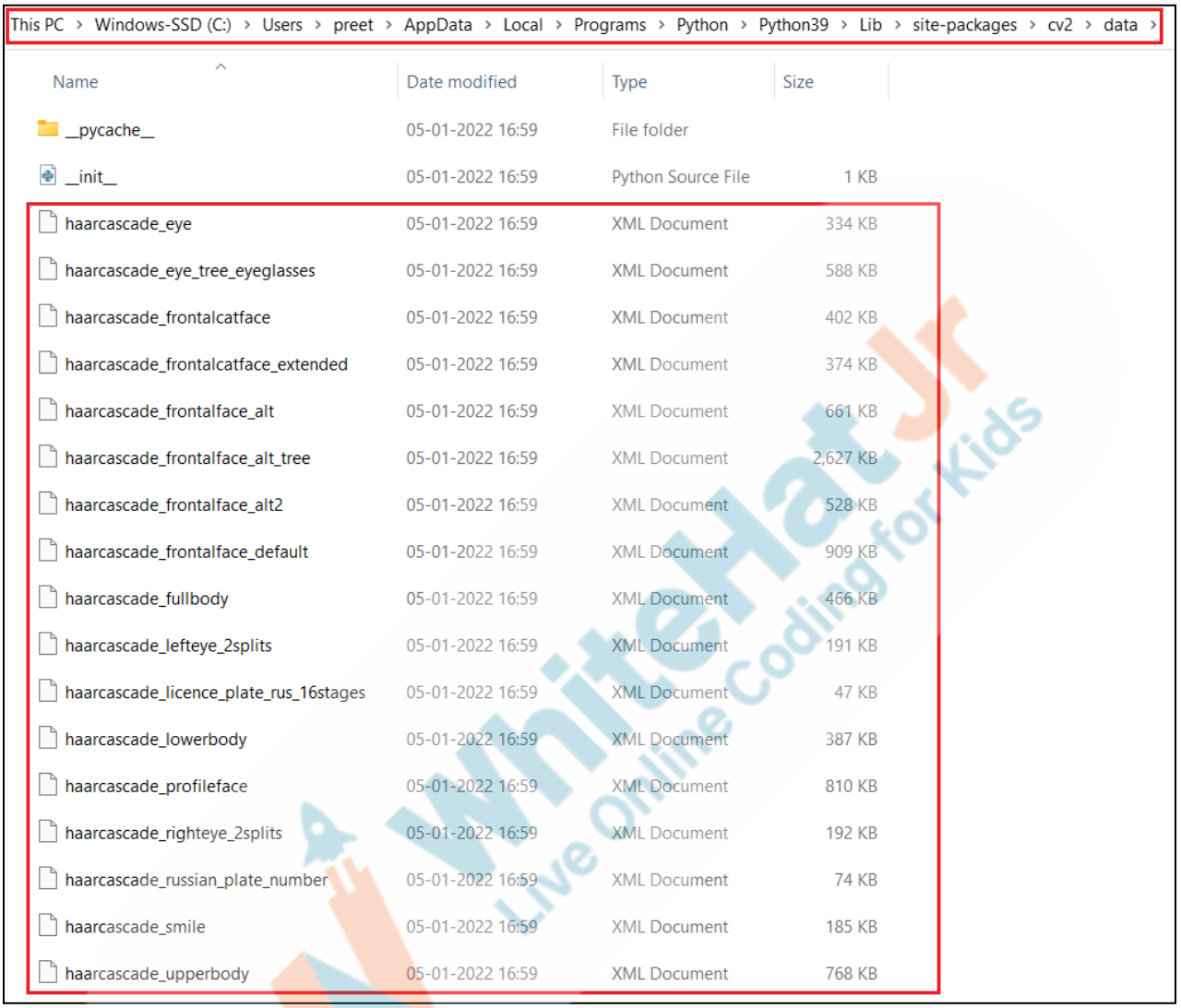
Full Path to **cv2/data** folder:

C:/Users/preet/AppData/Local/Programs/Python/Python39/Lib/site-packages/cv2/data

***Note 1**: The below image shows the path to the **cv2/data** folder in the Windows system which might vary from the system used by the teacher/student.*

***Note 2**: Teacher can check for the cv2 path in her system to show it to the student.*





Alternatively, the classifier files can be accessed from here.
OpenCV Haar Cascade Classifiers

While we understand about the Haar Cascade Classifier, we won't deep dive into the mathematics and calculation of how the classifier algorithm is constructed.

The general process of classification will be discussed during image classification in the upcoming classes. Haar Cascade Classifiers also use a similar approach to detect faces or any other objects in the image/video.

Today we will only learn what it does and how to use the classifier in our code.

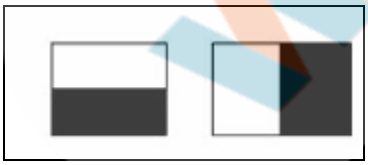
Haar Cascade is an Object Detection Algorithm used to identify object in an image or a real-time video

As far as the human face is considered, the **nose**, **lips**, **eyebrows**, **forehead**, and **eyes** are considered as the most relevant features of a face.

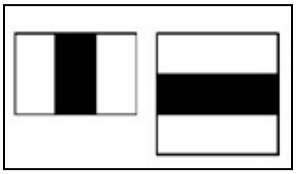
The haar cascade algorithm is looking for these features in an image to identify a face.

It does this by creating patterns with pixels next to each other, these patterns are either edge detection or line detection:

- **Edge Detection** pattern is a series of white pixels above a series of black pixels or reverse, this pattern can be horizontal or vertical.



- **Line Detection** pattern is series of black pixels sandwiched between series of white pixels or reserve i.e white pixels sandwiched between black, this pattern can also be vertical or horizontal.

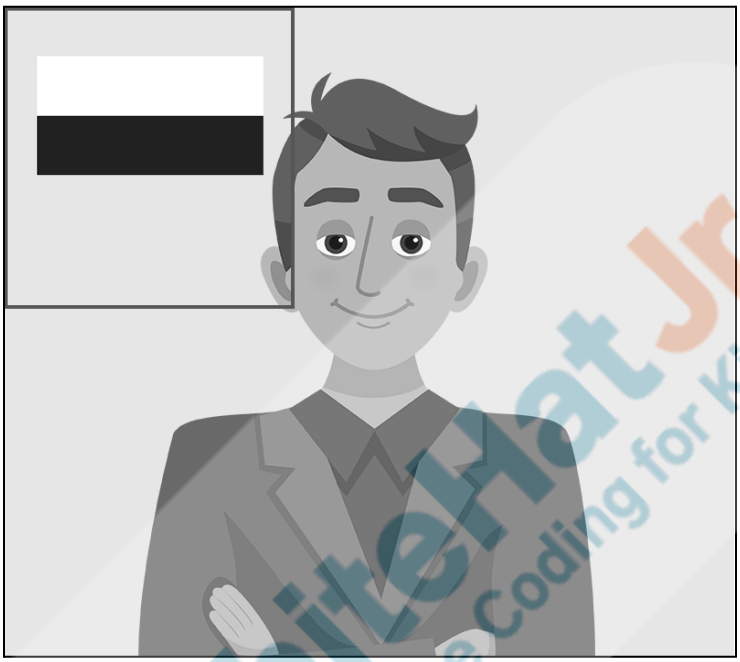


The Haar Cascade algorithm creates hundreds of such patterns and will then go through each of them to identify the closest feature of the face like **eyebrow** (edge pattern), **lips** (line pattern), **nose** (line pattern), **eyes** (edge pattern).

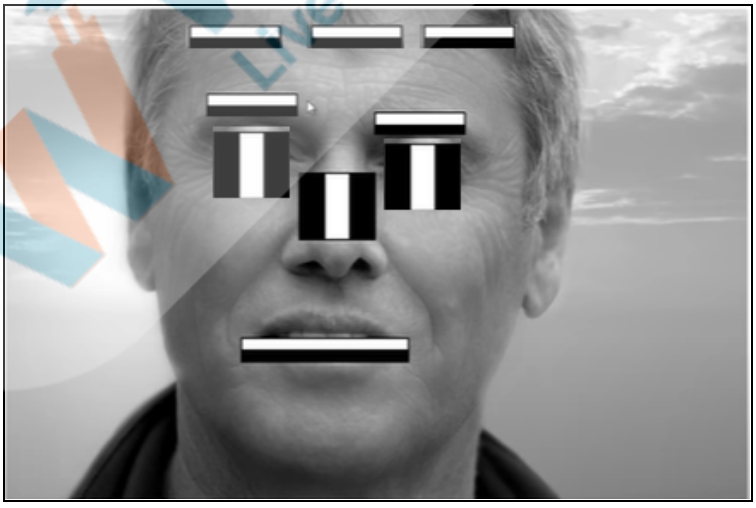
Face Detection using Haar Cascade Classifier:

- In order to identify all the features, the Haar Cascade **Edge/Lines Feature** runs over the image portion(the rectangle box), known as the **detection window**.

- The detection process is repeated multiple times, again and again till all the features are captured. The whole image traversed using the detection window and Haar Cascade features.



- Gradually, it will cover more features of the face, using edge and line patterns, like **eyebrow** (edge pattern), **lips** (line pattern), **nose** (line pattern), **eyes** (edge pattern).



- The classifier will keep running till it is able to identify the features and at the end it combines all the features and returns a rectangle output in the form of **x, y** the starting point from where face is detected, along with width & height.

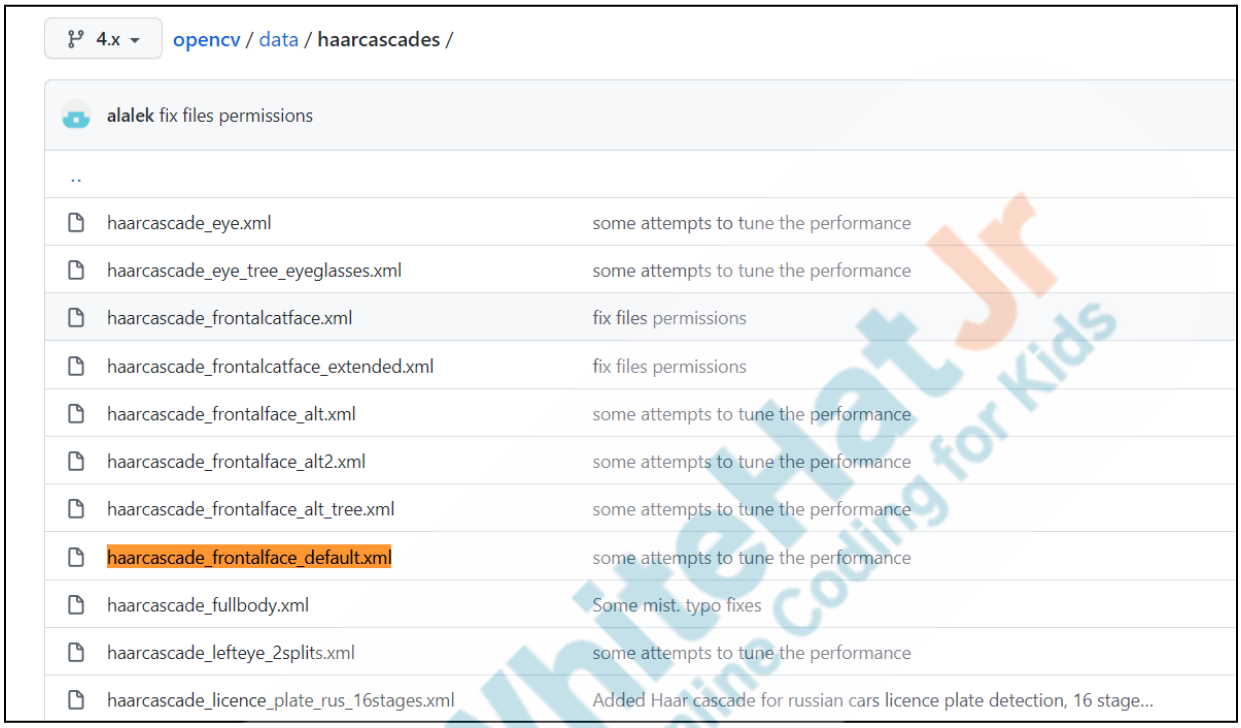
Let us now use the OpenCV library to detect faces in an image.

The teacher returns to VSC Editor.

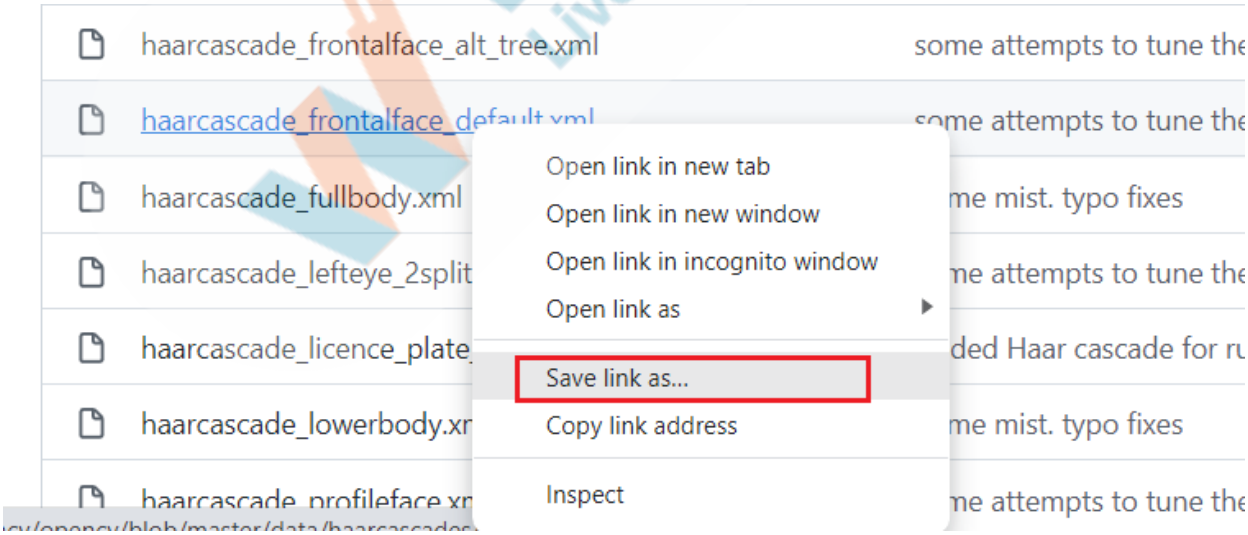
The other files we have here are :

- An image with a single face, **boy.jpg**
- An image with multiple faces, **4f.jpg** and
- An **XML** file, **haarcascade_frontalface_default.xml**, that contains Haar Cascade data for face detection.

The classifier files can be accessed from the **opencv** GitHub Repository here.
OpenCV Haar Cascade Classifiers



Right click on the **.xml** file, then click on "**Save link as…**" to download the file in your system



Let us create a new file, **face_detect_on_image.py** What do we need to import to work with images? Yes, let us **import cv2.** Which method do we need to read an image? We will make our code detect the face on the image with a single face first. *The teacher writes **cv2.imread("boy.jpg")***	**ESR:** We need to import **cv2** **ESR:** We use **cv2.imread()** method to read the image.

```
import cv2

img = cv2.imread("boy.jpg")
```

Haar cascade runs on **grayscale** images, so we need to convert our image to grayscale first.

The OpenCV reads images in BGR format, so we need to convert **BGR** to **GRAY**.

For this we can use **cvtColor(<image_name>, <property_name)** method to change a colored image to a grayscale image:

- **image_name** : The **img** variable used to read boy image
- **property_name** : The **cv2.COLOR_BGR2GRAY** can be used to change colored images tinto grayscale images.

```
gray= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

The **XML** file contains Haar Cascade data for face detection.
Load the classifier for the frontal face into a **face-cascade** variable using **cv2.CascadeClassifier()** method.

***Note 1**: If the XML file present in the working directory is not detected, then use the full path to the XML files downloaded along with the **opencv-python** installation.*

***Note 2**: The below path to the **cv2/data** folder in the Windows system which might vary from the system used by the teacher/student.*

*Full Path to **cv2/data** folder:*

C:/Users/preet/AppData/Local/Programs/Python/Python 39/Lib/site-packages/cv2/data

```
gray= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

We shall be using the **detectMultiscale()** module of the classifier.

The **detectMultiScale()** requires an image name, which is **gray** in our case, to be used for face detection. This function will return a rectangle with coordinates**(x,y,w,h)** around the detected face.

We can use **print(faces)** to check the result.

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

faces = face_cascade.detectMultiScale(gray)
print(faces)
```

Run the code:

```
[[343  84 242 242]]
```

The result shows a 2D array with four values, the first two values as **x,y** (343, 84) coordinates to start the rectangle and the last two values for the width & height of **242 x242.**

The next step is to loop over all the coordinates it returned and draw rectangles around them using **OpenCV**. We will be drawing a blue rectangle.

To draw a rectangular shape over an image we can use the **rectangle()** method.

Syntax:

cv2.rectangle(image, start_point, end_point, color, thickness)

- **image:** The name of the image used
- **start_point:** The x and y pixels values
- **end_point:** The x and y pixels values
- **color:** The color of the rectangle.
- **thickness:** The border thickness of the rectangle shape.

Can you tell me how many points in terms x and y coordinates do we need to draw a rectangle?

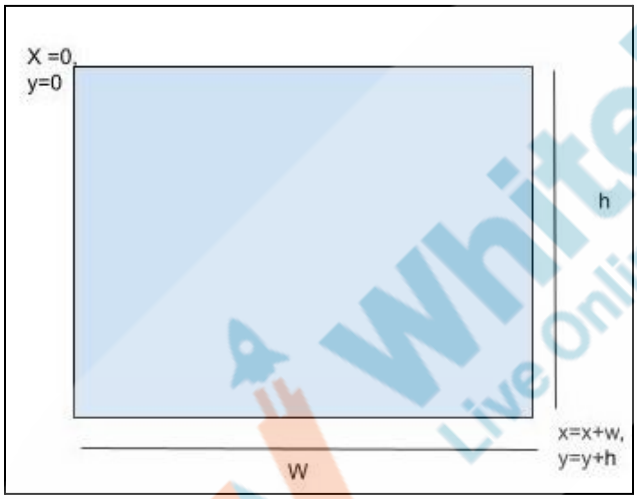
ESR: 4 points.

Yes, we need 4 points to draw a rectangle.

Another way to draw a rectangle is using only 2 points.

To draw a rectangle using only 2 points, we will need width and height of the rectangle.

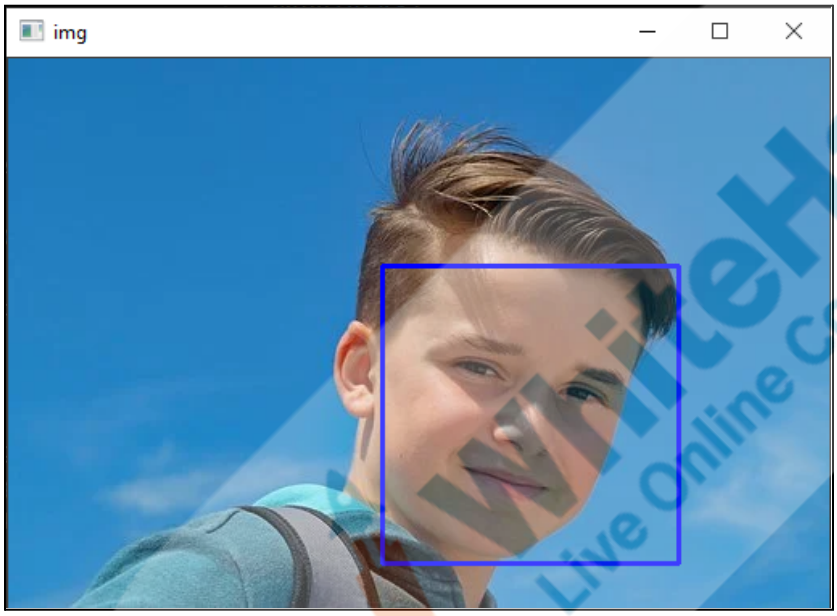
If we start with point **x, y** and the rectangle has **w** as width and **h** as height, then **x, y** as a starting point, and **x+w** and **y+h** will be an endpoint.



To check the result we will use **cv2.imshow() and cv2.waitKey()** to see the result.

```
print(faces)

for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

cv2.imshow('img',img)
cv2.waitKey(0)
```

OUTPUT(In New Window)



As we can see the rectangle is drawn around the face. Why don't you try at your end to detect a face in a picture with many people? We will also run the Haar Cascade to use on live view from the webcam. Share your screen, and we can start.	**ESR:** Ok

<table>
<tr><td colspan="2">Teacher Stops Screen Share</td></tr>
<tr><td colspan="2">Teacher Starts Slideshow 
Slide 13 to 14
Refer to speaker notes and follow the instructions on each slide.</td></tr>
<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>ESR: Yes</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>
• Apply Haar Cascade on Webcam live stream
• Detect multiple faces on image</td></tr>
<tr><td>Teacher Action</td><td>Student Action</td></tr>
<tr><td>The teacher guides the student to download the code from Student Activity 1

The folder includes; an image with a single person, image with multiple people, face_detect.py created by Teacher; live_detect.py contains boilerplate code from the previous class to read webcam images.</td><td>The student downloads the zipped folder from Student Activity 1. Unzip the folder and save it as C106. Opens the folder in VSC</td></tr>
</table>

Let us first go to **face_detect.py** and run to check the output.

Now that we already know the coordinates for a face we can easily crop the face and save it as a separate file.

Remember, in array format, the image takes in **height** first and then **width**. So we need to give a range for height from the starting position of the rectangle which is **y** till **y+h** and the same way for width. And save it in a different variable.

img[y:y+h, x:x+w]

Where should we write this code?

We will write it inside a **for** loop. We will also use the method **cv2.imwrite()** to save the face as a separate image.

The student runs the code to check the output as a rectangle drawn on a baby's face.

ESR: Varied

```
faces = face_cascade.detectMultiScale(gray,1.1, 5)
print(len(faces))

for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
       # Crop the image to save the face image.
        roi_color = img[y:y+h, x:x+w]
        cv2.imwrite("face.jpg",roi_color)

cv2.imshow('img',img)
cv2.waitKey(0)
```

OUTPUT

The teacher can guide student to check on the left side on VSC where a new file is

*created named **face.jpg***

Fantastic, you can now create a passport size pic from any of your photos.

Now, let us try to run the same code for an image with multiple faces.

Change the name of the image in the **imread()** method.
We also add a print() after faces to check how many faces are detected using **len()**

You can write **print(len(faces))**
And run the code.

The teacher can guide the student to modify only the code highlighted with red below. Rest of the code is the same.

```
import cv2

img = cv2.imread("4f.jpg")

gray= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

faces = face_cascade.detectMultiScale(gray)
print(len(faces))
```

Comment the rest of the code for a time being…

```
print(len(faces))

# for (x,y,w,h) in faces:
#         cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
#        # Crop the image to save the face image.
#         roi_color = img[y:y+h, x:x+w]
#         cv2.imwrite("face.jpg",roi_color)

# cv2.imshow('img',img)
# cv2.waitKey(0)
```

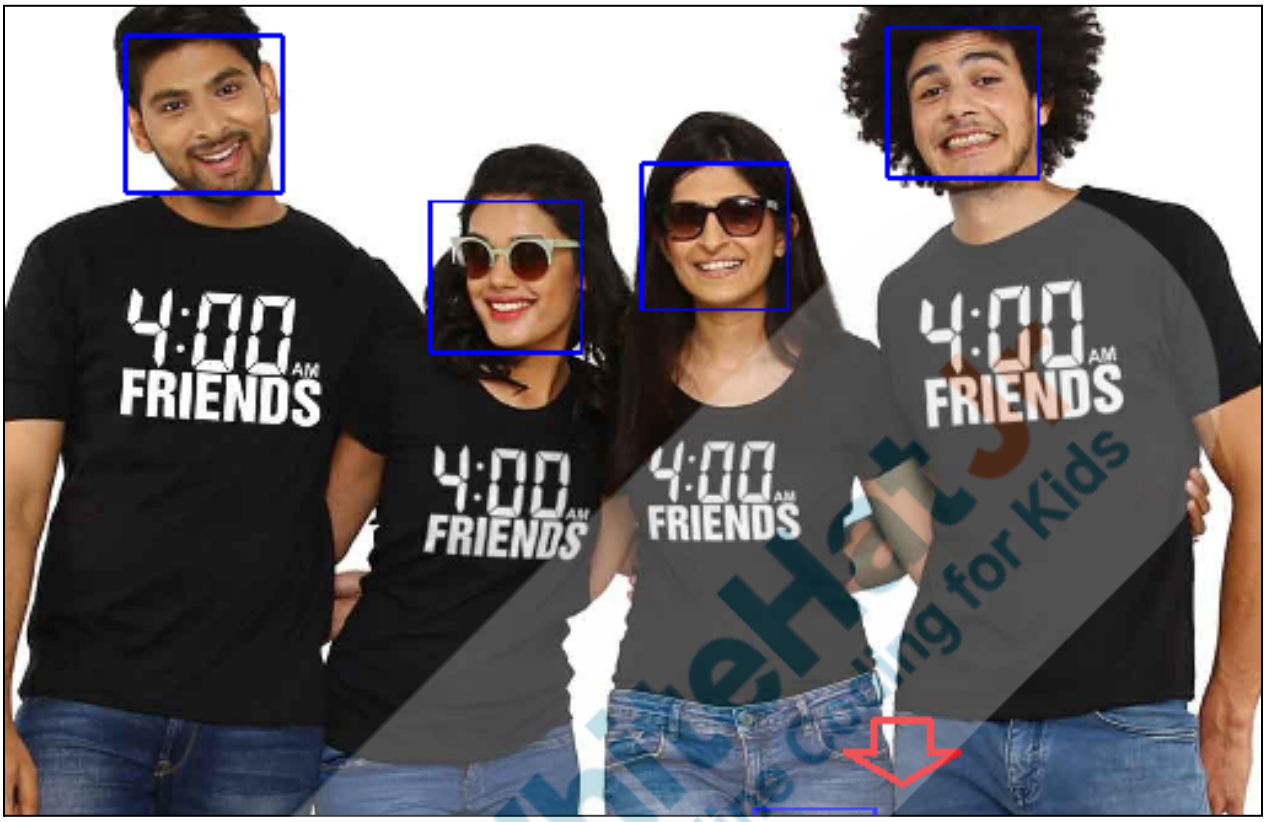
Run the code - OUTPUT:

```
5
```

The output shows 5 faces; can you open the image and check how many faces are there in the image? That means we have a bug. Let us draw the rectangle around the coordinates we have received from the classifier. Then we will fix the bug. You can uncomment the code except for the **roi_color** and **imwrite()** and run the code again.	**ESR:** There are only 4 faces.

```
print(len(faces))

for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
#        # Crop the image to save the face image.
#         roi_color = img[y:y+h, x:x+w]
#         cv2.imwrite("face.jpg",roi_color)

cv2.imshow('img',img)
cv2.waitKey(0)
```

Run the code - OUTPUT:



If you look closely, there is one more rectangle drawn near the end of the image.

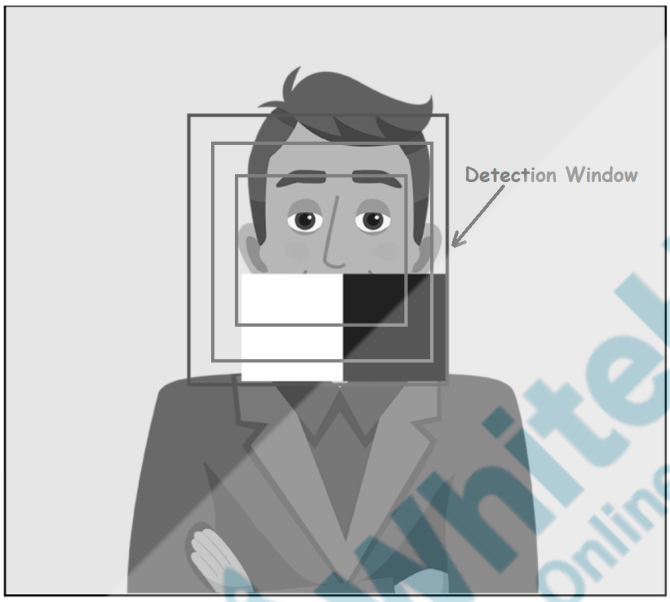
The algorithm can't be 100% accurate to detect the faces, but we can surely increase the precision by specifying other parameters of the **detectMultiScale()** method.

Along with image variable use for detection in **detectMultiScale()** we can specify:

- **scaleFactor** : This parameter sets the percentage amount to reduce the size of the **detection window**(it is the portion of the image used for detection features) after every detection.

This means the detection window size gets smaller and smaller after every round of detection, by the value chosen for **scaleFactor** to increase precision. Increasing the scaleFactor, helps to increase the detection accuracy.

Possible range between 1.1 to 1.9.



- **minNeighbors:** Parameter specifying how many facial features that need to be present, to detect the face.

To get more accurate result let us set values for the parameters **scalefactor** & **minNeighbors** in **detectMultiScale()**

```
gray= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

faces = face_cascade.detectMultiScale(gray,1.1, 3)
print(len(faces))

for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        # Crop the image to save the face image.
    #     roi_color = img[y:y+h, x:x+w]
    #     cv2.imwrite("babyface.jpg",roi_color)

cv2.imshow('img',img)
cv2.waitKey(0)
```

***Note**: The teacher can guide students to try with different numbers to check changes in output.*

Run the code with **3**.
If the extra square still there change it to **5**

```
faces = face_cascade.detectMultiScale(gray,1.1, 5)
print(len(faces))
```

Run the code: You can check in Terminal now we are getting the correct number of faces.

OUTPUT:

```
4
```

After class you can try with any of your group photos too.

Now let us try with a webcam. You come on the file name **live_detect.py**

We have covered in the previous class how to use openCV to capture video from a webcam, so can you explain to me the given code?

The teacher can help the student to explain the code.

ESR:

1. Use **cv2.VideoCapture(0)** to get a video capture object for the camera.

	2. Set up an infinite **while loop** and use the **read()** method to read the frames using the above-created object. 3. Use **cv2.imshow()** method to show the frames in the video. 4. Breaks the loop when the user presses a "**space**" key.

```
import cv2


vid = cv2.VideoCapture(0)

while(True):

    # Capture the video frame
    # by frame
    ret, frame = vid.read()

    cv2.imshow("Web cam", frame)

    if cv2.waitKey(25) ==32:
        break

vid.release()

cv2.destroyAllWindows()
```

In order to detect faces here, we first need to initiate the

Haar cascade. Let us create a variable **face_cascade.** Shall we call it inside while loop or before it?

We can keep it before starting a while loop.

The teacher can also ask student to copy the code from ***face_detect.py***

ESR: Varied

```
# define a video capture object
vid = cv2.VideoCapture(0)
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

What shall we do next?

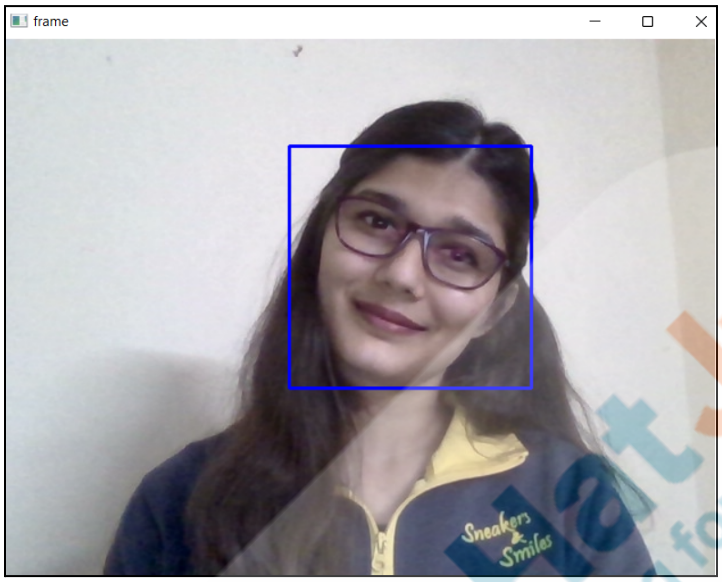
Yes, this we need to do with each frame. So we will call **detectMultiscale()** inside **while loop** for each frame captured while **cv2.read()**

ESR: Convert image in to grayscale.

```
while(True):

    # Capture the video frame
    # by frame
    ret, frame = vid.read()

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect the faces
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    # Draw the rectangle around each face
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

    # Display the resulting frame
    cv2.imshow('frame', frame)
```

Note*: Based on the student's speed the teacher can decide to make him type the code or copy from* ***face_detect.py****. While copying make sure to change* ***img*** *to* ***frame****.*

Run the code - **OUTPUT**:



***Note**: The student must keep his video off in class in order to use the webcam in the code.*
The output will show the student's face highlighted with a rectangle.
The student can call any of his/her family members in the range of the webcam then both faces will be detected.

You can see a rectangle drawn around your face. You can also change the thickness and color of the rectangle shape.

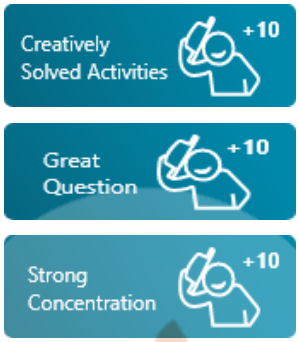
In this class, we learned about the concept of face detection using OpenCV in Python using Haar cascade. There are a number of detectors other than the face, which can be found in the library. Feel free to experiment with them and create detectors for eyes, license plates, etc.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins

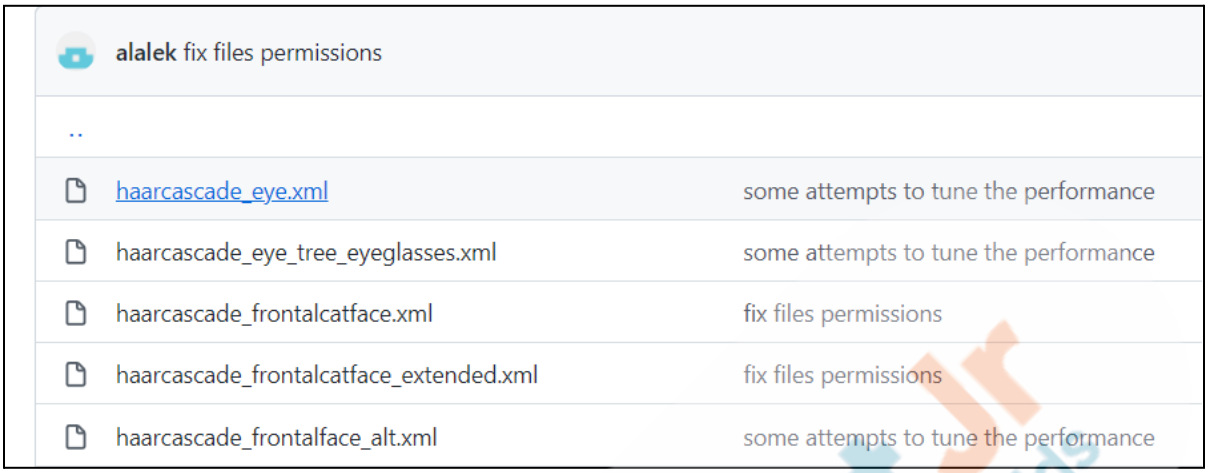
Teacher Starts Slideshow **Slide 15 to 20** 
Activity Details: **Following are the WRAP-UP session deliverables:** ● Appreciate the student. ● Revise the current class activities. ● Discuss the quizzes.
WRAP-UP QUIZ Click on In-Class Quiz
Continue WRAP-UP Session  **Slide 21 to 26**
Activity Details: **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.** ● **Ask the student to make notes for the reflection journal along with the code they wrote in today's 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	
Teacher Action	**Student Action**
Encourage the student to do **eye** detection on the face. Refer **live_detect_AA.py** file in the Reference Code(Teacher Activity 2).	

1. Save the **eye** classifier **.xml** file into the working directory: The classifier files can be accessed from the **opencv** GitHub Repository here.

   OpenCV Haar Cascade Classifiers



2. Update the **live_detect.py** and load the **haarcascade_eye.xml** files in the code script using **eye_cascade** variable.

   ***Note 1**: If the XML file present in the working directory is not detected, then use the full path to the XML files downloaded along with the **opencv-python** installation.*

   ***Note 2**: The below path to the **cv2/data** folder in the Windows system which might vary from the system used by the teacher/student.*

   *Full Path to **cv2/data** folder:*
   C:/Users/preet/AppData/Local/Programs/Python/Python39/Lib/site-packages/cv2/data

```
#Load the Cascade Classifier File
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

eye_cascade =
cv2.CascadeClassifier('C:/Users/preet/AppData/Local/Programs/Python/Python39/Lib/site-packag
es/cv2/data/haarcascade_eye.xml')
```

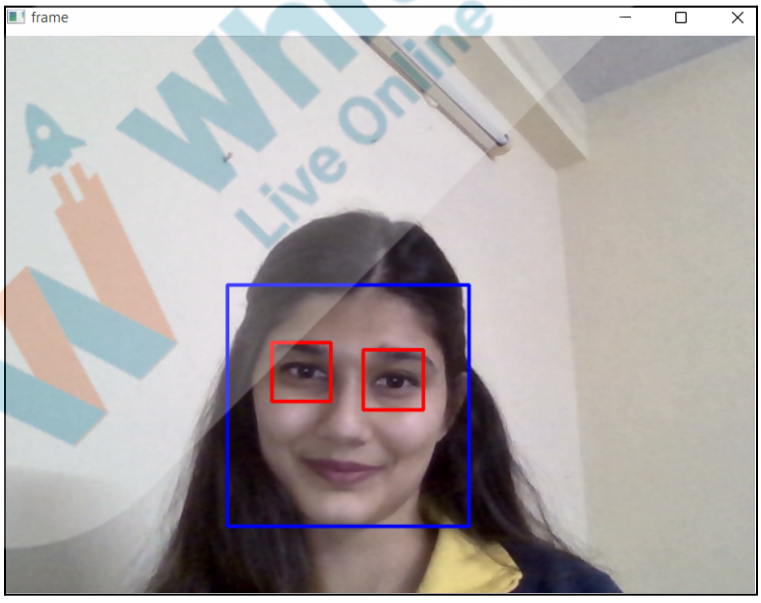
3. Set the **detectMultiScale()** method parameters for **eye** detection.

```
# Detect the faces, eyes and smile
faces = face_cascade.detectMultiScale(gray, 1.1, 5)

eyes = eye_cascade.detectMultiScale(gray, 1.1, 5)
```

4. Loop to draw rectangle around **eyes.**

```
# Draw the rectangle around the face, eyes and mouth
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

for (x, y, w, h) in eyes:
    cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

5. Run the code using **live_detect.py** and test the output.



The Haar Cascade can easily help us detect multiple features. Keep Exploring!

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	Teacher Boilerplate	https://github.com/procodingclass/PRO-C106-Teacher-Boilerplate
Teacher Activity 2	Reference Code	https://github.com/procodingclass/PRO-C106-Reference-Code
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C106-Student-Boilerplate
Teacher Reference 1	OpenCV Haar Cascade Classifiers	https://github.com/opencv/opencv/tree/4.x/data/haarcascades
Teacher Reference 2	Project Document	https://s3-whjr-curriculum-uploads.whjr.online/abdfc53a-726f-43a6-9d9d-100807a410f8.pdf
Teacher Reference 3	Project Solution	https://github.com/procodingclass/PRO-C106-ProjectSolution
Teacher Reference 4	Visual Aid Link	https://s3-whjr-curriculum-uploads.whjr.online/b9008e7d-dfac-44fe-9b67-361727d046e2.html
Teacher Reference 5	In Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/7ebcf395-f944-4220-b989-b2a8d98f6bf7.pdf